Doctoral theses at NTNU, 2023:261

Katrine Seel

# Learning for Model Predictive Control

Doctoral thesis

**NTNU**
Norwegian University of Science and Technology
Thesis for the Degree of Philosophiae Doctor
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of Science and Technology

Katrine Seel

# Learning for Model Predictive Control

Thesis for the Degree of Philosophiae Doctor

Trondheim, August 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

**NTNU**
Norwegian University of Science and Technology

Thesis for the Degree of Philosophiae Doctor

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

© Katrine Seel

ISBN 978-82-326-7222-6 (printed ver.)
ISBN 978-82-326-7221-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

Doctoral theses at NTNU, 2023:261

Printed by NTNU Grafisk senter

# Summary

This thesis focuses on learning-based control, with an emphasis on control designs for which we can analyze stability and robustness properties. The topic is motivated by the lack of available controllers for complex, nonlinear dynamical systems that are hard to model, and that are also applicable to safety-critical applications.

Recent successes in the field of machine learning (ML), as well as the availability of increased sensing and computational capabilities, have led to a growing interest in data-driven control techniques. For systems that require systematic handling of constraints, MPC has established itself as the primary control method. The combination of ML and MPC has therefore become a popular field of research, as data can be exploited to improve controller performance, while tools for stability and robustness analysis are well-established.

The most intuitive combination of MPC and ML is using available data to improve the MPC prediction model. Supervised ML methods based on e.g. rich function approximators such as neural networks (NNs) and Gaussian processes (GPs) can be leveraged to learn parts of or entire dynamical models from data. In part I of this thesis, we propose two different MPC formulations that leverage ML to learn the dynamics. Using techniques from robust control, we provide stability guarantees under core assumptions on the approximation error.

There has also been an increasing interest in inferring the parameterization of the MPC controller, of not only the prediction model but also the cost and constraints, that lead to the best closed-loop performance. Reinforcement learning (RL) is a framework for developing self-optimizing controllers that adjust their behaviors based on observed outcomes of their actions. As the policies are usually modeled using NNs, the resulting closed-loop behavior is difficult to analyze. In Part II of this thesis, we consider RL as a tool to infer the optimal parameterization of an

MPC scheme. Leveraging existing theory on stability analysis of MPC, we propose a cost parameterization and constrained RL parameter updates such that the nominal closed-loop stability of the learned MPC is ensured by design. We also consider different approaches for combining RL methods, as a way to speed up learning. Finally, we propose a new method for exploration during learning.

On a more general level, this thesis has considered two conceptually different approaches to learning-based MPC. Whereas the combination of supervised ML and MPC can be exploited to design MPC schemes with highly accurate prediction models and exploit potentially already available data sets in order to learn them, the models are not learned to optimize the closed-loop performance directly. The combination of MPC and RL on the other hand, allows us to learn not only the prediction model but also the cost and constraints that directly optimizes the closed-loop performance.

# Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor (PhD) at the Norwegian University of Science and Technology (NTNU). I carried out the presented work at the Department of Engineering Cybernetics (ITK) under the main supervision of Professor Jan Tommy Gravdahl and with Professor Sébastien Gros, Dr. Esten Ingar Grøtli (SINTEF), Professor Kristin Y. Pettersen, and Dr. Signe Moe (former SINTEF), as my co-supervisors. My work has been supported by the Norwegian Research Council (NFR) through the project "Towards Autonomy in Process Industry" (TAPI) with grant number 294544.

**Acknowledgements**

The following thesis has come into being through years of hard work, and a group of indispensable people. First of all, I want to thank my main supervisor, Jan Tommy Gravdahl, for always cheering me on and for being approachable for all kinds of discussions and ideas. Thank you for meeting me with enthusiasm and helping me build confidence as a researcher. During the first part of my PhD, I benefited greatly from Kristin's expertise in control and stability analysis. Signe, thanks for introducing me to the field of ML which at this time was still very new to me. In addition, I am very grateful for the collaboration I have had with Sébastien during the second half of my PhD. Thank you for including me in your research group and for giving me guidance, it has been challenging and inspirational. Finally, I would like to thank Esten, for all our valuable discussions on a wide range of topics throughout my PhD and for your hawk eyes on so much of my written work.

In addition to my supervisors, I have also had important collaborations with several other people. I spent 3 months at IMT School for Advanced Studies Lucca, where I was fortunate to work with Professor Alberto Bemporad. Moreover, I would like to thank Mark Haring from SINTEF and Arash Bahari Kordabad from NTNU for fruitful collaborations.

I also want to thank all my colleagues in SINTEF, both in Trondheim and in Oslo, for the coffee and lunch breaks that got my mind off things and for your insights into the hardships of doing a PhD.

The time has come to thank my closest friends. Vilde, getting to know you has been the most enjoyable part of moving back to Trondheim, and without your friendship, this experience would truly have been poorer. To my friends in Oslo, thank you for all the phone calls and messages letting me know that you were waiting for me on the other side. I can finally say that the time has come, and I cannot wait to spend more time with you.

Finally, my family cannot be thanked enough for their unwavering faith that I would make it, all the times I doubted it myself. Mother, although not formally listed, you have and will continue to function as my supervisor in life.

# Contents

# List of Abbreviations

| | |
|---|---|
| CSTR | Continous stirred tank reactor |
| DP | Dynamic programming |
| DPG | Deterministic policy gradient |
| EMPC | Econmic model predictive control |
| ENMPC | Economic nonlinear model predictive control |
| FNN | Feedforward neural network |
| GP | Gaussian process |
| ICNN | Input convex neural networks |
| IDW | Inverse distance weighting |
| ISS | Input-to-state stability |
| KKT | Karush-Kuhn-Tucker |
| LQR | Linear quadratic regulator |
| MDP | Markov decision process |
| ML | Machine learning |
| MPC | Model predictive control |
| NARX | Nonlinear autoregressive model with exogenous inputs |
| NMPC | Nonlinear model predictive control |

| | |
|---|---|
| NN | Neural network |
| OCP | Optimal control problem |
| RL | Reinforcement learning |
| RMPC | Robust model predictive control |
| TD | Temporal difference |
| wBLR | Weighted Bayesian linear regression |

# Chapter 1

# Introduction

## 1.1 Background and motivation

*Automation* describes the design of a process or a system that is self-governing, self-acting, or moving on its own. The characteristic is typically attributed to processes that require little or no human intervention. In more recent times, automation was taken one step further, by adding also the ability to learn and adapt to its environment, known as *autonomy*. Today, we see both automation and autonomous systems everywhere, essentially made possible by computerized control.

A prerequisite for autonomous control systems is the availability of measurements and/or accurate mathematical models to build control algorithms. However, industries that suffer from both sparse measurements as well as difficulties related to the mathematical modeling of their underlying processes, are falling behind when it comes to incorporating autonomous systems. As an example, we see this apply to the process industry. The process industry typically deals with dynamical systems that are highly nonlinear and potentially subject to harsh environments. In harsh conditions, sensors cannot survive permanently. Therefore, measurements need to be taken manually, which in turn makes them expensive, in addition to constituting a risk for the human operators that need to take them. Moreover, because dynamics are typically nonlinear, complex, and may also suffer from varying time delays, traditional model-based methods for control are not sufficient.

Machine learning (ML) is a sub-field of artificial intelligence (AI), and can broadly be categorized as the study of how computer algorithms can learn from new information to improve their operation. Due to the increasing amounts of data and the recent improvements in processing technology, we have seen a surge in the use and research of ML, as well as new areas of use constantly emerging. ML is usually divided into three subcategories, namely (1) supervised learning, (2) unsupervised learning, and (3) reinforcement learning (RL). Supervised learning algorithms involve training with labeled data sets, whereas unsupervised learning uses unlabelled data sets to find patterns or trends in data. Finally, RL uses trial and error to optimize sequential decision-making.

The intersection between control theory and ML is a growing research field with large potential for many types of control systems, see e.g. [1]. The literature on combinations of learning and control is vast, and this short background section does not aim to cover all of it but rather aims to give a flavor of some of the main directions within the field. Especially supervised learning and RL are suited for learning control algorithms.

RL is a framework for optimizing a policy, that maps from states to actions and is analogous to what the control community refers to as a controller. The policy is optimized through a reward function, that rewards actions that lead to desirable states. Recently, impressive results have been obtained using so-called deep RL, which are RL algorithms combined with deep neural networks (NNs) as function approximators. Deep RL has successfully been used for games such as Go [2] and Atari games [3], and in learning robots how to fly [4], walk [5] and perform complex manipulation [6]. Because the learned policies are based on NNs, their resulting behavior is difficult to analyze and therefore difficult to trust for safety-critical applications.

The term black box model is often used to describe models for which humans cannot understand how the predicted output was obtained. The need for transparency and trust in learned models has motivated the study of explainable AI, a new emerging research field devoted to developing methods for visualizing, explaining, and interpreting data-driven models, see e.g. [7]. This is particularly important when AI is used in sensitive domains with societal, ethical, and safety implications. Partly because explainable AI has mostly evolved for classification models in different domains, it has to a small extent been applied to learned control policies.

In the face of control of safety-critical systems, it is generally unacceptable that we cannot guarantee the resulting closed-loop behavior. To trust a control algorithm, we need to be able to assess properties such as stability, robustness and constraint satisfaction. From a control-theoretic perspective, stability properties are typically addressed using Lyapunov methods and passivity methods [8]. These tools are more directly applicable to the combination of learning and control, where the control law is cast in a more traditional form, but learning is added as an ingredient in the controller.

Controllers cast as NNs, have also been explored using supervised learning methods. This combination of learning and control is especially relevant in the case of using model-based control methods where the model of the underlying process is complex and therefore computationally expensive to evaluate. One obvious example is model predictive control (MPC) for complex processes, which is a control algorithm based on solving an optimal control problem at each sampling time, involving a prediction model of the system dynamics. The authors in [9], propose to use NNs as an approximator of explicit predictive control laws. Although the controller itself is modeled with an NN as in deep RL, the closed-loop behavior of the optimal policy, on which the NN is trained, can be analyzed.

Supervised learning methods can also be a tool for model-based controllers to perform system identification. An early combination of system identification and control dates back to the 1970s and is known within the control community as adaptive control [10]. Adaptive control considers systems with parametric uncertainties and adapts controller parameters online to optimize performance. The control technique is limited to a specific model structure, it tends to "overfit" to the latest observations, and convergence to the true parameters is generally not guaranteed [11], [12].

Among others, the aforementioned limitations of adaptive control have motivated learning-based approaches to deal with model inaccuracies. Dynamical models obtained as a result of learning may replace either the entire dynamical model or parts of it needed in model-based controllers, e.g. in feedback linearizing controllers as in [13], to design the prediction model in MPC as in [14] or in a feedforward controller as in [15]. Alternatively, supervised learning can be used to estimate immeasurable states, as to perform output feedback control as in [16].

For systems that require systematic handling of constraints, MPC has established itself as the primary control method. The MPC scheme relies on a sufficiently descriptive model of the system to optimize performance and ensure constraint satisfaction, thus rendering modeling critical for the success of the resulting controller. Perhaps because of the two aforementioned reasons, learning-based MPC has recently received increased attention from the research community. This has also included a focus on safety, which in this context usually is understood as constraint satisfaction when dealing with model mismatch. For a full review of safe learning-based MPC, the reader is referred to [17]. For safety-critical systems, we may argue that safety, in a broader understanding of the word, is not only ensured by guaranteeing constraint satisfaction but that we, in addition, need some guarantees related to the stability properties of the closed-loop system.

The study of the stability of learning-based control in general, including learning-based MPC, is challenging because of the black-box nature of traditional ML methods. One approach to ensuring stability is to make necessary assumptions regarding the prediction error of the ML model, and study how this affects the stability of the resulting controller, see e.g. [18], [13] and [19]. Stability can also be ensured in a probabilistic sense when applying probabilistic methods such as Gaussian processes (GP) for dynamical modeling, as in e.g. [20]. Generally, however, there is no fits-all analysis that can be made for all learning-based controllers, as the analysis needs to be adapted to the type of learning algorithm, as well as to which control architecture is used, and how the two are combined.

In between deep RL for control and supervised learning for system identification to be used in model-based controllers, we find an alternative option for learning for control. The authors in [17], refer to this alternative as performance-driven controller learning, which aims to infer the optimal parameterization of an MPC scheme w.r.t. the closed-loop performance. By adopting a parameterized MPC scheme as a function approximator, RL can be used to learn the parameters. This was first proposed in [21]. The mentioned framework allows using RL algorithms for adjusting the policy, approximated using a parameterized MPC optimization problem, thereby casting it in a form that offers rich tools to analyze the resulting closed-loop behavior. The proposed framework exploits the ability to learn from sampled state transitions and the associated rewards to adopt a policy with closed-loop properties that we can analyze. However, using RL to update the MPC parameters online must be done with care to ensure properties such as stability and constraint satisfaction as addressed in [22].

## 1.2 Research Objectives

The research resulting in this thesis is funded by the project "Towards autonomy in process industry" (TAPI). A central research objective for the TAPI project is combining data-driven and model-based methods for control, with the goal of understanding and analyzing the stability and robustness properties of the resulting closed-loop system. Motivated by this, as well as the described challenges and opportunities outlined in the preceding section, the main research objective of this thesis is to explore learning-based MPC approaches for systems that may be hard to model, while providing stability guarantees. More specifically, the thesis considers two conceptually different approaches to learning-based MPC, namely MPC in combination with supervised learning methods and MPC as a function approximator in RL.

In Part I of this thesis, we consider how supervised learning methods can be integrated with MPC. In particular, we consider recurrent NNs to be used as the prediction model in MPC. We then wish to answer the following research questions:

**R.1** What type of stability guarantees can be made for a system controlled by MPC using an NN as the prediction model?

Moreover, we consider supervised learning methods in combination with robust control techniques. In this context, we consider a specific class of systems, namely Lur'e systems. For this class of systems, we investigate:

**R.2** Can we use supervised learning methods to reduce the conservativeness of robust control of Lur'e systems?

Part II of the thesis builds upon existing research using MPC as a function approximator in RL. An important question for this framework is how to parameterize e.g. the cost function in the MPC scheme. We aim to answer the following research question:

**R.3** Can rich cost parameterizations improve controller performance for MPC schemes based on inaccurate models?

With the controller cast as a parameterized MPC problem, various RL methods have been tested for updating the parameters. In this thesis, we investigate the potential of combining RL methods, and ask the following question:

**R.4** Can we combine RL methods in order to speed up learning when using MPC as a function approximator in RL?

In an RL setting, we adjust the parameters of the policy based on experience gained from interacting with the system. In order to improve the performance of the initial policy, which in our case is cast by an MPC scheme, we depend on visiting interesting areas of state space, leading to meaningful parameter updates, i.e. *explore* efficiently. This leads us to the following research question:

**R.5** Can we do better than random exploration for MPC as a function approximator in RL?

## 1.3 Contributions and outline

The thesis is organized into two main parts, consisting of a total of 8 chapters, including this introduction and a concluding chapter. Chapter 2 aims at providing relevant general theory for the subsequent chapters. Part I covers the contributions

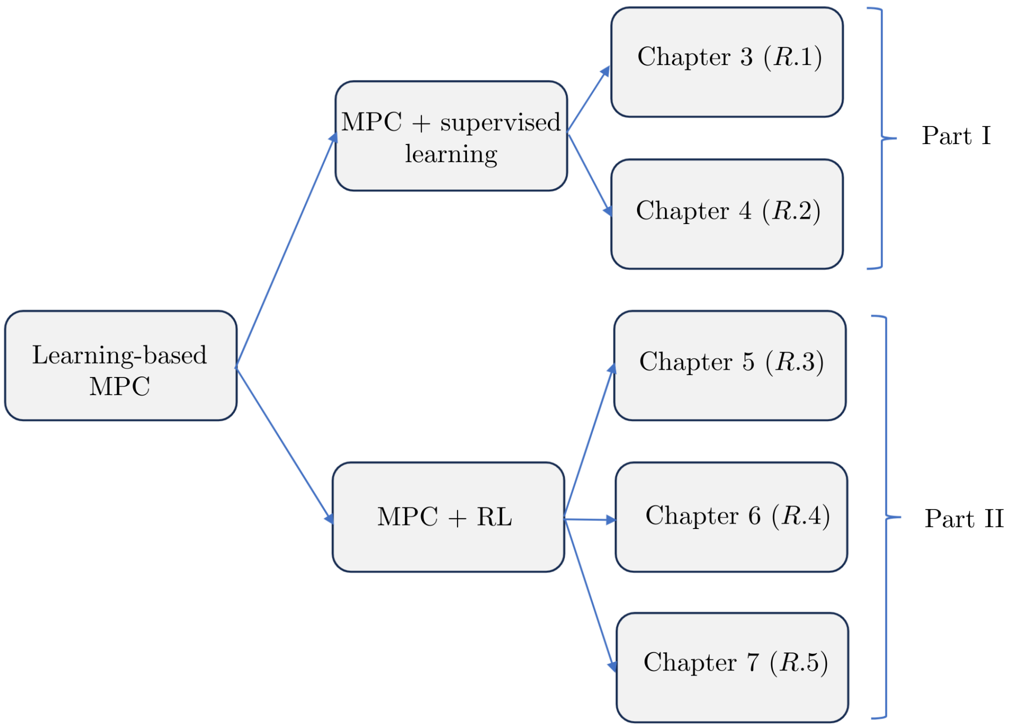


**Figure 1.1:** An overview of the thesis in terms of chapters and the associated research questions (R.1 - R.5).

made on MPC combined with supervised learning methods, while Part II covers variations of using MPC as a function approximator in RL. In the following, we look at the topic and contributions of Chapters 3-7. We will also list the papers on which the chapters are based, for which I am the main contributor. An overview of this part of the thesis is visualized in Figure 1.1.

**Chapter 3: Learning the MPC prediction model with NNs**

[23] **Katrine Seel**, Esten Ingar Grøtli, Signe Moe, Jan Tommy Gravdahl and Kristin Ytterstad Pettersen. Neural network-based model predictive control with input-to-state stability. *2021 American Control Conference* (ACC), pp. 3556-3563. IEEE, 2021.

In this chapter, we consider supervised learning methods for learning the dynamical model used to predict the system behavior in the MPC scheme. We consider an output feedback MPC scheme, for which we are not dependent on measuring the entire state, and learn offline using input-output data. The contributions we make are:

- Design of a learning-based MPC scheme, with an autoregressive prediction model based on an NN trained on input-output data.
- For the resulting closed-loop system, we show input-to-state (ISS) stability with respect to the prediction error.

**Chapter 4: Learning for robust control of sector-bounded systems**

[24] **Katrine Seel**, Mark Haring, Esten Ingar Grøtli, Kristin Ytterstad Pettersen and Jan Tommy Gravdahl. Learning-based Robust Model Predictive Control for Sector-bounded Lur'e Systems. *IFAC-PapersOnLine*, 54(20), 46-52, 2021. 1st IFAC Modeling, Estimation and Control Conference (MECC) 2021.

In Chapter 4 we treat a class of systems known as Lur'e systems, which are linear systems with sector-bounded nonlinearities. We consider specifically a robust MPC formulation typically used to control Lur'e systems and propose a probabilistic learning method to learn the sector-bound from data. We make the following contribution:

- Propose a stochastic sector formulation that reduces the conservativeness of robust control of Lur'e systems.

**Chapter 5: Cost modifications for learning-based MPC**

[25] **Katrine Seel,** Arash Bahari Kordbad, Sébastien Gros and Jan Tommy Gravdahl. Convex neural network-based cost modifications for learning model predictive control. *IEEE Open Journal of Control Systems*, 1, 366-379, 2022.

Chapter 5 is the first chapter of Part II, where we consider MPC as a function approximator in RL. In this chapter, we consider how to select the cost parameterization specifically, with a focus on obtaining a parameterization that ensures stability, also as parameters are updated, and that renders an optimization problem that can be solved within a reasonable time. The main contribution is described as:

- Design of convex cost modifications for MPC that ensure nominal stability of the closed-loop system.

## Chapter 6: Combining RL methods for learning-based MPC

[26] **Katrine Seel**, Sébastien Gros and Jan Tommy Gravdahl. Combining Q-learning and Deterministic Policy Gradient for Learning-based Model Predictive Control. *Accepted to 2023 62$^{nd}$ IEEE Conference on Decision and Control (CDC).*

In this chapter, we consider problems for which we may need a combination of RL methods in order to verify stability and capture the optimal policy. Generally, first-order parameter updates, i.e. parameter updates using gradient information of the optimization objective, are the most common in RL. This is because, among other things, it is hard to estimate second-order derivatives from data, which is needed to formulate a second-order parameter update. In this chapter, we make the following contributions:

- A multi-objective approach to combining RL methods.
- A novel second-order parameter update based on combining RL methods.

## Chapter 7: Variance-based exploration for learning MPC

[27] **Katrine Seel,** Alberto Bemporad, Sébastien Gros and Jan Tommy Gravdahl. Variance-based Exploration for Learning Model Predictive Control. *IEEE Access*, 11, 60724-60736, 2023.

Exploration is an essential component of all RL algorithms. The most common method for ensuring exploration is based on random perturbations of the policy, which for certain types of problems may yield slow learning and/or modest improvements in controller performance. More sophisticated exploration techniques have been developed specifically for NNs as function approximators. In Chapter 7 we make the following contribution:

- A novel method for variance-based exploration geared towards using MPC as a function approximator in RL.

The following articles were published during the PhD, but are not part of this thesis:

## Publications not included in this thesis

[28] Mark Haring, Esten Ingar Grøtli, Signe Riemer-Sørensen, **Katrine Seel** and Kristian Gaustad Hanssen. A Levenberg-Marquardt algorithm for sparse

identification of dynamical systems. IEEE Transactions on Neural Networks and Learning Systems (2022).

[29] Akhil Anand, **Katrine Seel**, Vilde Gjærum, Anne Håkansson, Haakon Robinson, and Aya Saad. Safe learning for control using control Lyapunov functions and control barrier functions: A review. *Procedia Computer Science* 192 (2021): 3987-3997, 2021.

[30] Anne Håkansson, Aya Saad, Akhil Anand, Vilde Gjærum and **Katrine Seel**. Robust reasoning for autonomous cyber-physical systems in dynamic environments. *Procedia Computer Science*, 192, pp.3966-3978, 2021.

# Chapter 2

# Preliminaries

## 2.1 Model Predictive Control

MPC has seen significant success in recent decades and has established itself as the primary control method for the systematic handling of constraints, with wide adaptation in diverse fields, such as process control, automotive systems, and robotics. MPC is based on solving an optimal control problem (OCP) at each instant the state is sampled. In the following, we consider (possibly nonlinear) discrete-time systems of the form

$$z_{k+1} = f(z_k, u_k), \tag{2.1}$$

with state $z_k \in \mathbb{X} \subseteq \mathbb{R}^n$, input $u_k \in \mathbb{U} \subseteq \mathbb{R}^m$, and dynamical model $f : \mathbb{X} \times \mathbb{U} \to \mathbb{X}$. For a system as in (2.1), we formulate the following OCP

$$\begin{aligned}
V_N(z) = \min_{x,u} \quad & T(x_N) + \sum_{i=0}^{N-1} \ell(x_i, u_i) && \text{(2.2a)}\\
\text{s.t.} \quad & \forall i \in \mathbb{I}_{0:N-1} : x_0 = z, && \text{(2.2b)}\\
& x_{i+1} = \hat{f}(x_i, u_i) && \text{(2.2c)}\\
& u_i \in \mathbb{U}, && \text{(2.2d)}\\
& h(x_i, u_i) \leq 0, && \text{(2.2e)}\\
& x_N \in \mathbb{X}_f && \text{(2.2f)}
\end{aligned}$$

where $N$ is the length of the prediction horizon, $x = \{x_0, \ldots, x_N\}$ denotes the predicted states and $u = \{u_0, \ldots, u_{N-1}\}$ the predicted inputs. It is also possible to use the notation $x_{k+i+1|k}$ and $u_{k+i|k}$ to emphasize that we are predicting $i$ steps into the future, for the initial state sampled at time $k$, although this notation is not used in this chapter. The objective of the optimization problem is defined by the stage cost, $\ell(\cdot)$, and the terminal cost, $T(\cdot)$. We differentiate between the true state $z$ and the predicted state in OCP, using $x$, and distinguish between the physical time $k$ and predicted time $i$. Moreover, we use $\hat{f}(\cdot,\cdot)$ to describe the prediction model, which may differ from the true dynamical model in (2.1). Mixed input-state constraints are described by $h(x_i, u_i)$, and $x_N \in \mathbb{X}_f$ is a terminal set constraint, that we will address further in Section 2.1.2. The addition of a terminal constraint is classically used to ensure the recursive feasibility of the MPC scheme, as defined next.

**Definition 1.** *The MPC problem is called recursively feasible if for all feasible initial states feasibility is guaranteed at every state along the closed-loop trajectory.*

However, this property can be destroyed in the face of uncertainty, in the form of additive disturbances, model error or state estimation error, causing $\hat{f} \neq f$.

The solution to the OCP in (2.2), will produce an optimal input sequence of length $N-1$, $u^\star = \{u_0^\star, \dots, u_{N-1}^\star\}$ for which we will apply only the first element to the system, $u_0^\star$. The rest of the input sequence is disregarded, and at the next sampling instance, the procedure is repeated. This is illustrated in Figure 2.1. Many control applications are naturally posed as tracking problems. For these problems, the main objective of MPC is to minimize the tracking error, that is the difference between a reference and the system states and inputs. We refer to this as tracking MPC, for which the stage cost is typically a quadratic function. We can also use MPC for optimizing economic performance rather than a tracking objective, referred to as economic MPC (EMPC). EMPC is introduced further in Part II of this thesis and treated in more detail in Chapter 5.

### 2.1.1 Linear Quadratic Regulator

The linear quadratic regulator (LQR) is a state-feedback controller that arises as the optimal solution to the unconstrained control problem for linear dynamics and quadratic objectives [32]. Throughout this thesis, we consider the discrete-time formulation of LQR, defined for linear systems of the form

$$z_{k+1} = Az_k + Bu_k, \tag{2.3}$$

with system matrices $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$. The objective we wish to minimize is defined as

$$V_\infty = \sum_{k=0}^{\infty} z_k^\top Q z_k + u_k^\top R u_k + 2 z_k^\top N u_k, \tag{2.4}$$

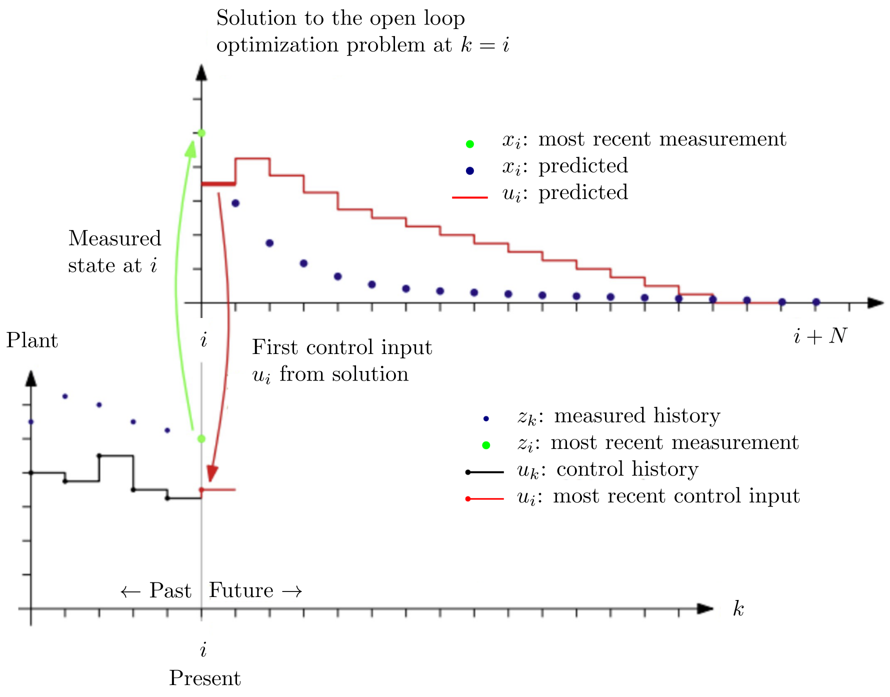


**Figure 2.1:** MPC illustration, modified version from [31].

where $Q \in \mathbb{R}^{n\times n} \succeq 0$, $R \in \mathbb{R}^{m\times m} \succ 0$ are weight matrices that penalize the state and input respectively and $N \in \mathbb{R}^{n\times m}$ is a cross term matrix. We obtain a state-feedback controller on the form

$$u_k = -Kz_k, \tag{2.5}$$

where the gain matrix, $K$, is given as

$$K = (R + B^\top PB)^{-1}(B^\top PA + N^\top), \tag{2.6}$$

where $P$ is the positive definite solution to the discrete-time Riccati equation (DARE), i.e.

$$P = A^\top PA - (A^\top PB + N)(R + B^\top PB)^{-1}(B^\top PA + N^\top) + Q. \tag{2.7}$$

### 2.1.2 Stability analysis of MPC

In the following, we will give a brief outline of the tools needed to perform stability analysis of systems controlled by MPC. The presented theory is taken from [33]. For more detailed analyses, we will make references to the relevant chapters in this thesis where the theory is applied.

As before, we consider the discrete, possibly nonlinear system, as defined in (2.1). Without loss of generality, we will assume that the reference setpoint is at the origin, i.e. $(z_e, u_e) = (0, 0)$ with $f(z_e, u_e) = 0$. To establish stability, we will make use of Lyapunov theorems defined in terms of function classes, i.e. $\mathcal{K}$, $\mathcal{K}_\infty$, and $\mathcal{KL}$. These are defined as follows.

**Definition 2.** *A continuous function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to class $\mathcal{K}$ if $\alpha(0) = 0$ and is strictly increasing.*

**Definition 3.** *A function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to $\mathcal{K}_\infty$ if it is in class $\mathcal{K}$ and is unbounded, i.e. $\alpha(j) \to \infty$ when $j \to \infty$.*

**Definition 4.** *A function $\beta(j, k) : \mathbb{R}_{\geq 0} \times Z_{\geq 0} \to \mathbb{R}_{\geq 0}$ belongs to $\mathcal{KL}$ if for each fixed $k$, $\beta(j, k)$ is class $\mathcal{K}$ w.r.t $j$ and if it for each fixed $j$, $\beta(j, k)$ is decreasing w.r.t $k$ and $\beta(j, k) \to 0$ as $k \to \infty$.*

In addition, we need the following property of a set.

**Definition 5.** *A set $\mathbb{X} \subseteq \mathbb{R}^n$ is positive invariant for $z^+ = f(z)$ if $z \in \mathbb{X}$ implies $f(z) \in \mathbb{X}$.*

We can then define asymptotic stability, which is what we want to show for the closed-loop system using MPC.

**Definition 6.** *Suppose $\mathbb{X}$ is positive invariant for $z^+ = f(z)$. The origin is asymptotically stable for $z^+ = f(z)$ in $\mathbb{X}$ if there exists a $\mathcal{KL}$ function $\beta(\cdot)$ such that, for every $z \in \mathbb{X}$*

$$\|\psi(k; z)\| \leq \beta(\|z\|, k), \forall k \in \mathbb{I}_{\geq 0}, \tag{2.8}$$

*where $\psi(k; z)$ denotes the solution to $z^+ = f(z)$ at time $k$ for initial state $z$ at time zero and $\|\cdot\|$ denotes the 2-norm.*

The set $\mathbb{X}$ is called a region of attraction. For a system controlled with MPC, the region of attraction is defined as follows.

**Definition 7.** *The region of attraction of an MPC scheme is the set of states which can be steered to the terminal region $\mathbb{X}_f$ in N steps or less.*

A standard approach to establish the asymptotic stability of the closed-loop system under MPC is to show that the value function $V_N(z)$ for the finite-horizon OCP in (2.2) is a Lyapunov function for the closed-loop system $z^+ = f(z, \kappa_N(z))$. Next, we formally define a Lyapunov function.

**Definition 8.** *Suppose $\mathbb{X}$ is positive invariant for $z^+ = f(z)$. A function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ is said to be a Lyapunov function in $\mathbb{X}$ for $z^+ = f(z)$ if there exits functions $\alpha_1$, $\alpha_2 \in \mathcal{K}_\infty$ and a continuous, positive definite function $\alpha_3$ such that for any $z \in \mathbb{X}$*

$$\alpha_1(\|z\|) \leq V(z) \leq \alpha_2(\|z\|), \tag{2.9a}$$
$$V(f(z)) - V(z) \leq -\alpha_3(\|z\|). \tag{2.9b}$$

Showing that the value function $V_N(z)$ satisfies the lower bound in (2.9a), is ensured by additionally assuming that the stage cost satisfies $\ell(z, u) \geq \alpha_1(\|z\|)$ where $\alpha_1 \in \mathcal{K}_\infty$ and $\ell(0, 0) = 0$. This is satisfied by the standard choice of a quadratic cost function, as the one used in (2.4).

To ensure the decrease condition in (2.9b), we will assume that the terminal cost is a Lyapunov function in the terminal set $\mathbb{X}_f$ and that the terminal set is invariant under a local control law $\kappa_f(z)$. These can be categorized as terminal conditions, and are soon discussed in more detail. For appropriate choices of the stage cost $\ell$, terminal cost $T$ and terminal region $\mathbb{X}_f$ that ensures the satisfaction of (2.9b), the upper bound in (2.9a) can also be also ensured. To ensure the existence of an upper bound on $V_N(z)$ outside the terminal region, i.e. for the region of attraction, we usually assume that the set $\mathbb{X} \times \mathbb{U}$ is compact, i.e. closed and bounded.

With the necessary assumptions for the stage cost and terminal conditions, the value function $V_N(z)$ satisfies the conditions in Definition 8 and is thereby a Lyapunov function for the closed-loop system, and stability is established. This is stated formally in the following theorem.

**Theorem 1.** *Suppose $\mathbb{X} \subset \mathbb{R}$ is positive invariant for $z^+ = f(z)$. If there exists a Lyapunov function in $\mathbb{X}$ for the system $z^+ = f(z)$, then the origin is asymptotically stable in $\mathbb{X}$ for the system $z^+ = f(z)$. If $\mathbb{X} = \mathbb{R}^n$ then the origin is globally asymptotically stable.*

*Proof.* See e.g. [33]. □

In case the stage cost is not a quadratic function, as for EMPC, stability analysis is generally more complicated. However, for certain types of problems, known as dissipative problems, we are able to show that the EMPC scheme can be re-cast as a tracking MPC scheme, for which we are able to prove the stability of the closed-loop system [34]. This is elaborated on mainly in Chapter 5.

The stability following from Theorem 1 is usually referred to as nominal, i.e. the stability analysis is done with respect to the prediction model in the MPC. If the true system exactly matches the prediction model, we have asymptotic stability for the true closed-loop system, whereas, for any disturbances in the form of noise or model errors, we must turn to robust stability techniques. In the case of bounded disturbances, we can achieve input-to-state stability (ISS). This is considered Chapter 3.

### Terminal conditions

For the terminal cost to be a Lyapunov function in the terminal region, three particular formulations of the terminal conditions in (2.2) can be used, as summarized in [35]:

1. $N \to \infty$, $\mathbb{X}_f = \{0\}$ and $T(z) = 0$. This corresponds to the infinite-horizon case but is difficult to implement with nonlinear dynamics.

2. $N$ is finite, $\mathbb{X}_f = \{0\}$, and $T(z) = 0$. This case mirrors the infinite horizon, but the specification of $N$ sufficiently large is difficult in general. In addition, a point constraint is known to severely reduce the feasible region of the MPC scheme [36].

3. $N$ is finite, a suitable terminal region $\mathbb{X}_f$ has been calculated for the MPC scheme (2.2), and a terminal controller $\kappa_f(z)$ is known with a suitable terminal cost $T(z)$.

In practice, MPC is often implemented without terminal constraints. Conditions for stability without terminal constraint have been thoroughly studied. In this thesis, we will consider the following two, i.e.

1. Replace the terminal cost $T$ by $\lambda T$ with $\lambda \geq 1$ sufficiently large, such that a suitable terminal constraint $\mathbb{X}_f$ is satisfied without being explicitly stated in the MPC (2.2) [37].
2. Use a general positive definite terminal cost, that can serve as a global Lyapunov function, and thereby there is no need for a terminal constraint. It has also been shown that for a sufficiently long horizon $N$, we can omit both the terminal cost and terminal constraint [38].

In Chapter 3, we consider option 1 for the terminal conditions without a terminal constraint, whereas in Chapter 5 we consider option 2. The advantages of omitting the terminal constraint are (i) the OCP is simpler i.e. easier to solve and (ii) a larger feasible set is obtained. On the other hand, establishing recursive feasibility becomes more challenging and requires additional assumptions. Addressing recursive feasibility both without a terminal condition as well as in the face of uncertainty, has, with the exception of Chapter 4, not been done in this thesis.

## 2.2 Markov decision process

Markov decision processes (MDPs) provide a standard framework for the optimal control of discrete-time stochastic processes, defined by a stage cost and a transition probability that completely characterizes the environment's dynamics. We denote the underlying conditional transition probability density as $p$, defined for the state $s \in \mathcal{S} \subseteq \mathbb{R}^n$ and action $a \in \mathcal{A} \subseteq \mathbb{R}^m$. The function $p$ gives the probability density of transitioning to state $s^+$ given action $a$ in state $s$, i.e.

$$s^+ \sim p(\cdot|s,a). \tag{2.10}$$

The probability of each possible value of $s^+$ should depend only on the immediately preceding state and action, i.e. $s$ and $a$. This is best viewed as a restriction on the state rather than on the decision process and is known as the Markov property. A Markovian state includes all information from the past that is relevant for the future. The dynamics in (2.10) include deterministic dynamics as a special case. In control theory, the dynamics in (2.10) are more often given as, only different in terms of notation,

$$s^+ = f_w(s,a,w), w \sim \mathcal{W}, \tag{2.11}$$

where $w \in \mathbb{R}^d$ is a random disturbance from distribution $\mathcal{W}$ and $f_w : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^n$. In the following, we will use the term cost to describe what is commonly referred to as reward and assume that this is given, i.e.

$$L(s, a), \tag{2.12}$$

is known. For a given MDP, we wish to find the optimal policy, $\pi^\star : \mathcal{S} \to \mathcal{A}$, that is the solution to

$$\pi^\star = \arg\min_{\pi} J(\pi), \tag{2.13}$$

where $J(\pi)$ is the performance function. This is defined as

$$J(\pi) = \mathbb{E}_{s_0 \sim p_0, s \sim p(\cdot|s,\pi(s))} \left[ \sum_{k=0}^{\infty} \gamma^k L(s_k, a_k) \mid a_k = \pi(s_k) \right], \tag{2.14}$$

where $p_0$ is a distribution of initial states and $\gamma \in (0, 1]$ is a discount factor used to establish the importance of future costs over immediate costs. In the following, we will use $\mathbb{E}_\pi[\cdot]$ as short for $\mathbb{E}_{s_0 \sim p_0, s \sim p(\cdot|s,\pi(s))}[\cdot]$, where $\mathbb{E}[\cdot]$ is the expected value operator. We note that the performance measure in (2.14) also includes the special case where the initial condition is fixed, and not drawn from a distribution. In (2.14) the performance function is defined for an infinite horizon, but the performance can also be defined for a finite horizon. We recognize that minimization of the sum of discounted costs in (2.14) aligns with the commonly used formulation of the MDP objective which is maximizing the sum of discounted rewards $r(s, a)$ by stating that $r(s, a) = -L(s, a)$.

The value function $V_\pi$ and action-value function $Q_\pi$ associated with a policy $\pi$ satisfy the Bellman equations [32]

$$Q_\pi(s, a) = L(s, a) + \gamma \mathbb{E}_{s^+ \sim p(\cdot|s,a)}[V_\pi(s^+)|s, a], \tag{2.15a}$$

$$V_\pi(s) = Q_\pi(s, \pi(s)). \tag{2.15b}$$

From the optimal action-value function $Q^\star(s, a)$, we can obtain the optimal policy as

$$\pi^\star(s) = \arg\min_{a} Q^\star(s, a). \tag{2.16}$$

Solving the MDP problem in (2.13) is possible by using dynamic programming (DP) techniques on the Bellman equations. However, the problem quickly becomes intractable as the dimension of the problem grows [32]. In addition, DP depends on the exact transition dynamics, that in most engineering applications are not readily available. RL provides alternative methods for solving MDPs, that do not depend on the exact transition dynamics, that will render an approximation of the optimal policy based on data.

## 2.3 Reinforcement Learning

The taxonomy of RL methods is defined by several meaningful categorizations. One important distinction is between model-based and model-free methods. Model-based methods use a model of the system dynamics that can either be known beforehand or learned jointly with the policy. Unlike model-based methods, model-free methods assume no knowledge of the system dynamics. In this thesis, we consider only model-free methods.

Model-free methods can coarsely be further divided into two groups, namely value-based and policy-based methods. The first category aims to learn an approximation of the value or action-value function and then obtains a policy estimate from this approximation, such as in (2.16). Policy-based methods, on the other hand, aim to directly learn the optimal policy. In the next coming section, we will visit one common value-based method, namely Q-learning, and look at a sub-category of policy-based methods, namely policy gradient methods. We will consider both of these methods in Chapters 5, 6 and 7.

### 2.3.1 Q-learning

Q-learning is a value-based method, that approximates the optimal action-value function via a parameterized function approximator $Q_\theta$ with parameter vector $\theta$. The goal of value-based RL methods is then to learn the optimal parameters $\theta^\star$, such that $Q_{\theta^\star}(s,a) \approx Q^\star(s,a)$. Q-learning aims to solve the following least squares problem

$$\min_\theta \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(Q^\star(s,a) - Q_\theta(s,a))^2\right]. \tag{2.17}$$

Because $Q^\star$ is generally not known, this can not be solved directly. One approach to solving (2.17) is the temporal difference (TD) learning [39]. The TD error is defined as

$$\delta_k = L(s_k,a_k) + \gamma V_\theta(s_{k+1}) - Q_\theta(s_k,a_k), \tag{2.18}$$

where $V_\theta$ is the parameterized value function. Let $y_k = L(s_k,a_k) + \gamma V_\theta(s_{k+1})$, and consider this a fixed target, evaluated using a sampled state transition and the cost. The parameter update is formulated to minimize the squared TD error, i.e.

$$\min_\theta \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(y - Q_\theta(s,a))^2\right]. \tag{2.19}$$

In order to solve (2.19) in practice, it is necessary to collect data also by deviating from the policy $\pi_\theta$, which is referred to as exploration. For the minimization problem in (2.19), we define the following first-order (semi)-gradient step

$$\theta \leftarrow \theta + \zeta\delta_k\nabla_\theta Q_\theta(s_k,a_k), \tag{2.20}$$

where $\zeta > 0$ is a scalar denoting the step size. The policy estimate is then defined as

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a). \tag{2.21}$$

For Q-learning techniques, it should be mentioned that there is no guarantee to find the optimal policy. This is because the parameter update of Q-learning methods is not designed to optimize closed-loop performance directly. Instead, Q-learning aims to fit $Q_\theta$ as closely as possible to $Q^\star$, and assumes that $Q_\theta \approx Q^\star$ results in $\pi_\theta \approx \pi^\star$. However, there are no guarantees that the former approximation implies the latter, and for certain shapes of Q-functions, it still may be challenging to capture the optimal policy, even with an almost correct Q-function estimate.

### 2.3.2 Policy gradient methods

The lack of convergence guarantees for Q-learning methods has motivated the need for alternative methods with more formal (local) convergence guarantees [40]. Using policy gradient methods, the parameters are updated towards improving the performance of the policy irrespective of the action-value function accuracy. For policy-based methods, the policy rather than the value functions are parameterized, and the policy parameters are adjusted in order to minimize $J(\pi_\theta)$. For policy gradient methods, this is done by estimating the policy gradient $\nabla_\theta J(\pi_\theta)$. A gradient descent step is then used to update the parameters, i.e.

$$\theta \leftarrow \theta - \omega \nabla_\theta J(\pi_\theta), \tag{2.22}$$

where $\omega > 0$ is the step size. For deterministic policies, we apply deterministic policy gradient methods (DPG), for which the policy gradient may be estimated as in [41]

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q_{\pi_\theta}(s,a) | a = \pi_\theta\right], \tag{2.23}$$

where $Q_{\pi_\theta}$ is the true action-value function for policy $\pi_\theta$ as defined in (2.15a). Instead of using the true action-value function, which is generally not known, we will replace it with a function approximator $Q_w(s,a)$, with parameter vector $w$. In general, assuming that $Q_{\pi_\theta}(s,a) \approx Q_w(s,a)$ will introduce a bias in the policy gradient estimate. However, under certain conditions, as outlined in [41], this approximation can be made without affecting the policy gradient, and in this case, we denote the function approximator as *compatible*. The function approximator of Q can e.g. take the form

$$Q_w(s,a) = \underbrace{(a - \pi_\theta(s))^\top \nabla_\theta \pi_\theta(s)^\top}_{\Psi(s,a)^\top} w + V_v(s), \tag{2.24}$$

where $\Psi(s,a)$ is a state-action feature vector, using the following value function approximation

$$V_v(s) = \Phi(s)^\top v, \tag{2.25}$$

where $\Phi(s)$ is a state feature vector and $v$ is a parameter vector. The state feature vector can be an NN, or monomials of the state for simpler systems. The gradient of the Q-function can then be approximated as

$$\nabla_a Q_{\pi_\theta}(s,a) \approx \nabla_a Q_w(s,a) = \nabla_\theta \pi_\theta(s)^\top w. \tag{2.26}$$

The parameters $v$ and $w$ of the action-value function approximation in (2.24) are given by the solution to the following least squares problem

$$\min_{v,w} \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(Q_{\pi_\theta}(s,a) - Q_w(s,a))^2\right]. \tag{2.27}$$

The problem in (2.27) can be tackled using e.g. a least squares TD-approach (LSTD), as detailed in [42], here specified for $m$ episodes of length $K$, i.e.

$$\begin{aligned} v = \frac{1}{m}\sum_{j=0}^{m}\Bigg\{\Bigg[\sum_{k=0}^{K}\left[\Phi(s_{k,j})(\Phi s_{k,j} - \gamma\Phi(s_{k+1,j}))^\top\right]\Bigg]^{-1} \\ \sum_{k=1}^{K}\left[\Phi(s_{k,j})L(s_{k,j},a_{k,j})\right]\Bigg\}, \end{aligned} \tag{2.28}$$

$$\begin{aligned} w = \frac{1}{m}\sum_{j=0}^{m}\Bigg\{\Bigg[\sum_{k=0}^{K}\left[\Psi(s_{k,j},a_{k,j})\Psi(s_{k,j},a_{k,j})^\top\right]\Bigg]^{-1} \\ \sum_{k=1}^{K}\left[(L(s_{k,j},a_{k,j}) + \gamma V_v(s_{k+1,j}) - V_v(s_{k,j}))\Psi(s_{k,j},a_{k,j})\right]\Bigg\}. \end{aligned} \tag{2.29}$$

## 2.3.3 MPC as a function approximator

In deep RL, NNs are used as function approximators of the value function and/or policy. This means that in theory, arbitrary complex (action)-value functions and policies can be approximated with high accuracy. However, it is difficult to analyze the associated closed-loop behavior. Alternatively, a parameterized MPC scheme can be used as a function approximator in both value- and policy-based RL methods.

We parameterize the OCP in (2.2), and obtain a value function estimate given as

$$
\begin{aligned}
V_\theta(s) = \min_{x,u,\sigma} \quad & \gamma^N (T_\theta(x_N) + \psi_N^\top \sigma_N) + \sum_{i=0}^{N-1} \gamma^i (\ell_\theta(x_i, u_i) + \psi^\top \sigma_i) && \text{(2.30a)} \\
\text{s.t.} \quad & \forall i \in \mathbb{I}_{0:N-1} : x_0 = s, && \text{(2.30b)} \\
& x_{i+1} = f_\theta(x_i, u_i), && \text{(2.30c)} \\
& u_i \in \mathcal{A} && \text{(2.30d)} \\
& h_\theta(x_i, u_i) \leq \sigma_i,\ h_\theta^N(x_N) \leq \sigma_N, && \text{(2.30e)} \\
& \sigma_i \geq 0, \sigma_N \geq 0 && \text{(2.30f)}
\end{aligned}
$$

Note that the OCP in (2.2) describes an undiscounted MPC problem, i.e. a special case of $\gamma = 1$. The function $f_\theta(\cdot, \cdot)$ describes the prediction model, $h_\theta(\cdot, \cdot)$ describes the mixed input and state constraints, and $h_\theta^N(\cdot)$ describes the terminal constraint as before, but now parameterized by $\theta$. Slack variables $\sigma_i$ and $\sigma_N$ are used to prevent the MPC scheme from becoming infeasible due to the possible model mismatch between the true system (2.10) and the prediction model $f_\theta$. We note that the potential stochasticity resulting from the disturbances in (2.10), is addressed firstly by the parameterized prediction model $f_\theta$, as well as by the parameterization of the cost and constraints. The constant vectors $\psi$ and $\psi_N$ should be selected sufficiently large, such that constraint violations are accepted as seldom as possible while still ensuring feasibility [43]. Moreover, the stage cost and terminal cost are also parameterized. Although not clearly visible in the performance measure as defined in (2.14), the mixed input and state inequality constraint in (2.30e) may be incorporated in the RL cost $L(s, a)$, by a term that penalizes constraint violations, e.g.

$$
L(s, a) = L_0(s, a) + \psi^\top \max(0, h(s, a)). \tag{2.31}
$$

For a value function estimate described by (2.30), the policy estimate is given by

$$
\pi_\theta(s) = u_0^\star(s, \theta). \tag{2.32}
$$

By adding an additional constraint to the OCP in (2.30), namely that the first control action equals a given action $a$, we obtain an action-value function estimate:

$$
\begin{aligned}
Q_\theta(s, a) = \min_{x,u,\sigma} \quad & \gamma^N (T_\theta(x_N) + \psi_N^\top \sigma_N) + \sum_{i=0}^{N-1} \gamma^i (\ell_\theta(x_i, u_i) + \psi^\top \sigma_i) && \text{(2.33a)} \\
\text{s.t.} \quad & (2.30b) - (2.30f), && \text{(2.33b)} \\
& u_0 = a. && \text{(2.33c)}
\end{aligned}
$$

The MPC scheme is a valid function approximator of $Q^\star$, given that it satisfies the following relationships

$$\pi_\theta(s) = \arg\min_a Q_\theta(s,a), \quad V_\theta(s) = \min_a Q_\theta(s,a) \tag{2.34}$$

With MPC as a function approximator, RL methods can be applied to adjust the parameters $\theta$ towards the optimal parameters $\theta^\star$. For the optimal parameters $\theta^\star$, the MPC scheme delivers the optimal policy through (2.32), i.e. $\pi_{\theta^\star} = \pi^\star$.

The central justification for using a parameterized MPC scheme in RL follows from the following theorem.

**Theorem 2.** *Suppose that the parametrized stage cost, terminal cost, and constraints in the MPC scheme (2.30) are rich function approximators with adjustable parameters $\theta$. Moreover, the MPC scheme has an exact relaxation (i.e. $\psi, \psi_N$ large enough). Then there exists parameters $\theta^\star$ such that*

$$T_{\theta^\star}(s) = V^\star(s) \tag{2.35a}$$

$$\ell_{\theta^\star}(s,a) = \begin{cases} Q^\star(s,a) - V^\star(f_{\theta^\star}(s,a)) & \textit{if } |V^\star(f_{\theta^\star}(s,a))| < \infty \\ \infty & \textit{otherwise} \end{cases} \tag{2.35b}$$

*and the following holds, $\forall\gamma$:*

1. $V_{\theta^\star(s)} = V^\star(s), \forall s \in \Omega$
2. $\pi_{\theta^\star}(s) = \pi^\star(s), \forall s \in \Omega$
3. $Q_{\theta^\star}(s,a) = Q^\star(s,a), \forall s \in \Omega$, *for the inputs $a \in \mathcal{A}$ such that* $|V^\star(f_{\theta^\star}(s,a))| < \infty$

*if the set*

$$\Omega =: \left\{ s \in \mathcal{S} \,\middle|\, |V^\star(x_k^\star)| < \infty,\ \forall k \leq N \right\}, \tag{2.36}$$

*is non-empty, where $x^\star$ is an optimal state trajectory generated by the MPC scheme in (2.30) i.e. $x_0^\star, \ldots, x_N^\star$.*

*Proof.* The proof follows from [21], [44]. □

The assumption in (2.36) can be interpreted as a type of stability condition for the prediction model $f_{\theta^\star}$ under the optimal policy $\pi^\star$. More specifically, the assumption requires the existence of a non-empty set such that the optimal value function $V^\star$

of the predicted optimal trajectory $x^\star$ based on the prediction model is finite for all initial states starting from this set. Theorem 2 then states that, for a given MDP, an MPC scheme with a possibly inaccurate model can deliver the optimal value functions and optimal policy. The result extends to EMPC, which is also treated in this thesis. A rich parameterization can be achieved by using universal function approximators for the stage, terminal cost and constraints, such as e.g. NNs [45]. In this thesis, we use the following definition of universal function approximators.

**Definition 9.** *A universal function approximator can approximate any continuous function on a closed and bounded set.*

If the parameterization is not rich enough to capture the optimal policy and value functions, RL will find the parameters that best fit the policy or value function for the given set of functions described by the selected parameterization, see e.g. [46].

**Remark 1.** *MPC is most commonly formulated without discounting, i.e. $\gamma = 1$, for which stability analysis can be conducted as outlined in Section 2.1.2. Introducing a discount factor $\gamma \neq 1$ generally complicates the stability analysis. However, recent work has newly defined conditions for the stability of closed-loop systems under discounted MPC schemes, see [47]. Stability analysis in the case of discounted MPC schemes is not considered in this thesis.*

### Sensitivity analysis of MPC

In order to calculate the parameter updates for Q-learning (2.20) and policy gradient methods (2.22), we calculate the gradients of the action-value function and the policy from sensitivity analysis for the underlying MPC scheme in (2.33) and (2.30) respectively.

The Lagrange function for the optimization problem in (2.33) is

$$\mathcal{L}_\theta = \Phi_\theta + \nu^\top G_\theta + \mu^\top H_\theta, \tag{2.37}$$

where $\Phi_\theta$ is the cost (2.33a), $H$ gathers the inequality constraints and $G$ the equality constraints in (2.33). The variables $\nu$ and $\mu$ are Lagrange multipliers associated with the equality constraints and inequality constraints respectively. Let $p$ label the primal decision variables and let $\eta = \{p, \nu, \mu\}$. The solution to the MPC problem (2.33) is then given by $\eta^\star$. The gradient of $Q_\theta(s, a)$ w.r.t the parameters is then

$$\nabla_\theta Q_\theta(s, a) = \nabla_\theta \mathcal{L}_\theta(s, \eta^\star). \tag{2.38}$$

The gradient of the policy estimate $\nabla_\theta \pi_\theta$ required in (2.23), is obtained from sensitivity analysis of the MPC scheme in (2.30). The primal-dual Karush-Kuhn-

Tucker (KKT) conditions are given by

$$R = \begin{bmatrix} \nabla_p \mathcal{L}_\theta \\ G_\theta \\ \mathrm{diag}(\mu)\mathrm{H}_\theta \end{bmatrix} = 0, \tag{2.39}$$

where $\mathrm{diag}(\mu)$ is a diagonal matrix with entries $\mu$. Using the implicit function theorem, it follows that

$$\nabla_\theta \pi_\theta(s) = -\nabla_\theta R(\eta^*, s, \theta) \nabla_\eta R(\eta^*, s, \theta)^{-1} \frac{\partial \eta}{\partial u_0}. \tag{2.40}$$

# Part I

# Combining MPC and supervised learning methods

## Introduction

System modeling represents the first step of an MPC design, and the resulting performance of an MPC scheme relies heavily on a sufficiently accurate model. While established methods for system modeling and identification exist, see e.g. [48], the increasing complexity of systems to be controlled in terms of nonlinearities or scalability challenges the classical system identification techniques.

In the control community, model errors have been accounted for explicitly in the field of robust control, established in the 1980s [49]. Robust control is mainly concerned with linear systems, and is based on incorporating an a priori estimate of the model uncertainty, and then provides guarantees for stability and performance for all uncertain parameters or disturbances that lie within some (typically compact) set. For large uncertainties, robust controllers are naturally conservative in order to maintain formal guarantees.

ML methods have been considered to overcome the limitations of approximate models. Both NNs [45] and GPs [50] are universal function approximators, and therefore suited for learning highly complex dynamics. Although not treated in this thesis, the opportunities for supervised learning methods for MPC go beyond data-driven model improvement. For example, the terminal sets and costs were learned in [51]. Moreover, the region of attraction for a given controller was learned in [20].

For model learning in MPC, the resulting control design will vary with the choice of ML model. One important distinction is between models that provide estimates of the model uncertainty, and those that do not. For learned models that traditionally do not provide a measure of model uncertainty, such as NNs, we can apply robust stability techniques in order to analyze the closed-loop system under a controller that does not consider model uncertainty explicitly. This is done in Chapter 3, where NNs are used to learn the dynamics of a chemical reactor and used as the prediction model in MPC to control the process.

For ML models that do provide estimates of model uncertainty, such as probabilistic formulations [52], we can incorporate this into the MPC formulation. This results in what is known as robust MPC (RMPC). Addressing model uncertainty is essential to ensure constraint satisfaction in case of model inaccuracy and noise. Typically the uncertainty descriptions for RMPC have been fixed and assumed available for control design. Learning-based approaches enable us to estimate the uncertainty based on data, and potentially update the uncertainty description over time, potentially reducing the conservativeness of the robust controller.

As MPC is based on solving an optimization problem at each sampling time, computational efficiency must be considered as learned models are used as the prediction model. Leveraging the full potential of e.g. probabilistic methods in MPC is therefore still an active research field. Convexity is another desired property for MPC. Convex formulations of the cost or constraints generally make the MPC problem much easier to solve [33]. In Chapter 4, we exploit probabilistic learning methods to reduce the conservativeness of a robust controller that can be cast as a convex optimization problem. The concept of convexity is revisited in Chapter 5, where we consider convex formulations of the MPC cost function.

The work presented in the following chapters builds on the introduction given to stability analysis of MPC in Section 2.1.2.

# Chapter 3

# Learning the MPC prediction model with NNs

Learning-based controllers, and especially learning-based model predictive controllers, have been used for a number of different applications with great success. In spite of good performance, a lot of these cases lack stability guarantees. In this chapter, we consider a scenario where the dynamics of a nonlinear system are unknown, but where input and output data are available. A prediction model is learned from data using an NN, which in turn is used in a nonlinear model predictive control (NMPC) scheme. The closed-loop system is shown to be ISS stable with respect to the prediction error of the learned model. The approach is tested and verified in simulations, by employing the controller to a benchmark system, namely a continuous stirred tank reactor (CSTR) plant. Simulations show that the proposed controller successfully drives the system from random initial conditions to a reference equilibrium point, even in the presence of noise. The results also verify the theoretical stability result. This chapter is based on [23].

## 3.1 Introduction

In the following, we focus on a scenario where either the entire or parts of the system dynamics are unknown, but with data available, so that a prediction model can be built or improved offline. In order to be used in MPC, the resulting prediction model should be accurate, while at the same time not too computationally complex.

Different learning methods have been used for building or improving prediction models from data, suited for MPC. In [53], GP regression is used to improve a nominal model by learning a disturbance model, used to design an MPC algorithm for a mobile robot. Similarly, in [54], GPs and local linear regression methods are compared in order to improve the prediction model of a robot. Because GPs are known to scale poorly with the number of data points used in training, learning-based MPC has been tested using sparse GPs, such as in [55]. NNs have also proven to be highly flexible function approximators, and have become state-of-the-art for a variety of challenging tasks, ranging from image analysis and classification to language processing [56], [57]. This learning method has also proved effective for learning the dynamics of nonlinear systems [58], [59], [60]. Consequently, NNs have been used to build prediction models suited for MPC, such as in [61], [62], [63], [64], [65].

For safety-critical applications, a necessary aspect of learning-based MPC is being able to prove the stability of the closed-loop system. For many references on learning-based MPC, this is currently missing. In [14], stability is analyzed for a general class of learning-based MPC, where the dynamics are divided into a nominal, linear model and a function to be learned. Using that the amplitude of the learned function is bounded, robust stability is proved for the linear MPC. Because the bound must hold for all unmodelled nonlinearities in addition to model errors, the assumption is rather conservative. In [19] and [66] the entire system dynamics are learned, using GPs and parameter optimized kinky inference (POKI) respectively, and stability is proved assuming boundedness of the modeling uncertainty. Stability is analyzed for learning-based MPC using NNs, in [67]. However, the suggested MPC design is different and computationally more complex compared to the design proposed in this chapter.

This work is inspired by the stability analyses done in [19] and [66] but is adapted to a case where the system dynamics are learned using an NN. The main contributions of this chapter are:

- Design of an output MPC scheme using a nonlinear autoregressive model with exogenous input (NARX) prediction model learned using an NN
- Conditions for which the closed-loop system is ISS with respect to the prediction error
- Implementation and testing of the control design for a benchmark system

## 3.2 Problem statement

We study nonlinear discrete-time systems, where $y_k \in \mathbb{R}$ is the measured output and $u_k \in \mathbb{R}$ is the control input. It is assumed that the dynamics can be described using a NARX model, given by

$$y_{k+1} = f(x_k, u_k) + w_k, \tag{3.1}$$

where the output is corrupted by noise, $w_k$, which is assumed to lie in a compact set $\mathcal{W}$, that is $w_k \in \mathcal{W} \subset \mathbb{R}$. The input is subject to hard constraints, $u_k \in \mathcal{U}$, and the output is subject to soft constraints, $y_k \in \mathcal{Y}$. The NARX state vector is given by

$$x_k = [y_k, \dots, y_{k-m_y}, u_{k-1}, \dots, u_{k-m_u}], \tag{3.2}$$

where $m_y$ and $m_u$ denotes the number of previous outputs and inputs used to describe the next output so that $x_k \in \mathbb{R}^{m_y+m_u+1}$.

It is assumed that the system dynamics of the plant are unknown and that $f(\cdot,\cdot)$ from (3.1) is an unknown function, but that input and output data are available. The data is used to build a prediction model

$$\hat{y}_{k+1} = \hat{f}(x_k, u_k). \tag{3.3}$$

Throughout this chapter, $(\hat{\cdot})$ will be used to denote either variables that are predicted or functions that are used to make predictions. We want to steer the process from random initial conditions, within the given input and output sets, to a desired reference equilibrium point, given by $(x_{\text{ref}}, u_{\text{ref}})$, where $x_{\text{ref}} \in \mathbb{R}^{m_y+m_u+1}$ and $u_{\text{ref}} \in \mathbb{R}$. To this end, we design an MPC scheme, using the prediction model from (3.3).

**Remark 2.** *Here, we have considered a case that is single-input single-output. However, the control design and the stability results in this chapter can easily be extended to the multi-input multi-output case.*

## 3.3 Data-based prediction model

### 3.3.1 NARX prediction model

In the MPC algorithm, we consider the prediction of the state. We therefore reformulate the NARX model into a state space representation on the form

$$\begin{aligned} x_{k+1} &= F(x_k, u_k, e_k) \\ y_k &= c^\top x_k + w_k, \end{aligned} \tag{3.4}$$

where $c^\top = [1 \quad 0 \quad \dots \quad 0]$ and $e_k$ represents modeling uncertainty, in the form of a prediction error, introduced because we will use the prediction model from

(3.3). The prediction error denotes the difference between the true model (3.1) and the learned prediction model (3.3)

$$\begin{aligned} e_k &:= y_{k+1} - \hat{y}_{k+1} \\ &= f(x_k, u_k) + w_k - \hat{f}(x_k, u_k). \end{aligned} \tag{3.5}$$

Using the definition of the NARX state vector from (3.2), the vector-valued function from (3.4) is given by

$$\begin{aligned} F(x_k, u_k, e_k) = [\hat{f}(x_k, u_k) + e_k, y_k, ..., y_{k-m_y+1}, \\ u_k, ..., u_{k-m_u+1}]. \end{aligned} \tag{3.6}$$

We now define the nominal state space model, for which $\hat{F}(x_k, u_k) \triangleq F(x_k, u_k, 0)$. This is written as

$$\hat{F}(x_k, u_k) = [\hat{f}(x_k, u_k), y_k, ..., y_{k-m_y+1}, u_k, ..., u_{k-m_u+1}], \tag{3.7}$$

where the first argument can be either the previous predicted state, $\hat{x}_k$, or the true state known from measurements, $x_k$, and is used to obtain the next state prediction $\hat{x}_{k+1} = \hat{F}(x_k, u_k)$.

## 3.4 NARX Neural Network

When a multi-layer perceptron (MLP) network is used to approximate the function $f(\cdot)$ in a NARX model, the resulting structure is known as a NARX network [68]. Here, a single hidden layer is proposed for approximation. It has been shown that a single hidden layer feedforward NN with a sufficient number of neurons, is capable of approximating any continuous function to an arbitrary degree of accuracy [69], [45].

Training of a NARX network can be done in two different modes [68]. If the network is in so-called series-parallel mode, the previous outputs are actual values of the system's output. This can be written as

$$\hat{y}_{k+1} = \hat{f}_{\mathrm{NN}}(y_k, ..., y_{k-m_y}, u_k, ..., u_{k-m_u}). \tag{3.8}$$

If the network is in parallel mode, predicted past outputs are used to predict the next output, according to

$$\hat{y}_{k+1} = \hat{f}_{\mathrm{NN}}(\hat{y}_k, ..., \hat{y}_{k-m_y}, u_k, ..., u_{k-m_u}). \tag{3.9}$$

In parallel mode, the NARX network is a recurrent neural network (RNN), with feedback connections enclosing the layers of the network.

Series-parallel mode is generally preferred in training if the actual values of the system's outputs are available. This has two benefits, the first being that using the actual output values gives better accuracy. In addition, the series-parallel architecture results in a purely feedforward network, so that static backpropagation methods can be used during training.

In order to make predictions, the NARX network should be converted to parallel mode after training. When the NARX network is making multi-step ahead predictions, the true output values will no longer be available, and in parallel mode the network will instead use its own output predictions. A NARX network in parallel mode is visualized in Figure 3.1.

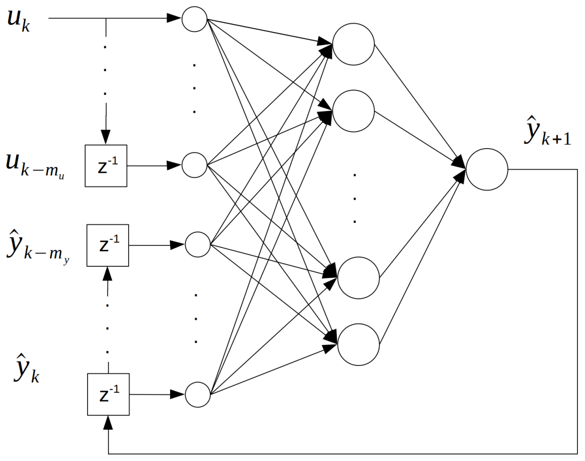


**Figure 3.1:** Visualization of a NARX network in parallel mode, reproduced from [70].

## 3.5 Stabilizing data-based MPC

In the following section, we define a nonlinear MPC scheme that uses the nominal model (3.7) learned from data using a NARX network. In the following we will use the class $\mathcal{K}$ functions as defined in Section 2.1.2.

### 3.5.1 Optimal control problem

MPC is based on solving an open-loop optimization problem at every time step. In each iteration, the optimization routine will minimize a given cost function.

Because we want the MPC scheme to drive the process to a reference state, we use a positive definite stage cost that penalizes deviations from the reference, given by $\ell_s(\cdot,\cdot)$. A barrier function, $\ell_b(\cdot)$, is added to the total stage cost, in order to fulfill soft output constraints, that is to ensure that the output stays in a certain set, $\mathcal{Y} \subseteq \mathbb{R}$, if possible. Based on the stability analysis in Section 3.5.2, $\ell_b(\cdot)$ is designed such that

$$\ell_b(\hat{y}_k) \geq \alpha_b(d(\hat{y}_k, \mathcal{Y})) \tag{3.10}$$

and $\ell_b(\hat{y}_k) = 0 \;\forall\; \hat{y}_k \in \mathcal{Y}$, where $\alpha_b(\cdot)$ is a class $\mathcal{K}$-function and $d(\cdot)$ is the shortest distance from a point $\hat{y}_k \in \mathbb{R}$ to the set $\mathcal{Y} \subseteq \mathbb{R}$.

The total stage cost is then

$$\ell(\hat{x}_k, u_k) = \ell_s(\hat{x}_k - x_{\text{ref}}, u_k - u_{\text{ref}}) + \ell_b(\hat{y}_k), \tag{3.11}$$

where $(x_{\text{ref}}, u_{\text{ref}})$ denotes the reference equilibrium point. Using the stage cost (3.11) and the nominal model (3.7), the final optimization problem is formulated. Note that this is done without defining a terminal constraint [37], resulting in

$$\begin{aligned}
\min_{\hat{x},u} \quad & \sum_{i=0}^{N-1} \ell(\hat{x}_{k+i|k}, u_{k+i|k}) + \lambda V_f(\hat{x}_{k+N|k} - x_{\text{ref}}) && \text{(3.12a)}\\
\text{s.t.} \quad & \forall i \in \mathbb{I}_{0:N-1}: && \text{(3.12b)}\\
& \hat{x}_{k+i+1|k} = \hat{F}_{\text{NN}}(\hat{x}_{k+i|k}, u_{k+i|k}) && \text{(3.12c)}\\
& \hat{x}_{k|k} = x_k && \text{(3.12d)}\\
& \hat{y}_{k+i|k} = c^\top \hat{x}_{k+i|k} && \text{(3.12e)}\\
& u_{k+i|k} \in \mathcal{U}, && \text{(3.12f)}
\end{aligned}$$

where $(\hat{\cdot})$ denotes predicted variables. We let $\hat{F}_{\text{NN}}(\cdot,\cdot)$ denote the nominal state space model (3.7), using the neural-based prediction model (3.9). The control sequence applied over the prediction horizon $N$ is given by $u_k = \{u_{k|k}, ..., u_{k+N-1|k}\}$ and the predicted states are $x_k = \{x_{k|k}, ..., x_{k+N|k}\}$. The optimization problem is subject to hard input constraints, where $\mathcal{U}$ is the set of feasible inputs. In order to guarantee stability, we make use of a terminal cost $V_f(\cdot)$ and an associated terminal control law $\kappa_f(\cdot)$. The terminal cost is weighted by a design parameter $\lambda \geq 1$ and $x_k$ is the initial condition of the NARX state.

### 3.5.2 Stability

We want to consider ISS of the closed-loop system

$$x_{k+1} = F(x_k, \kappa_{\text{MPC}}(x_k), e_k), \tag{3.13}$$

where $\kappa_{\text{MPC}}(x_k)$ is the nominal controller resulting from the optimization problem (3.12), and $F(x_k, \kappa_{\text{MPC}}, e_k)$ is the true model given by (3.6). The stability proof for the learning-based MPC in [19] and [66] is here adapted for an MPC scheme where the prediction model is approximated by an NN.

**Definition 10.** *A system $x_{k+1} = F(x_k, \kappa_{\mathrm{MPC}}(x_k), e_i)$ is ISS if there exists a class $\mathcal{KL}$-function $\beta$ and a class $\mathcal{K}$-function $\gamma$ such that*

$$\|x_k\| \leq \beta(\|x_0\|, k) + sup_{i=0,\ldots,k}\gamma\big(|e_i|\big) \tag{3.14}$$

*for all initial conditions $x_0$, modeling uncertainties $e_i$ and for all $k \geq 0$.*

We first establish stability in the nominal case, for which the prediction error is assumed to be zero so that the true and predicted state is the same. We then consider robust stability, when the prediction error is non-zero, but bounded. For the nominal case, we want to establish that the control error, $\tilde{x}_k = x_k - x_{\mathrm{ref}}$, converges asymptotically to zero.

For now, we consider the following closed-loop system using the nominal model (3.7), with the neural prediction model (3.9)

$$\begin{aligned} \tilde{x}_{k+1} &= \hat{F}_{\mathrm{NN}}(x_k, \kappa_{\mathrm{MPC}}(x_k)) - x_{\mathrm{ref}} \\ &= \tilde{F}_{\mathrm{NN}}(\tilde{x}_k, \kappa_{\mathrm{MPC}}(\tilde{x}_k)). \end{aligned} \tag{3.15}$$

We make use of the following assumptions regarding the total stage cost, $\ell(x_k, u_k)$, the terminal cost, $V_f(\cdot)$, and the terminal controller, $\kappa_f(\cdot)$.

**Assumption 1.** *There exists a terminal control law $\kappa_f(\tilde{x}_k)$, a control Lyapunov function $V_f(\tilde{x}_k)$ and a region defined by $\Omega = \{\tilde{x}_k \in \mathbb{R}^{m_y+m_u+1} : V_f(\tilde{x}_k) \leq \alpha\}$, where $\alpha > 0$, such that $\forall \tilde{x}_k \in \Omega$ the following holds*

*1.*

$$\begin{aligned} \alpha_1(\|\tilde{x}_k\|) \leq V_f(\tilde{x}_k) &\leq \alpha_2(\|\tilde{x}_k\|), \\ V_f(\tilde{F}_{\mathrm{NN}}(\tilde{x}_k, \kappa_f(\tilde{x}_k)) - V_f(\tilde{x}_k)& \\ \leq -\ell(\tilde{x}_k + x_{\mathrm{ref}}&, \kappa_f(\tilde{x}_k)), \end{aligned} \tag{3.16}$$

*2.*

$$\kappa_f(\tilde{x}_k) \in \mathcal{U}, c^{\top}(\tilde{x}_k + x_{\mathrm{ref}}) \in \mathcal{Y}, \tag{3.17}$$

*where $\tilde{F}_{\mathrm{NN}}(\cdot,\cdot)$ is the nominal model (3.15), $\ell(\cdot,\cdot)$ is the stage cost (3.11), $x_{\mathrm{ref}}$ is the reference state, $\alpha_1(\cdot)$ and $\alpha_2(\cdot)$ are class $\mathcal{K}_\infty$-functions and $\mathcal{U}$, $\mathcal{Y}$ are compact input and output sets, respectively.*

**Assumption 2.** *There exists a class $\mathcal{K}_\infty$-function, $\alpha_y(\cdot)$, such that the stage cost (3.11), satisfies $\ell(x_{\mathrm{ref}}, u_{\mathrm{ref}}) = 0$ and $\ell(\tilde{x}_k + x_{\mathrm{ref}}, u_k) \geq \alpha_y(\|\tilde{x}_k\|)$ for all $u_k \in \mathcal{U}$, where $(x_{\mathrm{ref}}, u_{\mathrm{ref}})$ denotes the reference equilibrium point.*

We now propose the following theorem, which is an adaption of Theorem 4 in [37].

**Theorem 3.** *Let Assumptions 1-2 hold, and let $\kappa_{\text{MPC}}(\tilde{x}_k)$ be the resulting control law of the optimization problem in (3.12). Then $\forall\lambda \geq 1$, where $\lambda$ is a weighting factor, there exists a domain of attraction, $\mathcal{X}_N(\lambda)$, defined without terminal constraint, such that for all initial conditions $x_0 \in \mathcal{X}_N(\lambda)$, the closed-loop system (3.15) is asymptotically stable at the origin.*

*Proof.* An outline of the proof is given as follows. Having satisfied Assumption 1-2, Theorem 4 in [37] states that the controller resulting from the solution of the optimization problem (3.12) will stabilize the system (3.15) asymptotically in the set $x_k \in \mathcal{X}_N(\lambda)$. For the complete proof, the reader is referred to [37]. □

The weight of the terminal cost, $\lambda$, can be used to adjust the size of the domain of attraction. The greater $\lambda$, the larger is $\mathcal{X}_N(\lambda)$, but the worse is the approximation, $\lambda V_f(\cdot)$, of the optimal cost.

Having established that the nominal system is asymptotically stable, the robust case will now be considered. To that end, we make use of the following assumptions:

**Assumption 3.** *There exists a constant $\mu < \infty$, such that the prediction error (3.5) is bounded $\forall k$ $|e_k| \leq \mu$.*

**Remark 3.** *The prediction error encompasses the approximation error as well as the measurement noise. The measurement noise is assumed to lie in a compact set and is therefore bounded. In general, it is not possible to prove that the approximation error of NNs is bounded. However, this is still a commonly used assumption [13], [71], [72]. In this case, we can argue that if the prediction error is bounded on the training set, and the training data is representative of data seen in closed-loop operation, then Assumption 3 is reasonable.*

**Assumption 4.** *The nominal model $\hat{F}_{\text{NN}}(x_k, \kappa_{\text{MPC}}(x_k))$ from (3.7) is uniformly continuous in $x_k$ for all $x_k \in \mathcal{X}_N(\lambda)$.*

**Remark 4.** *The nominal model $\hat{F}_{\text{NN}}(x_k, \kappa_{\text{MPC}}(x_k))$ will be uniformly continuous, if the prediction model $\hat{f}_{\text{NN}}(x_k, \kappa_{\text{MPC}}(x_k))$ is uniformly continuous. This is ensured by choosing uniformly continuous activation functions for the neurons in the layers of the NARX network. All the commonly used activation functions in RNNs, such as the sigmoid, tanh and ReLU functions, are uniformly continuous as their derivatives are uniformly bounded. Considering the interval [0,$\infty$), all the above-mentioned activation functions are in addition continuously differentiable, which is a stronger continuity property than uniform continuity.*

The following theorem is an adaption of Theorem 4 in [73].

**Theorem 4.** *Let $\kappa_{\mathrm{MPC}}(x_k)$ be the predictive controller derived from the optimal control problem (3.12) and let Assumptions 1-4 hold. Then, $\Omega_r \subseteq \mathcal{X}_N(\lambda)$ is a robust invariant set for a sufficiently small bound on the uncertainty. The system (3.13) fulfills the ISS property within the robust invariant set $\Omega_r$.*

*Proof.* An outline of the proof is given as follows. Satisfaction of Assumptions 1-2 guarantees asymptotic stability of the nominal system (3.15) as stated by Theorem 3. If Assumptions 3 and 4 also hold, then Theorem 4 of [73] holds directly. Satisfaction of Theorem 4 guarantees ISS for the closed-loop system (3.13). $\square$

**Remark 5.** *Because the continuity requirement to the neural-based prediction model relates to the choice of activation functions, the user has the freedom to select a different number of hidden layers as well as other network architectures, such as other standard RNNs. Nonetheless, MPC is a computational heavy control design, because an optimization problem is solved at every iteration. This provides a valid argument for keeping the architecture as simple as possible in a neural-based MPC.*

**Remark 6.** *ISS can also be shown by establishing uniform continuity of the optimal cost, by adding some additional assumptions regarding the terminal cost and the stage cost used in the objective function. For details, the reader is referred to C1 in Proposition 1 in [73].*

## 3.6 Case study

To test the performance and robustness of the proposed MPC design, the CSTR process is considered. This is often used as a benchmark process for learning-based MPC.

### 3.6.1 The continuous stirred tank reactor

A CSTR process describes the reaction where a reactant is converted from $A \to B$ [74]. The following differential equations are used to model this process

$$\dot{C}_A(t) = \frac{q_0}{V}(C_{Af} - C_A(t)) - k_0 e^{\frac{-E}{RT(t)}} C_A(t) \tag{3.18a}$$

$$\dot{T}(t) = \frac{q_0}{V}(T_f - T(t)) - \frac{\Delta H_r k_0}{\rho C_p} e^{\frac{-E}{RT(t)}} C_A(t) \tag{3.18b}$$

$$+ \frac{UA}{V\rho C_p}(T_c(t) - T(t)) \tag{3.18c}$$

$$\dot{T}_c(t) = \frac{T_r(t) - T_c(t)}{\tau}, \tag{3.18d}$$

**Table 3.1:** CSTR process parameters

| Param. | Definition | Value |
|---|---|---|
| $q_0$ | Reactive input flow | 10 l/min |
| $V$ | Liquid volume in the tank | 150 l |
| $k_0$ | Frequency constant | $6{\cdot}10^{10}$ l/min |
| $E/R$ | Arrhenius constant | 9750 K |
| $-\Delta H_r$ | Reaction enthalpy | 10000 J/mol |
| $UA$ | Heat transfer coefficient | 70000 J/(min K) |
| $\rho$ | Density | 1100 g/l |
| $C_p$ | Specific heat | 0.3 J/(g K) |
| $\tau$ | Time constant | 1.5 min |
| $C_f$ | $C_A$ in input flow | 1 mol/l |
| $T_f$ | Input flow temperature | 370 K |

where $C_A$ [mol/l] is the concentration of the reactant, $T$ [K] is the temperature in the tank, $T_c$ [K] is the temperature of the coolant, and $T_r$ [K] is the coolant temperature reference. For this control problem, we have input and output according to $u = T_r$ and $y = C_A$. The model parameters are given in Table 3.1, and are similar to those used in [19] and [66]. We define the following input constraints, $\mathcal{U} = \{335 \text{ K} \leq \mathrm{T_r} \leq 372 \text{ K}\}$, and output constraints, $\mathcal{Y} = \{0.35 \text{ mol/l} \leq \mathrm{C_A} \leq 0.65 \text{ mol/l}\}$.

### 3.6.2 Obtaining the dataset

The model equations in (3.18) were used in simulation and for generating training data, but otherwise assumed unknown. The equations were implemented in MATLAB, discretized using Euler's method and sampled at $T_s = 0.5$ min. An input signal was designed to do open-loop simulations. The requirement for input design for system identification is that the data needs to be persistently exciting, i.e. the data must contain sufficiently many distinct frequencies [48].

The input signal was designed to cover the relevant area of input-output space, considering the input and output constraints. To this end, the system was excited by a total of 7 sweeping chirp signals, with different amplitudes and length. A total of 17500 data points were used to train the NN. As the resulting training data covers a large part of the input-output space which is considered in closed-loop operation, we assume that a bounded prediction error for the training data implies a bounded prediction error in closed-loop operation.

## 3.6.3 Training the NARX network

Training the NARX network was done in MATLAB, using a series-parallel architecture, as described in Section 3.4. All data was normalized ahead of training, so that all values fall within a range $[-1, 1]$. This was done to compensate for the input data having different scales, as both past inputs and past outputs are fed to the NN. For a NARX network, the size of the input layer is dictated by the number of input and output delays in the NARX model, given by $m_u$ and $m_y$. Different architectures were tested, until we obtained satisfactory performance, using a single hidden layer, with 10 neurons, and $m_u = 1$ and $m_y = 2$ as the number of input and output delays, respectively. The sigmoid function was used in the neurons in the hidden layer, and a linear activation function was used in the output neuron. The data was split randomly into a train, test and validation set, that corresponded to 70/15/15 $\%$ of the data, respectively. A Levenberg-Marquardt algorithm was used for optimization during training. Early stopping was used to prevent the model from overfitting. For multi-step ahead prediction, the NARX network was converted to parallel mode (3.9).

The resulting architecture of the NN represents a trade-off between modeling flexibility, needed to capture the system dynamics, and computational complexity for the resulting control design. An independent dataset with noise, was set aside for validation, not previously seen in training. To quantify the prediction accuracy on the separate validation dataset, the mean squared error (MSE) is introduced

$$\text{MSE} = \frac{1}{N_{\text{pred}}} \sum_{k=1}^{N_{\text{pred}}} \|y_k - \hat{y}_k\|^2. \tag{3.19}$$

The validation results are seen in Figure 3.2, where the NN is tested for $N_{\text{pred}} = 30$, initialized using noisy measurements. For the given prediction horizon we get $\text{MSE} = 2.84 \cdot 10^{-6}$. Here the prediction horizon is relatively long, and the multi-step ahead prediction is fairly accurate for the unseen validation data. In closed-loop model-based control the prediction horizon will often be shorter, and we conclude that the prediction model should be suited.

## 3.6.4 Control of the reactor

The control objective is to drive the process from a random initial condition, to a reference equilibrium point. The following quadratic stage cost was used

$$\begin{aligned}\ell_s(x_k - x_{\text{ref}}, u_k - u_{\text{ref}}) = (x_k - x_{\text{ref}})^\top Q (x_k - x_{\text{ref}}) \\ +(u_k - u_{\text{ref}})^\top R (u_k - u_{\text{ref}}),\end{aligned} \tag{3.20}$$

where $x_{\text{ref}} = [y_{\text{ref}}, y_{\text{ref}}, y_{\text{ref}}, u_{\text{ref}}]$, for the selected choice of $m_u$ and $m_y$, as addressed in Section 3.6.3. The optimization problem defined in (3.12) was solved

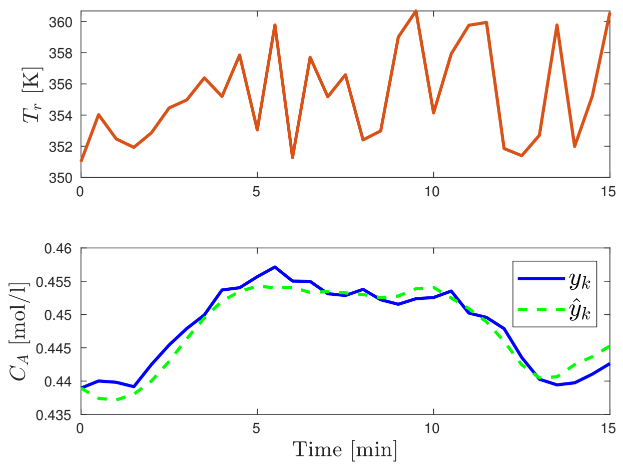


**Figure 3.2:** NARX NN prediction of CSTR concentration. Top: random steps in the coolant temperature reference. Bottom: true and multi-step ahead predictions of the concentration, with the network in parallel mode, using a separate validation set not previously seen by the network. The output is shown in the original scale. The multi-step ahead prediction seems to be fairly accurate for a random input sequence, and for that reason, more complex and larger architectures are avoided in order to keep the prediction model simple.

using `fmincon` in MATLAB. The prediction horizon was set to $N = 5$, and we used $Q = \text{diag}(50, 0, 0, 0)$, $R = 0.2$ and $\lambda = 1$.

### 3.6.5 Finding the reference equilibrium point

Because we consider a system with unknown dynamics, the input reference corresponding to the output reference is not necessarily known. Here it is assumed that the input and output reference point is known a priori, namely $u_{\text{ref}} = 356$ K and $y_{\text{ref}} = 0.439$ mol/l. If this is not the case, the learned prediction model can be used to obtain an estimate of the corresponding input reference, by solving an optimization problem. See for example [63].

### 3.6.6 Soft output constraints

A barrier function was added to the stage cost to account for soft output constraints. The following barrier function was adopted from [19], [66]

$$\ell_b(y_k) = \zeta\left(1 - e^{\frac{-d(y_k,\mathcal{Y})}{\delta}}\right), \tag{3.21}$$

where $\delta = 1$, $\zeta = 100$ and $d(\cdot)$ is given by

$$d(y, \mathcal{Y}) = \inf_{z\in\mathcal{Y}} \|z - y\|_\infty, \tag{3.22}$$

where $\|\cdot\|_\infty$ denotes the infinity norm. For the benchmark system, this means that the coolant temperature reference will not exceed its limit, since input constraints are hard, while the concentration limit is "desirable", given by soft output constraints.

## 3.6.7 Finding the terminal cost

The selected terminal controller and the terminal cost are defined as $\kappa_f(\tilde{x}_k) = K^\top \tilde{x}_k + u_{\text{ref}}$ and $V_f(\tilde{x}_k) = \tilde{x}_k^\top P \tilde{x}_k$ [33], where $\tilde{x}_k = x_k - x_{\text{ref}}$. The terminal cost is found by solving the LQR problem for the linearized model around the reference point, where $P$ is the solution to the discrete Riccati equation, given in (2.7).

The linearized model is found using the learned prediction model. For the choice of $m_y$ and $m_y$, the NARX state vector is given by $x_k = [y_k, y_{k-1}, y_{k-2}, u_{k-1}]$, and consequently the linearized model should be on the form

$$\begin{bmatrix} y_{k+1} \\ y_k \\ y_{k-1} \\ u_k \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_k \\ y_{k-1} \\ y_{k-2} \\ u_{k-1} \end{bmatrix} + \begin{bmatrix} b_{11} \\ 0 \\ 0 \\ 1 \end{bmatrix} u_k. \tag{3.23}$$

Because the learned model is not defined explicitly, the linearized model is obtained numerically [19], [66]. The coefficients of the linearized system can be approximated according to

$$\begin{aligned} a_{11} = \frac{\partial \hat{f}_{\text{NN}}}{\partial y_k},\ a_{12} = \frac{\partial \hat{f}_{\text{NN}}}{\partial y_{k-1}},\ a_{13} = \frac{\partial \hat{f}_{\text{NN}}}{\partial y_{k-2}} \\ a_{14} = \frac{\partial \hat{f}_{\text{NN}}}{\partial u_{k-1}}, b_{11} = \frac{\partial \hat{f}_{\text{NN}}}{\partial u_k}, \end{aligned} \tag{3.24}$$

where $\hat{f}_{\text{NN}}(\cdot,\cdot)$ is the prediction model (3.9). The coefficients in (3.24) were evaluated at $(u_{\text{ref}}, y_{\text{ref}})$ and $(y_{\text{ref}} + \Delta, u_{\text{ref}} + \Delta)$, where $\Delta$ represents a small perturbation. We used $\Delta = 0.015$, found by trial and error.

The coefficients for the linearized system evaluated at the reference point, were found to be $a_{11} = 2.1742$, $a_{12} = -1.5402$, $a_{13} = 0.3958$, $a_{14} = -0.0003$ and $b_{11} = 0.0$. Using `dlqr` in MATLAB, the following gain and solution to the Riccati equation were found:

$$K^\top = \begin{bmatrix} -838.08 & 897.23 & -300.55 & 0.20 \end{bmatrix}, \tag{3.25}$$

$$P = 10^6 \begin{bmatrix} 1.5672 & -1.6776 & 0.5619 & -0.0004 \\ -1.6776 & 1.7959 & -0.6015 & 0.0004 \\ 0.5619 & -0.6015 & 0.2015 & -0.0001 \\ -0.0004 & 0.0004 & -0.0001 & 0.0000 \end{bmatrix}. \tag{3.26}$$

Assumptions 1 - 4 from Section 3.5.2 are fulfilled given the choice of the terminal cost and terminal controller, the selected total stage cost, composed of (3.20) and (3.21), the considerations regarding the prediction error in closed-loop operation and the selected NN architecture and activation functions. By Theorem 4, the considered closed-loop system is ISS.

## 3.7 Simulation results

In order to evaluate the performance of the MPC-NN, using the NARX network as the prediction model, an MPC scheme using the ordinary differential equations (ODEs) (3.18), was implemented. To determine the terminal ingredients for this controller, the linearization of the system was also done numerically, as described by (3.24), but using the ODEs to determine the coefficients. The same MPC cost parameters were used for both the MPC-NN and the MPC-ODE design.

For each controller, simulations were run for $t = 15$ min. For each simulation, random initial conditions were selected from the valid input and output sets. The measurements were corrupted by $2.5\%$ sensor noise. A total of $100$ simulations were run to test for different realizations of noise and initial conditions.

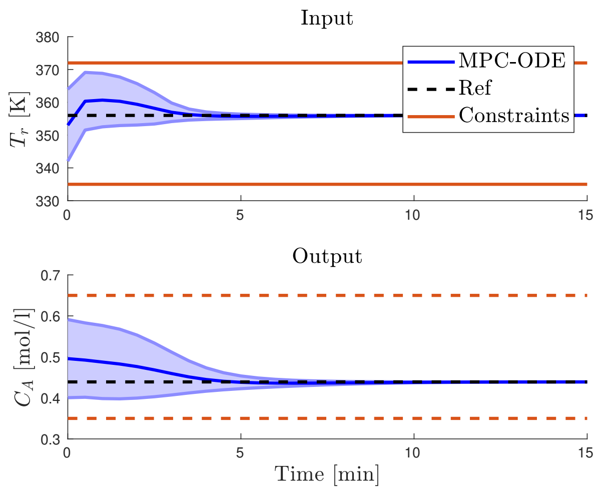
The following performance index was used to evaluate the controller's performance

$$\phi = \frac{1}{N_{\text{sim}}} \sum_{j=1}^{N_{\text{sim}}} \sum_{k=1}^{t_{\text{sim}}} \ell(x_k^j, u_k^j), \tag{3.27}$$

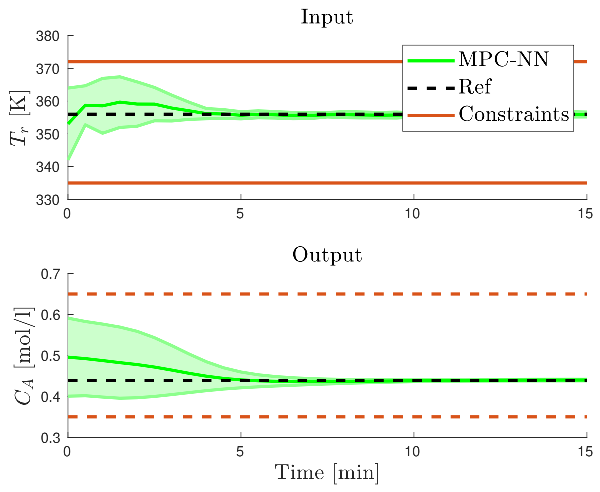
where $x_k$ and $u_k$ are the resulting state and control input for each iteration, $t_{\text{sim}}$ is the number of time steps in one simulation and $N_{\text{sim}}$ is the number of simulations.

In simulations, the MPC-ODE design was expected to be superior, because the true model was used as the prediction model. Comparing the closed-loop results seen in Figure 3.3a and 3.3b, we see that the MPC-NN performs almost as well as the ideal controller. This was also verified by the evaluated performance index (3.27), which was found to be $\phi_{\text{ODE}} = 273.86$ and $\phi_{\text{NN}} = 281.44$. For the tested scenario where the system dynamics are unknown, but input-output data is available, the proposed learning-based MPC design obtained close to ideal behavior.

Simulation results show that the MPC-NN successfully steered the system from all tested random initial conditions, to the reference equilibrium point, even in the presence of noise. In Figure 3.4 we also see that the one-step prediction error stays small and bounded for all realizations of noise. This is important with respect to Assumption 3 in the stability proof. The initial peak in the one-step prediction error is due to the fact that at the beginning of each simulation, the network is initialized using only one noisy, random initial condition.



(a) MPC-ODE



(b) MPC-NN

**Figure 3.3:** CSTR results from 100 simulations. The thick line represents the empirical mean, and the shaded area represents $\pm\sigma$. In each figure, the upper subplot shows the coolant temperature reference and the bottom subplot shows the concentration. Note that the input constraints are hard, while the output constraints are soft

## 3.8 Conclusion

We have presented a NMPC scheme that uses a learned prediction model. The prediction model was learned based on past input and output data, using an NN. The proposed controller is proved to be ISS with respect to the prediction error,

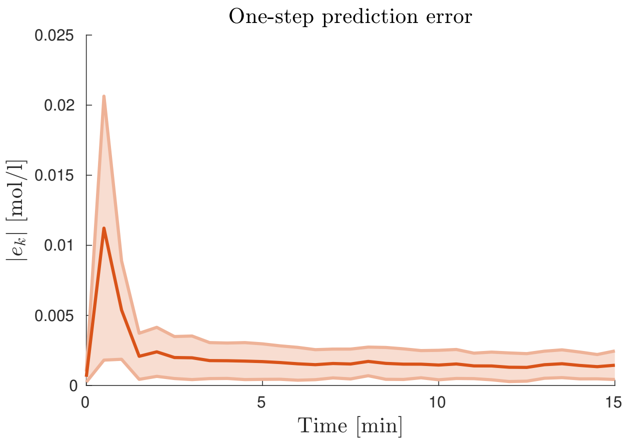


**Figure 3.4:** The one-step prediction error for the prediction model used by the MPC-NN. The thick line represents the empirical mean, and the shaded area represents $\pm\sigma$ from 100 simulations.

assuming that the latter is sufficiently bounded. Because this is an output feedback MPC algorithm, full state information is not necessary, so the control design may be applied to systems where the state might not be measured. The control design was implemented and tested for a CSTR. Compared to an ideal MPC implementation, the learning-based MPC design performed almost as well. Simulations also show the stability of the controlled system for all investigated cases and support the assumption that the prediction error is bounded.

# Chapter 4

# Learning for robust control of sector-bounded systems

For dynamical systems with uncertainty, robust controllers can be designed by assuming that the uncertainty is bounded. The less we know about the uncertainty in the system, the more conservative the bound must be, which in turn may lead to reduced control performance. If measurements of the uncertain term are available, this data may be used to reduce the uncertainty in order to make bounds as tight as possible. In this chapter, we consider a linear system with sector-bounded uncertainty. We develop an MPC algorithm to control the system and use a weighted Bayesian linear regression (WBLR) model to learn the least conservative sector condition using measurements collected in closed-loop. The resulting robust MPC algorithm therefore reduces the conservativeness of the controller and provides probabilistic guarantees of asymptotic stability and constraint satisfaction. The efficacy of the proposed method is shown in simulation. This chapter is based on [24].

## 4.1 Introduction

Even though modern solvers are able to handle NMPC algorithms, solving the resulting optimization problem is in general challenging. The main reason is that for non-convex optimization problems, the solvers are often not guaranteed to find global minima, and may instead get stuck in a local minimum. A way to get around this is to treat the nonlinearities present in the dynamics as uncertainties in a linear system. The linear system can then be controlled using algorithms that are robust with respect to some bounded uncertainty in the system, by solving convex optimization problems. In [75], robust MPC schemes are formulated for polytopic systems and for linear systems with structured uncertainty. A sector bound is particularly suited for modeling state-dependent uncertainty and is used with a similar MPC algorithm in [76], [77], [78].

For uncertain systems, the smallest possible bound on the uncertainty may not be known a priori. Using more conservative bounds will in turn lead to more conservative controllers and correspondingly reduced control performance. If the uncertainty in the system can be measured or estimated, learning-based methods may be used to provide robust controllers with improved performance. In [79], a GP is used to model the disturbances in a vehicle model. The learned model is used to enhance a nominal prediction model in an MPC scheme, resulting in improved path-tracking performance.

A similar approach is taken in [80], using wBLR to learn unknown dynamics in the prediction model. In [81], GP regression is used to learn unmodelled dynamics to be used in stochastic MPC. The learned model complements the prediction model, and the model uncertainty is used to update the chance constraints. Compared to using only a nominal model, the addition of the learned model results in cautious control with improved performance. A GP model is also used in [82] to model the uncertainty in a linear system, ensuring robust constraint satisfaction and resulting in less conservative control.

In this chapter, we consider linear systems with sector-bounded uncertainties, as in [76], [77] and [78]. We develop a robust MPC algorithm similar to the one formulated in [77]. However, instead of assuming that the smallest possible sector is known a priori, we use measurements of the uncertainty to learn the sector bound. This is done using a Bayesian linear regression (BLR) model [83], with weighted data points as in [80], that is particularly suited for finding local approximations of nonlinear functions.

To the best of the authors' knowledge, this has not been done before. The first contribution of this chapter is therefore to use closed-loop measurements to tighten a sector bound of an uncertain nonlinear term. Because the wBLR model estimates distributions rather than deterministic model parameters, it provides a measure of model uncertainty, which is used to formulate a stochastic sector bound, that in turn provides probabilistic stability and constraint satisfaction guarantees for the closed-loop system. The third contribution of this chapter is a reformulation of the optimal infinite horizon problem, formulated in [77], for discrete-time systems.

## 4.2 Problem statement

We consider a subclass of discrete-time nonlinear systems, namely sector-bounded Lur'e systems, which can be written in the form

$$\begin{aligned} x_{k+1} &= Ax_k + G\gamma(z_k) + Bu_k, \\ z_k &= Hx_k, \end{aligned} \tag{4.1}$$

where $x_k \in \mathbb{R}^n$ is the state vector, $u_k \in \mathbb{R}^m$ is the control input and $z_k$ is the input of the nonlinearity $\gamma(z) : \mathbb{R} \to \mathbb{R}$ for $z \in \mathbb{R}$. The system matrices have dimensions $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{n\times 1}$ and $H \in \mathbb{R}^{1\times n}$. We assume that the nonlinearity satisfies

$$(uz - \gamma(z))(\gamma(z) - lz) \geq 0 \quad \forall z, \tag{4.2}$$

where $u, l \in \mathbb{R}^+$, i.e. is bounded by the sector condition as visualized in Figure 4.1.

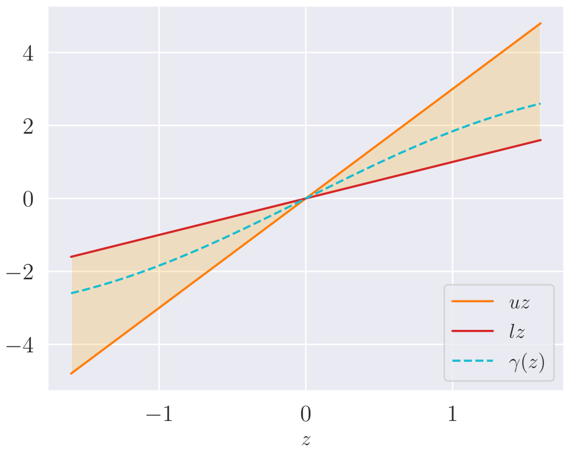


**Figure 4.1:** Sector-bounded nonlinear function, $\gamma(z)$.

The system is subject to $r$ polytopic state and input constraints, of the form

$$\mathscr{C}_k = \left\{ \begin{bmatrix} x_k \\ u_k \end{bmatrix} \in \mathbb{R}^{n+m} : c_j x_k + d_j u_k \leq 1, j = 1, ..., r \right\}, \tag{4.3}$$

that must be satisfied at every time instant $k$. The control objective is to steer the system (4.1) to the origin, for all nonlinearities that satisfy the sector condition (4.2), and for the input- and state-constraints in (4.3).

To this end, a sector condition that is as small as possible can improve the control performance by making it less conservative. For this purpose, both open-loop and closed-loop measurements can be used to reduce the bounds on the uncertainty. In this chapter, we focus on the latter and propose using closed-loop measurements to tighten an initially conservative sector bound. For systems of the form (4.1), where the full state is sampled for $k \geq 0$, we can estimate $\gamma(z_k)$ using the available closed-loop measurements. For each time step, we use $x_{k+1}$ and $x_k$, in combination with (4.1) and the known system matrices $A$, $B$, $G$ and $H$, to obtain an estimate of $\gamma(z_k)$.

## 4.3 Learning the sector bound

The goal of this section is to describe how closed-loop measurements can be used to learn a sector bound, in order to make it as tight as possible. For this purpose we use wBLR, which is an extension of BLR as presented in [83], and a modification of [80]. We assume that we have a dataset of $n$ training samples, $\mathcal{D} = \{z_i, y_i\}_{i=1}^{n}$, with $y_i = \gamma(z_i)$. Consider a local model

$$y_i = wz_i + \epsilon_i, \tag{4.4}$$

where $z_i$ and $y_i$ are the scalar input and output, respectively, and with zero mean Gaussian noise, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, where $\sigma^2$ is the variance. For the sampled region of input space, we want to approximate a locally linear model of the form

$$\hat{y} = wz, \tag{4.5}$$

where $w$ is a stochastic variable. Because the available data points are sampled from a closed-loop system with a fixed sampling frequency, we use scalar weights $l_i \in [0, 1]$ to determine the importance of each data point as done in [80]. As in [80], we assume that the data points are weighted ahead of learning. However, we weigh the datapoints differently, namely by considering the density of data points in input space. To this end, the closed-loop measurements are sorted, and then weighted according to the input points' similarity with both the previous and the next data point. The weight is then scaled using the largest difference between subsequent data points, according to

$$l_i = \frac{0.5\|z_i - z_{i-1}\| + 0.5\|z_i - z_{i+1}\|}{\max(\{\|z_2 - z_1\|, \dots, \|z_n - z_{n-1}\|\})}, \tag{4.6}$$

for $2 \leq i \leq n-1$. For the first and last data point we let $l_i = 0$. For a sample $z_i$ that is very similar to the previous sample $z_{i-1}$ and the next sample $z_{i+1}$, the weight is small, $l_i \approx 0$, and the data point will have little influence on the regression. For the opposite case, the sample will be weighted with $l_i \approx 1$, and the data point is fully included in the regression. If all weights are 1, we obtain standard BLR.

For the weighted data set $\mathcal{D}^l = \{z_i, y_i, l_i\}_{i=1}^n$, we assume that each data point is independent, and distributed according to

$$p(y|Z, w, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(y_i|wz_i, \sigma^2)^{l_i}, \tag{4.7}$$

known as the likelihood function, where $Z$ is a matrix with rows $z_i$ and $y$ is a vector with elements $y_i$. For this likelihood function, the conjugate prior is a Normal Inverse Gamma (NIG) distribution, resulting in the following priors for $w$ and $\sigma^2$ [83]

$$p(w|\sigma^2) \sim \mathcal{N}(w|w_0, \sigma^2 V_0), \tag{4.8a}$$

$$p(\sigma^2) \sim \text{IG}(\sigma^2|a_0, b_0), \tag{4.8b}$$

where $w_0$ is the prior mean and $\sigma^2 V_0$ is the prior variance of the regression weight, and $a_0$ and $b_0$ are the initial parameters determining the Inverse Gamma (IG) distribution. For the specified likelihood (4.7) and prior (4.8), one can show that the posterior distribution over $w$ and $\sigma^2$ has the form [83]

$$p(w, \sigma^2|\mathcal{D}^l) = \text{NIG}(w, \sigma^2|w_N, V_N, a_N, b_N), \tag{4.9}$$

with

$$w_N = V_N(V_0^{-1}w_0 + Z^T L y), \tag{4.10a}$$

$$V_N = (V_0^{-1} + Z^T L Z)^{-1}, \tag{4.10b}$$

$$a_N = a_0 + \frac{\text{tr}(L)}{2}, \tag{4.10c}$$

$$b_N = b_0 + \frac{1}{2}(w_0 V_0^{-1} w_0 + y^T L y - w_N V_N^{-1} w_N), \tag{4.10d}$$

where $\text{tr}(\cdot)$ is the trace operator, and $L$ is a diagonal matrix with the data weights $l_i$. From the joint posterior distribution (4.9), we obtain the marginal distributions for $w$ and $\sigma^2$ [83]

$$p(w|\mathcal{D}^l) = \mathcal{T}(w|w_N, \frac{b_N}{a_N}V_N, 2a_N), \tag{4.11a}$$

$$p(\sigma^2|\mathcal{D}^l) = \text{IG}(\sigma^2|a_N, b_N). \tag{4.11b}$$

In order to define a stochastic model of the sector bound, we use the uncertainty of $w$, given by the Student t-distribution as specified in (4.11a). To make use of the stochastic sector bound in the control design, we use the confidence interval for $w$ to define the upper $\hat{l}$ and lower $\hat{u}$ bound on the local linear approximation of the nonlinear function $\gamma(z)$.

**Assumption 5.** *For the local linear approximation $\hat{y}$*

$$Pr\big((\hat{u}z - \hat{y})(\hat{y} - \hat{l}z) \geq 0\big) \geq p_s \quad \forall \hat{y}, \tag{4.12}$$

*holds with probability, $Pr(\cdot)$, at least $p_s$.*

The stochastic sector bound is used to formulate a robust controller for a system where the exact sector bound is not known. In the next section, Assumption 5 is used in the stability analysis of the resulting closed-loop system.

## 4.4 Learning-based robust MPC

For the system (4.1), constraints (4.3) and the control objective as described in Section 4.2, we propose to use NMPC. The following section is dedicated to describing the control design, as well as analyzing the stability and recursive feasibility of the closed-loop system.

The basic idea of MPC is to solve an optimal control problem online, at each time instant $k$, based on the measured system state $x_{k|k}$. The goal of the optimal control problem is to find a control sequence that minimizes the infinite horizon cost function

$$J_{\infty|k} = \sum_{i=0}^{\infty} F(x_{k+i|k}, u_{k+i|k}), \tag{4.13}$$

as stated for the general LQR in (2.4), but while satisfying constraints. In (4.13) $i \geq 0$ is the prediction time variable and

$$F(x_{k+i|k}, u_{k+i|k}) = x_{k+i|k}^T Q x_{k+i|k} + u_{k+i|k}^T R u_{k+i|k}, \tag{4.14}$$

with $Q \geq 0 \in \mathbb{R}^{n \times n}$ and $R > 0 \in \mathbb{R}^{m \times m}$. Because of the infinite horizon cost function, the resulting constrained optimization problem is generally not solvable. This is overcome by assuming a controller of the form

$$u_{k+i|k} = K_k x_{k+i|k}, \tag{4.15}$$

and at each time instant $k$ calculating a constant feedback matrix $K_k \in \mathbb{R}^{m \times n}$, instead of the complete input trajectory. The resulting optimal control problem is a min-max problem, that determines the feedback matrix $K_k$, which minimizes the upper bound on the infinite horizon cost. The uncertain nonlinear term is treated as a sector condition so that the resulting optimization problem is convex, which we can solve efficiently and for which we can find a global minimum. By recalculating $K_k$, the controller can be more aggressive as the state evolves closer to the origin, compared to the corresponding static feedback law [75].

With the purpose of incorporating the learned sector from Section 4.3 in the control design, we define the following parameters

$$\delta = \frac{\hat{u} + \hat{l}}{2}, \tag{4.16}$$

$$\nu = \frac{\hat{u} - \hat{l}}{2}, \tag{4.17}$$

where $\hat{l}$ and $\hat{u}$ are the lower and upper bound of the learned sector, as given by the confidence interval for (4.11a). The parameters are used to shift the nonlinearity according to

$$\varphi(z) = \gamma(z) - \delta z. \tag{4.18}$$

This is done in order to work with a more convenient expression for the sector bound in the control design. To this end, we use (4.18) to express $\gamma(z)$ in the original sector condition (4.2), and obtain the following inequality

$$(\nu z - \varphi(z))(\varphi(z) + \nu z) \geq 0, \tag{4.19}$$

which implies that

$$|\varphi(z)| \leq |\nu H x|. \tag{4.20}$$

Let $E := \nu H \in \mathbb{R}^{1 \times n}$ and rewrite (4.20) so that

$$\varphi^2(z) \leq x^T E^T E x, \tag{4.21}$$

which in matrix form becomes

$$\begin{bmatrix} x \\ \varphi \end{bmatrix}^T \begin{bmatrix} E^T E & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ \varphi \end{bmatrix} \geq 0. \tag{4.22}$$

We now define

$$\bar{A} = A + \delta G H, \tag{4.23}$$

so that the system (4.1) can be written as

$$\begin{aligned} x_{k+1} &= \bar{A} x_k + G\varphi(z_k) + B u_k, \\ z_k &= H x_k. \end{aligned} \tag{4.24}$$

The following lemma provides conditions for obtaining a stabilizing feedback law (4.15) for a system (4.1) with sector condition (4.2) and an upper bound on the infinite horizon cost (4.13). This is similar to Lemma 2 in [77], but is modified to apply to discrete-time systems and formulates a convex optimization problem without fixing any of the optimization variables. Lemma 1 is also similar to Theorem 1 in [75], but is derived using the sector condition as formulated in (4.2).

**Lemma 1.** *Let Assumption 5 hold, so that for system (4.1), with a sector bound as given by (4.2), and for the matrices* $\Xi = \Xi^T > 0 \in \mathbb{R}^{n\times n}$, $Y \in \mathbb{R}^{m\times n}$, *and scalars* $\lambda > 0$, $\alpha > 0$, *the following inequality*

$$\begin{bmatrix} \Xi & Y^T R^{\frac{1}{2}} & \Xi Q^{\frac{1}{2}} & \Xi E^T & \Xi \bar{A}^T + Y^T B^T \\ R^{\frac{1}{2}} Y & \alpha I & 0 & 0 & 0 \\ Q^{\frac{1}{2}} \Xi & 0 & \alpha I & 0 & 0 \\ E\Xi & 0 & 0 & \lambda & 0 \\ \bar{A}\Xi + BY & 0 & 0 & 0 & \Xi - G\lambda G^T \end{bmatrix} \geq 0. \tag{4.25}$$

*is satisfied. For* $K = Y\Xi^{-1}$ *and* $P = \alpha\Xi^{-1}$, *it holds that*

a. *The feedback law* $u_{k+i|k} = Kx_{k+i|k}$ *asymptotically stabilizes the system (4.1), with the sector-bounded nonlinearity described by (4.2).*

b. $V(x_{k|k}) = x_{k|k}^T P x_{k|k}$ *is an upper bound on the infinite horizon cost (4.13).*

*Proof.* We define $\lambda = \frac{\alpha}{\tau}$ and use that $P = \alpha\Xi^{-1}$, $K = Y\Xi^{-1}$. After some matrix manipulations, we take the Schur complement to the matrix (4.25), and obtain that

$$\begin{bmatrix} \begin{bmatrix} (\bar{A}+BK)^T P(\bar{A}+BK) \\ -P + Q + K^T RK + \tau E^T E \end{bmatrix} & (\bar{A}+BK)^T PG \\ G^T P(\bar{A}+BK) & G^T PG - \tau \end{bmatrix} \leq 0. \tag{4.26}$$

Applying the lossless S-procedure, see e.g. [84], to (4.26), it follows that $v^T \mathcal{Q} v \leq 0$, with $v^T = \begin{bmatrix} x^T & \varphi \end{bmatrix}$ and

$$\mathcal{Q} = \begin{bmatrix} \begin{bmatrix} (\bar{A}+BK)^T P(\bar{A}+BK) \\ -P + Q + K^T RK \end{bmatrix} & (\bar{A}+BK)^T PG \\ G^T P(\bar{A}+BK) & G^T PG \end{bmatrix} \leq 0, \tag{4.27}$$

holds for all $x = x_{k+i|k}$ and $\varphi = \varphi(z_{k+i|k})$ that satisfies (4.22). This holds probabilistically under Assumption 5. The inequality (4.27), is equivalent to

$$\begin{aligned} x^T((\bar{A}+BK)^T P(\bar{A}+BK) - P + Q + K^T RK)x \\ +\varphi G^T P(\bar{A}+BK)x_k + x^T(\bar{A}+BK)^T PG\varphi \\ +\varphi G^T PG\varphi \leq 0. \end{aligned} \tag{4.28}$$

For a feedback law $u_{k+i|k} = Kx_{k+i|k}$ and a quadratic function of the form $V(x_{k|k}) = x_{k|k}^T P x_{k|k}$ where $V(0) = 0$ and $P > 0$, we then know that at sampling time $k$ and $i \geq 0$

$$\begin{aligned} V(x_{k+i+1|k}) - V(x_{k+i|k})) \leq \\ -(x_{k+i|k}^T Q x_{k+i|k} + u_{k+i|k}^T R u_{k+i|k}). \end{aligned} \tag{4.29}$$

Thus, for $Q \geq 0$ and $R > 0$, $V(x_{k|k}) = x_{k|k}^T P x_{k|k}$ is a Lyapunov function and the control law asymptotically stabilizes the system (4.1), proving part $(a)$ of Lemma 1.

Using that $x_{\infty|k} = 0$, so that $V(x_{\infty|k}) = 0$, and summing (4.29) from $i = 0$ to $i = \infty$, we get

$$J_{\infty|k} \leq V(x_{k|k}), \tag{4.30}$$

i.e. $V(x_{k|k})$ is an upper bound on the infinite horizon cost $J_{\infty|k}$, proving part $(b)$ of Lemma 1. □

Lemma 2 is used for showing constraint satisfaction:

**Lemma 2.** *The ellipsoid $\mathscr{E}_k = \{x_k \in \mathbb{R}^n : x_k^T P_k x_k \leq \alpha_k\}$ is contained in the constraint set $\mathscr{C}_k$, as described in 4.3), at time instant $k$ if and only if*

$$(c_j + d_j K_k)(\alpha_k P_k^{-1})(c_j + d_j K_k)^T \leq 1, j = 1, \ldots, r. \tag{4.31}$$

*Proof.* See e.g. [84]. □

Using the results from Lemma 1, we state the following theorem for an NMPC controller with probabilistic stability and constraint satisfaction guarantees, that minimizes the upper bound on the infinite horizon cost function (4.13). This is a discrete-time version of Theorem 1 in [77]. Contrary to this theorem, all matrix inequalities are linear, making the resulting optimization problem easier to solve.

**Theorem 5.** *Let Assumption 5 hold, so that for the system (4.1), with a sector-bounded nonlinearity (4.2), an NMPC scheme is given by solving the following optimization problem:*

$$\min_{\alpha_k, \Xi_k, Y_k, \lambda_k} \alpha_k \tag{4.32}$$

*subject to*

$$\begin{bmatrix} 1 & x_{k|k}^T \\ x_{k|k} & \Xi_k \end{bmatrix} \geq 0 \quad (4.33a)$$

$$\begin{bmatrix} \Xi_k & Y_k^T R^{\frac{1}{2}} & \Xi_k Q^{\frac{1}{2}} & \Xi_k E^T & \Xi_k \bar{A}^T + Y_k^T B^T \\ R^{\frac{1}{2}} Y_k & \alpha_k I & 0 & 0 & 0 \\ Q^{\frac{1}{2}} \Xi_k & 0 & \alpha_k I & 0 & 0 \\ E\Xi_k & 0 & 0 & \lambda_k & 0 \\ \bar{A}\Xi_k + BY_k & 0 & 0 & 0 & \Xi_k - G\lambda_k G^T \end{bmatrix} \geq 0 \quad (4.33b)$$

$$\begin{bmatrix} 1 & c_j\Xi_k + d_jY_k \\ (c_j\Xi_k + d_jY_k)^T & \Xi_k \end{bmatrix} \geq 0 \quad (4.33c)$$

$$j = 1, ..., r$$

*with* $P_k = \alpha_k \Xi_k^{-1}$ *and* $K_k = Y_k \Xi_k^{-1}$. *Then the NMPC scheme has the following properties:*

a. *The optimization problem is feasible for all future time instants* $k$ *if it is feasible at* $k = 0$.

b. *The solution to the optimization problem (4.32)-(4.33) minimizes the upper bound* $V(x_{k|k}) = x_{k|k}^T P_k x_{k|k}$ *on the infinite-horizon cost (4.13) at each time instant* $k$.

c. *If the optimization problem (4.32)-(4.33) is feasible at* $k = 0$, *then the control law*

$$u_{k+i|k} = K_k x_{k+i|k}, \quad i \geq 0, \qquad (4.34)$$

*asymptotically stabilizes the origin of the system (4.1), with sector condition (4.2) and state and input constraints (4.3) for all times* $k \geq 0$.

*Proof.* The proof is divided into three parts, to show that the properties (a)-(c) hold.
*Part (a)*: As only (4.33a) depends on $x_{k|k}$, we know that the solution to the optimization problem (4.32)-(4.33) satisfies constraints (4.33b) and (4.33c). As inequality (4.33b) is identical to (4.25) from Lemma 1, (4.29) in combination with (4.33a) means that $x_{k+1|k}^T P_k x_{k+1|k} \leq x_{k|k}^T P_k x_{k|k} \leq \alpha_k$. Hence, the solution to the optimization problem at time $k$ is also a solution at time $k+1$. By induction, feasibility at time $k+1$ leads to feasibility at $k+2, k+3, \ldots$
*Part (b)*: From Lemma 1 we have that $V(x_{k|k})$ is an upper bound on the cost function (4.13) at time $k$. Because (4.33a) is equivalent to $x_{k|k}^T P_k x_{k|k} \leq \alpha_k$, minimizing $\alpha_k$ implies minimizing the upper bound on the cost function.

*Part (c)*: We now consider stability for when $P$ is recalculated at every sampling instant, $k$. From Lemma 1, we have that applying the control law (4.34), leads to $x_{k+1|k}^T P_k x_{k+1|k} \leq x_{k|k}^T P_k x_{k|k}$. At the next sampling instant, the previous solution to the optimization problem is feasible, but not necessarily optimal, i.e. $x_{k+1|k+1}^T P_{k+1} x_{k+1|k+1} \leq x_{k+1|k+1}^T P_k x_{k+1|k+1}$. Combining these two inequalities, yields $x_{k+1|k+1}^T P_{k+1} x_{k+1|k+1} \leq x_{k|k}^T P_k x_{k|k}$. Thus, $x_{k|k}^T P_k x_{k|k}$ is a strictly decreasing Lyapunov function for the closed-loop system, implying that $x_{k|k} \to 0$ as $k \to \infty$.

It remains to show constraint satisfaction. Having satisfied the conditions of Lemma 1, we know that the state lies in the ellipsoid $\mathscr{E}_k = \{x_k \in \mathbb{R}^n : x_k^T P_k x_k \leq \alpha_k\}$. From Lemma 2, $\mathscr{E}_k$ lies in the constraint set $\mathscr{C}_k$ if (4.31) holds. It can be shown that (4.33a) is equivalent to (4.31). We let $u_{k+i|k} = K_k x_{k+i|k}$ replace $u$ in (4.3), so that $\mathscr{C}_k = \left\{x_k \in \mathbb{R}^n : (c_j + d_j K_k) x_k \leq 1, j = 1, ..., r\right\}$, followed by some matrix manipulation. Because $\mathscr{E}_k$ is invariant, and contained in the constraint set $\mathscr{C}_k$, all states are guaranteed to satisfy input and state constraints. $\square$

**Remark 7.** *Closed-loop stability, input and state constraint satisfaction are guaranteed probabilistically by feasibility of the linear matrix inequalities at initial time. Because we consider a confidence interval at every time step, the resulting probability may be smaller than that given by Assumption 5.*

**Remark 8.** *For some initial conditions, the sector condition may be too restrictive, so the set of linear matrix inequalities does not have a feasible solution. Reduction of the conservative sector using learning will increase the number of feasible initial conditions.*

## 4.5 Simulation results

To test the proposed control design, we consider the dynamics of a flexible link robotic arm, as given in e.g [77]. The system is discretized using the forward Euler method, to obtain the same form as (4.1), resulting in

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -48.6\Delta t & -0.25 & -48.6\Delta t & 0 \\ 0 & 0 & 1 & \Delta t \\ 19.5\Delta t & 0 & -16.7\Delta t & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 21.6\Delta t \\ 0 \\ 0 \end{bmatrix} \tag{4.35}$$
$$G^T = \begin{bmatrix} 0 & 0 & 0 & -3.33\Delta t \end{bmatrix}, \quad H = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix},$$

with time step $\Delta t = 0.01$s. For the robotic arm we have that $\left[x_1, x_2, x_3, x_4\right]^T = \left[\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\right]^T$, where $\theta_1, \theta_2$ are angles, and $\dot{\theta}_1, \dot{\theta}_2$ are angle rates. The nonlinearity, given by

$$\gamma(z_k) = \sin(z_k) + z_k, \tag{4.36}$$

is assumed unknown, but estimated as described in Section 4.2. The following state and input constraints apply

$$u_k \in [-1.5, 1.5], \quad x_{1,k}, x_{3,k} \in [-\frac{\pi}{2}, \frac{\pi}{2}], \quad k \geq 0. \tag{4.37}$$

The control objective is to steer the system (4.35) to the origin. The weighting matrices were selected as in [77]

$$Q = \text{diag}([1, 0.1, 1, 0.1]), \quad R = 0.1. \tag{4.38}$$

A conservative sector was defined using the prior distribution for $w$ and $\sigma^2$ (4.8). The parameters of the prior distribution were specified such that the resulting conservative sector was contained within the first and third quadrants of the input coordinate plane. We used $w_0 = 3.0$ and $\sigma^2 V_0 = 2.25$ in (4.8a), and $a_0 = 90$, $b_0 = 50$ in (4.8b). The conservative sector was then defined using the $95\%$ confidence interval for the prior distribution and is visualized in Figure 4.2. The first simulation was run for initial condition $x_0 = [1.2, 0.1, 0.1, 0.1]$, with the conservative sector formulation. The closed-loop measurements were then sorted, and weighted according to (4.6). The weighted data set was then used to train the wBLR model, using the equations (4.10). A $95\%$ confidence interval for the posterior distribution of $w$ (4.11a), resulted in the following upper bound, $\hat{u} = 2.63$, and lower bound, $\hat{l} = 1.39$, for the learned sector. The learned sector is visualized in Figure 4.2. For the next 20 simulations, the MPC algorithm was tested with two

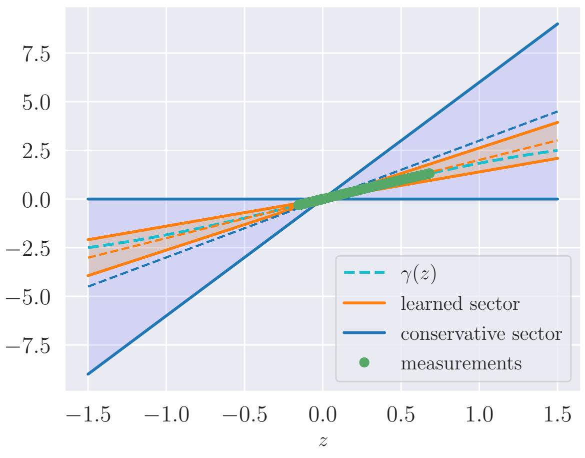


**Figure 4.2:** A conservative (blue) and a learned (orange) sector bound on the nonlinear function, $\gamma(z)$.

versions of the sector, namely

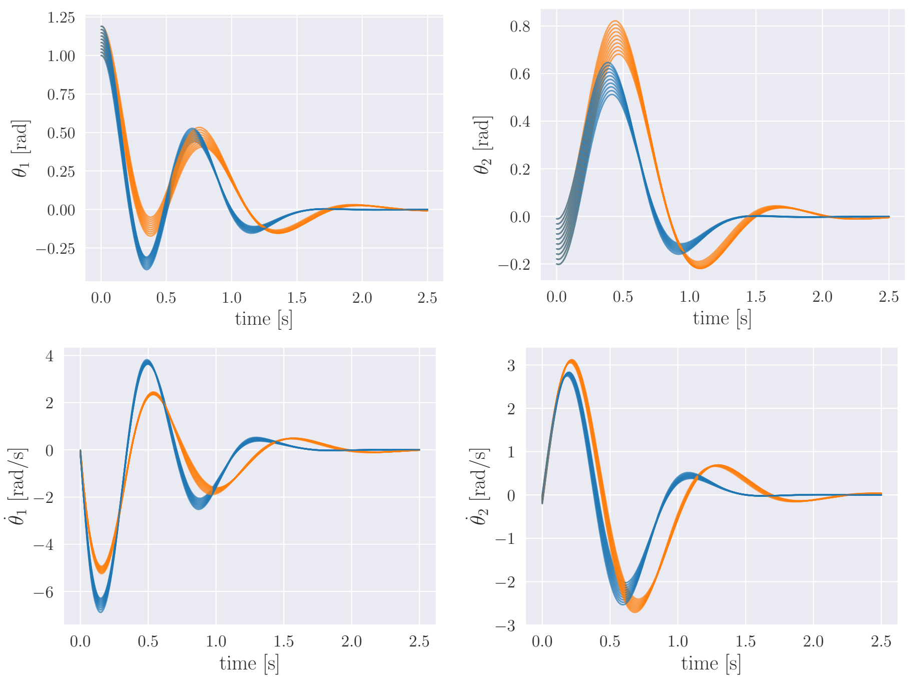


**Figure 4.3:** Simulation results generated using (1) a conservative sector (blue), (2) a sector learned offline (orange).

1. the conservative sector, as defined above
2. the sector learned offline with data from the first simulation

The optimization problem (4.32)-(4.33) was solved using *CVXOPT* in Python [85]. We tested 20 different initial conditions in a region around $x_0 = [1.2, 0, 0, 0]$, according to

$$\tilde{x}_{i,0} = x_{i,0} + [-\delta, \delta], \quad i = 1, \dots, 4 \tag{4.39}$$

with $\delta = 0.2$. The simulation results for when both sectors were feasible in closed-loop are plotted in Figure 4.3 and 4.4. In order to compare the closed-loop performance, we calculated the closed-loop cost according to

$$\phi = \sum_{k=1}^{k_{\text{sim}}} x_k^T Q x_k + u_k^T R u_k, \tag{4.40}$$

resembling (4.14), but summed over the simulation length denoted by the time index $k_{\text{sim}}$. Using the conservative sector, fewer of the initial conditions rendered

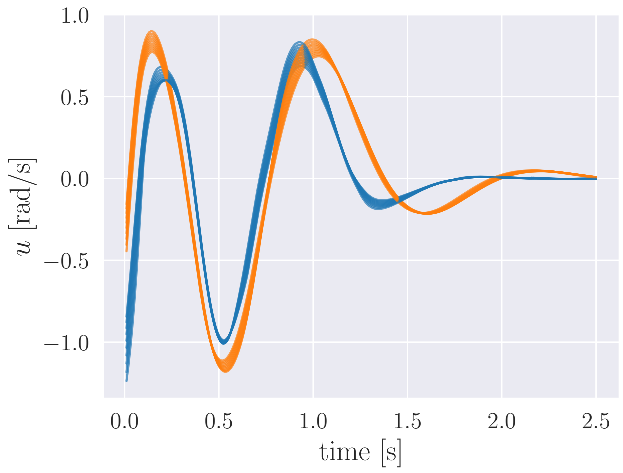


**Figure 4.4:** Control inputs using a conservative (blue) and a learned (orange) sector bound on the nonlinear function, $\gamma(z)$.

feasible optimization problems due to a more conservative sector bound in Theorem 5. Table 4.1 shows the mean of (4.40) for all simulations where both versions of the sectors were feasible, and in addition the total number of feasible simulations.

**Table 4.1:** Comparison of closed-loop performance

| MPC scheme w/ | $\phi_{\text{mean}}$ | Num. of feasible simulations |
|---|---|---|
| (1) Conservative sector | 171.97 | 10 |
| (2) Learned sector | 156.55 | 20 |

For the proposed control design, optimization is performed at every time step in order to recalculate $K$. Because the resulting optimization problem is convex, this can be solved very efficiently. For high-frequency systems, optimization may also be done at a lower frequency than control, maintaining stability but with a small performance loss in between updates of $K$.

Simulations verify that the proposed design can be used for robust control of linear systems with sector-bounded uncertainties, where the sector bound is not initially known. The stochastic sector allows for a conservative initial formulation, based on the best guess on the uncertainty. By exploiting available closed-loop measurements, the uncertainty of the initial, stochastic sector can be reduced, resulting in a smaller sector bound. For the smaller sector, the quadratic cost of the input and state is reduced, and the feasible region of the optimization problem is enlarged.

## 4.6 Conclusion

In this chapter, we have shown how measurements in closed-loop can be used to define a stochastic sector condition, for robust control of linear systems with sector-bounded nonlinearities. A wBLR model is trained with closed-loop measurements of the nonlinearity and used to reduce the conservativeness of the robust controller. The proposed MPC design is tested in simulations of a flexible link robotic arm. Comparing the closed-loop performance for a conservative sector with a sector that incorporates learning, shows that the latter reduces the quadratic cost of the state and input over the simulation length, and results in an enlarged feasible region for the control optimization problem. The approach is currently limited to systems without disturbances and with a certain constraint formulation.

# Part II

# MPC as a function approximator in RL

## Introduction

RL is a powerful tool for tackling MDPs without depending on a model of the system. RL has recently drawn increasing attention due to its accomplishments for robotics and games, see e.g. [4] and [2], using NNs as function approximators to capture the policy and potentially also the value functions. As the policies are based on NNs, analyzing the resulting closed-loop behavior is difficult.

The term safe RL is used to describe efforts made to learn policies that optimize performance while ensuring reasonable system performance and/or respecting safety constraints during learning and/or deployment. An extensive survey of these methods is detailed in [86]. Safety filters have also been considered for learning-based controllers in general, including policies learned with RL, based on e.g. invariance-based approaches [87] and stochastic MPC as in [88].

RL can also be exploited as a tool to improve the performance of an MPC scheme. MPC schemes are often a rough approximation of the true (possibly stochastic) optimal control problem due to the finite horizon, simplified models, and neglected stochastic disturbances. In [21], the authors suggest using parameterized MPC schemes as a function approximator of the policy and value function in RL. Parameterizing the MPC problem allows RL to improve the policy as data is acquired while maintaining an MPC structure, which offers rich tools to analyze the resulting closed-loop behavior. In the following, we will refer to MPC as a function approximator in RL, because we can use the parameterized MPC scheme to capture the policy or value function of an MDP. An alternative understanding of this combination of MPC and RL is that we use RL to tune the parameters of the MPC towards the optimal policy dictated by the RL performance measure, which in turn is defined by the given cost function.

Bayesian optimization has to some extent also been considered for adjusting the parameterization of optimal control problems, such as for unconstrained optimal control in [89], and for constrained optimal control in [90], where the parameters of a linear prediction model were learned for a system where the true dynamics were potentially nonlinear. Automatic tuning of the cost function was also considered in [91], specifically for controllers based on convex optimization problems.

Another approach to learning the parameterization of an optimal control problem is using inverse optimal control to learn from demonstrations. For most applications of inverse optimal control, the underlying model is assumed known and the aim is to infer the parameters of the underlying cost, typically assumed to be a quadratic cost with unknown weights. Moreover, it is assumed that the demonstrations are solutions to the optimal control problem. To account for potential suboptimal execution and noisy data, the optimality conditions are relaxed. This approach was demonstrated for linear system dynamics in e.g. [92] and for nonlinear systems in e.g. [93].

In this part of the thesis, we will focus on the framework where MPC is used as a function approximator in RL, which also extends to a more general type of MPC scheme, namely economic MPC (EMPC). In the preceding part of the thesis, we have considered tracking MPC schemes, with quadratic cost functions, defined by a squared distance measure from the predicted state and input to the reference state and input. Unlike tracking MPC, EMPC is characterized by generic cost functions, that typically represent economic or performance-oriented objectives, addressing the energy, time, or financial cost of running a system. Because the stage cost is not necessarily positive definite, stability analysis established for tracking MPC cannot be used directly. Dissipativity is a fundamental tool that can be used to address the stability of EMPC schemes with not necessarily positive definite stage costs [94].

In Chapter 5 we apply dissipativity theory, in order to design cost parameterizations that are stable by design, which can be learned using value-based RL methods. In Chapter 6 we address the lack of guarantees provided for value-based methods, in terms of obtaining the optimal policy, and propose using a combination of value-based and policy gradient-based methods. A combination of methods allows us to learn both the optimal value function as well as the optimal policy. The proposed combination of methods also extends to a second-order approach, for which we achieve a faster learning rate than using either value-based or policy gradient-based second-order methods on their own.

For the selected framework, we are learning from interactions with closed-loop systems, and efficient learning depends on efficient methods for exploration. In Chapter 7, we propose an exploration scheme based on an uncertainty measure of the MPC-based value function approximation.

The work presented in the following chapters is implemented in Python, using CasADI to formulate the MPC problem [95], and solving them with the IPOPT solver [96]. The general problem descriptions follow the notation used to describe MDPs as in Section 2.2, which will also result in some notational differences in this part of the thesis compared to the previous. In the following chapters, $s$ and $a$ are used to denote the true state and input, whereas $x$ and $u$ are used to denote the predicted state and input respectively. Moreover, we no longer use the notation $x_{k+i|k}$ to describe that $x$ is predicted $i$ steps ahead from time $k$ when the state was sampled, but use the simplified notation $x_i$. Finally, we use the RL methods as outlined in 2.3.

# Chapter 5

# Cost modifications for learning-based MPC

It has recently been shown that adapting not only the MPC model but also its cost function is conducive to achieving optimal closed-loop performance when an accurate model cannot be provided. In the learning context, this modification can be performed via parametrizing the MPC cost and adjusting the parameters via e.g. RL. In this framework, simple functions used as cost modifications can be effective, but the underlying theory suggests that rich functions in principle can be useful. In this chapter, we propose such a cost modification using a class of NNs that preserves convexity. This choice avoids creating difficulties when solving the MPC problem via sensitivity-based solvers. In addition, this choice of cost parametrization ensures nominal stability of the resulting MPC scheme. Moreover, we detail how this choice can be applied to EMPC problems where the cost function is generic and therefore does not necessarily fulfill any specific property. The following chapter is based on the work in [25].

## 5.1 Introduction

One approach to learning-based MPC is using cost modifications to handle model imperfection. In [21], it was proved that an MPC scheme can deliver the optimal policy for a system, even if the model in the MPC is inaccurate. This can be achieved under some fairly mild conditions via modifications of the cost of the MPC scheme, which compensates for model inaccuracy. This idea can be applied in practice by parametrizing the MPC cost and constraints and using learning techniques to adapt the parameters. In that context, RL has been extensively investigated as a learning tool for tuning the cost, constraints and MPC model, see e.g. [22], [97], [98], [99].

The idea of compensating model inaccuracy with cost modifications, has successfully been tested in [21], [100], [101], using fairly simple parametrizations of the cost. However, the theory underlying this result suggests that in principle the cost parametrization should be rich i.e. using universal function approximators. Rich parametrizations of the cost in the context of economic nonlinear MPC (ENMPC) were first considered in [46]. In this chapter, we elaborate on this early investigation and propose a more complete framework to provide such a rich parametrization. More specifically, we propose to use a class of NNs that preserve convexity. This choice has two important benefits. First, ensuring convexity of the MPC cost alleviates the difficulties inherent to solving MPC schemes numerically using sensitivity-based solvers. Second, the stage cost in the MPC scheme must be lower-bounded by a $\mathcal{K}_\infty$-function to ensure stability. A convex function can be designed to satisfy this lower bound, and in turn, ensures nominal stability of the resulting MPC scheme.

In this chapter, we will apply the framework of convex cost modifications to ENMPC problems. ENMPC is concerned with optimizing performance rather than penalizing deviations from a given reference. This means that the cost function not necessarily can be lower-bounded by a $\mathcal{K}_\infty$-function [33]. Dissipativity theory is a fundamental tool in order to understand the closed-loop stability of ENMPC. For dissipative problems, there exist corresponding tracking MPC schemes that yield the same policy as the ENMPC scheme. As tracking MPC schemes typically use quadratic cost functions, they intrinsically satisfy the lower bound. The dissipativity of a problem is verified through the existence of a storage function that satisfies the dissipativity inequality [102]. Finding a valid storage function for the general problem is hard, but may be captured using RL techniques as suggested in [21] and justified in [103]. As we are focusing on economic problems, we use deterministic systems as proof of concept, for which general dissipativity theory is valid.

Even for dissipative problems, the ideal modified stage cost may not be convex. This means that enforcing convexity as a means to ensure stability, may impose limitations on the learned cost function. In [104], the authors showed that for a dissipative problem, a tracking MPC with a quadratic stage cost is locally equivalent to ENMPC. In other words, a convex cost approximation is at least valid locally.

The main contribution of the chapter is the introduction of convex NNs as cost modifications in MPC schemes with imperfect prediction models. Using the MPC scheme as a function approximator for the value function and the policy, we will use RL to adjust the cost parameters, including the NN weights, in pursuance of the optimal economic policy. The second contribution of the chapter is the combination of RL methods for when neither value-based nor policy-based RL methods alone are sufficient. We let one simulation example serve as a proof of concept, and then consider a second simulation example to benchmark the addition of convex cost modifications against the standard quadratic cost parametrization.

## 5.2 Background and problem statement

In the following, we consider a special case of MDPs as outlined in Section 2.2, defined by discrete-time, constrained dynamic systems of the form:

$$s_{k+1} = f(s_k, a_k), \quad h(s_k, a_k) \leq 0, \tag{5.1}$$

where $k$ denotes the discrete time step, $s_k \in \mathbb{X} \subseteq \mathbb{R}^n$ denotes the state and $a_k \in \mathbb{U} \subseteq \mathbb{R}^m$ denotes the input. The (possibly nonlinear) dynamics are defined by $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. The function $h(s_k, a_k)$ describes a mixed input-state constraint. We propose how to formulate stable MPC schemes by parameterizing the cost function, for an economic problem where the stage cost $L : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ is indefinite, using a (potentially) inaccurate model of the system, $\tilde{f}$.

### 5.2.1 Economic NMPC

The following section will describe the ENMPC formulation and recall dissipativity theory as a tool to analyze stability. The following standard assumptions for ENMPC are used in the rest of the chapter. The set of states $s \in \mathbb{X}$ and inputs $a \in \mathbb{U}$ define the set of feasible state-input pairs as follows

$$\mathbb{Z} = \{(s, a) \in \mathbb{X} \times \mathbb{U} \mid f(s, a) \in \mathbb{X},\ h(s, a) \leq 0\}, \tag{5.2}$$

for which we need the following assumption.

**Assumption 6.** *(Properties of constraint sets) The set $\mathbb{Z}$ is compact and non-empty.*

**Assumption 7.** *(Continuity of cost and system) The functions $f(\cdot)$ and $L(\cdot)$ are continuous on $\mathbb{Z}$.*

The optimal steady state pair $(s_e, a_e)$ is defined as follows:

$$(s_e, a_e) = \arg\min_{(s,a)\in\mathbb{Z}} \{L(s, a) \mid s = f(s, a)\}. \tag{5.3}$$

We define a *shifted stage cost* as:

$$\ell(s, a) = L(s, a) - L(s_e, a_e), \tag{5.4}$$

so that $\ell(s_e, a_e) = 0$.

Let $\pi$ denote a deterministic policy, that maps from a state to an action, $\pi : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The optimal policy $\pi^\star$ is then given by the solution to the following infinite-horizon problem

$$\begin{aligned} V^\star(s) = \min_{\pi} \quad & \sum_{j=0}^{\infty} \ell(x_j, \pi(x_j)) && (5.5a)\\ \text{s.t.} \quad & \forall j \in \mathbb{I}_{\geq 0} : x_0 = s, && (5.5b)\\ & x_{j+1} = f(x_j, \pi(x_j)), && (5.5c)\\ & h(x_j, \pi(x_j)) \leq 0, && (5.5d) \end{aligned}$$

where $x_j$ denotes the predicted state, not to be confused with the true state $s_k$, so that $(x_j)_{j=1}^{\infty}$ is the predicted state trajectory for the system starting at some initial state $x_0 = s$, subject to policy $\pi$. The optimal action-value function is defined as follows

$$Q^\star(s, a) = \ell(s, a) + V^\star(f(s, a)). \tag{5.6}$$

The action-value function and the value function are related through the underlying Bellman equations [32]

$$\pi^\star(s) = \text{arg}\min_{a} Q^\star(s, a),\ V^\star(s) = \min_{a} Q^\star(s, a). \tag{5.7}$$

## 5.2.2 Strict dissipativity

A generic economic stage cost can make it challenging to establish the stability of the closed-loop system. Satisfaction of the dissipativity conditions entails that the ENMPC scheme can be recast as a tracking MPC scheme, for which closed-loop stability properties are straightforward to prove [102]. Next, we define the concept of strict dissipativity.

**Definition 11.** *(Strict dissipativity) The system (5.1) with stage cost $L(\cdot)$ is strictly dissipative if there exists a storage fuction $\lambda : \mathbb{X} \rightarrow \mathbb{R}$ satisfying*

$$\lambda(f(s, a)) - \lambda(s) \leq -\rho(||s - s_e||) + \ell(s, a), \tag{5.8}$$

*where $\rho \in \mathcal{K}_\infty$ and $||\cdot||$ denotes the Euclidean norm.*

**Assumption 8.** *(Strict dissipativity) The system (5.1) is strictly dissipative.*

For the storage function, we make the following assumption.

**Assumption 9.** *(Continuity of storage function) The storage function $\lambda(\cdot)$ is continuous on $\mathbb{Z}$.*

Without loss of generality, we can add a constant to the storage function, in order to ensure that $\lambda(s_e) = 0$, without invalidating inequality $(5.8)$. If $\lambda$ exists, we can define the rotated stage cost as

$$\bar{\ell}(s,a) = \ell(s,a) + \lambda(s) - \lambda(f(s,a)). \tag{5.9}$$

Combining (5.8) and (5.9) then yields

$$\rho(||s - s_e||) \leq \bar{\ell}(s,a), \quad \bar{\ell}(s_e, a_e) = 0. \tag{5.10}$$

For a strictly dissipative problem, the ENMPC scheme is equal to a tracking MPC, using the rotated stage cost $\bar{\ell}$. As the rotated stage cost is zero at the optimal steady state and lower-bounded by a $\mathcal{K}_\infty$-function, the closed-loop system is stable [33]. The corresponding tracking MPC is formulated as

$$\begin{aligned} V^\star(s) = \min_{\pi} \; & -\lambda(s) + \sum_{j=0}^{\infty} \bar{\ell}(x_j, \pi(x_j)) && \text{(5.11a)} \\ \text{s.t.} \; & (5.5b), (5.5d). && \text{(5.11b)} \end{aligned}$$

To formulate the finite-horizon MPC, we introduce the finite-horizon stage cost, $\hat{\ell}$, and add a terminal cost, according to

$$\begin{aligned} V^\star(s) = \min_{u,x} \; & -\lambda(s) + T(x_N) + \sum_{j=0}^{N-1} \hat{\ell}(x_j, u_j) && \text{(5.12a)} \\ \text{s.t.} \; & \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, && \text{(5.12b)} \\ & x_{j+1} = f(x_j, u_j), && \text{(5.12c)} \\ & h(x_j, u_j) \leq 0, && \text{(5.12d)} \\ & x_N \in \mathbb{X}_f, && \text{(5.12e)} \end{aligned}$$

where $N$ is the horizon length, $T : \mathbb{X}_f \to \mathbb{R}$ is a penalty on the terminal state and $\mathbb{X}_f$ is a compact terminal region containing the steady state operating point in its interior. The resulting input sequence is $u = \{u_0, \ldots, u_{N-1}\}$, and $x = \{x_0, \ldots, x_N\}$ is the corresponding state trajectory. Note that we use $\hat{\ell}$ to denote the finite-horizon stage cost, to clearly distinguish it from the infinite-horizon stage cost $\bar{\ell}$, as these may not be the same. The stage cost $\hat{\ell}$ must be selected such that it satisfies a lower bound as defined for $\bar{\ell}$ in (5.10). For the terminal cost, we make the following assumptions.

**Assumption 10.** *(Continuity of terminal cost) The terminal cost $T(\cdot)$ is continuous on $\mathbb{X}_f$.*

**Assumption 11.** *(Stability assumption) There exists a compact terminal region $\mathbb{X}_f \subseteq \mathbb{X}$, containing the point $s_e$ in its interior, and terminal control law $\kappa_f : \mathbb{X}_f \to \mathbb{U}$ such that*

$$T(f(s, \kappa_f(s))) - T(s) \leq -\hat{\ell}(s, \kappa_f(s)), \tag{5.13}$$

*$\forall s \in \mathbb{X}_f$ and $(s, \kappa_f(s)) \in \mathbb{Z}$. Moreover, $T(s_e) = 0$ and $T(s) > 0 \; \forall s \in \mathbb{X}_f \setminus \{s_e\}$.*

**Remark 9.** *This assumption requires that for each $s \in \mathbb{X}_f$, $f(s_k, \kappa_f(s)) \in \mathbb{X}_f$, i.e. the set $\mathbb{X}_f$ is control invariant, i.e. the set is invariant under the control law $a = \kappa_f(s)$.*

Assumption 11 is a standard assumption used with the purpose of analyzing the stability of the resulting closed-loop system.

**Theorem 6.** *Let Assumptions 6 - 11 hold. Then the steady state solution $s_e$ is an asymptotically stable equilibrium point of the system (5.1) using input $a$ where $a = u_0^\star$ and $u_0^\star$ is the first element in the optimal solution to* (5.12).

*Proof.* Note that the term $-\lambda(s)$ does not affect the optimal solution in (5.12), but shapes the action-value and value function. The rest of the proof is a standard result and can be found in e.g. [33]. □

### 5.2.3 Parameterized tracking MPC

We propose to use a finite-horizon MPC problem to approximate the value function (5.5), where the cost function is parameterized with parameters $\theta$, according to

$$V_\theta(s) = \min_{x,u} -\lambda_\theta(s) + T_\theta(x_N) + \sum_{j=0}^{N-1} \hat{\ell}_\theta(x_j, u_j) \tag{5.14a}$$

$$\text{s.t.} \; \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{5.14b}$$

$$x_{j+1} = \tilde{f}(x_j, u_j), \tag{5.14c}$$

$$h(x_j, u_j) \leq 0, \tag{5.14d}$$

$$x_N \in \mathbb{X}_f, \tag{5.14e}$$

where $\tilde{f}(x_j, u_j)$ is a potentially inaccurate prediction model. The resulting policy is given by the first element in the input sequence

$$\pi_\theta(s) = \bar{u}_0^\star(s, \theta), \tag{5.15}$$

where $\bar{u}^\star(s, \theta)$ is the optimal solution to (5.14). Using the MPC scheme as a function approximator, entails using RL to update the parameters $\theta$ in order to shape the value function estimate (5.14) and improve the policy (5.15). For stable economic problems, using rich parametrizations for the cost function enables the MPC scheme to deliver the optimal policy even with an inaccurate prediction model. This is stated in Theorem 1 in [21]. The following assumption is needed to ensure the nominal stability of the closed-loop system.

**Assumption 12.** *The stage cost $\hat{\ell}_\theta(s,a)$ satisfies*

$$\rho(||s-s_e||) \leq \hat{\ell}_\theta(s,a), \quad \hat{\ell}_\theta(s_e,a_e) = 0, \tag{5.16}$$

*where the optimal steady state pair $(s_e, a_e)$ may be part of the parametrization $\theta$.*

Because the MPC scheme in (5.14) may use an inaccurate prediction model, we propose to parameterize the steady state. This is not a new idea, but closely resembles an approach the real-time optimization (RTO) community refers to as modifier adaptation, see e.g. [105]. Our case differs in that we use RL to adjust the modifiers, or as we call them, parameters.

We note that as long as the MPC scheme is using an inaccurate prediction model $\tilde{f}(s,a)$, we can only guarantee nominal stability of the resulting closed-loop system, i.e. stability with respect to the MPC model. In order to guarantee the stability of the true system, we would have to apply robust techniques. Robust techniques for MPC with RL, are treated in [97]. The authors in [97] describe a robust technique for MPC that uses a nominal prediction model. A tube-based approach considers the system stochasticity and the model uncertainties and is used to perform a suitable tightening of the constraints. Because we are considering economic MPC problems, re-cast as tracking MPC schemes, the techniques from [97] directly extend to the proposed parameterized MPC scheme. For the sake of brevity, we have not treated robust techniques further in this chapter.

**Remark 10.** *The easiest way to ensure (nominal) stability of (5.14) is to use the so-called zero terminal equality i.e. $\mathbb{X}_f = \{s_e\}$, for which the terminal cost can be omitted. However, this is very restrictive and may yield feasibility issues. The terminal constraint may also be defined using an inequality constraint defined by a terminal region. For the standard quadratic stage and terminal cost, the terminal region can be approximated. For more generic stage and terminal cost approximations, the terminal region may be hard to find. However, in practice, we may use a general positive definite terminal cost, or select $N$ "large enough" and ensure stability without terminal constraint and terminal cost [38].*

**Proposition 1.** *The parameterized MPC scheme in (5.14), with a stage cost satisfying Assumption 12, using either a stabilizing terminal cost, i.e. satisfying Assumption 11, with terminal constraints or a general positive definite terminal cost and a long enough horizon $N$, will be stabilizing for any parameterization $\theta$.*

*Proof.* This is a standard result, and proofs are given in e.g. [33] and [38]. □

## 5.3 Convex cost parametrizations

For proving nominal closed-loop stability of the MPC scheme, the stage cost must be lower-bounded by a $\mathcal{K}_\infty$-function. A generic cost function lower-bounded by a $\mathcal{K}_\infty$-function is illustrated in Figure 5.1.

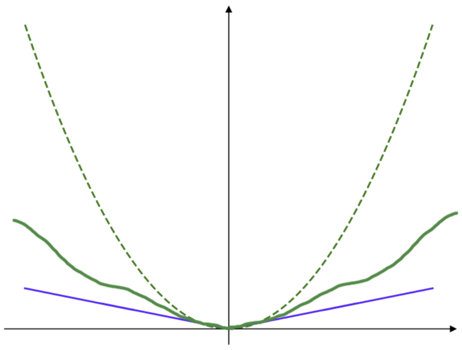


**Figure 5.1:** Generic cost function (green) lower-bounded by a $\mathcal{K}_\infty$-function (blue), with a quadratic approximation (dashed green).

The $\mathcal{K}_\infty$ lower bound in principle entails no restrictions regarding the convexity of the cost function. On the other hand, convexity may be selected as a tool to show that the same lower bound holds. The first reason for selecting convexity is that we are able to describe and therefore parameterize generic convex functions. Second, it is easier to handle convex functions in the MPC scheme. Moreover, a strictly convex function will satisfy the $\mathcal{K}_\infty$ lower bound by enforcing its global minimum to be zero at zero and the function to be radially unbounded.

For ENMPC schemes that are locally stabilizing, it can be shown that the modified stage cost obtained using dissipativity theory, is locally quadratic [104]. The quadratic approximation of the stage cost is also illustrated in Figure 5.1. In fact, the local quadratic approximation of the cost can be computed by solving a semi-definite program (SDP) [104]. As convex functions also describe quadratic functions, we can establish that the choice of a convex stage cost is at least valid locally.

### 5.3.1 Stage cost

In order to satisfy strict dissipativity as stated in Assumption 8, the approximated stage cost must satisfy the lower bound from Assumption 12 and continuity from Assumption 7. Without loss of generality, we let $(s_a, a_e) = (0, 0)$ so that $\bar{\ell}(0, 0) = 0$. The following lemma lists the function requirements for the stage cost.

**Lemma 3.** *Let $p(s, a) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be a strictly convex function. If the minimum of the function is $p(0, 0) = 0$, and $p(s, a)$ is radially unbounded with respect to s, i.e. $p(s, a) \to \infty$ as $s \to \infty$, then $\rho(\|s\|) \leq p(s, a)$ for some $\rho \in \mathcal{K}_\infty$.*

*Proof.* If the function is strictly convex, it will only have one global minimum. If this is at $p(0, 0) = 0$, it means that $\forall(s, a) \neq (0, 0)$ $p(s, a) > 0$, so that $p(s, a) \geq \beta(||s||)$, where $\beta \in \mathcal{K}$. Because $p(s, a)$ is radially unbounded with respect to $s$, we can also state that $p(s, a) \geq \rho(||s||)$, where $\rho \in \mathcal{K}_\infty$. □

### 5.3.2 Terminal cost

The terminal cost must satisfy continuity from Assumption 10 and Assumption 11. The latter assumption may be relaxed, as outlined in Remark 10.

### 5.3.3 Storage function

The storage function must satisfy continuity from Assumption 9, and zero at the optimal state, $\lambda(s_e) = 0$, as a result of (5.9).

## 5.4 NNs for cost modification

NNs are known to be universal function approximators. Consequently, a multilayered NN can represent any continuous function under mild assumptions, see e.g. [45]. More uniquely, NNs have successfully been applied to high-dimensional problems, for which traditional approximation methods tend to perform poorly. We propose to combine NNs with quadratic functions to parameterize the cost. The quadratic function is a good initial guess as we know it is locally valid and we know how to compute it [104]. NNs are then added as an attempt to capture everything beyond the locally valid quadratic function.

In this section, we will describe both regular NNs as well as convex NNs, and how these can be combined with quadratic functions to provide nominal stability of the MPC scheme by construction.

### 5.4.1 Regular neural networks

A (not necessarily convex) feedforward neural network (FNN) with $F$ layers, for which $i = 0, \dots, F-1$, can be formulated as

$$z_{i+1} = \sigma_i(W_i z_i + b_i), \quad v(s) = z_F, \tag{5.17}$$

where $z_0 = s$ is the network input, $z_i \in \mathbb{R}^{q_i \times 1}$ denotes the hidden state of layer $i$, $z_{i+1} \in \mathbb{R}^{q_{i+1} \times 1}$ denotes the hidden state of the next layer so that $W_i$ is a matrix of size $\mathbb{R}^{q_{i+1} \times q_i}$ containing the weights of layer $i$ and $b_i \in \mathbb{R}^{q_{i+1} \times 1}$ are bias terms. The nonlinear activation function used in layer $i$ is denoted by $\sigma_i$ and operates element-wise.

An FNN will be used to modify the parametrization of the storage function. From [104] we know that locally a quadratic storage function is sufficient to show strict dissipativity. We therefore combine the quadratic function with an FNN, according to

$$\lambda_\theta(s) = v_\theta(s) + \theta_{\lambda_0} + (s - \theta_{s_e})^\top D(\theta)(s - \theta_{s_e}), \tag{5.18}$$

where $v_\theta(s)$ is the FNN as defined in (5.17), $\theta_{s_e}$ is the parameterized steady state, $\theta_{\lambda_0}$ is a parameter that will be tuned so that $\lambda_\theta(\theta_{s_e}) = 0$ and $D(\theta)$ is a matrix with entries from the parameter vector $\theta$. All NN weights $W_{0:F-1}$ and bias terms $b_{0:F-1}$ are part of the parameter vector $\theta$.

The terminal cost is typically modeled with a quadratic function, and here combined with an FNN

$$T_\theta(s) = v_\theta(s)^2 + \theta_{T_0} + (s - \theta_{s_e})^\top (B(\theta)^\top B(\theta) + \epsilon I)(s - \theta_{s_e}), \tag{5.19}$$

where $B(\theta)$ is a parameter matrix, and $\theta_{T_0}$ is tuned to shift $T_\theta(\theta_{s_e})$ to zero. We use $B(\theta)^\top B(\theta) + \epsilon I$, where $\epsilon$ is a small positive constant, to ensure that the quadratic term is positive definite. For this reason we also square the output from the FNN.

**Remark 11.** *In (5.19) the terminal cost is modelled with an FNN that does not preserve convexity. To ensure nominal stability the terminal cost should be at least positive definite. However, for optimization reasons, it may be beneficial to model also the terminal cost using a convex NN, as detailed next.*

### 5.4.2 Convex neural networks

In order to build stable MPC schemes, the parameterized stage cost must satisfy Assumption 12. Making general FNNs respect the lower bound, would entail constraining the majority of the network's weights, giving an in practice intractable optimization problem for most applications. Instead, we select convex NNs to

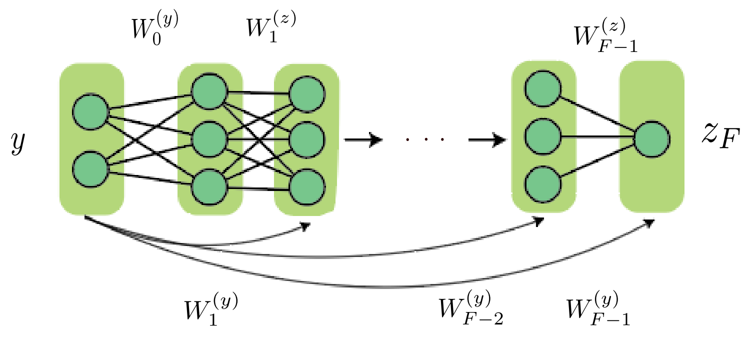


**Figure 5.2:** Input convex neural network.

parameterize the stage cost. In addition to the stability argument, a convex cost function is expected to alleviate difficulties when using sensitivity-based solvers. This section outlines how convex NNs can be adapted so that Assumption 12 is satisfied.

In recent years, several convex NN architectures have been developed. Using convex NNs as cost modifications in MPC has, to the best of the authors' knowledge, not been done before. We consider a class of fully input convex neural networks (ICNNs) first proposed in [106]. It was proven in [107] that ICNNs are universal approximators of convex Lipschitz functions. Alternative convex NN architectures exist, as described in e.g. [108], that are richer function approximators than ICNNs, but usually require a larger number of parameters. The specific type of ICNN is selected because it offers a simpler parametrization and training process, and requires fewer parameters.

Let $y = \{s, a\}$. An ICNN with $F$ layers as described in [106], for which $i = 0, \dots, F-1$, can be formulated as

$$z_{i+1} = \sigma_i(W_i^{(z)} z_i + W_i^{(y)} y + b_i), \quad g(y) = z_F, \tag{5.20}$$

where $g(y)$ denotes the output of the ICNN. The ICNN is similar to the FNN in (5.17), with the exception of the additional input weights $W_i^{(y)} \in \mathbb{R}^{q_{i+1} \times (n+m)}$ and the input to the ICNN $y \in \mathbb{R}^{(n+m)\times 1}$, that here enters every hidden layer and the output layer. Also, the ICNN requires the activation $\sigma_i$ to be a convex and non-decreasing nonlinear activation function. An example of this is given in Section 5.4.3. For the input layer, we have that $W_0^{(z)} = 0$ and $z_0 = 0$. The network is visualized in Figure 5.2.

**Proposition 2.** *The function $g$ is convex in $y$ provided that all terms in $W_{1:F-1}^{(z)}$ are non-negative, and all functions $\sigma_i$ are convex and non-decreasing.*

*Proof.* This is straightforward to prove, using the fact that non-negative sums of convex functions are also convex and that the composition of a convex and convex non-decreasing function is convex [109]. □

By limiting all weights $W^{(z)}_{1:F-1}$ to be non-negative, the function $g(y)$ will be a convex function with respect to its input. We model the stage cost as

$$\hat{\ell}_\theta(y) = g_\theta(y) + \theta_{\ell_0} + (y - \theta_e)^\top (M(\theta)^\top M(\theta) + \epsilon I)(y - \theta_e), \tag{5.21}$$

where $\theta_{\ell_0}$ is a dedicated parameter used to shift $\hat{\ell}_\theta(\theta_e)$ to zero and $\theta_e$ contains the parameterized steady state, i.e. $\theta_e = (\theta_{s_e}, \theta_{a_e})$. Moreover, we use $M(\theta)^\top M(\theta) + \epsilon I$ to ensure that the quadratic term is positive definite. The quadratic term will ensure that for a (not necessarily strictly) convex function, $\theta_e$ will be the only global minimum. The quadratic term will also ensure that the resulting function is radially unbounded with respect to $s$. In addition to the function being convex, we also need the global minimum of the function to be zero at steady state, i.e.

$$\hat{\ell}_\theta(\theta_e) = 0, \quad \nabla_y \hat{\ell}_\theta(\theta_e) = 0. \tag{5.22}$$

This is satisfied by construction as the parameters are updated. Next, we will formally establish that the stage cost is lower-bounded by a $\mathcal{K}_\infty$-function. As a result, the parameterized MPC scheme ensures nominal stability by construction.

**Theorem 7.** *Let (5.22) hold for $\hat{\ell}_\theta(s, a)$ modelled as in (5.21). Then the parameterized stage cost (5.21) satisfies*

$$\rho(||s - \theta_{s_e}||) \leq \hat{\ell}_\theta(s, a), \quad \hat{\ell}_\theta(\theta_{s_e}, \theta_{a_e}) = 0. \tag{5.23}$$

*Proof.* The ICNN term, $g_\theta(y) + \theta_{l_0}$, and the quadratic term, $(y - \theta_e)^\top (M(\theta)^\top M(\theta) + \epsilon I))(y - \theta_e)$, are both convex functions. The addition of the quadratic term ensures that the stage cost becomes strictly convex, so that it will have at most one global minimum. The quadratic term also ensures that the cost will be radially unbounded. Because (5.22) holds, the only global minimum will be at $(\theta_{s_e}, \theta_{a_e})$, and consequently the stage cost satisfies Lemma 3, and $\hat{\ell}_\theta(\theta_{s_e,}\theta_{a_e}) = 0$. □

## 5.4.3 Choice of activation functions

Convexity of the ICNN is dictated by Proposition 2, which requires convex and non-decreasing activation functions. By selecting a smoothed version of the rectified linear unit (ReLU), such as the softplus function, the specified convexity properties are ensured. The fact that this function is also continuously differentiable, may ease optimization of the MPC problem. The softplus function is given by

$$\sigma(x) = \ln(1 + \exp(x)). \tag{5.24}$$

For the cost terms modelled by the FNNs, we have no limitations on the choice of activation functions, except the requirement on continuity. As stated in Section 5.2.1, all cost terms should be continuous functions. For the NN-based cost terms this is dictated by the choice of activation function, and this is satisfied for all the most popularly used activation functions.

## 5.5 RL for parameter updates

Finding a storage function that allows us to recast ENMPC as a stable tracking MPC scheme, i.e. that satisfies the dissipation inequality, is generally difficult. Techniques for finding the storage function include e.g. sum-of-squares programming for polynomial dynamics and stage costs [110], or via techniques borrowed from RL as in [21]. The latter approach allows one to build ENMPC schemes that are optimal and whose stability is established by construction rather than by verification. This framework also introduces the additional flexibility to tackle non-dissipative problems. Indeed, for non-dissipative problems, the proposed framework can be used to find a controller that as closely as possible resembles the optimal unstable policy for the problem at hand. In this section, we will consider how RL can be used to perform parameter updates of parameterized MPC schemes such that the requirements outlined in Section 5.4, are ensured.

In the following, we propose to use Q-learning to update the parameters of the parameters in the MPC scheme, as outlined in Section 2.3.1, modified to an undiscounted setting by specifying $\gamma = 1$. However, there are no guarantees that Q-learning will converge to the optimal policy, and for certain shapes of Q-functions, it can be challenging to capture the optimal policy, even with an almost correct Q-function estimate. In this scenario, policy-based methods such as deterministic policy gradient methods may be more suited, as outlined in Section 2.3.2.

### 5.5.1 Constrained RL steps

In order to ensure the convexity of the ICNN, we need to perform constrained RL steps so that selected weights in the network stay non-negative. This applies to the hidden state weights $W^{(z)}_{1:F-1}$ in (5.20). We also use the constrained RL update to ensure that the NNs used in the stage cost, terminal cost, and storage function, are zero at steady state and to ensure that the global minimum of the stage cost is zero at steady state i.e. that (5.22) holds.

Let $d$ denote the proposed step by RL at time step $k$, using e.g. Q-learning or a deterministic policy gradient method. We can then define the following optimization problem to constrain $d$ so that the parameter update respects the constraints as detailed above

$$\begin{aligned}
&\min_{\Delta\theta} \frac{1}{2} \|\Delta\theta\|^2 - d^\top \Delta\theta && (5.25a)\\
&\text{s.t.} \quad w_i + \Delta\theta_i \geq 0 \quad \text{for} \quad i = 1, ..., r, && (5.25b)\\
&\qquad \hat{\ell}_{\theta_{k+1}}(\theta_{k+1,e}) = 0, && (5.25c)\\
&\qquad \nabla_y \hat{\ell}_{\theta_{k+1}}(\theta_{k+1,e}) = 0, && (5.25d)\\
&\qquad \lambda_{\theta_{k+1}}(\theta_{k+1,s_e}) = 0, && (5.25e)\\
&\qquad T_{\theta_{k+1}}(\theta_{k+1,s_e}) = 0, && (5.25f)
\end{aligned}$$

where $\Delta\theta = \theta_{k+1} - \theta_k$ is the constrained update of the entire parameter vector and $w_i$ are the constrained elements in the ICNN weight matrices $W^{(z)}_{1:F-1}$. The $r$ constrained weights $w_i$ are also part of the parameter vector $\theta$, and we have that $w_i = \theta_{i,k}$. It can be shown that if there exists a constrained parameter update $\Delta\theta$ at $k = 0$ by solving the optimization problem (5.25), such that (5.25b)-(5.25f) are satisfied, then there must exist a feasible solution $\Delta\theta$ $\forall k > 0$.

**Remark 12.** *The constraints in (5.25) are not necessarily ensured at $k = 0$ for the initial values of the parameters $\theta$, even for NNs that are pre-trained to quadratic functions. To guarantee that these constraints hold at $k = 0$, the constraints can be enforced either during or after pre-training.*

**Lemma 4.** *If learning converges, the constrained parameter update found from solving (5.25) will yield the true optimal parameters $\theta$, i.e. the parameter values that minimize the function with gradient step d.*

*Proof.* Let $\Psi(\theta)$ denote the function that RL is trying to minimize, i.e. $d = -\alpha\nabla\Psi(\theta)$. We formulate the original optimization problem as

$$\begin{aligned}
&\min_{\theta} \ \alpha\Psi(\theta) && (5.26a)\\
&\text{s.t.} \ Z(\theta) = 0, && (5.26b)\\
&\qquad \Gamma(\theta) \leq 0, && (5.26c)
\end{aligned}$$

where $Z$ and $\Gamma$ are matrices that gather the equality and inequality constraints in (5.25) respectively. For (5.26) the stationarity of the KKT conditions is given as

$$\alpha\nabla\Psi(\theta) + \phi^\top \nabla Z(\theta) + \xi^\top \nabla\Gamma(\theta) = 0, \qquad (5.27)$$

where $\phi$ and $\xi$ are the multipliers associated with equality and inequality constraints, respectively. The stationarity of the KKT conditions for (5.25) are

$$\Delta\theta - d + \phi^\top \nabla Z(\theta + \Delta\theta) + \xi^\top \nabla \Gamma(\theta + \Delta\theta) = 0. \tag{5.28}$$

As learning converges, $\Delta\theta \approx 0$. Using this, and the fact that $d = -\alpha\nabla\Psi(\theta)$, we see that we obtain the same expression for stationarity of the KKT conditions. Note that one can also show that the primal/dual feasibility conditions and the complementary slackness condition are the same. The two optimization problems therefore share the same optimal values of $\theta$. $\square$

## 5.6 Combining Q-learning and policy gradient methods

Policy-based methods have several advantages over value-based RL methods. First and most important, these methods are more reliable when it comes to improving the policy, as they are designed based on optimality of the closed-loop policy. Second, certain types of policy-based methods are also known to be more sample-efficient than Q-learning [39]. However, there may be parameters that the MPC policy gradient will be insensitive to. Mathematically this entails that certain parameters lie in the null space of the policy gradient.

This is especially relevant for rich parametrizations, as they contain more parameters. Although certain parameters may not influence the optimal policy, we may still want to update them, in order to e.g. capture the correct shape of the value and action-value function. In this context we propose to embed Q-learning, as a measure to handle the parameters that the MPC policy may not be sensitive to. As the Q-function and the policy are jointly unique functions, the parameters should affect at least one of these functions.

### 5.6.1 Null space method

For an ENMPC problem recast as a tracking MPC, policy gradient methods will not be sufficient for tuning the cost parametrization, as the MPC policy is insensitive to the storage function. Hence, we will use a policy gradient method and combine it with Q-learning using a null space method. More specifically we aim at using a policy gradient method to update parts of the parameters in order to converge to the correct policy and perform Q-learning steps in the null space of the policy gradient to shape the action-value function with the remaining parameters. For a parametrization that is rich enough, the correct action-value function should be captured without conflicting with the policy approximation. Whereas the use of both Q-learning and policy gradient methods for adjusting a parameterized MPC scheme is well established, we now introduce a new method for combining RL algorithms.

In order to formulate the null space of the policy gradient update, we consider an approximation of the Hessian of the policy gradient, given by the Fisher information matrix [111]:

$$\nabla_\theta^2 J(\pi_\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \pi_\theta(s) \nabla_\theta \pi_\theta(s)^\top]. \tag{5.29}$$

Alternatively, a more accurate approximation of the Hessian can be found in [112]. We then define the null space of $\nabla_\theta^2 J(\pi_\theta)$ as gathered by the matrix $\mathcal{N}$, such that $\nabla_\theta^2 J(\pi_\theta)\mathcal{N} = 0$. The parameter update resulting from Q-learning (2.20) is then projected to the null space of the policy gradient according to

$$\Delta\theta_Q^{\mathcal{N}} = \mathcal{N}(\mathcal{N}^\top \mathcal{N})^\dagger \mathcal{N}^\top \Delta\theta_Q, \tag{5.30}$$

where $\cdot^\dagger$ denotes the Moore-Penrose pseudo-inverse. The full parameter update resulting from combining policy gradient and Q-learning is then given by

$$\Delta\theta = \Delta\theta_J + \Delta\theta_Q^{\mathcal{N}}, \tag{5.31}$$

with $\Delta\theta_J$ as defined in (2.22).

## 5.7 Numerical examples

In this section, we propose two numerical examples to illustrate the proposed method. The first example is a seemingly simple case of an economic LQR, i.e. an LQR with weighting matrices that are not positive definite. The example becomes challenging because of the shape of the action-value function that calls for a combination of RL methods in order to capture both the correct policy and value function. The second simulation example is a chemical reactor with nonlinear dynamics and an economic cost function. The ENMPC scheme is recast as a tracking MPC scheme, using a parameterized cost and storage function. We combine the convex NN-based cost modifications and quadratic functions for all cost terms, ensuring nominal stability of the MPC, and benchmark its performance against the standard quadratic cost parametrization.

### 5.7.1 Economic LQR

We consider an economic LQR for a system with dynamics

$$s_{k+1} = 0.1 s_k + a_k, \tag{5.32}$$

and stage cost

$$L(s,a) = -s^2 + 10a^2. \tag{5.33}$$

For the sake of satisfying Assumption 6, we introduce the following artificial constraints

$$-100 \le a \le 100, \quad -100 \le s \le 100. \tag{5.34}$$

**Table 5.1:** NNs for LQR example

| Cost term | NN class | Num. hidden layers | Num. neurons | Pre-trained |
|---|---|---|---|---|
| $\hat{\ell}_\theta(s,a)$ | ICNN | 2 | 16, 16 | ✓ |
| $\hat{\ell}_\theta(s,a)$ | FNN | 2 | 16, 16 | ✓ |
| $\lambda_\theta(s)$ | FNN | 2 | 16, 16 | ✓ |
| $T_\theta(s)$ | FNN | 2 | 16, 16 | ✗ |

For the set of states that never activate the constraints, we can solve the Riccati equation for the discrete system, and obtain the optimal value function and policy. For the dynamics (5.32) and stage cost (5.33) in the unconstrained case, the optimal value function and policy is

$$V^\star(s) = -1.0113s^2,\ \pi^\star(s) = 0.0113s. \tag{5.35}$$

In the first set of simulations, we will make a comparison of the ICNN and the FNN. For this purpose, we will assume that both the true dynamics and the optimal steady state pair are available. We formulate the following finite-horizon linear MPC scheme

$$\begin{aligned}
&\min_{x,u} -\lambda_\theta(s) + T_\theta(x_N) + \sum_{j=0}^{N-1} \hat{\ell}_\theta(x_j,u_j) && \text{(5.36a)}\\
&\text{s.t.}\ \ \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, && \text{(5.36b)}\\
&\qquad x_{j+1} = 0.1x_j + u_j \quad , && \text{(5.36c)}\\
&\qquad -100 \le x_j \le 100, && \text{(5.36d)}\\
&\qquad -100 \le u_j \le 100, && \text{(5.36e)}
\end{aligned}$$

with prediction horizon $N = 10$. For this example, we used NNs to model all cost terms in (5.36). In order to get suitable initial values of the weights, we pre-trained the NNs to quadratic functions. This was done using Keras in Python [113]. The architecture used to parameterize each cost term, is reproduced in Table 5.1. We stress that this simulation example is mainly providing a proof of concept, and therefore that the choices related to the architectures of the NNs have not been optimized. System (5.32) is simulated from random initial conditions on the interval $[-1, 1]$, for episodes of length 10.

For this example, regular Q-learning manages to capture the Q-function fairly accurately but struggles to capture the optimal policy. This is likely explained by the shape of the Q-function, which turns out to be fairly insensitive to the policy, causing small errors in the Q-function to give large errors in the resulting policy. This clearly illustrates a known weakness in Q-learning, and we therefore resort to a combination of Q-learning and deterministic policy gradient methods using a null space method as described in Section 5.6. The gradient of $Q$ w.r.t. the action needed to formulate the deterministic policy gradient can be computed from data using a range of algorithms. For convenience, we use the true Q-function to formulate the gradient needed in (2.23), that is

$$Q_{\pi_\theta}(s,a) = -x^2 + 10a^2 + (0.1x + a)^2 P, \tag{5.37}$$

where $P$ is the solution to the Riccati equation. The derivative with respect to the action is then

$$\nabla_a Q_{\pi_\theta}(s,a) = 0.2Px + 2(10 + P)a. \tag{5.38}$$

Furthermore, we use gradient descent with learning rate $\alpha = 0.02$ for both the Q-learning and policy gradient update. A total of 2500 episodes were simulated in order to update the parameters, yielding a total of $2.5 \times 10^4$ learning samples. For the same hyperparameter values, we tested learning using both an ICNN and an FNN to model the stage cost. Given the learned storage function, we are able to obtain the rotated cost given by (5.9). This is plotted for the ICNN and the FNN in Figure 5.3. Because the curvature is much larger in the action dimension, we have adjusted the axes in order to highlight the curvature in $s$-direction.

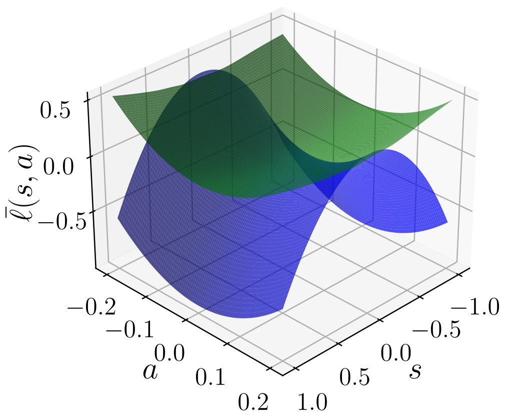


**Figure 5.3:** Resulting rotated stage cost (5.9) when using an ICNN (green) and an FNN (blue) to model the stage cost respectively.

We see that with an FNN to model the stage cost, learning may fail to capture a storage function such that we obtain a rotated stage cost lower-bounded by a $\mathcal{K}_\infty$-function. We note that for some of the simulations, using FNNs would also produce rotated stage costs that were lower-bounded by a $\mathcal{K}_\infty$-function. However, we saw that by using FNNs, we were not guaranteed to learn a stage cost that satisfied the desired lower bound.

For the second set of simulations, we will demonstrate that MPC with the proposed cost parametrization, successfully learns the optimal policy. For this demonstration, we assume that both the true dynamics and the optimal steady state are unknown. We used the following inaccurate prediction model in the MPC scheme:

$$x_{j+1} = 0.098x_j + 1.02u_j \tag{5.39}$$

We parameterized the steady state parameters and wrongly initialized the parameters with $\theta_e = (0.3, 0.3)$. We ran a total of 2500 episodes of length 10, resulting in

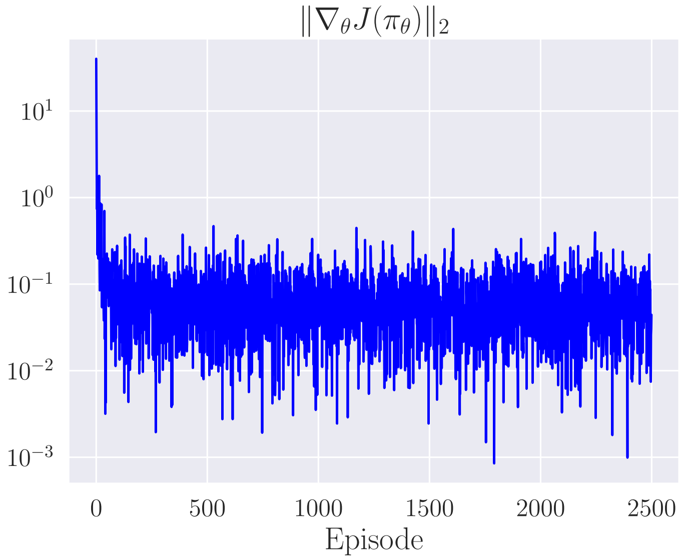


**Figure 5.4:** Policy gradient over episodes.

$2.5 \times 10^4$ learning samples, using $\alpha = 0.01$. In Figure 5.4 we have plotted the evolution of the policy gradient over episodes. We note that this is noisy due to random choices of initial conditions.

In Figure 5.5 we have plotted the approximation of the value function and the policy, using the final updated values of the parameters. We see that the value function is

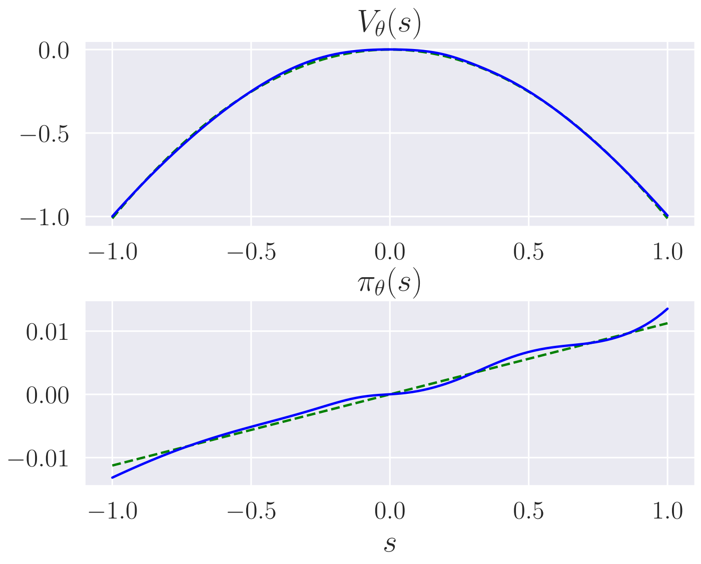


**Figure 5.5:** Approximated value function and policy using only NNs to parameterize all cost terms (blue). Optimal value function and policy for the unconstrained case (green, dashed).

captured accurately, whereas the policy approximation has some inaccuracies. This is likely improved by increasing the number of learning samples.

In Figure 5.6 we have plotted the evolution of the steady state parameters over episodes. We see that the steady-state parameters converge very close to the optimal steady state, namely $(\theta_{s_e}, \theta_{a_e}) = (0, 0)$.

### 5.7.2 ENMPC: Chemical reactor

The next simulation example has nonlinear dynamics and an economic stage cost, and we expect a quadratic cost function to be valid only locally. It is therefore suitable for testing the addition of NNs to a quadratic cost parametrization. We consider a CSTR, with an economic cost as described in [114]. The CSTR describes a non-isothermal reactor, where an exothermic reaction, converting reactant A to product B, takes place. The dynamics are given as

$$\dot{C}_A = \frac{F}{V_R}(C_{A0} - C_A) - k_0 e^{-E/RT} C_A^2 \tag{5.40a}$$

$$\dot{T} = \frac{F}{V_R}(T_0 - T) - \frac{\Delta H k_0}{\rho_R C_p} e^{-E/RT} C_A^2 + \frac{q}{\rho_R C_p V_R}, \tag{5.40b}$$

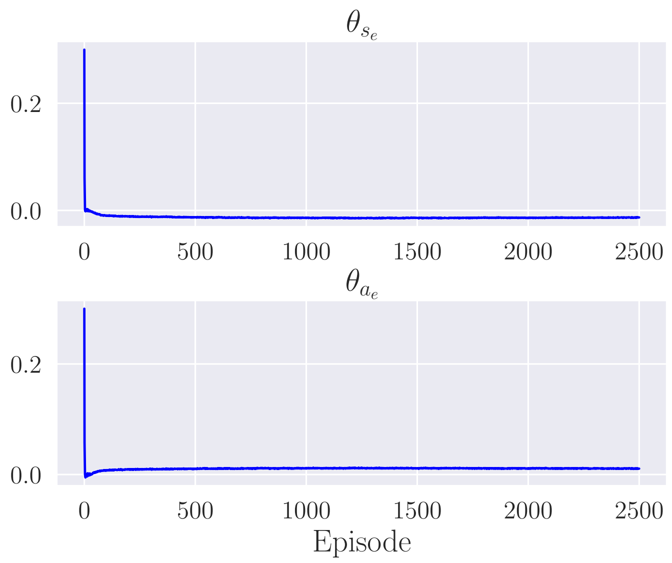


**Figure 5.6:** Evolution of steady state parameters.

where $T$ is the temperature in the reactor, $C_A$ is the concentration of the reactant A, $F$ is the flow rate and $q$ is the heat rate. The same quantities constitute the states and inputs, i.e. $s = [C_A, T]$ and $a = [F, q]$ respectively. The additional parameters are listed in Table 5.2. The inputs are constrained according to

$$[0, -2 \times 10^5] \leq a \leq [10, 2 \times 10^5]. \tag{5.41}$$

The economic cost is

$$L = -\omega F(C_{A0} - C_A) + \beta q, \tag{5.42}$$

where $\omega = 1.7 \times 10^4$ and $\beta = 1$ so that the production rate and energy consumption will be balanced. To get the dynamics on the form of (5.1), the equations in (5.40) were discretized using the Euler method with a step size of 0.02 hours. According to (5.3), the optimal steady state of the system is

$$s_e = [0.7572, 497.71], \quad a_e = [10, 1.38557 \times 10^5]. \tag{5.43}$$

**Table 5.2:** CSTR parameter definitions and values

| Symbol | Description | Value |
|---|---|---|
| $C_{A0}$ | Feed concentration of A | 3.5 $\mathrm{kmol/m^3}$ |
| $T_0$ | Feedstock temperature | 300 K |
| $V_R$ | Reactor fluid volume | 1.0 $\mathrm{m^3}$ |
| E | Activation energy | $5.04 \times 10^4$ kJ/kmol |
| $k_0$ | Pre-exponential rate factor | $8.46 \times 10^6$ $\mathrm{m^3/kmolh}$ |
| $\Delta H$ | reaction enthalpy change | $-1.16 \times 10^4$ kJ/kmol |
| $C_p$ | Heat capacity | 0.231 kJ/kgK |
| $\rho_R$ | Density | 1000 $\mathrm{kg/m^3}$ |
| R | Gas constant | 8.314 kJ/kmolK |

We assume that we only have an inaccurate prediction model available, which we define by introducing the following errors in the dynamics described by (5.40):

$$\dot{C}_A = 0.85\frac{F}{V_R}(C_{A0} - 0.85C_A) - 1.2k_0 e^{-E/R0.95T}C_A^2 \tag{5.44a}$$

$$\dot{T} = 0.85\frac{F}{V_R}(T_0 - T) - \frac{1.2\Delta H k_0}{\rho_R C_p}e^{-E/R0.95T}C_A^2 + \frac{0.8q}{\rho_R C_p V_R}. \tag{5.44b}$$

The MPC scheme is formulated with the inaccurate prediction model and a parameterized cost according to

$$\min_{x,u} -\lambda_\theta(s) + T_\theta(x_N) + \sum_{j=0}^{N-1} \hat{\ell}_\theta(x_j, u_j) \tag{5.45a}$$

$$\text{s.t. } \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{5.45b}$$

$$x_{j+1} = \tilde{f}(x_j, u_j), \tag{5.45c}$$

$$h(u_j) \leq 0, \tag{5.45d}$$

with a prediction horizon of $N = 10$, where $\tilde{f}(x_j, u_j)$ is the discretized version of (5.44). The stage and terminal cost were modelled with quadratic functions and convex cost modifications as in (5.21), whereas for the storage function, we used a quadratic function and an FNN as in (5.18). Because the true model is unknown, we also parameterized the steady state. The steady state parameters were initialized with values found by evaluating (5.3) for the inaccurate prediction model described by (5.44). The architecture for each NN is specified in Table 5.3. The quadratic terms were initialized with $M(\theta) = I_{n+m}$, $B(\theta) = I_n$, $D(\theta) = 10 \times I_n$, where $I$

**Table 5.3:** NNs for CSTR example

| Cost term | NN class | Num. hidden layers | Num. neurons | Pre-trained |
|---|---|---|---|---|
| $\hat{\ell}_\theta(s,a)$ | ICNN | 2 | 16, 16 | ✗ |
| $\lambda_\theta(s)$ | FNN | 2 | 16, 16 | ✗ |
| $T_\theta(s)$ | ICNN | 2 | 16, 16 | ✗ |

is the identity matrix of dimension given by the subscript. The NN weights and bias terms were initialized to small, random numbers. All parameters were updated with Q-learning. Because the combination of quadratic functions and NNs will introduce many parameters, that may also differ by orders of magnitude, we used Adam optimization for updating the parameters. Adam optimization differs from gradient descent by computing individual learning rates for each parameter. The hyperparameters typically require little tuning, and we used standard values, see [115]. Because the states and inputs span different orders of magnitude, we used input normalization to force the NN input variables into the range $[0, 1]$. Input normalization used correctly is known to reduce estimation errors as well as speed up convergence, see for instance [116].

The closed-loop system was first simulated with quadratic cost terms, i.e. we parameterize the cost with only the quadratic terms in (5.18), (5.19) and (5.21). We simulated for 1000 episodes of length 60, with a learning rate of $\alpha = 1 \times 10^{-3}$, until performance converged. The evolution of the quadratic parameters over the episodes is plotted in Figure 5.7. The result from the first 1000 episodes was used as a benchmark for the rich cost parameterization.

The rich cost parametrization was obtained by adding NNs to the already trained quadratic cost terms after 1000 episodes. We then continued learning for 500 more episodes of shorter length, as we were mainly hoping to improve performance in the transients. The closed-loop performance during learning is plotted for all episodes in Figure 5.8. We note that, as for the previous simulation example, this plot is noisy due to the random choice of initial conditions. The closed-loop performance initially worsens, before improving using the quadratic cost parameterization. After 1000 episodes we see that introducing the NNs creates a new peak in performance, followed by a new decrease in $J(\pi_\theta)$ i.e. an improvement in performance. We conjecture that the two peaks in performance stem from the fact that the reference state and input are parameterized, in combination with the system's states and inputs not being scaled, resulting in a less smooth learning process.

In order to further evaluate the performance of each cost parametrization, we

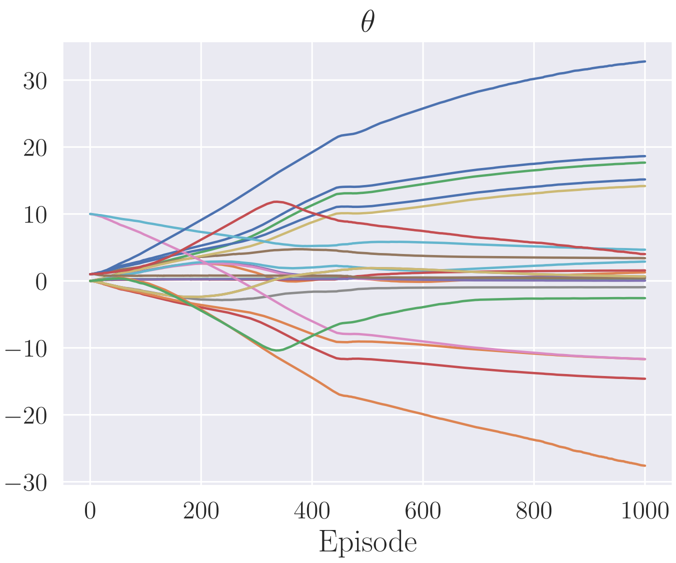


**Figure 5.7:** Evolution of the quadratic parameters during the first 1000 episodes.

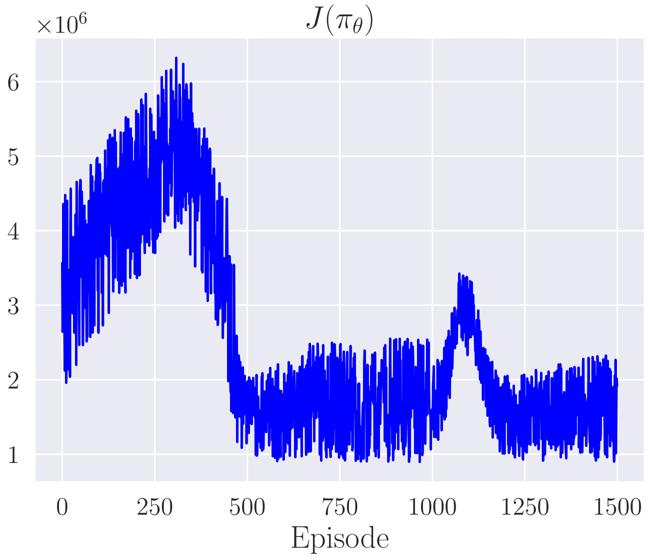


**Figure 5.8:** Closed-loop performance during learning of the quadratic parameters (0-1000 episodes), and of the quadratic and NN parameters (after 1000 episodes).

have also compared their performance in closed-loop using the final values of the updated parameters. In order to show that the learned controller is robust to

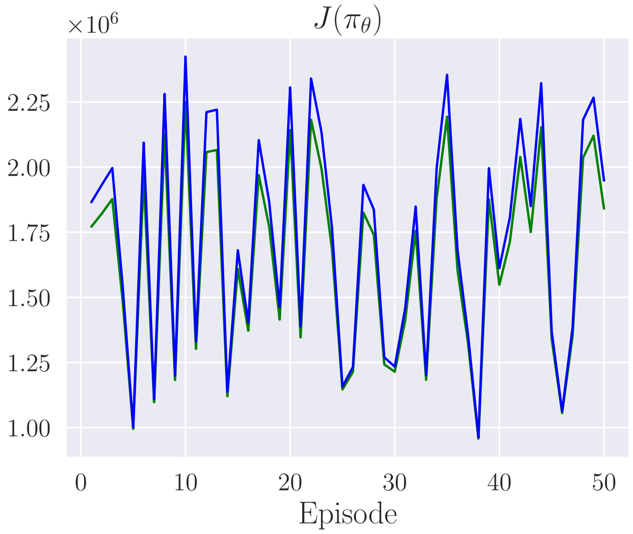


**Figure 5.9:** Closed-loop performance of controllers using the learned cost parameters of the quadratic parameterization (blue) and the combination of quadratic functions and NNs (green) in the MPC.

model error, we added parametric uncertainty in the pre-exponential rate factor $k_0$ in (5.40). For each random initial condition, we also drew a new value of $\tilde{k}_0$ where $\tilde{k}_0 \sim \mathcal{N}(k_0, \sigma_k^2)$ with $\sigma_k = 2.1156 \times 10^5$. In Figure 5.9, we have plotted the closed-loop performance of closed-loop trajectories starting from 50 randomly selected initial conditions and values of $\tilde{k}_0$. The addition of the NN-based cost modifications clearly gave a modest improvement in performance. It is possible that the convexity requirement to the NNs in this case is limiting the use of the cost modifications for this example.

In Figure 5.10 we have plotted the mean and two standard deviations of closed-loop trajectories from the same 50 simulations. We see that the addition of the NNs does not alter the closed-loop trajectories much, except noticeably for the flow rate $F$. Also, the controllers converge to slightly different steady states for the heat rate $q$. However, the shifted economic cost, $\ell(s, a)$, plotted in Figure 5.11, shows that this has a small influence on the cost. We see clearly that, as previously stated, at steady state the quadratic cost parametrization is sufficient to obtain the optimal economic cost. Although the effect of adding NN-based cost modifications on performance was limited in this case, we see that the small improvement that does occur happens in the transients.

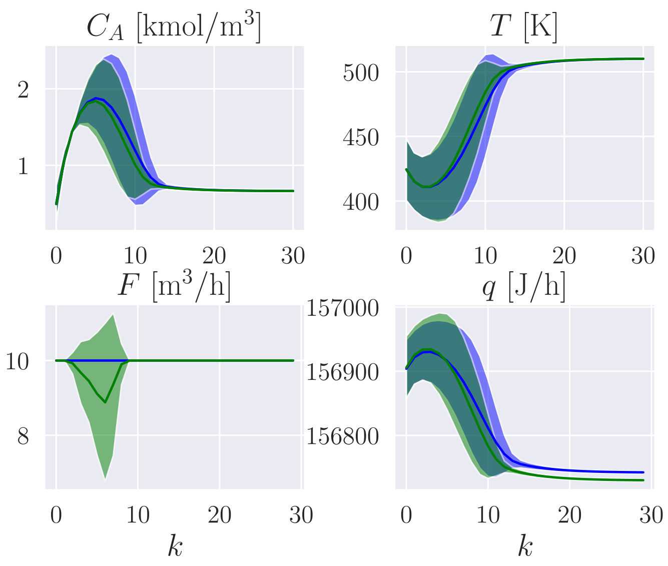


**Figure 5.10:** Closed-loop trajectories from 50 simulations using the learned cost parameters of the quadratic parameterization (blue) and the combination of quadratic functions and NNs (green) in the MPC scheme. The thick lines represent the empirical mean, and the shaded areas represent two standard deviations.

## 5.8 Conclusion

In this chapter we have considered the use of convex cost modifications, using NNs. We have applied this framework to economic ENMPC. By invoking dissipativity theory, we have recast the ENMPC as a tracking MPC scheme, with the additional storage function. Convexity properties of the learned stage cost have been leveraged in order to ensure the appropriate lower bound necessary to establish nominal closed-loop stability, as well as alleviate numerical difficulties when solving the MPC problem. We have outlined how RL can be used to adjust the parametrized cost, including the weights of the NN, enabling the tracking MPC scheme to deliver the optimal policy, even with an inaccurate prediction model. For a challenging case of economic LQR, we have demonstrated how a combination of RL methods can be used to update the parameters so that both the policy and value function is learned. For a nonlinear chemical reactor, we have benchmarked the combination of quadratic functions and NNs, against a standard quadratic parametrization. For this particular simulation example, the addition of NN-based cost modifications resulted in a small improvement in closed-loop performance.

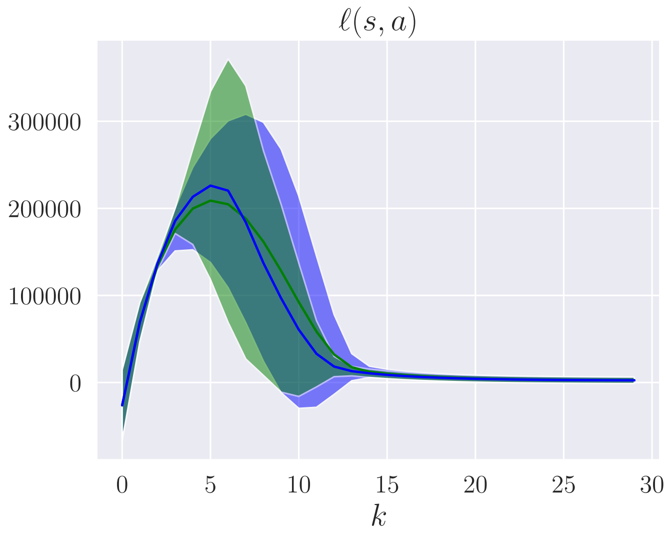


**Figure 5.11:** Economic cost in simulations using the learned cost parameters of the quadratic parameterization (blue) and the combination of quadratic functions and NNs (green) in the MPC. The thick lines represent the empirical mean, and the shaded areas represent two standard deviations.

# Chapter 6

# Combining RL methods for learning-based MPC

This chapter considers adjusting a fully parametrized MPC scheme to approximate the optimal policy for a system as accurately as possible. By adopting MPC as a function approximator in RL, the MPC parameters can be adjusted using Q-learning or policy gradient methods. However, each method has its own specific shortcomings when used alone. Indeed, Q-learning does not exploit information about the policy gradient and therefore may fail to capture the optimal policy, while policy gradient methods miss any cost function corrections not affecting the policy directly. The former is a general problem, whereas the latter is an issue when dealing with economic problems specifically. Moreover, it is notoriously difficult to perform second-order steps in the context of policy gradient methods while it is straightforward in the context of Q-learning. This calls for an organic combination of these learning algorithms, in order to fully exploit the MPC parameterization as well as speed up convergence in learning. The following chapter is based on work in [26].

## 6.1 Introduction

Updating the parameters of MPC schemes to improve the closed-loop performance has successfully been tested using Q-learning, see e.g. [97], and DPG, see e.g. [22]. While simple to use, Q-learning methods do not come with formal guarantees regarding the closed-loop optimality of the resulting policy [39]. Whereas policy gradient methods come with such (local) guarantees but do not fully exploit the MPC parameterization in learning the policy. This is crucial when using RL to verify *dissipativity* for economic problems. Dissipativity is verified by the existence of an appropriate *storage function*, which is generally difficult to find, but that can be learned using Q-learning as proposed in [103]. Using Q-learning alone, we are not guaranteed to learn the stable optimal policy. In this chapter, we detail how to combine these RL algorithms for MPC, such that their respective drawbacks are tackled.

To the best of the authors' knowledge, Q-learning and policy gradient methods have to a small extent been combined to formulate parameter updates. One exception is the work in Chapter 5, where Q-learning and DPG parameter updates were combined using a null space projection to alleviate the difficulties experienced when using the methods independently. An issue with this combination is the potential inaccuracies in the null space computation related to small, but not zero, eigenvalues. In [117], the authors propose to combine the parameter update for a regularized policy gradient technique with that of Q-learning. The authors provide empirical evidence of the combined update scheme resulting in improved data efficiency and stability. The idea of combining parameter updates from different RL methods is the same as in this chapter but the resulting combined parameter update is different. Also, the authors in [117] use an NN as a function approximator, and propose an augmentation to the standard architecture, to facilitate learning of both the policy as well as the action-value function. This is not necessary when using MPC as a function approximator.

Classical Q-learning and policy gradient methods typically use first-order methods to update the policy and value function parameters. Second-order methods tend to yield a much faster convergence. The natural policy gradient method is based on the Fisher information matrix to provide a policy Hessian approximation, and approximate second-order steps [111]. However, the convergence rate is affected by the quality of that Hessian approximation. For Q-learning on the other hand, the true Hessian is straightforward to obtain. We therefore propose to use the Q-learning Hessian to improve the second-order step of policy gradient methods.

In this chapter, the first contribution we make is a combination of RL methods using a multi-objective approach. Secondly, we propose a second-order method based on the multi-objective approach, to speed up convergence in learning. We demonstrate the performance of the combination of RL methods in simulation and compare the convergence rate with the natural policy gradient and a second-order Q-learning method.

## 6.2 Background

In this chapter, we consider problems that are described by MDPs and adopt a value function approximator using MPC as detailed in (2.30), parameterized by $\theta$. We note that for EMPC, MPC problems characterized by generic cost functions that are not necessarily positive definite, we may include a parameterized storage function in the MPC cost. For more details on this, the reader is referred to Chapter 5. Our goal is to find the parameters $\theta$ of a deterministic policy $\pi(s)$, that maps from state to action i.e. $\pi_\theta : \mathcal{S} \to \mathcal{A}$, so as to minimize the sum of discounted cost

$$J(\pi_\theta) = \mathbb{E}_{s_0 \sim p_0, s \sim p(\cdot|s,\pi_\theta(s))} \left[ \sum_{k=0}^{K} \gamma^k L(s_k, a_k) \mid a_k = \pi_\theta(s_k) \right], \tag{6.1}$$

where $p_0$ is a distribution of initial states, $p(\cdot|s, \pi_\theta(s))$ describes the transition dynamics and $\gamma \in (0, 1]$ is a discount factor used to establish the importance of future costs over immediate costs for an episode of length $K$. In order to explore, we use a stochastic policy i.e.

$$\varphi_\theta(a|s) = \pi_\theta(s) + \zeta_a, \tag{6.2}$$

with $\zeta_a \sim \mathcal{N}(0, \sigma_a^2 I_m)$, where $\sigma_a$ is the standard deviation and $I_m$ is the identity matrix. Next, we will describe two types of RL methods that can be used to adjust the parameters $\theta$.

### 6.2.1 Q-learning

We make use of the Q-function approximator as given in (2.33), and make the following assumption for the parameterization.

**Assumption 13.** *The parameterization is rich, i.e. there exists a parameter vector $\theta^\star$ such that*

$$Q_{\theta^\star}(s, a) = Q^\star(s, a). \tag{6.3}$$

**Remark 13.** *Assumption 13 is strong, although common in theoretical RL, see e.g. [21]. Making the parameterization rich, entails using universal function approximators for the cost terms and constraints in the MPC scheme. If the parameterization is not rich, RL will find the best parameters among the set of functions provided by the selected parameterization, see e.g. [46]. The need for using universal function approximators such as NNs is arguably problem dependent, but as shown in Chapter 5, simpler parameterizations of the MPC scheme, such as e.g. a quadratic cost parameterization, can be able to improve closed-loop performance considerably.*

For a rich parameterization, we can characterize the optimal parameters as those that minimize the following least-squares problem

$$\theta^\star = \arg\min_\theta \; \mathbb{E}_{\pi_\theta} \left[ \frac{1}{2} (Q^\star(s, a) - Q_\theta(s, a))^2 \right]. \tag{6.4}$$

As the optimal action-value function generally is unknown, the problem in (6.4) cannot be addressed directly. Next, we will present both a first-order and second-order method of Q-learning. First-order methods use gradient information, whereas second-order methods in addition to the gradient also use second derivatives, also known as the Hessian, to converge faster to the optimum.

### First-order Q-learning

A classical approach to Q-learning is trying to achieve (6.4) by updating the parameters using the TD error defined as

$$\delta_k = y_k - Q_\theta(s_k, a_k), \tag{6.5}$$

where $y_k = L(s_k, a_k) + \gamma V_\theta(s_{k+1})$. We think of $y$ as a fixed target, evaluated using a sampled state transition and the cost. The parameter updates are driven by minimizing the TD error, i.e.

$$\min \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(y - Q_\theta(s,a))^2\right]. \tag{6.6}$$

For the minimization problem in (6.6), we define the following first-order (semi)-gradient step

$$\Delta\theta_Q = \alpha_q \delta_k \nabla_\theta Q_\theta(s_k, a_k), \tag{6.7}$$

where $\Delta\theta_Q = \theta_{k+1} - \theta_k$, $\nabla_\theta Q_\theta(s_k, a_k)|_{\theta=\theta_k}$ and $\alpha_q > 0$ is a scalar denoting the step size. The gradient $\nabla_\theta Q_\theta(s_k, a_k)$ is obtained from sensitivity analysis. For more details on the sensitivity analysis of MPC for Q-learning, the reader is referred to [21].

### Second-order Q-learning

Rather than considering the TD error at each step as in (6.7), we can consider the sum of TD errors over a *batch* of state transitions, known as a least-squares TD (LSTD) algorithm [118]. We formulate a second-order LSTD algorithm for Q-learning (LSTDQ), by considering a root-finding problem, using the gradient in (6.7), i.e.

$$\mathbb{E}_{\pi_\theta}\left[\delta \nabla_\theta Q_\theta(s,a)\right] = 0. \tag{6.8}$$

For the root-finding problem in (6.8), we adopt the following Newton step i.e.

$$\Delta\theta_Q^H = -\alpha_d A^{-1} b, \tag{6.9}$$

using $\Delta\theta$ and $\Delta\theta^H$ to distinguish the first-order and second-order steps, respectively, and $\alpha_d > 0$ to denote the learning rate. We note that for a well-posed update in

(6.9), $A$ is negative definite, and hence generalizes to the first-order step in (6.7) by replacing $A$ with the negative identity matrix. The parameter update is given by

$$A = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \delta \nabla_\theta Q_\theta(s,a)^\top + \delta \nabla^2_\theta Q_\theta(s,a)\right], \tag{6.10a}$$

$$b = \mathbb{E}_{\pi_\theta}\left[\delta \nabla_\theta Q_\theta(s,a)\right], \tag{6.10b}$$

where

$$\nabla_\theta \delta = \gamma \nabla_\theta V_\theta(s^+) - \nabla_\theta Q_\theta(s,a), \tag{6.11}$$

with $\nabla_\theta Q_\theta(s,a)$ and $\nabla^2_\theta Q_\theta(s,a)$ obtained from sensitivity analysis of the MPC in (2.33). The expectations in (6.10a) are evaluated in an episodic manner, by considering $m$ episodes of length $K$, i.e.

$$A = \frac{1}{m}\sum_{j=1}^{m}\sum_{k=1}^{K}\left[\nabla_\theta \delta_{k,j} \nabla_\theta Q_\theta(s_{k,j},a_{k,j})^\top + \delta_{k,j}\nabla^2_\theta Q_\theta(s_{k,j},a_{k,j})\right], \tag{6.12}$$

$$b = \frac{1}{m}\sum_{j=1}^{m}\sum_{k=1}^{K}\delta_{k,j}\nabla_\theta Q_\theta(s_{k,j},a_{k,j}). \tag{6.13}$$

As discussed in Section 2.3.1, there are no guarantees of learning the optimal policy using Q-learning techniques, as small approximation errors in the Q-function estimate, may yield significant errors in the resulting policy estimate.

## 6.2.2 Deterministic policy gradient method

The lack of convergence guarantees for Q-learning methods has motivated the need for alternative methods with more formal (local) convergence guarantees [40]. Using policy gradient methods, the parameters are updated towards improving the performance of the policy irrespective of the action-value function accuracy. In the following, we focus on methods for deterministic policies.

For a rich enough parameterization, the optimal parameters $\theta^\star$ are characterized by

$$\theta^\star = \arg\min_\theta J(\pi_\theta). \tag{6.14}$$

### First-order DPG

Policy gradient methods typically solve (6.14) using gradient descent, which results in the following first-order parameter update

$$\Delta\theta_J = -\alpha_p \nabla_\theta J(\pi_\theta), \tag{6.15}$$

where $\alpha_p > 0$ is the learning rate and $\nabla_\theta J(\pi_\theta)|_{\theta=\theta_k}$. In this chapter, we use the estimation of the DPG as detailed in Section 2.3.2.

### Second-order DPG

For the policy gradient objective in (6.14), we can also formulate a second-order Newton step i.e.

$$\Delta\theta_J^H = -\alpha_r \nabla_\theta^2 J(\pi_\theta)^{-1} \nabla_\theta J(\pi_\theta), \tag{6.16}$$

with learning rate $\alpha_r > 0$. An analytic expression of the deterministic policy Hessian is derived in [112], revealing that it is difficult to estimate from data. The Hessian can be replaced by the Fisher information matrix, resulting in a parameter update known as the *natural policy gradient method* [111]. The Fisher information matrix for deterministic policies is defined as

$$F(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \pi_\theta \, \nabla_\theta \pi_\theta^\top\right], \tag{6.17}$$

and we note that this is only an approximation of the Hessian. The natural policy gradient method yields the following parameter update

$$\Delta\theta_J^H = -\alpha_r F^{-1}(\theta_k) \nabla_\theta J(\pi_\theta). \tag{6.18}$$

Because of the Hessian approximation, the natural policy gradient method still has a linear rate of convergence, despite being a second-order method [119]. This is the same rate of convergence as the first-order DPG method.

## 6.3 Combining RL methods

In the following, we propose to combine Q-learning and DPG methods, to learn a parameterization that optimizes both the Q-learning objective (6.4) and policy gradient objective (6.14) simultaneously. We propose to do so using multi-objective optimization. Multi-objective optimization is applied in many fields where optimal solutions are needed in the presence of trade-offs between two or more conflicting objectives. As opposed to most multi-objective problems, the objective of Q-learning and policy gradient methods are not necessarily in conflict. However, as we can not minimize the true action-value error directly, and may suffer from limitations related to the richness of our function approximator, the parameter update resulting from each RL method may be in conflict. This suggests that an alternative to the naive sum of update laws is needed.

### 6.3.1 Multi-objective RL

With the purpose of combining Q-learning and DPG methods, we define the following multi-objective problem

$$\min_\theta \omega \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(Q^\star(s,a) - Q_\theta(s,a))^2\right] + J(\pi_\theta), \tag{6.19}$$

where $\omega$ is a scalar that weighs the importance of the Q-learning objective relative to the policy gradient objective. In (6.19) we propose a weighted sum method in order to convert the multi-objective problem to a single-objective problem. This method is appealing because it is simple. Alternative methods for solving the multi-objective problem are discussed in e.g. [120].

Theorem 2 states that for a given MDP, an MPC scheme with a possibly inaccurate model can deliver the optimal value function, action-value function, and policy for an appropriate parameterization $\theta$. Building on Theorem 2, we state the following corollary, which is an important statement for the multi-objective RL approach.

**Corollary 1.** *Under Assumption 13 there is no trade-off between the Q-learning and policy gradient objective in (6.19). The minimum of the two objectives will coincide, independent of the scalar value $\omega$.*

*Proof.* For a rich parameterization, we have for the optimal parameters $\theta^\star$ that $Q_{\theta^\star}(s,a) = Q^\star(s,a)$, hence minimizing the Q-learning objective. Moreover, the optimal parameters $\theta^\star$ also minimize the performance $J(\pi_\theta)$ as stated in (6.14). $\square$

As pointed out earlier, we can not directly address the Q-learning objective in (6.19). We therefore adopt the TD approach to Q-learning, as outlined in Section 6.2.1. The multi-objective problem is then

$$\min_{\theta} \omega \mathbb{E}_{\pi_\theta}\left[\frac{1}{2}(y - Q_\theta(s,a))^2\right] + J(\pi_\theta). \tag{6.20}$$

The parameter updates as defined in Section 6.2.1 and 6.2.2, can then be used to define a Newton step that minimizes (6.20). The resulting parameter update is given as the solution to

$$\min_{\Delta\theta_m^{H'}} \frac{1}{2}\Delta\theta_m^{H'\top}(-\omega A + F(\theta) + \tau I)\Delta\theta_m^{H'} + \alpha_m(\nabla_\theta J(\pi_\theta) - \omega b)^\top \Delta\theta_m^{H'}, \tag{6.21}$$

where $A$ and $b$ are defined in (6.12) and (6.13) respectively. We regularize using a scalar $\tau > 0$ and the identity matrix $I$.

**Remark 14.** *As we replace the true action-value error with the TD error in (6.19), Corollary 1 no longer holds, even under Assumption 13. Generally, we cannot expect to achieve an average TD error of zero. This is because the parameter update in (6.9) involves taking the product of two expectations including the next state $s^+$. To obtain an unbiased sample of this product, two independent samples of $s^+$ are needed. During normal interaction with a system, this is not possible. However, we can expect to reduce the average TD errors toward the true minimum.*

The Fisher information matrix will be rank deficient in case we include a storage function in the MPC cost, and depending on the selected parameterization, the Q-learning Hessian can potentially also be ill-posed. The regularization term in (6.21) is therefore often added to prevent the Hessian approximation from becoming singular. Regularization may also be added to ensure positive definiteness.

**Remark 15.** *The Fisher information matrix in (6.17) is positive semi-definite by construction. A well-posed LSTDQ step is characterized by a negative definite Hessian. The multi-objective Hessian composed of the Fisher information matrix and the negative LSTDQ Hessian, should therefore be positive definite. Potentially indefinite Hessian approximations can be tackled using regularization, as proposed here, or trust-region methods, see e.g. [121].*

We note that the computation of the Q-learning Hessian, which is done using sensitivity analysis of (2.33) is not computationally heavy, and much cheaper than solving the optimization problem. The Fisher information matrix (6.17) often used as an approximation of the policy Hessian is rather crude, and we therefore propose the alternative second-order update, where the policy Hessian approximation is omitted, i.e.

$$\min_{\Delta\theta_m^H} \frac{1}{2}\Delta\theta_m^{H\top}(-\omega A + \tau I)\Delta\theta_m^H + \alpha_m(\nabla_\theta J(\pi_\theta) - \omega b)^\top \Delta\theta_m^H. \tag{6.22}$$

We can further simplify the step in (6.22), by replacing the multi-objective Hessian approximation with the identity matrix, and obtain the multi-objective first-order step as the solution to the following

$$\min_{\Delta\theta_m} \frac{1}{2}\Delta\theta_m^\top \Delta\theta_m + \alpha_v(\nabla_\theta J(\pi_\theta) - \omega b)^\top \Delta\theta_m. \tag{6.23}$$

## 6.4 Simulations

In the following section, we consider two simulation examples. The first simulation example is included as a motivating example and is a seemingly simple case of the economic LQR i.e. an LQR with weighting matrices that are not positive definite. The example becomes challenging due to the shape of the action-value function, which calls for a combination of RL methods in order to capture both the correct value function and policy. The second example is a linear MPC (LMPC) that we use to benchmark the convergence of the different parameter update regimes.

### 6.4.1 Economic LQR

We consider an ELQR for a system with dynamics

$$s_{k+1} = 0.1 s_k + a_k, \tag{6.24}$$

and stage cost

$$L(s,a) = -s^2 + 10a^2. \tag{6.25}$$

We introduce the following artificial constraints, to work with compact and bounded constraint sets and thereby comply with dissipativity theory for economic problems [34]

$$-100 \leq a \leq 100, \quad -100 \leq s \leq 100. \tag{6.26}$$

For the set of states that never activate the constraints, we can solve the Riccati equation for the discrete system, and obtain the optimal value function and policy. For the dynamics (6.24) and stage cost (6.25) in the unconstrained case, the optimal value function and policy is

$$V^\star(s) = -1.0113s^2, \; \pi^\star(s) = 0.0113s. \tag{6.27}$$

We formulate the following finite-horizon linear MPC scheme

$$\min_{x,u} \quad -\lambda_\theta(s) + T_\theta(x_N) + \sum_{i=0}^{N-1} \ell_\theta(x_i, u_i) \tag{6.28a}$$

$$\text{s.t.} \quad \forall i \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{6.28b}$$

$$x_{i+1} = 0.1x_i + u_i, \tag{6.28c}$$

$$-100 \leq x_i \leq 100, \tag{6.28d}$$

$$-100 \leq u_i \leq 100. \tag{6.28e}$$

We let $\lambda_\theta, T_\theta$ and $\ell_\theta$ be fully parameterized quadratic functions. The parameter vector $\theta$ is then a vector containing all elements in $\lambda_\theta, T_\theta, \ell_\theta$. All matrices were initialized with the identity matrix. We learned in an episodic manner, using trajectories of length $K = 10$ for initial condition $s_0 = 1.0$ with learning rate $\alpha = 0.1$ for all update schemes. We used a stochastic policy for exploration as described in (6.2), with Gaussian noise described by standard deviation $\sigma_a = 10^{-3}$. We tested learning using first-order Q-learning (6.7), denoted $\Delta\theta_Q$, DPG (6.15) denoted $\Delta\theta_J$ as well as first-order multi-objective combination (6.23) using $\omega = 1$, denoted $\Delta\theta_m$. As the true Q-function for this system is known, and this serves only as a motivational example, we use the analytical gradient in the DPG formulation in (2.23). We saw convergence for all methods after 20 batches consisting of $m = 1$ episodes.

In Figure 6.1 we have plotted the value function estimate and the policy using the final values of the parameters. We see that the pure DPG method fails to learn the value function, but estimates the true policy well. The pure Q-learning step, on the other hand, learns the value function well, but not the policy. The combination of

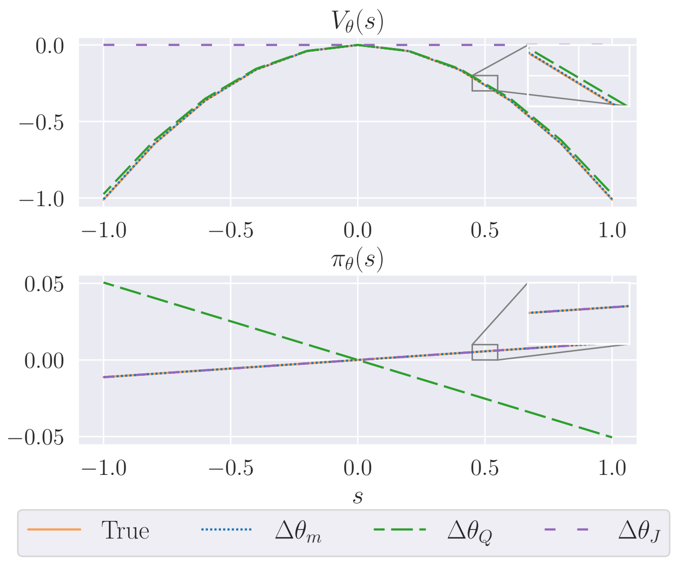


**Figure 6.1:** Economic LQR: Value function estimate (top) and policy (bottom).

methods using the first-order multi-objective update, however, manages to learn both the true value function and policy after 20 episodes.

For economic problems, we can use Q-learning as a tool to learn the storage function, needed to verify dissipativity as detailed in [103]. Verification is done by checking that the rotated cost, denoted by $\bar{\ell}_\theta(s,a)$ and defined by the learned storage function, $\lambda_\theta$, according to

$$\bar{\ell}_\theta(s,a) = L(s,a) - \lambda_\theta(s^+) + \lambda_\theta(s), \tag{6.29}$$

satisfies the following condition i.e.

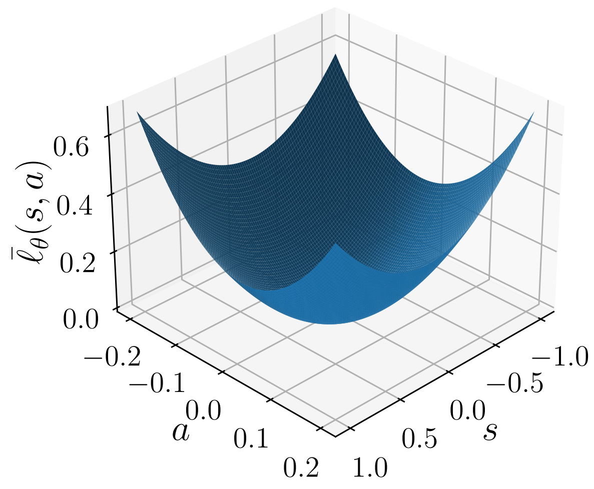
$$\bar{\ell}_\theta(s,a) \geq \rho(\|s\|), \tag{6.30}$$

where $\rho \in \mathcal{K}_\infty$. In Figure 6.2 we see that using DPG alone, the resulting rotated cost is not lower-bounded in $s$, whereas for the multi-objective combination, we obtain a rotated cost that clearly satisfies this lower bound. In summary, we are able to verify that this problem is dissipative, and also capture the correct policy, by using a combination of RL methods.

### 6.4.2 Linear MPC

We consider a discrete linear system of the form

$$s_{k+1} = As_k + Ba_k + n, \tag{6.31}$$



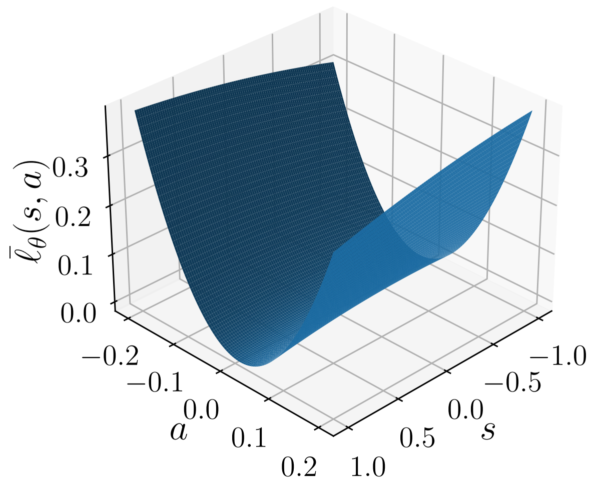


**Figure 6.2:** Economic LQR: The rotated cost (6.29) as defined by the learned storage function using multi-objective RL (top) and using DPG (bottom).

where $n$ describes Gaussian process noise i.e. $n \sim \mathcal{N}(0, \sigma_n^2\, I_n)$, with standard deviation $\sigma_n = 10^{-3}$ and where $I_n$ is the identity matrix. The system matrices are given as

$$A = \kappa \begin{bmatrix} \cos\beta & \sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}, \; B = \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix}, \tag{6.32}$$

where we use $\kappa = 0.95$, and $\beta = 22$ [deg]. The baseline stage cost is selected as

$$L(s, a) = \frac{1}{20}\|s - s_{\text{ref}}\|^2 + \frac{1}{2}\|a - a_{\text{ref}}\|^2 \tag{6.33}$$

where $s_{\text{ref}} = [0.1, 0.1]^\top$, and $a_{\text{ref}}$ is found according to (6.31). The parameterized MPC scheme reads as

$$\min_{x,u} V_0 + \gamma^N \|x_N - x_{\text{ref}}\|_P^2 + \sum_{i=0}^{N-1} \gamma^i f^\top \begin{bmatrix} x_i \\ u_i \end{bmatrix}$$
$$+ \sum_{i=0}^{N-1} \gamma^i \left\| \begin{bmatrix} x_i - x_{\text{ref}} \\ u_i - u_{\text{ref}} \end{bmatrix} \right\|^2 \tag{6.34a}$$
$$\text{s.t.} \quad \forall i \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{6.34b}$$
$$x_{i+1} = \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_3 & \theta_4 \end{bmatrix} x_i + \begin{bmatrix} \theta_5 & 0 \\ 0 & \theta_6 \end{bmatrix} u_i, \tag{6.34c}$$
$$\begin{bmatrix} -0.05 \\ -0.05 \end{bmatrix} \leq u_i \leq \begin{bmatrix} 0.05 \\ 0.05 \end{bmatrix} \tag{6.34d}$$

using a prediction horizon of $N = 10$ and discount factor $\gamma = 0.99$. The stage cost consists of a quadratic function of the state and input reference and a linear term that can be used to shift the minimum of the stage cost. We let $x_{\text{ref}} = s_{\text{ref}}$, but use the prediction model to obtain the input reference, i.e. $u_{\text{ref}} \neq a_{\text{ref}}$. The linear term is defined by a vector $f^\top = [f_1, f_2, f_3, f_4]$. The parameter vector is $\theta = \{V_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, f_1, f_2, f_3, f_4\}$. The prediction model is initialized with parameter values $\theta_1 = \theta_3 = \cos\hat{\beta}$ and $\theta_2 = \theta_4 = \sin\hat{\beta}$, where $\hat{\beta} = 30$ [deg], and $\theta_5 = 1.3, \theta_6 = 0.7$. We let $f_1 = f_2 = f_3 = f_4 = 0.3$ and $V_0 = 0$. We use a quadratic terminal cost, for which $P$ is found solving the discrete Riccati equation for the prediction model, using the initial values of the parameters.

We learned in an episodic manner, simulating the system from initial condition $s_0 = [0, 0]^\top$, for a total of 25 batches, consisting of $m = 10$ episodes of length $K = 50$. A stochastic policy is used for exploration as defined in (6.2), with a Gaussian noise term distributed as $\zeta_a \sim \mathcal{N}(0, \sigma_a^2\, I_m)$, where $\sigma_a = 5 \cdot 10^{-3}$. For this example, we tested both first- and second-order multi-objective RL as described in Section 6.3.1. We compared their performance with both the second-order and

**Table 6.1:** LMPC: Update schemes tested in simulation.

| Label | Name and eq. reference | Learning rate | $L_{\text{tot}}$ |
|---|---|---|---|
| $\Delta\theta_Q$ | Q-learning (6.7) | $\alpha_q = 0.01$ | 19.69 |
| $\Delta\theta_Q^H$ | LSTDQ (6.9) | $\alpha_d = 0.5$ | 18.63 |
| $\Delta\theta_m^H$ | Second-order multi-objective (6.22) | $\alpha_m = 0.5$ | **4.63** |
| $\Delta\theta_m^{H'}$ | Second-order multi-objective (6.21) | $\alpha_m = 0.5$ | 6.64 |
| $\Delta\theta_J^H$ | Natural policy gradient (6.18) | $\alpha_r = 0.5$ | 5.12 |
| $\Delta\theta_m$ | First-order multi-objective (6.23) | $\alpha_v = 0.5$ | 12.46 |
| $\Delta\theta_J$ | DPG (6.15) | $\alpha_p = 0.5$ | 12.37 |

first-order policy gradient methods, as well as classical Q-learning and LSTDQ. For the Q-learning Hessian and Fisher information matrix, we regularized using $\tau = 10^{-1}$ and $\tau = 10^{-2}$ respectively, at all time steps. The learning rates were selected by testing each method for an interval of learning rates and selecting the best-performing one. The sum of costs was evaluated for the best-performing case, i.e.

$$L_{\text{tot}} = \sum_{j=1}^{m} \sum_{k=1}^{K} \gamma^k L(s_{k,j}, a_{k,j}), \tag{6.35}$$

and is reported in Table 6.1. All plotting labels are also listed and explained in Table 6.1. In Figure 6.3 we see the closed-loop performance produced by each of the learning schemes. Because the RL cost in (6.33) is rather flat in $s$-direction, this is typically a challenge for Q-learning alone, and hence we see both classical TD-learning and LSTD-learning achieving the poorest performance. First-order DPG and the first-order multi-objective approach achieve more or less the same closed-loop performance. Moreover, we see that the closed-loop performance obtained using the natural policy gradient method converges faster than both first-order methods. The second-order multi-objective approach performs better without the Fisher information matrix in the Hessian, as given in (6.22), and speeds up convergence further. The aforementioned observations align also with the calculated sum of discounted costs during learning for the different update schemes as listed in Table 6.1. For the selected learning rates, we obtained the best closed-loop performance for the multi-objective approach by using $\omega = 0.1$ in (6.22), (6.21) and (6.23).

In Figure 6.4 we have plotted the states and inputs resulting from the learned parameters in exploitation, i.e. without exploration noise. We note that the upper bounds on the inputs become active constraints as we approach the optimal policy,

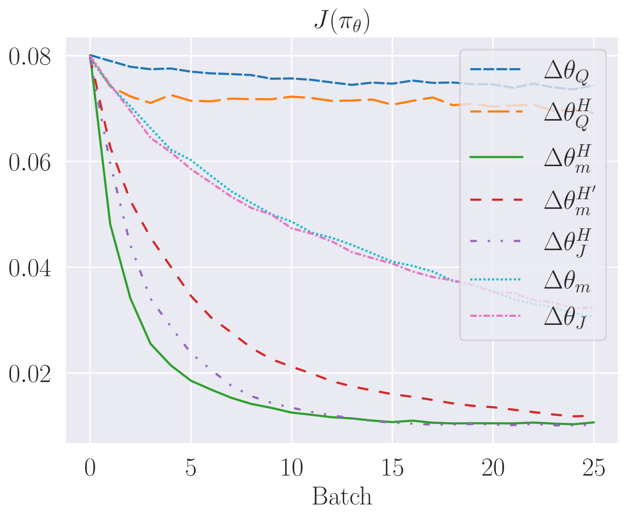


**Figure 6.3:** LMPC: Mean closed-loop performance over learning batches.

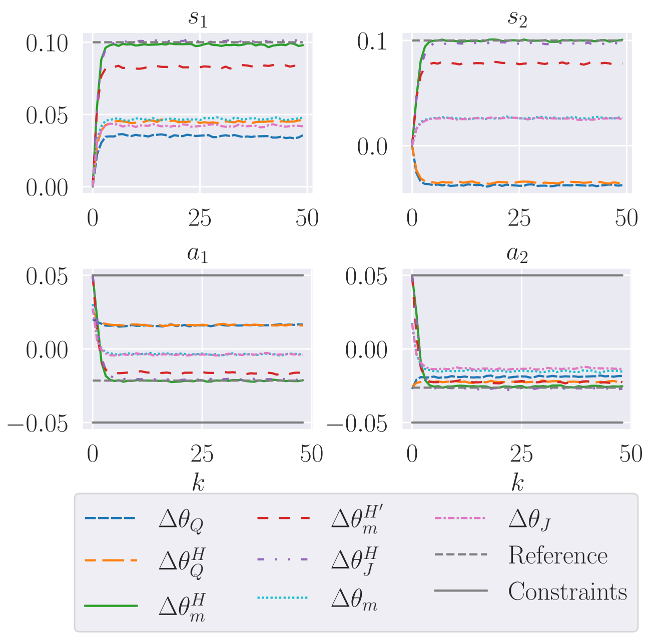


**Figure 6.4:** LMPC: Simulated states and actions using the learned parameters.

which is the case for the second-order multi-objective and natural policy gradient method.

For the second-order multi-objective approach, we saw that the Q-learning gradient part caused learning to converge to a policy that caused a small steady-state error from the desired state and input reference. This is likely caused by the same effect as addressed in Remark 14. We therefore removed the first-order Q-learning contribution to produce the results shown in Figures 6.3 and 6.4, at the cost of some reduction in convergence rate. Ideally, $\omega$ could be gradually reduced, to gain the full increase in convergence rate, while still converging to the true optimum.

## 6.5 Conclusion

In this chapter, we proposed a multi-objective approach for combining RL methods, in order to fully exploit the parameterization provided by the MPC scheme and increase the convergence rate of learning. The first simulation example illustrates that for certain economic policies, we need a combination of Q-learning and policy gradient methods in order to verify dissipativity and learn both the optimal value function and policy. The second simulation example demonstrates that the proposed multi-objective second-order step, combining Q-learning and policy gradient methods, speeds up convergence in learning compared to both a second-order policy gradient method and a second-order Q-learning method.

# Chapter 7

# Variance-based exploration for learning MPC

RL is dependent on exploration to learn, and currently, simple heuristics based on random perturbations is most common. This chapter considers variance-based exploration in RL geared towards MPC as function approximators. We propose to use a non-probabilistic measure of uncertainty of the value function approximator in value-based RL methods. Uncertainty is measured by a variance estimate based on inverse distance weighting (IDW). The IDW framework is computationally cheap to evaluate and therefore well-suited in an online setting, using already sampled state transitions, actions, and rewards. The gradient of the variance estimate is then used to perturb the policy parameters in a direction where the variance of the value function estimate is increasing. The proposed method is verified on two simulation examples, considering both linear and nonlinear system dynamics, and compared to standard exploration methods using random perturbations. The following chapter is based on work in [27].

## 7.1 Introduction

RL requires that the actions applied to the real system undergo some exploration. If the same, deterministic policy is always applied to the system, it is not possible to discover alternative actions that may improve closed-loop performance. One way to quantify an effective exploration is in terms of regret. The notion of regret in RL is defined as the loss in reward for choosing a suboptimal over an optimal action. An effective exploration strategy can then be defined as minimizing the cumulative sum of regrets. However, we cannot directly obtain the regret as the optimal action is not known. Hence, the concept of regret in itself cannot be used to perform effective exploration.

Currently, the most commonly used methods to explore are simple heuristics. For discrete action spaces, methods such as $\epsilon$-greedy [39] or Boltzmann exploration [122] are used. For continuous action spaces, stochastic policies for exploration are generated e.g. by adding Gaussian noise to a deterministic policy [123]. In the case of using stochastic policy gradient methods, the distribution of the stochastic policy itself is parameterized and adjusted by RL. Exploration is then ensured by sampling from the resulting distribution that describes the stochastic policy [39].

A collective term for the aforementioned exploration strategies is *dithering* strategies. Because the perturbation from one time step to the next is not coordinated, the exploration is not temporally extended or what we refer to as *deep*. For problems that require consistent exploration over several time steps to realize improved closed-loop performance, dithering strategies may in fact prevent efficient exploration. The most straightforward method for ensuring deep exploration, is random perturbations in parameter space, as suggested by the authors in [124]. A random perturbation in the parameters is introduced at the beginning of an episode, and fixed throughout that episode, such that a temporally coordinated sequence of actions is generated. However, the potential benefit of using random noise in parameter space rather than in action space is generally not obvious and needs to be evaluated on a case-by-case basis. Moreover, when using random parameter noise for exploration in large parameter spaces, we are at risk of adding a lot of disturbances that yield little effect on the resulting policy.

Although the aforementioned heuristics perform well for many tasks, they are all undirected and therefore may take exponentially long to learn the optimal policy [125]. To learn efficiently, the exploration method should prioritize potentially informative states and actions. To do this, exploration should be done with regard to a notion of uncertainty.

Directed exploration is well understood for the multi-armed bandit problem, which corresponds to a one-step stateless MDP problem. One strategy is “optimism in the face of uncertainty”, which corresponds to preferring actions with uncertain values. This strategy has led to e.g. the upper confidence bound (UCB) algorithm. The UCB algorithm acts greedily w.r.t. to the action-value function and an exploration bonus based on a confidence interval of the reward, see e.g. [126]. For the bandit problem, Hoeffding’s inequality can be applied to obtain the UCB. The UCB algorithm in [126] has been extended to RL in different forms, but the resulting algorithms have been considered mostly as theoretical results due to the computational complexity.

Thompson sampling (TS) is a related strategy developed for the bandit problem. TS sampling is based on a Bayesian approach to maintaining a posterior distribution over models. We sample from the posterior distribution and then select the action that optimizes the sampled model [127]. Extending TS to RL would involve maintaining a distribution of MDPs, which in general is difficult. Even updating a Bayesian model of the value function for an MDP will for most realistic problem sizes be computationally intractable. For bandit problems, both UCB and TS achieve a sublinear total regret. In comparison, $\epsilon$-greedy has a linear total regret, which is the same as with no exploration at all.

As a means to reduce the computational burden, yet inspired by TS, the authors in [128] introduced the concept of randomized value functions. The use of randomized value functions aims to approximate samples from the posterior distribution of the value function. However, the method is developed only for linear parameterizations of the value function. An extension was made to nonlinear parameterizations, more specifically to NNs, in [129], where bootstrapped deep Q-function NNs (DQN) were used to approximate the posterior distribution of the Q-function.

The concept of randomized value functions has also motivated the *NoisyNets* as proposed in [130]. Rather than training an NN with $K$ outputs or heads to build an approximate posterior distribution of Q as in [129], the authors in [130] inject noise in the NN parameters and use RL to tune the intensity i.e. the variance of the noise distributions. A new sample of the Q-function is then obtained by sampling parameter noise from the tuned noise distributions, and exploration is done by acting greedily with respect to that Q-function.

Bootstrapped DQNs have also been used to develop a practical UCB approach that applies to RL. Namely, the bootstrapped DQN was used to obtain an empirical estimate of the mean and standard deviation of the Q-function distribution [131]. This was in turn used to formulate a UCB, and the action was selected by maximizing the UCB. Along the same lines, the DQN framework was used by the authors in [132] in order to obtain confidence intervals to formulate a surrogate of the regret, which in turn was used to guide exploration. The use of randomized value functions constitutes an important step towards more effective exploration strategies in RL, although it for nonlinear value function parameterizations only applies to discrete action spaces.

### 7.1.1 Contribution

The goal of this work is to develop a directed and deep exploration strategy for continuous action spaces, that is suitable for problems where we wish to use MPC as a function approximator in RL. For this purpose, we will adopt the principle of "optimism in the face of uncertainty". To the best of the authors' knowledge, few studies exist on directed exploration strategies in continuous action spaces. One important exception is the work in [133], where $K$ value function approximators are trained independently, and the agent is encouraged to explore states where the value function approximators show the largest disagreement. Although the exploration strategy resembles ours, it is based on knowing the true model of the MDP, which is not a requirement in our case. We present an uncertainty-based exploration method not limited to, but particularly suited for MPC, and make the following contributions:

- we introduce the use of IDW to estimate the variance of the MPC function approximator at a low computational cost;
- we formulate variance-based exploration in parameter space via the IDW variance estimate;
- we compare the proposed method with random (Gaussian) perturbations in both action and parameter space.

The proposed method is verified on two simulation examples, considering both linear and nonlinear dynamics, for which variance-based exploration performs better in terms of significantly improving the cumulative rewards during learning.

## 7.2 Background

In the following, we will consider a special case of MDPs, described by the deterministic dynamics

$$s_{k+1} = f(s_k, a_k), \tag{7.1}$$

where $s_{k+1}$ denotes the next state and $k$ denotes the physical time of the system. The system is subject to the following constraints given by

$$h(s_k, a_k) \leq 0. \tag{7.2}$$

We will assume in the following that a stage cost $L(s_k, a_k)$, analogous to a negative reward, is provided. Our goal is to find a deterministic policy $\pi$ that maps from state to action i.e. $\pi : \mathcal{S} \to \mathcal{A}$, that minimizes the sum of discounted costs. The optimal policy, denoted $\pi^\star$, is the solution to the following infinite-horizon problem

$$\begin{aligned} V^\star(s) = \min_{\pi} \quad & \sum_{j=0}^{\infty} \gamma^j L(x_j, \pi(x_j)) && \text{(7.3a)} \\ \text{s.t.} \quad & \forall j \in \mathbb{I}_{\geq 0} : \; x_0 = s, && \text{(7.3b)} \\ & x_{j+1} = f(x_j, \pi(x_j)) && \text{(7.3c)} \\ & h(x_j, \pi(x_j)) \leq 0, && \text{(7.3d)} \\ & \pi(x_j) \in \mathcal{A}, && \text{(7.3e)} \end{aligned}$$

where $V^\star(s)$ is the optimal value function, $\gamma \in (0, 1]$ is a discount factor and $\{x_j\}_{j=1}^{\infty}$ is the predicted state trajectory under the policy $\pi$ for an initial state $s$. The action-value function for a policy $\pi$ can then be defined as follows

$$Q_\pi(s, a) = L(s, a) + \gamma V_\pi(f(s, a)), \tag{7.4}$$

and is related to the value function through the Bellman equality

$$V_\pi(s) = Q_\pi(s, \pi(s)) = \min_a Q_\pi(s, a). \tag{7.5}$$

We then parameterize the value function using parameter vector $\theta$ and adjust these using RL towards the optimal value function, and thereby the optimal policy.

### 7.2.1 MPC as a function approximator in RL

We consider parameterized MPC schemes to estimate the value function $V_\theta(s)$ as in (2.30) and action-value function $Q_\theta(s, a)$ as in (2.33).

#### Q-learning

A classical approach to Q-learning is parameter updates driven by the TD error defined as

$$\delta_k = L(s_k, a_k) + \gamma V_\theta(s_{k+1}) - Q_\theta(s_k, a_k). \tag{7.6}$$

At each time step the parameters are updated according to

$$\theta \leftarrow \theta + \alpha \delta_k \nabla_\theta Q_\theta(s_k, a_k), \tag{7.7}$$

where $\alpha > 0$ is a scalar denoting the step size. The gradient $\nabla_\theta Q_\theta(s_k, a_k)$ can be obtained from sensitivity analysis of the MPC scheme, as detailed in [21].

An alternative to the incremental update of parameters in (7.7), is a batch approach to Q-learning. This method is known to result in more stable learning [39]. A batch approach entails introducing an additional set of parameters $\tilde{\theta}$ that is continuously being updated

$$\tilde{\theta} \leftarrow \tilde{\theta} + \alpha \tilde{\delta}_k \nabla_{\tilde{\theta}} Q_{\tilde{\theta}}(s_k, a_k), \tag{7.8}$$

where $\tilde{\delta}_k = L(s_k, a_k) + \gamma V_{\tilde{\theta}}(s_{k+1}) - Q_{\tilde{\theta}}(s_k, a_k)$. The action $a_k$ is selected according to the action-value function $Q_\theta(s_k, a_k)$, defined by parameters $\theta$ that remain fixed for the duration of one batch. As the updated parameters $\tilde{\theta}$ converge, which marks the end of a batch, we replace the fixed parameters $\theta$ with the updated ones $\tilde{\theta}$ and begin a new batch.

Q-learning techniques aim to fit $Q_\theta(s,a)$ to $Q^\star(s,a)$, with the hope that $Q_\theta \approx Q^\star$ will result in $\pi_\theta \approx \pi^\star$. To make this fitting possible, we have to deviate from the current policy estimate, i.e. explore. A standard strategy for exploring in the case of continuous actions is adding random perturbations to the policy, e.g. in the form of Gaussian noise. This results in a stochastic policy that induces undirected exploration, i.e.,

$$\mu_\theta(a|s) = \pi_\theta(s) + \zeta_a, \tag{7.9}$$

where $\zeta_a$ is normally distributed according to $\zeta_a \sim \mathcal{N}(0, \sigma_a^2 I_a)$ where $\sigma_a$ is the standard deviation and $I_a$ is the identity matrix. Convergence properties for Q-learning are established in e.g. [134] and elaborated further in Section 7.5.2. The stochastic policy in (7.9) will serve as a baseline for the variance-based exploration method proposed in this chapter.

## 7.3 Parameter space exploration

Exploration in parameter space is closely related to the concept of randomized value functions, which may be used as an alternative to TS without the need for an intractable exact posterior representation. Exploration in parameter space has been studied in, e.g., [124] and [130]. The referenced works are similar in the sense that NNs are used for approximating the value function, and that a sample from an approximate posterior distribution of the value function is used to explore.

To the best of the authors' knowledge, only exploration in action space has been tested for MPC as a function approximator in RL. Comparing NNs and MPC schemes as function approximators, we conjecture that exploration in parameter space is particularly suited when using MPC, as the parametrization is smaller and less convoluted than for NNs (due to the layers and consecutive nonlinear activations), and also more easily interpreted.

We therefore propose to adopt exploration in the parameter space for MPC, by adding uncorrelated Gaussian noise to the parameters as follows

$$\hat{\theta} = \theta + \zeta_p, \tag{7.10}$$

where $\zeta_p \sim \mathcal{N}(0, \sigma_p^2 I_p)$. The exploration policy is then obtained by acting greedily with respect to the Q-function defined by the perturbed parameters, i.e.

$$\hat{\pi}_{\hat{\theta}} = \arg\min_a Q_{\hat{\theta}}(s,a). \tag{7.11}$$

The exploration method is summarized in Algorithm 1, here for a continuous task. We note that the method easily extends to an episodic task.

**Algorithm 1:** Exploration in parameter space

```
1 Input : Initial MPC parameters $\theta_0$, initial learning parameters $\tilde{\theta}_0$, initial state
    $s_0$, batch update frequency $b$, learning step size $\alpha$, parameter noise standard
    deviation $\sigma_p$, length of simulation $k_{\max}$ ;
2 Output : Policy $\pi_\theta$
3 while $k \leq k_{max}$ do
4     if $mod(k, b) = 0$ then
5         Update MPC parameters with learned parameters $\theta_k = \tilde{\theta}_k$
6         Perturb parameters to get $\hat{\theta}_k$ (7.10)
7     Act greedily w.r.t. current Q-estimate (7.11)
8     Update $\tilde{\theta}_k$ (7.8)
9     $k \leftarrow k + 1$
```

**Remark 16.** *We note that exploration in parameter space in combination with Q-learning, generally calls for a batch approach to Q-learning as given by (7.8). For an incremental approach as in (7.7) where we only have one set of parameters $\theta$, the resulting TD-error as we update the parameters according to (7.10) would be $L(s_k, a_k) + \gamma V_{\hat{\theta}}(s_{k+1}) - Q_{\hat{\theta}}(s_k, \pi_{\hat{\theta}})$ where $Q_{\hat{\theta}}(s_k, \pi_{\hat{\theta}}) = V_{\hat{\theta}}(s_k)$, i.e. we are fitting the V-function rather than the Q-function.*

Depending on the selected parameterization and the resulting range of parameters, which in turn depends on the problem at hand, we may choose to perturb the normalized parameters in order to use the same scale for perturbing the entire parameter vector. Alternatively, the states and actions in the problem can also be scaled, which will result in a smaller variation in parameter range, which is also known to speed up learning.

In the next section, we will consider an alternative to adding random perturbations to the parameters, namely adding a perturbation based on the uncertainty of the value function. We will use the exploration method in (7.10) as a second baseline for our proposed method.

## 7.4 Value function IDW variance

To guide exploration using the uncertainty of our value function estimate, a method is needed for quantifying such uncertainty. In Bayesian exploration, we use the covariance of the resulting posterior distribution of a GP, to guide exploration. Alternatively, interpolation methods can be used to define non-probabilistic uncertainty measures that are computationally cheaper to evaluate. In [135], radial basis functions (RBFs) were used to formulate a measure of uncertainty based on sampled points. In [136], an uncertainty measure based on IDW was compared to a Bayesian exploration for global optimization and showed competitive performance. We propose to use IDWs for quantifying the uncertainty of the value function.

The IDW framework can be used to estimate uncertainty for both the value function and the action-value function. Because we will use the IDW variance to measure parametric uncertainty only, we do not need to consider the additional argument of the action-value function, and therefore choose to apply IDW to the value function.

### 7.4.1 Inverse distance weighting

Given a data set, IDW is an interpolation method that also provides us with a variance estimate given a predictor of the function that is sampled. We assume that we have a data set consisting of $M$ samples $V_i = V(\eta_i)$ of $V : \mathbb{R}^q \to \mathbb{R}$ at corresponding points $\eta_1, \dots, \eta_M$. In the following, the function $V$ is the value function, that we for now assume that we can sample, and $\eta = [\theta, s]^\top$.

We may consider the following scaling function $\phi : \mathbb{R}^q \to \mathbb{R}^q$ to be immune to the different scaling of individual parameters and states

$$\phi(\eta) = \operatorname{diag}\left(\frac{2}{\eta_{\max} - \eta_{\min}}\right)\left(\eta - \frac{\eta_{\max} + \eta_{\min}}{2}\right), \tag{7.12}$$

so that $\phi(\eta) \in [-1, 1]$ for all $\eta \in [\eta_{\min}, \eta_{\max}]$, where $[\eta_{\max}, \eta_{\min}] \subset \mathbb{R}^q$. The min and max values of the states and parameters can be based on constraints and reasonable bounds on possible parameter values.

For a new instance of $\eta$, we consider the (scaled) squared Euclidean distance function $d^2 : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}$ given as

$$d^2(\eta, \eta_i) = (\phi(\eta_i) - \phi(\eta))^\top (\phi(\eta_i) - \phi(\eta)), \quad i = 1, \dots, M. \tag{7.13}$$

The IDW function $w_i : \mathbb{R}^q \to \mathbb{R}$ can then be defined as in [137]

$$w_i(\eta) = \frac{1}{d^2(\eta, \eta_i)}, \tag{7.14}$$

and assigns larger weights $w_i(\eta)$ to samples that are close to $\eta$ than to samples further away. In [138], the following alternative weighting function was suggested

$$w_i(\eta) = \frac{e^{-d^2(\eta, \eta_i)}}{d^2(\eta, \eta_i)}. \tag{7.15}$$

The weighting function in (7.15) is similar to (7.14) for small values of $d^2$, but more quickly reduces the effect of points $\eta_i$ far away from $\eta$ due to the exponential term. We then define the following function $v_i : \mathbb{R}^q \to \mathbb{R}$ as

$$v_i(\eta) = \frac{w_i(\eta)}{\sum_{j=1}^{M} w_j(\eta)} \tag{7.16}$$

which allows us to define

$$\bar{V}(\eta) = \sum_{i=1}^{M} v_i(\eta) V_i, \tag{7.17}$$

which is an IDW interpolation of $\{(\eta, V_i)\}_{i=1}^{M}$. For a new instance of $\eta$, an estimate of $V(\eta)$ is obtained by interpolating on the $M$ existing samples of $V$. In its original form, the selection function in (7.16), will include the option that in case we evaluate an already sampled instance of $\eta$, i.e. $\eta \in \{\eta_i, \dots, \eta_M\}$, we use that $v_i(\eta_j) = 1$ for $i = j$ and $v_i(\eta_i) = 0$ otherwise. However, for our purpose we will not evaluate already sampled values of $\eta$, as we wish to explore new values of the parameters. This is explained further in Section 7.5.1. It was shown in [139, Lemma 1], for both choices of the weighting function, (7.14) and (7.15), that the interpolation function $\bar{V}(\eta)$ is differentiable everywhere on $\mathbb{R}^q$.

Based on the IDW interpolation function, we define the IDW variance function $r^2 : \mathbb{R}^q \to \mathbb{R}$ as given by [138]–[140]

$$r^2(\eta) = \sum_{i=1}^{M} v_i(\eta) \big(V_i - \hat{V}(\eta)\big)^2, \tag{7.18}$$

where $V_\theta(s) = \hat{V}(\eta)$, $V_\theta(s)$ being the parameterized value function in (2.30) for which we want to estimate the variance. Essentially, the IDW variance estimate is a weighted average of the squared error between the sampled value functions $V_i$ and the predictor $\hat{V}(\eta)$. As $\hat{V}(\eta)$ is differentiable, it follows that the IDW variance estimate is also differentiable everywhere on $\mathbb{R}^q$. Figure 7.1 is one example of how the IDW variance estimate can be used to define error bounds for a predictor of a function $V$ with, in this case, a scalar argument $\eta$.

### 7.4.2 $p$-step TD prediction of V

Monte Carlo (MC) learning involves learning from experience, using sequences of states, actions, and costs. MC learning offers an alternative method to TD-learning (7.6) of the action-value or value function. For a general MDP with stochastic dynamics and possibly a stochastic policy, MC learning uses the mean sum of discounted costs from several episodes of experience as the value function estimate.

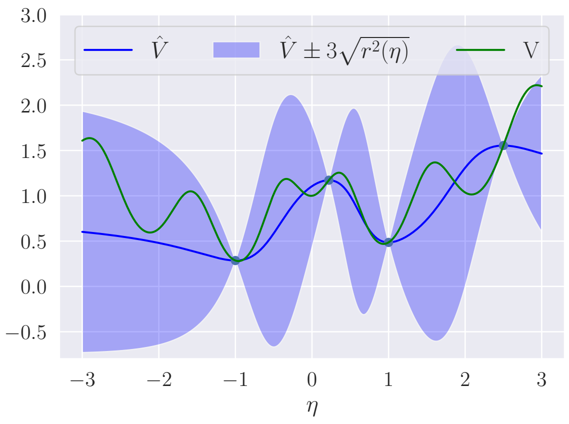


**Figure 7.1:** Example of IDW: Function $V$ (green) and samples $(\eta_i, V_i)$ (blue dots). Error bands are given using the IDW variance estimate in (7.18) i.e. $\pm 3\sqrt{r^2(\eta)}$ (shaded blue) evaluated for the predictor $\hat{V}(\eta)$.

In the following we propose to use MC learning of the value function as a target in the IDW framework.

As we are considering deterministic policies for deterministic environments, one episode is sufficient to provide an MC value function estimate. Because the true value function is defined for an infinite horizon (7.3), we bootstrap on our value function estimate to compensate for considering an episode with finite length. We estimate the value function as the sum of discounted costs, evaluated for $p$ samples of the state and $p-1$ samples of the action, bootstrapping on the value function approximator for the last state. This can be described as a $p$-step prediction of V, based on a *first-visit* approach to MC learning [39].

We now consider a policy $\pi_{\bar{\theta}}$, where $\bar{\theta}$ denotes the parameter vector. For notational convenience we let $\pi_{\bar{\theta}} = \bar{\pi}$. For $p \geq 1$ we consider sampled states and actions, i.e.

$$\{s_{k-p}, a_{k-p}, s_{k-p+1}, \dots, a_{k-1}, s_k\}, \tag{7.19}$$

obtained by acting according to policy $\bar{\pi}$. The data in (7.19) is used to predict the value function by evaluating the sum of discounted costs. The $V$ estimate is given as

$$\begin{aligned} V_k(\bar{\theta}_k, s_{k-p}) = & L(s_{k-p}, a_{k-p}) + \gamma L(s_{k-p+1}, a_{k-p+1}) \\ & + \gamma^2 L(s_{k-p+2}, a_{k-p+2}) + \dots \\ & + \gamma^{p-1} L(s_{k-1}, a_{k-1}) + \gamma^p V_{\bar{\theta}_k}(s_k), \end{aligned} \tag{7.20}$$

where we evaluate the value function estimate $V_{\bar{\theta}}$ for the last sampled state, using the current parameter vector $\bar{\theta}_k$. The number of samples $p$ thereby becomes a hyperparameter. Because we bootstrap on our value function, both the targets and the predictor in (7.18) are based on the function approximator in (2.30). It is therefore important that $p$ is large enough, such that the effect of bootstrapping is small. Moreover, the effect of discounting will reduce the bias of bootstrapping. Selecting $p$ large relative to the batch size means that we obtain samples of the value function for only a few instances of the state. In an episodic setting where we initialize the system from the same initial condition at the beginning of a new batch, we will evaluate the value function variance for the same state and therefore depend on fewer samples of the value function target. In a continuing task on the other hand, the system can be in different states at the beginning of each new batch, and we need to provide samples of the value function targets for more instances of the state. An alternative to the target we proposed in (7.20), is an exponentially weighted estimate as the authors propose for the advantage function in [141].

**Remark 17.** *As the variance-based exploration method is perturbing in parameter space, we should, as for the random exploration in parameter space, use a batch manner to Q-learning as given in (7.8). This entails having two sets of parameters, where one set of parameters is continuously updated $\tilde{\theta}$, whereas the other set of parameters $\theta = \bar{\theta}$ are fixed for one batch and is used to define a policy that visits informative states.*

Using the IDW variance function in (7.18), we obtain a variance estimate based on the weighted average of the squared difference between the targets in (7.20) and the MPC-based value function approximator in (2.30). The V-function estimate in (7.20) is particular for the current policy estimate $\bar{\pi}$, i.e. we are estimating $V_{\bar{\pi}}$. As the parameters are updated from $\bar{\theta}$ to $\bar{\theta}'$ at the beginning of a new batch, data is collected with an updated policy $\bar{\pi}'$, and we are making $p$-step predictions of $V_{\bar{\pi}'}$. Although the previously sampled $V$ targets are estimated for increasingly more outdated policies, they can be useful for estimating the uncertainty w.r.t the parameters. The weights in (7.18) are assigned using inverse distances according to either (7.14) or (7.15), i.e. we influence the impact of previously sampled targets on our variance estimate through the choice of IDW function.

### 7.4.3 Practical implementation

As the number of samples $M$ increases, the IDW variance function becomes increasingly computationally heavy to evaluate. For a practical implementation, that can run fast in real-time, we set a limit for the maximum number of samples $M_{\max}$ used to evaluate (7.18). If our data set already contains $M_{\max}$ samples, we evaluate the following criterion for a new instance of $\eta$,

$$\sum_{i=1}^{M_{\max}} d^2(\eta, \eta_i) > \min\left\{ \sum_{i=1}^{M_{\max}} d^2(\eta_i, \eta_1), \ldots, \sum_{i=1}^{M_{\max}} d^2(\eta_i, \eta_{M_{\max}}) \right\}. \tag{7.21}$$

If the summed distance from a new sample $\eta$ in (7.21) to our current samples $\eta_1, \ldots, \eta_{M_{\max}}$ is larger than the least different sample in our dataset, we will replace the old sample with the new. In the opposite case, we will not update our data set. The maximum number of samples $M_{\max}$ thereby becomes a hyperparameter.

**Remark 18.** *Although the size of the data set in this framework can be controlled by limiting the number of samples to include, the parameter and state dimension will also affect the size. The IDW variance (7.18) is only evaluated at the beginning of each batch, so the computation time of the variance itself may therefore not necessarily be a problem. Nonetheless, a small parameter space will ease the work related to handling the sampled data needed to evaluate the IDW variance. This framework is therefore particularly suited for using MPC as a function approximator, which typically uses much fewer parameters than the standard choice of NNs.*

## 7.5 Variance-based exploration

In this section, we will outline how we can leverage an IDW framework as detailed in the previous section, to direct exploration to where we have uncertainty in our value function estimate. We are then acting according to the strategy of "optimism in the face of uncertainty", and hoping that by exploring areas of high uncertainty our policy will visit more informative states and actions, and hence explore more efficiently.

Combining a variance-based exploration in the parameter space with a batch approach to Q-learning, allows us to consider a perturbation to the parameters at the beginning of each batch, and keep these parameter values for the duration of one batch. The advantage of this approach is that we induce a state-dependent change in the policy over multiple time steps, what is often referred to as *deep* exploration.

### 7.5.1 Variance-based perturbations in parameter space

We first define the gradient of the IDW variance function w.r.t. the parameters:

$$\begin{aligned}\nabla_\theta r^2(\eta) = \sum_{i=1}^{M} & \nabla_\theta v_i(\eta)(V_i - \hat{V}(\eta))^2 \\ & -2v_i(\eta)(V_i - \hat{V}(\eta))\nabla_\theta \hat{V}(\eta),\end{aligned} \tag{7.22}$$

to highlight the fact that the sensitivity of the MPC scheme, in terms of $\nabla_\theta \hat{V}(\eta)$, is used in evaluating the gradient of the variance. We propose to restrict the variance gradient by using the standard deviation used for Gaussian exploration in parameter space. This will, first of all, make the two methods highly comparable, in terms of how large the applied perturbation in the parameters can be, and also prevents the addition of yet another hyperparameter, i.e.

$$\nabla_\theta \hat{r}^2(\theta, s) = \mathrm{sat}(\nabla_\theta r^2(\theta, s), -2\sigma_p, 2\sigma_p). \tag{7.23}$$

We propose the following update of the parameters

$$\bar{\theta} = \theta + \frac{1}{2}\nabla_\theta \hat{r}^2(\theta, s) + \frac{1}{2}\zeta_p, \tag{7.24}$$

where $\zeta_p$ is a noise term as defined for (7.10). The gradient of the IDW function is added to the parameters, in the hope that exploring parameter values in a direction where our value function is uncertain, may improve our estimate. The noise term is added to ensure some random exploration in parameter space, to collect data such that the variance estimate of our function approximator is meaningful. A policy estimate based on the perturbed parameters in (7.24), is obtained according to

$$\pi_{\bar{\theta}}(s_k) = \arg\min_a Q_{\bar{\theta}}(s_k, a_k). \tag{7.25}$$

The formulation in (7.24) and (7.25) resembles the exploration method of randomized value functions, where a value function is sampled from an approximate posterior distributed and then used for greedy action selection. However, our approach is not completely random in sampling the value function but uses the gradient of the variance estimate to guide the sampling. We predict $V_i$ for each realization of $\bar{\theta}$ as well as different states, using $p$ samples of states and actions during a batch, and store it in the data set $\mathcal{D}$. The data set is used to re-evaluate the variance gradient at the beginning of the next batch, to generate a new (perturbed) parameter vector to be used. This is summarized in Algorithm 2.

### 7.5.2 Convergence properties

Without considering that we are using function approximators, Q-learning will converge under the following assumptions: (1) Greedy in the limit with infinite exploration (GLIE), (2) the step sizes $\alpha_k$ satisfy the Robbins-Munro sequence. For more details, the reader is referred to [134]. The GLIE assumption entails that (i) all state-action pairs are explored infinitely many times, and that, (ii) as time goes to infinity, the policy converges to a greedy policy. The second assumption (2) is not directly related to the exploration method and is most commonly satisfied in practice by selecting a small constant step size. Formally, we can ensure GLIE

**Algorithm 2:** Variance-based exploration

```
1  Input : Initial MPC parameters $\theta_0$, initial learning parameters $\tilde{\theta}_0$, initial state
     $s_0$, data set $\mathcal{D}$, batch update frequency $b$, learning step size $\alpha$, number of
     samples used to generate $V$ targets $p$, maximum number of samples in data set
     $M_{\max}$, parameter noise standard deviation $\sigma_p$, length of simulation $k_{\max}$ ;
2  Output : Policy $\pi_\theta$
3  while $k \leq k_{max}$ do
4      if $mod(k,b) = 0$ then
5          Update MPC parameters with learned parameters $\theta_k = \tilde{\theta}_k$
6          Evaluate gradient of IDW variance (7.22)
7          Ensure that the IDW variance gradient respects bound (7.23)
8          Perturb parameters to get $\bar{\theta}_k$ (7.24)
9      Act greedily w.r.t. current Q-estimate (7.25)
10     if $mod(k,b) \geq p$ then
11         if $|\mathcal{D}| \leq M_{max}$ or ($|\mathcal{D}| \geq M_{max}$ and (7.21)) then
12             Calculate $p$-step prediction of $V_k$ (7.20)
13             Add $\{V_k, s_{k-p}, \bar{\theta}_k\}$ to $\mathcal{D}$
14     Update $\tilde{\theta}_k$ (7.8)
15     $k \leftarrow k + 1$
```

for the proposed exploration method in Section (7.5.1). The first part of GLIE (i) is ensured by keeping a random term in (7.24) so that we ensure sufficient exploration even though the gradient of the variance eventually may converge to a small number. The second part of GLIE (ii) can be ensured by using a decaying scalar i.e. $\beta(\frac{1}{2}\nabla_{\theta_k}\hat{r}^2(\theta_k, s_k) + \frac{1}{2}\zeta_p)$ where $\beta = \beta_a \exp(-\omega k)$ and $\omega$ is a hyperparameter.

## 7.6 Simulation examples

We apply the proposed exploration method to two simulation examples, namely on an LQR problem and a cart pendulum system. The latter is a popular example in the control systems literature, as it is open-loop unstable and nonlinear. For each simulation example, we will benchmark our method with respect to both $(i)$ Gaussian action noise and $(ii)$ Gaussian parameter noise.

### 7.6.1 LQR

The following example is adapted from [22]. We consider a discrete linear system of the form

$$s_{k+1} = As_k + Ba_k, \tag{7.26}$$

with system matrices

$$A = \kappa \begin{bmatrix} \cos\beta & \sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}, \; B = \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix}, \tag{7.27}$$

where we use $\kappa = 0.95$, and $\beta = 22°$ [deg]. The baseline stage cost is selected as

$$L(s,a) = \frac{1}{2}\|s - s_{\text{ref}}\|^2 + \frac{1}{2}\|a - a_{\text{ref}}\|^2, \tag{7.28}$$

where $s_{\text{ref}} = [0.1, 0.1]^\top$, and the reference input is found accordingly. The (inaccurate) prediction model is defined as

$$A_0 = \kappa \begin{bmatrix} \cos\hat{\beta} & \sin\hat{\beta} \\ \sin\hat{\beta} & \cos\hat{\beta} \end{bmatrix}, \; B_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{7.29}$$

where $\hat{\beta} = 20°$. The parameterized MPC scheme reads as

$$\min_{x,u} V_0 + \gamma^N \|x_N - x_{\text{ref}}\|_P^2 + \sum_{j=0}^{N-1} \gamma^j \left\| \begin{bmatrix} x_j - x_{\text{ref}} \\ u_j - u_{\text{ref}} \end{bmatrix} \right\|^2 \tag{7.30a}$$

$$\text{s.t. } \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{7.30b}$$

$$x_{j+1} = A_0 x_j + B_0 u_j, \tag{7.30c}$$

(7.30d)

using a prediction horizon of $N = 10$, $\gamma = 0.99$ and $P$ is the solution to the discrete Riccati equation obtained using the inaccurate system dynamics in (7.29). The parameter vector is $\theta = \{x_{\text{ref},1}, x_{\text{ref},2}, u_{\text{ref},1}, u_{\text{ref},2}, V_0\}$. We consider a continuing task and simulate the system for a total of 5000 time steps. The parameters are updated in a batch manner, using a batch length of 200, i.e., we update the parameters in the MPC scheme every 200 time steps. A learning rate of $\alpha = 0.1$ was used. For each exploration method, we consider a range of noise distributions, defined by varying standard deviations. For brevity, only the best-performing ones are reported.

For the variance-based exploration method, we use $p = 10$, do not pose any restrictions on the data sampled, i.e., $M_{\text{max}} = 5000$ and use the weighting function in (7.15). The resulting IDW variance estimate (7.18) is plotted over learning batches in Figure 7.2. The system states and actions are plotted both during exploration and exploitation, i.e. we use MPC with the learned parameters, resulting from Gaussian action noise and variance-based exploration, to control the system, see Figure 7.3. We also plot the norm of the parameter updates resulting from the different exploration methods, to indicate when the algorithm converges. Additionally, we state the cost of exploration, i.e. the sum of cost over all time steps needed to see a convergence of the parameters, as well as the sum of cost over simulations in exploitation, see Table 7.1. The numbers reported in Table 7.1 are found for a total of 5 simulations run for each exploration method.

We see that the mean of the variance in Figure 7.2 is initially small but eventually grows. The peak around 5 batches, seems to result in a larger parameter update which can be spotted in the lower-right plot in Figure 7.3. As increasingly more parameter values are tried out in the simulation, and data is gathered, the variance estimate starts decreasing. From the resulting statistics in Table 7.1, we see that Gaussian action noise for this particular problem obtains the best performance, in terms of minimizing the cost during exploitation, but at a much higher cost during exploration than both random perturbations as well as variance-based perturbations in parameter space. Variance-based exploration in this case obtains a slightly higher cost in exploitation compared to Gaussian action noise, although the same as with Gaussian parameter noise, while being the cheapest alternative during exploration. In Figure 7.3, we see that the empirical standard deviation of the simulated states and actions are visibly larger for variance-based exploration than for Gaussian action noise in exploitation, however, we note that the resulting standard deviation in the accumulated cost is small in exploitation, see Table 7.1.

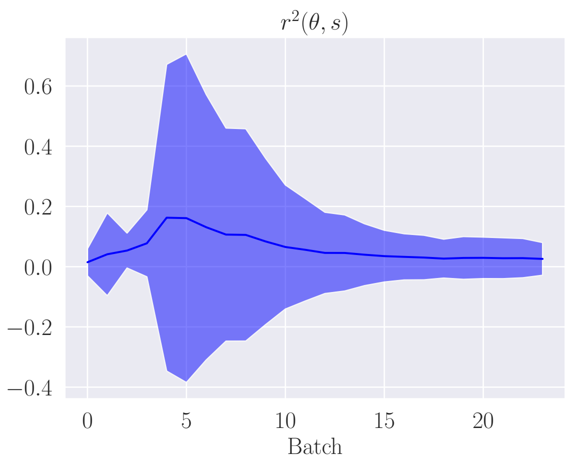


**Figure 7.2:** The mean and two standard deviations of the IDW variance estimate (7.18) over learning batches.

**Table 7.1:** Cost statistics for LQR simulations. The mean and standard deviation (in parentheses) are found for a total of 5 simulations for each exploration method.

| Exploration method | Exploration | | |
|---|---|---|---|
| | $\sigma$ | $k_{\max}$ | $\sum L(s,a)$ |
| Variance-based | 0.1 | 5000 | **28.74 (16.52)** |
| Gaussian noise in parameter space | 0.1 | 5000 | 57.66 (6.55) |
| Gaussian noise in action space | 0.1 | 5000 | 132.66 (1.721) |

| Exploration method | Exploitation | |
|---|---|---|
| | $k_{\max}$ | $\sum L(s,a)$ |
| Variance-based | 20 | 0.021 (0.00) |
| Gaussian noise in parameter space | 20 | 0.021 (0.00) |
| Gaussian noise in action space | 20 | **0.020 (0.00)** |

**Table 7.2:** Cart pendulum model parameters

| Description | Symbol | Value |
|---|---|---|
| Cart weight | $M$ | 2.4 kg |
| Pendulum weight | $m$ | 0.23 kg |
| Acceleration of gravity | $g$ | 9.81 $m/s^2$ |
| Pendulum length | $l$ | 0.36 m |

### 7.6.2 Cart pendulum

We consider the cart pendulum system as depicted in Figure 7.4. The dynamics of the cart pendulum, neglecting friction, are given by

$$(M+m)\ddot{z} + \frac{1}{2}ml\ddot{\phi}\cos\phi = \frac{1}{2}ml\dot{\phi}^2\sin\phi + \mathrm{u}, \tag{7.31a}$$

$$\frac{1}{3}ml^2\ddot{\phi} + \frac{1}{2}ml\ddot{z}\cos\phi = -\frac{1}{2}mgl\sin\phi, \tag{7.31b}$$

with model parameters as specified in Table 7.2 and where $u$ is the control input. The state vector is $s = [z, \dot{z}, \phi, \dot{\phi}]^\top$, consisting of the cart displacement and velocity along the horizontal axis, the angle between the pendulum and the vertical axis and angular velocity, respectively. The action is $a = u$. The dynamics in (7.31) are converted to a state space representation, discretized, and used to simulate the system. A linearized version of the state space representation is used as the prediction model in the MPC scheme. The state space representation and the linearized dynamics are found in e.g. [142].

A 4th-order Runge Kutta scheme is used to discretize the dynamics in (7.31), using the step size dt $= 0.1$ s. We consider the following constraint, in newtons, on the force acting on the cart

$$-7 \leq a \leq 7. \tag{7.32}$$

The RL cost is given as

$$L(s,a) = \begin{bmatrix} s - s_{\text{ref}} \\ a \end{bmatrix}^\top \begin{bmatrix} I_4 & 0 \\ 0 & 0.01 \end{bmatrix} \begin{bmatrix} s - s_{\text{ref}} \\ a \end{bmatrix}, \tag{7.33}$$

where $s_{\text{ref}} = [0.5, 0, 0, 0]^\top$. The linearized prediction model in the MPC scheme, defined by $\bar{A}$ and $\bar{B}$, is obtained from linearizing the system dynamics (7.31) at $\phi = 0$, corresponding to the pendulum being in an upright position. The parameterized MPC scheme reads as

$$\min_{x,u} V_0 + \gamma^N T_\theta(x_N) + \sum_{j=0}^{N-1} \gamma^j \ell_\theta(x_j, u_j) \tag{7.34a}$$

$$\text{s.t. } \forall j \in \mathbb{I}_{0:N-1} : x_0 = s, \tag{7.34b}$$

$$x_{j+1} = \bar{A}x_j + \bar{B}u_j, \tag{7.34c}$$

$$-7 \leq u_j \leq 7, \tag{7.34d}$$

where $N = 30$ and $\gamma = 0.99$. Moreover, we parameterize the stage cost using a quadratic function according to

$$\ell_\theta(x,u) = \begin{bmatrix} x - x_{\text{ref}} \\ u \end{bmatrix}^\top M(\theta) \begin{bmatrix} x - x_{\text{ref}} \\ u \end{bmatrix}, \tag{7.35}$$

where $M(\theta)$ is a positive definite matrix, with parameterized diagonal elements. We assume that we know all state references, except the cart position, i.e. $x_{\text{ref}} = [\theta_c, 0, 0, 0]$. A similar parameterization is also used for the terminal cost $T_\theta$. The resulting parameter vector consists of the elements in $\ell_\theta$ and $T_\theta$, as well as $V_0$ and $\theta_c$. We learn in an episodic manner, considering episodes of length 300, and let one episode correspond to one batch in terms of when we update the parameters. We learn for a total of 1000 batches and use a learning rate of $\alpha = 0.5$. We test all three exploration methods, for an interval of Gaussian distributions, defined by different standard deviations, and report the best-performing distribution in each category. For the variance-based exploration method, we use $p = 110$, $M_{\text{max}} = 5000$, and the weighting function in (7.15) .

As for the previous example, we have plotted the simulated states and actions during both exploration and exploitation, for Gaussian action noise and variance-based exploration, as well as the normed parameter updates, see Figure 7.5. Table 7.3 lists the sum of the cost for exploration and exploitation. The main goal of learning, in this case, is to obtain the true desired cart position, as well as tune the stage and terminal cost. We see from the plotted cart position during exploration, that variance-based exploration causes the system to visit positions closer to the true reference, than using Gaussian action noise. Here, this results in a small, but still visible improvement in both the plotted cart position in exploitation as well as the calculated cost in Table 7.3.

The statistics in Table 7.3 are found for a total of 3 simulations in each category. We see that Gaussian action noise in this case has very little effect, as the best performance in exploitation was obtained with $\sigma_a = 0.0001$. By using a Gaussian perturbation in parameter space we improve the performance in exploitation, while also reducing the cost of learning. Using variance-based exploration, we make exploration even cheaper while achieving a similar improvement in performance.

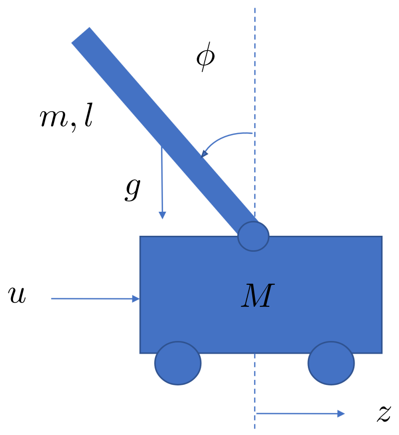


**Figure 7.4:** Cart pendulum system.

**Table 7.3:** Cost statistics for cart pendulum simulations. The mean and standard deviation (in parentheses) are found for a total of 3 simulations for each exploration method.

| Exploration method | Exploration | | |
|---|---|---|---|
| | $\sigma$ | $k_{\max}$ | $\sum L(s,a)$ |
| Variance-based | 0.1 | 300000 | **58071.39 (235.94)** |
| Gaussian noise in parameter space | 0.1 | 300000 | 60558.04 (155.20) |
| Gaussian noise in action space | 0.0001 | 300000 | 61586.95 (0.033) |

| Exploration method | Exploitation | |
|---|---|---|
| | $k_{\max}$ | $\sum L(s,a)$ |
| Variance-based | 100 | 47.86 (0.01) |
| Gaussian noise in parameter space | 100 | **47.46 (0.03)** |
| Gaussian noise in action space | 100 | 48.86 (0.00) |

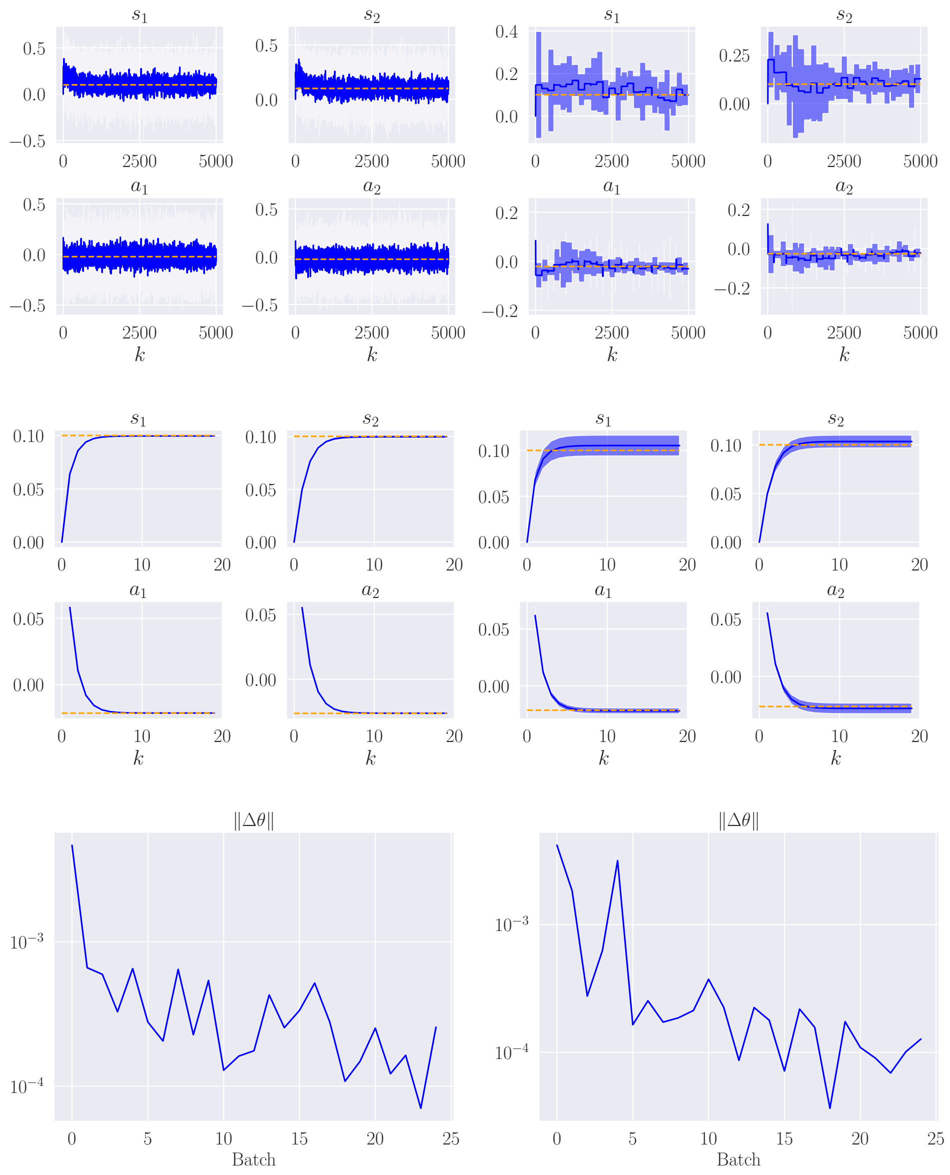


**Figure 7.3:** LQR simulation results. **Upper plots**: the mean and two standard deviations of states and actions during exploration, with Gaussian action noise (left) and variance-based exploration (right). **Middle plots**: the mean and two standard deviations of states and actions during exploitation, using parameters learned with Gaussian action exploration (left) and variance-based exploration (right). **Bottom plots**: the mean of parameter updates using Gaussian action noise (left) and variance-based exploration (right).

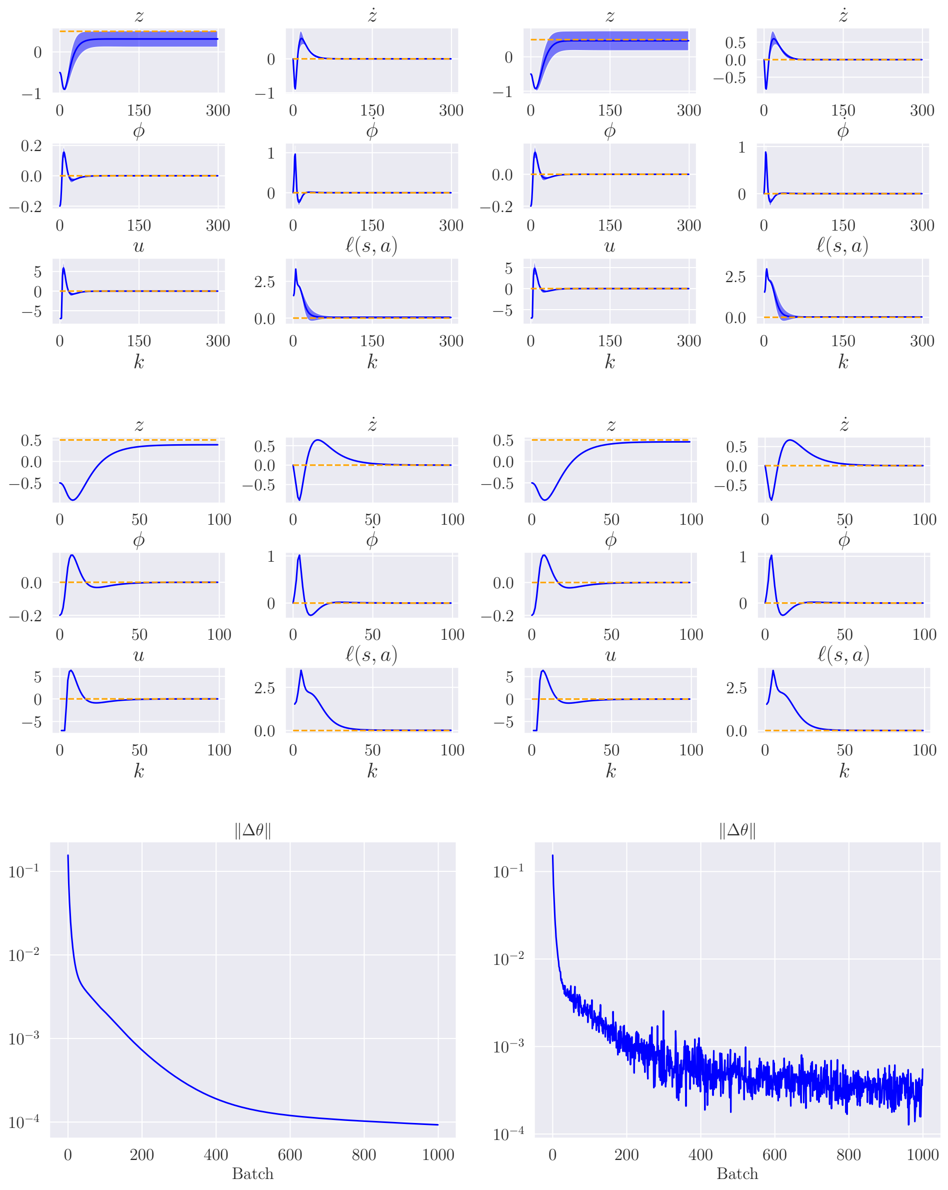


**Figure 7.5:** Cart pendulum simulation results. **Upper plots**: the mean and two standard deviations of states and actions during exploration, with Gaussian action noise (left) and variance-based exploration (right). **Middle plots**: the mean of states and actions during exploitation, using parameters learned with Gaussian action exploration (left) and variance-based exploration (right). **Bottom plots**: the mean of parameter updates using Gaussian action noise (left) and variance-based exploration (right).

## 7.7 Conclusion

We have presented a novel approach for variance-based exploration particularly suited for using an MPC scheme as a function approximator in RL. The method is based on IDW to build a variance estimate of the value function approximator, which is computationally cheap compared to probabilistic methods such as GPs and well-suited in an online setting. The proposed exploration method is tested in simulation and benchmarked against Gaussian perturbations in both action and parameter space. The results show that exploration in parameter space generally is cheaper than exploration in action space while achieving at least a similar performance in exploitation using the learned parameter values. This suggests that Gaussian exploration in parameter space, as already suggested for NNs as function approximators in RL, successfully can be used also with MPC. The simulation results also revealed that variance-based exploration in parameter space further reduces the cost of exploration, compared to Gaussian perturbations, with the same performance in exploitation. This means that exploration can be made even cheaper, with only a small increase in computational cost and with minor overall changes to the existing implementation.

# Chapter 8

# Conclusions and further work

This thesis has detailed contributions made on the topic of learning-based MPC for systems that may be hard to model accurately. A detailed conclusion is given at the end of each chapter. The following chapter presents a more general discussion and conclusion, as well as suggestions for future work.

## 8.1 Summary and discussion

In Part 1 of this thesis, we have considered two different approaches for combining MPC with supervised learning methods. Both of these approaches exploit data to learn the entire or parts of the dynamical model, which in turn is exploited in the MPC formulation. In Chapter 3, we looked into how NNs can be used in an autoregressive prediction model in the MPC. We answer research question **R.1** by showing that, for a system controlled with MPC using an NN as the prediction model, the closed-loop system is ISS w.r.t. the prediction error of the NN, under the core assumption of the prediction error being bounded.

In Chapter 4, we use learning only for the nonlinear component of what is known as a Lur'e system. By replacing the nonlinear component with a sector constraint, the optimal control problem can be formulated as a convex optimization problem, hence reducing computation time considerably and for which we are guaranteed that a global solution is obtained. By learning a stochastic sector we are able to improve the closed-loop performance of the controller and enlarge the feasible region of the controller compared to using a conservative sector estimate, thus answering research question **R.2**. The resulting closed-loop system comes with probabilistic stability guarantees.

As has been previously stated, the performance of the MPC scheme relies on sufficiently accurate models. When using data-driven models in the MPC scheme, the models are trained to reduce the prediction error as much as possible, hoping that this will yield the best closed-loop behavior. However, the modeling, in this case, is not directly connected to the closed-loop performance, and determining what is "sufficient accuracy" is therefore difficult to do. Moreover, it can be challenging to select model architectures when dealing with complex dynamical systems. Additionally, as MPC is a control algorithm based on solving an optimization problem at every iteration, introducing complex dynamical models in the MPC scheme is expected to increase the computation time needed to solve the optimization problem.

An important motivation for using MPC as a function approximator in RL is that the parameterization of the optimization problem, including the prediction model, can be adjusted to optimize the closed-loop behavior directly. As previously stated, parameterizing not only the prediction model but also the cost and constraints give additional flexibility for adjusting the policy in the face of model inaccuracies. This combination of MPC and RL was pursued in Part II of the thesis.

Parameterizing and adjusting the cost function in the MPC scheme is especially relevant for EMPC which aims to minimize an economic cost that is not necessarily positive definite. The generic cost function complicates the stability analysis of EMPC compared to the more standard tracking MPC scheme. By applying dissipativity theory, we can formulate and learn cost modifications that for certain types of problems allow us to capture the optimal economic policy using a provably stable MPC scheme. In Chapter 5 we propose convex NNs as cost modifications in order to estimate the economic policy using a stable MPC scheme based on an inaccurate prediction model.

The motivation for introducing NNs in the cost function is supported by [22], which states that an MPC scheme with a rich parameterization of the cost and constraints is able to capture the optimal policy even when based on an inaccurate model. Moreover, the motivation for selecting convex cost NNs, is that we can prove that these are lower bounded by $\mathcal{K}_\infty$-functions, which is needed in order to establish stability. Furthermore, convexity is in general a desirable property in optimization. For the selected simulation example in Chapter 5, we benchmarked the convex NN-based cost modifications with quadratic functions, and found little improvement in closed-loop performance. Thus, we were not able to provide an affirmative answer to research question **R.3**. The expected significance, in terms of improving the closed-loop performance, of adding richer cost modifications is arguably problem dependent. Also, by enforcing the convexity of the NNs we are imposing a stricter requirement than the stability condition itself.

In Chapter 5, we were faced with an economic problem for which Q-learning struggled to capture the optimal policy. In order to learn both the value function and the policy, we proposed a combination of RL methods using a null space projection. In Chapter 6, we build on this early result and propose a different combination based on a multi-objective approach. This naturally extends to a second-order method, that allows us to take full advantage of the MPC as a function approximator. Because the Hessian of the Q-learning objective is straightforward to obtain from sensitivity analysis of the MPC scheme, we were able to improve the learning rate of the second-order policy gradient step, which positively answers research question **R.4**.

Finally, we proposed a novel method for variance-based exploration for MPC in RL. This is the first work that proposes and tests exploration in parameter space for MPC, an approach that is currently adopted in deep RL to enhance exploration when learning the value function or policy using NNs. Using a variance-based exploration strategy, we achieved the same or better closed-loop performance of MPC-based policies at a significantly lower cost during learning, compared to using random perturbations in action space and parameter space. This is therefore an important step towards answering **R.5**.

## 8.2 Further work

Convex cost modifications were proposed to improve the closed-loop performance of MPC schemes based on inaccurate models. Restricting the cost modifications to be convex functions, is a stronger requirement than what we need to establish nominal stability, namely to show that the stage cost is lower bounded by a $\mathcal{K}_\infty$ function. However, convexity is used to establish stability by design, which is not necessarily easy to do for a generic cost modification, and also alleviates the computation of the optimization problem. An interesting open research question is therefore how rich the model parameterization ought to be in order to capture the optimal policy when using a convex cost function in the MPC scheme.

In this thesis, we proposed a method for variance-based exploration in RL. The method was formulated and verified for deterministic systems. A direction for future work is therefore extending and testing this method for stochastic systems. The exploration method is based on using IDW to estimate the uncertainty of the value function. Although IDW has been used successfully applied to data with noise, it is mainly considered a tool for noise-free data, and other methods may be considered when extending this exploration scheme to stochastic systems.

One benefit of combining supervised learning methods and MPC, as treated in Part I, is that there exist sophisticated tools for training e.g. NNs using e.g. GPUs. This makes training NNs on large data sets offline very efficient. For the combination of MPC and RL as used in Part II, we learn in an online setting. This allows us to update the controller as new data is available. MPC problems in general are cheap to differentiate, using sensitivity analysis, but typically expensive to solve. Due to the excellent tools that exist for solving MPC in real-time, performing RL online, i.e. while the system is being controlled, is feasible for many problems. However, when it comes to processing large amounts of data offline, the time needed to solve the optimization problems is impractical. The applied framework in Part II therefore suffers from one major drawback, namely the apparent difficulty to learn from already existing large data sets.

An early investigation of adopting MPC as a function approximator in what is known as an offline RL setting is done in [143]. This is a promising direction that alleviates the need for solving the optimization problem when learning MPC-based policies from large data sets.

As this thesis has considered only theoretical aspects of combining MPC and learning, the final suggestion for further work is testing the proposed methods on real systems.

# References

[1] S. Moe, A. M. Rustad and K. G. Hanssen, “Machine learning in control systems: An overview of the state of the art,” in *Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11–13, 2018, Proceedings 38*, Springer, 2018, pp. 250–265.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[4] P. Abbeel, A. Coates, M. Quigley and A. Ng, “An application of reinforcement learning to aerobatic helicopter flight,” *Advances in neural information processing systems*, vol. 19, 2006.

[5] S. Wang, W. Chaovalitwongse and R. Babuska, “Machine learning algorithms in bipedal robot control,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 5, pp. 728–743, 2012.

[6] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[7] L. Wells and T. Bednarz, “Explainable AI and Reinforcement Learning — A systematic review of current approaches and trends,” *Frontiers in artificial intelligence*, vol. 4, p. 550 030, 2021.

[8] H. K. Khalil, *Nonlinear control*. Pearson New York, 2015, vol. 406.

[9] B. Karg and S. Lucia, “Efficient representation and approximation of model predictive control laws via deep learning,” *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.

[10] L. Ljung, “Perspectives on system identification,” *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010.

[11] S. Sastry, M. Bodson and J. F. Bartram, *Adaptive control: stability, convergence, and robustness*. Dover Books on Electrical Engineering, 2011.

[12] D. Nguyen-Tuong and J. Peters, “Model learning for robot control: A survey,” *Cognitive processing*, vol. 12, pp. 319–340, 2011.

[13] G. Shi, X. Shi, M. O’Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue and S.-J. Chung, “Neural lander: Stable drone landing control using learned dynamics,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 9784–9790.

[14] A. Aswani, H. Gonzalez, S. S. Sastry and C. Tomlin, “Provably safe and robust learning-based model predictive control,” *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[15] P. W. Van De Ven, T. A. Johansen, A. J. Sørensen, C. Flanagan and D. Toal, “Neural network augmented identification of underwater vehicle models,” *Control Engineering Practice*, vol. 15, no. 6, pp. 715–725, 2007.

[16] T. Dierks and S. Jagannathan, “Neural network output feedback control of robot formations,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 2, pp. 383–399, 2010.

[17] L. Hewing, K. P. Wabersich, M. Menner and M. N. Zeilinger, “Learning-based model predictive control: Toward safe learning in control,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

[18] D. Limon, J. Calliess and J. M. Maciejowski, “Learning-based nonlinear model predictive control,” *IFAC-PapersOnLine (Proc. 20th IFAC World Congress)*, vol. 50, no. 1, pp. 7769–7776, 2017.

[19] M. Maiworm, D. Limon, J. M. Manzano and R. Findeisen, “Stability of Gaussian Process Learning Based Output Feedback Model Predictive Control,” *IFAC-PapersOnLine (Proc. 6th IFAC Conference on Nonlinear Model Predictive Control)*, vol. 51, no. 20, pp. 455–451, 2018.

[20] F. Berkenkamp, R. Moriconi, A. P. Schoellig and A. Krause, “Safe learning of regions of attraction for uncertain, nonlinear systems with Gaussian processes,” *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4661–4666, 2016.

[21] S. Gros and M. Zanon, “Data-driven economic NMPC using reinforcement learning,” *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2019.

[22] S. Gros and M. Zanon, “Learning for MPC with stability & safety guarantees,” *Automatica*, vol. 146, p. 110 598, 2022.

[23] K. Seel, E. I. Grøtli, S. Moe, J. T. Gravdahl and K. Y. Pettersen, “Neural network-based model predictive control with input-to-state stability,” in *2021 American Control Conference (ACC)*, IEEE, 2021, pp. 3556–3563.

[24] K. Seel, M. Haring, E. I. Grøtli, K. Y. Pettersen and J. T. Gravdahl, “Learning-based robust model predictive control for sector-bounded Lur’e systems,” *IFAC-PapersOnLine (Proc. 1st IFAC Modeling, Estimation and Control Conference (MECC))*, vol. 54, no. 20, pp. 46–52, 2021.

[25] K. Seel, A. B. Kordabad, S. Gros and J. T. Gravdahl, “Convex neural network-based cost modifications for learning model predictive control,” *IEEE Open Journal of Control Systems*, vol. 1, pp. 366–379, 2022.

[26] K. Seel, S. Gros and J. T. Gravdahl, “Combining Q-learning and deterministic policy gradient for learning-based model predictive control,” in *Accepted to 2023 62nd IEEE Conference on Decision and Control (CDC)*, 2023.

[27] K. Seel, A. Bemporad, S. Gros and J. T. Gravdahl, “Variance-based exploration for learning model predictive control,” *IEEE Access*, vol. 11, pp. 60 724–60 736, 2023.

[28] M. Haring, E. I. Grøtli, S. Riemer-Sørensen, K. Seel and K. G. Hanssen, “A Levenberg-Marquardt algorithm for sparse identification of dynamical systems,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[29] A. Anand, K. Seel, V. Gjærum, A. Håkansson, H. Robinson and A. Saad, “Safe learning for control using control Lyapunov functions and control barrier functions: A review,” *Procedia Computer Science*, vol. 192, pp. 3987–3997, 2021.

[30] A. Håkansson, A. Saad, A. Anand, V. Gjærum, H. Robinson and K. Seel, “Robust reasoning for autonomous cyber-physical systems in dynamic environments,” *Procedia Computer Science*, vol. 192, pp. 3966–3978, 2021.

[31] B. Foss and T. A. N. Heirung, “Merging optimization and control,” *Lecture Notes*, 2013.

[32] D. Bertsekas, *Dynamic programming and optimal control: Volume I*. Athena scientific, 2012, vol. 1.

[33] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Publishing, 2009.

[34] R. Amrit, J. B. Rawlings and D. Angeli, “Economic optimization using model predictive control with a terminal cost,” *Annual Reviews in Control*, vol. 35, no. 2, pp. 178–186, 2011.

[35] J. Jäschke, X. Yang and L. T. Biegler, “Fast economic model predictive control based on nlp-sensitivities,” *Journal of Process Control*, vol. 24, no. 8, pp. 1260–1272, 2014.

[36] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[37] D. Limon, T. Alamo, F. Salas and E. F. Camacho, “On the stability of constrained MPC without terminal constraint,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 832–836, 2006.

[38] A. Jadbabaie and J. Hauser, “On the stability of receding horizon control with a general terminal cost,” *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 674–678, 2005.

[39] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[40] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Advances in neural information processing systems*, vol. 12, 1999.

[41] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, “Deterministic policy gradient algorithms,” *31st International Conference on Machine Learning, ICML 2014*, vol. 1, pp. 605–619, 2014.

[42] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *Journal of Machine Learning Research*, vol. 4, no. 6, pp. 1107–1149, 2004.

[43] E. C. Kerrigan and J. M. Maciejowski, “Soft constraints and exact penalty functions in model predictive control,” *Proceedings of the UKACC International Conference on Control*, 2000.

[44] A. B. Kordabad, M. Zanon and S. Gros, “Equivalence of Optimality Criteria for Markov Decision Process and Model Predictive Control,” *IEEE Transactions on Automatic Control*, pp. 1–8, 2023.

[45] K. Hornik, M. Stinchcombe and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[46] A. B. Kordabad and S. Gros, “Q-learning of the storage function in economic nonlinear model predictive control,” *Engineering Applications of Artificial Intelligence*, vol. 116, p. 105 343, 2022.

[47] M. Zanon and S. Gros, “A new dissipativity condition for asymptotic stability of discounted economic MPC,” *Automatica*, vol. 141, p. 110 287, 2022.

[48] L. Ljung, *System Identification (2nd Ed.): Theory for the User*. USA: Prentice Hall PTR, 1999.

[49] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice Hall Upper Saddle River, NJ, 1998, vol. 104.

[50] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2.

[51] U. Rosolia and F. Borrelli, “Learning model predictive control for iterative tasks. A data-driven control framework,” *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[52] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[53] C. J. Ostafew, A. P. Schoellig and T. D. Barfoot, “Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments,” *IEEE International Conference on Robotics and Automation*, pp. 4029–4036, 2014.

[54] C. D. McKinnon and A. P. Schoellig, “Learning probabilistic models for safe predictive control in unknown environments,” in *2019 18th European Control Conference (ECC)*, IEEE, 2019, pp. 2472–2479.

[55] L. Hewing, A. Liniger and M. N. Zeilinger, “Cautious NMPC with Gaussian process dynamics for autonomous miniature race cars,” *European Control Conference*, pp. 1341–1348, 2018.

[56] I. Goodfellow, Y. Bengio and A. Courtville, *Deep Learning*. MIT Press, 2016, pp. 307–342.

[57] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[58] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," *IEEE International Conference on Robotics and Automation*, pp. 3223–3230, 2015.

[59] N. Mohajerin and S. L. Waslander, "Multistep prediction of dynamic systems with recurrent neural networks," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3370–3383, 2019.

[60] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," *IEEE 55th Conference on Decision and Control*, pp. 4653–4660, 2016.

[61] D. L. Yu and J. B. Gomm, "Implementation of neural network predictive control to a multivariable chemical reactor," *Control Engineering Practice*, vol. 11, no. 11, pp. 1315–1323, 2003.

[62] A. Wurzinger, H. Leibinger, S. Jakubek and M. Kozek, "Data driven modeling and nonlinear model predictive control design for a rotary cement kiln," *IFAC-PapersOnLine (Proc. 11th Symposium on Nonlinear Control Systems)*, vol. 52, no. 16, pp. 759–764, 2019.

[63] N. Lanzetti, Y. Z. Lian, A. Cortinovis, L. Dominguez, M. Mercangoz and C. Jones, "Recurrent neural network based MPC for process industries," *European Control Conference*, pp. 1005–1010, 2019.

[64] P. Kittisupakorn, P. Thitiyasook, M. A. Hussain and W. Daosud, "Neural network based model predictive control for a steel pickling process," *Journal of Process Control*, vol. 19, no. 4, pp. 579–590, 2009.

[65] H. G. Han, X. L. Wu and J. F. Qiao, "Real-time model predictive control using a self-organizing neural network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 9, pp. 1425–1436, 2013.

[66] J. M. Manzano, D. Limon, D. M. De la Peña and J. P. Calliess, "Output feedback MPC based on smoothed projected kinky inference," *IET Control Theory and Applications*, vol. 13, no. 6, pp. 795–805, 2019.

[67] Z. Wu, A. Tran, D. Rincon and P. D. Christofides, "Machine learning-based predictive control of nonlinear processes. Part I: Theory," *AIChE Journal*, vol. 65, no. 11, 2019.

[68] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[69] G. Cybenko, "Approximation by superposition of a sigmoidal function," *Mathmatics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[70] J. M. P. Menezes and G. A. Barreto, "Long-term time series prediction with the NARX network: An empirical evaluation," *Neurocomputing*, vol. 71, no. 16-18, pp. 3335–3343, 2008.

[71] L. Wang, T. Chai and L. Zhai, "Neural-network-based terminal sliding-mode control of robotic manipulators including actuator dynamics," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 9, pp. 3296–3304, 2009.

[72] T. Yang, N. Sun, H. Chen and Y. Fang, "Neural Network-Based Adaptive Antiswing Control of an Underactuated Ship-Mounted Crane with Roll Motions and Input Dead Zones," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 901–914, 2020.

[73] D. Limon, T. Alamo, D. M. Raimondo, D. M. De La Peña, J. M. Bravo, A. Ferramosca and E. F. Camacho, *Input-to-state stability: a unifying framework for robust model predictive control*. Springer, 2009, vol. 384, pp. 1–26.

[74] D. E. Seborg, T. F. Edgar and D. A. Mellichamp, *Process dynamics and control*, English. New York: Wiley, 1989.

[75] M. V. Kothare, V. Balakrishnan and M. Morari, "Robust constrained model predictive control using linear matrix inequalities," *Automatica*, vol. 32, no. 10, pp. 1361–1379, 1996.

[76] C. Böhm, R. Findeisen and F. Allgöwer, "Robust control of constrained sector bounded Lur'e systems with applications to nonlinear model predictive control," *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*, vol. 17, no. 6, pp. 935–958, 2010.

[77] C. Böhm, S. Yu, R. Findeisen and F. Allgöwer, "Predictive control for Lure systems subject to constraints using LMIs," *European Control Conference*, pp. 3389–3394, 2009.

[78] H. H. Nguyen, A. Savchenko, S. Yu and R. Findeisen, "Improved Robust Predictive Control for Lur'e Systems Using Set-based Learning," *IFAC-PapersOnLine (Proc. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC)*, vol. 51, no. 20, pp. 487–492, 2018.

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

[80] C. D. McKinnon and A. P. Schoellig, "Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2180–2187, 2019.

[81] L. Hewing, J. Kabzan and M. N. Zeilinger, "Cautious model predictive control using Gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[82] R. Soloperto, M. A. Müller, S. Trimpe and F. Allgöwer, "Learning-Based Robust Model Predictive Control with State-Dependent Uncertainty," *IFAC-PapersOnLine (Proc. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC)*, vol. 51, no. 20, pp. 442–447, 2018.

[83] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[84] S. Boyd, L. El Ghaoui, E. Feron and V. Balakrishnan, *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[85] M. Andersen, J. Dahl and L. Vandenberghe, "CVXOPT: Convex optimization," *Astrophysics Source Code Library*, [Online] Available: `https://cvxopt.org`, 2020.

[86] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[87] K. P. Wabersich and M. N. Zeilinger, "Scalable synthesis of safety certificates from data with application to learning-based control," in *2018 European Control Conference (ECC)*, IEEE, 2018, pp. 1691–1697.

[88] K. P. Wabersich, L. Hewing, A. Carron and M. N. Zeilinger, "Probabilistic model predictive safety certification for learning-based control," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 176–188, 2021.

[89] A. Marco, P. Hennig, J. Bohg, S. Schaal and S. Trimpe, "Automatic LQR tuning based on Gaussian process global optimization," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 270–277.

[90] S. Bansal, R. Calandra, T. Xiao, S. Levine and C. J. Tomlin, "Goal-driven dynamics learning via Bayesian optimization," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 5168–5173.

[91] A. Agrawal, S. Barratt, S. Boyd and B. Stellato, "Learning convex optimization control policies," in *Learning for Dynamics and Control*, PMLR, 2020, pp. 361–373.

[92] M. Menner and M. N. Zeilinger, "Convex formulations and algebraic solutions for linear quadratic inverse optimal control problems," in *2018 European control conference (ECC)*, IEEE, 2018, pp. 2107–2112.

[93] A. Aswani, Z.-J. Shen and A. Siddiq, “Inverse optimization with noisy data,” *Operations Research*, vol. 66, no. 3, pp. 870–892, 2018.

[94] M. Diehl, R. Amrit and J. B. Rawlings, “A Lyapunov function for economic optimizing model predictive control,” *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 703–707, 2010.

[95] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings and M. Diehl, “CasADi: A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.

[96] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[97] M. Zanon and S. Gros, “Safe reinforcement learning using robust MPC,” *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2020.

[98] S. Gros and M. Zanon, “Bias correction in reinforcement learning via the deterministic policy gradient method for MPC-based policies,” in *2021 American Control Conference (ACC)*, IEEE, 2021, pp. 2543–2548.

[99] S. Gros, M. Zanon and A. Bemporad, “Safe reinforcement learning via projection on a safe set: How to achieve optimality?” *IFAC-PapersOnLine (Proc. 21st IFAC World Congress)*, vol. 53, no. 2, pp. 8076–8081, 2020.

[100] W. Cai, A. B. Kordabad, H. N. Esfahani, A. M. Lekkas and S. Gros, “MPC-based reinforcement learning for a simplified freight mission of autonomous surface vehicles,” in *2021 60th IEEE Conference on Decision and Control (CDC)*, IEEE, 2021, pp. 2990–2995.

[101] A. B. Martinsen, A. M. Lekkas and S. Gros, “Combining system identification with reinforcement learning-based MPC,” *IFAC-PapersOnLine (Proc. 21st IFAC World Congress)*, vol. 53, no. 2, pp. 8130–8135, 2020.

[102] D. Angeli, R. Amrit and J. B. Rawlings, “On average performance and stability of economic model predictive control,” *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1615–1626, 2012.

[103] A. B. Kordabad and S. Gros, “Verification of dissipativity and evaluation of storage function in economic nonlinear MPC using Q-learning,” *IFAC-PapersOnLine (Proc. 7th IFAC Conference on Nonlinear Model Predictive Control)*, vol. 54, no. 6, pp. 308–313, 2021.

[104] M. Zanon, S. Gros and M. Diehl, “A tracking MPC formulation that is locally equivalent to economic mpc,” *Journal of Process Control*, vol. 45, pp. 30–42, 2016.

[105] B. Chachuat, B. Srinivasan and D. Bonvin, “Adaptation strategies for real-time optimization,” *Computers & Chemical Engineering*, vol. 33, no. 10, pp. 1557–1567, 2009.

[106] B. Amos, L. Xu and J. Z. Kolter, “Input convex neural networks,” in *International Conference on Machine Learning*, PMLR, 2017, pp. 146–155.

[107] Y. Chen, Y. Shi and B. Zhang, “Optimal control via neural networks: A convex approach,” *31st International Conference on Machine Learning, ICML 2014*, 2018.

[108] G. C. Calafiore, S. Gaubert and C. Possieri, “Log-sum-exp neural networks and posynomial models for convex and log-log-convex data,” *IEEE Transactions on neural networks and learning systems*, vol. 31, no. 3, pp. 827–838, 2019.

[109] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[110] S. Pirkelmann, D. Angeli and L. Grüne, “Approximate computation of storage functions for discrete-time systems using sum-of-squares techniques,” *IFAC-PapersOnLine (Proc. 11th IFAC Symposium on Nonlinear Control Systems (NOLCOS))*, vol. 52, no. 16, pp. 508–513, 2019.

[111] S. M. Kakade, “A natural policy gradient,” *Advances in neural information processing systems*, vol. 14, pp. 1531–1538, 2001.

[112] A. B. Kordabad, H. N. Esfahani, W. Cai and S. Gros, “Quasi-Newton iteration in deterministic policy gradient,” in *2022 American Control Conference (ACC)*, IEEE, 2022, pp. 2124–2129.

[113] F. Chollet *et al.*, “Keras: The python deep learning library,” *Astrophysics source code library*, [Online] Available: `https://github.com/fchollet/keras`, 2018.

[114] X. Li, L. Zhang, M. Nakaya and A. Takenaka, “Application of economic MPC to a CSTR process,” in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, IEEE, 2016, pp. 685–690.

[115] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, 2014.

[116] J. Sola and J. Sevilla, “Importance of input data normalization for the application of neural networks to complex industrial problems,” *IEEE Transactions on nuclear science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[117] B. O'Donoghue, R. Munos, K. Kavukcuoglu and V. Mnih, "Combining policy gradient and Q-learning," *International Conference on Learning Representations*, 2017.

[118] J. A. Boyan, "Least-squares temporal difference learning," in *International Conference on Machine Learning*, 1999, pp. 49–56.

[119] T. Furmston, G. Lever and D. Barber, "Approximate Newton methods for policy search in Markov decision processes," *Journal of Machine Learning Research*, vol. 17, 2016.

[120] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[121] Y. Yuan, "A review of trust region algorithms for optimization," in *International Council for Industrial and Applied Mathematics*, vol. 99, 2000, pp. 271–282.

[122] N. Cesa-Bianchi, C. Gentile, G. Lugosi and G. Neu, "Boltzmann exploration done right," *Advances in neural information processing systems*, vol. 30, 2017.

[123] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015, pp. 1889–1897.

[124] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel and M. Andrychowicz, "Parameter space noise for exploration," *International Conference on Learning Representations*, 2018.

[125] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine learning*, vol. 49, pp. 209–232, 2002.

[126] P. Auer, N. Cesa-Bianchi and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[127] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.

[128] I. Osband, B. Van Roy and Z. Wen, "Generalization and exploration via randomized value functions," in *International Conference on Machine Learning*, PMLR, 2016, pp. 2377–2386.

[129] I. Osband, C. Blundell, A. Pritzel and B. Van Roy, "Deep exploration via bootstrapped DQN," *Advances in neural information processing systems*, vol. 29, 2016.

[130] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell and S. Legg, "Noisy networks for exploration," in *International Conference on Learning Representations*, 2018.

[131] R. Y. Chen, J. Schulman, P. Abbeel and S. Sidor, "UCB and infogain exploration via Q-ensembles," *arXiv preprint arXiv: 1706.01502*, vol. 9, 2017.

[132] N. Nikolov, J. Kirschner, F. Berkenkamp and A. Krause, "Information-directed exploration for deep reinforcement learning," *International Conference on Learning Representations*, 2019.

[133] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov and I. Mordatch, "Plan online, learn offline: Efficient learning and exploration via model-based control," *International Conference on Machine Learning*, 2018.

[134] J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," *Machine learning*, vol. 16, no. 3, pp. 185–202, 1994.

[135] H.-M. Gutmann, "A radial basis function method for global optimization," *Journal of global optimization*, vol. 19, no. 3, pp. 201–227, 2001.

[136] A. Bemporad, "Global optimization via inverse distance weighting and radial basis functions," *Computational Optimization and Applications*, vol. 77, no. 2, pp. 571–595, 2020.

[137] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*, 1968, pp. 517–524.

[138] V. R. Joseph and L. Kang, "Regression-based inverse distance weighting with applications to computer experiments," *Technometrics*, vol. 53, no. 3, pp. 254–265, 2011.

[139] A. Bemporad, "Global optimization via inverse distance weighting and radial basis functions," *Computational Optimization and Applications*, vol. 77, no. 2, pp. 571–595, 2020.

[140] A. Bemporad, "Active learning for regression by inverse distance weighting," *Information Sciences*, 2023.

[141] J. Schulman, P. Moritz, S. Levine, M. I. Jordan and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *International Conference on Learning Representations*, vol. abs/1506.02438, 2015.

[142] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.

[143] S. Sawant, A. S. Anand, D. Reinhardt and S. Gros, “Learning-based MPC from big data using reinforcement learning,” *arXiv preprint arXiv:2301.01667*, 2023.

ISBN 978-82-326-7222-6 (printed ver.)
ISBN 978-82-326-7221-9 (electronic ver.)
ISSN 1503-8181 (printed ver.)
ISSN 2703-8084 (online ver.)

NTNU

Norwegian University of
Science and Technology