Risk-aware and Robust Approaches for Machine Learning-supported Model Predictive Control for Iterative Processes

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Bruno Morabito aus Reggio Calabria
Tag der Einreichung: 27.07.2023, Tag der Prüfung: 20.12.2023

1. Gutachten: Prof. Dr.-Ing. Rolf Findeisen
2. Gutachten: Prof. Dr.-Ing. Sergio Lucia
Darmstadt, Technische Universität Darmstadt, 2023

TECHNISCHE UNIVERSITÄT DARMSTADT

Electrical Engineering and Information Technology Department

Institute of Automatic Control and Mechatronics

Control and Cyber-Physical Systems

To my aunt Brunella

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 27.07.2023

Bruno Morabito

Acknowledgements

As I present this Ph.D. thesis, I am reminded of the numerous individuals who have contributed to this journey, each playing a vital role.

First off, I would like to express my sincere appreciation to Prof. Rolf Findensen for granting me the opportunity to embark on this project.

I am very gratefull to all my colleagues, especially to my closest fellow academic travelers: Johannes Pohlodek, Sebastián Espinel Ríos, Rudolph Kok, Hoang Hai Nguyen, Petar Andonov, Mohammed Soliman and Mohammed Ibrahim. We shared more than just research ideas and deadlines. From brainstorming sessions and arabic language classes, you made sure this journey was anything but dull. Our combined intellectual horsepower made for some memorable meetings and even more memorable pub gatherings.

I am also grateful to the international Max Planck Research School for its supportive and dynamic academic environment.

On a personal note, my deepest gratitude goes to my family, whose unwavering support and encouragement have been a constant source of strength. To my girlfriend, Izel Avcı, I owe special thanks for her understanding and unwavering support.

And to my friends, particularly the Forró community of Magdeburg, who proved that Ph.D. students can indeed dance – thank you for the much-needed breaks.

In conclusion, this thesis is a culmination of not just my efforts but also the collective support and encouragement of all those mentioned and many others who have been part of this journey. I am deeply grateful for their contributions to my academic and personal adventure.

Bruno Morabito
29.12.2023

Zusammenfassung

Die jüngsten Fortschritte im Bereich des maschinellen Lernens haben ein neues Interesse an der lerngestützten prädiktiven Regelung geweckt. Maschinellen Lernens verspricht, die Modellierung zu erleichtern und die Leistung der Prozesse zu verbessern. Allerdings bringen diese Ansätze auch einige Herausforderungen mit sich: Beispielsweise geht die Verbindung zu den physikalischen Gesetzen (teilweise) verloren, so dass maschinelle Lernmodelle äußerst ungenaue Ergebnisse liefern können. Es ist daher notwendig, Steuerungs- und Regelungsmethoden bereitzustellen, die die Modellunsicherheit dieser Modelle berücksichtigen. Bei iterativen Prozessen - also Prozessen, die nicht in einem stationären Zustand arbeiten - sind Unsicherheiten aufgrund der großen Änderungen der Prozessbedingungen während des Betriebs noch wichtiger. In dieser Arbeit werden zwei Methoden zur datengetriebenen Unsicherheitsmodellierung vorgeschlagen. Die erste Methode verwendet Gauß-Prozesse zum Lernen der Modellunsicherheit und neuronale Netze zum Lernen des Nominalmodells. Sie bietet eine einfache Möglichkeit, die Unsicherheit des Modells in einem einzigen Parameter zusammenzufassen, der von einem modellprädiktiven Regler verwendet werden kann, um risikobewusste Entscheidungen zu treffen. Diese Methode ist zwar einfach, garantiert aber nicht die Einhaltung von Beschränkungen. Die zweite Methode basiert auf tube-based modellprädiktiver Regelung und kann die Sicherstellung von Beschränkungen garantieren. Sie basiert auf dem Konzept der "sicheren Menge": Eine Menge, in der eine tube-based MPC eine zulässige Lösung hat. Wir zeigen, dass sich die sichere Menge unter bestimmten Annahmen bei jeder Iteration des Prozesses vergrößert, was eine Leistungssteigerung ermöglichen kann. Schließlich wird die neuartige Python-Bibliothek HILO-MPC vorgestellt, welche die auf maschinellem Lernen basierende modellprädiktive Regelung umsetzt. Diese Bibliothek verfügt über Schnittstellen zu TensorFlow und PyTorch und bietet leicht zugängliche Werkzeuge zur Definition von Regelungs- und Schätzungsproblemen unter Verwendung von maschinellen Lernmodellen.

Abstract

The recent advances in machine learning have catalyzed a renewed interest in machine-learning-supported model predictive control. Machine learning promises to facilitate modeling and improve the process' performance. Nevertheless, it brings some challenges: For instance, as the connection with physics law is (partially) lost, machine learning models can provide wildly inaccurate results. It is therefore necessary to provide control methods that take the model uncertainty of these models into account. Uncertainties are even more important for iterative processes - processes that do not operate at a steady state - due to the large changes in the process conditions during operation.

In this work, two methods for data-driven uncertainty modelling are proposed. The first method uses Gaussian processes to learn the model uncertainty and neural networks to learn the nominal model. It provides an simple way to summarize the uncertainty of the model into a single parameter, which can be used by a model predictive controller to make risk-aware decisions. This method, while being simple, does not guarantee constraint satisfaction. The second method is based on tube-based model predictive control and can guarantee constraint satisfaction. It is based on the concept of the ”safe set”: a set where a tube-based MPC has a feasible solution. We show that, under some assumptions, the safe set enlarges at every iteration of the process, potentially allowing increased performance. Finally, a novel Python library for machine-learning-based model predictive control, called HILO-MPC, is presented. This library interfaces with TensorFlow and PyTorch and provides easily-accesible tools for defining control and estimation problem using machine learning model.

Contents

1. Motivation 1
2. Background: predictive control and machine learning 1
3. Contribution 3
4. Structure of the thesis 3
5. Introduction 4
6. Machine learning models 5
    6.1. Artificial Neural Networks 5
    6.2. Gaussian processes 6
7. Hybrid models 7
    7.1. Example: Hybrid model of a fed-batch bioreactor 8
8. Training hybrid models 8
9. Summary 9
10. Introduction 9
11. Nominal Model Predictive Control 10
    11.1. Continuous processes 10
    11.2. Finite-time processes 12
12. Theoretical differences between continuous and finite-time processes . . . 13
13. Model Predictive Control under uncertainties 14
14. Summary 15
15. Introduction 15
    15.1. learning supported Model Predictive Control 16
    15.2. Notes on active and passive learning 16
16. Run-to-run learning & control 17
17. Safe learning & Risk-aware learning 18
18. Summary 19
19. Introduction 20
20. Problem setup 21
    20.1. Open-loop optimization 21
    20.2. Shrinking horizon MPC 21
21. Risk function 21

22. Examples . . . 23
22.1. Key Performance Indicators . . . 24
22.2. Case Study I: Production β-galactosidase . . . 24
22.3. Case Study II: Production of poly-β-hydroxybutyrate . . . 26
23. Summary . . . 38
24. Introduction . . . 38
25. Problem setup . . . 39
25.1. Robust control invariant set . . . 42
25.2. Tube-based shrinking horizon MPC . . . 44
25.3. Intra-run recursive feasibility . . . 44
26. Safe sets . . . 45
26.1. Computing an upper bound on the modeling error . . . 45
26.2. Safety conditions . . . 46
26.3. Guess of the first safe set . . . 48
26.4. Genesis experiments . . . 50
27. Guaranteed run-to-run expansion of the safe set . . . 50
27.1. Set-point error . . . 51
27.2. Design of experiment . . . 53
27.3. Example . . . 53
28. Accelerated run-to-run expansion of the safe set . . . 56
28.1. Example . . . 56
29. Example: Robotic arm . . . 56
30. Summary . . . 56
31. Introduction . . . 60
32. HILO-MPC modules . . . 60
33. Modeling module . . . 62
34. Machine Learning module . . . 64
34.1. Artificial Neural Networks . . . 64
34.2. Gaussian Processes . . . 65
35. Control module . . . 65
35.1. Model Predictive Control . . . 65
36. Observer module . . . 69
36.1. Moving Horizon Estimation . . . 69
36.2. Kalman filters . . . 70
36.3. Particle filter . . . 71
37. Embedded Module . . . 71
38. Examples . . . 72
38.1. Learning the dynamics - Race car . . . 73

38.2. Learning a reference - Cooperative robots 74
38.3. Learning the controller - String damper system 76
39. Summary 76
A. Notation and definitions 78
B. Theorems and Proofs 78
B.1. Proof of Theorem 0.2 78
B.2. Proof of Lemma 0.1 79
C. Miscellaneous 79
C.1. Computation of λ for robust control invariant set 79

1. Introduction

1.1. Motivation

A process is called iterative (or repetitive) if, to achieve the desired end result, the same process must be repeated in time. Such processes are widespread in chemical and biochemical engineering (consider, for example, fed-batch/batch reactors and extraction processes) and in manufacturing (e.g., robots used for manufacturing or storage in warehouses). Iterative processes, as depicted in Fig.2, typically embody three distinctive stages: A preparation phase, in which the necessary actions are taken to prepare the process; a productive stage during which the process aims to fulfill its designated objectives, and a final stage, which is tasked with terminating the process and restoring its initial state prior to the preparatory phase. The final and preparation phases can overlap. To ensure continuous production, these phases are repeated. We will refer to a repetition equivalently as *iteration*, *run*, or *batch*. The repeated nature of such processes introduces challenges that continuous processes do not have. The dynamics can change significantly at every run due to changes in the initial conditions and/or the environment. Furthermore, typically, iterative processes do not reach a steady state, and the system states vary over a wide range. Hence, the effect of nonlinear dynamics is more important than continuous processes [187, 178]. Often, innovative processes are iterative [187] (e.g., pilot plants in chemical engineering) or complex biological processes (e.g., in the biopharmaceutical industry). Hence, their dynamics are not always completely understood, the models might not exist or have limited predictive power, and the few available measurements are not sufficient to identify their dynamics with adequate accuracy [177, 28]. For these reasons, these processes are often operated with suboptimal control policies based on trial and error experiments at smaller scales that attempt to infer the sequence of control actions that need to be applied to reach acceptable results [183].

This is a substantial limitation that decreases the efficiency of such processes. Due to the market's fast-changing demands and increased efficiency requirements dictated, for example, by environmental sustainability regulations, tasks have to be performed increasingly faster and use fewer resources. Advanced control strategies can meet these needs. Nevertheless, to gain the trust of the manufacturers, these control strategies must

Figure 1.1.: Examples of systems performing repetitive tasks. On the left is a pilot-scale bioreactor for pharmaceutical production; on the right is a robotic arm used for packaging.

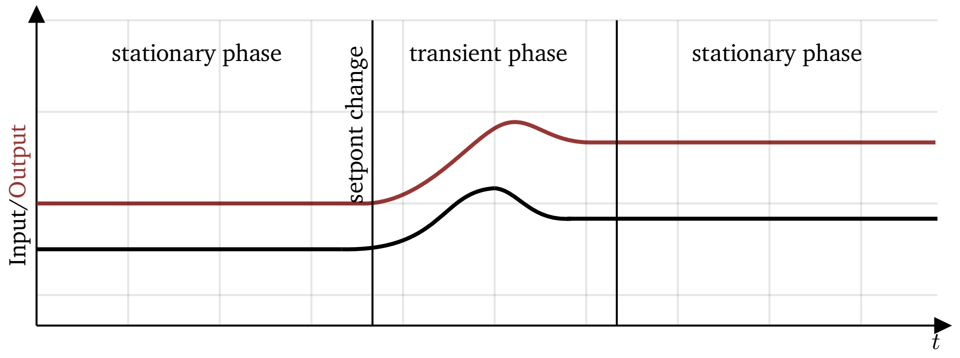
guarantee the safety of operation and process specifications dictated by regulators (e.g., the maximum content of pollutant in a given product) or by the process designers (e.g., maximum pH in a bioreactor) [182].

Advanced control strategies, such as Model Predictive Control (MPC), are key to achieving these demands and ensuring optimal performance [182]. Nevertheless, the lack of accurate prediction models, scarce measurements, and constraints present a challenge to applying MPC to these processes. The main motivation of this thesis is to provide methods that inform the MPC regarding the model uncertainty by using the (possibly small) datasets available directly in the control design.

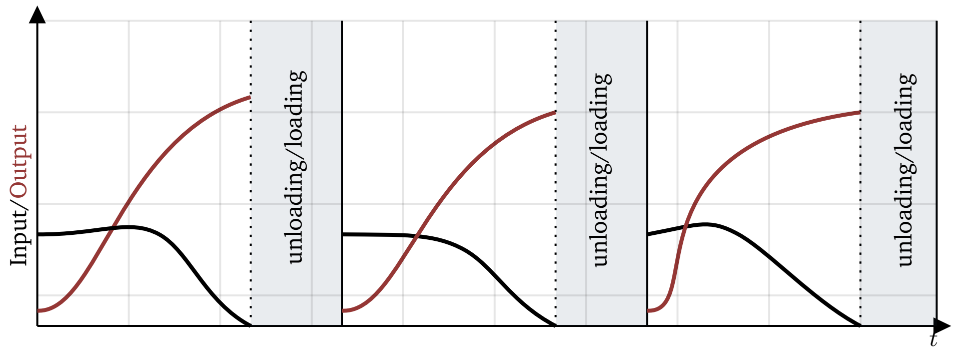
1.2. Background: predictive control and machine learning

MPC is an advanced model-based control strategy that repeatedly solves an optimal control problem. This method has been applied successfully in the industry for over 60 years, thanks to its ability to consider states and input constraints, and multi-input, multi-output systems. Furthermore, robustness and recursive feasibility guarantees can be given for some cases. Initially, MPC found applications in chemical engineering. Then it has been applied in many other fields, such as autonomous vehicles [95, 12], robotics [169], energy management [212, 170, 208], agriculture [47] and many more.

MPC is based on three “ingredients”: A model of the plant, an objective function, and constraints. The most critical part of the formulation is the model of the system. The model must describe the system with sufficient accuracy while being simple enough to ensure that the problem can be solved within the chosen sampling time. The quality of the model



(a) Continuous process



(b) (Fed)batch process

Figure 1.2.: Illustration of the two typical operating modes of manufacturing processes.

is even more critical for repetitive processes. Compared to continuous processes, where states are stabilized in a region close to a set point, the states of repetitive processes can change considerably within one repetition [28]. Hence, it has to be guaranteed that the model is valid for the entire range of realizations and variations of the state. First-principle models can be formulated for well-understood systems: These are built using physics laws, for example, mass or energy balances. But for complex systems, building an accurate first-principle model usable in an optimization-based control strategy is difficult or even impossible.

Thanks to the advances in computing power, deep learning, and data acquisition tools, in recent years, machine learning has been going through a *renaissance* period, not only in the field of computer science [22, 171]. The potential of these technologies has also been understood by researchers from other areas like control and automation, generating a rapid increase in publications with machine learning applications in control engineering (cf. [74] and references therein). Machine learning seems to be one possible solution to overcome the limitation of first-principle models [138, 193].

Nevertheless, there is a fundamental difference between computer science and process engineering that often does not allow to completely substitute first-principle models with machine learning models: In process engineering, data is usually scarce, expensive, and noisy [48]. Hence, methods to decrease the necessary data for the training should be explored. Hybrid models that use both first-principle and machine learning meet this need since they use the (somewhat limited) information that first-principle models give and augment that information with data using one or more machine learning approaches [193]. Nevertheless, despite all the efforts to improve the model's accuracy, it is practically impossible to have a perfect plant model; hence, advanced control strategies must deal with uncertainties. We will call *risk-aware* controllers control methods that consider uncertainty (e.g., in the constraints) but do not necessarily guarantee stability or constraint satisfaction. Instead, methods that guarantee stability and constraint satisfaction, which consider set-membership-type uncertainties, are commonly known as robust controllers.

The problem of robust control is old, dating back to the 1970s, when control engineers, especially in aerospace, realized the urgent need for control methods and stability concepts that take uncertainties into account [167]. Since then, many methods have been developed, such as H_2 [165] and H_∞ controllers [172], tube [101, 207, 79], multi-stage [114, 113] and scenario-based model predictive controllers [21].

The arrival of machine learning brought a new variable that we believe should be exploited when defining robust and risk-aware control approaches: Data. The approaches proposed in this thesis use data not only to gain more information about dynamical systems by training machine learning models but also to integrate data directly in the controller design as a way to measure the level of uncertainty of a system.

1.3. Contribution

In this thesis, we present two approaches dealing with the uncertainty of repetitive dynamical processes where machine learning influences control decisions. In the first approach, a *risk map* correlated to the model uncertainty is built using Gaussian processes. This uncertainty map enters the constraints of the MPC, which aim to achieve the control objective while limiting the risk. The second approach proposes a tube-based MPC approach that ensures constraint satisfaction despite the model uncertainty. This is based on the definition of a *safe set*: A subset of the state space where, also in the worst-case scenario, the constraints can be respected. As we will see, the safe set expands as the number of measurements used to train the machine learning model increases, reducing conservativeness. Both approaches take advantage of the repetitions to gain progressively more knowledge about the plant and improve the performance from run to run. The approaches are tested on examples such as bioreactors and robotic arms.

As a further outcome, this work also proposes HILO-MPC: A comprehensive, novel Python toolbox that can solve machine learning-supported optimization, predictive control, and estimation for research and teaching purposes.

To summarize, the main contributions of this work are

- a risk-aware run-to-run learning supported predictive controller that avoids areas with large model risk.
- a safe run-to-run learning supported predictive controller that ensures constraints satisfaction and recursive feasibility despite model uncertainty.
- a Python toolbox that allows the rapid development of machine learning-supported predictive control, estimation, and optimization problems.

The results of this work were used in a series of publications. Our book chapter [77] provides a literature review and a generalized framework for machine learning in the chemical and biochemical industry. In [131], we describe in a more concise form the results of Chapter 18. In [129], we present some preliminary concepts and results expanded in Chapter 23. The article [150] presents HILO-MPC. It describes the mathematical formulations of the problems that HILO-MPC can solve and shows some code snippets and application examples. HILO-MPC was also used in [130, 53, 149, 54] for more complex case studies.

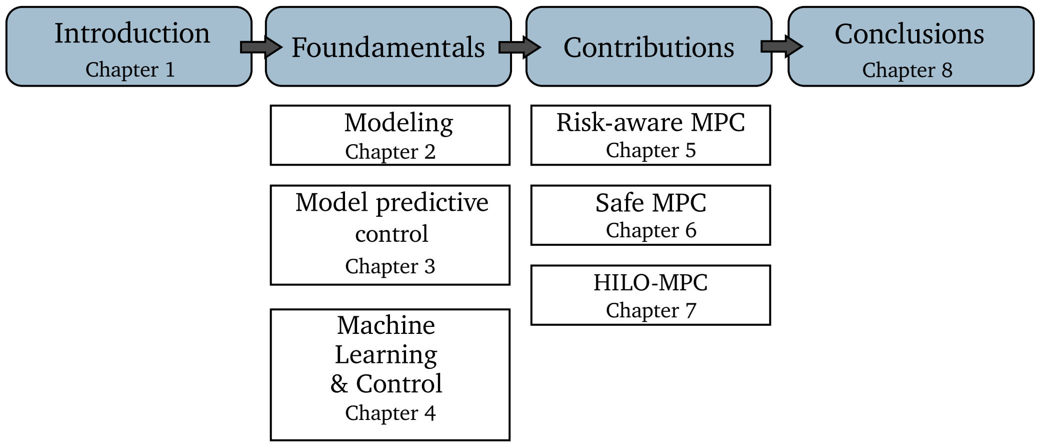


Figure 1.3.: Overview of the structure of this thesis.

1.4. Structure of the thesis

This thesis is structured in eight chapters (cf. Fig. 3). In Chapter 4, we cover the basics of the modeling frameworks used throughout the thesis: first-principle, machine learning (with a focus on neural networks and Gaussian processes), and hybrid models. In Chapter 9, the fundamentals of nominal MPC, robust MPC, and risk-aware MPC are given together with a literature review of the methods available. In Chapter 14, we review the literature on machine learning-supported MPC. Furthermore, we discuss the learning approaches and implementation details used for this thesis. Chapter 18 presents the first contribution of this thesis. This chapter proposes a risk-aware run-to-run learning and optimization strategy that improves the hybrid model accuracy run-to-run. This method is based on a risk function built with Gaussian process regressors that reflects the model uncertainty. This risk function is added to the MPC constraints to limit the risk the controller can take. We show how the model accuracy increases from run to run while avoiding areas where model uncertainty is large. While this method is simple and does not require expensive computations, it cannot guarantee constraint satisfaction. In Chapter 23, we propose a nonlinear machine learning supported tube MPC method that guarantees constraint satisfaction. This method is based on the definition of a *safe set*: A set where the model error is sufficiently small and nonlinear tube MPC can find a feasible solution. This allows to explore safely unknown process conditions despite model uncertainty. The safe set is usually small for the first runs, where only a few data points are available. As the number of runs (and consequently the measurements) increases, we show that this safe set expands and conservatism is reduced. We show the conditions that guarantee the expansion of the set at every run. Furthermore, recursive run-to-run feasibility is guaranteed.

In Chapter 30 a novel Python open-source toolbox called HILO-MPC[1] is introduced. HILO-MPC allows the easy and fast development of machine learning-supported control and estimation problems. To the best of the author's knowledge, this is the first toolbox specifically aimed at facilitating the use of machine learning in control with a special focus on optimization-based methods. We will showcase the toolbox with various problems, from control to estimation and machine learning.

We conclude with Chapter 39, where the proposed methods are summarized, and new possible research directions are discussed.

[1]HILO-MPC: macHIne Learning and Optimization for Modeling, Prediction and Control.

2. Modeling

In this chapter, we define the different types of models used in the rest of the thesis. The chapter starts with the general formulation of nonlinear continuous-time models, and it follows with a brief description of the two main machine learning models used in this thesis: Neural networks and Gaussian processes. Finally, the concept of hybrid machine learning/first principles models is explained. In this context, we will explain the concept of physics-informed learning and describe its advantages and disadvantages compared to classical data-driven learning.

2.1. Introduction

In this thesis, we consider nonlinear, continuous-time, time-invariant systems of the form:

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = \tilde{x}, \tag{2.1a}$$

$$y(t) = h(x(t), u(t)), \tag{2.1b}$$

where t is the time, $x \in \mathbb{R}^{n_x}$ the state vector, $u \in \mathbb{R}^{n_u}$ the input vector, $y \in \mathbb{R}^{n_y}$ the output vector, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the system of differential equations and $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}$ is the system of algebraic equations describing the measurements. The states will evolve in time, starting from the initial conditions $\tilde{x}$ under the influence of the input. We will indicate the solution of (1), starting at time t_k from a state $x(t_k)$ under the effect of an input $u(t)$ by

$$x(x(t_k), u(\cdot)) = x(t_k) + \int_{t_k}^{t} f(x(\tau), u(\tau))d\tau. \tag{2.2}$$

For some cases, we want to model the uncertainty, i.e., unknown effects acting on the system dynamics and measurements noise. One common way to achieve this is adding a state noise $w(t) \in \mathbb{R}^{n_w}$ to the model and the measurement noise $v(t) \in \mathbb{R}^{n_v}$ in the measurement equation

$$\dot{x}(t) = f(x(t), u(t), w(t)), \quad x(0) = \tilde{x} \tag{2.3a}$$

$$y(t) = h(x(t), u(t), v(t)), \tag{2.3b}$$

depending on the application, it is useful to model the noise vectors as random variables over a closed set with uniform distribution or over a distribution with infinite support, e.g., Gaussian normal distributed noise.

The models can be generated using first principles or machine learning models. First-principles models are derived using physics laws (e.g., conservation of mass or energy). They have the advantage of being interpretable and can be derived with a relatively small amount of data, which may reduce experimental efforts. Nevertheless, it is hard to build good first principles models in the case of complex and not well-understood systems. Machine learning models instead use data and no first-principles knowledge. They have the advantage of being able to represent complex relationships without an in-depth understanding of the system [193]. In the effort to combine the strength of both approaches, *hybrid models* use both first principles and machine learning components. In the next sections, we will go into more detail about machine learning models and hybrid models.

2.2. Machine learning models

Machine learning (ML) models are mathematical expressions (or maps) that try to infer relationships between inputs and outputs using data. ML can be used, e.g., to solve regression and classification problems and can be classified as supervised, unsupervised, or reinforcement learning [6]. The process through which the inference takes place is called *learning* or *training*. Before learning, the data are commonly pre-processed, e.g., to remove outliers and filter possible measurement noise. Then the data is used by the training algorithm to update the model parameters (also called *hyperparameters* or *weights*) in a way that improves their predictive or classification performance. In this thesis, we focus on *supervised* machine learning for regression tasks, i.e., both inputs and outputs are available during training, and the output of the machine learning takes continuous values. The advantage of machine learning models is that they can fit arbitrary complex functions if enough data and enough trainable parameters are provided. The interpolation power of data-driven models, in particular ANN, was demonstrated in [45, 82, 81] where it was proven that an ANN with a single hidden layer can approximate any continuous function (with some mild assumptions), provided that there are a sufficient number of neurons in the hidden layer. In general terms, a machine learning algorithm $\mathcal{L}_{\text{reg}}$ for regression can be defined as the following map [77]

$$(\mathcal{D} \times \Theta) \overset{\mathcal{L}_{\text{reg}}}{\to} \rho \in C^0\left(\mathcal{F}, \mathcal{L}\right), \tag{2.4}$$

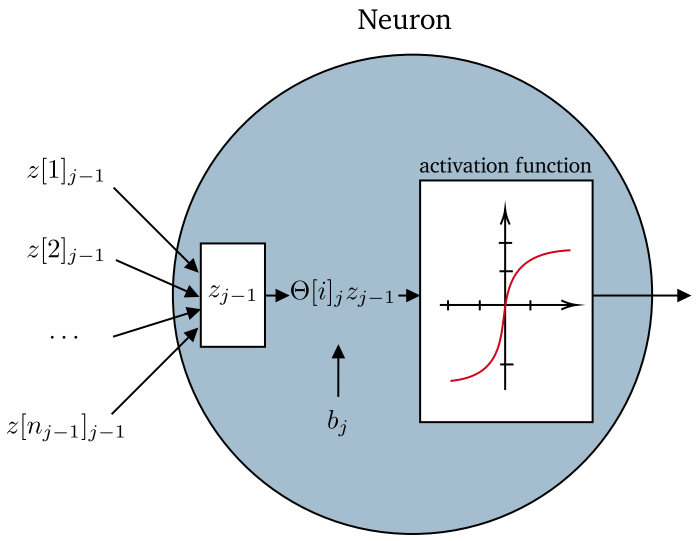


Figure 2.1.: Representation of the operations taking place in a neuron in a general layer j. $\Theta[i]_j$ represents the i-th row of the weight matrix of layer j, while $z[i]_j$ is the i-th element of the vector z_j.

where $\mathcal{D}$ is the dataset, Θ a set of parameters (for example neuron weights in case of a neural network), C^0 is the space of continuous functions, $\mathcal{F} \subset \mathbb{R}^{n_v}$ is the *feature space* and $\mathcal{L} \subset \mathbb{R}^{n_l}$ the *label space* and ρ is the machine learning function. The dataset $\mathcal{D}$ is a collection of tuples $\{(\hat{v}_1, \hat{l}_1), (\hat{v}_2, \hat{l}_2), \dots, (\hat{v}_{n_d}, \hat{l}_{n_d})\}$ containing the observed features $\hat{v}_i \in \mathbb{R}^{n_v}$ and labels $\hat{l}_i \in \mathbb{R}^{n_l}$ for $i = 1, \dots, n_d$ where n_d is the dataset size. For convenience we will also indicate $\mathcal{D} = \{\hat{V}, \hat{L}\}$ where $\hat{V} \in \mathbb{R}^{n_v \times n_d}$ the matrix of the stacked observed features and $\hat{L} \in \mathbb{R}^{n_l \times n_d}$ of observed labels. The function ρ depends on the machine learning method used. In this work, we focus on *artificial neural networks* (NNs) and Gaussian processes (GPs). In the following sections, we will give an introduction to these methods.

2.2.1. Artificial Neural Networks

Artificial Neural Networks are mathematical models that were initially inspired by the structure of biological neural networks [126]. They are among the most used machine learning models since they are relatively fast to train, can be trained with very large datasets, and are capable of learning relations with high complexity. The basic building blocks of every ANN are the neurons where some mathematical operations take place.

The set of operations in every neuron can be divided into multiplication, summation, and activation. The inputs of the network (i.e., features) are firstly multiplied with individual weights, and then they are summed together. The sum is finally passed through an *activation function* (Fig. 4). The result of these operations is passed to the next layer of neurons, where the operations are repeated (Fig. 5). Note that here, for simplicity, we consider feed-forward fully connected neural networks. The set of operations on the j-th layer is

$$z_j = \sigma_j \left(\Theta_j z_{j-1} + b_j\right), \tag{2.5}$$

where $z_j \in \mathbb{R}^{n_{z_j}}$ is the output vector of layer l, $\Theta_j \in \mathbb{R}^{n_{z_i} \times n_{z_{i-1}}}$ is the matrix of weights, $\sigma_j \colon \mathbb{R}^{n_{z_j}} \to \mathbb{R}^{n_{z_j}}$ is an element-wise activation function and $b_j \in \mathbb{R}^{n_{z_j}}$ the bias vector. The layer is initialized as $z_0 = v$. By stacking n_s layers together, we have the following function

$$\rho_\theta(v) \triangleq \sigma_{n_s}\Big(\Theta_{n_s}\sigma_{n_s-1}\big(\Theta_{n_s-1}\left(\ldots\sigma_1\left(\Theta_1 v + b_1\right)\ldots\right) + b_{n_s-1}\big) + b_{n_s}\Big) = l,$$

where $\theta = \{\Theta_1, ..., \Theta_{n_s}\}$ and n_s is the number of layers. Different activation functions can be used. Choosing nonlinear activation functions is fundamental for the network to learn nonlinear relationships. Common nonlinear activation functions are sigmoid, tangent hyperbolic, and ReLU functions [156]. The training of the ANN is achieved by adapting the weight matrices and the bias vectors via an optimization problem such that a user-defined loss function is minimized. For regression problems, one of the most commonly used loss functions is the mean squared error (MSE)

$$\begin{aligned} \theta^* \triangleq \underset{\theta}{\text{arg min}} \quad & \frac{1}{n_d}\sum_{i=1}^{n_d}(l_i - \hat{l}_i)^2 \\ \text{subject to} \quad & l_i = \rho_{\theta,}(v) \end{aligned} \tag{2.6}$$

where l_i are the values of the labels predicted by the ANN. Other loss functions can be used, such as weighted MSE or the mean absolute error. Commonly, the loss function is minimized with a stochastic gradient descent optimizer [97] and by using backpropagation [166]. The training is stopped when there is no significant change in the loss function over a defined number of iterations (this is known as *early stopping*) or if the number of maximum iterations (or epochs) is reached. Neural networks, while being a very flexible modeling approach, generally require a large amount of data and have the tendency to overfit.

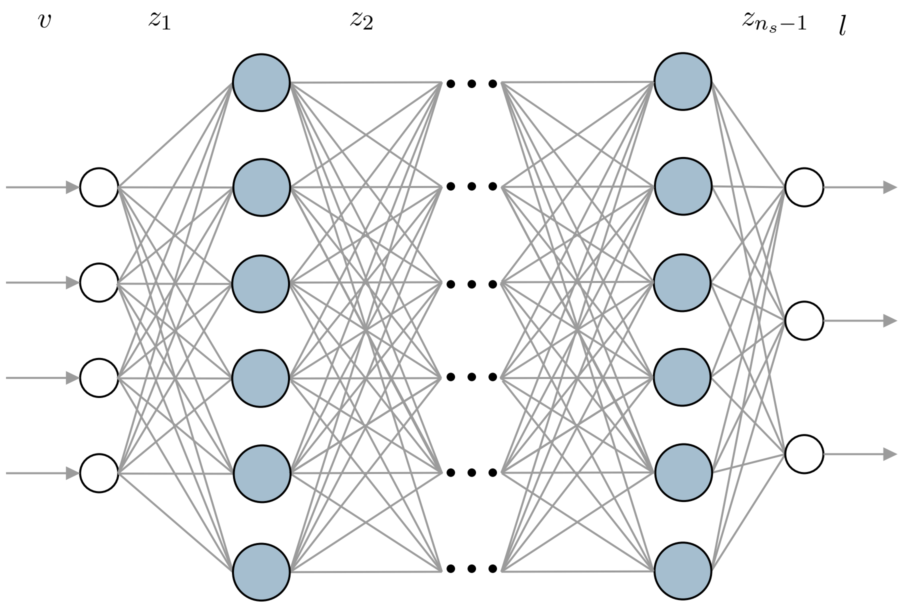


Figure 2.2.: Feed-forward fully-connected neural network with four features, n_s layers and three labels.

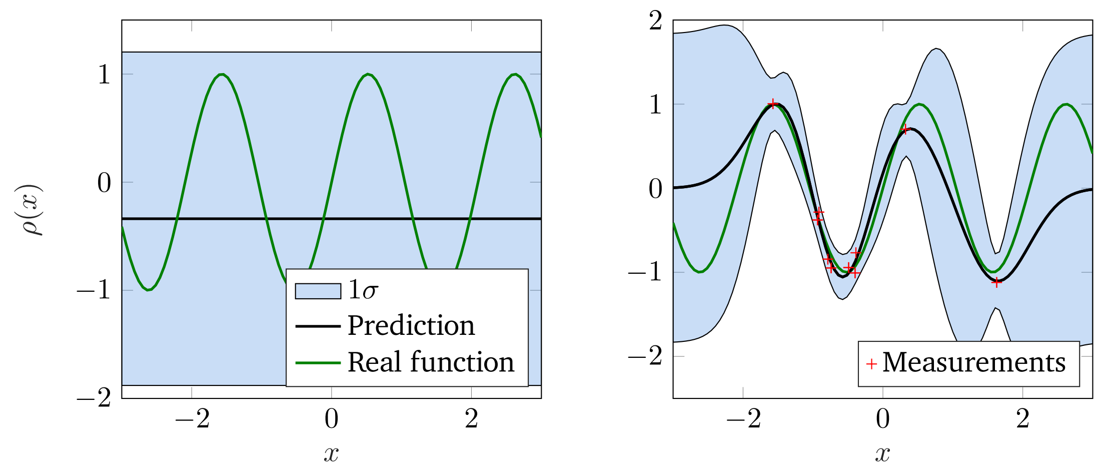


Figure 2.3.: Example of a GP fitting the function $\rho(x) = \sin(3x)$. The measurements are affected by noise. The left figure shows the prior mean and 1σ standard deviation. The right shows the posterior condition on the measurements and with optimal hyperparameters.

2.2.2. Gaussian processes

Gaussian processes (GPs) are machine learning models that assume Gaussian distribution over functions. This section is not meant to be a comprehensive description of GPs; rather, we want to convey the main idea and terminology. For further details on GPs, refer to [200]. The *prior* distribution, i.e., the distribution that describes the prior knowledge on the function distribution, is defined by its mean function $\mu(v)$ and covariance matrix $K \in \mathbb{R}^{n_d \times n_d}$

$$\rho_\theta(v) \sim \mathcal{N}(\mu(v), K). \tag{2.7}$$

The covariance matrix is built using *kernel* functions (also known as covariance functions) that describe the covariance of the random variables v. There is a large variety of kernel functions that can be used. One of the most common is the radial basis kernel

$$k(v_i, v_j) = \delta^2 \exp\left(\frac{(v_i - v_j)^2}{2l_\text{n}^2}\right), \tag{2.8}$$

where $k : \mathbb{R}^{n_v} \times \mathbb{R}^{n_v} \to \mathbb{R}$ is the kernel function, δ and l_n are hyperparameters that are optimized during training. The (i, j) element of K is given by $k(v_i, v_j)$. We can assume that the labels are affected by a Gaussian distributed noise ϵ as follows

$$l = \rho_\theta(v) + \epsilon, \tag{2.9}$$

with Gaussian distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Following the idea of Bayesian inference [29], when new observations on a process are available, the prior knowledge on this process is updated, forming the *posterior* distribution. The observations $\mathcal{D} = \{\hat{V}, \hat{L}\}$ are then used to condition the prediction and obtain an updated prediction at any test point v^*

$$\rho(v^*) \sim \mathcal{N}\left(\bar{\mu}, \bar{K}\right), \tag{2.10}$$

where

$$\bar{\mu}(v^*) = \mu(v^*) + K(v^*, \hat{V})^\intercal \left(K + \sigma^2 I\right)^{-1} (\hat{l} - \mu(v^*)), \tag{2.11}$$

$$\bar{K}(v^*) = k(v^*, v^*) - K(v^*, \hat{V})^\intercal \left(K + \sigma^2 I\right)^{-1} K(\hat{V}, v^*), \tag{2.12}$$

and $I \in \mathbb{R}^{n_d \times n_d}$ is the identity matrix. The GPs hyperparameters are commonly optimized by maximizing the log-marginal likelihood with respect to the hyperparameter vector $\theta = [\delta, l_\mathrm{n}, \sigma]$:

$$\theta^* \triangleq \underset{\theta}{\arg\max} \quad \left(-\frac{1}{2}\hat{L}^\intercal \left(K + \sigma^2 I\right)^{-1} \hat{L} - \frac{1}{2}\log|K + \sigma^2 I| - \frac{n}{2}\log 2\pi\right).$$

The advantage of GPs is that they provide not only an estimation of the unknown function but also the uncertainty of the estimation (Fig. 6). The main disadvantage is that the training requires the inversion of the covariance matrix K, which scales poorly with the number of data points, limiting the size of the datasets that can be used with GPs [200].

2.3. Hybrid models

There are several advantages in merging machine learning and first-principle models into so-called hybrid models[1]. In process engineering, data are often scarce, noisy, and expensive to obtain [209]. This is an obstacle to the use of purely machine learning models. Using existing (albeit limited) first-principle knowledge and limiting machine

[1]Note that here, the term hybrid model does not refer to hybrid systems, i.e., systems containing continuous and discrete states. The term *gray-box* models is also used, although in recent years, the term hybrid models became more common.

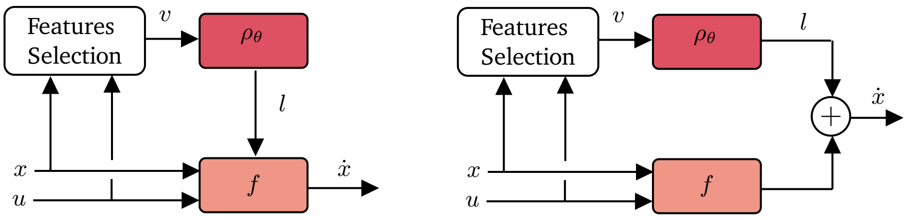


Figure 2.4.: Serial (left) and parallel (right) setups. The feature vector v can contain x, u, or other system variables and parameters.

learning to the most complex parts can help to reduce the amount of data required for model identification. As a result of these factors, hybrid modeling approaches have gained popularity [138, 193]. In this work, hybrid models have either a *serial* or *parallel* setup (cf. Fig. 7). The serial setup can be formulated as follows

$$\dot{x}(t) = f(x(t), u(t), \rho(v)). \tag{2.13}$$

This setup is advantageous when the structure of the first-principle models is accurate (cf. [193] and references therein). The parallel (or additive) setup is

$$\dot{x}(t) = f(x(t), u(t)) + \rho(v), \tag{2.14}$$

which is particularly indicated for the cases where the first-principle model structure is known not to be accurate. Note that the feature vector v can contain the states, inputs, parameters or other variables of the system.

2.3.1. Example: Hybrid model of a fed-batch bioreactor

Let us now, for example, consider a general model for a fed-batch bioreactor:

$$\dot{x}_{\text{bio}}(t) = (r_{\text{bio}}(t, x(t), u(t)) - F/V)\, x_{\text{bio}}(t), \tag{2.15}$$
$$\dot{x}_{\text{s}}(t) = r_{\text{s}}(t, x(t), u(t))x_{\text{bio}} + F/V(x_{\text{s,f}} - x_s(t)), \tag{2.16}$$
$$\dot{x}_{\text{p}}(t) = r_{\text{p}}(t, x(t), u(t))x_{\text{bio}} - F/V x_{\text{p}}(t), \tag{2.17}$$

where $x_{\text{bio}} \in \mathbb{R}$ is the biomass concentration, $x_{\text{s}} \in \mathbb{R}^{n_{x_s}}$ are the concentrations of substrates, $x_{\text{p}} \in \mathbb{R}^{n_{x_p}}$ are the concentrations of products, $F \in \mathbb{R}^{n_F}$ are the feed rates of substrates, $V \in \mathbb{R}$ the liquid volume of the reactor. $r_{\text{bio}} : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the biomass growth rate, $r_{\text{s}} : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_{x_s}}$ is the substrate uptake rate and $r_{\text{p}} : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_{x_p}}$ is the product production rate. The vector $u \in \mathbb{R}^{n_u}$ represents inputs that can influence the reaction rates (i.e. pH, temperature, light intensity of optogenetic systems [52]). The reaction rate also depends on the bacteria's metabolism, which in turn depends on both genetic and fermentation conditions. Reaction rates are often modeled with simple models fitted to experimental data, such as Monod, Droop, or Logistic models [189]. The prediction power of these models is limited within certain conditions. Therefore, it is usually not transferable to a reactor working in different conditions [2, 64]. Hence, to increase the prediction power, machine learning models can be used [138, 209, 193, 131]. For example, the reaction rates can be written as

$$r_i(t, x(t), u(t)) \approx \rho_{i,\theta}(t, x(t), u(t)) l_i(t, x(t), u(t)), \quad i \in [\text{bio}, \text{s}, \text{p}], \tag{2.18}$$

where $l(\cdot)$ is a known function representing some apriori knowledge. For example, it can be used to avoid the prediction of negative concentrations of the metabolites or products that can occur due to fitting errors of the ML model. Nevertheless, this function is not strictly necessary. This model is an example of a serial hybrid model commonly used for chemical and biochemical applications.

2.4. Training hybrid models

The training procedure of a hybrid model is an important design decision. Here, we briefly describe two training procedures for hybrid models that were used in this work; for details, please refer to [193]. A similar approach can be used for GPs. The dataset $\mathcal{D}$ is usually shuffled and divided in a *training dataset* $\mathcal{T}$ and *test dataset* $\mathcal{T}_{\text{st}}$ [2]. The network is trained on $\mathcal{T}$, and its generalization performance is tested on $\mathcal{T}_{\text{st}}$ [87]. A rule of thumb is using an $80\%/20\%$ split [3] i.e., 80% of the data is used for training and 20% for testing. In this work, two approaches will be used: a purely data-driven approach and a *physics-informed* approach [92]. In the purely data-driven approach, the machine learning function ρ is trained directly using the features and labels, that is, by minimizing the following objective function

$$\underset{\theta}{\text{minimize}} \quad \left(\frac{1}{\text{len}(\mathcal{T})} \sum_{i \in \mathcal{T}} \left(\hat{l}_i - \rho(\hat{v}_i, \theta) \right)^{\mathsf{T}} W \left(\hat{l}_i - \rho(\hat{v}_i, \theta) \right) \right) \tag{2.19}$$

where $\text{len}(\mathcal{T})$ is the number of data points in $\mathcal{T}$ and $W \in \mathbb{R}^{n_l \times n_l}$ is a weighting matrix. In this case, we explicitly wrote the dependency of the machine learning model ρ with

[2]For the results of this thesis we have not used a validation dataset, used for selecting the best machine learning model from a set of possible models.

[3]For big data applications, it is common to see a much smaller test set, e.g., $95\%/5\%$.

the parameters θ. Note that the first principles information does not enter the training procedure in this case. Instead, in the physics-informed approach the first principle information enters the optimization problem. Without loss of generality, we assume a parallel hybrid model setup. In this case, we have

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \left(\frac{1}{\text{len}(\mathcal{T})} \sum_{i \in \mathcal{T}} (\hat{x}_i - x_i)^{\mathsf{T}} W (\hat{x}_i - x_i)) \right), \\ \text{subject to} \quad & x_i = \hat{x}_{i-1} + \int_{t_i}^{t_{i+1}} f(\hat{x}, \hat{u}, \rho(\hat{v}, \theta) dt. \end{aligned} \tag{2.20}$$

Note that the ODE is integrated during the training procedure, hence the model influences the training. Note also that to use (19) we need the measurements of the labels, while to train with (20) we need only the measurements of the states and inputs. This can be useful when the measurements of the labels are not directly available, which can be the case, especially for parallel setups. The physics-informed approach has also been shown to have better performances compared to the classic approach [138]. Nevertheless, it requires the integration of the model, which can lead to numerical issues if the initial guess of θ is far away from the optimal or if ρ contains non-smooth components (e.g., non-smooth activation functions). Furthermore, the integration step can slow down the learning procedure. Note that this is only a way to obtain physics-informed learning. In general, the term physics-informed learning indicates all the learning approaches that introduce physics-derived observation, inductive or learning biases (cf. [92] and references therein). In this work, both classic and physics-informed learning approaches will be used to train hybrid models.

2.5. Summary

In this chapter, we outlined the modeling approaches that are used in this thesis. In particular, we focused on two machine learning models: Feed-forward fully-connected neural networks and Gaussian processes. We discussed the advantages and disadvantages of both models. The hybrid modeling approach that uses machine learning and first-principles models is also briefly discussed. Finally, we described the classic learning procedure and a physics-informed learning procedure.

3. Model Predictive Control

This chapter presents an overview of the fundamental concepts of Model Predictive Control (MPC). We start by formulating the nominal MPC problem for continuous systems and then extend it to repetitive finite-time systems while highlighting practical and theoretical distinctions. Additionally, we provide a concise summary of MPC approaches that account for uncertainties.

3.1. Introduction

Model Predictive Control (MPC) is an advanced control strategy with a solid track record of applications in many industries, such as chemical engineering [144, 111], autonomous driving [201, 161], power electronics [190], internet-of-things [38, 93] and robotics [103, 63] to cite just a few. Furthermore, theoretical properties, such as stability and recursive feasibility, have been thoroughly investigated by academics in the last 70 years [125]. The success of MPC can be attributed to its ability to address two critical challenges faced by classical controller approaches like PID controllers, namely accounting for system constraints and handling multi-input and multi-output systems. With the increasing complexity of modern systems and the growing demand for automation, MPC has emerged as a widely adopted solution. Model Predictive Control has its roots in the theory of optimal control, primarily derived by Lev Pontryagin and Richard Bellman in the 1950s [19]. The objective of optimal control is to find a control law that steers a dynamical system such that an objective function is optimized, i.e., minimized or maximized. This optimal control law can be found by solving the Hamilton-Jacobi-Bellman equation or Pontryagin's maximum principle. Nevertheless, finding a solution to these problems is challenging, except for some special cases. To solve this Model Predictive Control approximates the closed-loop optimal control solution by an open-loop solution within a receding finite horizon and repeatedly solves an optimal control problem at discrete time points to introduce feedback. Constraints can be easily incorporated into the optimization problem.

The main idea behind receding-horizon MPC is depicted in the left part of Fig. 8. MPC predicts the system in the future within the prediction horizon and tries to find an optimal control sequence that optimizes an objective function, e.g., minimizes the distance of states from some reference trajectories. Commonly, once a (possibly local) optimum input sequence is found, a given portion of the input sequence is applied to the plant. The plant proceeds in open-loop until the next sampling time, where the states are measured or estimated, and the optimization is repeated [159]. This “closes the loop” and gives the MPC some robustness to uncertainties.

One of the main drawbacks of MPC is that it requires a sufficiently good model since the prediction quality of the model influences the MPC performances. For specific applications, another drawback is the computational load of the optimization problem that has to be solved at every sampling time. This increases, e.g., with the prediction horizon’s length and the model’s complexity. Thus, a trade-off problem is posed where model prediction quality and complexity must be balanced to ensure real-time applicability. Models are never the exact representation of a system. Hence, the predictions will always be subject to some degree of uncertainty. Often, it is necessary to take this uncertainty into account. There are mainly two ways to achieve this: The model can be adapted (online or offline), or the model uncertainty can be considered in the optimization problem, e.g., by considering the propagation of the uncertainty into the future. The first approach is commonly called *adaptive* MPC, while the second is called *robust* or *stochastic* MPC, depending on the representation of the uncertainty [127]. If the MPC does not explicitly consider model uncertainty and assumes that the model is a perfect representation of the system, it is commonly referred to as *nominal* MPC.

In the following sections, we will formalize the nominal MPC problem for continuous processes, then for repetitive finite-time processes, and highlight practical and theoretical differences.

3.2. Nominal Model Predictive Control

3.2.1. Continuous processes

Continuous processes are processes that are interrupted only for maintenance or for their decommissioning. Often, they operate in a steady state around a set point. In this case, the task of the controllers in the lower levels of the control hierarchy is to keep the plant close to the set point, reject the disturbances, or bring the system to a new set point when a change is required. The set-point value can be dictated by, e.g., market requirements or by an optimization layer like Real Time Optimization [118]. For these processes, classic receding-horizon MPC can be used. In the following, we define the receding horizon MPC problem.

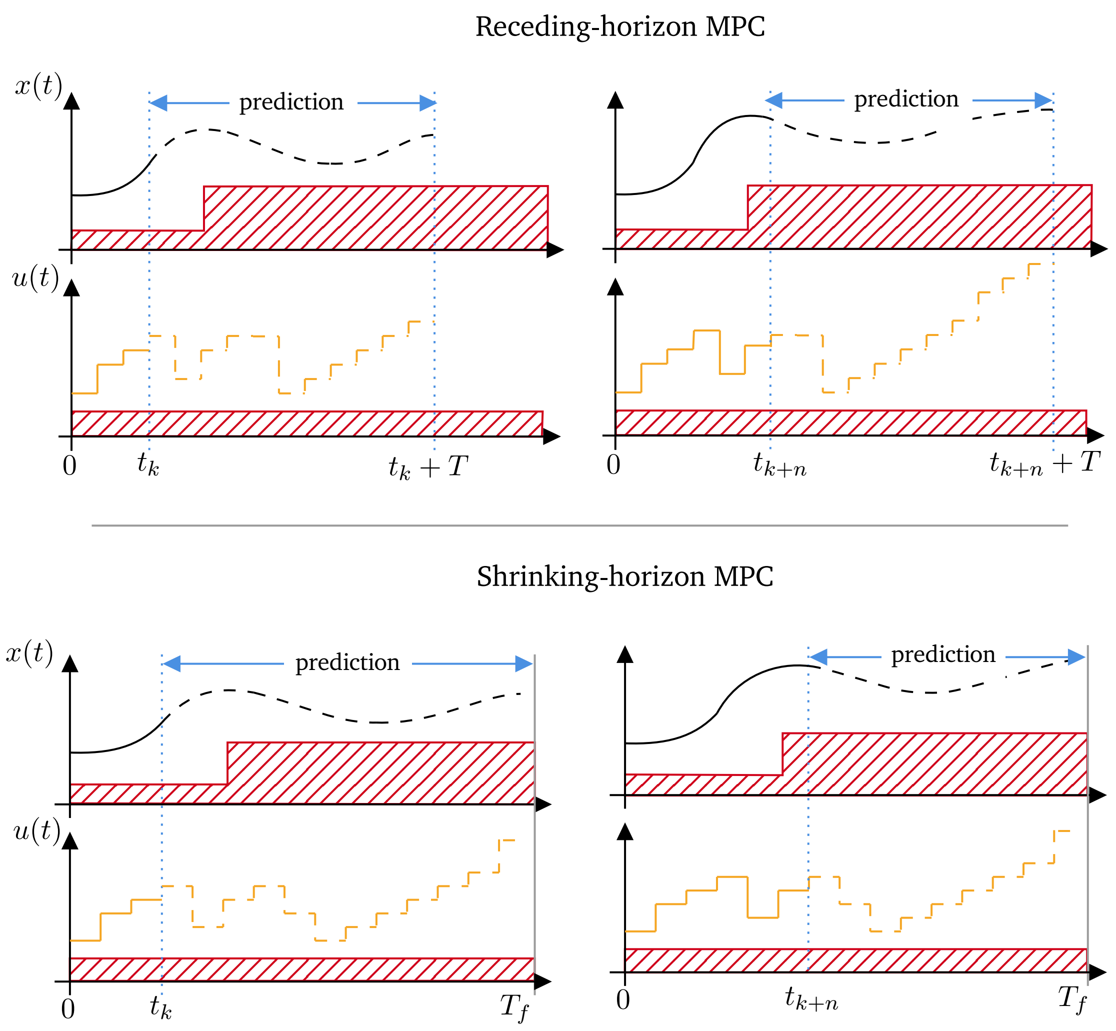


Figure 3.1.: Receding-horizon MPC and shrinking-horizon MPC. The solid black line is the real plant state, and the dashed black line is the predicted state. The solid orange lines are the input applied to the system, and the dashed orange lines are the predicted inputs. Let t_k be the $k-$th sampling time and $n \in \mathbb{N}$, in the receding horizon case, the horizon length for t_k and t_{k+n} is the same; hence the prediction shifts forward in the future. In the shrinking horizon case, the horizon at t_{k+n} is smaller than t_k as the prediction runs up to a fixed final time T_f. The red areas represent state and input constraints.

We consider continuous-time time-invariant nonlinear systems (1). At every sampling time t_k, the following optimal control problem is solved:

$$\begin{aligned}
\min_{\bar{u}(\cdot)} \quad & J(\bar{x}(\cdot), \bar{u}(\cdot)) && \text{(3.1a)}\\
\text{s.t.} \quad & \dot{\bar{x}}(t) = f(\bar{x}(t), \bar{u}(t)), \quad \bar{x}(t_k) = \tilde{x}(t_k), && \text{(3.1b)}\\
& \bar{x}(t) \in \mathbb{X}, \bar{u}(t) \in \mathbb{U}, && \text{(3.1c)}\\
& \bar{x}(t_k + T) \in \mathbb{E} \subseteq \mathbb{X} \;, \quad t \in [t_k, t_k + T] \subset \mathbb{R}, && \text{(3.1d)}
\end{aligned}$$

with objective function

$$J(\bar{x}(\cdot), \bar{u}(\cdot)) \triangleq \int_{t_k}^{t_k+T} l(\bar{x}(\tau), \bar{u}(\tau)) d\tau + e(\bar{x}(t_k + T)), \tag{3.2}$$

where $\bar{u}(\cdot)$ is the input function, $l : \mathbb{R}_0^+ \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}_0^+$ is the *stage cost*, $e : \mathbb{R}_0^+ \times \mathbb{R}^{n_x} \to \mathbb{R}_0^+$ is the *terminal cost*, $\tilde{x}(t_k)$ is the measured or estimated state at time t_k, $\mathbb{X} \subseteq \mathbb{R}^{n_x}$ and $\mathbb{U} \subseteq \mathbb{R}^{n_u}$ are state and input constraints, respectively. $\mathbb{E} \subseteq \mathbb{R}^{n_x}$ is called *terminal constraint* and it can be imposed to guarantee some stability and recursive feasibility properties. With the bar $\bar{(\cdot)}$, we indicate the predicted values, which are in general different from their counterpart in the system even for the nominal case (i.e., uncertainty-free) because we are solving the finite-horizon problem at every t_k.

We make the following assumptions

Assumption 3.1 *The function $f(\cdot, \cdot)$ is continuous, satisfies $f(0, 0) = 0$ and it is (locally) Lipschitz in x.*

Assumption 3.2 *$\mathbb{U}$ is compact, $\mathbb{X}$ is closed, $0 \in \mathbb{X}$ and $0 \in \mathbb{U}$.*

Assumption 3.3 *$l(\cdot, \cdot)$ is is continuous, satisfies $l(0, 0) = 0$, and it is lower bounded by a class $\mathcal{K}$ function $\alpha_F(\cdot)$, i.e., $\alpha_F(x) \leq l(x, u)$.*

In practice, Problem (21) is solved at finite sampling times. For example, assuming equally-spaced sampling times, $t_k = k\Delta t$ for $k \in \mathbb{N}_0$ with constant time interval Δt. In this case, the optimal input, denoted by $\bar{u}^*(\cdot, x(t_k))$, is applied open-loop until the next sampling time t_{k+1}. This is called *sampled data open-loop* MPC [62]. If Problem (21) is solved for all time instances, we talk about *istantaneous MPC*. From a theoretical point of view, the latter is easier to handle. However, it is impractical since solving Problem (21) often takes a non-negligible amount of time.

The choice of the objective function (21a) is essential: It should be defined so that the control task can be achieved. For example, for set-point tracking problems, the stage and terminal cost can be defined as[1]

$$\begin{aligned} l(\bar{x}(t), \bar{u}(t)) &= \|\bar{x}(t) - x_r\|_Q^2 + \|\bar{u}(t) - u_r\|_R^2, \quad t \in [t_k, t_k + T], \\ e(\bar{x}(t_k + T)) &= \|\bar{x}(t_k + T) - x_r\|_E^2 \end{aligned}$$

where $\|a\|_A^2 = a^\mathsf{T} A a$, x_r is the state set-point, u_r the input set-point and $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$ and $E \in \mathbb{R}^{n_x \times n_x}$ are weighting matrices. These two terms express the *cost* we pay when a given variable stays away from its given reference. Consequently, minimizing this cost, in turn, minimizes the distance of this variable to the reference.

Notes on numerical solutions

Note that $u(\cdot)$ is a function, which renders Problem (21) infinite dimensional, hence very challenging to solve, especially in the presence of constraints. In practice, two approaches are commonly used to approximate and solve Problem (21): *Indirect* and *direct* approaches. In the indirect approaches, the Pontryagin minimum principle [100] is used to obtain the optimality conditions in the form of a two-point boundary problem, which is then solved through discretization using single shooting, multiple shooting, or orthogonal collocation techniques (cf. [153] and references therein). In the direct approaches, Problem (21) is directly discretized in time to transform it into a finite-dimensional (static) nonlinear programming problem. This is achieved by parametrizing the input sequence into a finite number of parameters. In this work, a piece-wise constant input is used, which is kept constant between sampling intervals. This leads to a finite number of control actions $u = [u_0, ...u_{N-1}]$ where $N \triangleq \text{ceil}(T/\Delta t)$ is the prediction horizon length[2], i.e., the number of sampling times over which the system is predicted into the future, and ceil : $\mathbb{R} \to \mathbb{Z}$ is the ceiling function, i.e., the function that returns the smallest integer which is larger than a given real number. Once the control sequence $u = [u_0, ...u_{N-1}]$ is obtained, generally, only the first control action u_0 is applied to the plant, and then Problem (21) is solved again at the next sampling time. After the discretization, single-shooting, multiple-shooting, or orthogonal collocation can be used. The MPC problems solved in this thesis use direct approaches with orthogonal collocation.

[1]Here, we show a time-invariant stage and terminal cost, which is most commonly used.

[2]In this work, for simplicity we will assume that the prediction horizon is equal to the control horizon, i.e., the control is optimized up the time $t_k + T$ as for the state.

3.2.2. Finite-time processes

Contrary to continuous processes, repetitive processes are characterized by the presence of a finite terminal time. By design, at least some of the process states are not maintained at a steady state but vary considerably during the operation, e.g., can follow a predefined time-varying profile. Once a run is completed, the process is repeated. We will call *run, batch* or *iteration* a process repetition. The initial condition and the run time, i.e., the time that it takes to complete one run, can be the same or differ at every run. For these processes, there exist two time scales: The continuous time scale within the batch and the discrete scale, which indicates the batch repetitions. We refer to parameters, variables, and functions of a run using a *run index* $i \in \mathbb{Z}$ as superscript, i.e., $(\cdot)^i$. For simplicity of notation, in the following, we will use the same run time T_f for all runs, and we will assume that every run starts from the same initial condition.

Model Predictive Control can also be applied for repetitive processes, provided that the prediction horizon is shrunk as it approaches the final time. Let $t = 0$ be the starting time and T_f the final time, let t_k be the current sampling time, and T be the horizon length of the MPC. If the prediction does not reach the final time, i.e., $t_k + T < T_f$, then the receding-horizon MPC problem (21) can be used. As soon as the prediction reaches the finite time, i.e., $t_k + T \geq T_f$, the horizon is shrunk, such that the final prediction time corresponds to the final process time, i.e., by defining $T \triangleq T_f - t_k$ (see Fig. 8). Hence, the MPC horizon stops *receding* and starts *shrinking*. Problem (21) is valid for the receding horizon case. For a shrinking-horizon MPC case, the optimization problem is adapted as follows:

$$\min_{\bar{u}(\cdot)} \quad J(\bar{x}(\cdot), \bar{u}(\cdot)) \triangleq \int_{t_k}^{T_f} l(\bar{x}(\tau), \bar{u}(\tau))d\tau + e(\bar{x}(T_f)), \tag{3.3a}$$

$$\text{s.t.} \quad \dot{\bar{x}}(t) = f(\bar{x}(t), \bar{u}(t)), \quad \bar{x}(t_k) = \tilde{x}(t_k), \tag{3.3b}$$

$$\bar{x}(t) \in \mathbb{X}, \bar{u}(t) \in \mathbb{U}, \tag{3.3c}$$

$$\bar{x}(T_f) \in \mathbb{E} \subseteq \mathbb{X} \;, \quad t \in [t_k, T_f] \subset \mathbb{R}. \tag{3.3d}$$

Note that (23a) is defined between the current sampling time t_k and the final time T_f. Often, in continuous processes, the goal is to keep the system at a steady state as close as possible to a desired reference. For finite-time processes, the objective can be, e.g., to follow a time-varying profile or to maximize profit at the end of the process. In the first case, we solve a trajectory-tracking problem, and the stage and terminal cost can look like

$$\begin{aligned} l(\bar{x}(t), \bar{u}(t)) &= \|\bar{x}(t) - x_r(t)\|_Q^2 + \|\bar{u}(t) - u_r(t)\|_R^2, \quad t \in [t_k, T_f], \\ e(\bar{x}(T_f)) &= \|\bar{x}(T_f) - x_r(T_f)\|_E^2, \end{aligned}$$

where $x_r : \mathbb{R} \to \mathbb{R}^{n_x}$ and $u_r : \mathbb{R} \to \mathbb{R}^{n_u}$ are two time-varying functions given, e.g., from the solution of an economic optimization and/or from heuristics or small-scale experiments. Instead, in the case of economic objectives, we generally have an expression that describes the cost/profit of the process as a function of the states and inputs. In this case, the problem becomes a shrinking-horizon *economic* model predictive controller and will be used in Chapter 18.

3.3. Theoretical differences between continuous and finite-time processes

A basic property that a controller is required to satisfy is state and/or output stability. Stability can be defined in different terms: For controlled systems, input-to-state and input-output stability [176] is commonly used. Broadly speaking, stability definitions characterize a system in terms of the convergence of its states or outputs into a steady state or into a bounded set. Generally, they are based on the assumption that the system runs for long enough (or infinite) time. Hence, stability is a qualitative property of the controlled system: A system is either stable or not. These definitions of stability do not apply to finite-time processes since they do not run for an infinite time and do not necessarily operate at a steady state [178]. For these processes, other definitions of stability have been proposed [8, 80]. In [8], a finite-time system is said to be input-output stable if, given a certain class of bounded input over a time interval, the output does not exceed a predefined threshold. Note that this definition is quantitative, i.e., the same system can be finite-time stable or not, depending on the chosen threshold. In [80] a system (it could be continuous or finite-time) is called finite-time input-to-state stable if stability can be reached within a given finite time. In this work, we will not be concerned with any definitions of finite-time stability. The processes considered do not necessarily operate in a steady state, so the definition of [80] is not useful. Additionally, here, we are not interested in measuring the degree of stability as in [8]. Here, we will only assume that, for any of the admissible input, the system does not have a finite escape time smaller than the batch time, i.e.:

$$\forall u(\cdot) \in \mathbb{U}, \ \forall x(t_0) \in \mathbb{X}_0, \ \ \|x(t; x(t_0), u(\cdot))\| < \infty, \forall t \in [t_0, T_f] \subset \mathbb{R} \tag{3.4}$$

Other than this basic stability property, since the MPC repeatedly solves a constrained optimal control problem along a finite horizon, it is necessary to make sure that (21) is able to return a *feasible* solution at every iteration. In other words, there is a solution that guarantees constraint satisfaction. Guaranteeing a priori a feasible solution to Problem 21 is possible only for some special cases. Hence, it is commonly assumed that the problem is feasible at time t_0, and then conditions that guarantee the solution for $\forall\ t > t_0$ are found. This property is called *recursive feasibility*. In a receding horizon approach, recursive feasibility might not also be achieved for the nominal case since the prediction horizon is finite, and the controller might not “see” that, in the next sampling times, the problem might be unfeasible. Instead, in a shrinking horizon approach, where the MPC predicts until the end of the batch, recursive feasibility for the nominal case can be easily achieved if the MPC problem has a solution at time t_0 since, for this case, the MPC has full knowledge, because it predicts until the end of the process and the model is perfectly known.

In the presence of uncertainties, such as model-plant mismatch and measurement noise, ensuring the previous two properties becomes more challenging. In the next section, we will briefly describe how uncertainties can be considered in an MPC approach.

3.4. Model Predictive Control under uncertainties

In the previous sections, we showed the nominal MPC formulation. In the nominal MPC case, the model is considered to be a perfect representation of the plant. In practice, uncertainties are always present, e.g., in the form of measurement noise or model uncertainty (the latter is commonly called model-plant mismatch). Nominal MPC has some degree of *robustness*, i.e., it maintains its properties of constraint satisfaction, recursive feasibility, and stability also in the presence of some (small) uncertainty [68, 109], which can be sufficient for some practical applications. Nevertheless, in many other cases, uncertainties are an important issue and have to be regarded in the control design.

In the frame of MPC, there are three main approaches that deal with uncertainties: *adaptive*, *robust*, and *stochastic* MPC. The term adaptive MPC refers to all the techniques that adapt the model online or offline using the available measurements [3]. For example, for a nonlinear continuous-time time-invariant system as

$$\dot{x}(t) = f(x(t), u(t), p), \tag{3.5a}$$

$$y(t) = h(x(t), u(t)), \tag{3.5b}$$

where $p \in \mathcal{P} \subseteq \mathbb{R}^{n_p}$ are uncertain model parameters that have to be estimated. These can be either parameters of first principles models or machine learning models. An estimator is a map $\mathcal{E} : \mathcal{D} \mapsto \mathcal{P}$ that, given the data $\mathcal{D}$, estimates the parameter vector p. Let $\tilde{p}$ be the estimated parameter; the model is updated with $\tilde{p}$ when this estimate is available. The map $\mathcal{E}$ can be any estimation algorithm, e.g., Kalman Filters, [106, 152, 15, 133, 141], Moving Horizon Estimator [217, 179, 94, 86] or machine learning [4, 74]. Model adaptation is based on the *separation principle*, which assumes that estimation and control design can be separated [14]. According to this principle, the estimation can be carried out separately from the control problem. The estimated values can then be used to adapt the model, which is used by the controller as if it was perfect[3]. If the separation principle is valid, the deterministic optimal control law obtained with the adapted model is equal to the stochastic optimal control law. Unfortunately, in general, the separation principle is not valid for nonlinear systems, but it is valid for unconstrained linear problems with Gaussian distributed uncertainties [180]. Hence, also with model adaptation, the theoretical properties of stability and recursive feasibility might be lost, i.e., there are no more guarantees that the controller stabilizes the plant and respects the constraints. To solve this problem, the controller must consider the model uncertainty. This is the case for robust and stochastic MPC approaches.

In robust MPC approaches, the main assumption is that uncertainty is *bounded*, i.e., it can take (unknown) values within some known and bounded set. One of the first approaches of this class was the min-max MPC [154]. In this case, the optimization is carried out for the worst-case disturbance, but the uncertainties accumulate in the prediction, causing very conservative control actions or infeasibility. Other approaches that consider feedback in the prediction to reduce the spread of the uncertainty have been proposed, such as tube MPC [123] and multistage MPC [114]. Tube model predictive control is a formalization of the hierarchical control idea, where a slower higher-level control is used to define the trajectory of the plant, and a faster lower-level control stabilizes the plant around this trajectory by rejecting the disturbances. In tube MPC, an *ancillary* feedback controller is used to maintain the system states in a bounded set centered on the states of a known nominal (uncertainty-free) model. By tightening the constraints by the size of this set, it is ensured that if the nominal model satisfies the tightened constraints, then the plant satisfies the original constraints (cf. Fig. 9). Multistage MPC considers the case of parametric uncertainty. It assumes that the uncertain parameters can take a finite number of values that are known apriori. These values are taken in a way that represents the uncertainty of the parameters, for example, by sampling from a given distribution representing the parameter's uncertainty. By assuming that the parameters can change at the sampling times, a scenario tree is formed. Hence, constraint satisfaction, stability, and recursive feasibility can be guaranteed, assuming that the real system does evolve as predicted in one of the scenarios [113, 114]. Robust MPC approaches are, in general,

[3]This is also called *certainty equivalence* controller.

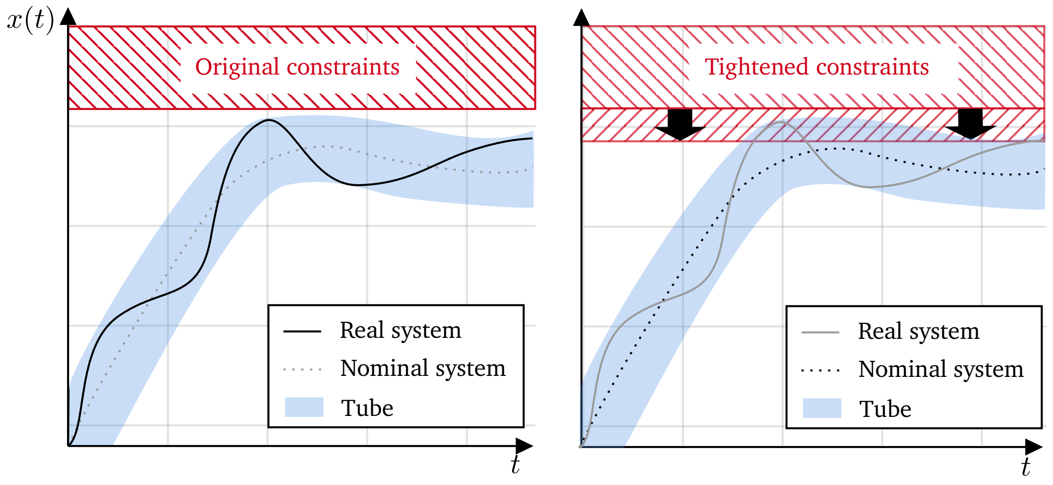


Figure 3.2.: Tube MPC approach. The ancillary controller keeps the real system bounded around the nominal trajectory. The constraint is shrunk by a length equal to the radius of the tube (right). Hence, if the nominal system respects the shrunk constraints, the real system will respect the original constraints (left).

conservative since they need to consider the worst-case realization of the uncertainty. In practice, the worst-case realization is an unlikely event. This motivates the use of stochastic MPC approaches.

In stochastic MPC, the model uncertainty is modeled as a stochastic process. Uncertainties might be unbounded, and large uncertainty usually has a low probability of occurring. Consequently, theoretical properties can be given only in expected value, e.g., as chance constraints, and are never one-hundred-percent certain [127, 96]. The disadvantage of stochastic MPC is computational complexity. This is caused by the propagation of the uncertainty into the future through the nonlinear system. It is well known that even a Gaussian distributed uncertainty, when propagated through a nonlinear system, results in a non-Gaussian distributed uncertainty that is challenging to describe [127].

3.5. Summary

This chapter introduced the concepts of receding-horizon and shrinking-horizon model predictive control. The main difference between the two approaches is that in receding-horizon MPC, the horizon rigidly shifts into the future, while in shrinking horizon MPC, it reduces its size at every sampling time. Furthermore, nominal MPC, adaptive, robust, and stochastic MPC approaches were briefly discussed. Nominal MPC assumes perfect knowledge of the plant, while adaptive MPC uses measurements that adapt the model and improve its accuracy, but it does not take the uncertainty directly into account in the solution of the open-loop OCP. Robust MPC and stochastic MPC approaches, instead, take the uncertainty into account and can guarantee constraint satisfaction and recursive feasibility despite the presence of uncertainties. Some robust MPC approaches, such as tube MPC, ensure constraint satisfaction by considering the uncertainty-free model and adequately tightening the constraints to guarantee the feasibility of the problem at every sampling time. Robust MPC approaches assume that the uncertainty is uniformly distributed in a bounded set. Stochastic MPC approaches usually consider a non-uniform stochastic distribution of the uncertainty.

4. Machine Learning & Control

This chapter reviews some of the most relevant developments concerning machine learning-supported/based control and run-to-run learning and control with a particular focus on robust and risk-aware control approaches. We will highlight the challenges of using these methods in process engineering. This chapter aims to give an overview of the state of the art and challenges of these approaches to set the basis for the contributions of this thesis.

4.1. Introduction

For many decades, data has always been used in control engineering, e.g., to fit the model parameters to measurements and build statistical models. Nevertheless, in recent years, thanks to a renewed and more substantial interest in machine learning, the number of applications in industry and research has dramatically increased. The main two factors that differentiate this new generation of machine learning solutions from the previous generation are the quantity of data and the available computational power [1]. Data quantity is less of a concern since sensor technology has become better and economically more accessible. This allowed techniques such as deep learning to solve the feature selection bottleneck: While before feature selection was carried out manually, using experience and statistical analysis (which would surely introduce bias and human errors), deep neural networks can automatically extract the features from the data, allowing them to find very complex nonlinear maps. This spurred the application of these approaches also in control engineering. Covering all the aspects of this developing field is obviously impossible in this thesis; the reader is referred to the review [71] for a more in-depth discussion of the subject. Here, we want to point out that while big datasets can be built for some systems (e.g., robotic systems), in others (e.g., chemical engineering, medicine), data is expensive and noisy. In chemical engineering, for example, measurements can be expensive, and the available data can be obsolete due to, e.g., substituting some plant components [168]. Furthermore, it is usually not possible to run the number of experiments necessary to obtain the required amount of data since these systems are usually large, highly interconnected,

[1]Commonly, the term “big data” is used to refer to this new machine learning paradigm.

expensive to run, have slow dynamics, and the experiments aiming at collecting data could jeopardize the safety and health of the system. Using machine learning models to describe the dynamics of repetitive processes is generally even more challenging than continuous processes since the model must be valid in a wide range of conditions [28].

Here, after a brief introduction to machine learning applied for control, we will review contributions that try to answer at least one of the following questions: How do we improve run-to-run performances of repetitive processes? How can we guarantee constraint satisfaction (or safety) despite uncertainties in machine learning-supported control?

4.1.1. learning supported Model Predictive Control

In the last years, the term learning *based* MPC has been used mainly to refer to control approaches that use machine learning to (at least partially) learn the process dynamics [74]. Note that other control design components can also be learned (cf. Fig. 10). For example in [121] the output function is learned, in [164, 163, 33] the terminal constraints of an MPC, in [34, 10, 131, 186] the path constraints and in [184, 16, 31, 128] the objective function. This thesis will refer to any control design that implements machine learning in any of its components as learning *supported* MPC. Hence, given this definition, learning-based MPC belongs to the class of learning-supported MPC. For example, in this thesis, machine learning was used to learn the risk constraints (Chapter 18) and the model dynamics (Chapter 23).

Some of the most common machine learning models used in combination with MPC are feedforward neural networks, Gaussian processes, and recurrent neural networks. In most cases, regression models are used; in other words, the output of the machine learning model is continuous. The learning of the model can occur online, offline, or in batches[2]. In online learning, the measurements are immediately used to update the model; in offline learning, the model is only updated before the controller is deployed, while in batch learning, the learning occurs in intervals when a certain amount of data has been collected. In this thesis, we will use offline learning; in particular, the learning occurs between every run.

It is important to notice that regardless of the method used, the learning and control influence each other, which brings us to the concept of active and passive learning.

[2]Not to be confused with the term batch that refers to the subset of data points used usually when training large datasets to decrease the effect of noise and speed up the training.

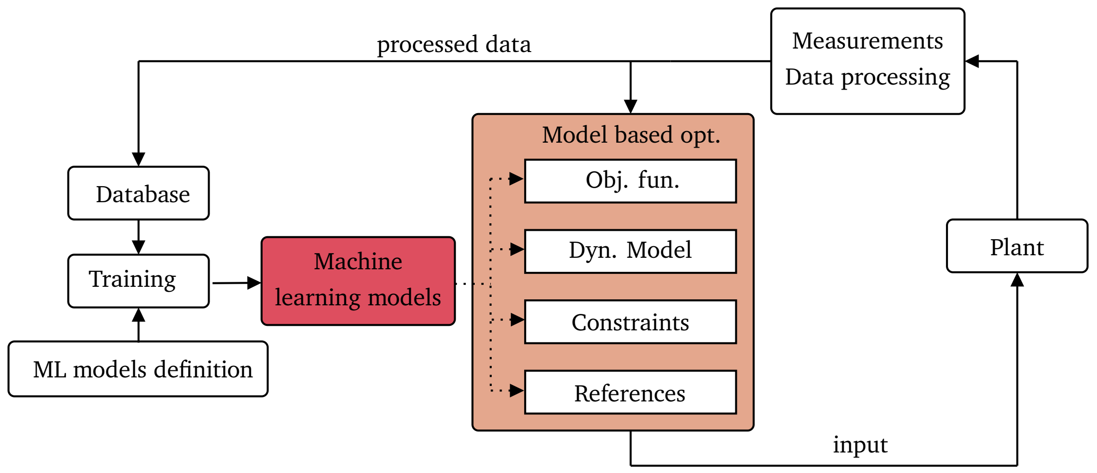


Figure 4.1.: Machine learning models can be used not only for the system dynamics but also for other parts of the MPC design.

4.1.2. Notes on active and passive learning

It is crucial to notice that when model identification is conducted in conjunction with the control task, the control action and the collected data mutually influence each other. In previous studies such as [57, 58, 59, 60], it has been suggested that in order to enhance overall control performance, the control input should exhibit a *probing effect*. This means that the control input should excite the system in a way that generates information-rich measurements, which can be used to improve the model's learning process. However, this may result in poorer transient system performance initially, but the more informative measurements obtained can eventually lead to improved overall control performance in the long term when used to adapt the model. This phenomenon is commonly referred to as the *dual* control problem and is well recognized in the field of control theory, as documented in [127, 71, 70] and other relevant literature.

One could think of explicitly considering both the identification task and the control task in the control design, in other words, to make the control "aware" of its effect on the system uncertainty. In this case, we talk about control with *active learning*. This can be achieved by designing a control that computes the input by considering effects at least on the covariance of the states (and parameters) or on higher-order moments [71]. If the controller is not aware of these effects on the uncertainties, i.e., only the effect on the mean of the states (and parameters) are considered, we talk about *passive* *learning*[3]. On the one hand, active learning can achieve an optimal balance between exploration and exploitation (i.e., probing effect and control); on the other hand, it is usually computationally expensive since it requires propagating the system uncertainty and predicting it into the future, and it requires expert knowledge to use and to maintain. The advantages of passive learning are that the control structure is easier to develop, maintain, and interpret, but it reaches sub-optimal performances, resulting from larger model uncertainty and sub-optimal use of data.

The control approaches used in this thesis learn passively. However, in Chapter 23, the overall strategy can be considered to learn "actively" since it implements the design of the experiment in between runs.

4.2. Run-to-run learning & control

As we previously mentioned, at least for the first runs, repetitive processes commonly aim to optimize the performance at every repetition and adapt to changes between repetitions. This can be achieved by using the information collected in the previous runs. This problem has been extensively studied, especially in robotics and chemical engineering [32]. Here, we will review the main contributions to this topic. One of the first techniques that was developed is Iterative Learning Control (ILC). ILC aims at improving the trajectory tracking performance, i.e., achieving offset-free tracking by learning from previous batches (cf. [32, 105] and references therein). For sampled data systems, the error between reference and output for the k-th iteration is $\mathbf{e}_k = \mathbf{r} - \mathbf{y}_k$, where $\mathbf{e}_k = [e_k(1)^\top \ldots, e_k(N)^\top]^\top$ and the input $\mathbf{u}_k = [u_k(1), \ldots, u_k(N-1)]$ where N is the number of samples for every run. The idea is to update $\mathbf{u}_k$ such that $\|\mathbf{e}_k\| \to \mathbf{0}$ as $k \to \infty$. The update algorithm can be written in general as

$$\mathbf{u}_k = r(\mathbf{u}_{k-1}) + Q(\mathbf{e}_{k-1}), \tag{4.1}$$

where r is called *updating law* and Q is the *Q-filter*. Both depend on the ILC tuning techniques used [32]. The open-loop control sequence $\mathbf{u}_k$ is applied to the plant. Often, ILC is coupled with a feedback controller to reject non-repeating disturbances (cf. e.g.[49]). This basic formulation was adapted to the model-based formulation that uses model plant inversion [7, 104]. ILC is usually applied to linear or linearized models in discrete time, but extensions to continuous-time, nonlinear systems are available (cf., e.g., [204, 40, 203]). Asymptotic stability can be guaranteed with some conditions on the filter matrix and, in some cases, also in the presence of non-repeating perturbations. For repetitive

[3]Note that this concept is related to the concept of certainty-equivalent control.

processes, whose trajectories, dynamics, and disturbance do not change much between iterations, ILC can be a valid control strategy to use in combination with a feedback control strategy. Non-repeating disturbances and noise can be detrimental to the performance of the ILC. For this, it is recommended to couple the ILC control algorithm with a closed-loop controller.

A run-to-run learning approach using MPC was presented in [163]. In this case, it is assumed that the reference trajectory is not fixed, but the initial conditions are the same at every run. It is proved that the cost of the j^{th} batch does not increase with respect to the batch $j-1$, state and input constraints are satisfied, and the closed-loop equilibrium is asymptotically stable. An extension considering a variable initial condition was proposed in [33]. In both cases, the model dynamics are assumed to be known. In [184] an MPC is run online for every batch with a relatively short prediction horizon. Between batches, offline, an MPC with a longer prediction horizon is run to obtain more insight into the plan of the next batch; a shaping function that enters the objective function is learned using a neural network in such a way that the control sequence of the MPC solved online is similar to the one solved offline. For new processes and scale-ups, it is often the case that not only is the model uncertain, but also the optimal state and input profile. This has been investigated in [76, 73, 108] where the model parameters were fitted to the new data. Model adaptation through parameter estimation that only aims to minimize the error between measured and predicted measurements can sometimes lead to bad performance due to a wrong assumption on the structure of the model. This is important because due to significant model mismatch, especially when only a few runs are available, the gradient of the plant and the OCP can point in a different direction, leading to a slow convergence to the optimal or, worst case, to a divergence. This problem was considered in [117], where parameters were adapted so that the difference between measured and modeled gradients is minimized. This approach is interesting but has two disadvantages: it relies only on parameter estimation and requires to excite and measure the plant to obtain a guess on the gradient. This might lead to a prohibitive number of measurements. The main drawback of the previous approaches is that structural model uncertainty is not considered. Reinforcement learning (RL) has recently received interest also in the process engineering community for run-to-run optimization [147, 148]. This can take model and measurement uncertainty into account and has a dual-control effect (see Section 15.2). However, there are two main disadvantages: Constraints cannot be easily implemented, and a large number of measurements or a good model to initialize the control policy is required. Those requirements are hard to satisfy, especially for new processes, scale-ups, or when a limited amount of measurements is available. An RL algorithm was combined with ILC in [205] where it was shown that the ILC change in references can be tracked better than ILC alone, and learning efficiency is higher for the same number of runs.

Optimal control approaches based on neural network models for batch processes were proposed in [51, 210, 211, 157, 202], where the data of every batch was used to train and improve the neural networks. In these contributions, no considerations of safety were made. In the following sections, we review some important approaches that consider uncertainty, some of them guaranteeing safety.

4.3. Safe learning & Risk-aware learning

In the context of learning-supported and learning-based model predictive control, with the term *safe learning*, we indicate all the approaches that execute the control and learning task online or offline in a way that the system constraints are guaranteed to be satisfied with a given probability and the problem is guaranteed to remain recursively feasible *at all times*. Instead, with the term risk-aware (or risk-sensitive) learning, we indicate approaches that inform the controller on the risk of modeling error: In this case, risk can be decreased, but constraints satisfaction and recursive feasibility are not necessarily guaranteed.

Recently, safe learning has received much attention in the control community [75]. A class of methods is based on the *barrier function* concept. In this case, there are two controllers: A performance-oriented controller used inside a safe set and a safe controller activated only when the system crosses the borders of a safe set and guarantees that the system returns to the safe set. The concept was developed in [199] then used for safe learning in [194, 185, 41, 42]. Another class uses *safety filters* [99, 195, 196, 46]. In this case, the (possibly unsafe) input signal is passed through a *filter* that modifies the input as little as possible so constraints are satisfied. For example, some approaches [99, 196] propose to use reinforcement learning to increase the performance of a possibly complex, discontinuous system by learning a control policy: Since reinforcement learning approaches cannot directly consider constraints, an MPC controller could be used as a filter. For example, the objective function of the MPC could look like this:

$$\begin{aligned}&\min_{u_{\text{safe}}} \|u_0^{\text{safe}} - u_0^{\text{perf}}\|_R^2\\ &\text{subject to } (21\text{b}) - (21\text{d}).\end{aligned}$$

where the terminal constraint is the safe set, u_0^{safe} and u_0^{perf} are the first pieces of the backup and performance-oriented input sequence, respectively. If necessary, this will modify the proposed control input to satisfy the constraints while remaining as close as possible to the proposed optimal input. In [99], the performance-oriented control input and the safe control input are calculated simultaneously, which allows to force the first performance-oriented input of the sequence to be equal to the safe control input. Furthermore, the MPC considers a safe set in the constraints. This provides a control sequence that *could* bring the system into a safe set at all times. For example, consider an autonomous car aiming to minimize lap time. While reinforcement learning would try to decrease the lap time, possibly taking very aggressive maneuvers, the MPC can ensure that, at all times, the car can break and follow the center of the track safely [99]. A further advantage of the previous approaches is that they can be applied to existing machine learning-supported controllers without modification of the existing controller. A disadvantage is that it is often assumed that the nominal model is good enough and can be used by the filter. Other approaches try to add direct constraints during the learning process, for example, in reinforcement learning (RL) [146, 139] or GPs [122]. In the case of RL, adding constraints is non-trivial. Since they solve a stochastic problem, the constraints must be approximated with a certain probability, and then they are usually added to the objective function. In another class of methods, the system is not allowed to leave a *safe sub-region* of the state space: By collecting data, the safe sub-region expands as the information on the unknown model dynamics increases [20, 129].

Other (more conservative) approaches are based on the separation between performance and safety. These consider the learned dynamics to be disturbances and use classic robust MPC theory. For example, the system

$$\dot{x} = f(x,u) + B_w^{\mathsf{T}}\rho(x,u) \tag{4.2}$$

where $\rho(x,u)$ is an unknown part and $f(x,u)$ is known. If an upper bound on $\rho(x,u)$ is known (e.g., for all x and u belonging to some sets), tube MPC approaches can be used. For example, in [11], a tube-based MPC was used for a discrete-time system of the form

$$x_{k+1} = Ax_k + Bu_k + \rho(x_k,u_k), \tag{4.3}$$

where the unknown part $\rho : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ (possibly nonlinear) was considered as bounded system disturbance $d \in \mathcal{W}$, i.e.,

$$x_{k+1} = Ax_k + Bu_k + d. \tag{4.4}$$

While the nominal model $\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k$ was used to guarantee robust constraint satisfaction, the learned dynamics 28 was used in the objective function to increase performance. This approach guarantees constraint satisfaction *regardless of the machine learning model*. This was achieved by treating the machine learning part as a disturbance and finding an upper bound on its value *for all admissible state and input values*. Hence, the trained machine learning model did not enter the construction of the tube nor the constraint tightening. This has the advantage of separating learning and constraint satisfaction but comes at the price of conservativeness. In fact, if $\rho(\cdot,\cdot)$ takes too large values, the resulting tube MPC problem could be infeasible. The previous idea was also used in [213], where a neural network was used as a feedforward controller in parallel with an MPC. The neural network's output layer was built so that the output value was bounded. This allows us to know the maximum error that the neural network can introduce, which consequently can be used in a tube-based MPC approach. In [214] instead, an output-bounded NN is used to learn part of the system dynamics. Also in this case tube-based MPC was used to guarantee constraint satisfaction.

In [23], the idea of [11] was expanded to systems with multiple operations modes with uncertain switching times among the modes. In this article, constraint satisfaction despite model switches and learning was proven. The previous approaches can be conservative because they do not consider that the uncertainty varies with state and input, and they use an upper bound *for all* states and inputs. In [174], the work of [11] is expanded by considering state-varying disturbances. This method needs a description of how the uncertainty bound varies with the states, which can be challenging to obtain for specific systems. In Chapter 23, we propose a method that does not require this description but progressively finds a local bound on the uncertainty from data. As already highlighted, the main advantage of the previous methods is that they ensure constraint satisfaction. Nevertheless, there are some disadvantages. Firstly, one crucial assumption of these methods is that at least an upper bound on the uncertainty is known. If the upper bound is too large, it is possible that no feasible solution can be found. Secondly, the methods are often complex and require expert knowledge to be applied in practice. The already mentioned risk-aware approaches do not necessarily ensure safety but are usually more straightforward to develop. In [186], a risk-aware run-to-run approach was proposed using a hybrid modeling approach. In this contribution, the risk associated with extrapolation of the NN was modeled using a number of radial exponential functions. These functions heuristically model the uncertainty so that the risk increases far away from the measurements used for training. A constraint on the maximum accumulated risk is imposed; hence, the controller tries to avoid areas with large risks. In Chapter 18, we use a similar approach but with a more rigorous uncertainty description derived from Gaussian processes.

4.4. Summary

This chapter provided some basic information and definitions concerning machine learning-supported MPC. The concept of active and passive learning was covered. We furthermore summarized some contributions in the area of robust and risk-aware learning supported

MPC and highlighted their strengths and weaknesses. The goal of this chapter was to highlight the critical points and research areas that are still open, which will be addressed in the following chapters.

5. Risk-aware optimization and Model Predictive Control

This chapter presents an approach that informs an open-loop optimal controller and a shrinking horizon economic MPC of the risk of model error associated with hybrid models through a risk map. The risk map is built using an ensemble of Gaussian processes, which assigns a certain degree of risk to each point of the feature space. The hybrid model can contain any machine learning algorithm. Once this risk map is built, the user can set the maximum risk that the control is allowed to take. In practice, this is a tuning parameter that balances exploration and exploitation. We show that the risk of the hybrid model decreases from run to run as the number of data points increases. The method is applied for the simulations of two different bioreactors. The results presented in this chapter are partially based on [131].

5.1. Introduction

The previous chapters highlighted the importance of considering model uncertainty in model predictive control approaches. We also stressed that this uncertainty is even more critical for repetitive processes since the models must be valid over a wide range of conditions despite the often very limited process knowledge. Nevertheless, one can exploit the repetitive nature of the process and try to increase its performance in consequent runs by paying the price of lower performances in the first runs. Following this line, the objective of this chapter's approach is twofold: We want to repeatedly improve the process performance by improving our knowledge of the plant at every run while avoiding areas where model uncertainty is significant. For this, we use concepts and strategies of run-to-run learning and risk-aware control. Here, the term risk is used to indicate the model uncertainty due to lack of knowledge (due to the absence of observations) or randomness (due to noisy data), i.e., *epistemic* and *aleatoric* uncertainty. In our approach, the risk function will not distinguish between aleatoric and epistemic uncertainty, rather reflects the overall effect of uncertainties. To keep the decision process simple, we want

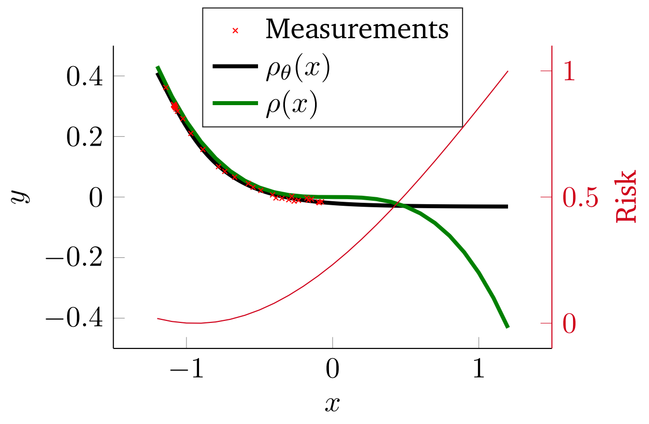


Figure 5.1.: Results showing a trained NN (ρ_θ) over data generated with the function $\rho(x) = -0.25x^3$. The measurements are affected by random noise. In this case, the risk is computed using the standard deviation of a GP trained on the same data normalized between [0,1].

to build a scalar-valued risk function that takes the features and returns a scalar between 0 (no risk) and 1 (maximum risk), i.e.,

$$f_r : \mathbb{R}^{n_v} \to [0,1] \subset \mathbb{R},$$

If the risk is high (close to 1), a significant error could affect the predictions. To explain this, take, for example, the function $\rho(\cdot)$ (green line) plotted in Fig. 11. The function is unknown, and it is learned with a machine learning model $\rho_\theta(\cdot)$ (black line). The figure shows that $\rho_\theta(\cdot)$ approximates the real function close to the training data due to the low measurement errors, but it extrapolates badly far away from them. As a measure of risk, let us take the standard deviation of a GP trained on the same data and scale it from 0 to 1. With this choice, we obtain the red line that nicely correlates to the modeling error. Using this information in an MPC approach could be helpful to avoid entering areas where the risk is high.

The question that arises now is how to build such a risk function for multi-dimensional cases, i.e., where the machine learning model has multiple inputs and outputs. In [186], a heuristic risk function was used: This was based on ellipsoids whose dimensions depended on the number of points and some tuning parameters. Here, we use the more rigorous Bayesian framework to represent the risk: The standard deviation of an ensemble of Gaussian processes measures the risk and is integrated as a constraint in the optimal control problem. Furthermore, in [186], open-loop optimal control was used. This has two main drawbacks: There is no feedback that counteracts the disturbances, and the real risk can be much higher than the predicted risk, de facto violating the maximum risk the controller was allowed to take. Hence, here we also use Model Predictive Control.

Here, we will use a hybrid model containing a neural network, but any other regression model can be used. In this way, we equip neural network prediction with a measure of uncertainty. The neural network is trained at the end of every run using the data collected in all the previous runs. Furthermore, since the computation associated with GP training scales poorly with the size of the dataset, we show how, if necessary, one can train a set of GPs on subsets of the data to reduce the computation load.

5.2. Problem setup

5.2.1. Open-loop optimization

Since online measurements might not be available for some applications, like biotechnological processes, we will first consider an open-loop optimization and then expand the approach to a shrinking horizon MPC to cover the case where it is possible to measure online. We consider the series hybrid model structure (cf. Section 7). At run i, the following nonlinear optimization problem is solved

$$
\begin{aligned}
\max_{\bar{u}(\cdot)} \quad & L(\bar{u}(\cdot), \bar{x}(T_f)) && \text{(5.1a)}\\
\text{s.t.} \quad & \dot{\bar{x}}(t) = f(\bar{x}(t), \bar{u}(t), \rho_\theta^i(\bar{v}(t))), \quad \bar{x}(0) = \tilde{x}(0), && \text{(5.1b)}\\
& r = f_r(\bar{v}(t); \mathcal{D}^i), && \text{(5.1c)}\\
& I(r) \leq I_{\text{max}}, && \text{(5.1d)}\\
& \bar{x}(t) \in \mathbb{X}, \bar{u}(t) \in \mathbb{U}, && \text{(5.1e)}\\
& \text{for} \quad t \in [0, T_f],
\end{aligned}
$$

where $L(u(\cdot), x(T_f))$ is an economic function to be maximized, (30b) is the hybrid model, $\mathcal{D}^i = \{(\hat{v}_1, \hat{l}_1), (\hat{v}_2, \hat{l}_2), \ldots, (\hat{v}_{n_\mathcal{D}^i}, \hat{l}_{n_\mathcal{D}^i})\}$ is the dataset of measured features $\hat{v}$ and labels $\hat{l}$ that have been collected up to the previous run $i-1$, $f_r : \mathbb{R}^{n_v} \mapsto \mathbb{R}$ is the risk function, $I : \mathbb{R} \mapsto \mathbb{R}$ is the risk constraint (as we will see in Section 21), T_f is the final time of the run and $\tilde{x}(0)$ is the vector of measured (or estimated) initial conditions. Note that $\mathcal{D}^i$ enters the risk function $f_r(\cdot)$ as parameter. At every time step, the states and inputs are constrained into $\mathbb{X}$ and $\mathbb{U}$ respectively. Note that, before the i^{th} run, $\rho_\theta^i(v(t))$ is trained on the accumulated data set $\mathcal{D}^i$. This improves model precision and leads to better performance as the number of runs increases.

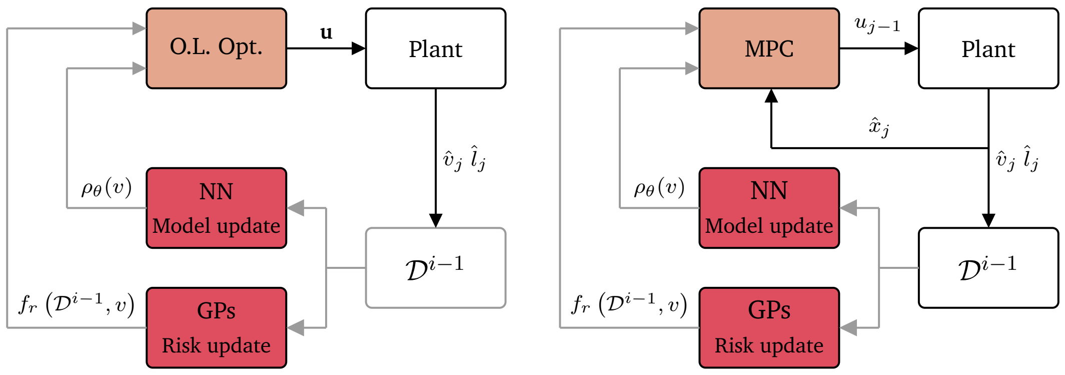


Figure 5.2.: Schematic idea of the approach. The controller is either an open-loop (OL Opt.) or a shrinking-horizon MPC (MPC). The gray arrows are active only at the end of every run.

5.2.2. Shrinking horizon MPC

The extension to a shrinking-horizon Model Predictive Control formulation is straightforward if the necessary state measurements or estimates are available online. The shrinking-horizon MPC consists in re-evaluating the previous open-loop optimization at discrete sampling times $t_k = k\Delta t$, where Δt is the sampling time and $k \in [1, ..., N]$, $N = \text{ceil}(T_f/\Delta t)$. Here, the sampling times are assumed to be uniform for simplicity, but non-uniform sampling times can also be used. The shrinking-horizon MPC problem reads

$$\begin{aligned}
\max_{\bar{u}(\cdot)} \quad & L(\bar{u}(\cdot), \bar{x}(T_f)) && (5.2a)\\
\text{s.t.} \quad & \dot{\bar{x}}(t) = f(\bar{x}(t),\ \bar{u}(t), \rho_\theta^i(\bar{v}(t))), \quad \bar{x}(t_k) = \tilde{x}(t_k), && (5.2b)\\
& r = f_r(\bar{v}(t); \mathcal{D}^i), && (5.2c)\\
& I(r) \leq I_{\text{max}}, && (5.2d)\\
& \bar{x}(t) \in \mathbb{X}, \bar{u}(t) \in \mathbb{U}, && (5.2e)\\
& \text{for} \quad t \in [t_k, T_f].
\end{aligned}$$

where $\tilde{x}(t_k)$ is the measured or estimated state at time t_k.

Figure 12 shows the control loop of both the open-loop and shrinking horizon approach. Note that although we used an economic objective function here, other objective functions can be used, e.g., to track a time-varying state or output trajectory.

5.3. Risk function

The risk function in this study is constructed using Bayesian inference, with Gaussian processes (GPs) being utilized, as explained in Section 6.2. The key concept is to utilize the covariance matrix of the posterior distribution as an indicator of prediction uncertainty. At the same time, a neural network is employed for the mean (or nominal) prediction. It may be argued that using a neural network for nominal prediction is unnecessary since GPs already provide a predicted mean. However, there are two reasons why incorporating a neural network can be advantageous.

Firstly, in GPs, the prior mean is typically initialized as zero across the entire feature space. This initialization is justified as the posterior mean is expected to be close to the measured outputs. However, the mean converges to the prior mean in regions far from the measurements, which can be inappropriate for describing unseen data if the prior mean is improperly chosen. Recent research has shown that selecting a non-zero prior mean can improve predictions in regions far from the measurements [85].

Secondly, Gaussian processes require inverting the covariance matrix during training, and the computational cost of this operation is proportional to the number of measurements. Inverting large covariance matrices can be computationally challenging, particularly for large datasets. One approach to address this issue is to divide the dataset into smaller subsets and train a GP on each subset. However, merging the mean estimates of multiple GPs to obtain a single prediction can be non-trivial and may result in discontinuous predictions. On the other hand, neural networks can be trained efficiently with very large datasets without the need for data splitting. Therefore, in this study, local GP models trained on smaller datasets are used to estimate local uncertainties, as it is easier to merge uncertainty measures smoothly compared to mean predictions. This allows for predictions with associated uncertainties across the entire feature space.

Note that $\rho_\theta : \mathbb{R}^{n_v} \to \mathbb{R}^{n_l}$ and $n_l \geq 1$ hence there can be more than one output dimension. Since training GPs with multiple outputs is challenging [200], as common when using GPs, we will assume that the outputs of the unknown functions are independent of each other. This will allow us to train n_l independent GPs, one for every output.

Assumption 5.1 *The outputs of the unknown function $\rho(\cdot)$ are independent of each other.*

Uncertainty averaging We now proceed to build the risk function. Let $\bar{K}_j$ be the posterior variance matrix of the j^{th} output (cf. Section 6.2) and K_j^∞ be the variance far away from the training points for the same output, i.e. $K_j^\infty = \lim_{v\to+\infty} K_j(v)$. Note that given a point v, $\bar{K}_j(v) \leq K_j^\infty$. Hence, we propose to use the ratio between the two variances as a measure of uncertainty as follows

$$\bar{r}(v) = \left(\prod_{j=1}^{n_l} \frac{\bar{K}_j(v)}{K_j^{\infty}} \right)^{1/n_l} , \tag{5.3}$$

where the geometric mean was used to average among the n_l outputs. Note that $\hat{r}(v)$ takes values between 0 and 1. The closer its value is to 1, the riskier the point v is.

Clustering As we previously mentioned, training the GP can be computationally expensive in the case of large datasets. Here, we want to show how it is possible to train multiple GPs on a finite number of clusters and then use the uncertainties of the single clusters to build an overall risk measure. Unsupervised clustering can be used to obtain the subsets of data. In our case, we used k-means clustering from scikit-learn [145]. The number of clusters is a design variable. Let n_c be the number of clusters, we compute the risk of every cluster by using (32), hence obtaining $\bar{r}_j : \mathbb{R}^{n_v} \mapsto \mathbb{R},\ \ \forall j \in [1, .., n_c]$ (cf. Figure 13).

Risk function Equation (32) is computed for every of the n_c cluster. Given a distance definition, one could choose the risk function of the cluster that is closer to the current v. Here, for simplicity, the final risk function merges all the clusters by taking the minimum as follows

$$r(v) = \text{min}(\bar{r}_1(v), \bar{r}_2(v), \bar{r}_{n_c}(v)). \tag{5.4}$$

where $r : \mathbb{R}^{n_v} \to [0, 1]$. Since the minimum operator is not differentiable everywhere, it could lead to some numerical problems during the optimization. For this reason, we used the smoothmin operator, which is a differentiable approximation of the min operator defined as

$$\text{smoothmin}(x_1, ..., x_n) = \frac{\sum_{i=1}^{n} x_i e^{\alpha x_i}}{\sum_{i=1}^{n} e^{\alpha x_i}} \tag{5.5}$$

where $\alpha < 0$ is a large negative value that can be chosen by the user. Figure 13 shows graphically the steps behind the construction of the risk function. Once it is available, it can be used in the risk constraint. One trivial choice of risk constraint is

$$I(r) = 100\ \ r(v(t)) \leq I_{\text{max}}. \tag{5.6}$$

Alternatively, in some cases, one could limit the accumulated risk along the batch to avoid small but steady errors. In this case, the risk constraint $I(r)$ can be defined as

$$I(r) = \frac{100}{T_f} \int_0^{T_f} r(v(t))dt \leq I_{\text{max}}. \tag{5.7}$$

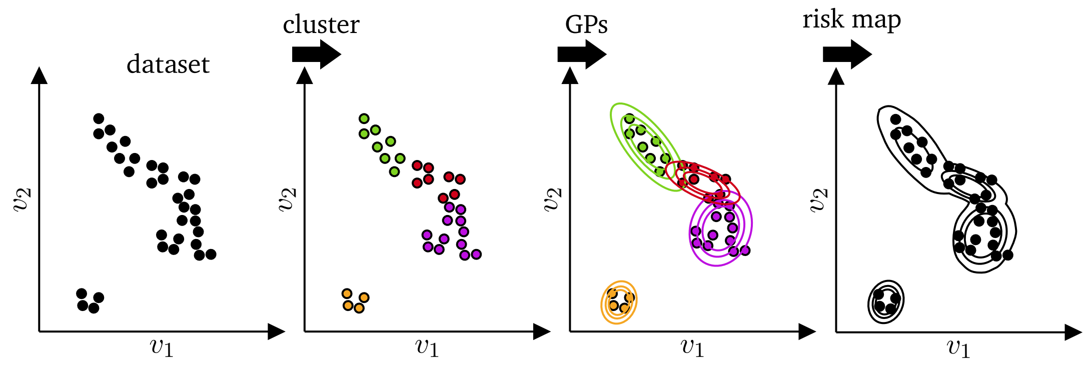


Figure 5.3.: Graphical representation of the risk function for a problem with two features v_1 and v_2. The dataset is divided into $n_c = 4$ clusters with an unsupervised clustering technique. Later, a Gaussian process is trained for each cluster. The risk (32) is computed for every cluster, and finally, it is fused with the other cluster with (33). Note that the contour lines represent a decreasing level of risk.

The first choice will limit the instantaneous risk, while the second limits the accumulated risk. Note that we multiplied by 100 hence $I(\cdot)$ takes values between 0 and 100 %. If we choose $I_{\text{max}} = 100\%$ in (31d), it is equivalent to solving (30) without any risk constraint. Note that other definitions of risk functions are possible (see, e.g., [186]) and that its choice may depend on the characteristics of the problem being solved.

Remark Note that the exploration-exploitation trade-off is not explicitly considered here, as we do not actively try to excite the system to obtain information at the cost of performance, e.g., in dual control. The proposed approach rather considers the risk of exploring possibly risky areas, hence prudently approaches them when necessary, i.e., when the direction of improvement of the objective function in (30) points towards them. Hence, the learning is *passive*.

5.4. Examples

The following sections show some examples that use the proposed method. The first example considers a fed-batch reactor producing β-glactosidase. The second is also a fed-batch reactor that produces poly-β-hydroxybutyrate (PHB). In both cases, an economic open-loop optimal control problem and an economic shrinking horizon MPC are solved, and the results are compared. The objective functions used are economic indexes of the processes. In the first case, the economic index is the profitability, representing the process's normalized profit (or loss) at the end of every run. In the second case is productivity, defined as grams of product produced at the end of a run. We will refer to both economic indexes with the variable Pr. All the examples were solved using HILO-MPC [150]. GPyTorch [67] was used to train the GP, and PyTorch [142] for training the NN.

5.4.1. Key Performance Indicators

We want to study the effect of different I_{max} on Pr, and how Pr changes from batch to batch. We expect that the controller might take sub-optimal control actions for high-risk levels, causing a slower increase or even a decrease of Pr in successive runs. In contrast, a lower risk level will constrain the controller to take very conservative control actions, limiting the exploratory effect and consequently reducing possible gains in Pr. To compare the effects of the risk, four *key performance indicators* (KPIs) are considered:

- Pr_{max}: Maximum Pr. This is the maximum achieved economic index for a given I_{max},

$$\text{Pr}_{\text{max}} = \max_{i=[1,\dots,n_r]} \text{Pr}^i.$$

- Pr_{avg}: Average Pr per run for a given I_{max},

$$\text{Pr}_{\text{avg}} = \frac{1}{n_r}\sum_{i=1}^{n_r} \text{Pr}^i.$$

- $\text{N}_{\text{Pr,min}}$: Number of batches necessary to achieve a minimum required economic index Pr_{min} for a given I_{max},

$$\text{N}_{\text{Pr,min}} = \min\{i \in [1,\dots,n_r] \subset \mathbb{N} \mid \text{Pr}^i \geq \text{Pr}_{\text{min}}\}.$$

- C_{Pr}: Number of batches necessary to achieve convergence between predicted and measured profit. This is only calculated for the open-loop case, and indicates the prediction accuracy of the hybrid model, i.e.,

$$\frac{|\text{Pr}^i - \overline{\text{Pr}^i}|}{\text{Pr}^i} \leq c_{\text{tol}}, \quad \forall i \in [C_{\text{Pr}},\dots,n_r].$$

  where $\overline{\text{Pr}}$ is the predicted Pr.

The number of clusters chosen for the GPs training is initialized as one and increases by one at every run. The clusters are re-computed at every batch once new data is available.

5.4.2. Case Study I: Production β-galactosidase

In this case study, we want to maximize the profitability of a fed-batch β-glactosidase production process using a recombinant *E. coli* D1210 [107, 188]. The plant model is given by the following ODE

$$\dot{X} = \mu X - DX, \tag{5.8a}$$
$$\dot{S} = -R_s(x)X - DS + F_S S_F/V, \tag{5.8b}$$
$$\dot{P} = R_{\text{fp}}(x)X - DP, \tag{5.8c}$$
$$\dot{I} = -DI + F_I I_F/V, \tag{5.8d}$$
$$\dot{I}_{\text{SF}} = -k_1(x)I_{\text{SF}}, \tag{5.8e}$$
$$\dot{I}_{\text{RF}} = k_2(x)(1 - I_{\text{RF}}), \tag{5.8f}$$
$$\dot{V} = F_I + F_S, \tag{5.8g}$$

where X, S and I are concentrations of biomass, glucose and inducer in [g/l], I_{SF} is the inducer shock factor, I_{RF} the inducer recovery factor in [U/l], V is the volume of the media in the bioreactor in [l], and P is the activity of β-galactosidase in [U/ml]. The vector x contains all the states i.e., $x = [X, S, P, I, I_{\text{SF}}, I_{\text{RF}}, V]$. The inputs are the glucose feeding rate F_S and inducer feeding rate F_I in [l/hr]. S_F is the concentration of glucose in the feed F_S and I_F is the concentration of inducer in the feed F_I. Furthermore, $D = (F_S + F_I)/V$. The kinetic rates $\mu(\cdot), R_s(\cdot)$ and $R_{\text{fp}}(\cdot)$ are assumed to be unknown and have to be learned from data. The real reaction rates used to simulate the plant are:

$$\begin{aligned}
\phi(x) &= \frac{0.407S}{(0.108 + S + S^2/14814.8)}, \\
\mu(x) &= \phi(x)\left(I_{\text{SF}} + \frac{0.22 I_{\text{RF}}}{0.22 + I}\right), \\
R_s(x) &= 2\mu(x), \\
R_{\text{fp}} &= \phi(x)\frac{0.0005 + I}{0.022 + I}, \\
k_1(x) &= k_2(x) = \frac{0.09}{0.034 + I}.
\end{aligned}$$

A further challenge is that the states I_{SF} and I_{RF} cannot be measured, hence and the equations (37e) and (37f) are not considered in the MPC model, which hence reads

$$\dot{X} = \mu X - DX, \tag{5.9a}$$

$$\dot{S} = -R_s X - DS + F_S S_F / V, \tag{5.9b}$$

$$\dot{P} = R_{fp} X - DP, \tag{5.9c}$$

$$\dot{I} = -DI + F_I I_F / V, \tag{5.9d}$$

$$\dot{V} = F_I + F_S. \tag{5.9e}$$

The objective function to maximize is

$$L(F_I(t), P(T_f)) = P(T_f)V(T_f) - Q \int_0^{T_f} F_I(t)\mathrm{d}t. \tag{5.10}$$

This represents the normalized profit of operation, and $Q = 5$ is the normalized cost of the inputs with respect to the value of the product P. For this process, the maximum theoretical profitability is known and equal to 4.03 [186].

Simulation settings For the open-loop and MPC cases, the batch time is 10 hours. The MPC uses a sampling time of 0.5 hours. A normal distributed Gaussian noise with a standard deviation of 0.1, 0.1, and 0.1 g/l was added to $X, S, andI$, respectively, and a standard deviation of 0.03 was added to P. A total of 20 batches were simulated with five levels of I_{max}, $10\,\%, 25\,\%, 50\,\%, 75\,\%$ and $100\,\%$. Note that $I_{\text{max}} = 100\,\%$ is equivalent to solving the optimization problems without risk constraints. An exploratory run is used to collect the first set of measurements, using exponentially increasing feeding rates. To compute the KPIs, we set the required minimum profit as $\text{Pr}_{\text{min}} = 3$g, and the convergence tolerance as $c_{\text{tol}} = 0.2$. The open-loop optimization and MPC were discretized with orthogonal collocation, and a piece-wise constant control input was considered. The constraints used are box constraints defined as

$$\mathbb{X} = \{x \in \mathbb{R}^5 \mid [0, 0, 0, 0, 0]^{\mathsf{T}} \leq x \leq [10, 42, 100, 100, 15]^{\mathsf{T}}\}.$$
$$\mathbb{U} = \{u \in \mathbb{R}^2 \mid [0, 0]^{\mathsf{T}} \leq u \leq [1, 1]^{\mathsf{T}}\}.$$

The initial conditions at every batch for the plant are $x(t_0) = [0.1,\ 40,\ 0,\ 0,\ 1,\ 0,\ 1]^{\mathsf{T}}$. The concentration of substrate and inducer in the feed are $S_F = 100$ g/l and $I_f = 4$ U/ml.

Machine learning settings We trained a single NN with two features, the substrate concentration S and the inducer I, and three outputs μ, R_s and R_fp. In this case, the feature selection was trivial since we already know the variables that influence the reaction rates from the plant model. In practice, the features are selected based on expert knowledge or by carrying out data-driven feature selection procedures, such as principal component analysis or recursive feature elimination. Contrary to [131], we use physics-informed learning to train the NN as described in Section 8. The neural network has four layers with ten neurons each. Labels and features were scaled with their respective mean values $(\mu_v^\mathcal{D}, \mu_l^\mathcal{D})$ and standard deviations $(\sigma_v^\mathcal{D}, \sigma_l^\mathcal{D})$ computed on the data set $\mathcal{D}$ as $\hat{v}_i^s = (\hat{v}_i - \mu_v^\mathcal{D})/\sigma_v^\mathcal{D}$ and $\hat{y}_i^s = (\hat{l}_i - \mu_l^\mathcal{D})/\sigma_l^\mathcal{D}$ where the s indicates the scaled variables.

The training was conducted using HILO-MPC, and PyTorch was used as a learning library. Stochastic gradient descent, using the Adam optimizer, was used as an optimization algorithm to train the NN [97] with a learning rate of 0.001. The maximum number of epochs is 3000. Early stopping with patience of 20 epochs was used to terminate the training: This means that the learning was stopped if the test loss function did not decrease in the last 20 epochs. An $80/20$ train/test split was used. As an activation function, we used the soft-plus function. The soft-plus function is a smooth approximation of the ReLU function. This choice was made since the ReLU function caused problems with the convergence of the optimization problems.

Results Figure 14 shows the difference between predicted and real profitability for the open-loop control for different I_max levels. The profitability of the exploratory experiment referred to as run 0, was also plotted for comparison. As expected, in general, the profitability increases as the number of batches increases (cf Fig. 14). Figure 20 shows the KPIs for different I_max levels. With a convergence tolerance $c_\text{tol} = 20\,\%$, the best convergence between predicted and real profitability was achieved for $I_\text{max} = 50\,\%$ (cf Fig. 20d). This can indicate that the learning is more efficient, i.e., the data collected are more relevant for the close-loop evolution of the plant.

Figure 20c shows that the number of batches necessary to reach the minimum profitability is 4 for $I_\text{max} = 50\,\%$ and increases for the other I_max levels. The average profitability Pr_avg also shows a peak at $I_\text{max} = 50\,\%$ (cf Fig. 20a). The maximum profitability Pr_max does not change considerably for the different degrees of risk (cf. Fig. 20b). Overall, for the open-loop control $I_\text{max} = 50\,\%$ has the best performance. This indicates that this value is a good balance between exploration and exploitation.

As mentioned at the beginning of the chapter, the risk function informs the controller of possible uncertainty in the output of the NN. Hence, one important question to answer is how well the risk function represents the actual NN prediction error. Figure 16 compares the risk map with the absolute errors maps of the three outputs of the NN, for the open-loop case, for the 18^{th} run with $I_{\text{max}} = 25\,\%$. To show the effect of the dataset, the measurements are also reported as red crosses. We can see that the risk value is higher in regions with little or no data available, while it is low in regions with dense data. The NN errors are also lower in the dense data region and larger for less dense data regions. Similar results were obtained when other I_{max} were used and for the MPC case. Hence, we can conclude that, in this case, the risk function condenses valuable information on the error of the NN.

If online measurements are available, the optimization problem can be solved repeatedly. Note that the network is trained only at the end of the batch and not every time a measurement is taken since adding a single measurement to the training dataset would not considerably change the network's output. Figure 15 shows the profitability with respect to the batch number for the open-loop optimal control and MPC for different levels of I_{max}. The MPC reaches high profitabilities already from the second batch (not counting the exploratory batch) for $I_{\text{max}} \geq 25\,\%$ (cf. Fig. 20c). Nevertheless, for $I_{\text{max}} \geq 50\,\%$ is showing worse profitabilities than the open-loop after the 6th run. Note that since the data collected depends on the controllers' actions in the previous runs, this result is not surprising. The fact that MPC closes the loop at the sampling points does not necessarily mean that the data collected will be more "information-rich" than an open-loop strategy. To show this, let us consider run 10 of $I_{\text{max}} = 75\,\%$. At this run, the MPC performs worse than the open-loop controller (cf. Fig. 15). Figure 17 compares the risk values and errors of the open-loop (left column) and close-loop (right column) for this run. As a comparison, an optimal trajectory of the features in the features space generated with a perfect model is plotted on top of the errors (solid red line). This trajectory is computed by solving an open-loop optimization with no model plant mismatch and achieving the maximum theoretical performance of 4.03. Hence, it can be considered a reference trajectory. As seen in the left column, the open-loop strategy (dashed red line) results in a trajectory close to the optimal trajectory. On the contrary, the MPC moves away from the optimal. This is probably due to the fact that, for the MPC case, the risk in the "optimal area" is higher, as can be seen from the contour plots. This is caused by a different dataset and, consequently, different NN outputs, as can be seen from the measurements reported in the first row for both open-loop and MPC cases. This explains why the MPC has worse performances for this run. However, one may ask why the performance of the MPC, for the same level of $I_{\text{max}} = 75\,\%$ was much higher for run 6 (where it is very close to the optimal performance). It is reasonable to think that this high performance must also be due to a good hybrid model, and since the model should not worsen with more data, the performance should remain approximately constant. If we plot the errors and risk function for this case (cf. Fig. 18) we notice that also, in this case, the trajectory of the

MPC is far away from the optimal. The higher profitability can be explained by looking at the other variables that influence the profitability, i.e., the feed of inducer F_I. As shown in Fig. 19, in this run, the MPC feeds much less inducer, which results in a lower cost and, hence, higher profitability. Nevertheless, as shown in the MPC case for $I_{\text{max}} = 50\,\%, 75\,\%$ and $100\,\%$ with this "small feed" strategy, the profitability is not reproducible at every run and has an erratic behavior. Instead for $I_{\text{max}} = 25\,\%$ the profitability was stable with the runs. This is achieved thanks to a better hybrid model, as shown in Fig. 21, representing the error and risk graph for both open-loop and MPC. As can be seen, the errors on the NN outputs are low, and the MPC follows the optimal trajectory that we obtained with the perfect model very closely.

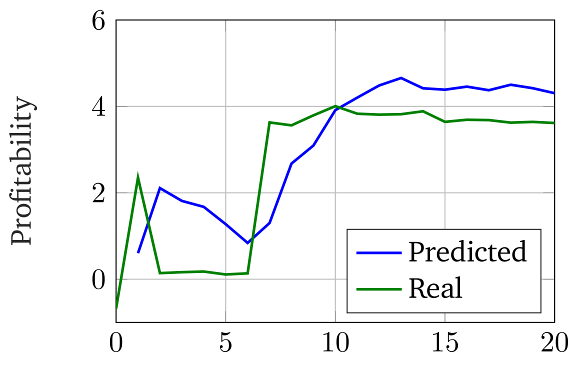
Even though the MPC performs worse than the OL in some cases, overall, thanks to the fact that the MPC is reaching higher productivity already in the first runs, the MPC shows better KPIs for most of the risk levels. as can be seen from Fig. 20.

5.4.3. Case Study II: Production of poly-β-hydroxybutyrate

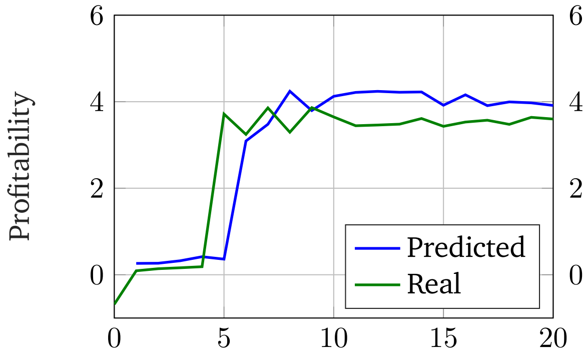
This case also considers a fed-batch bioreactor. The goal is maximizing the production of poly-β-hydroxybutyrate (PHB) using a culture of *Cupriavidus necator* [66]. This case is more challenging than the previous one. Firstly, the reaction rates are more complex, as we will see. Secondly, the NN will have three features and five outputs. The simulated plant uses a reduced version of a Hybrid Cybernetic Model [175, 65]. The ODE reads as follows

$$\begin{aligned}
\dot{X} &= \left(\mu - \frac{(F_A + F_F)}{V}\right) X, \\
\dot{A} &= q_A X + \frac{F_A}{V}(A_F - A) - \frac{(F_A + F_F)}{V} A, \\
\dot{F} &= q_F X + \frac{F_F}{V}(F_F - F) - \frac{(F_A + F_F)}{V} F, \\
\dot{P} &= q_P - \mu P, \\
\dot{V} &= F_A + F_F,
\end{aligned}$$

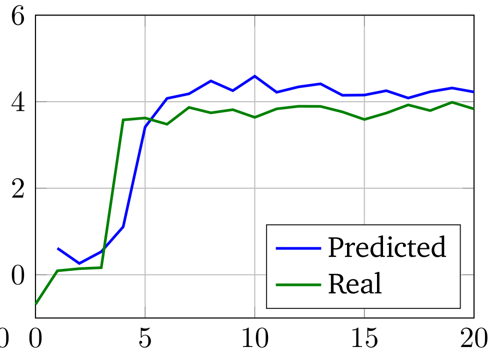
where X, A, F are respectively the total biomass, ammonium chloride, and fructose concentrations in [g/l], P is the percentage of PHB in the biomass, V is media volume in the reactor in [l], F_A and F_F the feed of ammonium chloride and fructose in [l/h], A_F and F_F the concentrations of ammonium and fructose in the feeds in [g/l]. The reaction rates q_A, q_F, q_P, and the growth rate μ are given by



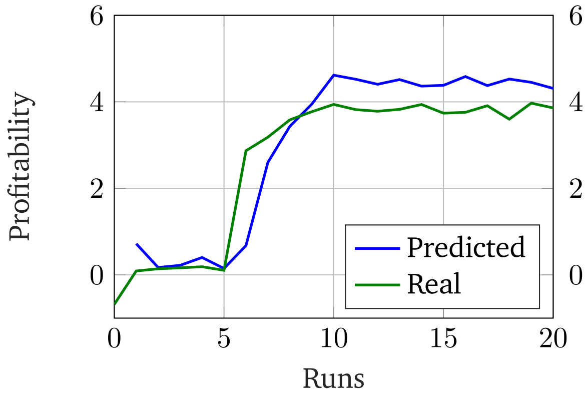
(a) $I_{\text{max}} = 10\%$



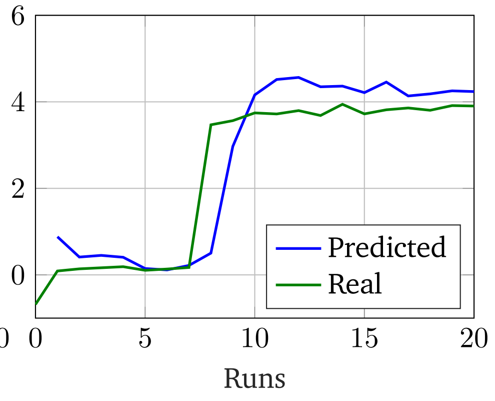
(b) $I_{\text{max}} = 25\%$



(c) $I_{\text{max}} = 50\%$

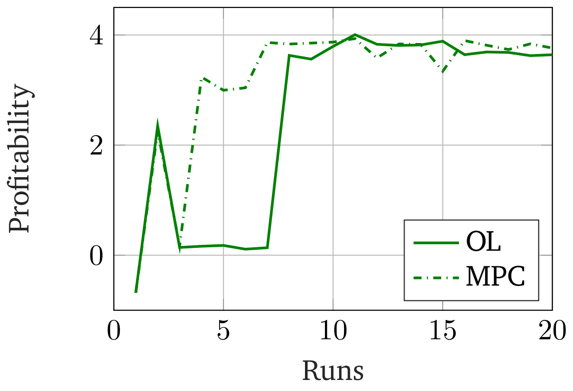


(d) $I_{\text{max}} = 75\%$

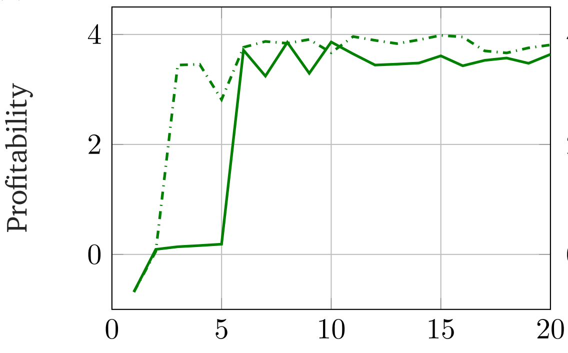


(e) $I_{\text{max}} = 100\%$

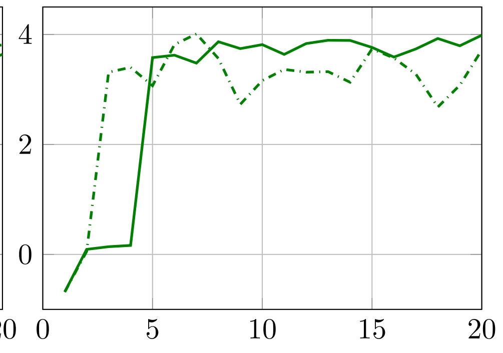
Figure 5.4.: Profitability for the β-galactosidase example for different levels of I_{max} for the open-loop optimal controller. Run 0 refers to the exploratory run done to create the first dataset.



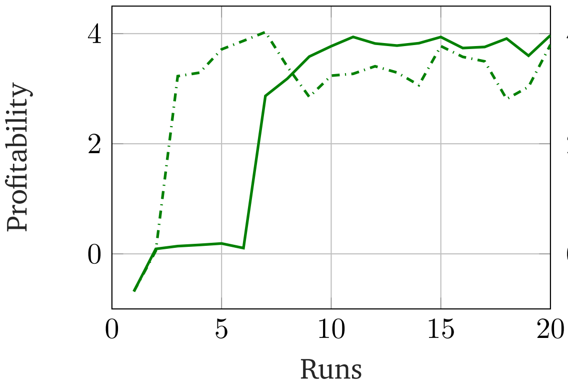
(a) $I_{\text{max}} = 10\%$



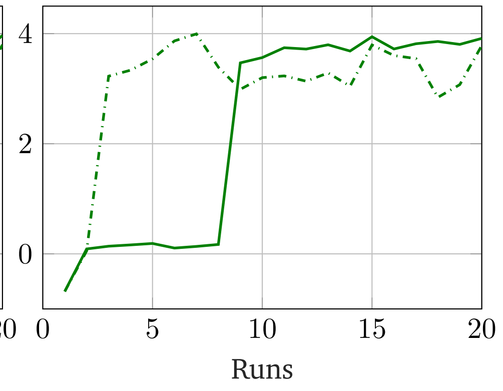
(b) $I_{\text{max}} = 25\%$



(c) $I_{\text{max}} = 50\%$



(d) $I_{\text{max}} = 75\%$



(e) $I_{\text{max}} = 100\%$

Figure 5.5.: Case study I. Profitability comparison between MPC and open-loop optimization.

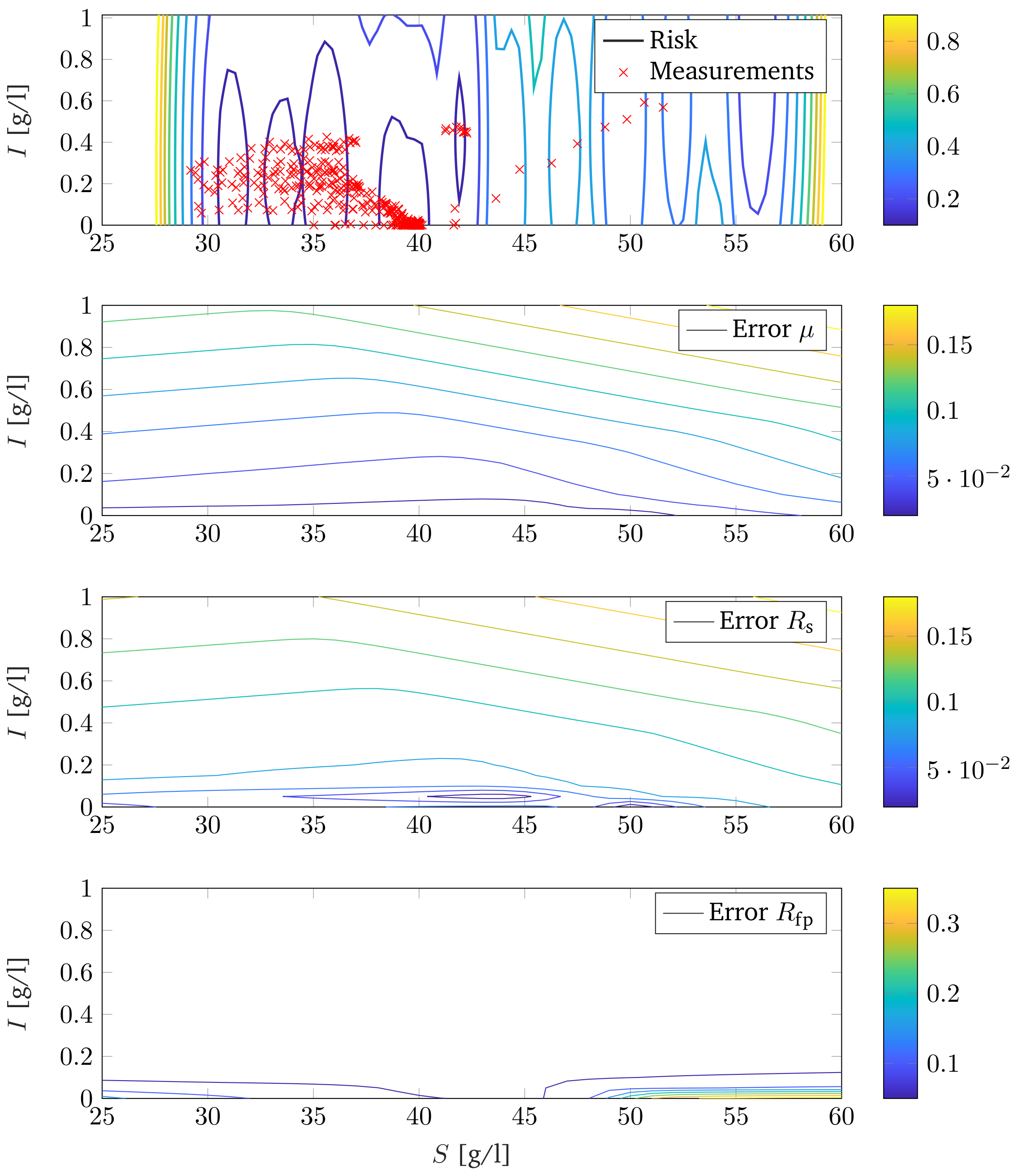


Figure 5.6.: comparison between risk values (top graph) and absolute error of μ, R_s and R_{fp} for the 18-th run with $I_{max} = 25\%$ for the open loop strategy.

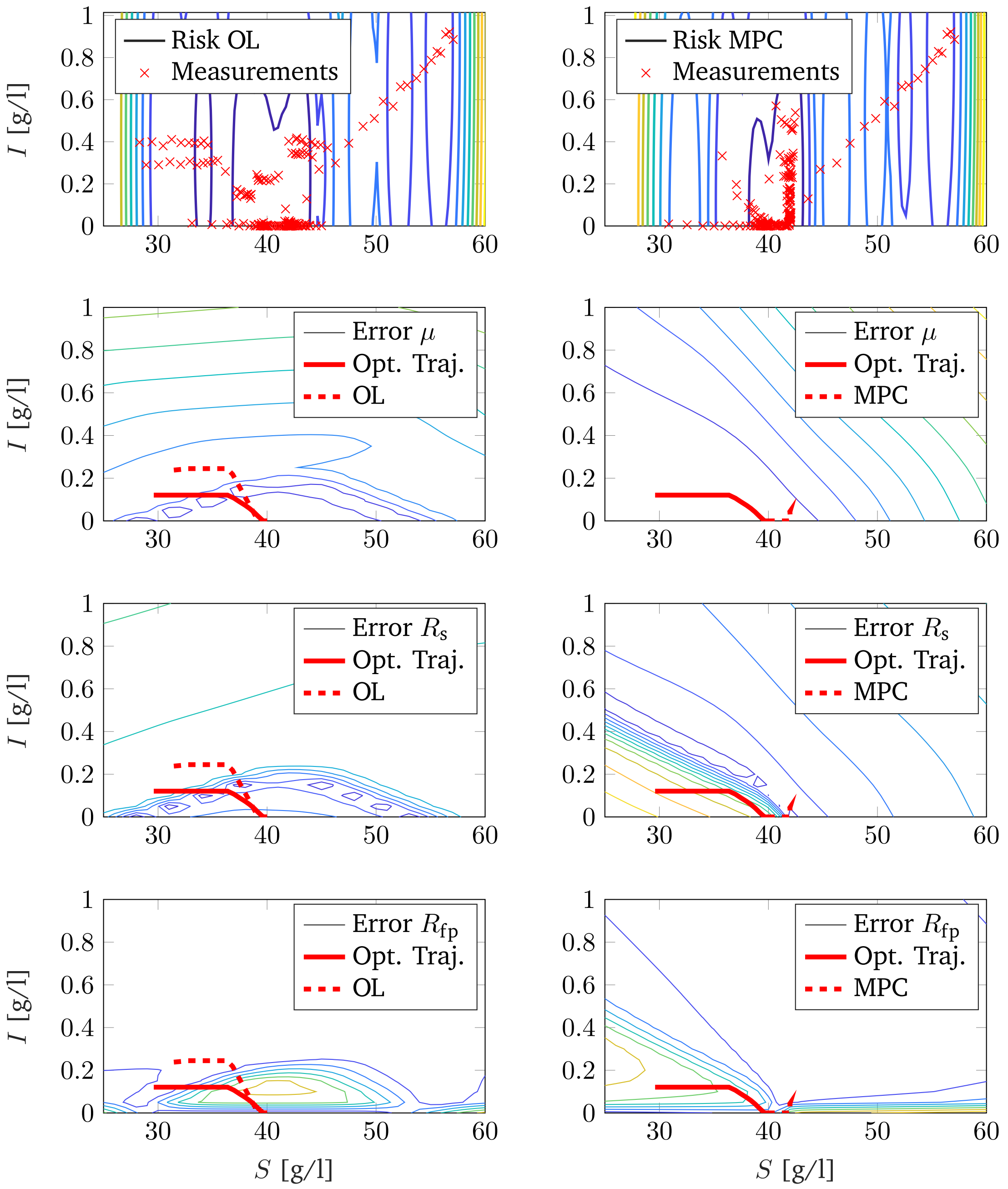


Figure 5.7.: comparison between open-loop optimization (left column) and MPC (right column) for run 10 with $I_{\text{max}} = 75\%$

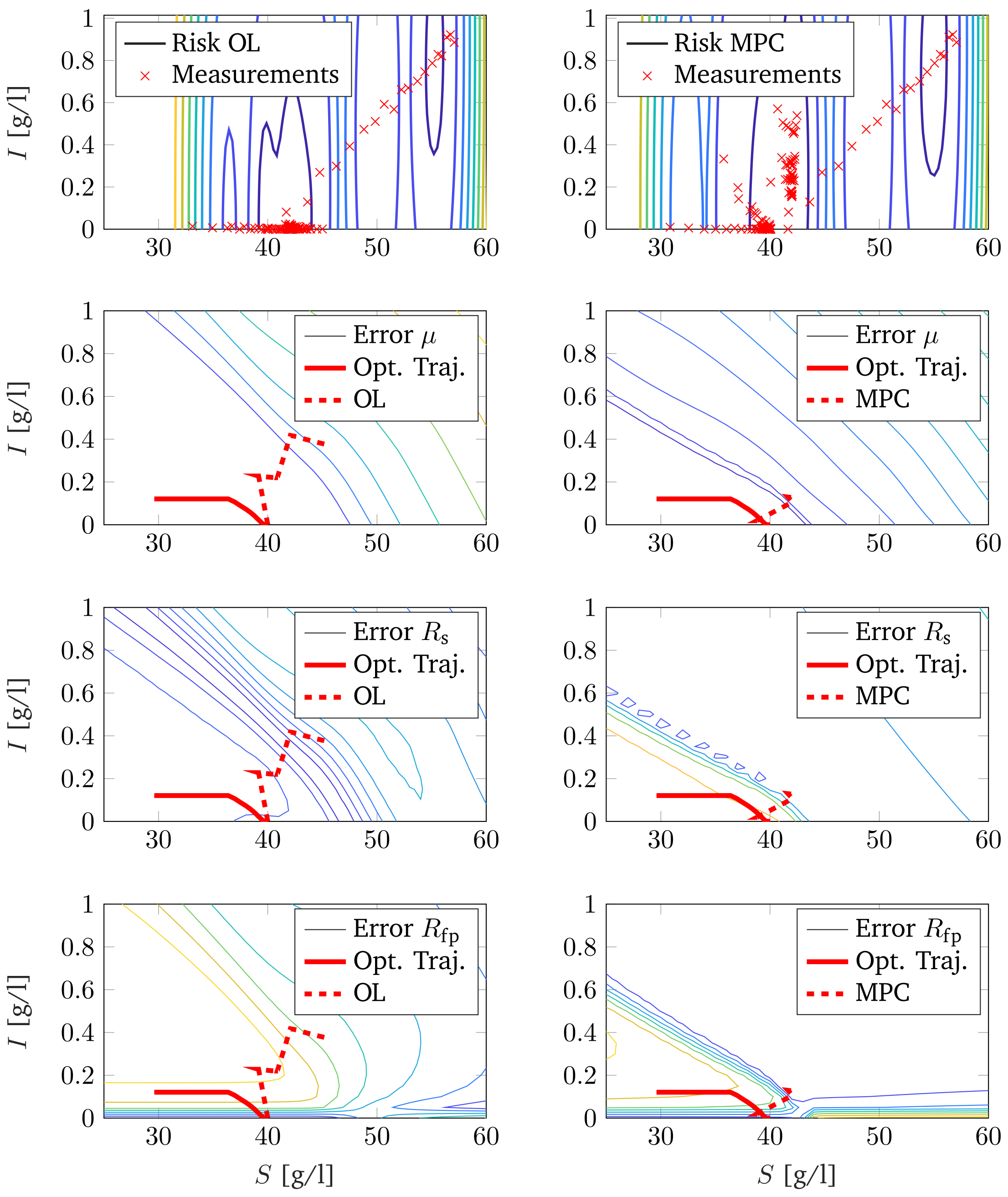


Figure 5.8.: comparison between open-loop optimization (left column) and MPC (right column) for run 6 with $I_{\text{max}} = 75\%$.

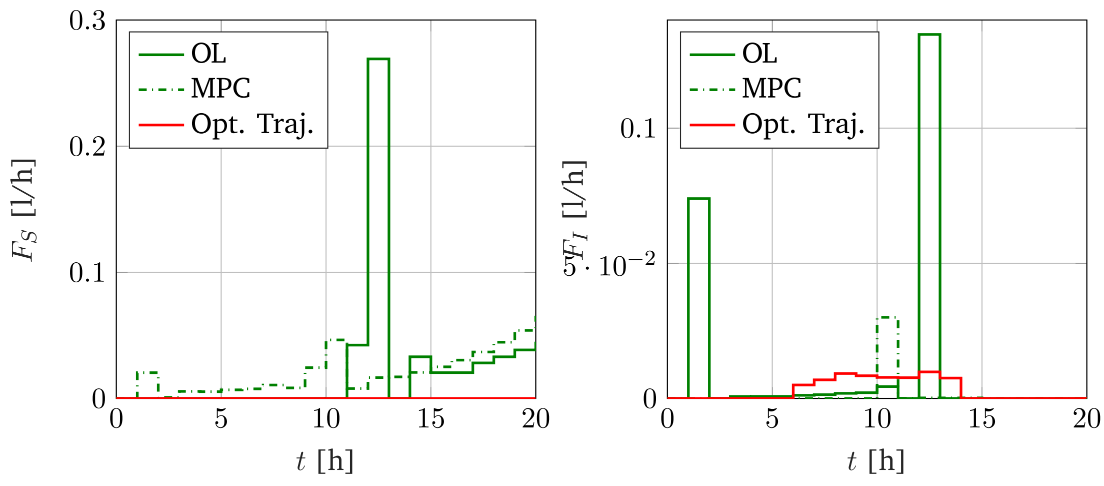


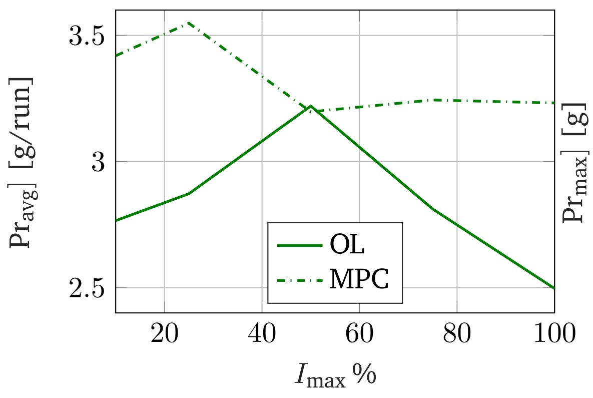
Figure 5.9.: comparison between the inputs for the open-loop control, MPC, and an open-loop with the exact model.

$$
\begin{aligned}
\mu &= S_\mu Z \rho r_M (1-P),\\
q_A &= S_A Z \rho r_M (1-P),\\
q_F &= S_F Z \rho r_M (1-P),\\
q_P &= S_P Z \rho r_M (1-P),
\end{aligned}
$$

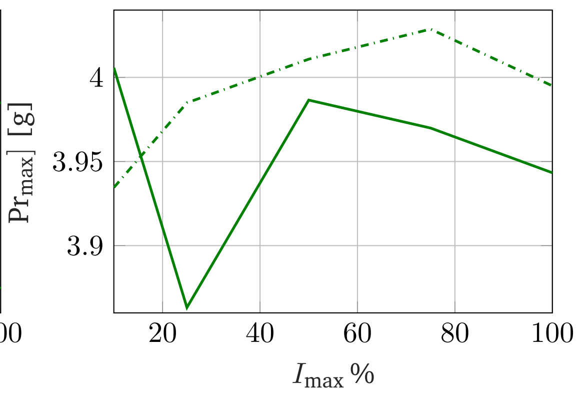
where $S_{(\cdot)}Z$ is the product between the stoichiometric matrix of the respective reactions and the elementary mode matrix, $r_M = [r_1, r_2, r_3, r_4, r_5]$ where

$$
\begin{aligned}
r_1 &= \frac{F}{K_F+F},\\
r_2 &= \frac{F}{K_F+F}\frac{A}{K_A+A},\\
r_5 &= \frac{F}{K_F+F}\frac{A}{K_A+A}\frac{P}{K_P+P},\\
r_3 &= r_4 = r_2,
\end{aligned}
$$

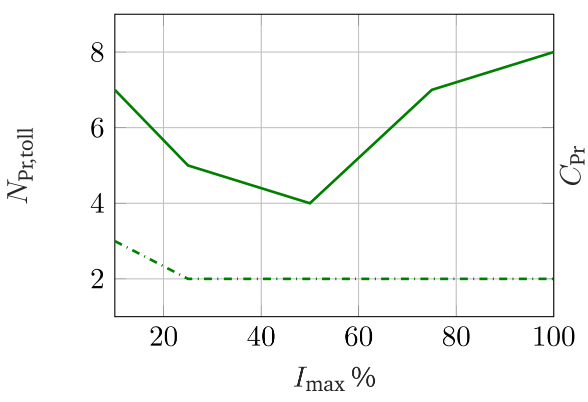
where K_A, K_F and K_P are known constant. The vector $\rho \in \mathbb{R}^5$ is unknown and must be



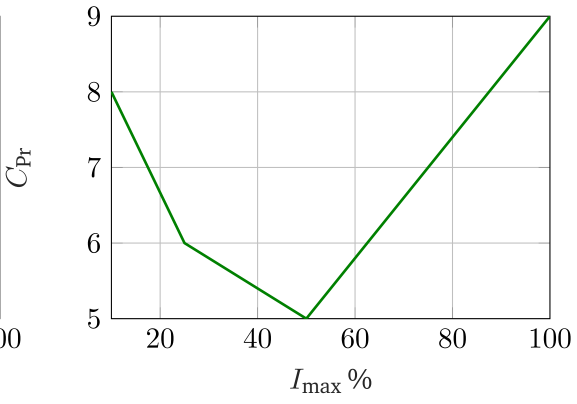
(a) Average profitability.



(b) Maximum profitability.



(c) minimum runs to minimum profitability.



(d) Runs to converge.

Figure 5.10.: KPIs of the open-loop controller (OL) and MPC. Note that the runs to converge were calculated only for the OL control since the convergence is computed with respect to the predicted profitability at $t = 0$.

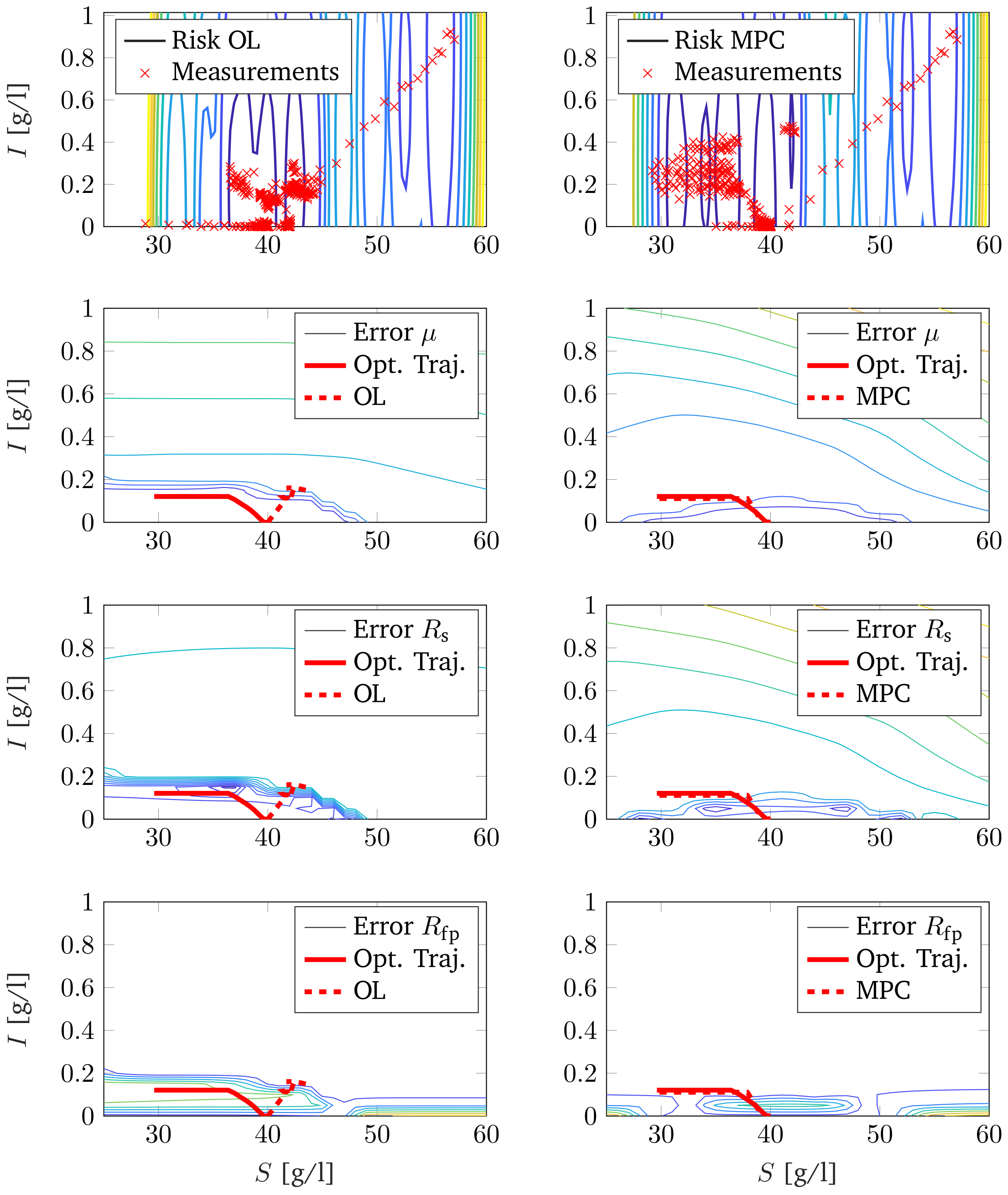


Figure 5.11.: comparison between open-loop optimization (left column) and MPC (right column) for run 15 with $I_{\text{max}} = 25\%$.

learned from data. The components ρ_i of the unknown function ρ are:

$$\rho_i = u_i v_i, \tag{5.11}$$

where u and v are the cybernetic control variables

$$v_i = \begin{cases} f_i r_i / \max(f_i r_i) & \text{if} \quad \max(f_i r_i) > 0, \\ \text{else} \quad 0, \end{cases}$$
$$u_i = \begin{cases} p_v / \sum(f_i r_i) & \text{if} \quad \sum(f_i r_i) > 0, \\ \text{else} \quad 0, \end{cases}$$

where r_i is the i-th component of r_M and f_i are constants that are taken from [65]. The objective function to maximize is the net product at the end of the batch, i.e.,

$$L(X(T_f), P(T_f), V(T_f)) = P(T_f)X(T_f)V(T_f), \tag{5.12}$$

Simulation settings The measurements of X, A, F, and P are affected by Gaussian noise with standard deviations of $0.5, 0.005, 0.002$, and 0.005, respectively. Furthermore, a 10% standard deviation was added to the labels. An initial batch using an exponential open-loop feeding strategy was used to collect the first set of measurements. A total of 10 batches were simulated, each 30 hours long. The measurements were collected every hour. The convergence tolerance $c_{\text{tol}} = 5\,\%$, while the minimum productivity Pr_{min} is 60 g/l. The constraints were defined as

$$\mathbb{X} = \{x \in \mathbb{R}^5 \mid [0,0,0,0,0]^\mathsf{T} \leq x \leq [10,10,10,1,100]^\mathsf{T}\},$$
$$\mathbb{U} = \{u \in \mathbb{R} \mid [0] \leq u \leq [1]\}.$$

Machine learning settings Machine learning settings for this study involved the use of a neural network with three layers and eight neurons per layer. The neural network was designed to predict five labels, namely $\rho_1, \rho_2, \rho_3, \rho_4$, and ρ_5, based on three features denoted as A, F, and P, as defined in Equation 40. The values of these features were known from the plant model used in the study. The simulation was conducted using HILO-MPC, and the neural network was trained using PyTorch with the Adam stochastic gradient descent optimizer. The soft-plus activation function was utilized in the neural network.

The training of the neural network was carried out using the classic approach, as described in Section 8 since the physics-informed training was not able to converge in this case, possibly due to the sensitivity of the states to the variations of the cybernetic control variables. An 80/20 train/test split was used for the dataset, where 80% of the data was used for training and 20% for testing.

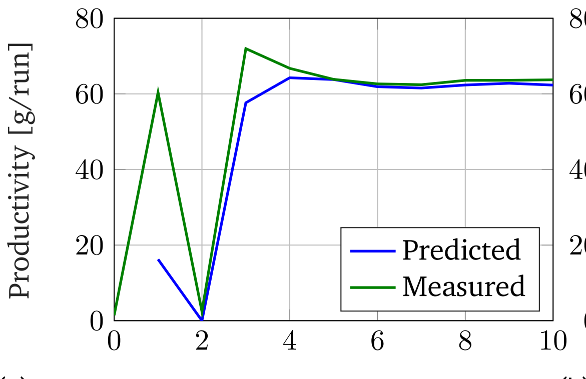
Results Figure 22 shows the predicted and real net product for every batch for different levels of I_{max}. The values for $I_{\text{max}} > 50\,\%$ are not shown, as no significant difference from $I_{\text{max}} = 50\,\%$ was present because of the low level of risk. In Fig. 23, the difference between open-loop control and MPC is shown. Also, in this case, the MPC does not always show better performance of the open-loop strategy because of the different sets of data collected in the runs. Figure 24 summarizes the KPIs for this case. $I_{\text{max}} = 25\,\%$ shows overall better KPIs than the other levels both for MPC and open-loop control

The effect of the learning can be seen in Fig. 25 where the predicted and real P was plotted for $I_{\text{max}} = 50\,\%$ for run 1 (left) and run 5 (right). As can be seen, the prediction of the P is better as the number of runs increases due to a better machine learning model. In this case, since the inputs and outputs have more than two dimensions, visualizing the risk map is challenging; for this reason, they are not shown.

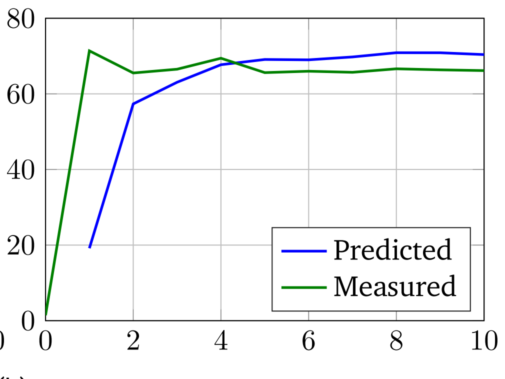
5.5. Summary

In this chapter, we proposed a model-based machine-learning-supported open-loop optimization approach for repetitive processes, as well as an economic shrinking-horizon MPC strategy, considering model and measurement uncertainties and the absence of a priori knowledge of the optimal trajectory. Our method utilized the data collected during each run to improve the plant prediction by training a neural network at the end of each run. Additionally, we incorporated the uncertainties of the model into the optimization problem through a risk function based on an ensemble of Gaussian process regressors, which allowed for the safe utilization of a small dataset from the first run and a progressive increase of confidence regions in subsequent runs, thereby limiting sub-optimal or unsafe control actions. The controller's level of exploration or aggressiveness could be easily tuned by manipulating the maximum risk allowed.

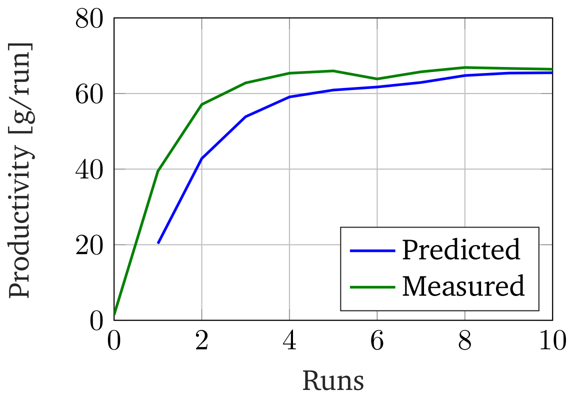
We demonstrated the effectiveness of our approach through two case studies involving a fed-batch bioreactor for the production of β-glactosidase using genetically modified *E. coli D1210*, and a fed-batch reactor for the production of poly-β-hydroxybutyrate (PHB) using *C. necator*. We showed the impact of the maximum allowed risk on the controller's performance and highlighted that intermediate-risk values often provided the best balance between exploration and exploitation. Optimal performance was achieved after several runs for the MPC and open-loop controller. We also demonstrated that using MPC did not



(a) $I_{\text{max}} = 10\%$

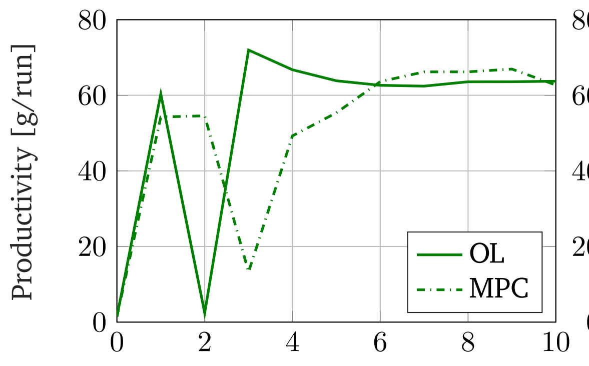


(b) $I_{\text{max}} = 25\%$

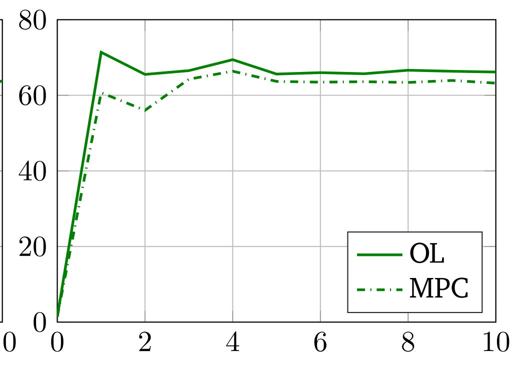


(c) $I_{\text{max}} = 50\%$

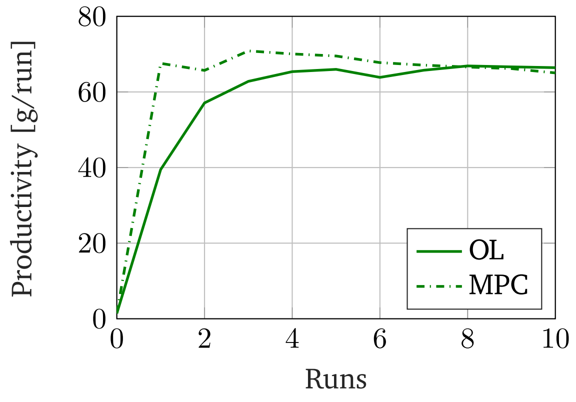
Figure 5.12.: Productivity of PHB in grams per batch for different levels of I_{max}. After $I_{\text{max}} = 50\%$, there is no noticeable change in the predicted and measured productivities; hence the results are not reported. Probably this is due to the fact that the risk is already low in the region of the optimal state trajectory corresponding to $I_{\text{max}} = 50\%$.



(a) $I_{\text{max}} = 10\%$



(b) $I_{\text{max}} = 25\%$



(c) $I_{\text{max}} = 50\%$

Figure 5.13.: comparison of the measured productivity of PHB in grams per batch for different levels of I_{max}, for the open loop (OL) and MPC case. After $I_{\text{max}} = 50\%$, there is no noticeable change in the predicted and measured productivities; hence the results are not reported. Probably this is due to the fact that the risk is already low in the region of the optimal state trajectory corresponding to $I_{\text{max}} = 50\%$.

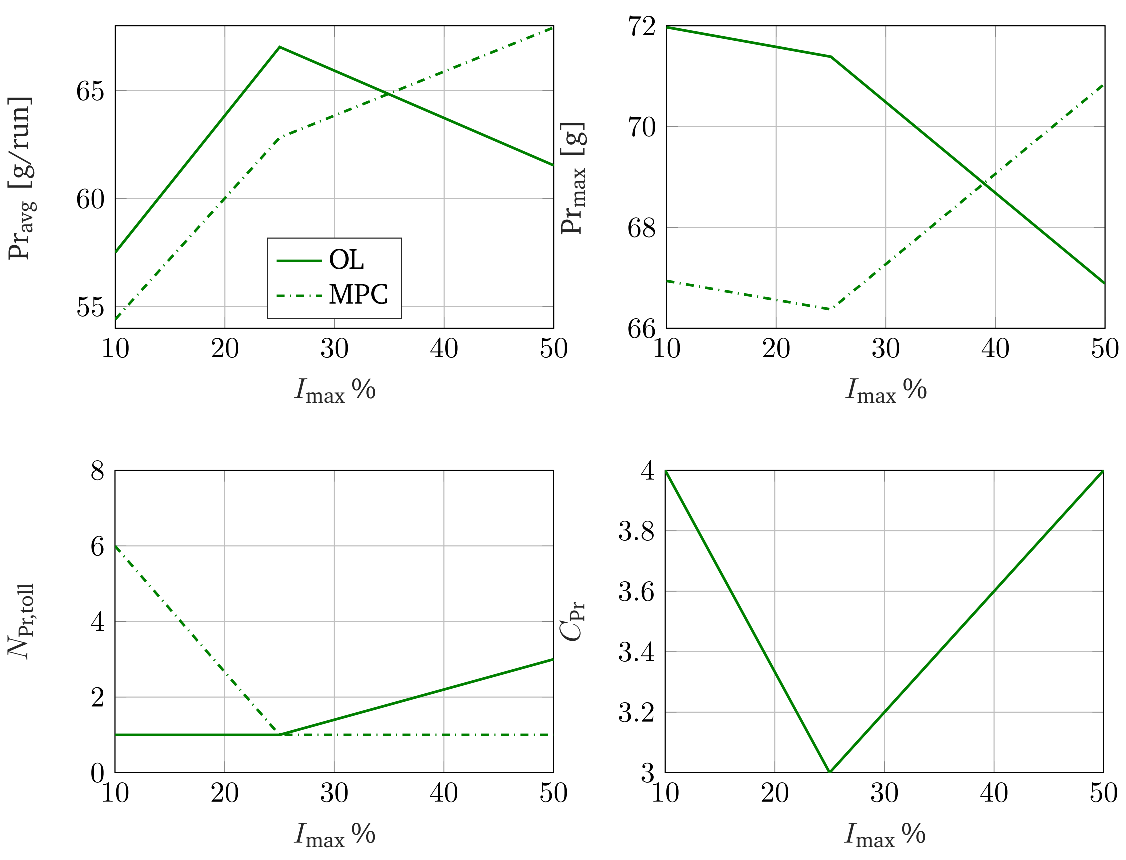


Figure 5.14.: KPIs of the open-loop controller (OL) and MPC. Note that the runs to converge were calculated only for the OL control since the convergence is computed with respect to the predicted profitability at $t = 0$.

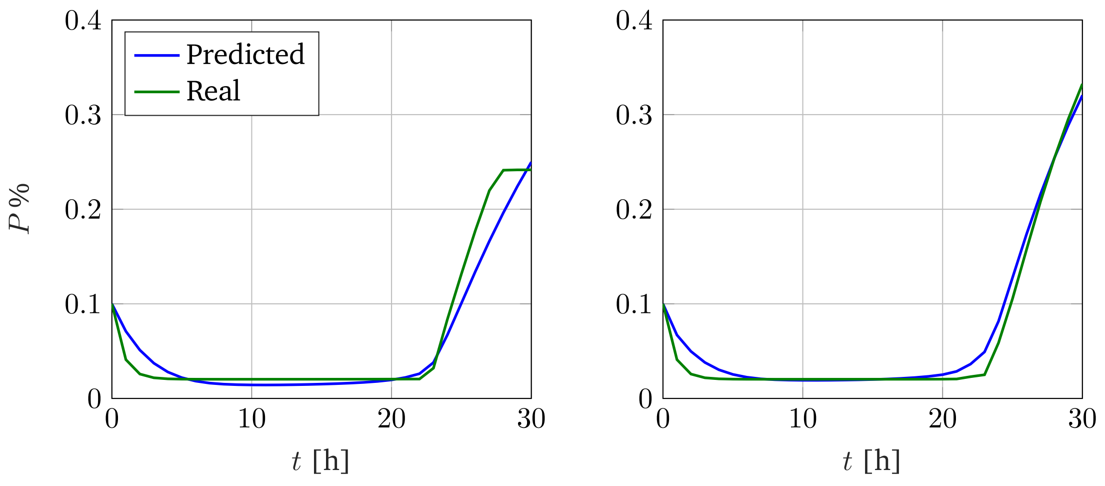


Figure 5.15.: Predicted and real values (noise-free) of P for run 1 (left) and run 5 (right).

always result in better performance for individual runs compared to open-loop control, as the identified model depended on the data collected during the runs, which in turn depended on the controller's actions.

It is important to note that our approach is not yet a robust MPC method, as it cannot guarantee constraint satisfaction for all possible uncertainty values. Furthermore, the controller does not actively decrease the uncertainty, as the learning takes place passively. In the next chapter, we will discuss how to incorporate tube-based shrinking horizon MPC to ensure constraint satisfaction for run-to-run learning, and how to leverage information on model error for design-of-experiment to enable active learning by the controller.

6. Safe Exploration Using Robust Learning Supported MPC

In this chapter, a robust learning-supported MPC using hybrid models with guaranteed constraint satisfaction is applied for run-to-run learning and control. This approach is based on the definition of a *safe set*, i.e., a set where the upper bound on the model error is sufficiently small. Given this upper bound, nonlinear shrinking-horizon tube-based MPC can be applied, and robust constraint satisfaction despite model uncertainty is guaranteed. We show how, under some assumptions, the safe region expands, decreasing conservativeness and increasing performance as the number of runs increases. This chapter is partially based on the results presented in [129].

6.1. Introduction

Using hybrid models can improve the prediction performance, but the quality of the prediction depends on the quality and quantity of the data used for training. Especially in process engineering, data quality and quantity can be an issue; hence, predictions and, consequently, closed-loop control performance can be negatively affected. Considering the model uncertainty in the prediction is fundamental to guaranteeing constraint satisfaction. Section 17 covered some contributions in safe and risk-aware control using machine learning models; Chapter 18 proposed a shrinking-horizon MPC approach that considers the risk of model uncertainty in the constraints, hence was able to avoid areas with high model uncertainty. Nevertheless, constraint satisfaction was not guaranteed. In this chapter, we propose an approach based on robust tube shrinking-horizon MPC that guarantees constraint satisfaction, hence allowing the safe operation of the system despite model uncertainty.

As for the risk-aware MPC shown in Chapter 18, this method is based on the following observation: Machine learning models tend to be good at predicting the underlying unknown function close to the training points and tend to be progressively bad far away from them. Hence, it is reasonable to think that the modeling error will be smaller in some regions of the feature space where training points were taken. In these regions, it is more likely to find an upper bound on the modeling error that is not excessively large, i.e., that does not render tube MPC infeasible. To illustrate this concept, let us take the following system as an example

$$\dot{x}(t) = \begin{bmatrix} -1 & 2 \\ -3 & 4 \end{bmatrix} x(t) + \begin{bmatrix} 0.5 \\ -2 \end{bmatrix} u(t) + \rho(x(t)). \tag{6.1}$$

Here $\rho(x(t)) = [0, -0.25x_2^3]^\top$ is assumed to be unknown and must be learned from data. The objective is to bring the second state, $x_2(t)$, to the reference $x_{2,\mathrm{r}} = 1.1$, while satisfying the box constraints on the state $x^{\mathrm{lb}} \leq x(t) \leq x^{\mathrm{ub}}$ and on the input $u^{\mathrm{lb}} \leq u(t) \leq u^{\mathrm{ub}}$ over a finite horizon $\forall t \in [0, T_f]$ where $x^{\mathrm{ub}} = [5, 1.2]^\top$, $x^{\mathrm{lb}} = [-5, -1.2]^\top$. We indicate the state constraints with the set $\mathbb{X} \triangleq \{x \in \mathbb{R}^{n_x} |\ x^{\mathrm{lb}} \leq x(t) \leq x^{\mathrm{ub}}\}$. Figure 26 (top-left) shows a neural network trained on noisy measurements of the unknown function $\rho(x(t))$. The neural network fits the function well close to the training datapoints, but the prediction accuracy deteriorates far away from them. Note that the reference point lies in a zone where the machine learning model has a significant error, which might cause constraint violation. Given an upper bound on the error, i.e., the error between the real function $\rho(x)$ and the learned function $\rho_\theta(x)$ over the set $\mathbb{X}$, nonlinear tube-based MPC could be used as in [174]. Nevertheless, the upper bound is too large, rendering the tube MPC approach infeasible. We instead propose the following approach: Suppose to take the set $\mathcal{X} = [-1.25,\ 0.5] \subset \mathbb{X}$ shown in Fig. 26 (top-right). In this subset, the error is small. With this error, let us compute the corresponding robust control invariant set Ω and shrink $\mathcal{X}$, i.e. $\bar{\mathcal{X}} = \mathcal{X} \ominus \Omega$, forming a smaller area. For this case, the shrunk set is not empty, the tube MPC problem has a solution, and a control sequence can safely be applied to the plant. Assume that the system comes closer to the edge of the $\mathcal{X}$ where new measurements can be taken safely. After some measurements are taken, the run is stopped, and the measurements are used to improve the learned model. In the next run, the error decreases and a new expanded safe set $\mathcal{X}$ can be computed since the confidence of the machine learning model expands as well (cf. Fig. 26 bottom-left). Eventually, after some runs, the system can reach the required reference (cf. Fig. 26 bottom-right). Note that, since the system naturally moves towards the direction where the cost function decreases (hence towards better performance), the data are collected towards the direction of the improvement[1]. In other words, in this way, the machine learning model is tailored towards the control-relevant regions, hence making a parsimonious use of data. Furthermore, note

[1]As we will see in the following sections, this might not always be the best approach, since it might happen that the system "sticks" in a set that cannot expand anymore. We do not discuss this further here for simplicity.

that in general, the reference will not be reached at the first run hence this approach is useful for new plants with a fairly uncertain model, which can be repeated[2], where the objective is to safely learn a model that, from the initial condition, would optimally guide the system to the required objective. Once the model has been safely identified, another approach can be used, for example, "vanilla" nonlinear tube MPC. Also, even though we focus on constant reference tracking problems, the same algorithm could be applied for time-varying references that repeat at every batch and for economic objectives.

This chapter is partially based on our paper [129], with expansions concerning a new, more complex system, algorithms to find the safe sets, and proof that guarantees, under some assumptions, the expansion of the safe set at every run. The following sections give more details on the method, i.e., how the upper bound on the model error is found and how the safe set is expanded.

6.2. Problem setup

We consider a partially unknown nonlinear system of the form

$$\dot{x}(t) = Ax(t) + Bu(t) + \tilde{\rho}(x(t)), \tag{6.2}$$

subject to constraints

$$x(t) \in \mathbb{X}, \;\; u(t) \in \mathbb{U}, \;\; \forall t \geq 0,$$

where $x(t) \in \mathbb{R}^{n_x}$ and $u(t) \in \mathbb{R}^{n_u}$ are the vector of states and inputs respectively, $A \in \mathbb{R}^{n_x \times n_x}$ and $B \in \mathbb{R}^{n_x \times n_u}$ are known matrices and $\tilde{\rho} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ is an unknown, possibly nonlinear, function.

Assumption 6.1 *The set $\mathbb{U} \subset \mathbb{R}^{n_u}$ is compact, and $\mathbb{X} \subset \mathbb{R}^{n_x}$ is bounded.*

The process is operated repetitively. The time of every run spans $t \in [0, T_f]$ where T_f is the final run time. For simplicity, we assume that T_f is known and equal for all runs, but the same method could also be applied if the run time differs at every run. The input is calculated using a shrinking-horizon MPC formulation that uses measurements obtained at each sampling time until the final time T_f, as shown in the following sections. We indicate with the superscript i the variables used in run i. We want to increase the process performance at every run by improving our knowledge on $\tilde{\rho}(\cdot)$ in regions relevant to closed-loop control while safely satisfying the constraints. Performance can be defined,

[2]Although here we are focusing on repeating processes, a similar approach could be used for continuous processes. We will give some ideas on how this could be achieved in the outlook section of this chapter.

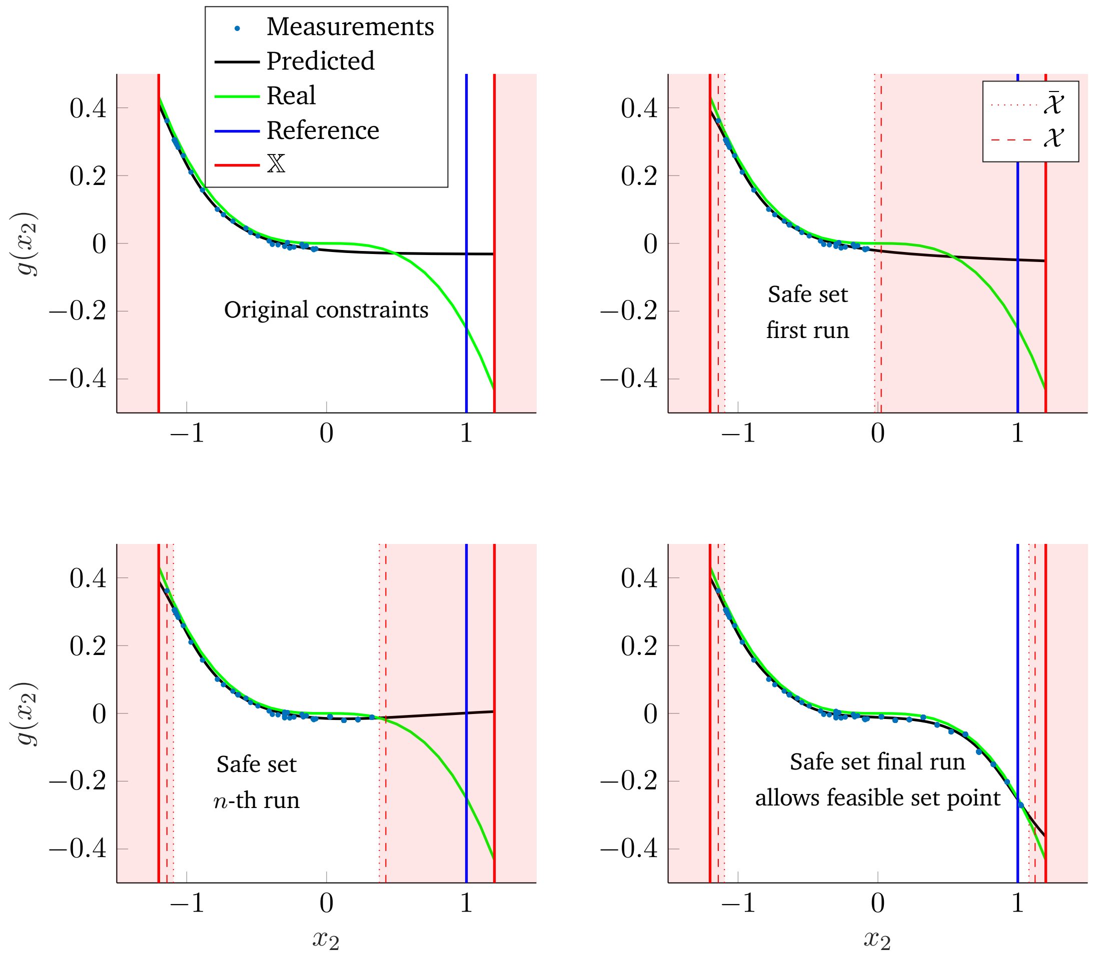


Figure 6.1.: Decreasing the conservatism during repeated batches due to improved process knowledge. Top-left figure: Real and predicted function with constraints on x_2, the learning error in the admissible set $\mathbb{X}$ away from the data points is large. Top-right figure: A small set $\mathcal{X}$ is found where the error on the learned function is small. Bottom-left figure: The set expands as the number of measurements increases. Bottom-right: the admissible set $\mathcal{X}$ has reached the reference, and the system can now safely reach it.

for example, in terms of tracking error or the economic objective's value. To this aim, after every run, we approximate the unknown function with a machine learning model $\rho^i_\theta(\cdot)$ where $\theta \in \mathbb{R}^{n_\theta}$ is the vector of hyperparameters or weights of the model. Note that the method does not depend on the machine learning model. The relation between the unknown function and its learned counterpart is

$$\tilde{\rho}(x(t)) = \rho^i_\theta(x(t)) + w^i(x(t)),$$

where $w^i : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ is an unknown state-varying model error. Let $\|(\cdot)\|$ be the 2-norm.

Assumption 6.2 *The function $\tilde{\rho}(x)$ of the real system is Lipschitz continuous over $\mathbb{X}$ with known Lipschitz constant $\tilde{L}_\mathbb{X}$ i.e.*

$$\|\tilde{\rho}(x) - \tilde{\rho}(\bar{x})\|_\infty \leq \tilde{L}_\mathbb{X}\|x - \bar{x}\|, \;\; \forall x \in \mathbb{X}. \tag{6.3}$$

Remark Assumption 6.6 requires us to know an estimate of the Lipschitz constant. This does not necessarily have to be the smallest Lipschitz constant. Nevertheless, it is important to remember that a constant that is too large can cause very conservative control actions or even render the problem unfeasible. In practice, an approximated value of the Lipschitz constant can be obtained from data, for example, as proposed in [36]. In this chapter, we will assume that a Lipschitz constant is given.

Assumption 6.3 *The nonlinear function $\rho_\theta(x(t))$ is Lipschitz continuous in $\mathcal{X}$ with constant $L_\mathcal{X}$, i.e.,*

$$\|\rho_\theta(x) - \rho_\theta(\bar{x})\|_\infty \leq L_\mathcal{X}\|x - \bar{x}\|, \;\; \forall x \in \mathbb{X}. \tag{6.4}$$

Remark Note that Assumption 6.7 is not excessively restrictive since a Lipschitz constant can be found at least locally for all continuously differentiable functions. This includes machine learning models such as Gaussian processes with appropriate kernels. Several algorithms for approximating the Lipschitz constant have been proposed for neural networks for differentiable as well as non-differentiable activation functions (cf. [56] and references therein).

The measurements of the real functions $\hat{y}$ are effected by a bounded noise $v \in \mathbb{V} \subset \mathbb{R}^{n_x}$

$$\hat{y} = \tilde{\rho}(x(t)) + v(t),$$

Before the run i, the machine learning model is trained with the dataset

$$\mathcal{D}^i \triangleq \{(\hat{x}_1, \hat{y}_1), (\hat{x}_2, \hat{y}_2), \ldots, (\hat{x}_{n_{\mathcal{D}^i}}, \hat{y}_{n_{\mathcal{D}^i}})\},$$

which contains all measurements of states $\hat{x}_k$ and labels $\hat{y}_k$ of all runs up to run $i-1$, where $n_{\mathcal{D}^i}$ is the total number of measurements.

For the $i-$th run, the system can be written as

$$\dot{x}(t) = Ax(t) + Bu(t) + \rho^i_\theta(x(t)) + w^i(x(t)). \tag{6.5}$$

We could now find an upper-bound on the modeling error $w^i(x(t))$ and use a nonlinear tube MPC approach, like in [174], to guarantee constraint satisfaction, but, as mentioned in the introduction, the upper bound on the error for all admissible states can be too large, especially when there is not sufficient data to learn $\tilde{\rho}(\cdot)$. Hence, we propose to find this upper bound in a smaller subset of the admissible state set. Let $\mathcal{X}^i \subseteq \mathbb{X}$ be this subset, compact and containing the plant's initial conditions. Associated with this set, there exists a set $\mathcal{W}_{\mathcal{X}^i}$ that contains all values that the model error can take over $\mathcal{X}^i$.

Assumption 6.4 *$w^i(x(t))$ lies in a compact set $\mathcal{W}_{\mathcal{X}^i}$ where $\mathcal{W}_{\mathcal{X}^i} \in \mathbb{R}^{n_x}$ for all $x \in \mathcal{X}^i$.*

Assumption 6.8 is not too restrictive and valid, for example, when $w^i(\cdot)$ is continuous since $\mathcal{X}$ is bounded. Hence, if Assumption 6.8 is true, there exist $w_{\text{max},\mathcal{X}^i} \in \mathbb{R}$ such that:

$$w_{\text{max},\mathcal{X}^i} \geq \|w(x(t))\|_\infty, \;\; \forall x \in \mathcal{X}^i.$$

In other words, a bound on the model error can be found. The plant (43) can be represented equivalently as:

$$\dot{x}(t) = Ax(t) + Bu(t) + \rho^i_\theta(x(t)) + d(t), \tag{6.6}$$

for $x(t) \in \mathcal{X}^i$ where $d(t) \in \mathcal{W}_{\mathcal{X}^i}$ is a disturbance reflecting the modeling error. In Section 26.1 we will show how to compute $w_{\text{max},\mathcal{X}^i}$. For now, we assume that it has been computed and proceed with computing the robust control invariant set.

6.2.1. Robust control invariant set

Given a bound on the model error $w_{\text{max},\mathcal{X}^i}$ over the set $\mathcal{X}^i$, we proceed with the computation of the robust control invariant set that encloses the evolution of the error between the true system and the model used in the controller as we outlined in chapter 9. The *nominal* system is given by setting $d(t) = 0$ as

$$\dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t) + \rho^i_\theta(\bar{x}(t)). \tag{6.7}$$

The error dynamics are defined as the difference between the nominal system and the real system (47) as

$$\dot{z}(t) = Az(t) + B(u(t) - \bar{u}(t)) + \rho_\theta^i(x(t)) - \rho_\theta^i(\bar{x}(t)) + d(t), \tag{6.8}$$

where $z(t) = x(t) - \bar{x}(t)$. Following the principle of tube-based MPC [124] the control that is applied to the plant contains two parts, the control signal that steers the nominal system to the desired trajectory or reference point $\bar{u}(t)$ and an ancillary feedback control $k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$ that keeps the real system trajectories in the robust control invariant set centered around the trajectory of the nominal system

$$u(t) = \bar{u}(t) + k(x(t), \bar{x}(t)). \tag{6.9}$$

Choosing a linear feedback with gain K, the input can be written as $u(t) \triangleq \bar{u}(t) + K(x(t) - \bar{x}(t)),\ \ K \in \mathbb{R}^{n_u \times n_x}$ and the error system (49) becomes

$$\dot{z}(t) = (A + BK)z(t) + \rho_\theta^i(x(t)) - \rho_\theta^i(\bar{x}(t)) + d(t). \tag{6.10}$$

Definition 6.1 (Robust control invariant set) *A set $\Omega \subset \mathbb{X} \subset \mathbb{R}^{n_x}$ is a* robust control invariant set *for the system* (51) *if there exist a feedback control law $k(\cdot,\cdot)$ with $k(\cdot,\cdot) + \bar{u}(\cdot) \in \mathcal{U}$ such that $\forall z(t_0) \in \Omega$ then $z(t) \in \Omega\ \forall t > t_0,\ \forall d(t) \in \mathcal{W}_{\mathcal{X}^i}$.*

If a robust control invariant set exists for the error dynamics (51), it is possible to keep the real systems in a bounded region around the nominal system trajectory. The next theorem defines conditions of existence of such a robust control invariant set (cf. [207]).

Theorem 6.1 *Let $S : \mathbb{R}^{n_x} \to [0, \infty)$ be a continuously differentiable function bounded between two $\mathcal{K}_\infty$ functions α_1 and α_2 via $\alpha_1(\|z\|) \leq S(z) \leq \alpha_2(\|z\|)$. If there exist a $\lambda > 0$ and $\mu > 0$ such that*

$$\dot{S}(z) + \lambda S(z) - \mu d^\mathsf{T} d \leq 0,\ \ \forall d \in \mathcal{W}_{\mathcal{X}^i}, \tag{6.11}$$

with $z \in \mathcal{X}^i$, then there exist a robust control invariant set $\Omega(w_{\max,\mathcal{X}^i}, \lambda, \mu)$ such that

$$\Omega(w_{\max,\mathcal{X}^i}, \lambda, \mu) \triangleq \left\{ z \in \mathbb{R}^{n_x} | S(z) \leq \frac{\mu w_{\max,\mathcal{X}^i}^2}{\lambda} \right\}. \tag{6.12}$$

The proof can be found in [206].

When clear from the context, we will write $\Omega^i = \Omega(w_{\max,\mathcal{X}^i}, \lambda, \mu)$ for simplicity of notation. The parameters λ, μ and the gain K of the ancillary controller can be found with the following theorem:

Theorem 6.2 *Suppose that there exist a positive definite matrix $X \in \mathbb{R}^{n_x \times n_x}$ and a matrix $Y \in \mathbb{R}^{n_u \times n_x}$ and two scalars $\lambda_0 > \lambda > 0$ and $\mu > 0$ such that $T(X, Y, \lambda_0, \mu) \leq 0$ where*

$$T(X, Y, \lambda_0, \mu) \triangleq \begin{bmatrix} (AX + BY)^\mathsf{T} + AX + BY + \lambda_0 X & I \\ I & -\mu I \end{bmatrix} \tag{6.13}$$

and let the Lipschitz constant $L_{\mathcal{X}^i}$ of $\rho_\theta^i(x(t))$ be such that

$$L_{\mathcal{X}^i} \leq \frac{(\lambda_0 - \lambda)\alpha_{\min}(P)}{2\|P\|_2}, \tag{6.14}$$

where $P = X^{-1}$ is a symmetric matrix. Then there exists a robust control invariant set Ω as in (53) *with ancillary control law $k(x, \bar{x}) = K(x - \bar{x})$ where $K = YX^{-1}$.*

The proof can be found in Appendix B.1. Hence, the problem of finding a robust control invariant set can be solved by finding a solution of the linear matrix inequality $T(X, Y, \lambda_0, \mu) \leq 0$ where the unknowns are X, Y, λ_0 and μ. The solution is not unique, and we can use this to our advantage. Note from (55) that we want λ_0 to be large, allowing for larger Lipschitz constants. Also, from (53), we note that μ should ideally be small. Hence, we propose to impose these conditions in an optimization problem. Since the product $\lambda_0 X$ would cause the optimization problem to be nonlinear with respect to these variables, we split it into two (successively solved) linear optimization problems. First we guess a value λ_0^{guess} for λ_0 and we solve:

$$\mu^*, X^*, Y^* = \arg\min_{\mu, X, Y} \mu, \tag{6.15a}$$

$$\text{s.t. } T(X, Y, \mu, \lambda_0^{\text{guess}}) \leq 0, \tag{6.15b}$$

$$X \geq 0, \tag{6.15c}$$

$$\mu > 0, \tag{6.15d}$$

and then we solve again for λ_0:

$$\lambda_0^* = \arg\max_{\lambda_0} \lambda_0, \tag{6.16a}$$

$$\text{s.t. } T(X^*, Y^*, \mu^*, \lambda_0) \leq 0, \tag{6.16b}$$

$$\lambda_0 > 0. \tag{6.16c}$$

Since we want to find the largest λ possible, with μ^* and λ_0^* and $L_{\mathcal{X}^i}$ we can now find λ from (55) by imposing equality[3]

$$\lambda^* := \lambda_0^* - 2\frac{\|P\| L_{\mathcal{X}^i}}{\alpha_{\min}(P)}. \tag{6.17}$$

[3]See Appendix C.1.

Hence, if $\lambda^* > 0$, the robust control invariant set (53) can be evaluated as:

$$\Omega^i \triangleq \left\{ z \in \mathbb{R}^{n_x} | S(z) \leq \frac{\mu^* w^2_{\text{max},\mathcal{X}^i}}{\lambda^*} \right\}. \tag{6.18}$$

This linear iterative approach, although conservative, allows us to solve the matrix inequality problem easily.

Reducing conservatism with the one-sided Lipschitz constant Using the Lipschitz constant to define the robust control invariant set often results in a large set, resulting in a conservatively large tube. This is due to the fact that $\alpha_{\text{min}}(P) \leq \|P\|$ hence the admissible $L_{\mathcal{X}^i}$ is small. For this reason, a similar condition based on the one-sided Lipschitz constant was proposed [207].

Definition 6.2 (One-sided Lipschitz continuous function) *A function $f(x) : \mathbb{R}^{n_x} \mapsto \mathbb{R}$ is one-sided Lipschitz continuous on a set $\mathcal{X}$ if there exist a $\mathcal{L} \in \mathbb{R}$ such that, for all $x_1, x_2 \in \mathcal{X}$,*

$$(f(x_1) - f(x_2))^\mathsf{T}(x_1 - x_2) \leq \mathcal{L}\|x_1 - x_2\|_2 \tag{6.19}$$

where $\mathcal{L}$ is the one-sided Lipschitz constant.

Note that the one-sided Lipschitz constant is smaller or equal to the Lipschitz constant and can also be negative or zero. This allows us to define a similar but less conservative condition for the existence of a robust control invariant set.

Corollary 6.3 *Suppose that there exist a positive definite matrix $X \in \mathbb{R}^{n_x \times n_x}$ and a matrix $Y \in \mathbb{R}^{n_u \times n_x}$ and two scalars $\lambda_0 > \lambda > 0$ and $\mu > 0$ such that $T(X, Y, \lambda_0, \mu) \leq 0$ where*

$$T(X, Y, \lambda_0, \mu) \triangleq \begin{bmatrix} (AX + BY)^\mathsf{T} + AX + BY + \lambda_0 X & I \\ I & -\mu I \end{bmatrix} \tag{6.20}$$

and let $P\rho^i_\theta(x(t))$ be one-sided Lipschitz continuous in $\mathcal{X}^i$ with constant $\mathcal{L}_{\mathcal{X}^i}$. If

$$\mathcal{L}_{\mathcal{X}^i} \leq \frac{(\lambda_0 - \lambda)\alpha_{\text{min}}(P)}{2}, \tag{6.21}$$

where $P = X^{-1}$ is a symmetric matrix, then there exists a robust control invariant set Ω as in (53) *with ancillary control law $k(x, \bar{x}) = K(x - \bar{x})$ where $K = YX^{-1}$.*

The proof can be found in [207]. In our case, the one-sided Lipschitz constant is computed as suggested in [136].

Remark The condition $\lambda^* > 0$ is checked at every run. If it is respected, the previous values of λ^* and μ^* are used. If it is not respected, one can, for example, choose another $\mathcal{X}^i$ or to guess a new λ^0_{guess} and solve (58) and (59) again. For our cases, we observed that when using the one-sided Lipschitz constant, the condition was always respected, so we did not have to change λ^* and μ^*.

6.2.2. Tube-based shrinking horizon MPC

Once we obtain Ω^i, the nominal shrinking horizon model predictive control at sampling time t_k for the i^{th} run is given by

$$\min_{\bar{u}(\cdot)} \quad L(\bar{u}(\cdot), x(t_k)) \tag{6.22a}$$

$$\text{s.t.} \quad \dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t) + \rho^i_\theta(\bar{x}(t)), \tag{6.22b}$$

$$\bar{x} \in \bar{\mathcal{X}}^i, \;\; \bar{u} \in \bar{\mathcal{U}}^i, \tag{6.22c}$$

$$\bar{x}(t_k) = x_k, \tag{6.22d}$$

$$\text{for } \; t \in [t_k, T], \tag{6.22e}$$

where $\bar{\mathcal{X}} = \mathcal{X}^i \ominus \Omega^i, \bar{\mathcal{U}} = \mathbb{U} \ominus G^i$ and

$$G^i = \{k(x, \bar{x}) \in \mathbb{R}^{n_u} | x - \bar{x} \in \Omega^i, \;\; x \in \mathcal{X}^i \;\text{ and }\; \bar{x} \in \bar{\mathcal{X}}^i\}.$$

The state constraint in (63c) forces the system to lie in a region with a known upper bound on the modeling error. The objective function $L(\bar{u}(\cdot), x(t_k))$ is defined as

$$L(\bar{u}(\cdot), x(t_k)) = \int_{t_k}^{T} l(x(\tau), u(\tau))d\tau + e(x(t_k + T)), \tag{6.23}$$

where $T - t_k$ is the prediction horizon, $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ and $e : \mathbb{R}^{n_x} \to \mathbb{R}$ are continuous and differentiable at least twice. Furthermore $l(x(t), u(t)) \geq 0$ and $e(x(t), u(t)) \geq 0$. Problem (63) is solved at discrete sampling times $t_k = k\Delta t, \;\; k \in [0, N-1]$ where $N = \text{ceil}(T/\Delta t)$ with sampling interval Δt which for simplicity and without loss generality, is assumed to be fixed. The optimal solution is denoted with $\bar{u}^*(\cdot)$. Consequently, the resulting control signal is given by

$$u(t) = \bar{u}(t) + k(x(t), \bar{x}^*(t)), \;\; t \in [t_k, t_k + \Delta t], \tag{6.24}$$

where $\bar{x}^*(t)$ is the optimal nominal trajectory.

Remark We want to stress the fact that the intra-run computational burden is equivalent to a nominal MPC since Problem 63 does not depend on the disturbances and the robust control invariant set is computed offline *before* every run. If the initial state is known, as proposed in [207], the computational time can be further decreased by pre-computing the nominal control action offline as in Algorithm 1.

Algorithm 1 Offline implementation of the shrinking-horizon tube MPC controller

```
Offline
    1. Set $\bar{x}(t_0) = x(t_0)$ and set $k \leftarrow 1$
    2. Solve Problem (63) at $t_k$, and store $\bar{u}(t_k)$
    3. Compute the next state $\bar{x}(t_{k+1})$ applying $\bar{u}(t_k)$ to the nominal system and store it
    4. Set $\bar{x}(t_k) \leftarrow \bar{x}(t_{k+1})$ and $k \leftarrow k+1$ and start from Step 2.
Online
    1. Apply the control action $u(t_k) = \bar{u}(t_k) + k(x(t_k), \bar{x}(t_k))$ to the real system during
       the interval $[t_k, t_{k+1})$ using the stored states and inputs of the nominal system
    2. Measure the state $x(t_{k+1})$ and set and $k \leftarrow k+1$
```

Figure 27 summarizes the components of the method.

6.2.3. Intra-run recursive feasibility

Since the shirking horizon MPC predicts until the end of the processes, guaranteeing recursive feasibility is easier than a receding horizon MPC. All we need is to assume initial feasibility.

Assumption 6.5 (Initial feasibility) *There exist a safe set $\mathcal{X}^{i,*}$ with the corresponding upper bound $w_{max,\mathcal{X}^{i,*}}$ such that Problem* (63) *is feasible a $t = t_0$.*

Corollary 6.4 *If Assumption 6.9 is true, Problem* (63) *is recursively feasible for all $t_k = k\Delta T$ with $k \in [0, N-1]$.*

Proof For the i^{th} run, let $\bar{u}^*(\tau)$ with $\tau \in [0, T]$ be the solution of (30) at time $t_0 = 0$, then, for the principle of optimality, at the next sampling time $t_1 = t_0 + \Delta t$ the truncated control

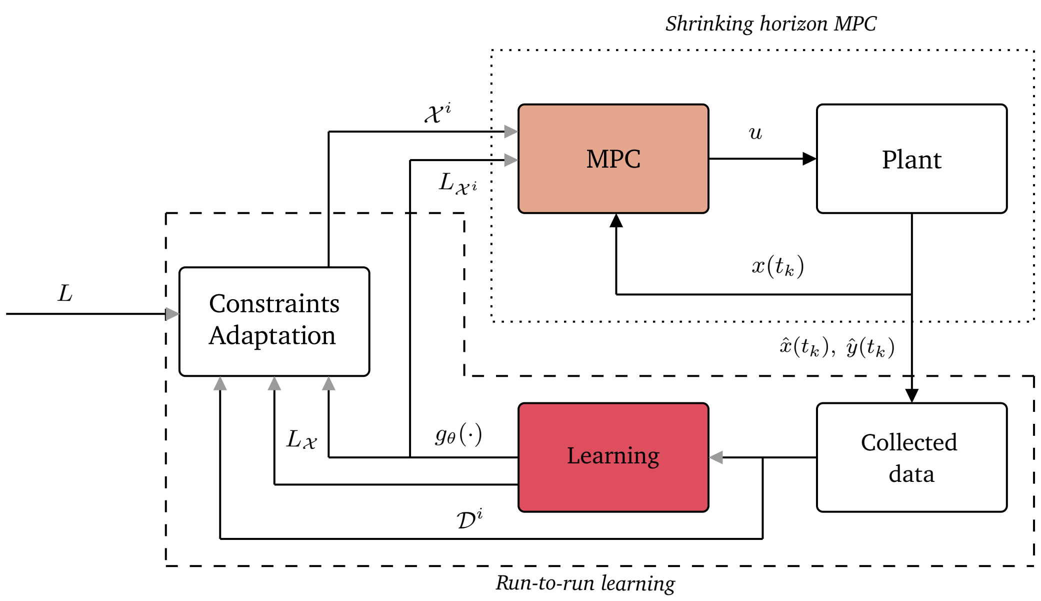


Figure 6.2.: Scheme of the proposed approach.

sequence $\bar{u}^*(\tau)$ with $\tau \in [\Delta t, T]$ is also a feasible solution. By induction, this is true for all sampling points; hence, a feasible solution always exists.

6.3. Safe sets

It remains to compute suitable safe sets. This task can be split into calculating an upper bound on the modeling error and the safe sets themself.

6.3.1. Computing an upper bound on the modeling error

So far, we have assumed to have an upper bound on the modeling error. In this section, a method to compute an upper bound is shown. This method is based on the assumption of Lipschitz continuity of both learned $\rho_\theta(\cdot)$ and unknown function $\tilde{\rho}(\cdot)$.

The idea is to use the Lipschitz constant of the real function to bound it with piece-wise linear functions as in [18] (cf. Fig. 28). Let us define, for the n^{th} element of $\tilde{\rho}(\cdot)$, the upper and lower bounding functions H^{u} and H^{l}, respectively, using its Lipschitz constant:

$$H^{\mathrm{u}}_n(x;\mathcal{D}^i) = \min_{k\in I_{\mathcal{D}^i}} (\hat{y}_{k,n} + \tilde{L}_{\mathbb{X}}\|\hat{x}_k - x\| + v_{n,\max}), \tag{6.25}$$

$$H^{\mathrm{l}}_n(x;\mathcal{D}^i) = \max_{k\in I_{\mathcal{D}^i}} (\hat{y}_{k,n} - \tilde{L}_{\mathbb{X}}\|\hat{x}_k - x\| - v_{n,\max}), \tag{6.26}$$

where $v_{n,\max}$ is the maximum absolute value of the n^{th} element of the noise vector and $(\hat{x}_k, \hat{y}_k) \in \mathcal{D}^i$ and $I_{\mathcal{D}^i} = \{1, ..., n_{\mathcal{D}^i}\}$ is the set of indices of measurements in $\mathcal{D}^i$. The function that describes the maximum value on the model error in absolute value is

$$\tilde{w}_{n,\max}(x;\mathcal{D}^i) = \max\left(e_{n,u}(x;\mathcal{D}^i),\ \ e_{n,l}(x;\mathcal{D}^i)\right) \tag{6.27}$$

where $e_{n,u}(x;\mathcal{D}^i) = |\rho_{\theta,n}(x) - H^{\mathrm{u}}_n(x;\mathcal{D}^i)|$ and $e_{n,l}(x;\mathcal{D}^i) = |\rho_{\theta,n}(x) - H^{\mathrm{l}}_n(x;\mathcal{D}^i)|$. The maximum possible error between the real system and the learned function is then

$$\tilde{w}_{n,\max,\mathcal{X}^i,\mathcal{D}^i} = \max_{x\in\mathcal{X}^i} \left(\tilde{w}_{n,\max}(x;\mathcal{D}^i)\right), \tag{6.28}$$

Finally, we define the maximum model error as the largest element of the maximum error vector.

$$w_{\max,\mathcal{X}^i,\mathcal{D}^i} \triangleq \|\tilde{w}_{\max,\mathcal{X}^i,\mathcal{D}^i}\|_\infty \geq \|w(x(t))\|_\infty,\ \ \forall x \in \mathcal{X}, \tag{6.29}$$

where $\tilde{w}_{\max,\mathcal{X}^i,\mathcal{D}^i} = [\tilde{w}_{1,\max,\mathcal{X}^i,\mathcal{D}^i}, ..., \tilde{w}_{n_x,\max,\mathcal{X}^i,\mathcal{D}^i}]$. For simplicity of notation, we omit the parametrization with the dataset when clear from the context, i.e., $w_{\max,\mathcal{X}^i} \triangleq w_{\max,\mathcal{X}^i,\mathcal{D}^i}$. Furthermore, let $\mathfrak{F}$ be the map $\mathfrak{F} : \mathcal{C}^0 \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_x} \to \mathbb{R}$ that returns $w_{\max,\mathcal{X}^i}$ given $\rho^i_\theta, \mathcal{D}^i, \tilde{L}_{\mathbb{X}}$ and $\mathcal{X}^i$. The following lemma will be useful later

Lemma 6.1 *The maximum error functions $\tilde{w}_{n,max}(x;\mathcal{D}^i)$ is Lipschitz continuous $\forall x \in \mathcal{X}^i$ with Lipschitz constant $\tilde{L}_{\mathbb{X}} + L_{\mathcal{X}}$.*

The proof of Lemma 6.1 can be found in Appendix B.2.

Remark So far we have considered only error in the measurements of y but not in the state error, i.e., the error in the query state x or in the measured state $\hat{x}$ (c.f. Section 4.2.4. in [35]). This can be done by invoking the Lipschitz continuity property. Assume that the error on only the query measurements is affected by an error $v(x)$ of which we have an upper bound $v_{\max}(x)$, then we have that, for the n-th component of the function $\rho_n(\cdot)$, $|\rho_n(x) - \rho_n(x + v_x)| \leq \|v_{\max}(x)\|$. In this case, the query error affects only the uncertainty bounds, and $v_{\max}(x)$ can be simply added to $e_{n,u}(x;\mathcal{D}^i)$ and $e_{n,l}(x;\mathcal{D}^i)$. If, instead, the training points $\hat{x}$ are noisy, the inference of all query points is affected. By the same argument used for the query points uncertainty, we can accommodate the training points uncertainty by adding the upper bound of the uncertainty to $v_{n,\max}$ in 66.

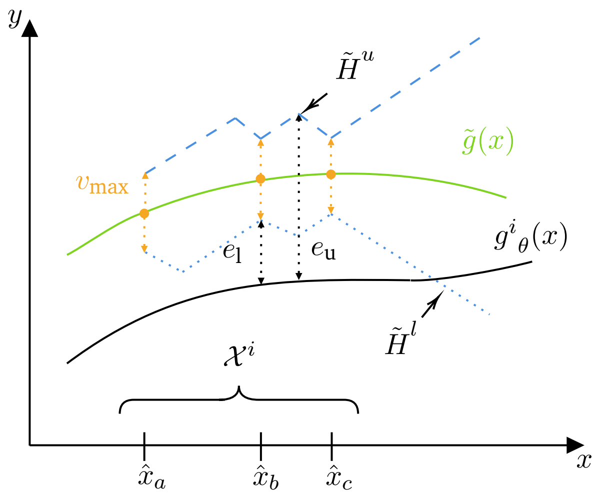


Figure 6.3.: Bound on the error between real function and learned model in case of noisy measurements.

6.3.2. Safety conditions

We now have the tools to compute $w_{\max,\mathcal{X}^i}$ for any given set $\mathcal{X}^i$. The choice of $\mathcal{X}^i$ influences $w_{\max,\mathcal{X}^i}$ and the Lipschitz constant of the learned function $L_{\mathcal{X}^i}$, hence it decides if a robust control invariant set exists and if it does exist, influences its size. For this reason, the choice of $\mathcal{X}^i$ must follow some criteria. First, the conditions necessary for the existence of the robust control invariant set must be satisfied. Second, we would like a "large" set to restrict the state and control actions as little as possible, allowing higher performance. In the following, we define the *necessary safety conditions*.

Definition 6.3 (Necessary safety conditions) *The conditions that guarantee the existence of a robust control invariant set according to Theorem 6.1 and the non-emptiness of the* *shrunk set are:*

$$\bar{\mathcal{X}}^i = \mathcal{X}^i \ominus \Omega(w_{max,\mathcal{X}^i}, \lambda, \mu) \neq \emptyset, \tag{6.30a}$$

$$\bar{\mathcal{U}}^i = \mathcal{X}^i \ominus G^i(\Omega(w_{max,\mathcal{X}^i}, \lambda, \mu)) \neq \emptyset \tag{6.30b}$$

$$T(X, Y, \mu, \lambda_0) \leq 0 \tag{6.30c}$$

$$L_{\mathcal{X}^i} \leq \frac{(\lambda_0 - \lambda)\alpha_{min}(P)}{2\|P\|} \tag{6.30d}$$

$$w_{max,\mathcal{X}^i} = \mathfrak{F}(\rho_\theta^i, \mathcal{D}^i, \tilde{L}_{\mathbb{X}}, \mathcal{X}^i) \tag{6.30e}$$

$$\mathcal{X}^i \subseteq \mathbb{X}, \;\; x_0 \in \mathcal{X}^i \tag{6.30f}$$

$$\lambda_0 > \lambda > 0, \;\; \mu > 0 \tag{6.30g}$$

These conditions are called necessary safety conditions.

We can now give a definition of a safe set.

Definition 6.4 (Safe set) *A set $\mathcal{X}^i \subseteq \mathbb{X}$ is* safe *if, for that set, the safety conditions are satisfied.*

We would like to have some procedures that help us to find a safe set for $i = 1$ (the first run) and t_0 and to update the set for the subsequent runs, i.e., for $i \in [2, ..., n_r]$ where n_r is the number of runs. Ideally, this update should be carried out in such a way that if $\mathcal{X}^1$ is safe, then all $\mathcal{X}^j, \; \forall j > 1$ are safe. We will see how this can be achieved in Section 27. For now, we note that according to the Definition 6.4, one of the conditions for safety is that the shrunk constraints $\bar{\mathcal{X}}^i = \mathcal{X}^i \ominus \Omega^i$ and $\bar{\mathcal{U}}^i = \mathbb{U} \ominus G^i$ are non-empty. It is crucial to ensure that this condition is met because, for example, in the case of large errors, the robust control invariant set (53) might be large, resulting in empty constraints for the nominal problem. This condition is rather easy to check for a useful class of constraints (such as polytopes and box constraints). Furthermore, to avoid excessively conservative constraints and to be able to reach the reference with as few runs as possible, we would like to find a *large* shrunk safe set $\bar{\mathcal{X}}^i$ that is contained in the original constraints $\mathbb{X}$. In practice, we will restrict ourselves to compact sets, Lebesgue measurable under the Euclidean metric. This restriction allows us to formulate the problem of finding the largest shrunk safe set as an optimization problem of the form:

$$\max_{\lambda_0, \lambda, \mu, X, Y, \mathcal{X}^i} \; \Lambda(\bar{\mathcal{X}}^i), \tag{6.31a}$$

$$\text{s.t.} \tag{6.31b}$$

$$\text{Necessary safety conditions, } (71a) - (71g), \tag{6.31c}$$

where $\Lambda(\cdot)$ is the Lebesgue measure. Solving this problem directly is very challenging, and in general, there is no guarantee that a solution exists. In the following, we will present some possible approximations that attempt to find a solution.

First, we notice that (71c) does not depend on $\mathcal{X}^i$ hence we can solve for λ, μ, X and Y in separate problems as in (56) and (57) obtaining $\lambda^*, \mu^*, Y^*, X^*$ and consequently P^*. Furthermore, we use (58) to fix λ, note from (58) that the condition $\lambda < \lambda_0$ is automatically satisfied, but it remains to check that $\lambda > 0$. Hence we obtain

$$\max_{\mathcal{X}^i} \Lambda(\bar{\mathcal{X}}^i), \tag{6.32a}$$

$$\text{s.t.} \tag{6.32b}$$

$$\bar{\mathcal{X}}^i = \mathcal{X}^i \ominus \Omega^i(w_{\text{max},\mathcal{X}^i}, \lambda^*, \mu^*) \neq \emptyset, \tag{6.32c}$$

$$\bar{\mathcal{U}}^i = \mathcal{X}^i \ominus G^i(\Omega^i) \neq \emptyset, \tag{6.32d}$$

$$\lambda^* = \lambda_0^* - 2\frac{\|P^*\|L_{\mathcal{X}^i}}{\alpha_{\text{min}}(P^*)}, \tag{6.32e}$$

$$w_{\text{max},\mathcal{X}^i} = \mathfrak{F}(\rho_\theta^i, \mathcal{D}^i, \tilde{L}_{\mathbb{X}}, \mathcal{X}^i), \tag{6.32f}$$

$$\mathcal{X}^i \subseteq \mathbb{X}, \;\; x_0 \in \mathcal{X}^i, \tag{6.32g}$$

$$\lambda^* > 0. \tag{6.32h}$$

Second, we relax the problem by trying to find *any* set $\bar{\mathcal{X}}^i$ that is non-empty, which is not necessarily the maximum. This is achieved by guessing a set $\mathcal{X}^i$ and then checking if the conditions (73c) to (73h) are met[4].

If the safety conditions are respected, we proceed in attempting to solve Problem (63). If no solution can be found, one can try to guess another set (cf. Fig. 30). Note that this can be done offline before the batch starts; hence, it does not affect the real-time feasibility of the algorithm, assuming that there is enough time between the runs. Algorithm 2 resumes the steps that must be followed to ensure the guessed set is safe.

6.3.3. Guess of the first safe set

This section presents two possible algorithms for the initial guess of the safe set of the first batch $\mathcal{X}^1$.

Remark In general, there is no guarantee that a safe set exists since the model error or the Lipschitz constant of the unknown function may be too large. In the first case,

[4]If it is possible to change the initial condition of the system, the requirement $x_0 \in \mathcal{X}^i$ can be relaxed by placing the initial conditions inside the set.

Algorithm 2 Check if a set is safe

```
Set the iteration index $j = 0$. Provide a guess $\mathcal{X}_0^i \subseteq \mathbb{X}$:
    1. Find a bound on the error $w_{\max,\mathcal{X}_j^i}$ by solving (69).
    2. Compute a (one-sided) Lipschitz constant ($\mathcal{L}_{\mathcal{X}_j^i}$) $L_{\mathcal{X}_j^i}$ of $g_\theta^i(x)$.
    3. Guess a $\lambda_0^{\text{guess}}$ and solve (56) then (57) to obtain $X_j^*, Y_j^*, \mu^*$ and $\lambda_0^*$.
    4. Given $X_j^*, Y_j^*, \mu^*$ and $\lambda_0^*$ solve (55) imposing equality to obtain $\lambda_j$ and check if
       $\lambda_j > 0$ if not, update the index $j \leftarrow j+1$ guess another $\mathcal{X}_j^i$ and start again from 1.
    5. Find $\Omega_j^i$ from (53) and check if $\mathcal{X}_j^i \ominus \Omega_j^i \neq \emptyset$ and $\mathbb{U} \ominus K\Omega_j^i \neq \emptyset$.
    6. If one of the sets is empty, update the index $j \leftarrow j+1$, and guess another $\mathcal{X}_j^i \subseteq \mathbb{X}$
       and start again from 1.
    7. If the sets are not empty solve (63) with
       $\Omega^i \triangleq \Omega_j^i$ and $\mathcal{X}^i \triangleq \mathcal{X}_j^i$.
```

if the primary source of error is not irreducible measurement noise, one can run more experiments and collect more data, attempting to decrease the model error, at least locally. With $\mathcal{X}_j^i$, we indicate the j-th iteration over the set of run i.

Initialization as convex hull The first guess of the safe set can be set as the convex hull of the measured features collected during the *genesis experiments*, i.e., the runs used to collect the first dataset (cf. Section 26.4). This is defined by

$$\mathcal{X}_0^1 \triangleq \operatorname{conv}(\mathcal{D}^1) = \left\{ \sum_{i=1}^{n_{\mathcal{D}^1}} v_i \lambda_i \,\middle|\, v_i \in \mathcal{D}^1,\ \sum_{i=1}^{n_{\mathcal{D}^1}} \lambda_i = 1,\ \lambda_i \geq 0 \right\} \tag{6.33}$$

where $\mathcal{D}^1$ is the dataset collected in the genesis experiments. For example, the convex hull can be found using the Graham Scan algorithm [102]. This guess is justified because machine learning algorithms are usually good at interpolating data. Hence, we expect lower errors close to the training points. Note that this is an initial guess, and it might need to be adjusted. The disadvantage of this method is that in the case of sparse data or outliers, the convex hull can result in a set with a significant model error. In this case, other approaches, like the following, could be used.

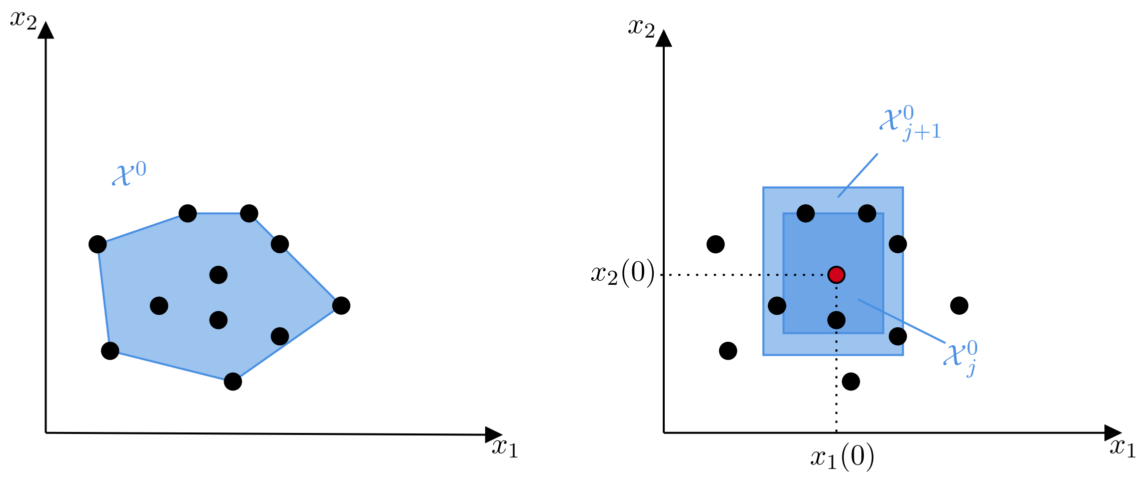


Figure 6.4.: Guesses of the safe sets for a two-dimensional state vector using the convex hull of $\mathcal{D}$ (left) and a set around the initial condition (right).

Gradual expanding set While the previous approach was based on the data points, this is based on expanding a set starting from the initial condition. Since it is necessary that $x_0 \in \mathcal{X}^1$, we can choose a (small) $\delta\mathcal{X} \subset \mathbb{R}^{n_x}$ and define the first guess as

$$\mathcal{X}^1_0 = \{x_0\} \oplus \delta\mathcal{X}$$

If this first guess does not satisfy the necessary conditions for safety, the set is enlarged again. For the j-th iteration we have

$$\mathcal{X}^1_{j+1} = \mathcal{X}^1_j \oplus \delta\mathcal{X}.$$

Figures 29 depicts the two proposed approaches.

6.3.4. Genesis experiments

To build the first dataset $\mathcal{D}^1$, it is necessary to run some preliminary experiments. Since, at this stage, there is no data and no learned function is available, it is not possible to use the proposed approach; hence, constraint satisfaction could be at risk. In our examples, these experiments were run using a randomly generated input. We assumed to measure all the system states accurately and shrunk all constraints with safety margins. The systems were stopped if one state violated these safety margins. In practice, the data of old runs can be used as well.

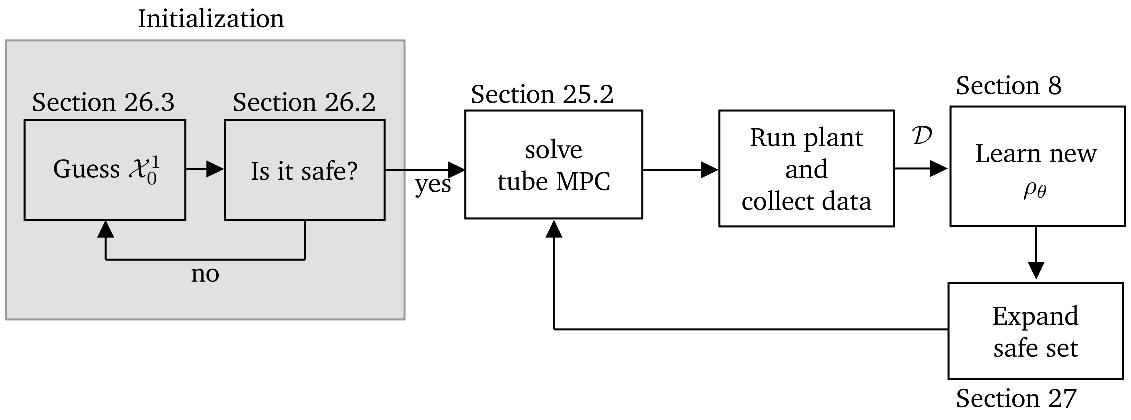


Figure 6.5.: Block diagram of the complete approach. Note that the most time-consuming step is the initialization, which can be done offline, before every run.

6.4. Guaranteed run-to-run expansion of the safe set

Let us suppose that $\mathcal{X}^* \triangleq \mathcal{X}_{j^*}^i$ is safe, for some $j^* \in [1, I_{\max}]$ where $I_{\max}$ is the maximum number of iterations. We want to study what are the conditions under which it is possible to find a larger shrunk safe set $\check{\mathcal{X}} \ominus \Omega(\check{w}_{\max}, \lambda, \mu)$, i.e., a shrunk safe set such that $\mathcal{X}^* \ominus \Omega(w_{\max}, \lambda, \mu) \subset \check{\mathcal{X}} \ominus \Omega(\check{w}_{\max}, \lambda, \mu)$. To achieve this, we first make the following assumptions

Assumption 6.6 *The learned functions $\rho_\theta^i(\cdot)$ are Lipschitz continuous over $\mathbb{X}$ with a common Lipschitz constant $L_{\mathbb{X}}$. That is:*

$$|\rho_\theta^i(x) - \rho_\theta^i(\tilde{x})| \leq L_{\mathbb{X}} \|x - \tilde{x}\|, \quad \forall i \in [1, ..., n_r]. \tag{6.34}$$

Note that $L_{\mathbb{X}} \geq L_{\mathcal{X}^i}, \ \forall \mathcal{X}^i \subseteq \mathbb{X}$.

Assumption 6.7 *There exist a known safe set $\mathcal{X}^*$ with maximum error $w_{max} = \mathfrak{F}(\mathcal{X}^*, \mathcal{D}, \tilde{L}_{\mathbb{X}}, \rho_\theta)$.*

Remark Note that since $L_{\mathbb{X}}$ is not anymore changing with $\mathcal{X}^i$, we can fix λ and μ for all runs (see (58)).

Assumption 6.10 allows us to upper bound all learned functions, of every run, with a common Lipschitz constant. Note that this is not a strong assumption: By Assumption 6.6 we already know a Lipschitz constant of the real function over the set $\mathbb{X}$. One could use this information on the real function and reject any learned function with a Lipschitz constant over $\mathbb{X}$ that is larger than $\tilde{L}_{\mathbb{X}}$ hence enforcing the condition (75) by setting $L_{\mathbb{X}} = \tilde{L}_{\mathbb{X}}$. With this in mind, we formulate the following theorem that guarantees the expansion of the safe set from run to run. The indices i are omitted for simplicity.

Theorem 6.5 (Sufficient condition for run-to-run expansion) *For a given dataset $\mathcal{D}$, let $\mathcal{X}^* \subset \mathbb{X}$ be a safe set with maximum error $w_{max} = \mathfrak{F}(\mathcal{X}^*, \mathcal{D}, \tilde{L}_{\mathbb{X}}, \rho_\theta)$ and control invariant set $\Omega(w_{max}, \lambda, \mu)$. If Assumption 6.11 is satisfied and if there exist a dataset $\breve{\mathcal{D}}$ and a learned function $\breve{\rho}_\theta$ trained with $\breve{\mathcal{D}}$ such that $\breve{w}^*_{max} = \mathfrak{F}(\mathcal{X}^*, \breve{\mathcal{D}}, \tilde{L}_{\mathbb{X}}, \breve{\rho}_\theta)$ is smaller than w_{max}, i.e. $\breve{w}^*_{max} < w_{max}$ then there exists a $\delta\mathcal{X} \subset \mathbb{R}^{n_x}/\{\emptyset\}$ such that the error on the set $\breve{\mathcal{X}} \triangleq (\mathcal{X}^* \oplus \delta\mathcal{X}) \cap \mathbb{X} \subset \mathbb{X}$ defined by $\breve{w}_{max} = \mathfrak{F}(\breve{\mathcal{X}}, \breve{\mathcal{D}}, \tilde{L}_{\mathbb{X}}, \breve{\rho}_\theta)$ is less than or equal than w_{max} i.e. $\breve{w}_{max} \leq w_{max}$. It follows that $\breve{\mathcal{X}}$ is a safe set and $\mathcal{X}^* \ominus \Omega(w_{max}, \lambda, \mu) \subset \breve{\mathcal{X}} \ominus \Omega(\breve{w}_{max}, \lambda, \mu)$. Hence, the safe set and the shrunk safe set expand until $\mathcal{X}^* = \mathbb{X}$.*

Proof Let $\mathcal{X}^*$ be a safe set and let us choose $\delta\mathcal{X}$ equal to a ball $\mathcal{B}(x^*, \delta)$ with center x^* and radius δ defined as follows

$$\mathcal{B}(x^*, \delta) = \{x \in \mathbb{R}^{n_x} | \ \|x^* - x\| \leq \delta\}\,. \tag{6.35}$$

By definition (cf. (70)) we have that

$$\max_{x \in \mathcal{X}^*} \tilde{w}_{n,\text{max}}(x; \breve{\mathcal{D}}, \tilde{L}_{\mathbb{X}}) \leq \breve{w}^*_{\text{max}}.$$

From Lemma 6.1 it follows that the error functions $\tilde{w}_{n,\text{max}}(x; \mathcal{D}, \tilde{L}_{\mathbb{X}})$ are Lipschitz continuous with Lipschitz constant $L_{\mathbb{X}} + \tilde{L}_{\mathbb{X}}$. Hence we have that $\forall n \in [1, ..., n_x]$

$$\max_{x \in \mathcal{X}^* \oplus \mathcal{B}(0,\delta)} \tilde{w}_{n,\text{max}}(x; \breve{\mathcal{D}}, \tilde{L}_{\mathbb{X}}) = \breve{w}_{\text{max}} \leq \breve{w}^*_{\text{max}} + (L_{\mathbb{X}} + \tilde{L}_{\mathbb{X}})\delta.$$

In other words, by expanding the set $\mathcal{X}^*$ with a ball of radius δ the increase in error respect to $\mathcal{X}^*$ can be maximum $(L_{\mathbb{X}} + \tilde{L}_{\mathbb{X}})\delta$. We can now find the δ:

$$\breve{w}_{\text{max}} \leq \breve{w}^*_{\text{max}} + (L_{\mathbb{X}} + \tilde{L}_{\mathbb{X}})\delta = w_{\text{max}} \rightarrow \delta = \frac{w_{\text{max}} - \breve{w}^*_{\text{max}}}{L_{\mathbb{X}} + \tilde{L}_{\mathbb{X}}}$$

where $\delta > 0$ since by assumption $\breve{w}^*_{\text{max}} < w_{\text{max}}$ the Lipschitz constants are strictly positive. It follows that, according to the definition of the control invariant set,

$$\Omega(\breve{w}_{\text{max}}) \subseteq \Omega(w_{\text{max}})$$

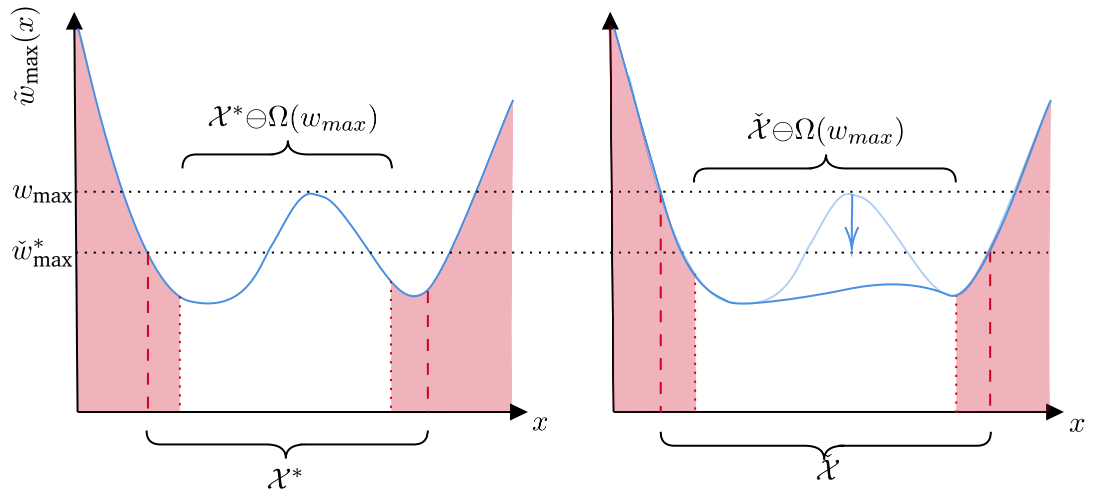


Figure 6.6.: Depiction of the idea behind Theorem 6.5 for a one-dimensional case. The blue graph (left) represents the error function for the dataset $\mathcal{D}$. After generating the next dataset $\check{\mathcal{D}}$, by the Assumption 6.11 $\breve{w}^*_{\max}$ is smaller. Hence, the new set (projection of the red area over x) can be safely expanded until $\breve{w}_{\max} = w_{\max}$ since, at that point $\breve{w}_{\max} = \Omega(w_{\max})$.

where the constant λ and μ are the same so are omitted for simplicity. Since $\check{\mathcal{X}} = (\mathcal{X}^* \oplus \mathcal{B}(0,\delta)) \cap \mathbb{X}$ it follows that

$$\begin{aligned} \text{if } \check{\mathcal{X}} \subset \mathbb{X}, \quad \mathcal{X}^* \ominus \Omega(w_{\max}) &\subset \check{\mathcal{X}} \ominus \Omega(\breve{w}_{\max}) \\ \text{else if } \check{\mathcal{X}} = \mathbb{X}, \quad \mathcal{X}^* \ominus \Omega(w_{\max}) &\subseteq \check{\mathcal{X}} \ominus \Omega(\breve{w}_{\max}) \end{aligned}$$

that completes the proof. See Fig. 31 for a graphical description of the main idea.

Remark Note that the Theorem 6.5 provides a way to guarantee the existence of a larger safe set *a priori*; hence, there is no need to check the safety conditions again.

Remark Theorem 6.5 assumed that the new dataset and the new learned function decrease the maximum error on the previous safe set. This, in general, cannot be ensured a priori. Nevertheless, it is possible to check if, with the new machine learning model, the error has decreased or not. If not, the model can be rejected, and another can be trained to decrease the maximum error.

Clearly, there is a limit to how much the error can be reduced. For example, it is impossible to reduce the error below the measurement noise $v_{\max}$. If the error cannot be reduced, it follows, from the same arguments of Theorem 6.5, that the set at least does not shrink.

6.4.1. Set-point error

Theorem 6.5 guarantees that both safe set and shrunk safe set expand at every batch if the upper bound on the error decreases from batch to batch until the safe set reaches the constraints $\mathbb{X}$. It remains to guarantee that this expansion will allow the system to reach an area around the reference point from a fixed x_0 at the end of the batch T_f. As previously mentioned, in this thesis, we consider the case where the initial condition is fixed at every batch. Let us take a batch bioreactor producing a given product P as an example. For such processes, the initial condition is often fixed, and a given concentration of metabolites in the product must be reached within some tolerance at the final batch time T_f. This can be seen as a reachability problem of the real system subject to input and state constraints. To this aim, we make the following assumption:

Assumption 6.8 (Reachability of the nominal system) *Let $\mathcal{B}(x_r, R)$ be a ball with center x_r and radius $R \in \mathbb{R}^{n_x}$. There exist a safe set $\mathcal{X}_r \subseteq \mathbb{X}$ such that the system $\dot{\bar{x}} = A\bar{x}(t) + B\bar{u}(t) + g_\theta(\bar{x}(t))$ under the implicit control law of* (63) *can reach $\mathcal{B}(x_r, R)$ at time T under the constraints $\bar{x} \in \mathcal{X}_r - \Omega(w_{max,\mathcal{X}_r})$ and $\bar{u} \in \mathbb{U} - G_r$. In other words, the $\mathcal{B}(x_r, R) \subseteq \mathcal{S}$ where $\mathcal{S}$ is the reachable set of the nominal system i.e.*

$$\mathcal{S} \triangleq \{\bar{x} \in \mathbb{R}^{n_x} |\ \bar{x}(T_f; x_0, u(\cdot)) \in \mathcal{S},\ \ \bar{x}(t) \in \mathcal{X}_r,\ \ \bar{u}(\cdot) \in \mathbb{U} - G_r\} \tag{6.36}$$

where $\bar{x}(T_f; x_0, u(\cdot))$ refers to the evolution of the nominal state evaluated at time T_f with initial condition x_0 and control policy $u(\cdot)$ and $G_r = G(\Omega(w_{max,\mathcal{X}_r}))$.

Theorem 6.6 (Reachability of the real system) *If Assumption 6.12 is true, it follows that the real state $x(T; x_0, u(\cdot)) \in \mathcal{R}$ where*

$$\mathcal{R} \triangleq \left\{ x \in \mathbb{R}^{n_x} | x \in \mathcal{B}(x_r, R) \oplus \Omega^i \right\}. \tag{6.37}$$

Furthermore, we can find an upper bound on the tracking error at time T:

$$\|x(T_f) - x_r\| \leq R + \left(\alpha_{max}(\bar{P})\right)^{-1/2},$$

where $\alpha_{max}(\bar{P})$ is the maximum eigenvalue of the matrix

$$\bar{P} \triangleq \frac{\lambda}{\mu w^2_{max,\mathcal{X}_r}} P.$$

Proof The first part of the theorem can be easily proven, recalling the definition of the robust control invariant set (cf. Definition 6.1). Let $\Omega \triangleq \Omega(w_{\max,\mathcal{X}_r})$ we have that

$$x(T_f) - \bar{x}(T_f) \in \Omega.$$

For the Assumption 6.12 we know that $\bar{x}(T_f) \in \mathcal{B}(x_r, R)$ hence

$$x(T_f) \in \{x \in \mathbb{R}^{n_x} | \{\bar{x}(T_f)\} \oplus \Omega,\ \forall \bar{x}(T_f) \in \mathcal{B}(x_r, R)\} = \Omega \oplus \mathcal{B}(x_r, R).$$

The second part can be proven by noticing that the maximum Euclidean distance from $\bar{x}(T_f)$, $\|x(T_f) - \bar{x}(T_f)\|$ is the major semi-axis of the hyperellipsoid $x^\mathsf{T}\bar{P}x = 1$ which the inverse of squared root of its largest eigenvalue. This ball is defined by the following set

$$\Omega^c = \left\{x \in \mathbb{R}^{n_x} | \|x - \bar{x}(T_f)\| \leq \left(\alpha_{\max}(\bar{P})\right)^{-1/2}\right\}.$$

Note that since $P > 0$ and $\lambda/(\mu w^2_{\max,\mathcal{X}_r}) > 0$ then $\bar{P} > 0$ hence $\alpha_{\max}(\bar{P}) > 0$. From this, it follows that

$$\begin{aligned}
\|\bar{x}(T_f) - x_r\| + \|x(T_f) - \bar{x}(T_f)\| &\leq R + \left(\alpha_{\max}(\bar{P})\right)^{-1/2}, \\
\|\bar{x}(T_f) - x_r + x(T_f) - \bar{x}(T_f)\| &\leq \|\bar{x}(T_f) - x_r\| + \|x(T_f) - \bar{x}(T_f)\|, \\
\|x(T_f) - x_r\| &\leq R + \left(\alpha_{\max}(\bar{P})\right)^{-1/2},
\end{aligned}$$

which completes the proof. Figure 32 shows graphically the idea of the Theorem.

Corollary 6.7 *From Theorem 6.5, it follows that there is a run $i^* \in \mathbb{N}$ such that the real system can reach the reference with maximum error $R + \left(\alpha_{max}(\bar{P})\right)^{-1/2}$.*

Proof From Theorem 6.5 if follows that $\mathcal{X}^i \subset \mathcal{X}^{i+1}$ if $\mathcal{X}^{i+1} \subset \mathbb{X}$ and $\mathcal{X}^i$ are safe $\forall i \in \mathbb{N}$. Since, from Assumption 6.12, it follows that there exist a set $\mathcal{X}^* \subseteq \mathbb{X}$ such that $\|x(T) - x_r\| \leq R + \left(\alpha_{\max}(\bar{P})\right)^{-1/2}$ then, there must be a $i^* \in \mathbb{N}$ such that $\mathcal{X}_r \subseteq \mathcal{X}^*$ hence the reference is reachable with error $R + \left(\alpha_{\max}(\bar{P})\right)^{-1/2}$.

The previous Theorem and its Corollary give useful bounds on the tracking error for the real system that can be used to ensure that the required tracking tolerance is respected.

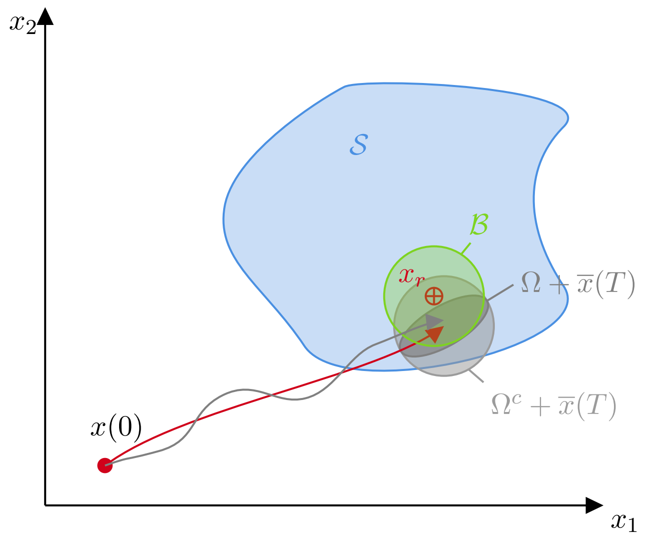


Figure 6.7.: Illustration of the reachable set for a two-state system. The nominal trajectory (red) finishes at time T in the ball $\mathcal{B}(x_r, R)$ (green set). The real system (gray line) evolves around the nominal. Note that the real system might fall outside the ball $\mathcal{B}(x_r, R)$.

6.4.2. Design of experiment

To ensure the expansion of the safe set at every run, it is necessary that the model error over the current safe set $\mathcal{X}^i$ decreases with the new data used for the run $i+1$ (cf. Assumption 6.11). Also, from Theorem 6.5, we see that the size of the expansion is proportional to how much the error has decreased. Since the error function $\tilde{w}_{\max}(x)$ is available, it is natural to use this information for an optimal design of experiment, i.e., to try to decrease the error with as much as possible, hence expanding the safe with fewer runs. The design of experiment is planned as follows: As long as x_r is not reachable by the plant with the required tolerance, the reference for the run i is set to

$$x_r^i = \arg\max_x \tilde{w}^i_{\max}(x). \tag{6.38}$$

By doing so, the plant will collect data close to x_r^i and possibly reduce the modeling errors. In the following section, we provide an example.

6.4.3. Example

Let us consider again the example with which we started this chapter. The system is given by

$$\dot{x}(t) = \begin{bmatrix} -1 & 2 \\ -3 & 4 \end{bmatrix} x(t) + \begin{bmatrix} 0.5 \\ -2 \end{bmatrix} u(t) + \rho(x(t)), \tag{6.39}$$

where $\rho(x(t)) = [0, -0.25x_2^3]^\top$ is unknown. The system constraints are $\mathbb{X} = \{x \in \mathbb{R}^2 |\ [-5, -1.2] \leq x \leq [5, 1.2]\}$ and $\mathbb{U} = \{u \in \mathbb{R}|\ -10 \leq u \leq 10\}$. The first data set was obtained by running a single experiment as outlined in Section 26.4. The collected data are artificially perturbed with bounded white noise with $v_{\max} = 0.1$ and then used to train a neural network $\rho^1_\theta(\cdot)$ with two layers and five neurons for each layer. The simulations were run using HILO-MPC [150]. The first guess of the safe set is obtained using the convex hull method (cf. Section 26.3), which respected the safety conditions (cf. Definition 6.3). Our approach is compared with a nominal shrinking horizon MPC (non-tube-based) using the hybrid model (48) since the model error over the set $\mathbb{X}$ is $w_{\max,\mathbb{X}} = 1.15$, which is too large and renders the normal tube MPC unfeasible. The plant initial conditions are $x(0) = [1, -0.3]$. The set was enlarged at every run according to Theorem 6.5. The goal is reaching the reference $x_2(T) = 1$. The set was expanded only in the direction of increasing x_2, since in this case, it is clear that x_2 needs to move towards increasing values to reach the reference. Design of experiment was used as described in Section 27.2. Figure 33 shows the simulation result for the first run and Figure 34 for the last run. While our approach satisfies the constraints, the nominal MPC violates them. In total, 55 runs were necessary to expand the set safely to the required reference. In comparison, without design of experiment 95 runs where necessary (results are not shown). The design of experiment reduced the number of runs by 55%.

Theorem 6.5 gives a way to expand the safe set guaranteed to return a safe set *a priori*, but it is rather conservative; hence, a large number of runs might be needed to reach the reference safely. In the next section, we will propose an algorithm that allows a faster expansion of the safe set at the cost of losing *a priori* guaranteed safety.

6.5. Accelerated run-to-run expansion of the safe set

Note that some of the conditions of Theorem 6.5 are only sufficient conditions that allow expanding the set safely. In practice, we can try to decrease further the number of runs necessary to reach the reference by allowing a larger expansion rate of the set. By doing so, we lose the *a priori* guarantee that the resulting expanded set is safe. Thus, we must ensure that the safety conditions are respected before every run. Since the conditions can be checked offline, this does not influence the real-time capability of the algorithm. Algorithm 3 shows how this is achieved.

Algorithm 3 Algorithm for accelerated set expansion. For simplicity, the index of the run i is omitted.

```
 1: Given $\mathcal{D}$, $\rho_\theta$, $\alpha^*_{\text{min}}(P)$, $P$, $\lambda^*_0$, $\mathcal{X}_0$
 2: Set $I_{\text{max}}$, $\delta\hat{\mathcal{X}}$ and $k \in [0, 1)$
 3: for $j \in [1, I_{\text{max}}]$ do                              ▷ Try to increase the set
 4:     $\hat{\mathcal{X}}_{j+1} \leftarrow \mathcal{X}_j \oplus \delta\bar{\mathcal{X}}$
 5:     $w_{\text{max},\hat{\mathcal{X}}_{j+1}} \leftarrow \mathfrak{F}(\rho_\theta, \mathcal{D}, L_{\hat{\mathcal{X}}_{j+1}}, \hat{\mathcal{X}}_{j+1})$
 6:     $\lambda \leftarrow \lambda^*_0 - 2\frac{\|P\| L_{\hat{\mathcal{X}}_{j+1}}}{\alpha_{\text{min}}(P)}$
 7:     if Safety conditions 6.3 are satisfied then
 8:         $\mathcal{X}_{j+1} \leftarrow \hat{\mathcal{X}}_{j+1}$
 9:     else
10:         $\delta\hat{\mathcal{X}} \leftarrow k\delta\hat{\mathcal{X}}$
11:     end if
12: end for
13: return $\mathcal{X}_{j+1}$
```

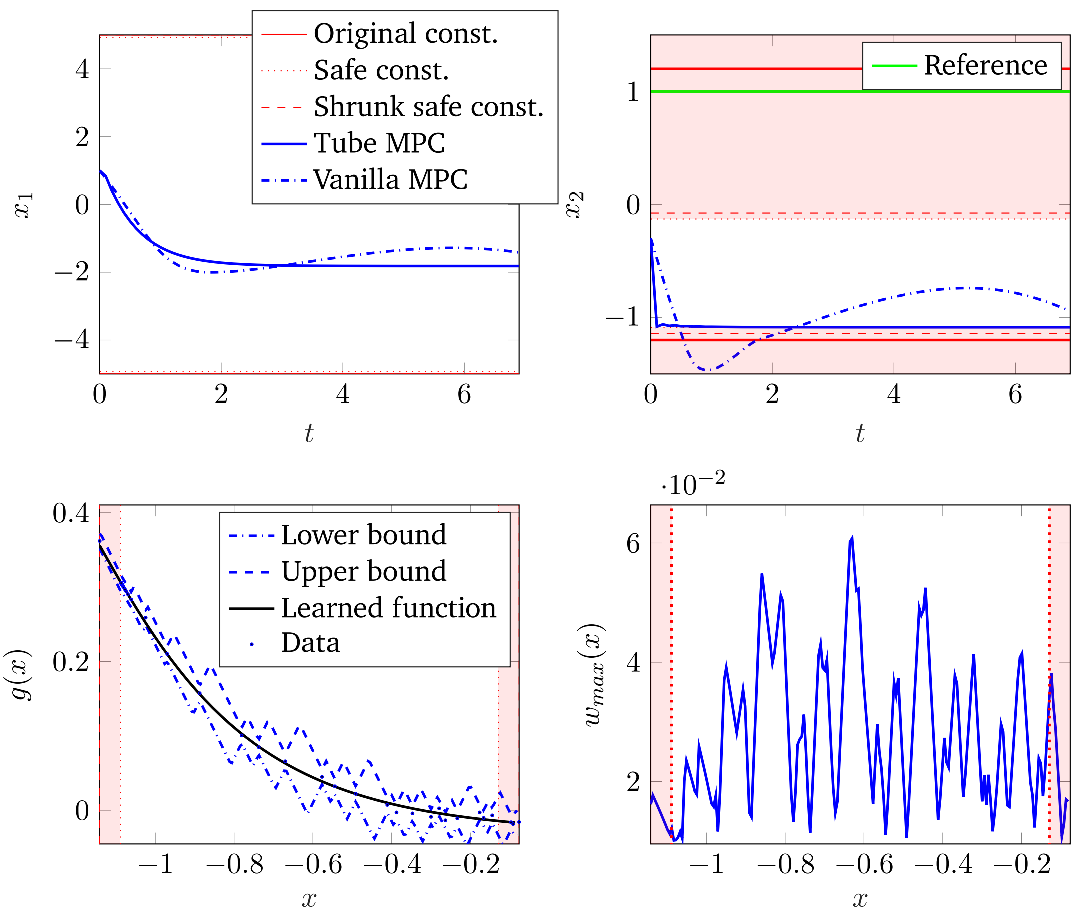


Figure 6.8.: Results of the simulation for the first run. In the first row, the state of the plant using the proposed approach (Tube MPC) and vanilla nonlinear MPC (Vanilla MPC) are shown. The second row shows the learned function (on the left) with the bounds computed with (66) and the upper bound on the model error (right).

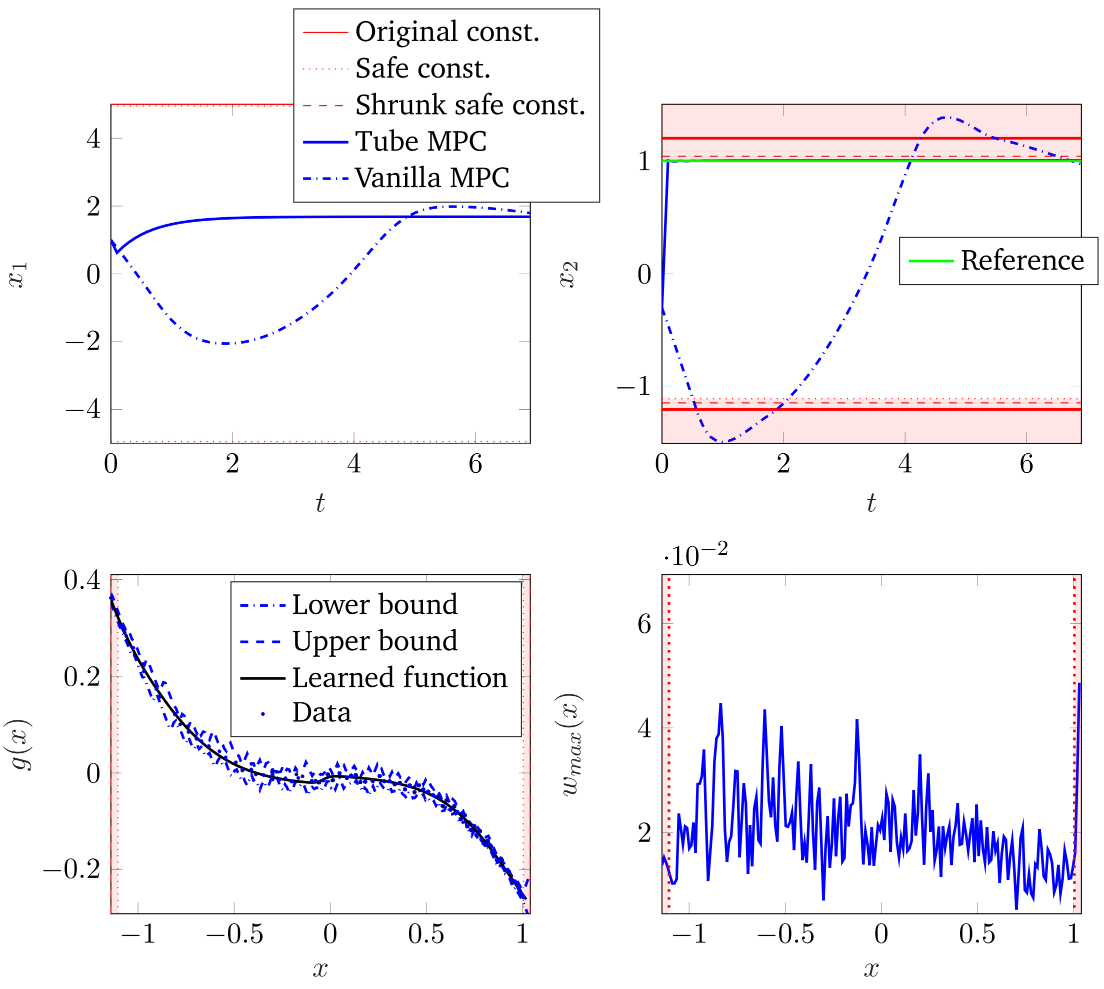


Figure 6.9.: Results of the simulation for the last run. In the first row, the state of the plant using the proposed approach (Tube MPC) and vanilla nonlinear MPC (Vanilla MPC) are shown. The second row shows the learned function (on the left) with the bounds computed with (66) and the upper bound on the model error (right).

6.5.1. Example

We take the same example shown in Section 27.3, but we apply the accelerated expansion of the safe set where $\delta\mathcal{X} = \mathcal{B}(0, 0.1)$. Figure 35 shows the results of the simulation. In this case, only 13 runs were necessary to reach the reference.

6.6. Example: Robotic arm

We consider the model of a one-link robotic arm modeled as a linear torsional spring connected to a motor [155]

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -48.6 & -1.25 & 48.6 & 0 \\ 0 & 0 & 0 & 1 \\ 19.5 & 0 & -19.5 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 21.6 \\ 0 \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1.3 \end{bmatrix} g(x(t)). \qquad (6.40)$$

where $g(x) = \sin(x_3)$ is assumed to be unknown and must be learned from data. The objective is to bring $x_3(t)$ to the reference $x_{3,\mathrm{r}} = 0.5$, while satisfying the box constraints on the states $\mathbb{X} = \left\{x \in \mathbb{R}^{n_x} | \; x^{\mathrm{lb}} \leq x(t) \leq x^{\mathrm{ub}}\right\}$ and on the input $\mathbb{U} = \{u \in \mathbb{R}^{n_u} | \; u^{\mathrm{lb}} \leq u(t) \leq u^{\mathrm{ub}} \, \forall t \in [0, T_f]\}$ where $x^{\mathrm{ub}} = [1.57, 10, 1.57, 10]^\top$, $x^{\mathrm{lb}} = [-1.57, -10, -1.57, -10]^\top$, $u^{\mathrm{ub}} = 10$ and $u^{\mathrm{lb}} = -10$.

To generate the first data set, one experiment was run as described in 26.4. The collected data are artificially perturbed with bounded white noise with $v_{\mathrm{max}} = 0.05$ and then used to train a neural network $\rho^1_\theta(\cdot)$ with one layer with ten neurons and ReLU activation functions. The optimization problems were solved using HILO-MPC [150]. For this case, we used the accelerated expansion of the safe set with $\delta = 0.05$ using design of experiment. The reference was reached after eight runs (cf. Figures 36,37).

6.7. Summary

We proposed a robust nonlinear model predictive control scheme with a shrinking horizon for repetitive processes that use machine learning to improve the model and increase performance from run to run. Robust constraint satisfaction and repeated feasibility are guaranteed using a tube-based approach for nonlinear systems. Since the uncertainty might be large in some regions of the state space, at every run, the system is forced to stay in a safe region where the uncertainty is small enough that a feasible solution can be guaranteed despite uncertainty. For this safe region, the robust control invariant set is computed by upper bounding the error between the real system and machine learning

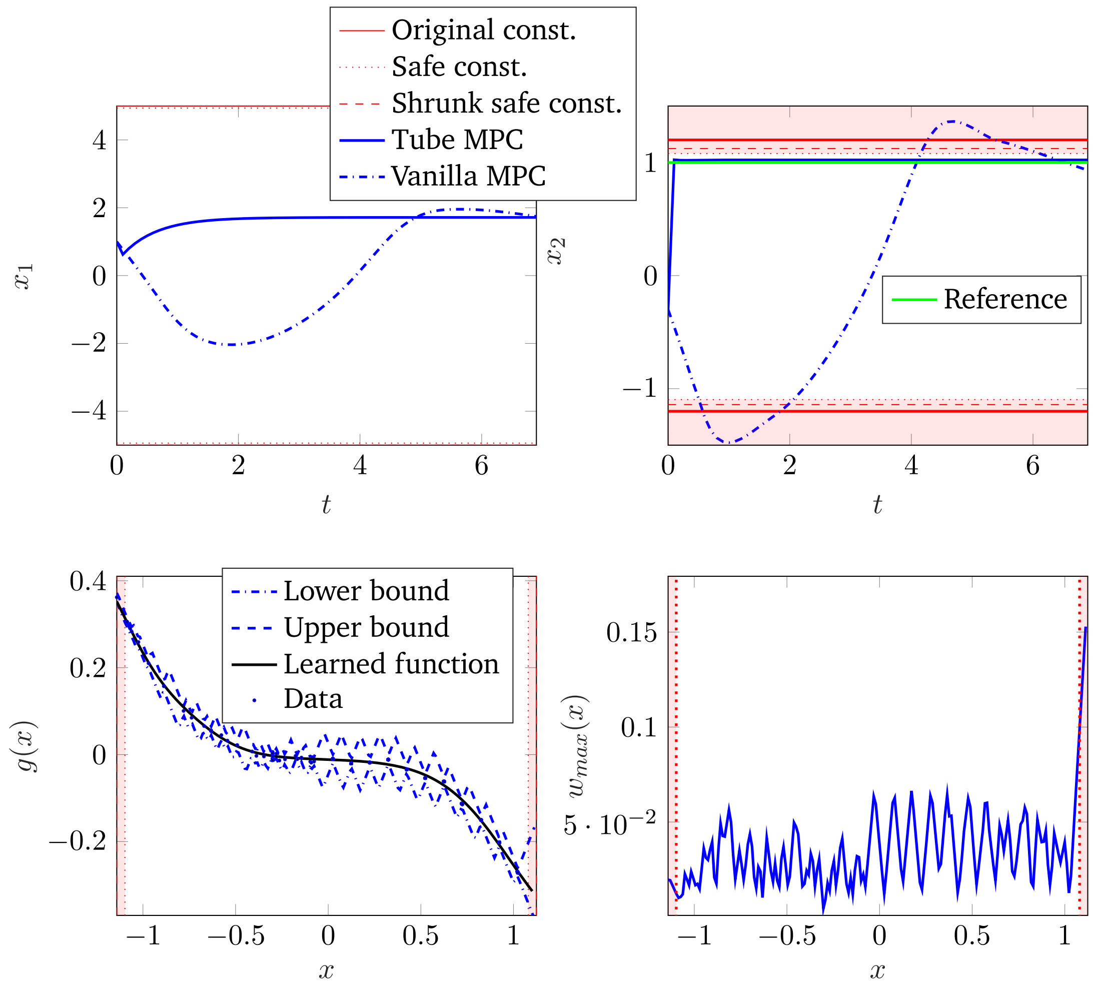


Figure 6.10.: Results of the 13-th run for in the case of accelerated expansion of the safe set. In the first row, the state of the plant using our approach (Tube MPC) and vanilla nonlinear MPC (Vanilla MPC). The second row shows the learned function (on the left) with the bounds computed with (66) and the upper bound on the model error (right).

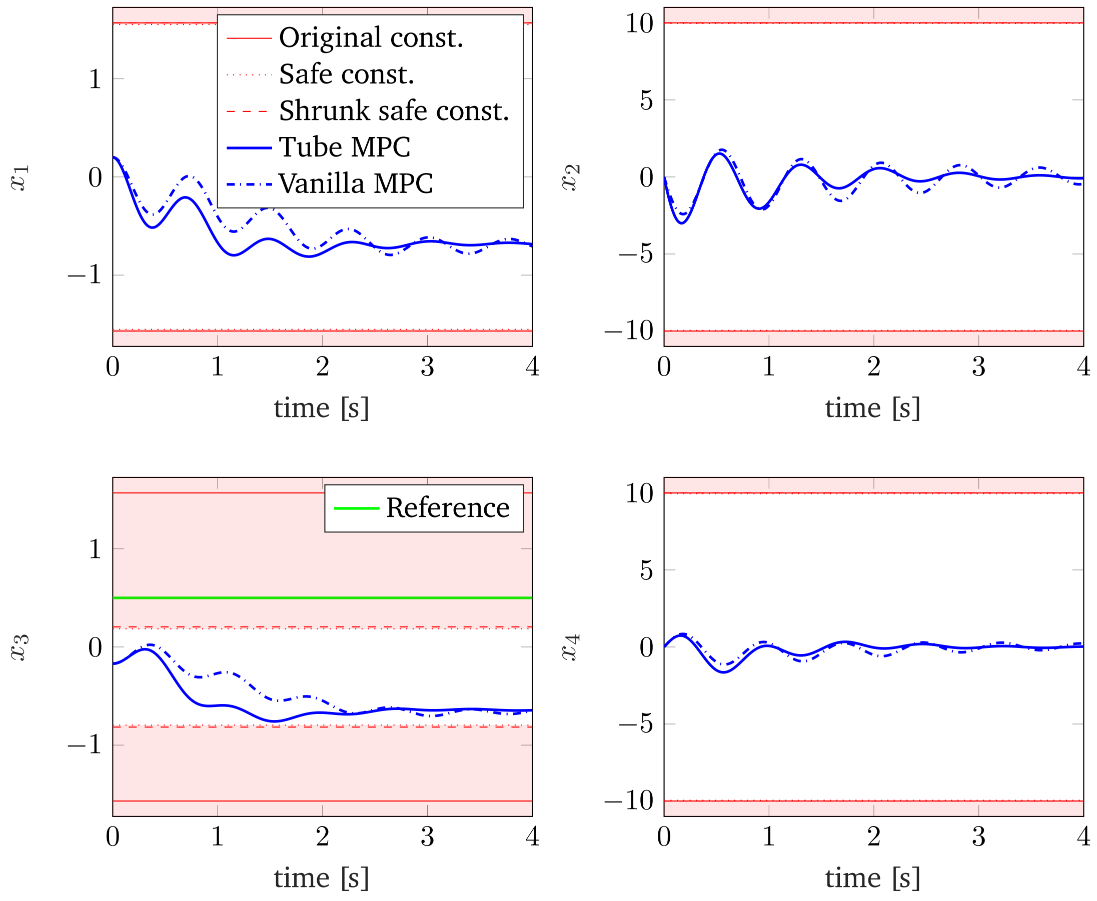


Figure 6.11.: Results of the robotic arm model. First run.

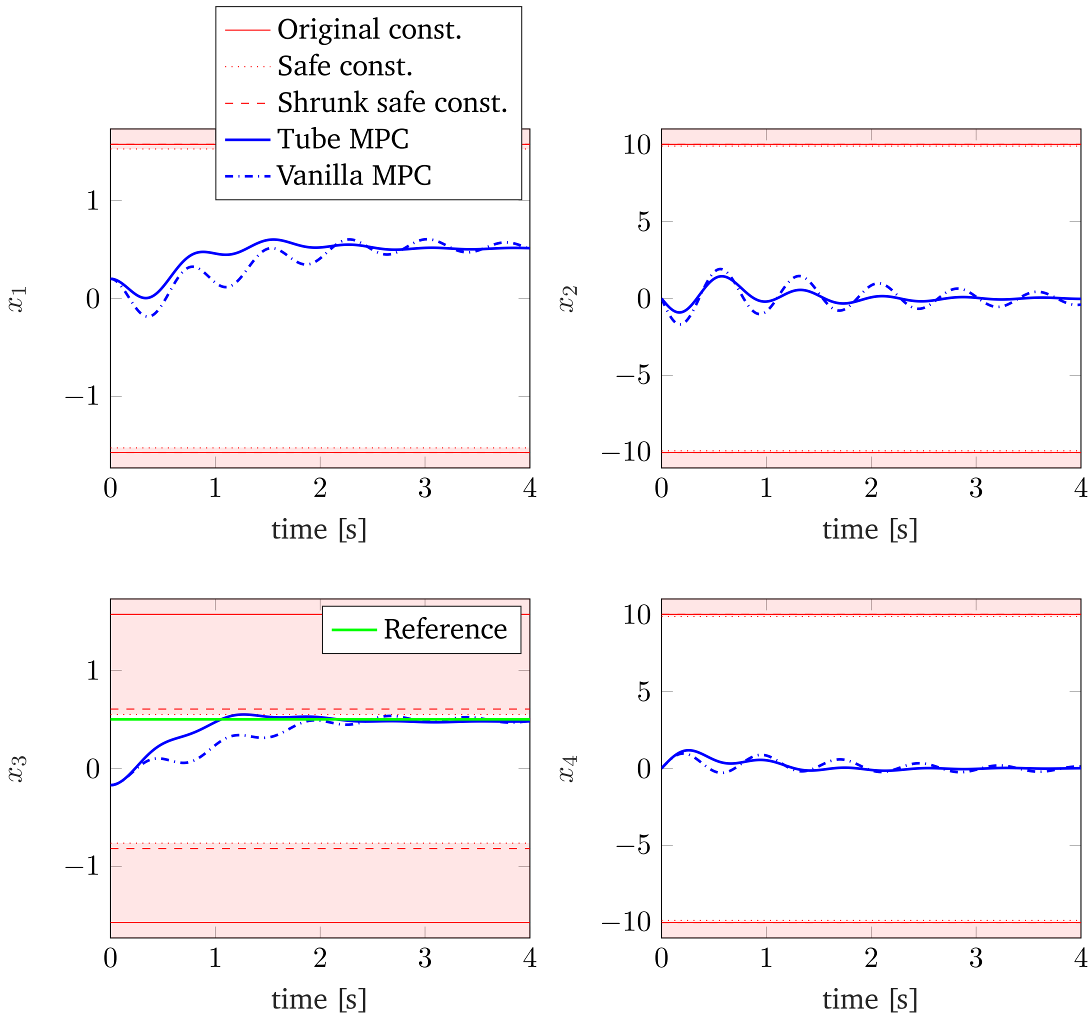


Figure 6.12.: Results of the robotic arm model. 8^{th} run.

model using Lipschitz continuity. We proved that the safe set expands from run to run under the assumption of decreasing modeling error. Furthermore, we provided an upper bound on the tracking error for the real unknown system. The proposed approach is particularly useful for new processes with large model uncertainty or for improving existing processes by exploring new state regions where the model might have a large uncertainty and constraint satisfaction is critical.

7. HILO-MPC: A Toolbox for Learning-supported Control, Modelling and Estimation

While the number of new methods and theoretical findings using machine learning for predictive control and optimization is increasing rapidly, a software tool that facilitates the solutions to these problems is still missing. This chapter presents HILO-MPC: An open-source and flexible Python library that allows a quick and efficient implementation of machine learning-supported optimization, model predictive control, and estimation problems. HILO-MPC was used in Chapter 18 and 23 to generate the results of the simulations. In this chapter, we will give an overview of the capability of the toolbox and show its syntax with the help of some examples.

7.1. Introduction

Machine learning libraries help control practitioners build machine learning models, but using these models in the control design requires ad-hoc solutions, which are time-consuming, prone to bugs, and not easily transferable to other applications. HILO-MPC is an open-source Python library that aims to provide a way to implement machine learning models easily into a wide variety of control, optimization, and estimation problems. HILO-MPC interfaces with state-of-the-art machine learning libraries, PyTorch [143] and TensorFlow [1] to train neural networks, and uses scikit-learn [145] or an in-house learning library to train Gaussian processes. Once trained, the models can be deployed in the control or estimator design. HILO-MPC does not only focus on machine learning: To speed up the development time, HILO-MPC provides ways to define and solve a broad spectrum of predefined optimal control and estimation problems quickly and with minimum effort. Tab. 1 summarizes the different problems that can be solved with the version 1.0.3 of HILO-MPC.

The philosophy that HILO-MPC follows is ease of use and modularity. Ease of use is achieved by providing an intuitive high-level interface for the problem definition. At the same time, modularity is guaranteed by ensuring effortless interfaces between the tools in the toolbox (see Fig. 38). The ease of use makes it suitable for teaching purposes, and the modularity and personalization capabilities render it a useful tool for more advanced applications such as research.

Comparing HILO-MPC with other toolboxes is challenging since it can solve a wide variety of control and estimation problems. Here, we consider only open-source toolboxes primarily focused on solving model predictive control problems. The software ACADO [83] and ACADOS [191] aims at efficiency and fast solutions while keeping embedded applications in mind. MPT3 [72] focuses on computational geometry, particularly useful for some classes of robust MPC approaches. YALMIP [112] instead offers a broad spectrum of optimization problems, such as bilevel optimization, mixed-integer programming, and some MPC schemes. The toolbox do-MPC [115] implements robust multi-stage MPC and Moving Horizon Estimation. MPC-Tools [162] offers some interfaces that help to build MPC and MHE control problems in CasADi. In Tab. 2, we summarize the main differences between the toolboxes.

Using machine learning models in the previously mentioned toolboxes is not straightforward. HILO-MPC solves this problem by providing wrappers with machine learning libraries. Furthermore, HILO-MPC focuses on a lightweight and intuitive interface that makes it easy for beginners.

The backbone of HILO-MPC is CasADi [9]. CasADi is a tool for algorithmic differentiation and optimization, and it is used to build models, optimization, and estimation problems. If TensorFlow or PyTorch are used, HILO-MPC offers an easy way to interface with those libraries and automatically translate the models into CasADi-compatible structures. Hence, the CasADi problem is defined and solved. CasADi has interfaces with a wide range of solvers for linear and quadratic programming (CLP, qpOASES [61]), nonlinear programming such as IPOPT, [197], quadratic mixed-integer programming (CPLEX [43], GUROBI [69]), for non-quadratic nonlinear mixed-integer problems (BONMIN [27], KNITRO [135]) and large nonlinear problems WORHP [134].

7.2. HILO-MPC modules

The toolbox can be divided into five modules: Model, control, observer, machine learning, and embedded (cf. Fig 38). Each of these contains a set of tools, each one of which solves a different problem. The model module is used to generate dynamic models. The control module uses the model module to generate, e.g., optimal control and predictive control problems. The observer module also uses the model module to generate state and parameter estimators. The machine learning model is responsible for defining and training

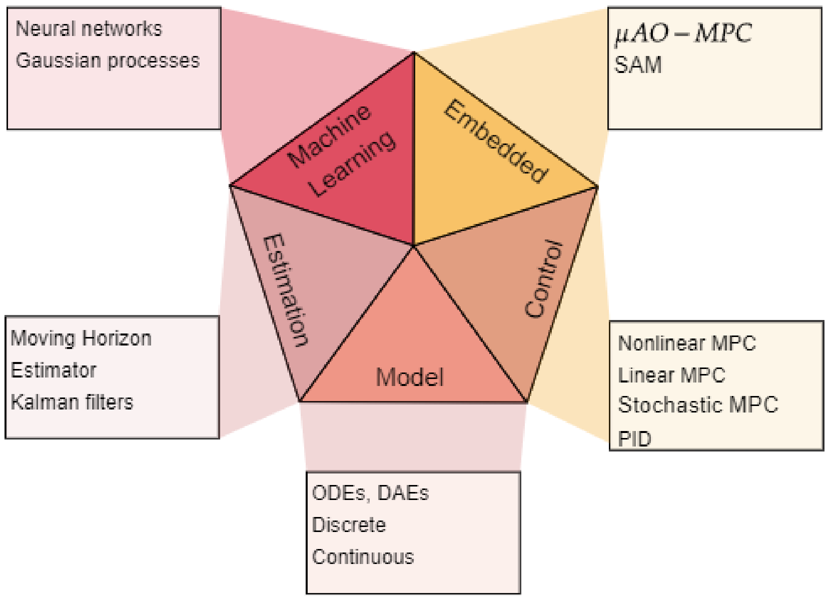


Figure 7.1.: Overview HILO-MPC

Table 7.1.: Current tools implemented in HILO-MPC.

Modules	Tools
Models	Linear/nonlinear Time-invariant/variant Continuous and discrete ODEs, DAEs
Controllers	Optimal Control, Linear discrete-time MPC Nonlinear MPC Trajectory tracking MPC Path-following MPC PID LQR
Machine Learning	Neural Networks Gaussian processes
Observers	Moving Horizon Estimation Kalman filter Extended Kalman filter Unscented Kalman filter Particle filter
Embedded	*μAO-MPC* SAM

Table 7.2.: Comparison overview on the different open-source MPC software. ML: machine learning, MPC: model predictive control, MHE: moving horizon estimator, LMPC: linear MPC, PFMPC: path-following MPC, KF: Kalman filter, EKF: extended Kalman filter, UKF: unscented Kalman filter.

	Problems	ML	Interface	Focus	Article
HILO-MPC	MPC, PFMPC MHE,KF,EKF,UKF SMPC	Yes	Python	Machine Learning	[150]
do-mpc	MPC, MHE Multistage MPC	No	Python	Multistage MPC	[115]
ACADO	MPC	No	C++, Matlab	Embedded	[83]
ACADOS	MPC, MHE	No	C, Matlab, Octave, Python, Simulink	Embedded	[191, 192]
NMPC tools	MPC, MHE	No	Octave, Python.	MPC,MHE	[162]
MPT3 Toolbox	LMPC	No	Matlab	Comp. Geometry	[72]
YALMIP	MPC, MHE	No	Matlab	Optimization	[112]

machine learning models that can be used in any previous modules. The embedded module interfaces with embedded MPC software for linear MPC (*μAO-MPC*) [216] and nonlinear MPC (SAM) [88]. We will briefly describe the different modules in more detail in the following sections.

7.3. Modeling module

At the core of HILO-MPC sits the modeling module. It is a high-level interface to generate representations of dynamical systems that can be used for model-based controllers and observers, like MPC or MHE, or inside a control loop to simulate the plant. The system properties supported by HILO-MPC are reported in column "Models" of Tab. 1.

A general time-variant continuous-time nonlinear system can be modeled using the following DAEs

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), z(t), u(t), p), \\ 0 &= q(t, x(t), z(t), u(t), p), \\ y(t) &= h(t, x(t), z(t), u(t), p), \end{aligned} \tag{7.1}$$

where $x(t) \in \mathbb{R}^{n_x}$ is the differential state vector, $z(t) \in \mathbb{R}^{n_z}$ the vector of algebraic states, $u(t) \in \mathbb{R}^{n_u}$ is the input vector and $p \in \mathbb{R}^{n_p}$ is a vector of parameters. The function

$f\colon \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}^{n_x}$ represents the ODEs of the model and the function $q\colon \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}^{n_z}$ describes the algebraic equations. This forms a semi-explicit DAE system overall [1]. The function $h\colon \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}^{n_y}$ describes the measurement equations mapping to some measurable quantity of the system. Note that if these measurement equations are omitted during the setup of the HILO-MPC model object, the controllers and observers connected to this model will assume that all states are measurable.

Furthermore, HILO-MPC also supports the modeling of discrete-time systems and discretization of continuous-time models. The available discretization methods are explicit Runge-Kutta methods up to the fourth order and implicit collocation schemes [24]. For the explicit Runge-Kutta methods, a series of Butcher tableaus are available, while the implicit collocation schemes only support Gauss-Legendre polynomials and Gauss-Radau polynomials for the calculation of the collocation points. If a nonlinear model is given, it is possible to linearize this model around a steady-state point. It is also possible to create linear models directly by supplying the required matrices during model setup.

As an example, we show how to set up a simple bike model, which will also be used as an example model in the following sections:

$$\begin{aligned}
\dot{p}_\text{x} &= v \cos(\phi(t) + \beta) \\
\dot{p}_\text{y} &= v \sin(\phi(t) + \beta) \\
\dot{v} &= a \\
\dot{\phi} &= v/l_\text{r} \sin(\beta) \\
\beta &= \arctan\left(\frac{l_r}{l_\text{r} + l_\text{f}} \tan(\delta)\right).
\end{aligned}$$

Where p_x and p_y are the x and y coordinates of the bike center of mass, v is the module of the velocity of the center of mass, ϕ is the orientation angle of the vehicle with respect to the x coordinate. The inputs are the acceleration of the center of mass a and the steering angle δ. l_r is the distance between the center of mass and the rear wheel, and l_f the distance between the center of mass and front wheel. One possible way to set up the model is as follows

```python
# Import the Model class
from hilo_mpc import Model
```

[1]Note that HILO-MPC only supports semi-explicit DAE systems of index 1. The index 1 indicates that the corresponding DAE system can be transformed into a pure ODE system by differentiating the algebraic equations once.

```
# Initialize empty model
model = Model(name='Bike')

# Define model equations in string form
equations = """
# ODEs
dpx/dt = v(t)*cos(phi(t) + beta)
dpy/dt = v(t)*sin(phi(t) + beta)
dv/dt = a(k)
dphi/dt = v(t)/lr*sin(beta)

# Algebraic equations
beta = arctan(lr/(lr + lf)*tan(delta(k)))
"""
# Pass the equations to the model
model.set_equations(equations=equations)

# Sampling time in seconds
dt = 0.01

# Setup the model
model.setup(dt=dt)

# Pass the initial conditions
model.set_initial_conditions(x0=[0,0,0,0])
```

where the whole model equations are supplied in form of a string. HILO-MPC is able to parse the most common functions and operations in this format. Note also that the function `beta` is automatically substituted in the ODE.

Alternatively, the same model can be defined using variables directly as follows

```
# Initialize empty model
model = Model(name='Bike')

# Set the variables
inputs = model.set_inputs(['a', 'delta'])
states = model.set_states(['px','py','v','phi'])

# Unwrap states
px = states[0]
py = states[1]
v = states[2]
phi = states[3]

# Unwrap states
a = inputs[0]
```

```
delta = inputs[1]

# Parameters
lr = 1.4  # [m]
lf = 1.8  # [m]
beta = ca.arctan(lr / (lr + lf) * ca.tan(delta))

# ODE
dpx = v * ca.cos(phi + beta)
dpy = v * ca.sin(phi + beta)
dv = a
dphi = v / lr * ca.sin(beta)

# Pass the differential equations to the model
model.set_differential_equations([dpx, dpy, dv, dphi])

# Sampling time in seconds
dt = 0.01

# Setup the model
model.setup(dt=dt)

# Pass the initial conditions
model.set_initial_conditions(x0=[0,0,0,0])
```

While the first way is more compact, it might be prone to misspelling errors, hence it is recommended for small models. The second way is more verbose but less prone to errors. Units, labels, and descriptions of all variables can also be set for convenience and for use in the plotting functionality.

Once the initial conditions are set, to simulate the system of `n_steps` time steps `dt`, we system can be simulated. This can be achieved, for example,

```
for i in range(n_steps):
    model.simulate(u=u,p=p)
```

where u is the input to the system and p the vector of parameters. Note that, for every integration interval dt the input is kept constant. Finally, HILO-MPC makes it easy to store, access, and plot solutions generated by simulations. Every time the method `simulate` is run, the solution is automatically saved in the `solution` object. The states and input evolution of the system can be easily visualized with the command `model.solution.plot()`. HILO-MPC uses matplotlib [84] and bokeh [26] as plotting libraries.

7.4. Machine Learning module

This module is responsible for the definition and training of machine learning models. The current version 1.0.3 contains two approaches: Artificial feed-forward fully connected neural networks and Gaussian processes. Once the models are trained on the given dataset, they can be easily deployed alone or together with a first-principle model. Here, we focus on regression problems, i.e., the result of the training is a function whose output is a continuous variable. Classification problems, where the output takes discrete variables, are not discussed here since they play a minor role in control and estimation applications. It is nevertheless possible to train such problems with HILO-MPC.

7.4.1. Artificial Neural Networks

We already covered the basics of artificial neural networks in Section 6.1. Currently, HILO-MPC supports feed-forward, fully-connected NNs[2]. To train NNs, HILO-MPC interfaces with PyTorch or TensorFlow, depending on the user's preferences. Nevertheless, the definition of the network, the training, and deployment (e.g., for dynamic optimization) are carried on using HILO-MPC syntax. Under the hood, HILO-MPC transforms the problem into an equivalent PyTorch or TensorFlow problem, solves it, and then converts it into a CasADi-compatible function. Hence, HILO-MPC inherits all the optimization algorithms (e.g., Adam stochastic gradient descent) and loss functions available in PyTorch or TensorFlow.

In the following example, we show how to define an ANN consisting of two layers with 10 neurons each, with a $80\%/20\%$ train/test split, using a sigmoid function as activation function, a batch size of 64, with maximum 1000 epochs, with early stopping with patience of 100 epochs (cf. Section 6.1)

```python
from hilo_mpc import ANN, Dense

# Initialize NN
ann = ANN(features, labels)
ann.add_layers(Dense(10, activation='sigmoid'))
ann.add_layers(Dense(10, activation='sigmoid'))

# Add training data set
ann.add_data_set(pdf)

# Set up NN
```

[2]Different Bayesian NNs models and recurrent NNs are currently being tested and will be available in the next releases

```
ann.setup()

# Train the NN
batch_size = 64
epochs = 1000
ann.train(batch_size, epochs, test_split=.2, patience=100,
         scale_data=True)
```

where `df` is a Pandas dataframe containing the dataset, `Dense(...)` is the command for creating a fully connected layer. Once the model is trained, it can be, for example, easily used to define a hybrid model. Hybrid models with series structure (cf. Fig. 7 left) can be defined with the command `model.substitute_from(ann)` where `model` is the previously defined model containing, e.g., first-principle equations. HILO-MPC will try to find the names of the labels in the variables of the model and substitute the entire network into that label.Note that the names of the features and labels are defined as column names in the data frame `df`. To create a parallel hybrid model (cf. Fig. 7 right), the network can be simply added or subtracted to a first-principle model as follows `hybrid_model = model + ann`. Once the hybrid model is generated, it can be used where necessary in the simulation.

7.4.2. Gaussian Processes

Training a GP in HILO-MPC is similar to an NN, but the setup is slightly different since the kernels have to be defined. The kernels currently available are Rational quadratic, exponential sine, gamma exponential, polynomial, and squared exponential. HILO-MPC uses scikit-learn and an in-house Gaussian process library. The user can select which library to use. The in-house GP trainer can be done using nonlinear constrained optimization software, e.g., using IPOPT. The kernel class can take the prior value of the hyperparameters and the bounds where these are allowed to lie when the corresponding object is initialized. Here is a simple example using a GP with squared exponential kernel trained on some predefined `features` and `labels`

```
from hilo_mpc import GPR, SquaredExponential

# Initialize kernel
kernel = SquaredExponential(variance=0.002,
                            bounds={'length_scales': (0.0001, 10),
                            'variance': (0.0001, 10)})

# Initialize GP
gpr = GPR(features, labels, prior_mean=0, kernel=kernel)
```

```
# Add training data set
gpr.set_training_data(train_in, train_out)

# Set up GP
gpr.setup()

# Fit the GP
gpr.fit_model()
```

If needed, as for the NN, the trained GP can be integrated into a hybrid model in a parallel or series structure.

7.5. Control module

The control module contains a nonlinear MPC, a linear MPC for discrete systems, and a proportional-integral-derivative (PID) controller. Here, we describe only the nonlinear MPC for brevity.

7.5.1. Model Predictive Control

We consider a sampled-data nonlinear MPC problem with a continuous time-varying DAE system. A similar formulation can be obtained for a discrete-time nonlinear system. At the sampling time t_k, the optimal control problem to solve reads

$$
\begin{aligned}
\min_{u(\cdot)} \quad & \int_{t_k}^{t_k+T} l(t, x(t), z(t), u(t), p)\mathrm{d}t + e(t_k + T, x(t_k + T), z(t_k + T), p), && \text{(7.2a)}\\
\text{s.t.} \quad & \dot{x}(t) = f(t, x(t), z(t), u(t), p), \quad x(t_k) = \hat{x}(t_k), && \text{(7.2b)}\\
& 0 = q(t, x(t), z(t), u(t), p), && \text{(7.2c)}\\
& g(t, x(t), z(t), u(t), p) \leq 0, && \text{(7.2d)}\\
& g_T(t_k + T, x(t_k + T), z(t_k + T), p) \leq 0, && \text{(7.2e)}\\
& y(t) = h(t, x(t), z(t), u(t), p) && \text{(7.2f)}\\
& u(t) = u(t_k + T_c), \ \text{ for } \ t \geq t_k + T_c && \text{(7.2g)}\\
& \text{for} \quad t \in [t_k, t_k + T] \subset \mathbb{R}. && \text{(7.2h)}
\end{aligned}
$$

Where $u(\cdot)$ is the optimal input function, $l : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}$ is the stage cost, $e : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \rightarrow \mathbb{R}$ the terminal cost and T is the prediction horizon and T_c is the control horizon. $\hat{x}(t_0)$ is the measured or estimated state vector. The equation

$g : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_g}$ represents the nonlinear path constraints and $g_T : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{g,\mathrm{T}}}$ the nonlinear terminal constraints.

Problem (83) cannot be solved directly since $u(\cdot)$ and the constraints are infinite-dimensional. HILO-MPC uses *direct approaches* to transform the problem to an equivalent discrete finite-dimensional optimization problem. These approaches parametrize the input with a finite number of parameters, for example, using piece-wise constant inputs, and force the nonlinear constraints only on a finite number of points. This reads as follows

$$\begin{aligned}
\min_{\mathbf{u}} \quad & \sum_{i=k}^{N} \int_{t_i}^{t_i+\Delta t} l(t, x(t), z(t), u_i, p)dt + e(t_k + T, x(t_k + T), z(t_k + T), p), && \text{(7.3a)}\\
\text{s.t.} \quad & x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_i+\Delta t} f(t, x(t), z(t), u_i, p)dt, && \text{(7.3b)}\\
& x(t_k) = \hat{x}(t_k), && \text{(7.3c)}\\
& 0 = q(t_i, x(t_i), z(t_i), u_i, p), && \text{(7.3d)}\\
& g(t_i, x(t_i), z(t_i), u_i, p) \leq 0, && \text{(7.3e)}\\
& g_T(t_k + T, x(t_k + T), z(t_k + T), p) \leq 0, && \text{(7.3f)}\\
& y(t_i) = h(t_i, x(t_i), z(t_i), u_i, p) && \text{(7.3g)}\\
& u_i = u_{N_c}, \text{ for } i \geq N_c && \text{(7.3h)}\\
& \text{for} \quad i \in [k, ..., N] \subset \mathbb{N}_0. && \text{(7.3i)}
\end{aligned}$$

where Δt is the sampling time, $N_c = \text{ceil}(T_c/\Delta t)$ is the control horizon and $\mathbf{u} = [u_k, ..., u_{N_c}]^\top$ is a sequence of piece-wise constant inputs, i.e. $u(t) = u_i, \ \forall t \in [t_i, t_i + \Delta t] \subset \mathbb{R}$. To solve (84), HILO-MPC implements a multiple shooting approach [25]. The integration of the system model in between the shooting points can be done with Runge-Kutta methods of various orders, orthogonal collocation [137] or the dynamics can be integrated with CVODES or IDAS solvers [78]. The default method is orthogonal collocation, and the user can select any other method if needed. As default, piece-wise constant input is assumed at every control interval, but other parametrizations are also possible by defining an appropriate control signal in the system model. Notice that the actual formulation of Problem (83) depends on which method is used for the discretization. Here, we do not go into the details of the formulation of the approximated problem. The reader is invited to refer to the previously cited papers for more details on the formulation of the finite-dimensional optimal control problem for single cases.

Remark Any of the equations in (83) could be at least partially learned from data. For example in [121] the output function (83f) is learned, in [163, 164, 33] the terminal constraints (83e) and in [34, 10] the path constraints. Also, the objective function can be learned, see, e.g., [184, 17, 31]. HILO-MPC allows the use of machine learning models in any of these components.

Remark HILO-MPC can also optimize the sampling intervals. This will add to Problem (84) the vector $[\Delta t_k, ..., \Delta t_N]$ as an optimization variable. This allows the solution of minimum time problems and optimal sampling time problems.

The stage and arrival cost depends on the goal of the MPC. We will go through the different MPC problems HILO-MPC can solve in the next sections.

MPC for set-point tracking

In set-point tracking (or reference tracking) MPC, some outputs, states, or inputs need to track given fixed references. If a quadratic cost is used, the stage and terminal cost can be defined as follows

$$\begin{aligned} l(x(t),u(t),y(t)) &= \|x(t)-x_r\|_Q^2 + \|u(t)-u_r\|_R^2 + \|y(t)-y_r\|_P^2, \\ e(x(t_k+T),y(t_k+T)) &= \|x(t_k+T)-x_r\|_{Q_\mathrm{T}}^2 + \|y(t_k+T)-y_r\|_{P_\mathrm{T}}^2 \end{aligned}$$

where x_r, u_r, y_r are fixed references, and $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$ $P \in \mathbb{R}^{n_y \times n_y}$ $P_\mathrm{T} \in \mathbb{R}^{n_y \times n_y}$ and $Q_\mathrm{T} \in \mathbb{R}^{n_x \times n_x}$ are weighting matrices. For example, once we have defined the bike model like in section 33. A possible set-point tracking MPC that tracks a reference speed of $v_\mathrm{r} = 2\ m/s$, with a prediction horizon of 20 steps, can be defined as follows

```python
from hilo_mpc import NMPC

nmpc = NMPC(model)
nmpc.horizon = 20
nmpc.quad_stage_cost.add_states(names=['v'], ref=2, weights=10)
nmpc.quad_term_cost.add_states(names=['v'], ref=2, weights=10)
nmpc.setup()
```

Note that in the keyworded argument `names` takes the names of the variable as they are defined in the model. Generic objective functions can be defined as follows. Let us assume we want to minimize an economic objective that tries to limit the actuation level of the two inputs while forcing the bike to go to the reference point [1,1]

$$l(u(t)) = \delta a^2 + (p_\mathrm{x} - 1)^2 + (p_\mathrm{y} - 1)^2$$

this can be set using the method `stage_cost.cost`

```
nmpc.stage_cost.cost = model.u[1]*model.u[0]**2
nmpc.stage_cost.cost =(model.x[0]- 1)**2 + (model.x[1]- 1)**2
```

Note that, in this case, the generic function takes the variables of the model directly. Note also that the method can be called multiple times: Every time it is called, the new term is added to the stage or terminal cost. Quadratic cost and generic cost can be used in the same MPC definition.

MPC for trajectory-tracking

For trajectory tracking problems, the variables need to track a time-varying reference. For quadratic cost functions, this looks as follows:

$$l(x(t), u(t), y(t)) = \|x(t) - x_r(t)\|_Q^2 + \|u(t) - u_r(t)\|_R^2 + \|y(t) - y_r(t_i)\|_P^2, \quad (7.4)$$

$$e(x(t_0 + T), y(t_0 + T)) = \|x(t_0 + T) - x_r(t_0 + T)\|_{Q_\mathrm{T}}^2 + \|y(t_0 + T) - y_r(t_0 + T)\|_{P_\mathrm{T}}^2, \quad (7.5)$$

where $x_r : \mathbb{R} \rightarrow \mathbb{R}^{n_x}, u_r : \mathbb{R} \rightarrow \mathbb{R}^{n_u}, y_r : \mathbb{R} \rightarrow \mathbb{R}^{n_y}$ are time-varying references. For example, if we want the bike to move in an elliptic trajectory, described by:

$$p_{\mathrm{x},tr}(t) = 30 - 14\cos(t) \quad (7.6)$$

$$p_{\mathrm{y},tr}(t) = 30 - 14\sin(t) \quad (7.7)$$

the MPC problem can be set as follows

```
from hilo_mpc import NMPC
import casadi as ca

nmpc = NMPC(model)
nmpc.horizon = 20

t = nmpc.create_time_variable()
traj_x = 30 - 14 * ca.cos(t)
traj_y = 30 - 16 * ca.sin(t)

nmpc.quad_stage_cost.add_states(names=['px', 'py'],
    ref=[traj_x, traj_y],
    weights=[10, 10], trajectory_tracking=True)
nmpc.quad_term_cost.add_states(names=['px', 'py'],
    ref=[traj_x, traj_y],
    weights=[10, 10], trajectory_tracking=True)

nmpc.setup()
```

Note that instead of fixed references, here we passed two time-varying functions `traj_x` and `traj_y`. These are defined using CasADi functions. If only their values are available instead of the trajectory functions, these can also be passed as a dictionary.

MPC for path-following

While in trajectory tracking MPC, both the value of the reference and time are fixed simultaneously, in path-following MPC, the controller has the freedom of choosing *when* to be on the path [119, 55]. In this case, the model is augmented with a *virtual path state* as follows:

$$\begin{aligned}
\min_{u(\cdot),u_{\text{pf}}(\cdot)} \quad & \int_{t_k}^{t_k+T} l(t,x(t),z(t),u(t),p)\text{d}t + e(t_k+T,x(t_k+T),z(t_k+T),p) && \text{(7.8a)}\\
\text{s.t.} \quad & \dot{x}(t) = f(t,x(t),z(t),u(t),p), \quad x(t_k)=\hat{x}(t_k), && \text{(7.8b)}\\
& 0 = q(t,x(t),z(t),u(t),p), && \text{(7.8c)}\\
& \dot{\theta} = u_{\text{pf}}, \quad \theta(t_k)=0, && \text{(7.8d)}\\
& g(t,x(t),z(t),u(t),p,\epsilon) \leq 0, && \text{(7.8e)}\\
& g_T(t_k+T,x(t_k+T),z(t_k+T),p) \leq 0 && \text{(7.8f)}\\
& y(t) = h(t,x(t),z(t),u(t),p) && \text{(7.8g)}\\
& \text{for} \quad t \in [t_k,t_k+T] \subset \mathbb{R}, && \text{(7.8h)}
\end{aligned}$$

where $\theta \in \mathbb{R}^{n_\theta}$ is a virtual path state vector and $u_{\text{pf}}(\cdot)$ is the virtual input that can controller can choose. Hence, the objective function looks like

$$\begin{aligned}
l(x(t_i),u(t_i),y(t_i)) =& \|x(t)-x_r(\theta(t))\|_Q^2 + \|u(t)-u_r(\theta(t))\|_R^2\\
&+ \|y(t_i)-y_r(\theta(t))\|_P^2\\
e(x(t_k+T),y(t_k+T)) =& \|x(t_k+T)-x_r(\theta(t_k+T))\|_{Q_{\text{T}}}^2 +\\
&+ \|y(t_k+T)-y_r(\theta(t_k+T))\|_{P_{\text{T}}}^2.
\end{aligned}$$

Usually, to force the controller to move into only one direction along the path usually a lower bound on u_{pf} is added, i.e. $u_{\text{pf}} \geq u_{\text{pf,min}}$ with $u_{\text{pf,min}} \in \mathbb{R}_+^{n_{u,\theta}}$.

HILO-MPC also allows to track a constant u_{pf}, so that the MPC tries to maintain a constant *speed* of the virtual state:

$$\begin{aligned}
l(x(t), u(t), y(t)) = & \|x(t) - x_r(\theta(t))\|_Q^2 + \|u(t) - u_r(\theta(t))\|_R^2 + \\
& + \|y(t) - y_r(\theta)\|_P^2 + \|u_{\text{pf}}(t) - u_{\text{pf,ref}}\|_{R_{\text{pf}}}^2 \\
e(x(t_k + T), y(t_k + T)) = & \|x(t_k + T) - x_r(\theta(t_k + T))\|_Q^2 + \\
& + \|y(t_k + T) - y_r(\theta(t_k + T))\|_P^2.
\end{aligned}$$

Contrary to the other available toolboxes, path-following MPC problems are automatically generated. The user needs just to activate the path-following mode for the desired variables. For the same trajectory of the previous section, the problem reads

```python
from hilo_mpc import NMPC
import casadi as ca

nmpc = NMPC(model)
nmpc.horizon = 20
# Create path variable to use in the definition path definition
theta = nmpc.create_path_variable()
traj_x = 30 - 14 * ca.cos(theta)
traj_y = 30 - 16 * ca.sin(theta)

nmpc.quad_stage_cost.add_states(names=['px', 'py'],
       ref=[traj_x, traj_y],
       weights=[10, 10],
       path_following=True)
nmpc.quad_term_cost.add_states(names=['px', 'py'],
       ref=[traj_x, traj_y],
       weights=[10, 10],
       path_following=True)

nmpc.setup()
```

Note that in this case, we used the path variable and not the time variable.

Remark HILO-MPC allows mixing the previous problems with minimum effort, e.g., the user can track some constant references for some of the variables while tracking a path with other variables.

Constraints

HILO-MPC implements box constraints and nonlinear path and terminal constraints constraints. The box constraints can be added as follows

```
nmpc.set_box_constraints(x_ub=[5,5,2,1], x_lb=[-5,-5,-2,-1],
u_lb = [-1,-1], u_ub=[1,1])
```

these will limit $p_\mathrm{x} \in [-5,5]$, $p_\mathrm{y} \in [-5,5]$, $v \in [-2,2]$, $\phi \in [-1,1]$, $a \in [-1,1]$ and $\delta \in [-1,1]$. Suppose we now want to limit the position of the bike in a circle with a radius of 2 m centered in the origin. In this case, the following nonlinear constraint can be added

```
nmpc.stage_constraint.constraint = (model.x[0]**2 +
      model.x[1]**2)**(0.5)
nmpc.stage_constraint.ub = 4
nmpc.stage_constraint.lb = 0
```

The same follows for the terminal constraint. Soft constraints can be also easily activated with the command `nmpc.stage_constraint.is_soft = True`. When soft constraints are selected, HILO-MPC automatically adds the slack variables $\epsilon_\mathrm{p} \in \mathbb{R}^{n_g}$ and $\epsilon_\mathrm{T} \in \mathbb{R}^{n_{g,\mathrm{T}}}$ to the path and terminal constraints respectively as follows

$$
\begin{aligned}
\min_{u(\cdot),\epsilon_\mathrm{T},\epsilon_\mathrm{p}} \quad & \int_{t_k}^{t_k+T} l(\cdot)\mathrm{d}t + e(\cdot) + \|\epsilon_\mathrm{s}\|^2_{E_\mathrm{p}} + \|\epsilon_\mathrm{T}\|^2_{E_\mathrm{T}} && \text{(7.9a)}\\
\text{s.t.} \quad & \dot{x}(t) = f(t, x(t), z(t), u(t), p), \quad x(t_k) = \hat{x}(t_k), && \text{(7.9b)}\\
& 0 = q(t, x(t), z(t), u(t), p) && \text{(7.9c)}\\
& g(t, x(t), z(t), u(t), p) \leq \epsilon_\mathrm{p}, && \text{(7.9d)}\\
& g_T(t_k+T, x(t_k+T), z(t_k+T), p) \leq \epsilon_\mathrm{T} && \text{(7.9e)}\\
& y(t) = h(t, x(t), z(t), u(t), p) && \text{(7.9f)}\\
& 0 \leq \epsilon_\mathrm{p} \leq \epsilon_\mathrm{p,max}, \quad 0 \leq \epsilon_\mathrm{T} \leq \epsilon_\mathrm{T,\,max} && \text{(7.9g)}\\
& \text{for} \quad t \in [t_k, t_k+T] \subset \mathbb{R}, && \text{(7.9h)}
\end{aligned}
$$

If desired, the user can choose the weighting matrix $E_\mathrm{p} \in \mathbb{R}^{n_g \times n_g}$ and $E_\mathrm{T} \in \mathbb{R}^{n_{g,\mathrm{T}} \times n_{g,\mathrm{T}}}$ that limit the increase of the slack variables, and also $\epsilon_\mathrm{p,max}$ and $\epsilon_\mathrm{T,max}$, which are the maximum constraint violations of path and terminal constraints, respectively.

7.6. Observer module

The observer model contains the Kalman filters, the moving horizon estimator, and the particle filter. As for the MPC, these can also use machine learning models in their design.

7.6.1. Moving Horizon Estimation

The moving horizon estimator (MHE) is an observer based on the solution of an optimization problem similar to the MPC problem [160]. In a nutshell, at time t_k it considers a window of $\bar{N}$ past measurements $[\hat{y}(t_{k-\bar{N}}), ..., \hat{y}(t_k)]$ and tries to fit the predicted system output to these measurements, by optimizing the model parameters, the state estimate at time $t_{k-\bar{N}}$ and the state noise. The output is the estimated state and parameters at time t_k. The problem is repeated at every sampling time, similar to the receding horizon MPC. We consider a sample-data problem with equidistant sampling times as it is more common in practice. At these sampling times, measurements are taken. Since measurements are discontinuous, the objective function is usually in discrete form. For simplicity we indicate with $(\cdot)_{i|k}$ the variable at time t_i estimated at time t_k. The MHE problem solved at time t_k reads

$$\min_{x_{k-N|k}, p_k, w_{(\cdot)|k}, z_{(\cdot)|k}} \quad \left\| \begin{bmatrix} x_{k-\bar{N}|k} - \hat{x}_{k-\bar{N}|k} \\ p_k - \hat{p}_k \end{bmatrix} \right\|^2_{P_k} + \sum_{i=k-\bar{N}}^{k} \|\hat{y}_i - y_{i|k}\|^2_R + \|w_{i|k}\|^2_W \tag{7.9a}$$

$$\text{s.t.} \quad x_{i+1|k} = x_{i|k} + \int_{t_i}^{t_i+\Delta t} f(t, x(t), z(t), \hat{u}(t), p_k) + w(t), \tag{7.9b}$$

$$y_{i|k} = h(t_i, x_{i|k}, z_{i|k}, \hat{u}_i, p_k) + v_{i|k} \tag{7.9c}$$

$$g(t_i, x_{i|k}, u_i) \leq 0, \tag{7.9d}$$

$$\text{for } i \in [k-\bar{N}, k], \quad k, \bar{N} \in \mathbb{N}, \tag{7.9e}$$

where $\bar{N}$ is the horizon length, $\hat{y}_i$ and $\hat{u}_i$ are the output and input measurements at time t_i respectively. The first term of the objective function is called *arrival cost*, while the second and third weight the measurements and state noise respectively [160, 5]. $R \in \mathbb{R}^{n_y \times n_y}$, $W \in \mathbb{R}^{n_x \times n_x}$ and $P_k \in \mathbb{R}^{(n_x+n_p)\times(n_x+n_p)}$ are the weighting matrix for the outputs, state noise and arrival cost. Note that P_k is often updated at every sampling time, using, for example, an extended Kalman filter [158]. The optimization variables are the state $x_{k-N|k}$, i.e., the state at the beginning of the horizon, the state noise $w_{(\cdot)|k} = \{w_{i|k}, \ \forall i \in [k-\bar{N}, k]\}$, the algebraic states (for DAE systems) $z_{(\cdot)|k} = \{z_{i|k}, \ \forall i \in [k-\bar{N}, k]\}$, and the system parameters p_k. Note that the parameters are considered constant in the horizon but can be every time the optimization is run to adapt to the new measurements.

As for the MPC, also in this case we use direct approaches to solve (9). For the bike model, an MHE model with $R = W = P_k = 10I$ (where I is the identity matrix of appropriate dimensions) can be easily defined as follows:

```python
from hilo_mpc import MHE

mhe = MHE(model)
mhe.horizon = 20
mhe.quad_arrival_cost.add_states(weights=[10,10,10,10], guess=x0_est)
mhe.quad_stage_cost.add_measurements(weights=[10,10,10,10])
mhe.quad_stage_cost.add_state_noise(weights=[10,10,10,10])
mhe.setup()
```

7.6.2. Kalman filters

The Kalman filter (KF) is an algorithm that allows for the estimation of observable states via available measurement data. In general, it consists of two steps. In the prediction step, the estimated states from the previous iteration are propagated through the model dynamics to obtain preliminary values for the states at the current time step and the so-called a priori estimates. In the update step, the a priori estimates are updated using the measurement data to obtain the a posteriori estimates. The original formulation of the KF was developed for linear discrete-time systems [89]

$$\begin{aligned} x_k &= A_k x_{k-1} + B_k u_k + w_k, \\ y_k &= C_k x_k + v_k, \end{aligned}$$

where w_k is the process noise and v_k is the measurement noise. The process noise $w_k \sim \mathcal{N}(0, Q_k)$ and the measurement noise $v_k \sim \mathcal{N}(0, R_k)$ are assumed to be zero-mean normal distributions with the covariance matrices Q_k and R_k, respectively. Accordingly, the prediction step and update step of the KF are as follows

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1} + B_k u_k, \tag{7.10}$$

$$P_{k|k-1} = A_k P_{k-1} A_k^\mathsf{T} + Q_k, \tag{7.11}$$

$$\hat{x}_k = \hat{x}_{k|k-1} + K(y_k - \hat{y}_k), \tag{7.12}$$

$$P_k = P_{k|k-1} - K P_{y_k y_k} K^\mathsf{T}, \tag{7.13}$$

with

$$\begin{aligned} K &= P_{x_k y_k} P_{y_k y_k}^{-1}, \\ \hat{y}_k &= C_k \hat{x}_{k|k-1}, \\ P_{x_k y_k} &= P_{k|k-1} C_k^\mathsf{T}, \\ P_{y_k y_k} &= C_k P_{k|k-1} C_k^\mathsf{T} + R_k, \end{aligned}$$

where P_k is the error covariance matrix, $P_{x_k y_k}$ is the state-measurement cross-covariance matrix, $P_{y_k y_k}$ is the measurement covariance matrix, K is called the Kalman gain and $\hat{y}_k$

is the predicted output. The notation $\hat{x}_{k|k-1}$ represents the estimated state $\hat{x}$ at time step k given observations up to and including time step $k-1$. The covariance matrix P_k is a measure of the estimation accuracy of the estimated states, i.e., the lower the value, the closer the estimated state is to the actual one assuming normally distributed noise.

There are also extensions to the original KF formulation for dealing with nonlinear models. The two most known extensions are the extended Kalman filter (EKF) [181] and the unscented Kalman filter (UKF) [198]. In the EKF, while a priori estimates of the states can be easily obtained by propagating through the nonlinear model dynamics, the propagation of the error covariance matrix is approximated by a system linearization. One drawback is that the linearization can produce highly unstable Kalman filters [198]. The UKF uses the unscented transform to generate a set of samples, the so-called sigma points, that represent the initial probability distribution. These sigma points are then propagated through the nonlinear system dynamics and afterward used to approximate the a priori estimates and the predicted output and the various covariance matrices. The update step is then the same as for the KF and EKF.

Here, for brevity, we show only an example of a UKF. The other Kalman filters follow a similar syntax. Assuming that we can measure all the states of the bike, we have

```python
from hilo_mpc import UKF

ukf = UKF(model)
ukf.setup()

# Set R matrix
ukf.R = [1e-4,1e-4,1e-4,1e-4]

# Set initial guess state
ukf.set_initial_guess([0., 0., 0., 0.])

# Run simulations
for k in range(200):
    model.simulate(u=u)

    # Get noisy measurement
    yk = model.solution.make_some_noise('yf', var={'y': 1e-4})

    # Calculate estimates
    ukf.estimate(y=yk, u=u)
```

Note that we used the method `make_some_noise` to generate noisy measurements. The noise is assumed to be Gaussian-distributed. In this case, the variance was set to $1e^{-4}$ for all outputs.

7.6.3. Particle filter

The particle filter [173] is another algorithm that can be used for estimation. It works similarly to the UKF in that a set of samples, the "particles", is propagated through the nonlinear system dynamics. This set of samples is drawn from a probability density function that is assumed to be known. Next, the relative likelihood of each propagated particle is determined

$$q_i^* = \frac{q_i}{\sum_{i=1}^{n_p} q_i},$$

with

$$q_i \sim \mathrm{pdf}(y_k|\hat{y}_k),$$

where n_p is the number of particles in the set. This relative likelihood measures how well a particle matches the measurement y_k and is used in the resampling step to generate the set of a posteriori particles. This resampling step is analogous to the update step of the Kalman filters. The resampling can be achieved by using one of the several available sampling strategies, like e.g., survival of the fittest [90] or regularization [132]. Similar to the UKF, the a posteriori particles can be used to determine the estimated states via computation of the mean of the distribution. For highly nonlinear system dynamics or non-Gaussian probability density functions, the particle filter usually outperforms the Kalman filter algorithms [173]. The setup and usage of a particle filter in HILO-MPC is similar to that of the Kalman filters; hence, we do not show it here for brevity.

7.7. Embedded Module

Implementing optimization-based controllers/MPC controllers on embedded systems requires a tailored implementation. By now, a series of tailored automatic code generation tools for MPC exist[83, 192, 216, 50].

HILO-MPC provides an interface for the code generation of MPC for linear time-invariant discrete-time model predictive control, tailored for small embedded systems, like micro-controllers[216].

The code generation is limited to discrete-time linear time-invariant systems of the form

$$x^+ = Ax + Bu \tag{7.14}$$

subject to input and state constraints $\underline{u} \leq u \leq \overline{u}$, $\underline{x} \leq x \leq \overline{x}$, where x and u denote the state and input at the current sampling time, and x^+ denotes the state at the next sampling time.

The considered linear MPC problem formulation is

$$\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \frac{1}{2}\sum_{j=0}^{N-1}(\|x_j\|_Q^2 + \|u_j\|_R^2) + \frac{1}{2}\|x_N\|_P^2, \\
\text{subject to} \quad & x_{j+1} = Ax_j + Bu_j, \qquad j = 0, \dots, N-1, \\
& \underline{u} \leq u_j \leq \overline{u}, \qquad j = 0, \dots, N-1, \\
& \underline{x} \leq x_j \leq \overline{x}, \qquad j = 0, \dots, N-1, \\
& x_0 = x,
\end{aligned} \tag{7.15}$$

where the integer $N \geq 2$ is the prediction horizon, the matrices $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$ and $P \in \mathbb{R}^{n_x \times n_x}$ are the state, input, and terminal weighting matrix, respectively. The input sequence $u = \begin{bmatrix} u_0 & \dots & u_{N-1} \end{bmatrix}$ is the optimization variable.

For automatic code generation of the problem (15) tailored towards microcontrollers, we rely on *μAO-MPC*,[216] a code generation tool that uses an augmented Lagrangian and Nesterov's fast gradient method [215, 98]. By design, the algorithm's implementation is simple and relies only on matrix-vector operations (i.e., no divisions), it can easily be warm started and computes good MPC inputs in very few iterations of the algorithm. For well-behaved problems, the computations can be made more efficient by using fixed-point arithmetic (instead of floating-point).

The code generation reformulates the MPC optimization problem (15) as a condensed parametric QP $\mathcal{P}(x)$

$$\begin{aligned}
\underset{u \,\in\, \mathcal{U}}{\text{minimize}} \quad & \frac{1}{2}u^H u + u^g(x), \\
\text{subject to} \quad & \underline{z}(x) \leq Eu \leq \overline{z}(x),
\end{aligned} \tag{7.16}$$

where the constant Hessian matrix $H^\mathsf{T} = H > 0 \in \mathbb{R}^{Nn_u \times Nn_u}$ and the gradient vector $g(x)$ depend on the system (14) and the objective function in (15). Additionally, $g(x)$ depends on the current state x (i.e., the parameter). The set $\mathcal{U} = \{u \mid \underline{u} \leq u_j \leq \overline{u},\ j = 0, \dots, N-1\}$ defines the input constraints. The constant matrix E and the constraint vectors $\underline{z}(x)$ and $\overline{z}(x)$ depend on the constraints in (15).

The generated code is a highly portable C-code. It is split into the data for the QP $\mathcal{P}(x)$ (e.g., the constant matrix H and the parametric vector $g(x)$) and the implementation of the optimization algorithm to solve the QP [215, 98].

Figure 39 depicts an overview of the embedded module. Similarly to other modules, the `LMPC` class is initialized using a `Model` object. In this case, a linear time-invariant discrete-time model is used. Calling the `LMPC.setup()` method, specifying the solver `'muaompc'`, will trigger the code generation procedure. The code generation automatically

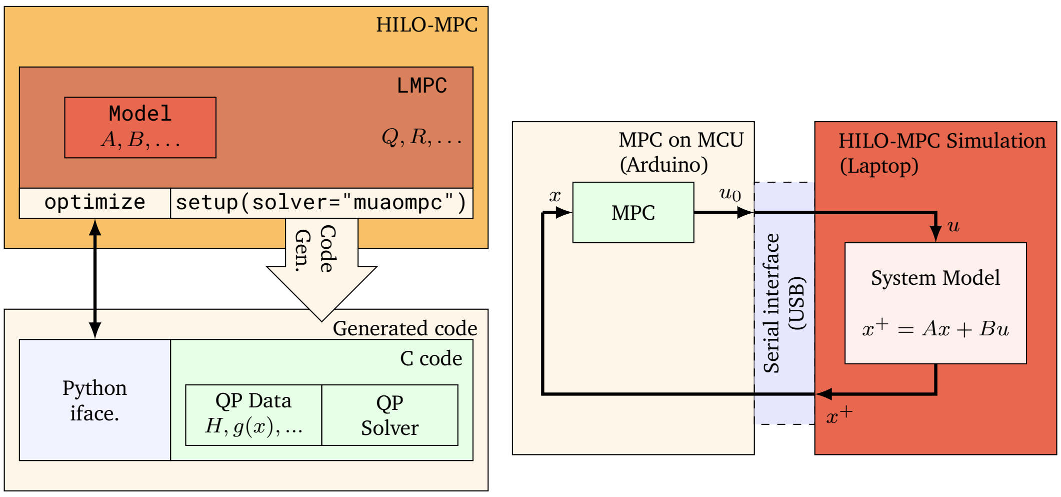


Figure 7.2.: A scheme of the Embedded module. Left, the typical use of the module, including automatic code generation provided by *μAO-MPC*. Right, a hardware-in-the-loop simulation is used as an example.

creates C-code, which can be compiled to run on an embedded target. Additionally, a Python interface to the generated C-code is also created. This Python interface is used by the `LMPC.optimize()` method to obtain the input that minimizes 15. The basic use of the Embedded Module is exemplified in the following Python code

```python
from hilo_mpc import LMPC, Model
# The Model class uses the linear system matrices A, B, and
# the LMPC class uses the weighting matrices Q, R, P, constraints, etc.,
# to create the mpc object. Using the solver muaompc triggers the code
   generation
mpc.setup(nlp_solver='muaompc')
# the generated MPC code can be used directly on an embedded target using C,
# or via the Python interface as HILO-MPC does on the next line
u = mpc.optimize(x0=x)
```

7.8. Examples

In the next section, we present some examples that have been solved using HILO-MPC. These examples can also be found in the example repository and Python files or Jupyter notebook files. For this reason, we do not go into the implementation details but give just a high-level description of the problems and present the results. The reader is invited to read the documentation and the example codes for details.

7.8.1. Learning the dynamics - Race car

We consider a realistic example of an autonomous racing minicar. The goal is to follow a complex track using a path-following formulation in the presence of disturbances. We use a bike model with nonlinear tire models and a drive train model (cf. Fig. 40a) that has been identified and validated in experiments by [110]. To this model, we add a component describing the effect of lateral drag forces due to, for example, a strong wind. The model is represented by the following system of nonlinear differential equations

$$\dot{p}_x = v_x \cos(\psi) - v_y \sin(\psi), \tag{7.17}$$
$$\dot{p}_y = v_x \sin(\psi) + v_y \cos(\psi), \tag{7.18}$$
$$\dot{\psi} = w, \tag{7.19}$$
$$\dot{v}_x = \frac{1}{m}\left(F_{r,x} - F_{a,x} - (F_{f,y} - F_{a,y})\sin(\delta) + m v_y \omega\right), \tag{7.20}$$
$$\dot{v}_y = \frac{1}{m}\left(F_{r,y} - F_{a,y} - (F_{f,y} - F_{a,y})\cos(\delta) - m v_x \omega\right), \tag{7.21}$$
$$\dot{\omega} = \frac{1}{I_z}\left(F_{f,y} l_f \cos(\delta) - F_{r,y} l_r\right), \tag{7.22}$$

where p_x and p_y are the coordinates of the center of gravity, v_x and v_y are longitudinal and later velocities of the center of gravity. The orientation is denoted by ψ and the yaw rate by ω. The control inputs are the motor duty cycle d and steering angle δ. The two parameters l_f and l_r are the distances from the center of gravity to the front axle and rear axle, respectively, m is the mass of the car, and I_z is the inertia. The path to follow is the center of a racing track which has been interpolated using splines. The tire forces are modeled with a simplified Pacejka Tire Model [13]

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_f)), \tag{7.23a}$$
$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_r)), \tag{7.23b}$$
$$F_{r,x} = (C_{m1} - C_{m2} v_x) d - C_r - C_d v_x^2, \tag{7.23c}$$
$$\alpha_f = -\arctan\left(\frac{w l_f + v_y}{v_x}\right) + \delta, \tag{7.23d}$$
$$\alpha_r = \arctan\left(\frac{w l_r - v_y}{v_x}\right), \tag{7.23e}$$

where $D_f, D_r, B_f, B_r, C_{m1}, C_{m2}, C_r$ and C_d are parameters. The longitudinal and lateral drag forces are defined, respectively, as

$$\begin{aligned} F_{a,x} &= 0.5\ c_w \rho A v_x, \\ F_{a,y} &= 0.5\ c_w \rho A v_{y,\text{wind}}, \end{aligned}$$

where c_w is the drag coefficient, ρ is the air density, A the effective flow surface and $v_{y,wind}$ is the lateral wind velocity.

The model used by the MPC does not have the drag effect. The goal is to learn this effect from data using a Neural Network and then augment the first principle model with a machine learning component that models the drag effect. After discretization, the hybrid model can be written as:

$$x_{k+1} = f(x_k, u_k) + B^T m(x_k, u_k), \tag{7.24}$$

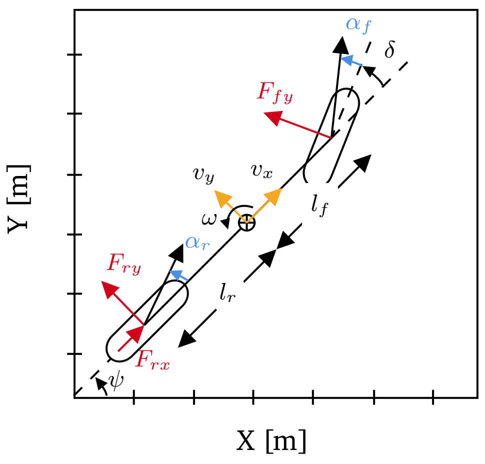
where $m(x_k, u_k)$ is an NN model and

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.25}$$

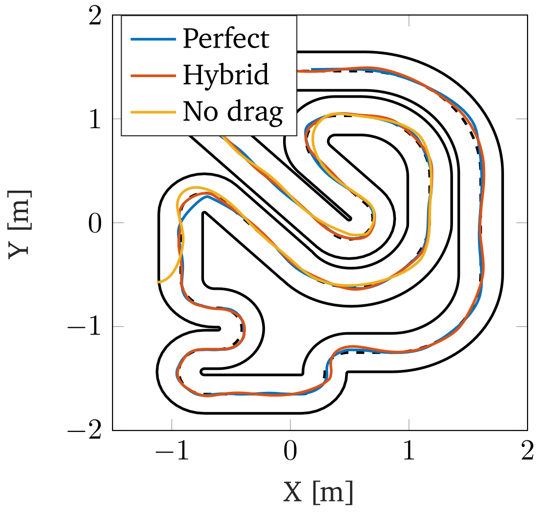
The features of the NN are the $v_{y,\text{wind}}$ and v_x, the labels are correction terms on ϕ, v_x, v_y and ω. To show the effectiveness of the learning, we compare the results of MPC using the perfect model (i.e., with known drag force effects), the hybrid model (using the NN), and the model without drag forces. Furthermore, the measurements of position, velocity, and directions are affected by Gaussian noise and estimated using an Unscented Kalman Filter. Figure 40b shows the results of the simulation. While the hybrid model has results similar to the perfectly-known model, the model without drag exits the race area after the fifth curve. The complete code can be found in the HILO-MPC repository.

7.8.2. Learning a reference - Cooperative robots

Inspired by [120], a follower robot has to track the position of a leader robot. The leader moves in a periodic but unknown trajectory. The objective is to learn the trajectory of the leader with GPs and pass the learned trajectory to the follower. Hence, in this case,



(a) Model used for the racing car.



(b) Results of the path-following problem

the machine learning model enters the reference function (cf. Fig. 41). The nonlinear dynamics of the robots are described by

$$\dot{x}_1 = u_1 \sin(x_3), \tag{7.26a}$$
$$\dot{x}_2 = u_1 \cos(x_3), \tag{7.26b}$$
$$\dot{x}_3 = u_2, \tag{7.26c}$$

where x_1 and x_2 are the horizontal and vertical position of the root and x_3 its heading angle, u_1 is the speed and u_2 the turning rate. The problem is a trajectory-tracking problem of the form (83) with an objective function (85) where $x_r(\cdot)$ is the mean function of a GP trained on the data set collected from the position of the leader. The trajectory generated by the leader results from applying the following time-varying forces

$$u_1(t) = 2 + 0.1 \sin(t), \tag{7.27a}$$
$$u_2(t) = 0.5 + \sin(t). \tag{7.27b}$$

Figures 42 show the results of the reference learning and closed-loop simulation.

7.8.3. Learning the controller - String damper system

Solving an MPC requires the solution of a (nonlinear) optimization problem online. This is possible only for applications where the available computational power and energy are

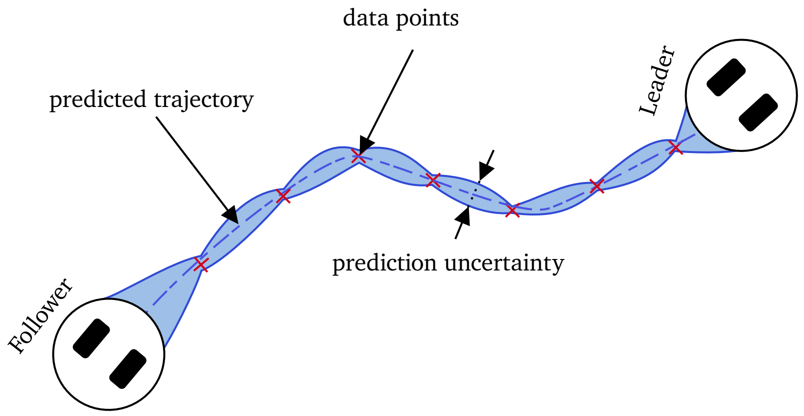


Figure 7.4.: Cooperative robot example. The follower needs to track the trajectory of the leader. The trajectory is learned from data using a GP regressor.

sufficient to guarantee a solution within the sampling times. For embedded applications with low computational power or that can use only a limited amount of energy (for example, battery-powered systems), this is often not possible. Hence, methods that provide at least a close-to-optimal solution without solving the optimization approach at every time step are necessary. This can be done using *explicit* MPC approaches. Some of these approaches are based on learning the solution of an MPC offline, i.e., the map $x \mapsto \rho_\theta(x)$ that approximates the implicit MPC control law, and then using the learned controller online [140, 91, 116, 37, 39, 44, 151], (cf. Fig. 43). In this way, the control action can be found with a simple and fast function evaluation. In this example, we want to control a mass-spring-damper system using a learned controller (Fig. 43. The model is

$$\dot{x}_1 = x_2, \tag{7.28a}$$

$$\dot{x}_2 = \frac{1}{m}(u - kx_1 - dx_2), \tag{7.28b}$$

where x_1 is the vertical position, x_2 the vertical velocity, u the vertical force and k, d the system parameters. The equilibrium point is $x = [0, 0]^\mathsf{T}, u = 0$. The objective is to maintain the reference $x_{\text{ref}} = [1, 0]^\mathsf{T}$ using a learned MPC. To do so, we use the results of just one closed-loop MPC simulation starting from the initial conditions $x(0) = [12, 0]^\mathsf{T}$. In total, 667 data points are collected. We use the data collected to train an NN with three fully connected layers, with ten neurons each. The features of the NN are x_1 and x_2; the labels are the input u. We test the learned controller starting from a different initial condition $x(0) = [10, 0]^\mathsf{T}$. In Fig. 44, the simulation results are shown. The learned controller is able

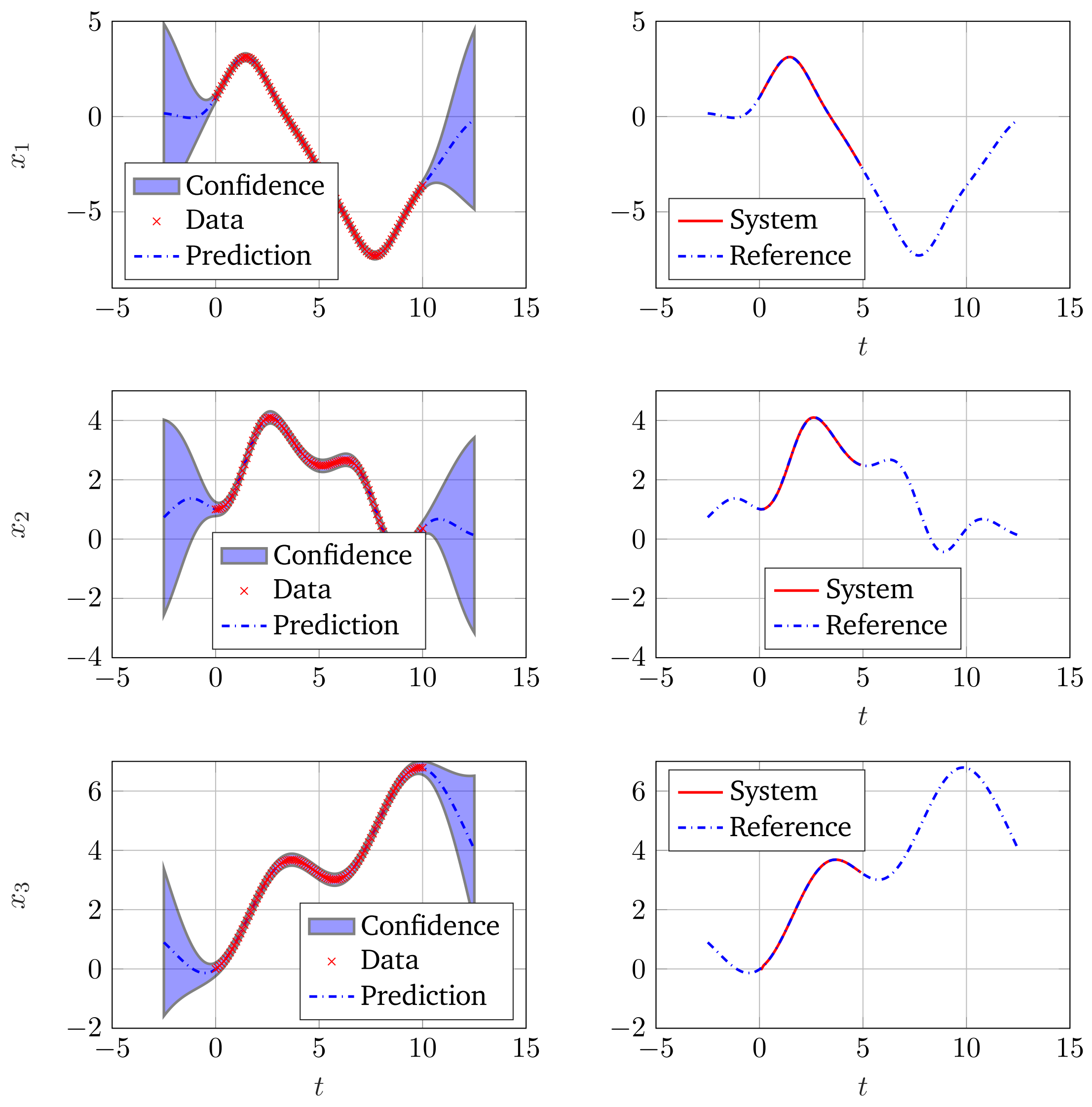


Figure 7.5.: Left column: learned GP. Right column: states of the follower robot following the learned references.

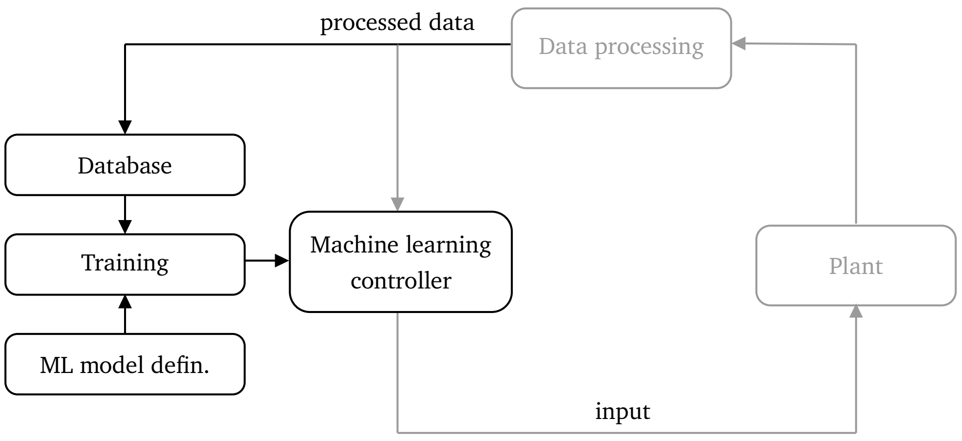


Figure 7.6.: Scheme of a machine learning controller.

to bring the system to the reference as the original controller, but while the MPC takes 14 ms per iteration, the evaluation of the NN took a time below the sensitivity of the profiler used to compute the elapsed time, that in this case is of 1 ms, so we obtained at least a 14 times reduction in computational time. The main disadvantage of a learned controller is that constraint satisfaction is not guaranteed (cf. [91]). Here, for brevity, we do not discuss this aspect.

7.9. Summary

We introduced HILO-MPC, a toolbox developed in the frame of this work, for the fast development of predictive control and estimation problems that facilitates the use of machine learning models, which can use state-of-the-art machine learning toolboxes such as PyTorch, TensorFlow and scikit-learn. The toolbox can solve a large variety of problems, such as model predictive control, moving horizon estimation, Kalman filters, and particle filters. We showed three applications of the toolbox. In the first application, a hybrid model of a minicar was used in an MPC for an autonomous driving example under the presence of model uncertainty due to external forces. In the second application, the machine learning model was instead used to learn the reference trajectory of a follower robot, whose task is to follow a leader robot. Finally, in the third example, machine learning was used to learn the implicit control law of an MPC offline. This is particularly interesting for applications where the computational power and/or energy is limited. Hence, the optimal control problem cannot be solved in the required sampling time. These three examples were

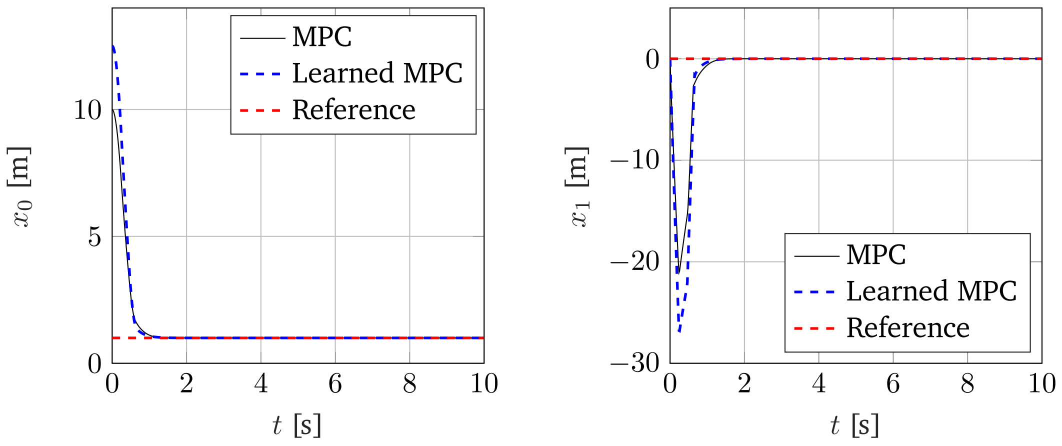


Figure 7.7.: Results of the learned MPC.

chosen to show how machine learning models can be used to learn not only the dynamics, as commonly done in literature but also other components of the controller. The code of the presented examples is openly available in the HILO-MPC repository. Upon the submission of the thesis, Stochastic MPC was also developed but has not yet been released officially. Hence, it was not discussed here. We believe that HILO-MPC's flexibility and simplicity can make it a useful tool for both research and teaching.

8. Conclusions and Outlook

Machine learning can offer valid support to MPC, where using purely first-principles models is difficult. Hence, the integration of machine learning models with MPC methods is currently a very active research area. In this thesis, we proposed two machine learning-supported MPC approaches that consider data directly in the control design, i.e., not only to learn the unknown dynamics but also to force the system to stay in regions where the model is more accurate. The approaches proposed focused on repetitive processes, i.e., they do not usually work in steady-state conditions.

The first approach was a risk-aware MPC. In this approach, an ensemble of Gaussian processes was used to build a risk map that entered the MPC constraints and informed the MPC on the possible modeling risk, i.e., modeling error, across the feature space. We show in an example that the risk, in our case studies, correlated well with model error. Thanks to this risk function, the MPC could avoid high-risk areas. As the number of repetitions increases, the risk over the feature spaces decreases because the machine learning model enlarges its predictive confidence across the feature space. This enlargement is reflected in a greater performance of the plant as the number of runs increases. We applied the method on two fed-batch bioreactors, using open-loop optimization and MPC. The first is useful in case no online measurements are available. We showed that a balance exists between exploration and exploitation and that this balance can be tuned by selecting the maximum allowed risk that the controller is allowed to take. We also showed that the MPC performance run-to-run is not necessarily better than an open-loop control since the data collected, and hence the model, are different in the two cases, resulting in different control performances. While this approach is simple, it does not guarantee robust constraint satisfaction.

The second approach uses nonlinear tube MPC to guarantee constraint satisfaction. This method also takes the measurements directly into account in the design, together with the Lipschitz constant of the real system. Thanks to this information, it is possible to upper-bound the modeling error. Then, a *safe set* is found, i.e., a set where the upper bound on the modeling error is small enough, and a feasible solution of nonlinear tube MPC can be found. The system is then forced to remain in this safe set. In the run, new measurements are collected. If a machine learning model exists such that the modeling error on the safe set decreases from run to run, we proved that the safe set expands. We also provided a set expansion method that guaranteed a priori that if a set at run i is safe, the updated set $i + 1$ is also safe. To accelerate the expansion of the set, we proposed using design-of-experiment, which aims to decrease the error on the safe set quickly, and an accelerated set expansion (at the price of apriori guarantees that the set is safe). Finally, in the frame of this work, we developed HILO-MPC: An open-source Python toolbox that allows us to easily solve learning-supported problems such as MPC, moving horizon estimator, Kalman filter, and particle filters. The toolbox interfaces with state-of-the-art learning libraries such as TensorFlow and PyTorch. It can use feed-forward neural networks and Gaussian processes as machine learning models. Thanks to its integration with software for embedded MPC, such as *μAO-MPC*and SAM, the machine learning-supported problem can also be easily deployed in embedded applications.

In the frame of the proposed risk-aware MPC, future research could be focused on using a multi-dimensional risk function that describes the risk of every output of the machine learning model without lumping the risk in one single measure.

Furthermore, the tube MPC approach proposed used the robust control invariant set definition that is valid for continuous processes. Less conservative constraint tightening could be achieved if, instead, one would take into account the finite-time nature of the process. Regarding the HILO-MPC toolbox, upon submission of this thesis, we also implemented a Stochastic MPC formulation using Gaussian processes and a UKF-based stochastic MPC [30]. We are currently working on implementing other MPC formulations, such as multi-mode MPC and stochastic MPC, using Bayesian Neural Networks.

A. Appendix

A.1. Notation and definitions

Here we state some definitions and notations used throughout the paper.

Definition A.1 (Class $\mathcal{K}$ functions) *A function $\alpha : [0, \infty) \to [0, \infty)$ belongs to the set of class $\mathcal{K}$ functions if it is continuous, strictly increasing and $\alpha(0) = 0$.*

Definition A.2 (Set addition and difference) *Consider two sets $\mathcal{A}, \mathcal{B} \subset \mathbb{R}^n$, the set difference (or Pontryagin difference) is defined as*

$$\mathcal{A} \ominus \mathcal{B} = \{x \in \mathbb{R}^n | x + y \in \mathcal{A},\ \ \forall y \in \mathcal{B}\} \tag{A.1}$$

the set addition (or Minkowski sum) is defined as

$$\mathcal{A} \oplus \mathcal{B} = \{x + y \in \mathbb{R}^n | x \in \mathcal{A},\ \ y \in \mathcal{B}\} \tag{A.2}$$

Definition A.3 (Multiplication of set by a matrix) *A set $\mathcal{B} \subset \mathbb{R}^n$ multiplied by a matrix $A \in \mathbb{R}^{n \times m}$ is*

$$A\mathcal{B} = \{c \mid \exists b \in \mathcal{B},\ \ c = Ab\} \tag{A.3}$$

A.2. Theorems and Proofs

A.2.1. Proof of Theorem A.2

The proof follows closely [206] with the difference that here we do not consider any matrix multiplying the noise vector. The idea is guessing a function $\tilde{S}$ that satisfies the condition in (52).

Proof We consider first a linear system with disturbances

$$\dot{s}(t) = (A + BK)s(t) + d(t)$$

Let us define the function $\tilde{S}(s(t)) \triangleq s(t)^\mathsf{T} P s(t)$. By substituting this in (52) we obtain

$$\begin{aligned}
&\dot{\tilde{S}}(s(t)) + \lambda_0 \tilde{S}(s(t)) - \mu d(t)^\mathsf{T} d(t) = \\
&= s(t)^\mathsf{T} \left[(A+BK)^\mathsf{T} P + P(A+BK) \right] s(t) \\
&+ d(t)^\mathsf{T} P s(t) + s(t)^\mathsf{T} P d(t) + \lambda_0 s(t)^\mathsf{T} P s(t) - \mu d(t)^\mathsf{T} d(t).
\end{aligned}$$

For simplicity define $H(s(t)) = \dot{\tilde{S}}(s(t)) + \lambda_0 \tilde{S}(s(t)) - \mu d(t)^\mathsf{T} d(t)$. If we multiply both sides of (54) with $\text{diag}(P, I)$ and substitute $P = X^{-1}$ and $K = YX^{-1}$ we obtain:

$$\begin{bmatrix} (A+BK)^T P + P(A+BK) + \lambda_0 P & P \\ P & -\mu \end{bmatrix} \leq 0 \tag{A.4}$$

By multiplying both sides of (32) with $[s(t)\ d(t)]$ and $[s(t)\ d(t)]^T$ respectively, we have $H(s(t)) \leq 0$. Because of Theorem A.1, there exists a robust control invariant set Ω for the system $\dot{s}(t) = (A+BK)s(t) + d(t)$, where $\Omega = \{s \in \mathbb{R}^{n_x} | s^\mathsf{T} P s \leq \frac{\mu w^2_{\max,\mathcal{X}}}{\lambda_0}\}$. Denote $M(z(t)) = \dot{S}(z(t)) + \lambda S(z(t)) - \mu d(t)^\mathsf{T} d(t)$ with $\lambda < \lambda_0$, and use the error system (51), we obtain:

$$\begin{aligned}
M(z(t)) =& \dot{S}(z(t)) + \lambda S(z(t)) - \mu d(t)^\mathsf{T} d(t) \\
=& z(t)^\mathsf{T} \left[(A+BK)^\mathsf{T} P + P(A+BK) \right] z(t) \\
& + 2 d(t)^\mathsf{T} P z(t) + \lambda z(t)^\mathsf{T} P z(t) - \mu d(t)^\mathsf{T} d(t) \\
& + 2[g_\theta(x(t)) - g_\theta(\bar{x}(t))]^\mathsf{T} P z(t) \\
=& H(z(t)) + (\lambda - \lambda_0) z(t)^\mathsf{T} P z(t) \\
& + 2[g_\theta(x(t)) - g_\theta(\bar{x}(t))]^\mathsf{T} P z(t).
\end{aligned}$$

Since $H(z(t)) \leq 0$ and $\alpha_{\text{min}}(P)\|z(t)\|^2 \leq z(t)^\mathsf{T} P z(t) \leq \alpha_{\text{max}}(P)\|z(t)\|^2$ we obtain:

$$\begin{aligned}
M(z(t)) &\leq (\lambda - \lambda_0) z(t)^\mathsf{T} P z(t) + 2[g_\theta(x(t)) - g_\theta(\bar{x}(t))]^\mathsf{T} P z(t) \\
&\leq (\lambda - \lambda_0) \alpha_{\text{min}}(P)\|z(t)\|^2 + 2[g_\theta(x(t)) - g_\theta(\bar{x}(t))]^\mathsf{T} P z(t)
\end{aligned}$$

By using the definition of Lipschitz continuity and (55), we have

$$\begin{aligned}
M(z(t)) \leq& (\lambda - \lambda_0) \alpha_{\text{min}}(P)\|z(t)\|^2 + 2L_\mathcal{X}\|P\|\|z(t)\|^2 \\
=& (2L_\mathcal{X}\|P\| + (\lambda - \lambda_0)\alpha_{\text{min}}(P))\|z(t)\|^2 \leq 0
\end{aligned}$$

hence Ω^i is robust invariant for the error system.

A.2.2. Proof of Lemma A.1

Proof We prove first that the functions $H_n^{\mathrm{u}}(x;\mathcal{D})$ and $H_n^{\mathrm{l}}(x;\mathcal{D})$ are Lipschitz continuous with respect to x. We prove it for $H_n^{\mathrm{u}}(x;\mathcal{D})$ since for $H_n^{\mathrm{l}}(x;\mathcal{D})$ a very similar proof follows. We have that $\forall k \in I_{\mathcal{D}}$

$$\left|\left(\hat{y}_{k,n} + \tilde{L}\|\hat{x}_k - x\| + v_{n,\max}\right) - \left(\hat{y}_{k,n} + \tilde{L}\|\hat{x}_k - \tilde{x}\| + v_{n,\max}\right)\right| = \\ = \tilde{L}\,|\|\hat{x}_k - x\| - \|\hat{x}_k - \tilde{x}\|| \leq \tilde{L}(\|\hat{x}_k - x - (\hat{x}_k - \tilde{x}))\| \leq \tilde{L}\|\tilde{x} - x\|$$

Note that the upper-bound $\tilde{L}\|\tilde{x} - x\|$ does not depend on k hence

$$|H_n^{\mathrm{u}}(x;\mathcal{D}) - H_n^{\mathrm{u}}(\tilde{x};\mathcal{D})| \leq \tilde{L}\|\tilde{x} - x\|. \tag{A.5}$$

Next, for Assumption A.7, ρ_θ is Lipschitz continuous. Let L be a *Lipschitz* constant. Hence it holds that

$$|\rho_\theta(x) - \rho_\theta(\tilde{x})| \leq L\|\tilde{x} - x\| \tag{A.6}$$

summing this with (33) we obtain

$$|\rho_{\theta,n}(x) - \rho_{\theta,n}(\tilde{x})| + |H_n^{\mathrm{u}}(\tilde{x};\mathcal{D}) - H_n^{\mathrm{u}}(x;\mathcal{D})| \leq (\tilde{L} + L)\|\tilde{x} - x\|$$
$$|\rho_{\theta,n}(x) - \rho_{\theta,n}(\tilde{x}) + H_n^{\mathrm{u}}(\tilde{x};\mathcal{D}) - H_n^{\mathrm{u}}(x;\mathcal{D})| \leq |\rho_{\theta,n}(x) - \rho_{\theta,n}(\tilde{x})| + |H_n^{\mathrm{u}}(x;\mathcal{D}) - H_n^{\mathrm{u}}(\tilde{x};\mathcal{D})|$$
$$|\rho_{\theta,n}(x) - H_n^{\mathrm{u}}(x;\mathcal{D}) - (\rho_{\theta,n}(\tilde{x}) - H_n^{\mathrm{u}}(\tilde{x};\mathcal{D}))\,| \leq (\tilde{L} + L)\|\tilde{x} - x\|$$
$$|e_{n,u}(x;\mathcal{D}) - e_{n,u}(\tilde{x};\mathcal{D})| \leq (\tilde{L} + L)\|\tilde{x} - x\|.$$

The same follows for $e_{n,l}(x;\mathcal{D})$. Hence, since the maximum operator in (71e) will either select $e_{n,l}(x;\mathcal{D})$ or $e_{n,u}(x;\mathcal{D})$ it follows that:

$$|\tilde{w}_{n,\max}(x;\mathcal{D}) - \tilde{w}_{n,\max}(\tilde{x};\mathcal{D})| \leq (\tilde{L} + L)\|\tilde{x} - x\|. \tag{A.7}$$

Hence $\tilde{w}_{n,\max}(x;\mathcal{D}),\ \ n \in [0, ..., n_x]$ are Lipschitz continuous with Lipschitz constant $L + \tilde{L}$.

A.3. Miscellaneous

A.3.1. Computation of λ for robust control invariant set

We have the following conditions on the Lipschitz constant cf.(55) and one-side Lipschitz constant respectivelly:

$$L_{\mathcal{X}^i} \leq \frac{(\lambda_0^* - \lambda)\alpha_{\text{min}}(P)^*}{2\|P^*\|}, \tag{A.8}$$

$$\mathcal{L}_{\mathcal{X}^i} \leq \frac{(\lambda_0^* - \lambda)\alpha_{\text{min}}(P)^*}{2}, \tag{A.9}$$

The values of λ_0 and P found by solving (54). $L_{\mathcal{X}^i}$ and $\mathcal{L}_{\mathcal{X}^i}$ are also given. The unknown is then λ. For the case with the vanilla Lipschitz constant we have

$$\lambda \leq \lambda_0^* - 2\frac{L_{\mathcal{X}^i}\|P^*\|}{\alpha_{\text{min}}^*(P)} \tag{A.10}$$

Since P is positive definite, $\|P\| > 0$ and $\alpha_{\text{min}}(P) > 0$, furthermore $L_{\mathcal{X}^i} \geq 0$. Hence the condition $\lambda < \lambda_0$ is automatically satisfied. From (53) we see that we would like λ to be as large as possible. Hence we chose

$$\lambda := \lambda_0^* - 2\frac{L_{\mathcal{X}^i}\|P^*\|}{\alpha_{\text{min}}^*(P)} \tag{A.11}$$

The one-sided Lipschitz constant could also be negative. If it is negative we obtain

$$\lambda \leq \lambda_0^* + 2\frac{|L_{\mathcal{X}^i}|\|P^*\|}{\alpha_{\text{min}}^*(P)} \tag{A.12}$$

which is always guaranteed. So it is sufficient to take a non-zero value very close, but smaller, than λ_0^* to guarantee $0 < \lambda < \lambda_0^*$ as required by Theorem A.2.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[2] Victoria O. Adesanya, Matthew P. Davey, Stuart A. Scott, and Alison G. Smith. "Kinetic modelling of growth and storage molecule production in microalgae under mixotrophic and autotrophic conditions". In: *Bioresource Technology* 157 (2014), pp. 293–304. ISSN: 0960-8524. DOI: `https://doi.org/10.1016/j.biortech.2014.01.032`. URL: `https://www.sciencedirect.com/science/article/pii/S0960852414000571`.

[3] V. Adetola, D. DeHaan, and M. Guay. "Adaptive model predictive control for constrained nonlinear systems". In: *Systems and Control Letters* 58.5 (2009), pp. 320–326. DOI: `10.1016/j.sysconle.2008.12.002`.

[4] V.A. Akpan and G.D. Hassapis. "Nonlinear model identification and adaptive model predictive control using neural networks". In: *ISA Transactions* 50.2 (2011), pp. 177–194. DOI: `10.1016/j.isatra.2010.12.007`.

[5] Frank Allgöwer, Thomas A Badgwell, Joe S Qin, James B Rawlings, and Steven J Wright. "Nonlinear predictive control and moving horizon estimation—an introductory overview". In: *Advances in control* (1999), pp. 391–449.

[6] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[7] N. Amann, D. Owens, and E. Rogers. "Iterative learning control for discrete-time systems with exponential rate of convergence". In: 1996.

[8] Francesco Amato, Roberto Ambrosino, Marco Ariola, Carlo Cosentino, Gianmaria De Tommasi, et al. *Finite-time stability and control*. Vol. 453. Springer, 2014.

[9] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.

[10] Leopoldo Armesto, Jorren Bosga, Vladimir Ivan, and Sethu Vijayakumar. "Efficient learning of constraints and generic null space policies". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 1520–1526. DOI: 10.1109/ICRA.2017.7989181.

[11] Anil Aswani, Humberto Gonzalez, S. Shankar Sastry, and Claire Tomlin. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013), pp. 1216–1226.

[12] Mithun Babu, Yash Oza, Arun Kumar Singh, K Madhava Krishna, and Shanti Medasani. "Model predictive control for autonomous driving based on time scaled collision cone". In: *2018 European Control Conference (ECC)*. IEEE. 2018, pp. 641–648.

[13] Egbert Bakker, Lars Nyborg, and Hans B Pacejka. "Tyre Modelling for Use in Vehicle Dynamics Studies". In: *SAE Transactions* (1987), pp. 190–204. DOI: 10.4271/870421.

[14] Y. Bar-Shalom and E. Tse. "Concepts and Methods in Stochastic Control". In: *Control and Dynamic Systems* 12.C (1976), pp. 99–172. DOI: 10.1016/B978-0-12-012712-2.50009-3.

[15] A.G. Beccuti, S. Mariethoz, S. Cliquennois, S. Wang, and M. Morari. "Explicit model predictive control of DC-DC switched-mode power supplies with extended Kalman filtering". In: *IEEE Transactions on Industrial Electronics* 56.6 (2009), pp. 1864–1874. DOI: 10.1109/TIE.2009.2015748.

[16] Lukas Beckenbach, Pavel Osinenko, and Stefan Streif. "Addressing infinite-horizon optimization in MPC via Q-learning". In: *IFAC-PapersOnLine* 51.20 (2018). 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018, pp. 60–65. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.10.175. URL: https://www.sciencedirect.com/science/article/pii/S2405896318326478.

[17] Lukas Beckenbach, Pavel Osinenko, and Stefan Streif. “Addressing infinite-horizon optimization in MPC via Q-learning”. In: *IFAC-PapersOnLine* 51.20 (2018). 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018, pp. 60–65. DOI: 10.1016/j.ifacol.2018.10.175.

[18] Gleb Beliakov. “Interpolation of Lipschitz functions”. In: *J. Comput. Appl. Math.* 196.1 (2006), pp. 20–44. ISSN: 0377-0427.

[19] Richard Bellman. “Dynamic programming and a new formalism in the calculus of variations”. In: *Proceedings of the National Academy of Sciences* 40.4 (1954), pp. 231–235. ISSN: 0027-8424. DOI: 10.1073/pnas.40.4.231. eprint: https://www.pnas.org/content/40/4/231.full.pdf. URL: https://www.pnas.org/content/40/4/231.

[20] Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. “Safe model-based reinforcement learning with stability guarantees”. In: *arXiv preprint arXiv:1705.08551* (2017).

[21] Daniele Bernardini and Alberto Bemporad. “Scenario-based Model Predictive Control of Stochastic Constrained Linear Systems”. In: (2009).

[22] Massimo Bertolini, Davide Mezzogori, Mattia Neroni, and Francesco Zammori. “Machine Learning for industrial applications: A comprehensive literature review”. In: *Expert Systems with Applications* 175 (2021), p. 114820.

[23] Johanna Bethge, Bruno Morabito, Janine Matschek, and Rolf Findeisen. “Multi-Mode Learning Supported Model Predictive Control with Guarantees”. In: *Proc. of Nonlinear Model Predicitve Control*. Madison, 2018, To appear.

[24] Lorenz T. Biegler. *Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2010. DOI: 10.1137/1.9780898719383.

[25] H.G. Bock and K.J. Plitt. “A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems*”. In: *IFAC Proceedings Volumes* 17.2 (1984). 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984, pp. 1603–1608. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)61205-9. URL: https://www.sciencedirect.com/science/article/pii/S1474667017612059.

[26] Bokeh Development Team. *Bokeh: Python library for interactive visualization*. 2018. URL: https://bokeh.pydata.org/en/latest/.

[27] Pierre Bonami and Jon Lee. “BONMIN user’s manual”. In: *Numer Math* 4 (2007), pp. 1–32.

[28] DOMINIQUE BONVIN, BALA SRINIVASAN, and DAVID HUNKELER. “Control and optimization of batch processes”. In: *IEEE Control Systems Magazine* 26.6 (2006), pp. 34–45. DOI: 10.1109/MCS.2006.252831.

[29] George EP Box and George C Tiao. *Bayesian inference in statistical analysis*. Vol. 40. John Wiley & Sons, 2011.

[30] Eric Bradford and Lars Imsland. “Economic Stochastic Model Predictive Control Using the Unscented Kalman Filter□□This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 675215.” en. In: *IFAC-PapersOnLine*. 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018 51.18 (Jan. 2018), pp. 417–422. ISSN: 2405-8963. DOI: 10.1016/j.ifacol.2018.09.336. URL: https://www.sciencedirect.com/science/article/pii/S2405896318320196 (visited on 05/07/2023).

[31] Eric Bradford and Lars Imsland. “Stochastic Nonlinear Model Predictive Control Using Gaussian Processes”. In: *2018 European Control Conference (ECC)*. 2018, pp. 1027–1034. DOI: 10.23919/ECC.2018.8550249.

[32] D. A. Bristow, M. Tharayil, and A. G. Alleyne. “A survey of iterative learning control”. In: *IEEE Control Systems Magazine* 26.3 (2006), pp. 96–114. DOI: 10.1109/MCS.2006.1636313.

[33] M. Brunner, U. Rosolia, J. Gonzales, and F. Borrelli. “Repetitive learning model predictive control: An autonomous racing example”. In: *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017, pp. 2545–2550. DOI: 10.1109/CDC.2017.8264027.

[34] Monimoy Bujarbaruah, Charlott Vallon, and Francesco Borrelli. *Learning to Satisfy Unknown Constraints in Iterative MPC*. 2020. arXiv: 2006.05054 [eess.SY].

[35] Jan-Peter Calliess. “Conservative decision-making and inference in uncertain dynamical systems”. PhD thesis. University of Oxford, 2014.

[36] Jan-Peter Calliess. *Lazily Adapted Constant Kinky Inference for Nonparametric Regression and Model-Reference Adaptive Control*. 2021. arXiv: 1701.00178 [math.OC].

[37] Yankai Cao and R. Bhushan Gopaluni. “Deep Neural Network Approximation of Nonlinear Model Predictive Control”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 11319–11324. DOI: 10.1016/j.ifacol.2020.12.538.

[38] R. Carli, G. Cavone, S.B. Othman, and M. Dotoli. “IoT based architecture for model predictive control of HVAC systems in smart buildings”. In: *Sensors (Switzerland)* 20.3 (2020). DOI: 10.3390/s20030781.

[39] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D. Lee, Vijay Kumar, George J. Pappas, and Manfred Morari. “Approximating Explicit Model Predictive Control Using Constrained Neural Networks”. In: 2018 Annual American Control Conference (ACC). IEEE. 2018, pp. 1520–1527. DOI: 10.23919/ACC.2018.8431275.

[40] Yangquan Chen and Changyun Wen. *Iterative learning control: convergence, robustness and applications*. Springer London, 1999.

[41] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. “End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3387–3395.

[42] Jason Choi, Fernando Castaneda, Claire J Tomlin, and Koushil Sreenath. “Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions”. In: *arXiv preprint arXiv:2004.07584* (2020).

[43] IBM ILOG Cplex. “V12. 1: User’s Manual for CPLEX”. In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[44] Lehel Huba Csekő, Michal Kvasnica, and Béla Lantos. “Explicit MPC-Based RBF Neural Network Controller Design With Discrete-Time Actual Kalman Filter for Semiactive Suspension”. In: *IEEE Transactions on Control Systems Technology* 23.5 (2015), pp. 1736–1753. DOI: 10.1109/TCST.2014.2382571.

[45] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[46] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. *Safe Exploration in Continuous Action Spaces*. 2018. arXiv: 1801.08757 [cs.AI].

[47] Y. Ding, L. Wang, Y. Li, and D. Li. “Model predictive control and its application in agriculture: A review”. In: *Computers and Electronics in Agriculture* 151 (2018), pp. 104–117. DOI: 10.1016/j.compag.2018.06.004.

[48] Maarten R. Dobbelaere, Pieter P. Plehiers, Ruben Van de Vijver, Christian V. Stevens, and Kevin M. Van Geem. “Machine Learning in Chemical Engineering: Strengths, Weaknesses, Opportunities, and Threats”. In: *Engineering* 7.9 (2021), pp. 1201–1211. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2021.03.019. URL: https://www.sciencedirect.com/science/article/pii/S2095809921002010.

[49] Tae-Yong Doh. “Robust iterative learning control with current feedback for uncertain linear systems”. In: *International Journal of Systems Science* 30.1 (1999), pp. 39–47. DOI: 10.1080/002077299292650. URL: https://doi.org/10.1080/002077299292650.

[50] Alexander Domahidi and Juan Jerez. *FORCES Professional*. Embotech AG. https://embotech.com/FORCES-Pro. 2014–2019.

[51] Dong Dong, Thomas J. McAvoy, and Evanghelos Zafiriou. “Batch-to-batch optimization using neural network models”. In: *Industrial and Engineering Chemistry Research* 35.7 (1996), pp. 2269–2276. ISSN: 08885885. DOI: 10.1021/ie950518p.

[52] Sebastian Espinel Rios, Katja Bettenbrock, Steffen Klamt, and Rolf Findeisen. “Maximizing batch fermentation efficiency by constrained model-based optimization and predictive control of adenosine triphosphate turnover”. In: *AIChE Journal* n/a.n/a (), e17555. DOI: https://doi.org/10.1002/aic.17555. eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.17555. URL: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.17555.

[53] Sebastián Espinel-Ríos, Gerrich Behrendt, Bruno Morabito, Jasmin Bauer, Johannes Pohlodek, Andreas Schülze, Katja Bettenbrock, Steffen Klamt, and Rolf Findeisen. “Experimentally implemented dynamic optogenetic optimization of the ATPase expression using knowledge-based and Gaussian-process-supported models”. In: *Journal of Process Control* (2023). To be submitted.

[54] Sebastián Espinel-Ríos, Nicolas Huber, Edgar Alberto Alcalá-Orozco, Bruno Morabito, Thomas F.T. Rexer, Udo Reichl, Steffen Klamt, and Rolf Findeisen. “Cell-free biosynthesis meets dynamic optimization and control: a fed-batch framework”. In: *IFAC-PapersOnLine* 55.23 (2022). 9th IFAC Conference on Foundations of Systems Biology in Engineering FOSBE 2022, pp. 92–97. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.01.021. URL: https://www.sciencedirect.com/science/article/pii/S2405896323000241.

[55] Timm Faulwasser, Tobias Weber, Pablo Zometa, and Rolf Findeisen. “Implementation of nonlinear model predictive path-following control for an industrial robot”. In: *IEEE Transactions on Control Systems Technology* 25.4 (2016), pp. 1505–1511.

[56] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. “Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks”. In: *CoRR* abs/1906.04893 (2019).

[57] AA Feldbaum. “Dual control theory. I”. In: *Avtomatika i Telemekhanika* 21.9 (1960), pp. 1240–1249.

[58] AA Feldbaum. “Dual control theory. II”. In: *Avtomatika i Telemekhanika* 21.11 (1960), pp. 1453–1464.

[59] AA Feldbaum. “Dual control theory III”. In: *Automation remote control* 22 (1961), pp. 1–12.

[60] Alexander Aronovich Feldbaum. *Dual control theory, IV*. Tech. rep. FOREIGN TECHNOLOGY DIV WRIGHT-PATTERSON AFB OHIO, 1961.

[61] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. “qpOASES: A parametric active-set algorithm for quadratic programming”. In: *Mathematical Programming Computation* 6.4 (2014), pp. 327–363.

[62] Rolf Findeisen. “Nonlinear model predictive control: a sampled data feedback perspective”. PhD thesis. 2006.

[63] Michael G. Forbes, Rohit S. Patwardhan, Hamza Hamadah, and R. Bhushan Gopaluni. “Model Predictive Control in Industry: Challenges and Opportunities”. In: *IFAC-PapersOnLine* 48.8 (2015). 9th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2015, pp. 531–538. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.09.022. URL: https://www.sciencedirect.com/science/article/pii/S2405896315011039.

[64] Swanny Fouchard, Jérémy Pruvost, Benoit Degrenne, Mariana Titica, and Jack Legrand. “Kinetic modeling of light limitation and sulfur deprivation effects in the induction of hydrogen production with Chlamydomonas reinhardtii: Part I. Model development and parameter identification”. In: *Biotechnology and Bioengineering* 102.1 (2009), pp. 232–245. DOI: https://doi.org/10.1002/bit.22034. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bit.22034. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/bit.22034.

[65] A. Franz. “Nonlinear dynamics of PHB production in Ralstonia eutropha and Rhodospirillum rubrum”. PhD thesis. Otto-von-Guericke-Universität Magdeburg, 2015.

[66] A. Franz, H.-S. Song, D. Ramkrishna, and A. Kienle. “Experimental and theoretical analysis of Poly(β-hydroxybutyrate) formation and consumption in *Ralstonia eutropha*”. In: *Biochemical Eng. J.* 55 (2011), pp. 49–58.

[67] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. “GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration”. In: *Advances in Neural Information Processing Systems*. 2018.

[68] Gene Grimm, Michael J. Messina, Sezai E. Tuna, and Andrew R. Teel. “Examples when nonlinear model predictive control is nonrobust”. In: *Automatica* 40.10 (2004), pp. 1729–1738. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2004.04.014. URL: https://www.sciencedirect.com/science/article/pii/S0005109804001402.

[69] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2021. URL: http://www.gurobi.com.

[70] Tor Aksel N. Heirung, Joel A. Paulson, Shinje Lee, and Ali Mesbah. “Model predictive control with active learning under model uncertainty: Why, when, and how”. In: *AIChE Journal* 64.8 (2018), pp. 3071–3081. DOI: https://doi.org/10.1002/aic.16180. eprint: https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.16180. URL: https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.16180.

[71] Tor Aksel N Heirung, Tito LM Santos, and Ali Mesbah. “Model predictive control with active learning for stochastic systems with structural model uncertainty: Online model discrimination”. In: *Computers & Chemical Engineering* 128 (2019), pp. 128–140.

[72] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. “Multi-Parametric Toolbox 3.0”. In: *Proc. of the European Control Conference*. http://control.ee.ethz.ch/~mpt. Zürich, Switzerland, 2013, pp. 502–510.

[73] Martin Wijaya Hermanto, Richard D Braatz, and Min-Sen Chiu. “Integrated batch-to-batch and nonlinear model predictive control for polymorphic transformation in pharmaceutical crystallization”. In: *AIChE journal* 57.4 (2011), pp. 1008–1019.

[74] Lukas Hewing, Kim P. Wabersich, Marcel Menner, and Melanie N. Zeilinger. "Learning-Based Model Predictive Control: Toward Safe Learning in Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 269–296. ISSN: 2573-5144. DOI: 10.1146/annurev-control-090419-075625.

[75] Lukas Hewing, Kim P. Wabersich, Marcel Menner, and Melanie N. Zeilinger. "Learning-Based Model Predictive Control: Toward Safe Learning in Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1 (2020), pp. 269–296. DOI: 10.1146/annurev-control-090419-075625. eprint: https://doi.org/10.1146/annurev-control-090419-075625. URL: https://doi.org/10.1146/annurev-control-090419-075625.

[76] Rubin Hille, Jasdeep Mandur, and Hector M Budman. "Robust batch-to-batch optimization in the presence of model-plant mismatch and input uncertainty". In: *AIChE Journal* 63.7 (2017), pp. 2660–2670.

[77] Andreas Himmel, Janine Matschek, Rudolph Kok, Bruno Morabito, Hoang Hai Nguyen, and Rolf Findeisen. "Machine Learning for Process Control of (Bio)Chemical Processes". In: *Artificial Intelligence in Manufacturing: Concepts and Methods*. Ed. by Soroush Masoud and Braatz Richard. To appear, preprint available via arXiv. Elsevier, 2023. DOI: 10.48550/ARXIV.2301.06073. URL: https://arxiv.org/abs/2301.06073.

[78] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers". In: *ACM Transactions on Mathematical Software* 31.3 (2005), pp. 363–396. DOI: 10.1145/1089014.1089020.

[79] Hirokazu Ishida. *Robust tube MPC*. https://github.com/HiroIshida/robust-tube-mpc. (accessed 2022/28/1).

[80] Yiguang Hong, Zhong-Ping Jiang, and Gang Feng. "Finite-time input-to-state stability and applications to finite-time control design". In: *SIAM Journal on Control and Optimization* 48.7 (2010), pp. 4395–4418.

[81] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[82] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

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

[84] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[85] Tomoharu Iwata and Zoubin Ghahramani. "Improving output uncertainty estimation and generalization in deep learning via neural network Gaussian processes". In: *arXiv preprint arXiv:1707.05922* (2017).

[86] H.A. Izadi, Y. Zhang, and B.W. Gordon. "Fault tolerant model predictive control of quad-rotor helicopters with actuator fault estimation". In: vol. 44. 1 PART 1. 2011, pp. 6343–6348. DOI: 10.3182/20110828-6-IT-1002.03709.

[87] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[88] Christian Kallies. "Approximated adaptive explicit parametric optimal control". PhD thesis. Otto-von-Guericke-Universität Magdeburg, Fakultät für Elektrotechnik und Informationstechnik, 2021. URL: http://dx.doi.org/10.25673/38707.

[89] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (1960), p. 35. ISSN: 00219223. DOI: 10.1115/1.3662552. arXiv: NIHMS150003. URL: https://www.cs.unc.edu/welch/kalman/media/pdf/Kalman1960.pdfhttp://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402.

[90] Keiji Kanazawa, Daphne Koller, and Stuart Russell. "Stochastic simulation algorithms for dynamic probabilistic networks". In: UAI'95: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. ACM. 1995, pp. 346–351.

[91] Benjamin Karg and Sergio Lucia. "Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning". In: *IEEE Transactions on Cybernetics* 50.9 (2020), pp. 3866–3878. DOI: 10.1109/TCYB.2020.2999556.

[92] George Karniadakis, Yannis Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. "Physics-informed machine learning". In: (May 2021), pp. 1–19. DOI: 10.1038/s42254-021-00314-5.

[93] A. Kathirgamanathan, M. De Rosa, E. Mangina, and D.P. Finn. "Data-driven predictive control for unlocking building energy flexibility: A review". In: *Renewable and Sustainable Energy Reviews* 135 (2021). DOI: 10.1016/j.rser.2020.110120.

[94] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys. “Learning in centralized nonlinear model predictive control: Application to an autonomous tractor-trailer system”. In: *IEEE Transactions on Control Systems Technology* 23.1 (2015), pp. 197–205. DOI: 10.1109/TCST.2014.2321514.

[95] Irfan Khan, Stefano Feraco, Angelo Bonfitto, and Nicola Amati. “A Model Predictive Control Strategy for Lateral and Longitudinal Dynamics in Autonomous Driving”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 83938. American Society of Mechanical Engineers. 2020, V004T04A004.

[96] Kwang-Ki K Kim and Richard D Braatz. “Generalised polynomial chaos expansion approaches to approximate stochastic model predictive control”. In: *International journal of control* 86.8 (2013), pp. 1324–1337.

[97] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[98] Markus Kögel and Rolf Findeisen. “Fast predictive control of linear systems combining Nesterov’s gradient method and the method of multipliers”. In: *2011 50th IEEE conference on decision and control and european control conference*. IEEE. 2011, pp. 501–506.

[99] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. “Learning-based model predictive control for safe exploration”. In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 6059–6066.

[100] Richard E Kopp. “Pontryagin maximum principle”. In: *Mathematics in Science and Engineering*. Vol. 5. Elsevier, 1962, pp. 255–279.

[101] Wilbur Langson, Ioannis Chryssochoos, SV Raković, and David Q Mayne. “Robust model predictive control using tubes”. In: *Automatica* 40.1 (2004), pp. 125–133.

[102] Robert Laurini. “6 - Geographic Ontologies”. In: *Geographic Knowledge Infrastructure*. Ed. by Robert Laurini. Elsevier, 2017, pp. 111–137. ISBN: 978-1-78548-243-4. DOI: https://doi.org/10.1016/B978-1-78548-243-4.50006-2. URL: https://www.sciencedirect.com/science/article/pii/B9781785482434500062.

[103] Jay H Lee. “Model predictive control: Review of the three decades of development”. In: *International Journal of Control, Automation and Systems* 9.3 (2011), pp. 415–424.

[104] Jay H. Lee, K. S. Lee, and W. Kim. "Model-based iterative learning control with a quadratic criterion for time-varying linear systems". In: *Autom.* 36 (2000), pp. 641–657.

[105] Jay H. Lee and Kwang S. Lee. "Iterative learning control applied to batch processes: An overview". In: *Control Engineering Practice* 15.10 SPEC. ISS. (2007), pp. 1306–1318. ISSN: 09670661. DOI: 10.1016/j.conengprac.2006.11.013.

[106] J.H. Lee and N.L. Ricker. "Extended Kalman Filter Based Nonlinear Model Predictive Control". In: *Industrial and Engineering Chemistry Research* 33.6 (1994), pp. 1530–1541. DOI: 10.1021/ie00030a013.

[107] Jongdae Lee and W Fred Ramirez. "Optimal fed-batch control of induced foreign protein production by recombinant bacteria". In: *AIChE J.* 40.5 (1994), pp. 899–907.

[108] Kwang Soon Lee and Jay H. Lee. "Model predictive control for nonlinear batch processes with asymptotically perfect tracking". In: *Computers & Chemical Engineering* 21 (1997). Supplement to Computers and Chemical Engineering, S873 –S879. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(97)87612-0. URL: http://www.sciencedirect.com/science/article/pii/S0098135497876120.

[109] Daniel Limon, Teodoro Alamo, Davide M Raimondo, D Muñoz De La Peña, José Manuel Bravo, Antonio Ferramosca, and Eduardo F Camacho. "Input-to-state stability: a unifying framework for robust model predictive control". In: *Nonlinear model predictive control*. Springer, 2009, pp. 1–26.

[110] Alexander Liniger, Alexander Domahidi, and Manfred Morari. "Optimization-based autonomous racing of 1:43 scale RC cars". In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647. DOI: 10.1002/oca.2123.

[111] L. Liu, B. Huang, and S. Dubljevic. "Model predictive control of axial dispersion chemical reactor". In: *Journal of Process Control* 24.11 (2014). cited By 17, pp. 1671–1690. DOI: 10.1016/j.jprocont.2014.08.010. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84908556454&doi=10.1016\%2fj.jprocont.2014.08.010&partnerID=40&md5=79e79366a0c30c04837d27c491ba6b5b.

[112] J. Lofberg. "YALMIP : a toolbox for modeling and optimization in MATLAB". In: *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*. 2004, pp. 284–289. DOI: 10.1109/CACSD.2004.1393890.

[113] S. Lucia and S. Engell. “Multi-stage and Two-stage Robust Nonlinear Model Predictive Control”. In: *IFAC Proceedings Volumes* 45.17 (2012). 4th IFAC Conference on Nonlinear Model Predictive Control, pp. 181 –186. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120823-5-NL-3013.00015. URL: http://www.sciencedirect.com/science/article/pii/S1474667016314471.

[114] Sergio Lucia, Tiago Finkler, and Sebastian Engell. “Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty”. In: *Journal of process control* 23.9 (2013), pp. 1306–1319.

[115] Sergio Lucia, Alexandru Tatulea-Codrean, Christian Schoppmeyer, and Sebastian Engell. “Rapid development of modular and sustainable nonlinear model predictive control solutions”. In: *Control Engineering Practice* 60 (2017), pp. 51–62. ISSN: 0967-0661. DOI: https://doi.org/10.1016/j.conengprac.2016.12.009. URL: https://www.sciencedirect.com/science/article/pii/S0967066116302970.

[116] E.T. Maddalena, C.G. da S. Moraes, G. Waltrich, and C.N. Jones. “A Neural Network Architecture to Learn Explicit MPC Controllers from Data”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 11362–11367. DOI: 10.1016/j.ifacol.2020.12.546.

[117] Jasdeep S Mandur and Hector M Budman. “Simultaneous model identification and optimization in presence of model-plant mismatch”. In: *Chemical Engineering Science* 129 (2015), pp. 106–115.

[118] A.G. Marchetti, G. François, T. Faulwasser, and D. Bonvin. “Modifier adaptation for real-time optimization - Methods and applications”. In: *Processes* 4.4 (2016). DOI: 10.3390/pr4040055.

[119] Janine Matschek, Tobias Bäthge, Timm Faulwasser, and Rolf Findeisen. “Nonlinear Predictive Control for Trajectory Tracking and Path Following: An Introduction and Perspective”. In: *Handbook of Model Predictive Control*. Ed. by Saša V. Raković and William S. Levine. Cham: Springer International Publishing, 2019, pp. 169–198. ISBN: 978-3-319-77489-3. DOI: 10.1007/978-3-319-77489-3_8.

[120] Janine Matschek, Johanna Bethge, Mohamed Soliman, Bahaaeldin Elsayed, and Rolf Findeisen. “Constrained reference learning for continuous-time model predictive tracking control of autonomous systems”. In: *IFAC-PapersOnLine* 54.6 (2021). 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021, pp. 329–334. DOI: 10.1016/j.ifacol.2021.08.565.

[121] Janine Matschek, Tim Gonschorek, Magnus Hanses, Norbert Elkmann, Frank Ortmeier, and Rolf Findeisen. *Learning References with Gaussian Processes in Model Predictive Control applied to Robot Assisted Surgery*. 2019. arXiv: `1911.10793 [math.OC]`.

[122] Janine Matschek, Andreas Himmel, Kai Sundmacher, and Rolf Findeisen. "Constrained Gaussian process learning for model predictive control". In: *IFAC-PapersOnLine* 53.2 (2020), pp. 971–976.

[123] D. Q. Mayne, M. M. Seron, and S. V. Raković. "Robust model predictive control of constrained linear systems with bounded disturbances". In: *Automatica* 41.2 (2005), pp. 219–224. ISSN: 00051098. DOI: `10.1016/j.automatica.2004.08.019`. arXiv: `arXiv:1011.1669v3`.

[124] David Q Mayne and Eric C Kerrigan. "TUBE-BASED ROBUST NONLINEAR MODEL PREDICTIVE CONTROL". In: (). URL: `http://www2.ee.ic.ac.uk/publications/p5582.pdf`.

[125] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. "Constrained model predictive control: Stability and optimality". In: *Automatica* 36.6 (2000), pp. 789–814. ISSN: 0005-1098. DOI: `https://doi.org/10.1016/S0005-1098(99)00214-9`. URL: `https://www.sciencedirect.com/science/article/pii/S0005109899002149`.

[126] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: `10.1007/BF02478259`.

[127] Ali Mesbah. "Stochastic model predictive control with active uncertainty learning: a survey on dual control". In: *Annual Reviews in Control* 45 (2018), pp. 107–117.

[128] Bruno Morabito, Achim Kienle, Rolf Findeisen, and Lisa Carius. "Multi-mode Model Predictive Control and Estimation for Uncertain Biotechnological Processes". In: *12th International-Federation-of-Automatic-Control (IFAC) Symposium on Dynamics and Control of Process Systems including Biosystems (DYCOPS)*. Elsevier. 2019, pp. 709–714.

[129] Bruno Morabito, Hoang Hai Nguen, Janine Matschek, and Rolf Findeisen. "Safe Exploration Learning Supported Model Predictive Control of Repetitive Processes". In: *Proceeding American Control Conference*. Accepted - to appear. 2022.

[130] Bruno Morabito, Johannes Pohlodek, Lena Kranert, Sebastián Espinel-Ríos, and Rolf Findeisen. “Efficient and Simple Gaussian Process Supported Stochastic Model Predictive Control for Bioreactors using HILO-MPC”. In: *IFAC-PapersOnLine* 55.7 (2022). 13th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems DYCOPS 2022, pp. 922–927. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2022.07.562. URL: https://www.sciencedirect.com/science/article/pii/S2405896322009685.

[131] Bruno Morabito, Johannes Pohlodek, Janine Matschek, Anton Savchenko, Lisa Carius, and Rolf Findeisen. “Towards Risk-aware Machine Learning Supported Model Predictive Control and Open-loop Optimization for Repetitive Processes”. In: *IFAC-PapersOnLine* 54.6 (2021). 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021, pp. 321–328. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2021.08.564. URL: https://www.sciencedirect.com/science/article/pii/S2405896321013392.

[132] Christian Musso, Nadia Oudjane, and Francois Le Gland. “Improving Regularised Particle Filters”. In: *Sequential Monte Carlo Methods in Practice*. Ed. by Arnaud Doucet, Nando de Freitas, and Neil Gordon. Statistics for Engineering and Information Science. Springer, 2001, pp. 247–271. DOI: 10.1007/978-1-4757-3437-9_12.

[133] Z.K. Nagy and R.D. Braatz. “Robust nonlinear model predictive control of batch processes”. In: *AIChE Journal* 49.7 (2003), pp. 1776–1786. DOI: 10.1002/aic.690490715.

[134] Tim Nikolayzik, Christof Büskens, and Matthias Gerdts. “Nonlinear large-scale Optimization with WORHP”. In: *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*. 2010, p. 9136.

[135] Jorge Nocedal. “Knitro: An integrated package for nonlinear optimization”. In: *Large-Scale Nonlinear Optimization*. Springer, 2006, pp. 35–60.

[136] Sebastian A. Nugroho, Vu Hoang, Maria Radosz, Shen Wang, and Ahmad F. Taha. “New Insights on One-Sided Lipschitz and Quadratically-Inner Bounded Nonlinear Dynamic Systems”. In: *Proceedings of the American Control Conference* 2020-July.July (2020), pp. 4558–4563. ISSN: 07431619. DOI: 10.23919/ACC45564.2020.9147812. arXiv: 2002.02361.

[137] S. H. Oh and R. Luus. “Use of orthogonal collocation method in optimal control problems”. In: *International Journal of Control* 26.5 (1977), pp. 657–673. DOI: 10.1080/00207177708922339. eprint: https://doi.org/

10.1080/00207177708922339. URL: https://doi.org/10.1080/00207177708922339.

[138] Rui Oliveira. “Combining first principles modelling and artificial neural networks: a general framework”. In: *Computers & Chemical Engineering* 28.5 (2004), pp. 755–766.

[139] Elton Pan, Panagiotis Petsagkourakis, Max Mowbray, Dongda Zhang, and Ehecatl Antonio del Rio-Chanona. “Constrained model-free reinforcement learning for process optimization”. In: *Computers and Chemical Engineering* 154.November 2020 (2021). ISSN: 00981354. DOI: 10.1016/j.compchemeng.2021.107462. arXiv: 2011.07925.

[140] T. Parisini and R. Zoppoli. “A receding-horizon regulator for nonlinear systems and a neural approximation”. In: *Automatica* 31.10 (1995), pp. 1443–1451. DOI: 10.1016/0005-1098(95)00044-W.

[141] R.S. Parker, F.J. Doyle, and N.A. Peppas. “A model-based algorithm for blood glucose control in type I diabetic patients”. In: *IEEE Transactions on Biomedical Engineering* 46.2 (1999), pp. 148–157. DOI: 10.1109/10.740877.

[142] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[143] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[144] A.A. Patwardhan and T.F. Edgar. “Nonlinear Model Predictive Control of a Packed Distillation Column”. In: *Industrial and Engineering Chemistry Research* 32.10 (1993). cited By 24, pp. 2345–2356. DOI: 10.1021/ie00022a018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0027676259&doi=10.1021\%2fie00022a018&partnerID=40&md5=55760bc31a8e31eed6f677e9a1209425.

[145] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journak of Machine Learning Research* 12 (2011), pp. 2825–2830.

[146] P. Petsagkourakis, I. O. Sandoval, E. Bradford, D. Zhang, and E. A.Rio Chanona Del. “Constrained Reinforcement Learning for Dynamic Optimization under Uncertainty”. In: *arXiv* (2020). ISSN: 23318422. arXiv: 2006.02750.

[147] P. Petsagkourakis, I.O. Sandoval, E. Bradford, D. Zhang, and E.A. del Rio-Chanona. “Reinforcement Learning for Batch-to-Batch Bioprocess Optimisation”. In: *Computer Aided Chemical Engineering* 46 (2019). cited By 2, pp. 919–924. DOI: 10.1016/B978-0-12-818634-3.50154-5. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85069718918&doi=10.1016\%2fB978-0-12-818634-3.50154-5&partnerID=40&md5=93872b9b408d6c68452de91814a8b28b.

[148] Panagiotis Petsagkourakis, Ilya Orson Sandoval, Eric Bradford, Dongda Zhang, and Ehecatl Antonio del Rio-Chanona. “Reinforcement learning for batch bioprocess optimization”. In: *Computers & Chemical Engineering* 133 (2020), p. 106649.

[149] J. Pohlodek, H. Alsmeier, B. Morabito, C. Schlauch, A. Savchenko, and R. Findeisen. *Stochastic Model Predictive Control Utilizing Bayesian Neural Networks*. 2023. arXiv: 2303.14519 [eess.SY].

[150] Johannes Pohlodek, Bruno Morabito, Christian Schlauch, Pablo Zometa, and Rolf Findeisen. *Flexible development and evaluation of machine-learning-supported optimal control and estimation methods via HILO-MPC*. 2022. arXiv: 2203.13671 [eess.SY].

[151] Steven Spielberg Pon Kumar, Aditya Tulsyan, Bhushan Gopaluni, and Philip Loewen. “A Deep Learning Architecture for Predictive Control”. In: *IFAC-PapersOnLine* 51.18 (2018). 10th IFAC Symposium on Advanced Control of Chemical Processes ADCHEM 2018, pp. 512–517. DOI: 10.1016/j.ifacol.2018.09.373.

[152] V. Prasad, M. Schley, L.P. Russo, and B. Wayne Bequette. “Product property and production rate control of styrene polymerization”. In: *Journal of Process Control* 12.3 (2002), pp. 353–372. DOI: 10.1016/S0959-1524(01)00044-0.

[153] Radoslaw Pytlak. *Numerical methods for optimal control problems with state constraints*. Springer Science & Business Media, 1999.

[154] Davide Martino Raimondo, Daniel Limon, Mircea Lazar, Lalo Magni, and Eduardo Fernández Camacho. “Min-max Model Predictive Control of Nonlinear Systems: A Unifying Overview on Stability”. In: *European Journal of Control* 15.1 (2009), pp. 5–21. ISSN: 0947-3580. DOI: https://doi.org/10.3166/ejc.15.5-21. URL: https://www.sciencedirect.com/science/article/pii/S0947358009707034.

[155] R. Rajamani and Y. M. Cho. “Existence and design of observers for nonlinear systems: Relation to distance to unobservability”. In: *International Journal of Control* 69.5 (1998), pp. 717–731. DOI: 10.1080/002071798222640. eprint: https://doi.org/10.1080/002071798222640. URL: https://doi.org/10.1080/002071798222640.

[156] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

[157] K. Yamuna Rani and Sachin C. Patwardhan. “Data-driven modeling and optimization of semibatch reactors using artificial neural networks”. In: *Industrial and Engineering Chemistry Research* 43.23 (2004), pp. 7539–7551. ISSN: 08885885. DOI: 10.1021/ie0305521.

[158] James B Rawlings. “Moving horizon estimation”. In: *Encyclopedia of Systems and Control* (2013), pp. 1–7.

[159] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI, 2017.

[160] J.B. Rawlings, D.Q. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. ISBN: 9780975937730. URL: https://books.google.de/books?id=MrJctAEACAAJ.

[161] M. Rick, J. Clemens, L. Sommer, A. Folkers, K. Schill, and C. Büskens. “Autonomous Driving Based on Nonlinear Model Predictive Control and Multi-Sensor Fusion”. In: vol. 52. 8. cited By 4. 2019, pp. 458–473. DOI: 10.1016/j.ifacol.2019.08.068. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85076257719&doi=10.1016\%2fj.ifacol.2019.08.068&partnerID=40&md5=3cc01fa2ddf34a9911252e3986b2c39b.

[162] M Risbeck, N Patel, and J Rawlings. *Nonlinear model predictive control tools for casadi*. 2015.

[163] U. Rosolia and F. Borrelli. “Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework”. In: *IEEE Transactions on Automatic Control* 63.7 (2018), pp. 1883–1896.

[164] Ugo Rosolia, Ashwin Carvalho, and Francesco Borrelli. “Autonomous racing using learning model predictive control”. In: *2017 American Control Conference (ACC)*. IEEE. 2017, pp. 5115–5120.

[165] Mario A. Rotea. “The generalized H2 control problem”. In: *Automatica* 29.2 (1993), pp. 373–385. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(93)90130-L. URL: https://www.sciencedirect.com/science/article/pii/000510989390130L.

[166] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0.

[167] Michael G. Safonov. “Origins of robust control: Early history and future speculations”. In: *Annual Reviews in Control* 36.2 (2012), pp. 173–181. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2012.09.001. URL: https://www.sciencedirect.com/science/article/pii/S1367578812000363.

[168] Artur M. Schweidtmann, Erik Esche, Asja Fischer, Marius Kloft, Jens-Uwe Repke, Sebastian Sager, and Alexander Mitsos. “Machine Learning in Chemical Engineering: A Perspective”. In: *Chemie Ingenieur Technik* 93.12 (2021), pp. 2029–2039. DOI: https://doi.org/10.1002/cite.202100083. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cite.202100083. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cite.202100083.

[169] Y. Shi, S. Kumar Singh, and L. Yang. “Classical Control Strategies Used in Recent Surgical Robots”. In: vol. 1922. 1. 2021. DOI: 10.1088/1742-6596/1922/1/012010.

[170] S. Shimamoto, K. Kobayashi, and Y. Yamashita. “Stochastic Model Predictive Control of Energy Management Systems with Human in the Loop”. In: 2020, pp. 60–61. DOI: 10.1109/GCCE50665.2020.9291914.

[171] Pramila P. Shinde and Seema Shah. “A Review of Machine Learning and Deep Learning Applications”. In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*. 2018, pp. 1–6. DOI: 10.1109/ICCUBEA.2018.8697857.

[172] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[173] Dan Simon. *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches*. ISBN 978-0-471-70858-2. Hoboken, New Jersey: John Wiley & Sons, 2006.

[174] Raffaele Soloperto, Matthias A. Müller, Sebastian Trimpe, and Frank Allgöwer. “Learning-Based Robust Model Predictive Control with State-Dependent Uncertainty”. In: *IFAC-PapersOnLine* 51.20 (2018), pp. 442–447. ISSN: 24058963. DOI: 10.1016/j.ifacol.2018.11.052.

[175] H.-S. Song and D. Ramkrishna. “Reduction of a set of elementary modes using yield analysis”. In: *Biotechnology and Bioengineering* 102.2 (2009), pp. 554–568.

[176] Eduardo D Sontag. “Input to state stability: Basic concepts and results”. In: *Nonlinear and optimal control theory*. Springer, 2008, pp. 163–220.

[177] B. Srinivasan, D. Bonvin, E. Visser, and S. Palanki. “Dynamic optimization of batch processes: II. Role of measurements in handling uncertainty”. In: *Computers & Chemical Engineering* 27.1 (2003), pp. 27–44. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(02)00117-5. URL: https://www.sciencedirect.com/science/article/pii/S0098135402001175.

[178] B Srinivasan and Dominique Bonvin. “Controllability and stability of repetitive batch processes”. In: *Journal of Process Control* 17.3 (2007), pp. 285–295.

[179] K.S. Stadler, J. Poland, and E. Gallestey. “Model predictive control of a rotary cement kiln”. In: *Control Engineering Practice* 19.1 (2011), pp. 1–9. DOI: 10.1016/j.conengprac.2010.08.004.

[180] Robert F Stengel. *Stochastic optimal control: theory and application*. John Wiley & Sons, Inc., 1986.

[181] “Stochastic Processes and Filtering Theory”. en. In: *Mathematics in Science and Engineering*. Vol. 64. Elsevier, 1970, pp. 162–193. ISBN: 978-0-12-381550-7. DOI: 10.1016/S0076-5392(09)60375-1. URL: https://linkinghub.elsevier.com/retrieve/pii/S0076539209603751 (visited on 05/07/2023).

[182] Qinglin Su, Sudarshan Ganesh, Mariana Moreno, Yasasvi Bommireddy, Marcial Gonzalez, Gintaras V. Reklaitis, and Zoltan K. Nagy. “A perspective on Quality-by-Control (QbC) in pharmaceutical continuous manufacturing”. In: *Computers & Chemical Engineering* 125 (2019), pp. 216–231. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2019.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0098135418309116.

[183] Yuelong Su and Fengqin Yu. “Data mining applications for finding golden batch benchmarks and optimizing batch process control”. In: *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. 2016, pp. 1058–1063. DOI: 10.1109/WCICA.2016.7578815.

[184] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel. “Learning from the hindsight plan — Episodic MPC improvement”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 336–343.

[185] Andrew Taylor, Andrew Singletary, Yisong Yue, and Aaron Ames. “Learning for safety-critical control with control barrier functions”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 708–717.

[186] Ana P Teixeira, João J Clemente, António E Cunha, Manuel JT Carrondo, and Rui Oliveira. “Bioprocess iterative batch-to-batch optimization based on hybrid parametric/nonparametric models”. In: *Biotechnology progress* 22.1 (2006), pp. 247–258.

[187] Peter Terwiesch, Mukul Agarwal, and David W.T. Rippin. “Batch unit optimization with imperfect modelling: a survey”. In: *Journal of Process Control* 4.4 (1994), pp. 238–258. ISSN: 0959-1524. DOI: https://doi.org/10.1016/0959-1524(94)80045-6. URL: https://www.sciencedirect.com/science/article/pii/0959152494800456.

[188] Arun Tholudur and W Fred Ramirez. “Optimization of fed-batch bioreactors using neural network parameter function models”. In: *Biotechnology Progress* 12.3 (1996), pp. 302–309.

[189] Ivayla Vatcheva, Hidde de Jong, Olivier Bernard, and Nicolaas J.I. Mars. “Experiment selection for the discrimination of semi-quantitative models of dynamical systems”. In: *Artificial Intelligence* 170.4 (2006), pp. 472–506. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2005.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0004370205002092.

[190] S. Vazquez, J.I. Leon, L.G. Franquelo, J. Rodriguez, H.A. Young, A. Marquez, and P. Zanchetta. "Model predictive control: A review of its applications in power electronics". In: *IEEE Industrial Electronics Magazine* 8.1 (2014). cited By 661, pp. 16–31. DOI: 10.1109/MIE.2013.2290138.

[191] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Niels van Duijkeren, Andrea Zanelli, Rien Quirynen, and Moritz Diehl. "Towards a modular software package for embedded optimization". In: *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC)*. 2018.

[192] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. *acados: a modular open-source framework for fast embedded optimal control*. 2020. arXiv: 1910.13753 [math.OC].

[193] Moritz Von Stosch, Rui Oliveira, Joana Peres, and Sebastião Feyo de Azevedo. "Hybrid semi-parametric modeling in process systems engineering: Past, present and future". In: *Computers & Chemical Engineering* 60 (2014), pp. 86–101.

[194] 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)*. 2018, pp. 1691–1697. DOI: 10.23919/ECC.2018.8550288.

[195] Kim P Wabersich and Melanie N Zeilinger. "Linear model predictive safety certification for learning-based control". In: *2018 IEEE Conference on Decision and Control (CDC)*. IEEE. 2018, pp. 7130–7135.

[196] Kim P Wabersich and Melanie N Zeilinger. "Safe exploration of nonlinear dynamical systems: A predictive safety filter for reinforcement learning". In: *arXiv preprint arXiv:1812.05506* (2018).

[197] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[198] E.a. A Wan and Rudolph Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Technology* v (2000), pp. 153–158. ISSN: 15270297. DOI: 10.1109/ASSPCC.2000.882463. URL: https://www.seas.harvard.edu/courses/cs281/papers/unscented.pdfhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=882463.

[199] Peter Wieland and Frank Allgöwer. "Constructive safety using control barrier functions". In: *IFAC Proceedings Volumes* 40.12 (2007), pp. 462–467.

[200] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[201] G. Williams, P. Drews, B. Goldfain, J.M. Rehg, and I.A. Theodorou. “Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving”. In: *IEEE Transactions on Robotics* 34.6 (2018). cited By 29, pp. 1603–1622. DOI: 10.1109/TRO.2018.2865891. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85058870968&doi=10.1109\%2fTRO.2018.2865891&partnerID=40&md5=7a4f71629dfea0ba61753044e1b5f481.

[202] Zhihua Xiong and Jie Zhang. “Improved operation of a batch polymerisation reactor through batch-to-batch iterative optimisation”. In: *IFAC Proceedings Volumes (IFAC-PapersOnline)* 37.1 (2004), pp. 945–950. ISSN: 14746670. DOI: 10.1016/s1474-6670(17)38856-0. URL: http://dx.doi.org/10.1016/S1474-6670(17)38856-0.

[203] J.-X. Xu. “A survey on iterative learning control for nonlinear systems”. In: *International Journal of Control* 84.7 (2011). cited By 288, pp. 1275–1294. DOI: 10.1080/00207179.2011.574236.

[204] Jian-Xin Xu and Ying Tan. *Linear and nonlinear iterative learning control*. Vol. 291. Springer, 2003.

[205] Xinghai Xu, Huimin Xie, and Jia Shi. “Iterative Learning Control (ILC) Guided Reinforcement Learning Control (RLC) Scheme for Batch Processes”. In: *Proceedings of 2020 IEEE 9th Data Driven Control and Learning Systems Conference, DDCLS 2020* Ilc (2020), pp. 241–246. DOI: 10.1109/DDCLS49620.2020.9275065.

[206] S. Yu, C. Böhm, H. Chen, and F. Allgöwer. “Robust model predictive control with disturbance invariant sets”. In: *Proc. Amer. Cont. Conf. (ACC)*. 2010, pp. 6262–6267. DOI: 10.1109/ACC.2010.5531520.

[207] Shuyou Yu, Christoph Maier, Hong Chen, and Frank Allgöwer. “Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems”. In: *Systems & Control Letters* 62.2 (2013), pp. 194–200. ISSN: 0167-6911.

[208] M. Zhai, Y. Liu, T. Zhang, and Y. Zhang. “Robust model predictive control for energy management of isolated microgrids”. In: vol. 2017-December. 2018, pp. 2049–2053. DOI: 10.1109/IEEM.2017.8290252.

[209] Dongda Zhang, Ehecatl Antonio Del Rio-Chanona, Panagiotis Petsagkourakis, and Jonathan Wagner. “Hybrid physics-based and data-driven modeling for bioprocess online simulation and optimization”. In: *Biotechnology and Bioengineering* 116.11 (2019), pp. 2919–2930. ISSN: 10970290. DOI: 10.1002/bit.27120. URL: http://dx.doi.org/10.1002/bit.27120.

[210] Jie Zhang. “A neural network-based strategy for the integrated batch-to-batch control and within-batch control of batch processes”. In: *Transactions of the Institute of Measurement & Control* 27.5 (2005), pp. 391–410. ISSN: 01423312. DOI: 10.1191/0142331205tm156oa.

[211] Jie Zhang. “Batch-to-batch optimal control of a batch polymerisation process based on stacked neural network models”. In: *Chemical Engineering Science* 63.5 (2008), pp. 1273–1281. ISSN: 00092509. DOI: 10.1016/j.ces.2007.07.047.

[212] J. Zhu, Y. Liu, H. Lei, and T. Zhang. “A Robust and Model Predictive Control Based Energy Management Scheme for Grid-Connected Microgrids”. In: 2018. DOI: 10.1109/EI2.2018.8582556.

[213] T Zieger, A Savchenko, T Oehlschlägel, and R Findeisen. “Towards safe neural network supported model predictive control”. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 5246–5251.

[214] Tim Zieger, Hoang Hai Nguyen, Erik Schulz, Thimo Oehlschlägel, and Rolf Findeisen. “Non-diverging neural networks supported tube model predictive control”. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. 2022, pp. 3066–3073. DOI: 10.1109/CDC51059.2022.9993089.

[215] Pablo Zometa, Markus Kögel, Timm Faulwasser, and Rolf Findeisen. “Implementation aspects of model predictive control for embedded systems”. In: *2012 American Control Conference (ACC)*. IEEE. 2012, pp. 1205–1210.

[216] Pablo Zometa, Markus Kögel, and Rolf Findeisen. “μAO-MPC: A free code generation tool for embedded real-time linear model predictive control”. In: *2013 American Control Conference*. IEEE. 2013, pp. 5320–5325.

[217] C. Zou, X. Hu, Z. Wei, T. Wik, and B. Egardt. “Electrochemical Estimation and Control for Lithium-Ion Battery Health-Aware Fast Charging”. In: *IEEE Transactions on Industrial Electronics* 65.8 (2018), pp. 6635–6645. DOI: 10.1109/TIE.2017.2772154.