IMT School for Advanced Studies, Lucca
Lucca, Italy

Machine learning methods for control, identification, and estimation

PhD Program in Systems Science

Track in Computer Science and Systems Engineering

XXXIV Cycle

By

Daniele Masti

2021

The dissertation of Daniele Masti is approved.

PhD Program Coordinator: Prof. Rocco De Nicola, IMT School for Advanced Studies Lucca

Advisor: Prof. Alberto Bemporad, IMT School for Advanced Studies Lucca

The dissertation of Daniele Masti has been reviewed by:

Dr. Dario Piga, Dalle Molle Institute for Artificial Intelligence

Dr. Jan Drgona, Pacific Northwest National Laboratory

IMT School for Advanced Studies Lucca
2021

To my family

Contents

List of Figures

List of Tables

Acknowledgements

I want to thank my advisor, Prof. Alberto Bemporad. Without his precious insights, patience, and guidance, neither this thesis nor my MS thesis would have become a reality. It has been a privilege and an honor for me to be able to work with him. I want to thank Prof. Mario Zanon. While we often clashed, his counseling helped me profoundly in my journey. I am also thankful to Dr. Emilio Incerto, Dr. Yuriy Zacchia Lun, Dr. Valentina Breschi, and Laura Ferrarotti. Our time together has been precious, and your advice has always been welcome. I am deeply grateful to all the people of IMT who made me feel at home even in the darkest moments.

Finally, I would like to thank my family and all my friends. Without their encouragement, nothing of this would have happened.

Vita

October 25, 1993	Born, Siena, Italy
2012-2015	BSc in "Ingeneria informatica e dell'Informazione" Final mark: 110/110 cum laude Università degli Studi di Siena, Italy
2016-2018	MSc in "Ingeneria elettrica e dell'Automazione" Final mark: 110/110 cum laude Università degli Studi di Firenze, Italy
2018-2021	PhD Program in Computer Science and Systems Engineering IMT School for Advanced Studies Lucca, Italy

Publications

1. B. Allotta, L. Pugi, M. Montagni, A. Corrieri, D. Masti and L. Vanni, "Development of an innovative and sustainable sail-drone," 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe), 2017, pp. 1-6.
2. D. Masti and A. Bemporad, "Learning nonlinear state-space models using deep autoencoders," in Proceedings of 2018 57th Conference on Decision and Control (CDC), Miami Beach, FL, USA, 2018, pp. 3862–3867.
3. D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in Proceedings of 2019 European Control Conference, Naples, Italy, 2019, pp. 1494–1499.
4. D. Masti, D. Bernardini and A. Bemporad, "Learning virtual sensors for estimating the scheduling signal of parameter-varying systems," 2019 27th Mediterranean Conference on Control and Automation (MED), 2019, pp. 232-237
5. D. Masti, F. Smarra, A. D'Innocenzo, and A. Bemporad, "Learning affine predictors for MPC of nonlinear systems via artificial neural networks," in Proceedings of the 21st IFAC World Congress, 2020.
6. D. Masti, T. Pippia, A. Bemporad, and B. De Schutter, "Learning approximate semi-explicit hybrid MPC with an application to microgrids," in Proceedings of the 21st IFAC World Congress, 2020.
7. D. Masti, V. Breschi, S. Formentin, and A. Bemporad, "Direct data-driven design of neural reference governors," in Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC), Jeju Island, Republic of Korea, 2020, pp. 4955–4960.
8. V. Breschi, D. Masti, S. Formentin, and A. Bemporad, "NAW-NET: Neural anti-windup control for saturated nonlinear systems," in Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC), Jeju Island, Republic of Korea, 2020, pp. 3335–3340.
9. D. Masti and A. Bemporad, "Learning nonlinear state–space models using autoencoders," Automatica, vol. 129, p. 109666, 2021.
10. D. Masti, D. Bernardini, and A. Bemporad, "A machine-learning approach to synthesize virtual sensors for parameter-varying systems," European Journal of Control, vol. 61, pp. 40-49, 2021.

11. D. Masti, M. Zanon and A. Bemporad, “Tuning LQR controllers: a sensitivity-based approach,” in IEEE Control Systems Letters, vol. 6, pp. 932-937, 2022.

Presentations

1. D. Masti and A. Bemporad, "Learning nonlinear state-space models using deep autoencoders," in Proceedings of 2018 57th Conference on Decision and Control (CDC), Miami Beach, FL, USA, 2018, pp. 3862–3867
2. D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in Proceedings of 2019 European Control Conference, Naples, Italy, 2019, pp. 1494–1499.
3. D. Masti, V. Breschi, S. Formentin, and A. Bemporad, "Direct data-driven design of neural reference governors," in Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC), Jeju Island, Republic of Korea, 2020, pp. 4955–4960

Abstract

Over the last decades, the landscape of control theory and system identification has changed significantly in response to the new challenges arising from the industry. This is not surprising: classical model-based techniques are not suitable to handle real-world applications for which it is often too expensive to derive even an approximate model using first principles. Data-driven approaches represent a solution to such an issue. Thanks to the ever-increasing availability of a large quantity of data, they have quickly become central topics within the control theory community.

This thesis collects some results regarding using machine learning approaches to answer some open questions in control theory by formulating novel techniques and lessening some undesirable aspects of existing methods. We first present a system identification approach based on deep learning to learn state-space models for nonlinear systems. We then propose a data-driven virtual sensor synthesis approach, inspired by the Multiple Model Adaptive Estimation framework, for reconstructing normally unmeasurable quantities such as scheduling parameters in parameter-varying systems. Three data-driven control approaches, two of which are based on the well-known Virtual Reference Feedback Tuning framework, are finally presented to synthesize constrained controllers for unknown nonlinear dynamical systems from the data without identifying first a model of the plant. Tuning guidelines for the proposed methods are also provided.

Chapter 1

Introduction

Over the last decades, the landscape of control theory and system identification has changed significantly in response to the new challenges arising from the industry. This is not surprising: classical model-based approaches are not suitable to handle real-world scenarios for which it is often simply too expensive—if at all possible— to derive even an approximate model of the plant we want to control. This has become especially evident in the recent years, in which such kind of applications has become more and more common. Data-driven approaches represent a natural solution to such an issue. Indeed, thanks also to the ever-increasing availability of a large quantity of experimental and simulated data, they have quickly become one of the central topics within the control theory community both in academia and in the industry.

While such ever-growing enthusiasm has sparked renewed attention also in traditionally "data-centric" subfields of control theory [1], [2], it is undeniable that much interest nowadays revolves around methods that heavily draw from other data-oriented fields such as machine learning [3] and global/black-box optimization. Examples of such phenomena are, to give an example, the numerous system identification and adaptive control techniques based on neural networks (see, for instance, [4], [5]), which development has also been enabled by the large availability of software tools developed by the machine learning com-

munity [6]–[8], or the many recent contributions relying on black-box optimization to tune controllers [9]. Another example of such contamination is the abundance of works using reinforcement learning [10], [11], which has quickly become a central approach for designing new control laws for possibly unknown plants also within the control community.

Most contributions, however, often describe completely novel classes of models and controllers. This is problematic because it does not allow one to reuse existing literature to assess the properties of the controlled/identified system (one above all: internal stability). For this reason, a strong interest in both proving the properties of such new approaches, but also into improving the weakest points of traditional techniques in a way that does not compromise their already proven desirable characteristics has grown within the community. An example in this sense is the use of Reinforcement Learning based on Model predictive control [12] (which elegantly solves the ever-challenging problem to tune such a controller while maintaining strong stability guarantees), and the works that lighten the computational burden associated with (hybrid) predictive controllers [13]–[15].

1.1 Thesis outline

This thesis collects some results on using machine learning techniques to answer some open questions in control, identification, and estimation both by formulating totally novel ideas and by lessening some undesirable properties of some existing techniques. In more details, we will present a neural-network-based system identification approach, a data-driven technique to design virtual sensors for partially measurable systems, and three connected contributions in the area of data-driven control.

The outline of the chapters, which are meant to be as self-contained as possible, is the following:

- Chapter 2 proposes a methodology for identifying nonlinear state-space models from input/output data using machine-learning

techniques based on autoencoders and neural networks. The framework simultaneously identifies the nonlinear output and state-update maps of the model. After formulating the approach and providing guidelines for tuning the related hyper-parameters (including the model order), we show its capability in fitting nonlinear models on different nonlinear system identification benchmarks. Performance is assessed in terms of open-loop prediction on test data and by controlling the system via an especially crafted nonlinear model predictive control (MPC) scheme based on the identified nonlinear state-space model.

The content of this Chapter and this abstract are reprinted from:

D. Masti and A. Bemporad, "Learning nonlinear state–space models using autoencoders," Automatica, vol. 129, p. 109 666, 2021

- Chapter 3 proposes a model-free approach to synthesize virtual sensors to estimate dynamical quantities that are unmeasurable at runtime but are available for design purposes on test benches. After collecting a dataset of measurements of such quantities, together with other variables that are also available during online operations, the virtual sensor is obtained using machine learning techniques by training a predictor whose inputs are the measured variables and the features extracted by a bank of linear observers fed with the same measures. The approach is applicable to infer the value of quantities such as physical states and other time-varying parameters that affect the dynamics of the system. The proposed virtual sensor architecture - whose structure can be related to the Multiple Model Adaptive Estimation framework - is conceived to keep computational and memory requirements as low as possible. The effectiveness of the approach is shown in different numerical examples, involving the estimation of the scheduling parameter of a nonlinear parameter-varying system, the reconstruction of the mode of a switching linear system, and the estimation of the state of charge (SoC) of a lithium-ion battery.

The content of this Chapter and this abstract are reprinted from:

D. Masti, D. Bernardini, and A. Bemporad, "A machine-learning approach to synthesize virtual sensors for parameter-varying systems," European Journal of Control, vol. 61, pp. 40–49, 2021

A preliminary version of the same content can be also found in [18].

- In Chapter 4 , we exploit the fact that MPC controller for nonlinear systems can often be designed using linear time-varying (LTV) MPC formulations in which, at each sampling step, a quadratic programming (QP) problem based on linear predictions is constructed and solved at runtime. To reduce the associated computation burden, we explore and compare two methodologies for learning the entire output prediction over the MPC horizon as a nonlinear function of the current state but affine with respect to the sequence of future control moves to be optimized. Such a learning process is based on input/output data collected from the process to be controlled. The approach is assessed in a simulation example and compared to other similar techniques proposed in the literature, showing that it provides accurate predictions of the future evolution of the process and good closed-loop performance of the resulting MPC controller. Guidelines for tuning the proposed method to achieve a desirable memory occupancy/quality of fit trade-off are also given.

  The material presented in this Chapter and this abstract are reprinted from:

  D. Masti, F. Smarra, A. D'Innocenzo, and A. Bemporad, "Learning affine predictors for MPC of nonlinear systems via artificial neural networks," in Proceedings of the 21st IFAC World Congress, 2020.

- Chapter 5 further develops this idea and proposes a direct data-driven approach to synthesize model reference controllers for constrained nonlinear dynamical systems. To this aim, we employ a hi-

erarchical structure composed of a receding-horizon reference governor and a data-driven low-level controller. Unlike most existing approaches, we design the two blocks exploiting the fact that the inner controller will never be used alone. The performance of the proposed method is assessed using two simulation examples involving the control of two highly nonlinear benchmark systems.

The content of the Chapter and this abstract are from:

- One major issue in industrial control applications is how to handle input constraints due to the physical limitations of the actuators using as little computational power as possible. We develop an idea to answer this question in Chapter 6, where we show an off-line strategy to learn a neural anti-windup control scheme (NAW-NET) from a set of open-loop data collected from an unknown nonlinear process. Such a scheme includes a feedback controller and an anti-windup compensator and it is trained to reproduce the desired closed-loop behavior while simultaneously accounting for actuator limits. We illustrate its effectiveness in a simulation example involving the control of a Hammerstein-Wiener process with saturated inputs.

  The content of this Chapter and this abstract are from:

Concluding remarks on the presented approaches and open problems which will be the base for future research are finally collected in Chapter 7.

Not included in this thesis but co-authored by the author while within the PhD program, there are also the paper following papers: [13], [22]–[24].

Chapter 2

Learning nonlinear state-space models using autoencoders

2.1 Introduction

Nonlinear system identification has gained increasing popularity in recent years [25], [26], also due to massive advances in machine-learning methods for nonlinear function regression. Such methods have been employed with high success for extending classical linear techniques to nonlinear systems, such as for the estimation of neural autoregressive models with exogenous inputs (NARX) [25] and of reproducing kernel Hilbert space (RKHS) models [26], as well as for piecewise-affine regression [27], and for developing novel approaches based on long short-term memory (LSTM) neural networks [28].

Most of the aforementioned techniques, however, identify nonlinear models in *input/output* form, without an explicit definition of a (minimal) Markovian state. On the other hand, *state-space* models are the basis for most modern control design techniques, such as nonlinear control, model predictive control (MPC), as well as for noise filtering and smoothing, such as extended Kalman filtering (EKF).

2.1.1 Machine-learning methods for the identification of state-space models

The idea of applying machine-learning approaches to identify state-space representations of a dynamical system from input/output data has been widely explored in the literature. For example, we mention here the classical dynamic mode decomposition (DMD) and refer the reader to the review in [29]. Learning a state-space model typically requires a nonlinear transformation of a vector of past input/output samples to a state vector. One drawback of many machine-learning methods performing such a dimensionality reduction is that the resulting state-space model (a.k.a. "latent representation") may result in non-regular and stiff mappings, therefore creating issues when the model is used to design a controller and/or an observer, for instance when the linearization of the model is used. Although expections exist [30], most of the proposed solutions address this issue by relying on variations of two families of approaches [31]: (i) imposing a "regular geometry" on the learned latent space; (ii) envision learning schemes in which the capability of the latent representation to predict future output values is directly taken into account during the learning phase.

Among the first family, many works are inspired by the well known variational autoencoders [32]. Here, the latent representation is learned so that the distribution of the resulting states is a prescribed one, usually a Gaussian one. Different types of VAE have been widely used both for system identification [33]–[37] and in reinforcement learning [38], [39].

The contributions related to the second family of approaches are often extensions of the DMD idea. We mention here papers based on the Koopman formalism [40]–[42] and the contributions [43], [44], which use autoencoders in conjunction with the Koopman operator for learning representations of autonomous systems. Closely related to the DMD idea is also the so-called "SINDy" framework [45]. In these frameworks, the original input/output signals are lifted to a possibly higher-dimensional space, in which the dynamic evolution of the system can be modeled in linear time-invariant (LTI) form or, more rarely, in bilinear form. A char-

acteristic of the approach is that the resulting state-space dimension can be larger than that of the original vector of past input/output samples.

Some techniques combine methods from both families. For instance, in [46], [47] recurrent neural networks are trained to predict the evolution of the state-update maps that govern a time-varying time series, while in [48] a variational recurrent neural network architecture is envisioned.

2.1.2 Contribution

We propose a methodology that uses artificial neural networks (ANNs), and in particular autoencoders (AEs) [49], to learn a nonlinear model in state-space from a given input/output dataset. The main idea is the following: we train an AE that reproduces a collection of output signals from a collection of input and output signals and take the central layer of the AE as the state vector. Such a dimensionality reduction problem is solved jointly with problem of learning the nonlinear state-update function (parameterized by a deep neural network) that maps the values of such a state into its next values.

Contrarily to the contributions [40], [44], [50], we do not require the dynamics to be linear in the learned latent space. On the other hand, to ease the design of controllers and state observers, we also consider a quasi linear parameter-varying (LPV) formulation of the model, in which each of the coefficients of the state-update matrices is the output of an ANN.

In contrast with the works [33]–[35], [48], we do not rely on variational inference arguments, as we do not impose specific structures on the learned latent space. Our approach brings two advantages: (i) it avoids making an assumption about the distribution to impose; (ii) as we will show in Section 2.4.3, it enables the use of classical shrinkage operators to tune some of the hyper-parameters of the method, that would be instead hard to adopt within the variational framework.

To improve both the accuracy of the resulting models and the numerical stability of the approach, we also use a fitting criterion based on multi-step predictions. The main hyper-parameter of the approach is the

order of the state-space model, that must be chosen (as typically in system identification methods) to obtain a tradeoff between model accuracy in reproducing test data and model complexity. We provide a heuristic approach based on group sparsification methods to help tuning the number of states to include in the nonlinear model and the number of past input and outputs the autoencoder consumes.

Preliminary ideas and results related to the contents of this chapter were presented in [51], where we proposed AEs to extract a compressed representation of I/O data to be used as state representation for a classical one-step-ahead Prediction Error Method (PEM) approach.

The method we present is also related to the approaches presented in some very recent contributions [52]–[54], which also develop ideas related to the preliminary work presented in [51].

This Chapter is organized as follows. In Section 2.2 we formulate the nonlinear identification problem we want to solve and present a solution method based on autoencoders in Section 2.3. After detailing the learning algorithm in Section 2.4 and discussing the use of the identified model for state estimation and control in Section 2.5, we report results in Section 2.6 based on several nonlinear benchmark problems. Finally, we draw conclusions in Section 2.7.

2.2 Nonlinear identification problem

We are given a training dataset of input/output samples $Z = \{u_0, y_0, \ldots, u_N, y_N\}$ collected from a dynamical system, where $u_k \in \mathbb{R}^{n_u}$ is the vector of exogenous inputs and $y_k \in \mathbb{R}^{n_y}$ the vector of measured outputs. Our goal is to identify a dynamical model of the system in the following state-space form

$$\begin{cases} x_{k+1} &= f(x_k, u_k) \\ \hat{y}_k &= g(x_k) \end{cases} \tag{2.1}$$

with $x \in \mathbb{R}^{n_x}$, that, starting from an appropriate condition x_{k_0} and excited by the same inputs $u_{k_0}, \ldots, u_{N-1}$, produces a output signal $\hat{y}_k$ that is as close as possible to the one y_k recorded on the system. Although we assume that the collected input and output signals may be affected by

measurement noise, we do not make any particular assumption about the properties of such noise.

Given a number of past outputs $n_a \geq 1$, of past inputs $n_b \geq 1$, and a desired state dimension $n_x \geq 1$, the problem can be recast to the problem of finding a triplet of maps e, f, g, $e : \mathbb{R}^{n_I} \to \mathbb{R}^{n_x}$, $n_I \triangleq n_a n_y + n_b n_u$, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ that solves the following optimization problem:

$$\min_{e,f,g} \mathcal{L}(e, f, g, Z) \tag{2.2a}$$

where

$$\begin{aligned} \mathcal{L}(e, f, g, Z) \quad &= \sum_{k=k_0}^{N} L(\hat{y}_k, y_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\ & \hat{y}_k = g(x_k), \quad k = k_0, \dots, N \\ & x_{k_0} = e(I_{k_0 - 1}) \end{aligned} \tag{2.2b}$$

In (2.2), $L : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to [0, +\infty)$ is a suitable loss function that penalizes the discrepancy between the predicted and the measured output, and I_k is the following information vector

$$I_k = [y_k' \ \dots \ y_{k-n_a+1}' \ u_k' \ \dots \ u_{k-n_b+1}']' \tag{2.3}$$

where $k_0 \triangleq \max\{n_a, n_b\}$. In (2.2), e is the dimensionality-reduction mapping from the vector I_k of past inputs and outputs to the state vector x_k, whose role will be further explained in the next sections.

In general problem (2.2) has infinitely many solutions. For example, if the data were generated by a linear system of order n, for any dimension $n_x \geq n$ all the infinitely many (possibly non-minimal) state-space realizations leading to the same transfer function would be equally optimal. The problem of recognizing the smallest state-dimension n_x that provides an acceptable mismatch between the predictions $\hat{y}_k$ and the measured outputs y_k is of main interest and has been widely explored in the literature [55], [56]. A similar problem of data compression is also widely studied in the machine learning literature in the context of *feature extraction* [57]. There, the goal is to reduce the dimension of the input space by identifying a nonlinear function that projects the original

(large) input space into a (smaller dimensional) feature space, without losing significant information content.

In the following, we solve problem (2.2) by parameterizing the maps e, f, g as ANNs, due to their universal approximation properties [58] and efficient numerical packages available for training them.

2.3 State selection via autoencoders

The idea behind an autoencoder is to train an ANN to reproduce the identity mapping from a certain information vector $I_k \in \mathbb{R}^{n_I}$ to I_k itself, under the topological constraint that one of its hidden layers contains $n_x < n_I$ neurons. Such a constraint forces the network to learn a description of I_k that lives in the lower-dimensional space $\mathbb{R}^{n_x}$ without losing information. The smaller the fitting error between I_k and the reconstructed I_k, the less information is lost when passing through the network across the hidden "bottleneck" layer. As a result, when excited by an input value I_k, the corresponding value $x_k \in \mathbb{R}^{n_x}$, taken by the neurons of the bottleneck layer represents the desired lower-dimensional vector concentrating the information contained in I_k. This approach has been shown to be successful in a large variety of applications. For completely linear networks, it has been shown in [59] that it has a strong relation with the standard principal component analysis (PCA) technique.

2.3.1 Partial predictive autoencoders

Given the dataset Z of input/output samples, applying a standard autoencoder to compress the information vector I_k defined by (2.3) into a reduced-order vector $x_k \in \mathbb{R}^{n_x}$ would not be optimal to learn a state representation for two reasons: (i) it would treat the samples I_k as independent, missing the fact that consecutive samples I_k share common (time-shifted) components, and therefore fail in capturing the capability of predicting the next output y_{k+1}, and (ii) it would be redundant, as we are not really interested in reproducing the input signals u_{k-i+1}, $i = 1, \ldots, n_b$, that are components of I_k. Therefore, we introduce here

a *partial predictive autoencoder* (PPE) that maps I_{k-1} (i.e., the information available up to time $k-1$) into the following vector of outputs

$$O_k = [y'_k \ \dots \ y'_{k-m}]' \tag{2.4}$$

with $0 \leq m \leq n_a$. By fitting an ANN with a hidden layer of size n_x, $n_x \leq n_a n_y + n_b n_u$, that tries to predict O_k given I_{k-1}, we obtain an intermediate compressed representation $x_k \in \mathbb{R}^{n_x}$. Such a vector x_k can be treated as a model state, as it captures the information required to predict y_k from I_{k-1}, and even filter y_{k-1} (if $m \geq 1$) and smooth past outputs (if $m > 1$).

We note here that we could make x_k depend on y_k too, but we do not consider such a case in this work.

The PPE amounts to the cascade of two different ANNs: (i) an encoding function $e : \mathbb{R}^{n_I} \to \mathbb{R}^{n_x}$ representing the transformation from I_{k-1} (past inputs and outputs) to x_k (state vector), (ii) a decoding mapping $d : \mathbb{R}^{n_x} \to \mathbb{R}^{m n_y}$ from x_k to O_k, whose first n_y components constitute the desired output function $g : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$.

2.4 Model learning

Having defined a structure to map I_{k-1} into x_k, we also need a structure to fit a function $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ mapping x_k and u_k into the next state x_{k+1}. A first approach would be to fit the PPE described above to get functions e and d, compute the set of states $x_k = e(I_{k-1})$, $k = \max(n_a, n_b) + 1, \dots, N$, and then fit a model f mapping (x_k, u_k) to x_{k+1}. We propose instead a better method that learns e, d, and f simultaneously. We define a multi-objective learning problem whose solution is a set of sub-networks implementing the state-update and output functions of the desired state space model. The corresponding structure is schematically depicted in Figure 1, in which we use two PPEs that share exactly the same weights (to be determined), one fed by I_{k-1} and the other by I_k. The goal is to reproduce, respectively, O_k and O_{k+1}. In this way, the generated state x_k in the first AE and x_{k+1} in the second AE will be coherent. A third ANN must be trained to map u_k and x_k into the shifted state x_{k+1}, therefore getting the state-update mapping f.

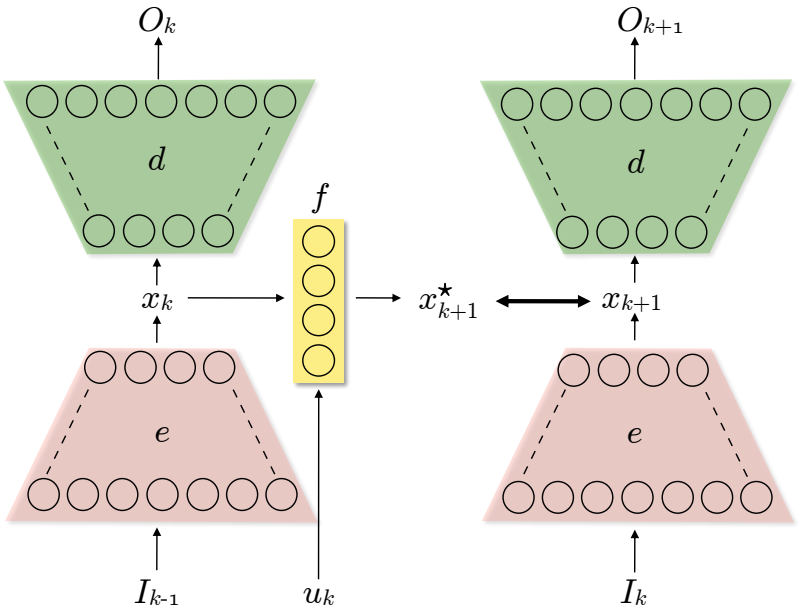


Figure 1: Schematic representation of the computational graph of the proposed nonlinear model structure

The overall training problem described above is formulated as the following optimization problem

$$\begin{aligned}
\min_{e,f,d} \quad & \sum_{k=k_0}^{N-1} \alpha \left(L_1(\hat{O}_k, O_k) + L_1(\hat{O}_{k+1}, O_{k+1}) \right) \\
& + \beta L_2(x^\star_{k+1}, x_{k+1}) + \gamma L_3(O_{k+1}, O^\star_{k+1}) \\
\text{s.t.} \quad & x_k = e(I_{k-1}),\ k = k_0, \ldots, N \\
& x^\star_{k+1} = f(x_k, u_k),\ k = k_0, \ldots, N-1 \\
& \hat{O}_k = d(x_k),\ k = k_0, \ldots, N \\
& O^\star_k = d(x^\star_k),\ k = k_0+1, \ldots, N
\end{aligned} \tag{2.5}$$

where L_i are loss functions, $\alpha, \beta, \gamma \geq 0$ are scalar weights, O_k is defined by (2.4), and I_k by (2.3), and $d : \mathbb{R}^{n_x} \to \mathbb{R}^{(m+1)n_y}$ is the decoder part of the PPE. Note that the output function g in (2.1)–(2.2) is retrieved from d by taking the components corresponding to y_k.

In (2.5), the loss function L_1 extends the original loss L in (2.2b), for instance $L_1(\hat{O}_k, O_k) = \sum_{j=k-m}^{k} L(\hat{y}_j, y_j)$. The loss L_2 can be seen as a relaxation of the state update equation $x_{k+1} = f(x_k, u_k)$ in (2.2b). The loss L_3 attempts avoiding that the error introduced by the bridge function f gets amplified by the nonlinear decoder d and results in a large deviation, on between the predicted outputs $O^{\star}_{k+1}$ and the measured outputs O_{k+1}. While clearly (2.5) may not solve the original problem (2.2) exactly, it attempts to provide a good sub-optimal solution to it.

Minimizing L_3 and L_2 in (2.5) is the objective with the most priority, as it captures the one-step ahead properties of the model. The "vertical" objective related to L_1 is instead only ancillary and serves to guide the process of learning the correct bridge/decoder pair. This suggests that a good approach for better solving the posed learning problem (2.2) is to start the training procedure with small values of β, γ and a high value of α, and then use the solution as the initial guess of another instance of (2.5) with a smaller value of α, possibly reiterating the procedure multiple times for decreasing values of α.

In the rest of the Chapter we will consider that L_{1-3} are L_{MAE} loss functions [60][1].

Besides deciding the *topology* and *activation functions* of the ANNs employed in the two identical PPEs, and the *learning algorithms* employed in optimizing their weights and bias terms given the available training dataset, several tuning hyper-parameters are involved in the proposed nonlinear system identification method described above. These are summarized in Table 1.

2.4.1 Multiple-step ahead fitting procedure

Penalizing the *one-step ahead* prediction errors in (2.5) does not guarantee that the identified model will provide good *open-loop* predictions. An approach to solve this issue would be to resort to a backpropagation through time (BPTT) learning scheme [61], in which the initial es-

[1]For a given dataset $\{Q_k\}_{k=1}^{N_Q}$, $Q_k \in \mathbb{R}^{n_Q}$, and its estimate $\{\hat{Q}_k\}_{k=1}^{N_Q}$, $L_{\mathrm{MAE}}(Q_k, \hat{Q}_k) = \frac{1}{N_Q n_Q} \|Q_k - \hat{Q}_k\|_1$.

timated condition x_0 is propagated through each step k of the whole dataset. The main issue of BPTT is that it involves optimizing cost functions that are both computationally expensive to evaluate and to differentiate, and are highly nonlinear, which makes the learning problem difficult to solve [62], [63]. A good compromise is to resort on the so-called truncated BPTT [64], [65], where the propagation of the estimate $\hat{x}_k$ is carried only for a limited number of steps $F \ll N$. In our identification scheme, the idea of truncated BPTT translates into the following multi-step ahead modification of (2.5):

$$\begin{aligned}
\min_{e,f,d} \quad & \sum_{k=k_0}^{N-F} \left(\sum_{f=0}^{F} \alpha L_1(\hat{O}_{k+f}, O_{k+f}) \right. \\
& \left. + \sum_{f=1}^{F} \sum_{r=0}^{f-1} \beta L_2(\hat{x}_{k+f}^{k+r}, x_{k+f}) + \gamma L_3(O_{k+f}, \hat{O}_{k+f}^{k+r}) \right) \\
& \text{s.t.} \quad \begin{aligned}
\hat{x}_{k+\bar{f}+1}^{k+r} &= f(\hat{x}_{k+\bar{f}}^{k+r}, u_{k+\bar{f}}), \\
& \quad \bar{f} = r+1, \ldots, F \\
\hat{x}_{k+r+1}^{k+r} &= f(x_{k+r}, u_{k+r}) \\
x_{k+r} &= e(I_{k+r-1}) \\
\hat{O}_k &= d(x_k) \\
\hat{O}_{k+f}^{k+r} &= d(\hat{x}_{k+f}^{k+r})
\end{aligned}
\end{aligned} \tag{2.6}$$

where in (2.6) we consider the multi-step ahead predictions $\hat{x}_{k+f}^{k+r}$ of the state vector at step $k+f$ based on iterating the model in open-loop for $f-r$ steps from the initial state $x_{k+r} = e(I_{k+r-1})$. Note that, compared to a standard truncated BPTT, here we also evaluate explicitly the quality of the predictions generated from any step to any other step within the interval $[k, k+F]$, i.e., from any x_{k+r} to any x_{k+f}, $f > r$. At first sight the use of the index r may seem redundant (for example the index $k_1 + k_2$ is covered by $k = k_1$, $r = k_2$ and by $k = k_1 + k_2$, $r = 0$). However, introducing such a redundancy is useful when stochastic gradient descent (SGD) is adopted to solve (2.6), as most commonly used in deep learning [66]. In fact, in SGD a set of random, possibly non-consecutive, values of the index k are selected at each optimization step to form the mini-batch. Hence, the time-windows between k and $k+F$ may not overlap in the mini-batch.

Note that the proposed truncated BPTT approach includes the one-step ahead approach for $F = 1$.

2.4.2 Network topology

In principle, each sub-network e, f, and d could be designed with different topologies and activation functions, thus allowing the user to incorporate possible prior knowledge of the process to identify. In this Chapter we restrict our analysis to two alternative architectures: (i) a fully connected feed-forward (FF) topology for all the sub-networks e, f, and d; (ii) a quasi-LPV parameterization of the maps f and d while maintaining a general FF topology for the encoder e. The latter option allow us to learn models in the following quasi-LPV form

$$\begin{cases} x_{k+1} = A(x_k, u_k)[x_k' \; 1]' + B(x_k, u_k)u_k \\ y_k = C(x_k)[x_k' \; 1]' \end{cases} \tag{2.7}$$

whose usefulness will be detailed in Section 2.5.2. Each coefficient of A, B and C in (2.7) is in turn defined as the output of a FF network.

2.4.3 Feature selection and model reduction

The most important hyper-parameters of our approach for achieving a good fit are n_a, n_b and n_x. As it is common in most identification techniques, tuning such parameters often requires physical insight and/or extensive trial and error. Here we propose a simple heuristics to facilitate the selection process.

It is known that the inclusion of ℓ_1-penalties in an optimization problem (a.k.a. the *shrinkage operator*) induces sparse solutions [67]. Consider the general FF topology (i). An approach to attempt reducing the number n_x of states of the model is to introduce the following variation of the so-called group LASSO operator [68]

$$L_{n_x}(\omega) = \chi_1 \sum_{i=1}^{n_x} i^2 \|\omega_{[i]}\|_1 \tag{2.8}$$

where $\chi_1 > 0$, $\chi_1 \in \mathbb{R}$, ω is the vector of weights of the initial layer of the maps f and d, and $\omega_{[i]}$ is the subvector of ω corresponding to the state

component $x_{k,i}$. After solving (2.5) with the additional penalty (2.8), all $x_{k,i}$ such that $\omega_{[i]}$ is negligible are considered as redundant, and n_x is decreased accordingly. A similar argument can be used in case the quasi-LPV topology (2.7) is used, by inspecting whether A, B, C depend on $x_{k,i}$ and whether the $i-$th column of A and C is negligible.

Regarding the encoder e, we penalize more the weights in the first layer of e associated with the components of $I_k = [y_k' \ \dots \ y_{k-n_a+1}' \ u_k' \ \dots \ u_{k-n_b+1}']'$ corresponding to less recent input/output values. To this end, let θ be the vector of weights corresponding to the first layer of e, and let $\theta_{[\ell]}$ the subvector of θ corresponding to weights associated with either $u_{k-\ell}$ or $y_{k-\ell}$, $\ell = 0, \dots, T$, $T = \max\{n_a, n_b\} - 1$. Consider the following group LASSO penalty function

$$L_e(\theta) = \sum_{\ell=0}^{T} \chi_2 (\ell + 1)^2 \|\theta_{[\ell]}\|_1 \tag{2.9}$$

where $\chi_2 > 0$, $\chi_2 \in \mathbb{R}$, is a new hyper-parameter to choose. A possible way of choosing χ_2 is to solve (2.5) with the additional penalty (2.9) for different values of χ_2 and choose the value that best trades off between quality of fit and small values of n_a, n_b. Then, after fixing the best values of n_a, n_b found, problem (2.5) is solved again without adding (2.9).

As we will describe in Section 2.6, we will also include small ℓ_2-regularization terms in the objective function to minimize.

2.5 Nonlinear state estimation and control

2.5.1 Filtering and state reconstruction

The encoding part e of the PPE network provides x_k as a static function of the past n_a outputs and n_b inputs, collected in vector I_{k-1}. To avoid storing I_{k-1} and to filter noise out, we consider here standard recursive filtering and state-reconstruction technique to recover x_k iteratively from input and output measurements.

A standard approach to estimate x_k is to use model-based state-estimation techniques such as extended Kalman filters (EKF). However, this

would require additional tuning effort and would restrict the choice of the activation functions used in f, g to be differentiable for linearization.

An alternative method is to extend the overall learning objective (2.5) to also train a "neural observer". Similar to the bridge function f introduced to forward the state x_k and new input u_k to the next state x^*_{k+1}, we can introduce a similar structure to build, together with e, d, f, an additional map $s : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_x}$ from the current state estimate $\hat{x}_k$, input u_k, *and* new measured output y_k to the updated state estimate $\hat{x}_{k+1}$. This can be achieved by adding

$$\beta_4 L_4(\hat{x}_{k+1}, x_{k+1}) + \gamma_5 L_5(\hat{O}^\star_{k+1}, O_{k+1}) \tag{2.10a}$$

in the loss function in (2.5), where

$$\begin{aligned} \hat{x}_{k+1} &= s(x_k, u_k, y_k) \\ \hat{O}^\star_k &= d(\hat{x}_k),\ k = 0, \ldots, N-1 \end{aligned} \tag{2.10b}$$

L_4, L_5 are appropriate loss functions, and $\beta_4, \gamma_5 \geq 0$.

A clear benefit of the approach in (2.10) is that no separate tuning process is required *after* training the process model f, g, in that the observer s is trained directly on the data set. On the other hand, having coupled the fit of the model with the synthesis of the observer leaves no freedom to re-tune the observer without fitting again both of them. For a detailed comparison between the two approaches we refer the interested reader to [51].

2.5.2 Nonlinear model predictive control

Having identified a model in the state-space form (2.1) or (2.7), we can employ any state-feedback controller synthesis technique to control the system generating the data. Among such techniques, model predictive control (MPC) is probably the most flexible [69], [70] for dealing with multivariable systems under constraints on process variables. To deal with nonlinear systems, one of the most commonly used MPC scheme is the so-called linear time-varying (LTV) formulation (a.k.a. real-time iteration scheme [71]), which is based on linearizing the model around the previous optimal solution [72].

As for EKF, this method requires the activation functions used in f, g to be differentiable, and the computation of the Jacobian matrices of the model along the prediction horizon. This can be mitigated by using the quasi-LPV affine form (2.7) by directly computing the matrices A, B, C for the nominal values of x_k, u_k along the prediction horizon and neglecting their sensititivies with respect to x_k, u_k. This scheme is summarized in Algorithm 1.

2.6 Experimental Results

2.6.1 Synthetic benchmark problems

We first apply the proposed nonlinear state-space identification approach to the following synthetic benchmark problems: the Hammerstein-Wiener system

$$\Sigma_{HW} = \begin{cases} x_{k+1} &= \begin{bmatrix} 0.7555 & 0.25 \\ -0.1991 & 0 \end{bmatrix} x_k + \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} v_k \\ v_k &= \begin{cases} \sqrt{u_k} & \text{if } u > 0 \\ u_k & \text{otherwise} \end{cases} \\ w_k &= \begin{bmatrix} 0.6993 & -0.4427 \end{bmatrix} x_k \\ y_k &= w_k + 5\sin(w_k) \end{cases} \tag{2.11}$$

and the discrete-time nonlinear system

$$\Sigma_T = \begin{cases} x_{k+1,1} &= x_{k,1} - k_1\sqrt{x_{k,1}} + k_2 v_k \\ x_{k+1,2} &= x_{k,2} + k_3\sqrt{x_{k,1}} - k_4\sqrt{x_{k,2}} \\ y_k &= x_{k,2} \end{cases} \tag{2.12}$$

that describes a tank system with possible overflows neglected [73]. In (2.12) we set $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, $k_4 = 0.3$ and consider two cases: one where we directly control v_k (i.e: $v_k = u_k$) that will be referred to as Σ_{T1}, and one where $v_k = \text{sign}(u_k)u_k^2$ that will be referred to as Σ_{T2}.

For both Σ_{T1} and Σ_{T2} we collect a training dataset consisting of 20,000 training samples generated by exciting the system with a sequence of step signals of length 5 steps with random amplitudes drawn from the Gaussian distribution $\mathcal{N}(1, 1)$. For Σ_{HW} we consider a training set of 10,000 samples generated using the same methodology but with amplitudes of the input signals distributed in $\mathcal{N}(0, 1)$ and length equal to 7

steps. As test cases, for all these systems, we consider the open simulation accuracy of the learned model when excited by 1,000 test input samples consisting of random-amplitude step signals with the same characteristics as the one used for training, in comparison with the noise-free response produced by the real process when subject to the same input sequence.

All the signals are scaled by subtracting the empirical mean and dividing by the standard deviation computed on the training set. Once normalized, a zero-mean Gaussian white-noise with standard deviation $\sigma = 0.02$ is added on both the training and test signals to mimic measurement noise.

2.6.2 Experimental and simulation benchmarks

We further test the capabilities of the proposed approach on three publicly available dataset. The first taken from an experimental magneto-rheological fluid damper system [74], [75], the second one from a simulated physically-accurate continuous-time two-tank system [76], the third one from the well known "Silverbox" system [77]. The systems generating the three dataset will be refereed to as Σ_{RH}, Σ_{Tank}, and Σ_{S}, respectively.

The first two datasets contain 3,499 and 3,000 samples, respectively. In both cases, the first 2,000 samples of the set are used for training/ validation purposes while the remaining samples for testing. The Silverbox dataset is extracted as in [5]. Again the signals in the dataset are normalized but no additional noise is superimposed.

In all cases, since during the training of the neural networks we employ an early-stopping criterion based on validation data, 5% of the training dataset is reserved for validation.

2.6.3 Hyperparameter selection and implementation details

For both the FF network topology (2.1) and the affine quasi-LPV one (2.7), described in Section 2.4.2 as type (i) and type (ii), respectively, the hid-

den part of all the involved networks consist of 3 layers of 30 neurons each with ReLU [78], [79] activation functions, followed by a final linear output layer.

The network has been implemented in Keras [80] using Tensorflow [6] as back-end, and trained via the AMSgrad algorithm [81]. The training procedure is carried out in two stages: first the ANNs are trained with $\alpha = 10, \beta = 0.3, \gamma = 0$ until convergence, and then trained again (using as a starting point the weights obtained in the previous phase) using $\alpha = 0, \beta = 10, \gamma = 1$. Unless otherwise noted, ℓ_2-regularization terms with penalty $\lambda = 0.0001$ are added on all but bias terms coefficients of the nonlinear and, for the type (i) topology only, linear neurons to smooth the fitting process out. Moreover, the same ℓ_2-penalty is added on the final linear layer of the neural network modeling the last column of matrix $A(x_k, u_k)$ in case the quasi-LPV topology (2.7) is used. Under the same topology, we also zero the coefficients of the last column of matrix $C(x_k)$.

The above two-step procedure is used to ease the learning process. As the focus of the first step is mostly on the L_1 objective, the learning of e and d is emphasized ($\alpha = 10$), with a small contribution ($\beta = 0.3$) of L_2 maintained to avoid that the resulting state vector cannot be properly updated over time. Once a good estimate of the decoder/encoder pair has been determined in the first stage, the second stage takes this as the initial guess to completely focus ($\alpha = 0$) on enforcing that the model propagates correctly over time ($\beta = 10$, $\gamma = 1$).

Note that in our experiments we have not made any attempt to optimize the selected values of α, β, γ. When best identification performance is sought, global optimization algorithms based on surrogate functions [9], [82] could be used for optimal tuning of α, β, γ, and possibly other hyper-parameters.

Performance results are measured in terms of the best fit ratio (FIT)

$$\text{FIT} = \max\left\{0, 1 - \frac{\|y - \hat{y}\|_2}{\|y - \bar{y}\|_2}\right\} \tag{2.13}$$

where $\bar{y}$ is the average of the output signal y over all the samples, and $\hat{y} = \{\hat{y}_k\}$ is the estimated response, where $\hat{y}_k$ is the *open-loop prediction*

extracted from $\hat{O}_{k+1} = d(\hat{x}_{k+1})$ with

$$\begin{array}{rcl} \hat{x}_{k+1} & = & f(\hat{x}_k, u_k),\ k = k_0, \ldots, N-1 \\ \hat{x}_{k_0} & = & e(I_{k_0-1}) \end{array} \tag{2.14}$$

We set $m = 1$ in all tests as we are only interested in predicting the next output y_{k+1} and filter the current y_k. As we plan to never exceed the value 15 for n_a, n_b, all simulations start from k_0 =15 in order to have enough samples to form I_{k_0-1}. Unless otherwise noted, we report the best and average FIT and its standard deviation obtained on 10 different runs with different random seeds.

2.6.4 Fit results

We evaluate the FIT open-loop simulation performance (2.13) when the fitting process is either based one-step ahead criterion ($F = 1$) or truncated BPTT ($F = 4$), for both the quasi-LPV and FF topologies. In all cases we set $n_x = 6$, $n_a = n_b = 10$.

Results are reported in Table 2 for the synthetic benchmarks and in Table 3 for the experimental/simulation benchmarks. The results highlight that the quasi-LPV parameterization (2.7) well reproduces the behavior of the system in all the proposed benchmarks, and slightly outperforms the FF architecture. This is especially evident for Σ_{T2}. The results also confirm the beneficial effect of truncated BPTT.

For comparison, for the three non-synthetic benchmark problems we train a nonlinear ARX models of the same order, equipped with a dense feedforward ANN regressor with 3 layers of 30 ReLU neurons each, trained using the System Identification Toolbox for MATLAB [83]. We consistently achieve roughly the following FITs: 0.94 (Σ_{Tank}), 0.55 (Σ_{RH}), and 0.94 (Σ_S).

2.6.5 Feature selection and model reduction

We test the model-selection technique presented in Section 2.4.3 to reduce n_a, n_b and n_x. Starting with $n_x = 6$ and $n_a = n_b = 10$, we attempt reducing n_a, n_b by imposing the group LASSO penalty (2.9) with

$\chi_2 = 0.0003$, while maintaining the ℓ_2-regularization penalty $\lambda = 0.0001$ on the remaining regularized coefficients. Similarly, starting with the same values of n_x, n_a, n_b, we impose the group LASSO penalty (2.8) to attempt reducing n_x with $\chi_1 = 0.0003$. Numerical tests are performed on Σ_{T2}, with training carried out using truncated BPTT with $F = 4$.

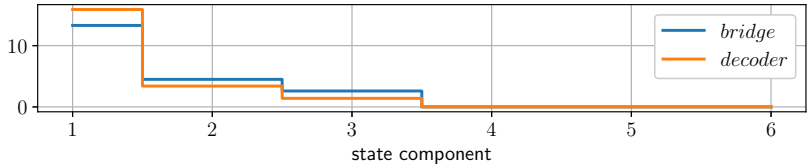
The number of coefficients whose absolute value is larger than 0.001 (which we selected as the threshold used to consider the coefficient as negligible) is reported in Figure 2. In particular, Figure 2a clearly shows that only 3 state components are commonly used, so that n_x can be reduced to $n_x = 3$. Similarly, from Figure 2b we can note that only y_k, $\ldots, y_{k-2}$ and $u_k, \ldots, u_{k-4}$ contribute significantly to the encoder function e, so that we can decide to limit I_k to them. This suggests setting $n_a = n_b$ to either 3, 4, or 5.

This is confirmed by the results reported in Table 4, which shows the fit performance occurring when the models are trained with $n_x = 2$, $n_a = n_b = 5$, with $n_x = 3$, $n_a = n_b = 5$, and with $n_x = n_a = n_b = 5$, without including ℓ_1-penalties for sparsification. The difference in performance between using $n_x = 3$, $n_a = n_b = 5$, and both using $n_x = n_a = n_b = 5$ and the original values $n_x = 6, n_a = n_b = 10$, in the original feedforward topology is negligible. Very good performance are also obtained by $n_x = 2$, $n_a = n_b = 5$, confirming the marginal relevance of introducing a third state shown in Figure 2a.

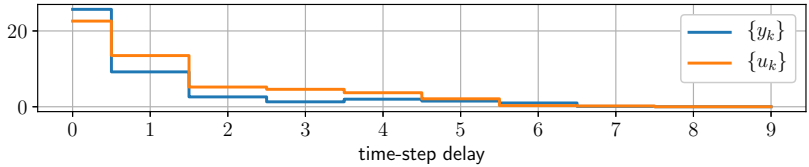
2.6.6 LTV-MPC based on learned model

We evaluate the performance of the MPC controller presented in Section 2.5.2 in controlling Σ_{T2}. The MPC controller is based on the quasi-LPV model obtained by training from the same dataset with $n_x = 6$ and $n_a = n_b = 10$, and by setting $n_p = 5$, $W_y = 1$, $W_u = 0.001$, $W_{\Delta u} = 0.01$, under the box constraints $|u_k| \leq 0.8$.

Figure 3 shows the obtained closed-loop performance when tracking a shifted sine-sweep reference signal. To close the MPC loop, in this example we used the encoder function to estimate the state x_k from past input/output samples contained in I_{k-1}, although other observers (like



(a) Average number of non-negligible neurons in f and d as a function of the state component $x_{k,i}, i = 1, \ldots, n_x$



(b) Average number of non-negligible neurons in function e acting on I_k, with respect to the time-step delay i of either the input sample u_{k-i} or the output sample y_{k-i}, $i = 0, \ldots, n_a - 1 = n_b - 1$, the neuron is acting on.

Figure 2: Number of active neurons in e, f, d.

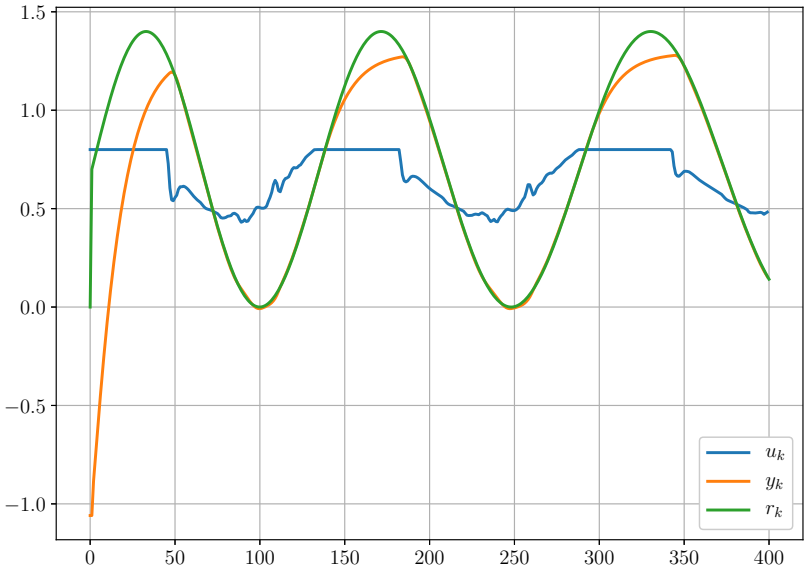


Figure 3: Tracking performance of LTV-MPC (Algorithm 1) applied to control system Σ_{T2} (quantities are in normalized units).

a time-varying Kalman filter based on the same quasi-LPV model) could be used as well.

2.6.7 Computational aspects

The overall CPU time to simulate the closed-loop LTV-MPC system, running the decoder, and computing the control action via the general purpose L-BFGS-B solver [84] over 400 sampling steps is ≈ 5 seconds on a laptop equipped with an Intel Core i5 6200u CPU and 16 GB of RAM. Note that we did not make any attempt towards efficiency of implementation, for example a more efficient solver like the one proposed in [85] could be used to compute the MPC action.

Regarding the CPU time spent for training the models, the proposed procedure is carried out on average in ≈ 10 minutes on the aforementioned machine for dataset consisting of 20,000 samples. Interestingly,

the computation effort is not very sensitive to the particular choice of n_x, n_a, and n_b.

Models with $n_x = 6$, $n_a = n_b = 10$ require $\approx$ 10,000 single precision coefficients. Clearly, the number of model coefficients heavily depends on the chosen topology, so that it can be drastically reduced if memory footprint and/or throughput are of concern. Moreover, we mention that one could apply recent results on model reduction for ANNs, see, e.g., [86] and references therein. In general, one must find the best trade-off between the number of optimized model coefficients and the obtained closed-loop tracking performance when training a nonlinear state-space model for control purposes.

2.7 Conclusions

We have proposed a viable approach to learn nonlinear state-space models from input/output data for model-based control systems design. The method can be applied either to identify a nonlinear model from experimental data, or from a high-fidelity simulator, or to reduce the order of an existing nonlinear model. Indeed, the approach allows the direct control of the number of states, which in turn dictates the complexity of model-based nonlinear estimators and MPC controllers.

Parameter	Symbol	Meaning
past I/O window width	n_a, n_b	size of the information vector I_k employed to construct the state x_{k+1}
autoencoding window	m	time length of the output vector O_k
state dimension	n_x	number of neurons in the "bottleneck" layer of the PPE
multistep prediction horizon	F	length of the prediction horizon used in truncated BPTT
relative weights	α, β, γ	relative weights scalarizing the multi-objective fitting criterion in (2.5)

Table 1: Hyper-parameters of the proposed learning method

System		A/1	B/1	A/F	B/F
Σ_{HW}	average	0.983	0.989	0.989	0.991
	standard deviation	0.002	0.001	0.001	0.001
	best	0.986	0.991	0.990	0.992
Σ_{T2}	average	0.873	0.913	0.943	0.961
	standard deviation	0.024	0.019	0.013	0.015
	best	0.903	0.938	0.963	0.977
Σ_{T1}	average	0.944	0.969	0.973	0.980
	standard deviation	0.020	0.003	0.003	0.004
	best	0.960	0.976	0.977	0.985

Table 2: Performance results (FIT) on synthetic benchmarks for different combinations of topologies and training procedures: A = FF topology, B = quasi-LPV topology; 1 = training based on one-step ahead loss ($F = 1$), F = training based on multi-step ahead loss ($F = 4$).

Algorithm 1 LTV-MPC algorithm based on model (2.7)

Input: Prediction horizon n_p, control horizon n_m, weight matrices $W_y \in \mathbb{R}^{n_y \times n_y}$, $W_u \in \mathbb{R}^{n_u \times n_u}$, $W_{\Delta u} \in \mathbb{R}^{n_u \times n_u}$; output and input reference signals $r_t \in \mathbb{R}^{n_y}$, $u_t^r \in \mathbb{R}^{n_u}$, $t = 0, 1, \ldots$; current state estimate x_t; set $\bar{x}_t = x_t$.

1. **compute** the sequence of predicted states $\{\bar{x}_{t+1}, \ldots, \bar{x}_{t+n_p}\}$ given the current guess of the input sequence $\{\bar{u}_t \cdots \bar{u}_{t+n_p-1}\}$, with $\bar{u}_{t+k} = \bar{u}_{t+n_m-1}$ for all $k \geq n_m - 1$;

2. **compute**
$$A_k = A(\bar{x}_{t+k}, \bar{u}_{t+k}), \quad B_k = B(\bar{x}_{t+k}, \bar{u}_{t+k})$$
$$C_k = C(\bar{x}_{t+k})$$

3. **solve** the quadratic programming problem
$$\begin{aligned} \arg\min_{u_t, \ldots, u_{t+n_m}} & \sum_{k=0}^{n_p-1} \|W_y(y_{t+k+1} - r_{t+k+1})\|_2^2 \\ & + \|W_u(u_{t+k} - u_{t+k}^r)\|_2^2 + \|W_{\Delta u} \Delta u_{t+k}\|_2^2 \end{aligned}$$
$$\begin{aligned} \text{s.t. } & x_{j+1} = A_k[x_j' \ 1]' + B_k u_j \\ & y_{j+1} = C_{k+1}[x_{j+1}' \ 1]' \\ & \Delta u_j = u_j - u_{j-1}, \ j = t+k, \ k = 0, \ldots, n_p - 1 \\ & \text{linear constraints on } \Delta u_{t+k}, u_{t+k}, x_{t+k}, y_{t+k} \\ & \Delta u_{t+k} = 0, \ n_m \leq k < n_p \end{aligned}$$
to get the optimal sequence $u_t^\star, \ldots, u_{t+n_m-1}^\star$;

4. **set** the current input $u_t = u_t^\star$;

5. **update** the nominal input sequence $\bar{u}_{t+k} = u_{t+k}^\star$, $1 \leq k \leq n_m - 1$, $\bar{u}_{t+k} = u_{t+n_m-1}^\star$, $n_m \leq k \leq n_p - 1$.

Output: Command input u_t, updated nominal input sequence $\{\bar{u}_{t+1}, \ldots, \bar{u}_{t+n_p}\}$.

System		A/1	B/1	A/F	B/F
Σ_{RH}	average	0.815	0.815	0.818	0.882
	standard deviation	0.040	0.044	0.015	0.025
	best	0.859	0.870	0.839	0.908
Σ_{Tank}	average	0.896	0.904	0.916	0.915
	standard deviation	0.008	0.020	0.007	0.009
	best	0.911	0.923	0.932	0.927
Σ_S	average	0.905	0.951	0.966	0.986
	standard deviation	0.015	0.013	0.004	0.003
	best	0.931	0.963	0.974	0.991

Table 3: Performance results (FIT) on experimental and simulation benchmarks for different combinations of topologies and training procedures: A = FF topology, B = quasi-LPV topology; 1 = training based on one-step ahead loss ($F = 1$), F = training based on multi-step ahead loss ($F = 4$).

System		FF-2/5	FF-3/5	FF-5	FF-6
Σ_{T2}	average	0.905	0.903	0.925	0.943
	std. deviation	0.059	0.061	0.026	0.013
	best	0.948	0.952	0.964	0.963

Table 4: Performance (FIT) of the reduced-order models in learning Σ_{T2}. FF-5 = feedforward topology with $n_x = n_a = n_b = 5$. FF-3/5 = feedforward topology with $n_x = 3$, $n_a = n_b = 5$. FF-2/5 = feedforward topology with $n_x = 2$, $n_a = n_b = 5$. FF-6 = feedforward topology with $n_x = 6, n_a = n_b = 10$ (reported for comparison)

Chapter 3

A machine-learning approach to synthesize virtual sensors for parameter-varying systems

3.1 Introduction

Most real-world processes exhibit complex nonlinear dynamics that are difficult to model, not only because of the interactions between input and output variables, but also because of the presence of time-varying signals that change the way the involved quantities interact over time. A typical instance is the case of systems subject to wear of components, in which the dynamics slowly drift from a nominal behavior to an aged one, or systems affected by slowly-varying unknown disturbances, such as unmeasured changes of ambient conditions. These systems can be well described using a *parameter-varying* model [87] that depends on a vector $\rho_k \in \mathbb{R}^S$ of parameters, that in turn evolves over time:

$$\Sigma_P \triangleq \begin{cases} x_{k+1} = f(x_k, u_k, \rho_k) \\ \rho_{k+1} = h(\rho_k, k, u_k) \\ y_k = g(x_k, \rho_k) \end{cases} \tag{3.1}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state vector, $y_k \in \mathbb{R}^{n_y}$ is the output vector, $u_k \in \mathbb{R}^{n_u}$ is the input vector, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^S \to \mathbb{R}^{n_x}$, $g : \mathbb{R}^{n_x} \times \mathbb{R}^S \to \mathbb{R}^{n_y}$ and $h : \mathbb{R}^S \times \mathbb{R}^{n_u} \times \mathbb{R} \to \mathbb{R}^S$. In the following we assume that the mappings in (3.1) are *unknown*.

Special cases of (3.1) widely studied in the literature are linear parameter-varying (LPV) systems [88], in which f, g are linear functions of x_k, u_k, and switched affine systems [89], in which ρ_k only assumes a value within a finite set.

Inferring the value of ρ_k in real time from input/output data can be useful for several reasons. In predictive maintenance and anomaly/fault detection [90]–[92], detecting a drift in the value of ρ_k from its nominal value or range of values can be used first to detect a fault and then to isolate its nature. In gain-scheduling control [93], [94], ρ_k can be used instead to decide the control law to apply at each given time instant.

Due to the importance of estimating ρ_k, various solutions have been proposed in the literature to estimate it during system operations. If the model in (3.1) were known, even if only approximately, nonlinear and robust state estimators could be successfully applied [95]. On the other hand, if the mechanism regulating the interaction between ρ_k and the measurable quantities (usually u_k and y_k) is not known, but a dataset of historical data is available, the classical indirect approach would be to identify an overall model of Σ_P using a system identification technique [96] and then build a model-based observer to estimate ρ_k. The drawbacks of such an indirect approach are that it can be a very time-consuming task and that the resulting model-based observer can be complex to implement. This issue is especially cumbersome if one is ultimately interested in just getting an observer and have no other planned use for the model itself.

Virtual sensors [97], [98] provide an alternative approach to solve such a problem: the idea is to build an *end-to-end* estimator for ρ_k by directly learning from data the mapping from measured inputs and outputs to ρ_k itself. The approach is interesting because it does not require identifying a full model of the system from data, nor it requires simplifying an existing model (such as a high-fidelity simulation model) that would be

otherwise too complex for model-based observer design. Similar estimation problems have been tackled in the context of novelty detection [99] and of time-series clustering [100], [101].

3.1.1 Contribution

The goal of this Chapter is to show an approach to synthesize virtual sensors that can estimate ρ_k when its measurements are not available by using data acquired when such a quantity is directly measurable. Such a scenario often arises in serial production, in which the cost of components must be severely reduced. The purpose of the proposed approach is to enable replacing physical sensors with lines of code.

The method developed here is loosely related to Multiple Model Adaptive Estimation [102] (MMAE) and consists of three main steps:

1. Learn a finite set of simple linear time-invariant (LTI) models from data that roughly covers the behavior of the system for the entire range of values of ρ_k of interest;

2. Design a set of standard linear observers based on such models;

3. Use machine-learning methods to train a lightweight predictor that maps the estimates obtained by the observers and raw input and output signals into an estimate $\hat{\rho}_k$ of ρ_k.

To do so, we extends the preliminary results presented in [22] in several ways: it formulates the problem for nonlinear systems; it explores the performance of the approach for mode-discrimination of switching systems and it provides a thorough performance analysis of various light--weight machine-learning techniques that can be used to parameterize the virtual sensor architecture. In doing so, it also provides an entirely off-line alternative strategy for identifying the local linear models required to synthesize the bank of observers based on an interpretation of well-known decision tree regressors as a supervised clustering scheme.

The intuition behind our approach is that, in many cases of practical interest, the dynamics of ρ_k are slower than the other dynamics of the

system. This fact suggests that a linear model identified on a dataset in which ρ_k is close to a certain value $\bar{\rho}$ will well approximate Σ_P for all $\rho_k \approx \bar{\rho}$. Following this idea, we envision a scheme in which N_θ values $\bar{\rho}_i$, $i = 1, \ldots, N_\theta$ are automatically selected and, for each value $\bar{\rho}_i$, a linear model is identified and, finally, a corresponding linear observer synthesized. A machine-learning algorithm is then used to train a predictor that is fed with the performance indicators constructed from such observers, together with raw input and output data, to produce an estimate $\hat{\rho}_k$ of ρ_k at each given time k.

The rest of the Chapter is organized as follows: in Section 3.2 we recall the MMAE framework and introduce the necessary steps to bridge such a model-based technique to a data-driven framework. In Section 3.3, we detail the overall virtual sensor architecture and the internal structure of its components. Section 3.4 is devoted to studying the quality of estimations and the numerical complexity of the synthesized virtual sensor on some selected nonlinear and piecewise affine (PWA) benchmark problems, including the problem of estimating the state of charge of a battery, to establish both the estimation performance of the approach and the influence of its hyper-parameters. Finally, some conclusions are drawn in Section 3.5.

3.2 Multiple Model Adaptive Estimation

Following the formulation in [103], [104], this section recalls the main concept of the MMAE approach. Consider the dynamical system

$$\Sigma \triangleq \left\{ \begin{array}{rcl} x_{k+1} & = & f(\theta_k, x_k, u_k) \\ y_k & = & h(\theta_k, x_k, u_k) \end{array} \right. \tag{3.2}$$

in which $\theta_k \in \mathbb{R}^{n_\theta}$ is a generic parameter vector. The overall idea of MMAE is to use a bank of N_θ state estimators[1] — each one associated to a specific value $\theta_i \in \Theta \triangleq \{\theta_1, \ldots, \theta_{N_\theta}\}$ — together with a hypothesis testing algorithm to infer information about (3.2), e.g.: to build an estimate $\hat{x}_k$ of x_k. In this scheme, the intended purpose of the latter component is

[1] Usually Kalman filters (KF) are employed, but exceptions exist [102] .

to infer, from the behavior of each observer, which one among the different models ("hypotheses") is closest to the underlying process, and use such information to construct an estimate $\hat{x}$ of the state of Σ. For linear time-invariant (LTI) representations, a classical approach to do so is to formulate the hypothesis tester as an appropriate statistical test, exploiting the fact that the residual signal produced by a properly matched KF is a zero-mean white-noise signal.

MMAE is a model-based technique in that it requires a model of the process, a set Θ of parameter vectors, and a proper characterization of the noise signals supposed to act on the system. Among those requirements, determining Θ is especially crucial to get reliable results as, at each time, at least one value $\theta_j \in \Theta$ must describe the dynamics of the underlying system accurately enough. In many practical situations, it is not easy to find a good tradeoff between keeping N_θ large enough to cover the entire range of the dynamics and, at the same time, small enough to limit the computational burden manageable and avoid the tendency of MMAE to work poorly if too many models are considered [105]. Another difficulty associated with MMAE schemes is the reliance on first principles to synthesize the hypothesis tester. Moreover, many approaches require sophisticated statistical arguments, which can hardly be tailored to user-specific needs.

3.3 Data-driven determination of linear models

The first step to derive the proposed data-driven virtual-sensor is to reconcile the MMAE framework with the parameter-varying model description in (3.1). Assume for the moment that f and g in (3.1) are known and differentiable. Then, in the neighborhood of an arbitrary tuple $(\bar{\rho}, \bar{x}, \bar{u})$ it is possible to approximate (3.1) by

$$\begin{aligned} x_{k+1} - \bar{x} &\approx f(\bar{x}, \bar{u}, \bar{\rho}) - \bar{x} + \nabla_x f(\bar{x}, \bar{u}, \bar{\rho})(x_k - \bar{x}) + \\ &\quad + \nabla_u f(\bar{x}, \bar{u}, \bar{\rho})(u_k - \bar{u}) \\ y_k &\approx g(\bar{x}, \bar{\rho}) + \nabla_x g(\bar{x}, \bar{u}, \bar{\rho})(x_k - \bar{x}) \end{aligned} \tag{3.3}$$

In (3.3) the contributions of the Jacobians with respect to ρ is neglected due to the fact that, as mentioned earlier, ρ_k is assumed to move slowly

enough to remain close to $\bar{\rho}$ within a certain time interval, meaning the neglected Jacobians would be multiplied by $\rho_k - \bar{\rho} \approx 0$. Hence, from (3.3) we can derive the following affine parameter-varying (APV) approximation of (3.1)

$$\begin{array}{rcl} x_{k+1} & \approx & A(\rho_k)x_k + B(\rho_k)u_k + d(\rho_k) \\ y_k & \approx & C(\rho_k)x_k + e(\rho_k) \end{array} \tag{3.4}$$

in which the contribution of the constant terms $\bar{x}$, $\bar{u}$ is contained in the bias terms $d(\rho_k)$, $e(\rho_k)$. In conclusion, if Σ_P were known, a MMAE scheme could be used to compute the likelihood that the process is operating around a tuple $(\bar{x}, \bar{u}, \rho_i)$, where $\rho_i \in \Theta^\rho \triangleq \{\rho_1, \ldots, \rho_{N_\theta}\}$ is used in place of the parameter vector θ_k in (3.2).

3.3.1 Learning the local models

As model (3.1) is not available, we need to identify the set of linear (affine) models in (3.4) from data. Assuming that direct measurements of the state x_k of the physical system are not available, we restrict affine autoregressive models with exogenous inputs (ARX) of a fixed order, each of them uniquely identified by a parameter vector $\gamma \in \mathbb{R}^{n_\gamma}$.

Learning an APV approximation of Σ_P amounts to train a functional approximator $M_{LPV} : \mathbb{R}^S \to \mathbb{R}^{n_\gamma}$ to predict the correct vector γ_i corresponding to any given $\bar{\rho}_i$. Given a dataset $D_N \triangleq \{u_k, y_k, \rho_k\}$, $k = 1, \ldots, N$, of samples acquired via an experiment on the real process, such a training problem is solved by the following optimization problem

$$\begin{array}{lll} \min\limits_{M_{LPV}} & \sum\limits_{k=k_1}^{N} & L_{M_{LPV}}(\hat{y}_k, y_k) \\ \text{subject to} & & \hat{y}_k = [-y_{k-M}, \ldots, -y_{k-1}, u_{k-M}, \ldots, u_{k-1}\ 1]\gamma_k \\ & & \gamma_k = M_{LPV}(\rho_k) \\ & & k = k_1, \ldots, N \end{array} \tag{3.5}$$

where $k_1 \triangleq M + 1$, and $L_{M_{LPV}}$ is an appropriate loss figure. Note that, as commonly expected when synthesizing virtual sensors, we assume that measurements of ρ_k are available for training, although they will

not be directly measurable during the operation of the virtual sensor. Moreover, note that problem M_{LPV} is solved offline, so the computation requirements of the regression techniques used to solve (3.5) are not of concern.

Compared to adopting a recursive system identification technique to learn a local linear model of the process at each time k, and then associate each γ_k to its ρ_k (e.g., by using Kalman filtering techniques [1], [22]), the approach in (3.5) does not require tuning the recursive identification algorithm and takes into account the value of ρ_k at each k. This prevents that similar values of ρ are associated with very different values of γ, assuming that the resulting function M_{LPV} is smooth enough.

An end-to-end approach to select the representative models

By directly solving (3.5), a set $\Gamma \triangleq \{\gamma_i\}_{i=M+1,\dots,N}$ of local models is obtained. Using $\Theta = \Gamma^\rho$ in an MMAE-like scheme would result in an excessively complex scheme. To address this issue, a smaller set of models could be extracted by running a clustering algorithm on the dataset Γ, and the set Θ^ρ of representative models selected as the set of the centroids of the found clusters.A better idea comes from observing that some regression techniques, such as decision-tree regressor [3] (DTRs), naturally produce piece-wise constant predictions, which suggests the following alternative method: (i) train a DTR to learn an predictor $\hat{M}_{LPV} : \mathbb{R}^S \rightarrow \mathbb{R}^{n_\gamma} \times \mathbb{R}^S$ (in an autoencoder-like fashion [49], [106]), possibly imposing a limit on its maximum depth; (ii) set Θ as the leaves $\bar{\gamma}_j$ of the grown tree $\hat{M}_{LPV}$.

Compared to using a clustering technique like K-means [107], [108], the use of DTRs does not require selecting a fixed number of clusters a priori and also actively takes into account the relation between ρ and γ. In fact, with the proposed DTR-based approach, the regression tree will not grow in regions where ρ is not informative enough about γ, therefore aggregating a possibly large set of values of γ with the same representative leaf-value $\bar{\gamma}_j$. This latter aspect is enabling for our ultimate goal of exploiting the resulting set of models to build a bank of linear observers.

Once the set $\Theta^\rho = \{\bar{\gamma}_j\}_{j=1}^{N_\theta}$ of local ARX models has been selected,

each of them is converted into a corresponding minimal state-space representation in observer canonical form [109]

$$\Sigma_j := \left\{ \begin{array}{rcl} \xi^j_{k+1} & = & A_j \xi^j_k + B_j u_k + d_j \\ y_k & = & C_j \xi^j_k + e_j \end{array} \right. \qquad j = 1, \ldots, N_\theta \tag{3.6}$$

As all the vectors $\bar{\gamma}_j$ have the same dimension n^γ, we assume that all states ξ^j have the same dimension v. The models Σ_j in (3.6) are used to design a corresponding linear observer, as described next.

3.3.2 Design of the observer bank

For each model Σ_j, we want to design an observer providing an estimate $\hat{\xi}^j_k$ of the state ξ^j_k of Σ_j. Let $i^j_k \in R^v$ be the *information vector* generated by the observer at time k, which includes $\hat{\xi}^j_k$ and possibly other quantities, such as the covariance of the output and state estimation error in the case of time-varying Kalman filters are used. As the goal is to use i^j_k, together with u_k, y_k, to estimate ρ_k, it is important to correctly tune the observers associated with the N_θ models in Θ to ensure that each i^j_k is meaningful. For example, a slower observer may be more robust against measurement noise, but its "inertia" in reacting to changes may compromise the capabilities of the resulting virtual sensor.

The computational burden introduced by the observers also needs to be considered. As it will be necessary to run the full bank of N_θ observers in parallel in real-time, a viable option is to use the standard Luenberger observer [110]

$$\left\{ \begin{array}{rcl} \hat{\xi}^j_{k+1} = & A_j \hat{\xi}^j_k + d_j + B_j u_k - L_j(\hat{y}^j_k - y_k) \\ \hat{y}^j_k = & C_j \hat{\xi}^j_k + e_j \end{array} \right. \tag{3.7}$$

where L_j is the observer gain, and set $i^j_k = \hat{\xi}^j_k$. Since minimal state-space realizations are used to define Σ_j, each pair (A_j, C_j) is fully observable, and the eigenvalues of $A_j - L_j C_j$ can arbitrarily be placed inside the unit circle. Note also that any technique for choosing L_j can be employed here, such as stationary Kalman filtering.

3.3.3 A model-free hypothesis testing algorithm

After the N_θ observers have been synthesized, we now build a hypothesis testing scheme based on them using a *discriminative* approach [111]. To this end, the initial dataset $\mathcal{D}$ is processed to generate the information vectors i_k^j, $k \in [1, N]$. Let $D_{\text{aug}} \triangleq \{i_k^1, \ldots, i_k^{N_\theta}, u_k, y_k, \rho_k\}$, $k = 1, \ldots, N$, denote the resulting augmented dataset that will be used to train a predictor $f_\theta : \mathbb{R}^v \times \ldots \times \mathbb{R}^v \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}^S$ such that

$$\hat{\rho}_k = f_\theta(i_k^1, \ldots, i_{k-\ell}^1, \ldots, i_k^{N_\theta}, \ldots, i_{k-\ell}^{N_\theta}, u_k, y_k) \tag{3.8}$$

is a good estimate of ρ_k, where $\ell \geq 0$ is a window size to be calibrated. Consider the minimization of a loss function $L : \mathbb{R}^S \times \mathbb{R}^S \to \mathbb{R}$ that penalizes the distance between the measured value ρ_k and its reconstructed value $\hat{\rho}_k$, namely a solution of

$$\min_\theta \sum_{k=\ell+1}^{N} L(\rho_k, f_\theta(i_k^1, \ldots, i_{k-\ell}^{N_\theta}, u_k, y_k)) \tag{3.9}$$

Solving the optimization problem (3.9) directly, however, may be excessively complex, as no additional knowledge about the relation between ρ_k and $\{i_k^1, \ldots, i_{k-\ell}^{N_\theta}, u_k, y_k\}$ is taken into account. In order to model such a relation, one can rewrite f_θ as the concatenation of two maps g_θ and e_θ^{FE} such that

$$\begin{array}{rcl} \hat{\rho}_k & = & g_\theta(\mathcal{I}_k) \\ \mathcal{I}_k & = & e_\theta^{\text{FE}}(i_k^1, i_{k-\ell}^1, \ldots, i_k^{N_\theta}, \ldots, i_{k-\ell}^{N_\theta}, u_k, y_k) \end{array} \tag{3.10}$$

where $\mathcal{I}_k$ is a *feature vector* constructed by a given feature extraction (FE) map $e_\theta^{\text{FE}} : \mathbb{R}^v \times \ldots \times \mathbb{R}^v \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_I}$ from i_k^1, $i_{k-\ell}^1$, …, $i_k^{N_\theta}$, …, $i_{k-\ell}^{N_\theta}$, u_k, and y_k, and $g_\theta : \mathbb{R}^{n_I} \to \mathbb{R}^S$ is the prediction function to learn from the dataset D_{aug}.

We propose the following two alternatives for the FE map, namely

$$e_\theta^{\text{FE}}(\mathcal{I}_k) = \{\hat{e}_k^1, \hat{e}_{k-\ell}^1, \ldots, \hat{e}_k^{N_\theta}, \ldots, \hat{e}_{k-\ell}^{N_\theta}, u_k, y_k\} \tag{3.11a}$$

where $\hat{e}_m^j \triangleq (\hat{y}_m^j - y_m)$ and, to further reduce the number of features, the more aggressive and higher compression FE map

$$e_\theta^{\text{FE}}(\mathcal{I}_k) = \{\nu_k^1, \ldots, \nu_k^{N_\theta}, u_k, y_k\} \tag{3.11b}$$

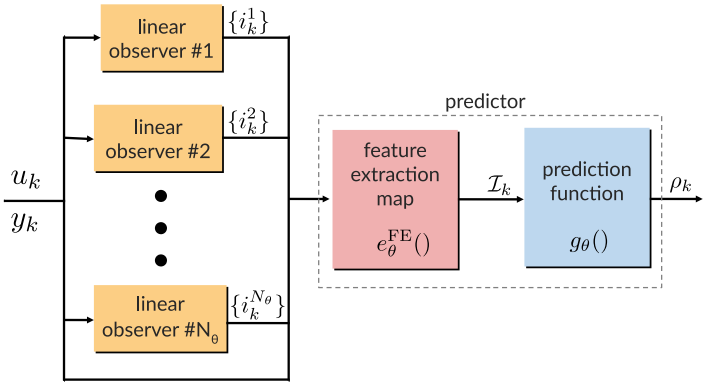


Figure 4: Virtual sensor architecture: bank of linear observers, feature extraction map e_θ^{FE}, and prediction function g_θ.

where

$$\nu_k^i = \sqrt{\frac{1}{\ell+1}\sum_{r=k-\ell}^{k} m(r-k)(\hat{y}_r^i - y_r)'(\hat{y}_r^i - y_r)}$$

and $m : \mathbb{Z} \to \mathbb{R}$ is an appropriate weighting function.

The rationale for the maps in (3.11) is that one of the most common features used in hypothesis testing algorithms is the estimate of the covariance of the residuals produced by each observer. Thus, this approach can be thus considered a generalization of the window-based hypothesis-testing algorithms explored in the literature, such as in [104], [112]. The FE map (3.11b) brings this idea one step further so that $e_\theta^{\mathrm{FE}}(\mathcal{I}_k)$ has only $n_I = (N_\theta + 1)n_y + n_u$ components. This means that the input to g_θ, and therefore the predictor itself, can get very compact.

Note that both the window-size ℓ and the weighting function m are hyper-parameters of the proposed approach. In particular, the value of ℓ must be chosen carefully: if it is too small the time window of past output prediction errors may not be long enough for a slow observer. On the other hand, if ℓ is too large the virtual sensor may become exces-

sively slow in detecting changes of ρ_k. The weighting function m acts in a similar fashion and can be used to further tune the behavior of the predictor.

Note also that our choices for e_θ^{FE} in (3.11) are not the only possible ones, nor necessarily the optimal ones. For example, by setting $e_\theta^{\mathrm{FE}}(\mathcal{I}_k) = \mathcal{I}_k$, and therefore $f_\theta = g_\theta$, one recovers the general case in (3.9). Finally, note that our analysis has been restricted to a pre-assigned function e_θ^{FE}, although this could also be learned from data. To this end, we refer the interested reader to [51], [57], [113] and the references therein.

Choice of learning techniques

As highlighted in [13], in order to target an embedded implementation, it is necessary to envision a learning architecture for g_θ that has a limited memory footprint and requires a small and well predictable throughput. To do so, instead of developing an application-specific functional approximation scheme, we resort to well-understood machine-learning techniques. In particular, three possible options are explored in this Chapter, all well suited for our purposes and which require a number of floating-point operations (flops) for their evaluation which is independent of the number of samples used in the training phase, in contrast for example to K-nearest neighbor regression [3].

Remark: Such choices are not the only possible ones and other approaches may be better suitable for specific needs. For example, if one is interested in getting an uncertainty measure coupled to the predictions, the use of regression techniques based on Gaussian processes [114] could be more suitable.

Compact artificial neural networks

Artificial neural networks (ANN) are a widely used machine-learning technique that has already shown its effectiveness in other MMAE-based schemes [115]. An option to make ANN very lightweight is to resort on very compact feed-forward topologies comprised of a small number of layers and a computationally cheap activation function in their hidden

neurons [116], such as the Rectified Linear Unit (ReLU) [79]

$$f_{\mathrm{ReLU}}(x) = \max\{0, x\} \tag{3.12}$$

As we want to predict real-valued quantities, we consider a linear activation function for the output layer of the network.

Decision-tree and random-forest regression

DTRs of limited depth are in general extremely cheap to evaluate yet offer a good approximation power [23]. Other advantages of DTRs are that they can also work effectively with non-normalized data, they can be well interpreted [117], and the contribution provided by each input feature is easily recognizable. The main disadvantage of DTRs is instead that they can suffer from high variance. For this reason, in this work, we also explore the use of random-forest regressors (RFRs) [118], which try to solve the issue by bagging together multiple trees at the cost of both a more problematic interpretation and higher computational requirements.

3.3.4 Hyper-parameters and tuning procedures

The overall architecture of the proposed virtual sensor is shown in Figure 4. Its main hyper-parameters are:

1. the number N_θ of local models to learn from experimental data, related to the number of leaves of the DTR (see Section 3.3.1);
2. the order M of the local models (see Section 3.3.1);
3. the window size ℓ of the predictor, which sets the number of past and current input features provided to the predictor at each time to produce the estimate $\hat{\rho}_k$ (see Section 3.3.3).

From a practical point of view, tuning M is relatively easy, as one can use as the optimal cost reached by solving (3.5) an indirect performance indicator to properly trade-off between the quality of fit and storage constraints. Feature selection approaches such as the one presented in [119]

can also be used. The window size ℓ of the predictor is also easily tunable by using any feature selection method compatible with the chosen regression technique. A more interesting problem is choosing the correct number of local models N_θ, especially if one considers that MMAE-like schemes often do not perform well if too many models are considered [120]. Finally, we mention that the proposed method also requires defining the feature extraction map and the predictor structure.

As with most black-box approaches, and considering the very mild assumption we made on on the system Σ_P that generates the data, the robustness of the virtual sensor with respect to noise and other sources of uncertainty can only be assessed *a posteriori*. For this reason, Section 3.4.2 below reports a thorough experimental analysis to assess such robustness properties.

3.4 Numerical results

In this section, we explore the performance of the proposed virtual sensor approach on a series of benchmark problems. All tests were performed on a PC equipped with an Intel Core i7 4770k CPU and 16 GB of RAM. The models introduced in this section were used *only* to generate the training datasets and to test the virtual sensor and are *totally unknown* to the learning method. We report such models to facilitate reproducing the numerical results reported in this section.

3.4.1 Learning setup

All the ANNs involved in learning the virtual sensors were developed in Python using the `Keras` framework [80] and are composed of 3 layers (2 ReLU layers with an equal number of neurons and an linear output layer), with overall 60 hidden neurons. The ANNs are trained using the AMSgrad optimization algorithm [81]. During training, 5% of the training set is reserved to evaluate the stopping criteria.

Both DTR and RFR are trained using `scikit-learn` [121] and, in both cases, the max depth of the trees is capped to 15. RFRs consist of

10 base classifiers. The DTR used to extract the set of local models, as shown in Section 3.3.1, is instead constrained to have a maximum number of leaves equal to the number N_θ of linear models considered in each test. The loss function $L_{M_{LPV}}$ used is the well known mean absolute error [60]. In all other cases, the standard mean-squared error (MSE) [122] is considered.

The performance of the overall virtual sensor is assessed on the testing dataset in terms of the fit ratio (FIT) already used in the previous Chapter, and normalized root mean-square error (NRMSE)

$$\text{FIT} = \max\left\{0, 1 - \frac{\|\rho_{\mathcal{T}} - \hat{\rho}_{\mathcal{T}}\|_2}{\|\bar{\rho} - \rho_{\mathcal{T}}\|_2}\right\} \tag{3.13a}$$

$$\text{NRMSE} = \max\left\{0, 1 - \frac{\|\rho_{\mathcal{T}} - \hat{\rho}_{\mathcal{T}}\|_2}{\sqrt{\mathcal{T}}|\max(\rho_{\mathcal{T}}) - \min(\rho_{\mathcal{T}})|}\right\} \tag{3.13b}$$

computed component-wise, where $\bar{\rho}$ is the mean value of the test sequence $\rho_{\mathcal{T}} = \{\rho_i\}_{i=1}^{\mathcal{T}}$ of the true values and $\hat{\rho}_{\mathcal{T}}$ is its estimate.

For each examined test case, we report the mean value and standard deviation of the two figures in (3.13) over ten different runs, each one involving different realizations of all the excitation signals u_k, p_k, and measurement noise.

3.4.2 A synthetic benchmark system

We first explore the performance of the proposed approach and analyze the effect of its hyper-parameters on a synthetic multi-input single-output benchmark problem. Consider the nonlinear time-varying system

$$\Sigma_{\text{S}} = \begin{cases} x_{k+1} &= Hx_k + \frac{\alpha}{2}\operatorname{atan}(x_k) + \log(\rho_k + 1)Fu_k \\ \rho_{k+1} &= h(\rho_k, u_k, k) \\ y_k &= -(1 + e^{\rho_k})\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} x_k \end{cases} \tag{3.14a}$$

where $x \in \mathbb{R}^5$, atan is the arc-tangent element-wise operator, $\alpha, \rho_k \in \mathbb{R}$, matrices H and F are defined as

$$H = \begin{bmatrix} 0.0 & 0.1 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ -0.00909 & 0.0329 & 0.29013 & -1.05376 & 1.69967 \end{bmatrix} \tag{3.14b}$$

$$F = \begin{bmatrix} -0.71985 & -0.1985 \\ 0.57661 & 0.917661 \\ 1.68733 & -0.68733 \\ -2.14341 & 2.94341 \\ 1. & 1. \end{bmatrix}$$

and function h is defined by

$$h(\rho_k, u_k, k) = \begin{cases} p_k \text{ if } p_k \in [-0.95, 0.95] \\ \frac{p_k}{2} \text{ otherwise} \end{cases} \tag{3.14c}$$

$$p_k = 0.999\rho_k + 0.03\omega_k, \quad \omega_k \sim \mathcal{N}(0,1) \tag{3.14d}$$

mimicking the phenomenon of a slow parameter drift. Unless otherwise stated, in the following we consider $\alpha = 1$.

Training datasets of various sizes (up to $25,000$ samples) and a dataset of $5,000$ testing samples are generated by exciting the benchmark system (3.14) with a zero-mean white Gaussian noise input u_k with unit standard deviation. All signals are then normalized using the empirical average and standard deviation computed on the training set and superimposed with a zero-mean white Gaussian noise with a standard deviation of 0.03 to simulate measurement noise.

We consider local ARX linear models involving past $M = 5$ inputs and outputs and solve Problem (3.5) via a fully connected feed-forward ANN. In the feature extraction process, we set the window $\ell = 6$ on which the feature extraction process operates. In all tests we also assume $m(i) \equiv 1$, $\forall i \in \mathbb{Z}$. Unless otherwise noted, we consider deadbeat Luenberger observers, i.e., we place the observer poles in $z = 0$ using the `Scipy` package [84].

No. of acquired samples N	5000	15000	25000
	FE map (3.11b)		
average FIT (3.13a)	0.711	0.783	0.796
standard deviation	0.062	0.028	0.023
average NRMSE (3.13b)	0.937	0.954	0.956
standard deviation	0.014	0.004	0.002
	FE map (3.11a)		
average FIT (3.13a)	0.695	0.773	0.794
standard deviation	0.072	0.030	0.028
average NRMSE (3.13b)	0.934	0.952	0.956
standard deviation	0.016	0.003	0.002

Table 5: Accuracy of the virtual sensor using datasets of different size K.

3.4.3 Dependence on the number N of samples

We analyze the performance obtained by the synthesized virtual sensor with respect to the number N of samples acquired for training during the experimental phase. Assessing the scalability of the approach with respect to the size of the dataset is extremely interesting because most machine learning techniques, and in particular neural networks, often require a large number of samples to be effectively trained.

Table 5 shows the results obtained by training the sensor with various dataset sizes when $N_\theta = 5$ observers and an ANN predictor are used both using the proposed high-compression FE map (3.11b) and using the map in (3.11a). It is apparent that good results can already be obtained with 15,000 samples. With smaller datasets, fit performance instead remarkably degrades, especially when using the less aggressive FE map (3.11a).

3.4.4 Robustness toward measurement noise

We analyze next the performance of the proposed approach in the presence of various levels of measurement noise. In particular, we test the capabilities of the virtual sensor with $N_\theta = 5$ deadbeat observers when trained and tested using data obtained from 3.14a and corrupted with

a zero-mean additive Gaussian noise with different values of standard deviation σ_N. As in the other tests, noise is applied to the signal after normalization. The training dataset contains $25,000$ samples.

The results, reported in Table 6, show a very similar trend for all three functional approximation techniques and, in particular, a very steep drop in performance when moving from $\sigma_N = 0.03$ to $\sigma_N = 0.06$. This fact suggests that a good signal-to-noise ratio is necessary to achieve good performance with the proposed approach. This finding is not surprising, as our method is entirely data-driven, it has more difficulties in filtering noise out compared to model-based methods.

Performance of ANN, RFR, and DTR is similar to what observed in the previous tests.

Predictor	Standard deviation σ_N of additive noise		
	0.01	0.03	0.06
DTR	0.752 (0.026)	0.736 (0.031)	0.698 (0.036)
RFR	0.804 (0.021)	0.787 (0.023)	0.755 (0.029)
ANN	0.815 (0.018)	0.796 (0.023)	0.764 (0.032)

Table 6: Average FIT (3.13a) (standard deviation) for the three proposed learning architectures different sensor noise intensity.

3.4.5 Dependence on the prediction function

We analyze the difference in performance between the three proposed learning models for function g_θ when using $N_\theta = 5$ linear models. The corresponding results are reported in Table 7, where it is apparent that as soon as enough samples are available, both RFRs and ANNs essentially perform the same, especially when using the more aggressive FE map (3.11b). For smaller training datasets, the ANN-based predictor performs slightly better, especially in terms of variance. Regression trees show worst performance but they are still able to produce acceptable estimates.

Regarding the results obtained using the FE map (3.11a), ANNs are remarkably more effective than the other two methods. In particular,

while RFRs still show acceptable performance, DTRs fail almost completely.

Predictor	FE map	Number N of acquired samples		
		5000	15000	25000
DTR	(3.11b)	0.593 (0.161)	0.713 (0.041)	0.736 (0.031)
RFR		0.663 (0.145)	0.766 (0.033)	0.787 (0.023)
ANN		0.711 (0.062)	0.783 (0.028)	0.796 (0.023)
DTR	(3.11a)	0.376 (0.158)	0.568 (0.061)	0.615 (0.042)
RFR		0.568 (0.168)	0.715 (0.046)	0.740 (0.024)
ANN		0.695 (0.072)	0.773 (0.030)	0.794 (0.028)

Table 7: Average FIT (3.13a) (standard deviation) for the three proposed learning architectures for different numbers K of samples in the training dataset.

3.4.6 Dependence on the observer dynamics

The dynamics of state-estimation errors heavily depend on the location of the observer poles set by the Luenberger observer (3.7), such as due to the chosen covariance matrices in the case stationary KFs are used for observer design. In this section, we analyze the sensitivity of the performance achieved by the virtual sensor with respect to the chosen settings of the observer.

Using $N_\theta = 5$ models again, the Luenberger observers were tuned to have their poles all in the same location $z \in \mathbb{C}$ inside the unit disk and vary such a location in different tests. In addition, we also consider stationary Kalman filters designed assuming the following model

$$\begin{cases} \xi_{k+1}^j &= A_j \xi_k + B_j u_k + d_j + w_k \\ y_k^j &= C_j \xi_k + e_j + v_k \end{cases} \tag{3.15}$$

where $w_k \sim \mathcal{N}(0, I)$ and $v_k \sim \mathcal{N}(0, \lambda I)$ are uncorrelated white noise signals of appropriate dimensions and $\lambda \geq 0$.

The resulting virtual sensing performance figures are reported in Table 8 for a training dataset of $N = 25,000$ samples and RFR-based prediction. While performance is satisfactory in all cases, fast observer poles

allow better performance when pole placement is used. Nevertheless, in all but the deadbeat case, KFs provide better performance regardless of the chosen covariance term λ.

	Pole placement		
Observer settings	$z = 0.0$	$z = 0.4$	$z = 0.8$
average FIT (3.13a)	0.787	0.708	0.467
standard deviation	0.023	0.030	0.053
average NRMSE (3.13b)	0.954	0.937	0.886
standard deviation	0.002	0.003	0.005
	Kalman filter		
Observer settings	$\lambda = 1$	$\lambda = 10$	$\lambda = 0.1$
average FIT (3.13a)	0.785	0.777	0.787
standard deviation	0.021	0.022	0.024
average NRMSE (3.13b)	0.954	0.952	0.954
standard deviation	0.002	0.002	0.002

Table 8: Average prediction performance with respect to observer settings.

3.4.7 Dependence on the number N_θ of local models

To explore how sensitive the virtual sensor is with respect to the number N_θ of local LTI model/observer pairs employed, we consider the performance obtained using $N_\theta = 2, 3, 4, 5, 7$ local models on the 25,000 sample dataset. The results obtained using RFR based virtual sensors are reported in Table 9 and show that performance quickly degrades if too few local models are employed. At the same time, one can also note that a large number of models is not necessarily more effective. This finding suggests that the proposed virtual sensor can be easily tuned by increasing the number of models until the accuracy reaches a plateau.

Figure 5 shows the estimates of ρ_k obtained by the virtual sensor for $N_\theta = 5$, using deadbeat observers and a RFR predictor, for a given realization of (3.14d).

N_θ	2	3	5	7
average FIT (3.13a)	0.684	0.781	0.787	0.793
standard deviation	0.033	0.023	0.023	0.024
average NRMSE (3.13b)	0.932	0.953	0.954	0.956
standard deviation	0.005	0.001	0.002	0.001

Table 9: Prediction performance of the virtual sensor with respect to the number N_θ of LTI models.

3.4.8 Dependence on the dynamics of ρ_k

Let us consider a different model than (3.14c)–(3.14d) to generate the value of p_k that defines the signal ρ_k, namely the deterministic model

$$p_k = \cos\left(\frac{1}{\beta}k\right) \tag{3.16}$$

with $\beta = 200$. In this way, it is possible to test the effectiveness of our approach in a purely parameter-varying setting and its robustness against discrepancies between the way training data and testing data are generated.

The results reported in Tables 10 and 11 are obtained by using $N_\theta = 5$ models and 25,000 samples using all the three different prediction functions and deadbeat observers. While (3.16) is used to generate both training and testing data to produce the results shown in Table 10, Table 11 shows the case in which the training dataset is generated by (3.14c)–(3.14d) while the testing dataset by (3.16). It is apparent that the proposed approach is able to work effectively also in the investigated parameter-varying context in all cases and able to cope with sudden changes of ρ_k.

3.4.9 A mode observer for switching linear systems

An interesting class of systems which can be described by (3.1) are linear switching systems [123], a class of linear parameter varying systems in which ρ_k can only assume a finite number s of values $\rho^1, \ldots, \rho^s$. In this case, model (3.1) becomes the following discrete-time switching linear

Predictor	DTR	RFR	ANN
average FIT (3.13a)	0.842	0.876	0.873
standard deviation	0.005	0.004	0.003
average NRMSE (3.13b)	0.953	0.963	0.962
standard deviation	0.001	0.001	0.001

Table 10: Average accuracy of the virtual sensor employing various kind of prediction functions when both training and testing data are generated by using (3.16).

system

$$\Sigma \triangleq \begin{cases} x_{k+1} &= A_{\rho_k} x_k + B_{\rho_k} u_k \\ y_k &= C_{\rho_k} x_k \end{cases} \tag{3.17}$$

For switching systems, the problem of estimating ρ_k from input/output measurements is also known as the *mode-reconstruction* problem. In order to test our virtual sensor approach for mode reconstruction, we let the system generating the data be a switching linear system with $s = 4$ modes obtained by the scheduling signal

$$\rho_{k+1} = \frac{1}{2} \left\lfloor \frac{4k}{N} \right\rfloor \tag{3.18}$$

where N is the number of samples collected in the experiment and $\lfloor \cdot \rfloor$ is the downward rounding operator. For this test, the measurements about the mode acquired during the experiment are noise-free. As we are focusing on linear switching systems, we set $\alpha = 0$.

As in Section 3.4.7, we test the performance of the RFR-based virtual sensor trained on 25,000 samples to reconstruct the value of ρ_k when

Predictor	DTR	RFR	ANN
average FIT (3.13a)	0.753	0.801	0.844
standard deviation	0.066	0.046	0.009
average NRMSE (3.13b)	0.924	0.939	0.952
standard deviation	0.020	0.014	0.003

Table 11: Average accuracy of the virtual sensor for different prediction functions with training data generated from (3.14d) and testing data from (3.16).

equipped with a different number N_θ of local models. The corresponding results are reported in Table 12 and show that the performance of the sensor again quickly saturates once the number of local models matches the actual number of switching modes.

N_θ	2	3	4	5
average FIT (3.13a)	0.800	0.938	0.951	0.950
standard deviation	0.030	0.014	0.007	0.009
average NRMSE (3.13b)	0.925	0.977	0.982	0.981
standard deviation	0.011	0.005	0.003	0.003

Table 12: Accuracy of the virtual sensor employing different predictors for the switching linear system in (3.18).

The time evolution of the actual mode and the mode reconstructed by the virtual sensor is shown in Figure 6.

Performance obtained using a classifier in place of a regressor

The special case of mode reconstruction for switching systems can be also cast as a multi-category classification problem. Table 13 reports the F1-score [124] obtained by applying a virtual sensor based on a Random Forest Classifier (RFC) and 5 deadbeat observers to discern the current mode of the system. We consider only the case of samples correctly labeled, with the RFC subject to the same depth limitation of the non-categorical hypothesis tester. Table 13 also reports the classification accuracy of the non-categorical virtual sensor when coupled with a minimum-distance classifier (i.e., at each time k the classifier will predict the mode i associated with the value ρ_i that is closest to $\hat{\rho}_k$). The results refer to a virtual sensor equipped with RFR and 5 deadbeat observers. It is interesting to note that the classifier architecture is also very effective with respect to the FIT metric (3.13a), achieving an average score of 0.946 with with a standard deviation of 0.011.

The time evolution of the actual mode and the mode reconstructed by the classifier-based virtual sensor is shown in Figure 7.

F1-score / mode #	1	2	3	4
RFC	0.997	0.995	0.996	0.998
standard deviation	0.001	0.002	0.001	0.001
RFR	0.997	0.995	0.996	0.998
standard deviation	0.001	0.001	0.001	0.002

Table 13: F1-score [124] obtained by the RFC-based virtual sensor (RFC) and by the RFR-based virtual sensor + minimum-distance classifier (RFR) on the 4-mode switching linear system (3.18) over 10 runs.

3.4.10 Nonlinear state estimation

This sections compares the proposed approach with standard model-based nonlinear state-estimation techniques on the problem of estimating the state of charge (SoC) of a lithium-ion battery, using the model proposed in [125].

In [125], the battery is modeled as the following nonlinear third-order dynamical system

$$\Sigma_{\text{Battery}} = \begin{cases} \dot{x}_1(t) = \dfrac{-i(t)}{C_c} \\ \dot{x}_2(t) = \dfrac{-x_2(t)}{R_{ts}(x_1(t))C_{ts}(x_1(t))} + \dfrac{i(t)}{C_{ts}(x_1(t))} \\ \dot{x}_3(t) = \dfrac{-x_3(t)}{R_{tl}(x_1(t))C_{tl}(x_1(t))} + \dfrac{i(t)}{C_{tl}(x_1(t))} \\ y(t) = E_0(x_1(t)) - x_2(t) - x_3(t) - i(t)R_s(x_1(t)) \end{cases} \tag{3.19}$$

where $x_1(t)$ is the SoC (pure number $\in [0, 1]$ representing the fraction of the battery rated capacity that is available [126]), $y(t)$ [V] the voltage at the terminal of the battery, $i(t)$ [A] the current flowing through the battery,

$$\begin{aligned} E_0(x_1) &= -a_1 e^{-a_2 x_1} + a_3 + a_4 x_1 - a_5 x_1^2 + a_6 x_1^3 \\ R_{ts}(x_1) &= a_7 e^{-a_8 x_1} + a_9 \\ R_{tl}(x_1) &= a_{10} e^{-a_{11} x_1} + a_{12} \\ C_{ts}(x_1) &= -a_{13} e^{-a_{14} x_1} + a_{15} \\ C_{tl}(x_1) &= -a_{16} e^{-a_{17} x_1} + a_{18} \\ R_s(x_1) &= a_{19} e^{-a_{20} x_1} + a_{21} \end{aligned}$$

and the values of the coefficients a_{ij} correspond to the estimated values

reported in Tables 1, 2, 3 of [125].

We analyze the capability of the proposed synthesis method of virtual sensors to reconstruct the value $\rho = x_1$ in comparison to a standard extended Kalman filter (EKF) [127] based on model (3.19) and assuming the process noise vector $w_k \in \mathbb{R}^3$ entering the state equation, $w_k \sim \mathcal{N}(0, Q)$, and measurement noise $v_k \perp w_k \sim \mathcal{N}(0, R)$ on the output y for various realization of $R \in \mathbb{R}^{3\times 3}, Q \in \mathbb{R}$. Model (3.19) is integrated by using an explicit Runge-Kutta 4 scheme.

The simulated system is sampled at the frequency $f_s = \frac{1}{5}$ Hz, starting from a fully charged state $x(0) = [1,\ 0,\ 0]'$ and excited with a variable-step current signal $i(t)$ with constant amplitude

$$i(t) = \frac{1}{5} \max \left\{ 0, \cos\left(\frac{k}{100}\right) + \cos\left(\frac{k}{37}\right) \right\} + u_k \tag{3.20}$$

during the sampling step k, with u_k drawn from the uniform distribution $\mathcal{U}(0, 0.4)$.

As the battery will eventually fully discharge, every time the SoC falls below the value 0.05 the whole state is reset to the initial condition $x(0)$. The signals $y(t)$ and $i(t)$, once normalized, are processed as described in Section 3.4.1.

For this benchmark, an RFR-based virtual sensor with $N_\theta = 5$ local linear models is selected. The corresponding KFs are also designed as described in Section 3.4.6 with $\lambda = 0.1$, and FE map (3.11b). Training is performed over 25,000 samples.

The results obtained by the virtual sensor and EKFs designed with different values of the covariance matrices Q, R of process and measurement noise, respectively, are reported in Figure 8. While EKF is, in general, more effective in tracking and denoising the true value of the SoC, it performs poorly in terms of bandwidth compared to the proposed virtual sensor, whose performance in terms of filtering noise out remains anyway acceptable. While both techniques are successful in estimating the SoC of the battery, we remark a main difference between them: EKF *requires* a nonlinear model of the battery, the virtual sensor *does not*.

3.4.11 Complexity of the prediction functions

The ANNs used in our tests require approximately between 1,000 and 3,000 weights to be fully parameterized. While this number is fixed by the network topology and depends on the number of inputs to the network, regularization techniques such as ℓ_1-norm sparsifiers could be used here to reduce the number of nonzero weights (see for instance [66] and the reference therein), so to further reduce memory footprint.

Regarding the tree-based approaches, practical storage requirements are strongly influenced by the specific implementation and, in general, less predictable in advance due to their non-parametric nature. In any case, evaluating the prediction functions on the entire test set on the reference machine only requires a few tens of milliseconds, which makes the approach amenable for implementation in most modern embedded platforms.

The training procedure for all the proposed architectures is similarly affordable: on the reference machine, the whole training process is carried out in a few tens of seconds for a training set of 25,000 samples with negligible RAM occupancy.

3.5 Conclusions

In this chapter we proposed a data-driven virtual sensor synthesis approach, inspired by the MMAE framework, for reconstructing normally unmeasurable quantities such as scheduling parameters in parameter-varying systems, hidden modes in switching systems, and states of nonlinear systems.

The key idea is to use past input and output data (obtained when such quantities were directly measurable) to synthesize a bank of linear observers and use them as a base for feature-extraction maps that greatly simplify the learning process of the hypothesis testing algorithm that estimates said parameters. Thanks to its low memory and CPU requirements, the overall architecture is particularly suitable for embedded and fast-sampling applications.

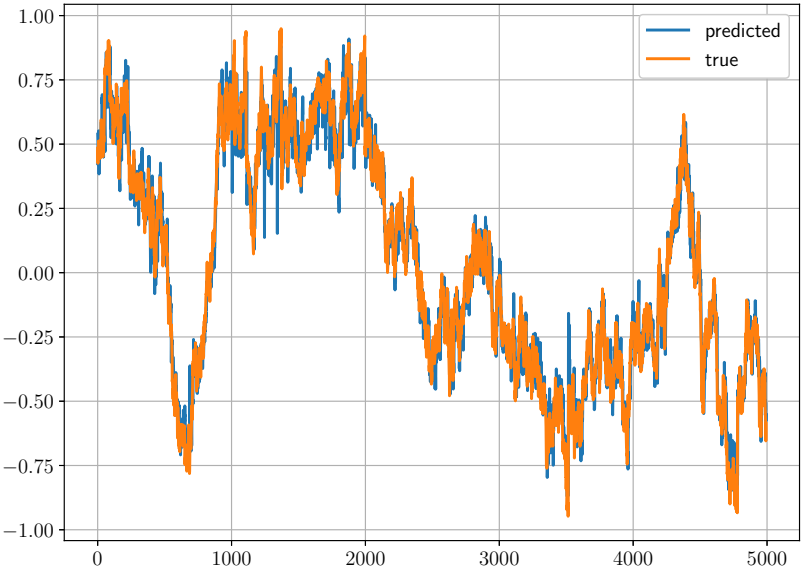


Figure 5: Example of reconstruction of ρ_k by the virtual sensor based on $N_\theta = 5$ local models, using deadbeat observers and a RFR predictor. The figure reports the actual value of ρ_k (orange line) and its estimate $\hat{\rho}_k$ (blue line).

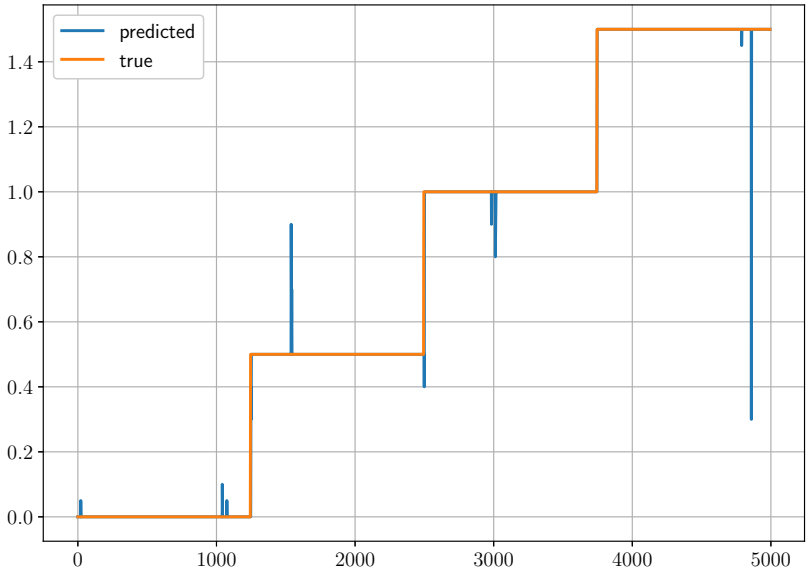


Figure 6: Mode reconstruction for switching linear systems (3.17): actual value of the mode ρ_k (orange line) and its estimate $\hat{\rho}_k$ (blue line) provided by a RFR-based virtual sensor with a bank of 5 deadbeat observers.

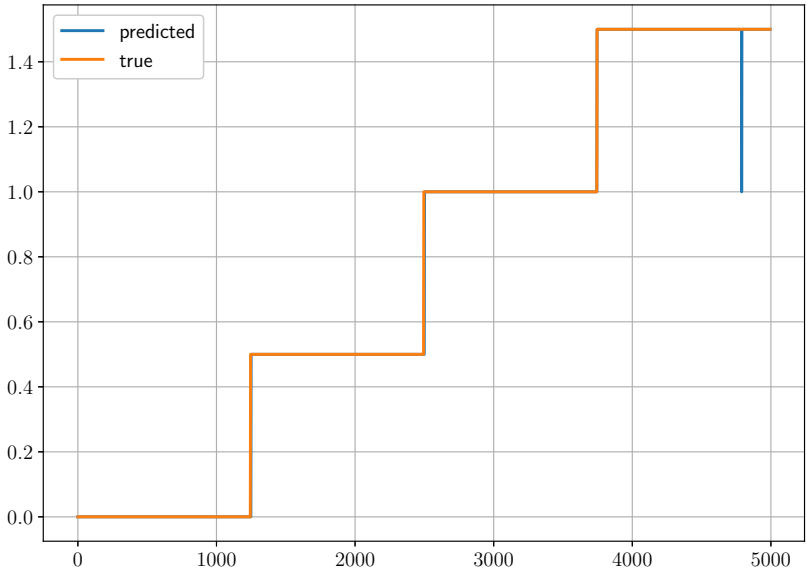


Figure 7: Mode reconstruction for switching linear systems (3.17): actual value of the mode ρ_k (orange line) and its estimate $\hat{\rho}_k$ (blue line) provided by a RFC-based virtual sensor and 5 deadbeat observers.

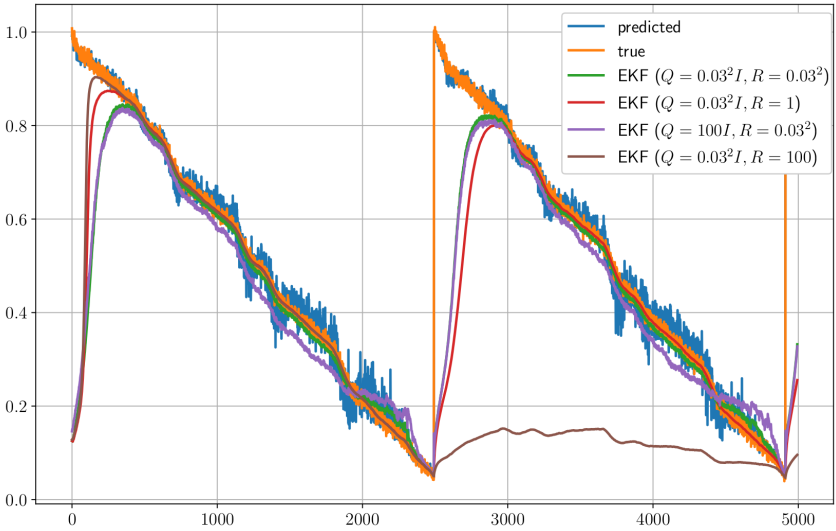


Figure 8: Estimation of the SoC of the battery: true value ρ_k (orange line), value $\hat{\rho}_k$ estimated by the virtual sensor (blue line), values $\hat{\rho}_k$ estimated by EKF for different settings of Q and R (green, red, violet, and brown lines).

Chapter 4

Learning affine predictors for MPC of nonlinear systems via artificial neural networks

4.1 Introduction

Designing a control law directly from experimental data has always been a topic of central interest in the control systems community. The classical approach is to first identify a model of the open-loop process via system identification [1] from experimental data, validate it, and then proceed with the design of a model-based controller. The main advantage of this approach is that the identification and control design phases are decoupled. However, unnecessary burden due to excessively accurate system identification procedures might occur. In fact, many control design techniques do not need a very accurate model of the target process to synthesize the control law: for instance when designing a Model Predictive Controller (MPC) [128] there is no need of having a model that predicts accurately the evolution of the output beyond the prediction horizon. In other words, getting the best possible accuracy of the model in fitting

the available input/output data is often not required for good closed-loop performance. In addition, an accurate fit often comes at the price of an excessive model complexity, that in turn makes model-based control laws more complicate (or even impossible) to synthesize.

The separation between system identification and model-based control design is also clear from the fact that, while in recent years many state-of-the-art system identification techniques focus on learning input-output models (see [26], [28], [129]), most of the existing modern control and filtering techniques are based on state-space models. Many authors have tried to bridge this gap, such as the technique presented in Chapter 2, with the goal of synthesizing state-feedback controllers such as nonlinear MPC controllers.

Recently, *model-free* data-driven techniques were proposed to completely bypass the phase of first identifying an open-loop model of the process, such as using direct controller synthesis methods [130], machine learning [131]–[133], policy-search [11], and reinforcement learning [12]. While very attractive in practice, the main drawbacks of model-free methods are the need for large amounts of data to synthesize optimal control laws and the different way one tunes the controller compared to model-based control approaches.

For these reasons, we explore a different way of approaching data-driven control design, tailoring the system identification procedure to the particular use one wants to make of the resulting model for control design. In particular, to handle nonlinear control problems within the linear time-varying (LTV) MPC framework, such as the real-time iteration scheme [71], we explore how to fit the entire output prediction for LTV-MPC over a horizon of T steps as a nonlinear function of the current state (or, equivalently, past input/output pairs) and as an affine function of future control moves. In this way, the MPC problem can be solved via quadratic programming (QP) in spite of the nonlinearity of the system.

This approach has been explored in various forms by many authors but, as a whole, the literature on the topic is both scarse and scattered. In [134], [135], the authors proposed to learn fixed-horizon affine models via regression trees (RT) and random forests (RF) for nonlinear systems;

in [136] an approach based on set-membership (SM) identification was devised to learn robust predictor for linear systems. The authors in [137] addressed instead the problem of building an adaptive control scheme in which fixed-horizon predictors based on Radial-Basis-Function (RBF) Artificial Neural Networks (ANNs) were learned online. Another comparable approach is the one shown in [133], where the authors used an ANN to identify the dynamics of the system and setup an associated MPC problem. However, the resulting ANN model is nonlinear, so the MPC problem requires a more complex nonlinear solution method, while our approach requires only a quadratic programming solver, or even admits a simple analytic solution in the absence of constraints.

In particular, we investigate if ANN-based solutions can be competitive with traditional machine learning approaches to learn fixed short-term horizon predictors for receding horizon control applications. To this end, we perform an in-depth comparison between the RT/RF-based approaches presented in [138], [139] and modern reinterpretation of the idea presented in [137]. In particular, with respect to this latter work, we focus on replacing the original non-parametric single-layer RBF-ANN architecture with a specifically tailored state-of-the-art deep-learning technique and introduce a completely offline learning procedure in place of the original online learning scheme.

The rest of the chapter is organized as it follows. After formulating the problem in Section 4.2, in Section 4.3 we present a specifically deep-learning based parametrization to learn a fixed-horizon affine model of a non-linear process. In Section 4.4 we provide a background on the methodology in [135], that will be used in Section 4.5 to compare the proposed approach via numerical simulations. Section 4.6 will provide guidelines for tuning the proposed approach in practical problems, with attention to reducing its computational requirements.

4.2 Problem formulation

Assume that we have collected data from a process described by the following (usually unknown) nonlinear discrete-time dynamical model

$$\Sigma = \begin{cases} x_{k+1} = f_\Sigma(x_k, u_k) \\ y_k = h_\Sigma(x_k) \end{cases} \tag{4.1}$$

where $k \in \mathbb{Z}$ is the sampling step, $x_k \in \mathbb{R}^{n_x}$ the state of the system, $y_k \in \mathbb{R}^{n_y}$ the output vector, $u_k \in \mathbb{R}^{n_u}$ the input vector, and $f_\Sigma : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$, $h_\Sigma : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$. In case there is no clear definition of what the state vector x_k should be, due for example to physical insights, we assume that

$$x_k = \begin{bmatrix} y_k' & \cdots & y_{k-\delta_y}' & u_{k-1}' & \cdots & u_{k-\delta_u}' \end{bmatrix}' \tag{4.2}$$

for some model-order integers $\delta_y, \delta_u \geq 0$.

With the goal of synthesizing an MPC controller with prediction horizon $T \in \mathbb{Z}$, we want to learn a mapping from x_k and $u_k, u_{k+1}, \ldots, u_{k+j-1}$ to y_{k+j}, where $j = 1, \ldots, T$ is the prediction step, from a dataset $\mathcal{D} = \{(y_k, u_k, x_k)\}_{k=0}^{N-1}$ collected from Σ, or simply $\mathcal{D} = \{(y_k, u_k)\}_{k=0}^{N-1}$ in case x_k just collects past inputs and outputs as in (4.2).

In order to avoid the recursive modeling approach commonly used in the literature (see for instance [133]), we focus ourselves on creating T predictors that depend only on the available state measurements/estimations and on the inputs to be optimized. More precisely, we want to identify maps $h_j : \mathbb{R}^{n_x} \times \mathbb{R}^{jn_u} \to \mathbb{R}^{n_y}$, $j = 1, \ldots, T$, each one taking x_k and $u_k, \ldots, u_{k+j-1}$ as inputs, such that

$$y_{k+j} = h_j(x_k, u_k, \ldots, u_{k+j-1}), \ \forall\, j = 1, \ldots, T. \tag{4.3}$$

We can rewrite (4.3) in the more compact form

$$Y_T = \begin{bmatrix} y_{k+1} \\ \vdots \\ y_{k+T} \end{bmatrix} = H_T(x_k, U_{T-1}), \tag{4.4}$$

where $H_T : \mathbb{R}^{n_x} \times \mathbb{R}^{Tn_u} \to \mathbb{R}^{Tn_y}$ has a block-triangular structure due to (4.3) and $U_{T-1} = [u_k' \ \ldots \ u_{k+T-1}']'$.

Consider now the following MPC problem at each step k

$$\begin{aligned}
\min_{U_{T-1}} \quad & (Y_T - R_T)' W_Q (Y_T - R_T) + \\
& (U_{T-1} - U^R_{T-1})' W_R (U_{T-1} - U^R_{T-1}) + \delta U' \, W_S \, \delta U \\
\text{s.t. :} \quad & \\
& Y_T = H_T(x_k, U_{T-1}) \\
& Y_{\min} \leq Y_T \leq Y_{\max} \\
& U_{\min} \leq U_{T-1} \leq U_{\max} \\
& \delta U_{\min} \leq \delta U_{T-1} \leq \delta U_{\max}
\end{aligned} \tag{4.5}$$

where W_Q, W_R, W_S are weight matrices of appropriate dimensions, $R_T \in \mathbb{R}^{Tn_y}$ is the reference signal to track, $U^R_T \in \mathbb{R}^{Tn_u}$ is a corresponding input reference (possibly $U^R_T = 0$), $\delta U = [u'_k - u'_{k-1}, \ \ldots, u'_{k+T-1} - u'_{k+T-2}]'$, and $Y_{\min}, Y_{\max}$, $U_{\min}, U_{\max}$, $\delta U_{\min}, \delta U_{\max}$ are vectors of lower and upper bounds.

Problem (4.5) can be recast as a QP problem if we fit T ANN-based *affine* predictors of the following form

$$Y_T = F_T(x_k, \bar{U}_{T-1}) + G_T(x_k, \bar{U}_{T-1}) U_{T-1} \tag{4.6}$$

where $\bar{U}_{T-1}$ is a nominal input sequence, $F_T : \mathbb{R}^{n_x} \times \mathbb{R}^{Tn_u} \to \mathbb{R}^{Tn_y}$ and $G_T : \mathbb{R}^{n_x} \times \mathbb{R}^{Tn_u} \to \mathbb{R}^{Tn_y} \times \mathbb{R}^{Tn_u}$.

The affine predictor (4.6) can be interpreted also as the first-order Taylor approximation of the non-linear mapping in (4.4). In fact, assuming that H_T is at least of class $\mathcal{C}^1$, we get

$$\begin{aligned}
H_T(x_k, U_{T-1}) \approx & H_T(x_k, \bar{U}_{T-1}) \\
& + \sum_{j=0}^{T-1} \frac{\partial H_T}{\partial u_{k+j}} (x_k, \bar{U}_{T-1})(u_{k+j} - \bar{u}_{k+j}).
\end{aligned}$$

A simplified form of (4.6), obtained by fixing $\bar{U}_{T-1}$ a priori (for example, equal to a steady-state nominal input, or zero), is the following predictor

$$Y_T = F_T(x_k) + G_T(x_k)(U_{T-1} - \bar{U}_{T-1}) \tag{4.7}$$

with $F_T : \mathbb{R}^{n_x} \to \mathbb{R}^{Tn_y}$ and $G_T : \mathbb{R}^{n_x} \to \mathbb{R}^{Tn_y} \times \mathbb{R}^{Tn_u}$.

4.3 Training affine predictors via ANNs

Given that the underlying system (4.1) generating the output signals y_{t+k} is Markovian, the components f_j, g_j of F_T, G_T are not completely independent functions between different prediction steps. Therefore, to regularize the training procedure we impose the following recursive and causal structure

$$y_{k+j} = f_j(x_k) + g_j(x_k) \begin{bmatrix} U_{j-1} - \bar{U}_{j-1} \\ y_{k+j-1} \end{bmatrix} \tag{4.8}$$

for $j = 2, \dots, T$. This is especially useful if we use ANN techniques [66] or deep kernel machines [140], that naturally benefit from stacking nonlinear layers/components.

Note that the affine nature of the predictor is maintained in (4.8), as the composition of affine functions remains affine. To simplify the notation, in the rest of the Chapter we will consider $\bar{U}_{T-1} \equiv 0$.

The problem of learning the maps f_j and g_j in (4.8) can be posed as the following optimization problem

$$\begin{aligned} &\underset{\{f_j, g_j\}_{i=1}^{T}}{\arg\min} \quad \sum_{k=\max\{\delta_y, \delta_u\}}^{N-T-1} L(\hat{O}_k, O_k) \\ &\text{s.t.:} \\ &\qquad \hat{y}_{k+j} = f_j(x_k) + g_j(x_k)\, [U'_{j-1},\ \hat{y}'_{k+j-1}]' \\ &\qquad \hat{y}_{k+1} = f_1(x_k) + g_1(x_k) U_0 \\ &\qquad O_k = [y_{k+1} \ \cdots \ y_{k+T}]' \\ &\qquad \hat{O}_k = \left[\hat{y}_{k+1} \ \cdots \ \hat{y}_{k+T}\right]' \\ &\qquad j = 2, \dots, T, \end{aligned} \tag{4.9}$$

where the constraints in (4.9) impose the causal structure arising from (4.8) and $U_{j-1} = [u'_k, \dots, u'_{k+j-1}]'$. The loss function $L : \mathbb{R}^{Tn_y} \times \mathbb{R}^{Tn_y} \to \mathbb{R}$ can be any loss function and can be chosen based on the physical meaning of the predicted output, for example using a cosine distance for an angular quantity or a cross entropy for a categorical output signal.

The optimization problem (4.9) has two conflicting objectives, due to trading off between short-term prediction accuracy and the ability to

carry useful information from prediction $k+j$ to prediction $k+i$, for $i > j$. Note that learning the maps f_i, g_i, for $i = 1, \dots, T$ can be either carried out sequentially (one time step j at time) or in one shot. In this Chapter we focus on the latter method.

The unconventional structure (4.8) restricts the pool of function approximators that can be employed to parameterize f_j and g_j. We exploit the nature of ANNs of being direct acyclic computational graphs to build into the topology of the network itself the constraints structure of (4.9). In this way, we reduce the learning procedure to a regression problem while retaining the capability of easily accessing the outputs of f_i, g_i as intermediate results of the single sub-components of the network. A schematic of the considered network is reported in Figure 9.

Choosing ANNs is also convenient for their flexibility to be trained with a wide class of loss functions (which allows one to easily add, for instance, regularizers or additional objectives) and to their theoretical property of being universal approximators [58], as well as to the large availability of well-maintained and mature frameworks for their training [6], [7].

ANN structure. The topology of the ANN used here closely follows the structure highlighted in (4.8). In particular, all the components f_i are grouped in a single stand-alone network, while each component g_i is instead a separate entity. In this work, we restrict our analysis to a densely connected feed-forward topology for each subnetwork, but in principle this is not mandatory. Each sub-network is thus composed by a series of nonlinear hidden layers and a final linear output layer with appropriate output shape. As we analyze output signals in $\mathbb{R}^{Tn_y}$, we rely on the well known mean absolute error (MAE) [60] figure as the loss function L.

4.4 Switching affine RT and RF predictors

In this section we briefly recall the approach proposed in [138], recently experimentally validated in [141], to create a switching affine modeling framework based on regression trees (RTs) and random forests (RFs), that

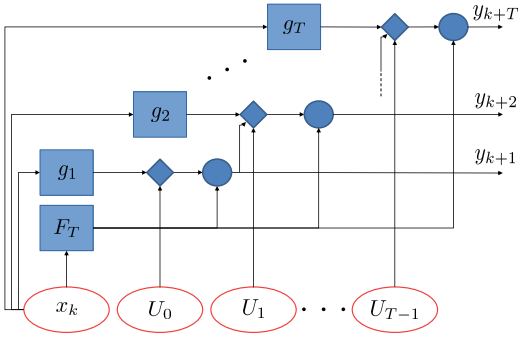


Figure 9: ANN structure for predictions affine in the input.

can be used in the MPC formulation (4.5). For simplifying reading, we limit the discussion to scalar outputs ($n_y = 1$), although the approach can be easily extended for $n_y > 1$ as illustrated in [138].

We partition the original dataset $\mathcal{D}$ in 2 disjoint sets: $\mathcal{D}_c = \{u_k\}_{k=1}^N$, of data associated with input variables, and $\mathcal{D}_{nc} = \{x_k\}_{k=1}^N$ related to the remaining variables. As done in Section 4.3, the idea is to create T different predictors to predict y_{k+j}, and to derive a modeling framework in order to setup an MPC problem that leads to a QP.

We create T regression trees $\mathcal{T}_j,\ j = 1, \ldots, T$, by applying the CART algorithm to the dataset $\mathcal{D}_{nc}$ (see [138], [142] for more details). In particular, for each tree $\mathcal{T}_j$, the CART algorithm partitions the set $\mathcal{D}_{nc}$ into subsets $\mathcal{D}_{nc,i,j},\ i = 1, \ldots, |\mathcal{T}_j|,\ j = 1, \ldots, T$, where $|\mathcal{T}_j|$ is the number of regions of the partition associated with $\mathcal{T}_j$ (i.e., the number of leaves of $\mathcal{T}_j$). Then, using the control data in $\mathcal{D}_c$, we associate to each leaf i of each tree $\mathcal{T}_j$, corresponding to the partition $\mathcal{D}_{nc,i,j}$, the following affine model

$$x_{k+j} = A_{i_j} x_k + \sum_{\alpha=0}^{j-1} B_{i_j,\alpha} u_{k+\alpha} + f_{i_j},\ \forall i_j,\ \forall j, \tag{4.10}$$

where matrices A'_{i_j}, $B'_{i_j,\alpha}$ and f'_{i_j} are in turn structured as

$$A_{i_j} = \begin{bmatrix} a_1 & a_2 & \cdots & a_{\delta_y+1+\delta_u} \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \quad f_{i_j} = \begin{bmatrix} f \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.11}$$

$$B_{i_j,\alpha} = \begin{bmatrix} b_{1,\alpha} & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ b_{n_u,\alpha} & 0 & \cdots & 0 \end{bmatrix}'.$$

The coefficients of matrices A_{i_j}, $B_{i_j,\alpha}$ and f_{i_j} are obtained by fitting the set of samples

$$\{(x_{k+j}, u_k, \ldots, u_{k+j-1}) : \; x_k \in \mathcal{D}_{nc,i,j}\} \tag{4.12}$$

via least squares as defined in Problem 2 of [138]. In particular, in order to consider the same information as in (4.8), we will constrain the parameters related to the input auto-regressive terms to be zero, i.e. a_{δ_y+2}, $\ldots, a_{\delta_y+1+\delta_u} = 0$.

The approach discussed above for RTs can be easily extended to the case of RFs [143]. The idea behind RFs is to extend the RT concept by growing multiple trees, considering different random subsets of the dataset to train each tree. The prediction is given by averaging the response of all the trees in the forest. At the price of an increased computational burden, the pros of this approach are the reduction of the overall variance of the prediction error and the mitigation of overfitting.

In our context, we create T RFs $\mathcal{F}_j$, $j = 1, \ldots, T$, with $|\mathcal{F}_j|$ the number of trees of forest $\mathcal{F}_j$. We can derive a model as in (4.10) by solving the least squares problem introduced above for each leaf $\mathcal{D}_{i_j,\tau}$ of each tree $\mathcal{T}_\tau$, $\tau = 1, \ldots, |\mathcal{F}_j|$, of each forest $\mathcal{F}_j$. In this way, we obtain a vector of parameters for the matrices in (4.11) associated to each leaf $\mathcal{D}_{i_j,\tau}$. The final model as in (4.10) can be obtained for each forest $\mathcal{F}_j$ by averaging the coefficients of all the matrices associated with $\mathcal{D}_{i_j,\tau}$.

It can be shown that for both RT and RF models (4.10) is a switching affine representation of (4.1), where the switching signal follows the

partitioning imposed by the tree structures, and each leaf represents an operating mode (see [138] for details).

4.5 Simulation results

4.5.1 Benchmark problem setup

Let the system generating the data be the, as in the previous Chapters, following discrete-time approximation of the well-known nonlinear two-tank system [73]

$$\begin{cases} x_{1,k+1} = x_{1,k} - k_1\sqrt{x_{1,k}} + k_2 u_k \\ x_{2,k+1} = x_{2,k} + k_3\sqrt{x_{1,k}} - k_4\sqrt{x_{2,k}} \\ y_k = x_{2,k} \end{cases} \tag{4.13}$$

with $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, and $k_4 = 0.3$ and where $x_{i,k}$ represents the i-th component of $x_k \in \mathbb{R}^2_+$. On this system we analyze the performance of the affine model over an horizon of $T = 10$. To do so, we collect a training dataset $\mathcal{D}$ of $N = 10,000$ samples by exciting (4.13) with a sequence of step signals of length 7 steps, each of random amplitude extracted from the univariate Gaussian distribution $G_u \backsim \mathcal{N}(\mu_u, \sigma_u^2)$, with $\mu_u = \sigma_u = 2$. The testing dataset is generated similarly and consist of $1,000$ samples. Both dataset have been normalized using the empirical mean and standard deviation computed on the training set. White zero-mean Gaussian noise with $\sigma_w = 0.02$ is superimposed on input/output signals.

Since we employ an early-stopping strategy as termination criterion for the training process of the ANN, 5% of the training dataset is used as a validation set to check the stopping criterion. Each computational node of the ANN is composed of two rectified linear units (ReLU) [79] layers with 20 neurons each, followed by a final linear output layer.

As state x_k we choose past input/output values as in (4.2) with $\delta_y = \delta_u = 6$. The predictors are written in Python using Keras [80] with Tensorflow as back-end, using AMSgrad [81] for optimizing the weights. The total training procedure was carried out in around a minute on a

Intel Core i5-6200U CPU machine with 16GB of RAM and required a negligible amount of memory.

4.5.2 Fitting performance

We first investigate the accuracy of the affine predictors as in (4.8) on the test set for the prediction horizon $T = 10$. Fitting performance over the horizon is computed in terms of FIT and NRMSE figures also used in the previous Chapters, defined as follows:

$$e_{\text{FIT}} = \max\left\{0, 1 - \frac{\|\hat{y} - y\|_2}{\|y - \bar{y}\|_2}\right\} \tag{4.14}$$

$$e_{\text{NRMSE}} = \max\left\{0, 1 - \frac{\|\hat{y} - y\|_2}{\sqrt{\mathcal{S}_{\mathcal{T}}}(\max(y) - \min(y))}\right\} \tag{4.15}$$

where $\hat{y}$ is the vector stacking the estimates of the true values vector y, $\bar{y}$ is the mean of y, and $\mathcal{S}_{\mathcal{T}}$ is the number of elements in y.

The results, reported in Table 14, show a very good prediction capability, that clearly decreases over the horizon as expected. This behavior can be also linked to the specific affine form of the predictors that is not able to correctly reproduce the nonlinear effect of past inputs on future outputs. This is not a severe limitation, due to the receding-horizon mechanism of MPC.

4.5.3 Performance comparison between ANN, RT, and RF

In this section we provide a comparison with the methodology introduced in [138] that we recalled in Section 4.4. Using data generated through the benchmark example (4.13), we built a model as in (4.8) as shown in Sections 4.3 and 4.5.1, and model (4.10), considering both RT and RF approaches. Validation results on the test set of 900 samples are shown in Figure 10 and Figure 11, where we compare the predicted trajectories over the horizon at $k + 1$ and $k + 10$.

Figure 12 reports e_{NRMSE} over the horizon for the three approaches. It is apparent that on the considered benchmark problem ANN and RF performance are overall quite close, and both outperform RT.

Prediction step	e_{FIT}	e_{NRMSE}
1	0.957	0.99
2	0.957	0.99
3	0.95	0.989
4	0.948	0.988
5	0.94	0.986
6	0.928	0.983
7	0.914	0.98
8	0.9	0.977
9	0.88	0.972
10	0.858	0.967

Table 14: Accuracy of the affine ANN predictors for the benchmark (4.13)

4.5.4 Evaluating MPC closed-loop performance

We explore now the effect of using the learned affine predictors in a LTV-MPC control law in closed loop with (4.13). The LTV-MPC controller solves the QP problem (4.5) at each step, with H_T affine in the inputs as defined in (4.7). We set $W_Q = 1, W_R = W_S = 0.01, U^R_{T-1} = 0$ and impose the constraint $-1.5 \leq u_{k+j} \leq 1.5, \forall j = 0, \ldots, T-1$ with $T = 5$. Quantities are in normalized units. The resulting performance of the controller in tracking a unit step superimposed to a sine sweep signal is reported in Figure 13. Despite predictions are only approximate, the controller provides satisfactory closed-loop performance. Regarding computation

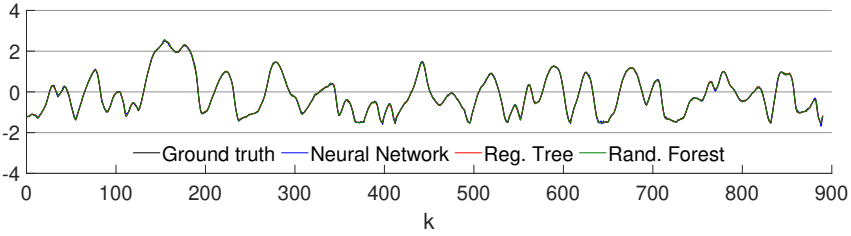


Figure 10: Output prediction at $k + 1$

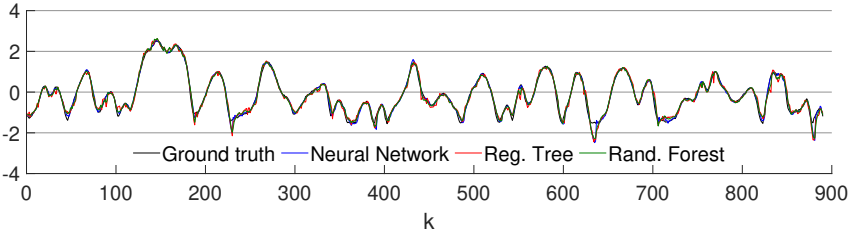


Figure 11: Output prediction at $k + 10$

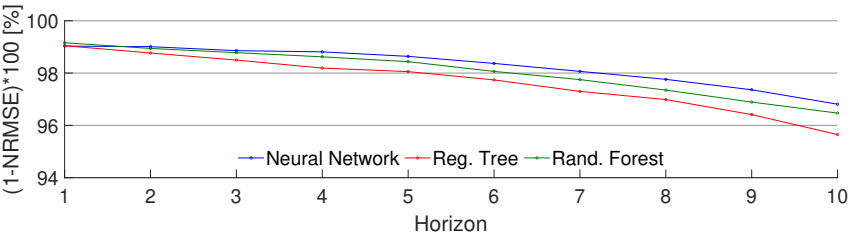


Figure 12: Normalized root mean square error (NRMSE) over the prediction horizon

time, the evaluation of the affine predictors G_T and F_T and solving the QP problem (carried out by the general purpose solver based on [144]) required ≤ 0.05 seconds on the same reference machine. Since the constructed LTV-MPC problem maps into a box-constrained least-squares problem, the efficient solution method proposed in [85] could be used here for example to speed up computations even further.

4.6 Complexity reduction

In the example presented in the Section 4.5.2, the ANN has $\approx 9,000$ parameters. Evaluating the network over the whole test set on the same machine used for training takes less than a second in total for the whole prediction horizon. For comparison, we fitted a nonlinear autoregressive model with exogenous inputs (NLARX) with comparable prediction

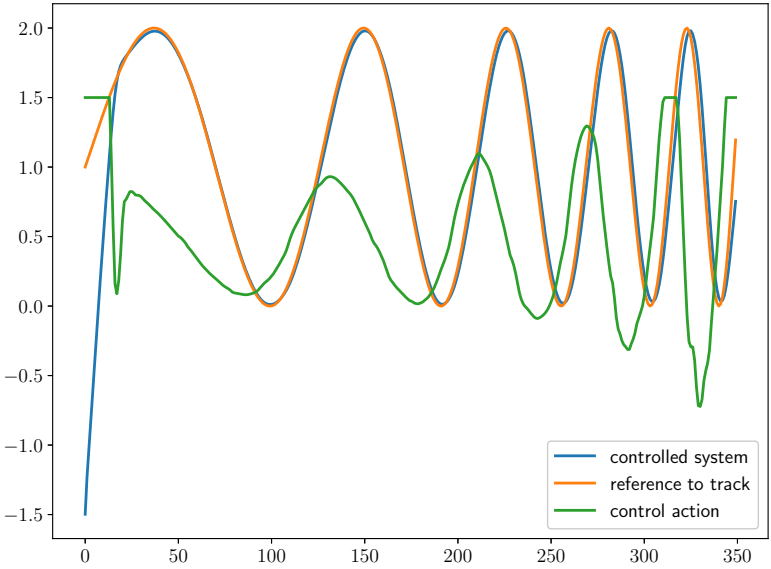


Figure 13: Illustrative example of the performance of the LTV MPC for system (4.13) using ANN-based affine models via dynamic parametrization. x axis is for time steps, y is for the magnitude of the signals. All signals are in normalized units

performance, composed by two hidden layers NNs (each one composed of 20 neurons), by using MATLAB System Identification Toolbox [145], [146]. Such NLARX model requires less than $\frac{1}{10}$ of the coefficients of the affine ANN predictor. This is not surprising, as we fit an entire horizon of predictions rather than a recursive model, and because neither in the design of the ANNs nor in the training process we took any action aimed at reducing the number of network parameters. Although storage requirement is already quite small in our approach, we discus next how to reduce memory occupancy of the ANN predictors, and therefore of the resulting MPC setup.

4.6.1 Memory occupancy vs. quality of fit tradeoff

It is well known in the literature that the addition of $\mathcal{L}_1$-penalties (a.k.a. the *shrinkage operator*) in an learning problem induces sparse solutions [67]. When the optimization problem arises from fitting a model, $\mathcal{L}_1$-penalties also reduces possible overfitting issues.

Accordingly, to reduce the memory occupancy of the resulting ANNs, we modify the cost function in problem (4.9) as follows:

$$\underset{\{f_j,g_j\}_{k=1}^{T}}{\arg\min} \sum_{k=\max\{\delta_y,\delta_u\}}^{N-T-1} L(\hat{O}_k, O_k) + \lambda L_1(\Theta) \tag{4.16}$$

where Θ is the overall set of non-bias weights $\theta_1^f, \theta_1^g, \ldots, \theta_T^f, \theta_T^g$ characterizing the ANNs associated with the predictors f_j and g_j, $j = 1, \ldots, T$, respectively, and

$$L_1(\Theta) = \sum_{j=1}^{T} \|\theta_j^f\|_1 + \|\theta_j^g\|_1 \tag{4.17}$$

The scalar hyper-parameter $\lambda \geq 0$, decides the tradeoff between quality of fit and number of nonzero weights. Table 15 reports a realization of quality of fit, computed over all $T = 10$ steps, and of number of coefficients with absolute value $\geq 10^{-3}$ for different choices of λ. In this test all the remaining coefficients not satisfying such condition were manually set to zero after the training process for all but the $\lambda = 0$ case.

λ	e_{FIT}	NZ weights
0.01	0.853	325
0.005	0.864	350
0.001	0.901	560
0.0005	0.911	902
0	0.917	9001

Table 15: Illustrative example of the number of nonzero (NZ) weights and prediction fit obtained for different choices of λ.

4.7 Conclusions

We developed an approach for learning an affine parameterization of output predictions from data via artificial neural networks, conceived for efficiently formulating and solving LTV-MPC problems for nonlinear systems. We showed in numerical simulations that good performance, both in terms of capturing the dynamics of a nonlinear process and of closed-loop behavior, is achieved, with light computational load. We also showed that memory occupancy of the solution can be traded off with prediction accuracy by simply introducing $\mathcal{L}_1$-penalties during the training phase.

Experimental tests on embedded platforms on real application use cases will be performed in future work.

Chapter 5

Direct data-driven design of neural reference governors

5.1 Introduction

Designing effective controllers without knowing the physics of the plant has always been a topic of interest in the control system community. The indirect approach usually employed in this case is to first identify an open-loop model of the underlying system via system identification procedures like the one presented in Chapter 2, validate the learned model, and then proceed with the design and the tuning of a model-based controller. While this strategy is convenient, as it effectively introduces a decoupling layer between modeling and control design, it can also cause an undue burden on the system identification step, since many control techniques do not actually require a fully-featured model to synthesize an appropriate control law. Although exceptions exist (as we discussed in Chapter 4 or in [135], [136], [147] in the context of predictive controllers), this means that there exists a large misalignment between the target of most identification techniques, which aim for the best open-loop simulation accuracy, and the actual requirements of control design.

This discrepancy has stimulated research into *direct data-driven control* techniques [148], [149], that rely on input-output data to directly syn-

thesize and tune a control law, without first identifying an open-loop model and/or predictor of the behavior of the plant. Within this setting, approaches exist that rely either on the real-time acquisition of data (like in Reinforcement Learning [12]), or exploiting in batch mode a previously collected dataset [150]. Among the latter class of approaches, we recall the *Virtual Reference Feedback Tuning* (VRFT) method, which has been applied successfully to control linear and nonlinear dynamical systems [151]–[153]. Within the VRFT framework, the design of the control law is recast into an identification problem, where the aim is to reproduce the collected input sequence via a *virtual tracking error* specially crafted so that the attained closed-loop behavior matches the response of a user-provided reference model. Nonetheless, off-the-shelf architectures of this kind do not allow taking signal constraints into account. For this reason, several extensions have thus been proposed to overcome this limitation. For instance, in [154] a Q-learning approach is used to enhance a baseline VRFT controller; in [155] a method to deal with constraints on the sensitivity function is proposed in the context of $\mathcal{H}_\infty$ control. Nonetheless, to obtain the desired results, all such methods are no longer one-shot learning procedures as the VRFT approach, and they can deal only with specific kinds of constraints. A more general approach is presented in [130], where a reference governor [156] is used on top of a standard VRFT-based controller to ensure constraints satisfaction and to boost closed-loop performance. This approach is still a one-shot data-acquisition procedure, but it also seeks to synthesize an internal control capable of working by itself, which is a possibly non-trivial and likely non-needed task if the controller will be always used with the governor.

In this work, we focus on overcoming the latter limitation while still envisioning a data-driven receding-horizon control solution that can guarantee constraints satisfaction, based on the idea of [130]. To do so, we resort to ideas taken from control-oriented system identification and learn short-term fixed-horizon models inspired to the one presented in Chapter 4 to obtain a simplified linear time-varying representation of the implicit controller.

The rest of the Chapter is organized as follows. In Section 5.2 we for-

mally state the problem of learning both the external receding-horizon reference governor and the inner controller. Section 5.3 is devoted to the presentation of the proposed controller design approach. Numerical results are reported in Section 5.4 and some concluding remarks are drawn in Section 5.5.

5.2 Setting and goals

Consider the following discrete-time dynamical system

$$\Sigma_P = \begin{cases} x_{k+1} = f_\Sigma(x_k, u_k) \\ y_k = h_\Sigma(x_k) \end{cases} \tag{5.1}$$

with $u_k \in \mathbb{R}^{N_U}$, $y_k \in \mathbb{R}^{N_Y}$, and f_Σ, h_Σ being two *unknown* nonlinear functions.

Suppose that a set $Z_\mathcal{N} = \{(u_1, y_1), \ldots, (u_\mathcal{N}, y_\mathcal{N})\}$ comprising $\mathcal{N}$ input/output pairs acquired from Σ_P is available.

Our first aim is to exploit the available dataset $Z_\mathcal{N}$ to learn a controller C_θ, a-priori parameterized by a vector θ of parameters, so that the reference-to-output behavior of the resulting closed-loop system corresponds to the input-output behavior of a given reference model $\mathcal{M}$. We describe the latter by a the possibly nonlinear mapping $f_\mathcal{M}$

$$y^\mathcal{M}_{k+1} = f_\mathcal{M}(y^\mathcal{M}_k, \ldots, y^\mathcal{M}_{k-M}, r_k, r_{k-M}) \tag{5.2}$$

with $y^\mathcal{M}_k \in \mathbb{R}^{N_Y}$ and $r_k \in \mathbb{R}^{N_Y}$. Our second and concurrent goal is to design an outer loop with a model predictive reference governor, so to boost performance and handle signal constraints [156].

Recalling the VRFT framework, let us define as *virtual reference* the sequence $\{r^\star_k\}$, $r^\star_k \in \mathbb{R}^{N_Y}$, that would reproduce $\{y_k\}$ if fed to $\mathcal{M}$, namely the minimizer of the optimization problem

$$\begin{aligned} \{r^\star_k\} = \arg\min_{\{r_k\}} \quad & \sum_{k=M+2}^{\mathcal{N}} LR(y^\mathcal{M}_k, y_k) \\ \text{subject to} \quad & \text{Equation (5.2)} \end{aligned} \tag{5.3}$$

where $LR : \mathbb{R}^{N_Y} \times \mathbb{R}^{N_Y} \to \mathbb{R}$ is an appropriate loss function. We stress that many approaches have been devised to solve problem (5.3), and that explicit exact recursive expressions for $\{r_k^\star\}$ have been found for various classes of reference models $\mathcal{M}$, see *e.g.*, [152].

Once $\{r_k^\star\}$ has been computed, one way to synthesize the controller C_θ that achieves closed-loop performance closest to the one of $\mathcal{M}$ is to look for the controller that best fits $\{u_k\}$ when fed by the *virtual tracking error* $e_k^\star \triangleq r_k^\star - y_k$:

$$\begin{aligned} \min_{C_\theta \in \mathcal{H}} \quad & \sum_{k=M+2}^{\mathcal{N}-1} LC(\hat{u}_k, u_k) \\ \text{subject to} \quad & \hat{u}_{k+1} = C_\theta(u_k, \dots, u_{k-M}, e_{k+1}^\star, \dots, e_{k-M}^\star) \end{aligned} \tag{5.4}$$

where $LC : \mathbb{R}^{N_U} \times \mathbb{R}^{N_U} \to \mathbb{R}$ is again an appropriate loss function. The above *model-free* VRFT technique was first introduced in [151] in the linear time-invariant setting, and then successfully employed to design controller for various classes of nonlinear systems (see *e.g.*, [152]).

Albeit appealing due to its simple nature, solving a one-step-ahead prediction-error minimization problem like the one in (5.4) can be problematic. Indeed, we cannot guarantee that the resulting control law may be suited to be repeatedly iterated over time, although accurate over a short horizon. Various approaches have been designed to overcome this issue: in [130], for instance, instrumental variables are used. Still, they usually result in a tangible increase of the computational complexity of the overall learning scheme [157]. While this is unavoidable if the aim is to synthesize a stand-alone controller C_θ, we will show here that *this is not the case if our goal is to use C_θ within a reference governor scheme*.

Indeed, as described in [130], a reference governor for an already existing VRFT controller consists of solving at each sampling step k the

following optimization problem:

$$
\begin{aligned}
\min_{R_k} \quad & LRG(Y_k, U_k, R_k, \bar{Y}) \\
\text{s.t.} \quad & Y_k = \mathbf{F}^{\mathcal{M}}(\bar{x}_k, R_k) \left\{ \begin{array}{l} \text{Fixed horizon} \\ \text{closed loop} \\ \text{reference dynamics} \end{array} \right. \\
& U_k = \mathbf{C}(\bar{x}_k, E_k) \left\{ \begin{array}{l} \text{Fixed horizon VRFT} \\ \text{controller dynamics} \end{array} \right. \\
& R_k = [r_k, \ldots, r_{k+T}]' \in \mathcal{R} \\
& U_k = [u_k, \ldots, u_{k+T}]' \in \mathcal{U} \\
& Y_k = [y_{k+1}, \ldots, y_{k+T}]' \in \mathcal{Y} \\
& E_k = [r_k - y_k, \ldots, r_{k+T} - y_{k+T}]'
\end{aligned}
\tag{5.5}
$$

where $\bar{Y} \in \mathbb{R}^{N_Y T}$ is the reference to be tracked, T is the chosen prediction horizon, LRG is a suitable loss function, and $\bar{x}_k$ is a feedback information vector, which encompasses the internal states of both the controller and the underlying system. By looking at the optimization problem in (5.5), it is clear that the performance of the governor is linked to the prediction capabilities of $\mathbf{F}^{\mathcal{M}}$ and the capability of $\mathbf{C}$ to make the internal loop behave as closely as possible to $\mathbf{F}^{\mathcal{M}}$ itself *within the chosen prediction horizon*.

This means that if we do not want to use the internal controller alone, we can directly learn the mapping $\mathbf{C}$ from $\bar{x}_k$ and R_k to U_k, rather than solving (5.4).

In other words, there is no need to seek a standalone viable internal controller, but only a "surrogate" able to predict its short-term behavior.

Remark 1 *Without loss of generality, LRG may also depend on Y_k and U_k to further shape the overall characteristics of the closed loop.*

5.3 Data-driven design of reference governors

Based on the above intuition, we consider the problem of fitting a fixed horizon predictor of the real VRFT controller $\mathbf{C}$ introduced in (5.5). At time k and over the horizon T, the value of the signals $\hat{u}_{k+n}, n = 0, \ldots, T$, are only a function of the virtual tracking error $e^\star_k, \ldots, e^\star_{k+n}$ and the initial

state x_k, *i.e.*, there exists a set of maps f_n so that

$$\hat{u}_{k+n} = f_n(x_k, e_k^\star, \dots, e_{k+n}^\star) \text{ for } n = 0, \dots, T \tag{5.6}$$

for each $T \in \mathbb{Z}$. Without loss of generality, we can set

$$x_k = [u_{k-1}, \dots, u_{k-M}, e_{k-1}^\star, \dots, e_{k-M+1}^\star]$$

and define, similarly to what shown in Chapter 4, the map $H_T : \mathbb{R}^{N_x} \times N_Y(T+1) \to \mathbb{R}^{N_U(T+1)}$ compactly as

$$\hat{U}_T \triangleq \begin{bmatrix} \hat{u}_k \\ \vdots \\ \hat{u}_{k+T} \end{bmatrix} = H_T(x_k, E_k^T) \tag{5.7}$$

where $E_k^T \triangleq [e_k^\star, \dots, e_{k+T}^\star]'$.

Compared to learning a recursive law C_θ as in (5.4), retrieving H_T in (5.7) from data is more convenient as (i) it completely circumvents the problem of using a fitting procedure able to minimize the *simulation error* of C_θ, and (ii) it allows us to resort to standard *prediction error* approaches. However, this kind of map is still quite problematic for the additional design of a reference governor, due to its nonlinear behavior with respect to the decision variable E_k. A first solution to overcome this issue would be to resort to nonlinear optimal control schemes, but this would be computationally expensive as it would require the evaluation of the Jacobian of the involved map at least once for each step k.

Nonetheless, in a large number of applications the expressiveness of nonlinear schemes like the one in (5.7) may as well be not necessary, especially when one aims at obtaining accurate controllers for short-term predictions only.

To exploit this fact, we fit a map in which the predicted control action is nonlinearly dependent *only* on the information vector x_k (that comes from feedback), and possibly on some reference trajectory $\bar{E}_k^T = [\bar{e}_k, \dots, \bar{e}_{k+T}]'$ (which will likewise not be a decision variable), whereas it is *linearly* dependent on the control variables $\Delta E_k^T \triangleq E_k^T - \bar{E}_k^T$. This is equivalent to fit a partial first-order Taylor approximation of the map in (5.7).

Indeed, by assuming H_T to be at least C^1 in the neighborhood of a given trajectory, it holds that

$$\begin{aligned} H_T(x_k, e_k, \ldots, e_{k+T}) \approx H_T(\bar{x}_k, \bar{e}_k, \ldots, \bar{e}_{k+T}) + \\ \frac{\partial H_T}{\partial x_k}(\bar{x}_k, \bar{e}_k, \ldots, \bar{e}_{k+T})\Delta x_k + \\ \sum_{j=0}^{T} \frac{\partial H_T}{\partial e_{k+j}}(\bar{x}_k, \bar{e}_k, \ldots, \bar{e}_{k+T})\Delta e_{k+j}. \end{aligned} \tag{5.8}$$

Since identifying $\frac{\partial H_T}{\partial x_k}\Delta x_k$ is not necessary as x_k is not a decision variable for the governor, our goal becomes to design a controller by fitting a predictor of the form

$$\hat{U}_T = G_T(x_k, \bar{E}_k^T) + F_T(x_k, \bar{E}_k^T)\Delta E_k^T. \tag{5.9}$$

The above architecture requires introducing the quantity $\bar{E}_k$. A possible approach to bypass this requirement is to exploit the fact that holding an explicit relationship to a time-dependent reference trajectory can still be more expressive then necessary in short-term predictions [137]. In this light, one can remove the dependence on $\bar{E}_k$ from (5.9) by assuming $\bar{E}_k = \mathbf{0}$, so as to obtain a controller parameterized as a predictor of the form:

$$\hat{U}_T = G_T(x_k) + F_T(x_k)E_k^T, \tag{5.10}$$

which is effectively equivalent to the one presented in Chapter 4.

While the affine form in (5.10) is quite convenient also from a computational point of view, it severely limits the expressiveness of the predictor as a whole, which is particularly concerning as this architecture cannot be used to represent any nonlinear input characteristic that is dependent on the input itself.

To overcome this issue, we drop the dependency from $\bar{E}_T$ only in G_T, while maintaining it in F_T. This allows us to choose the real input sequence E_k^T as reference trajectory and to set $\bar{E}_k^T = E_k^T$, following the rationale of classical linearization-based control, according to which the most accurate linearization point is the one corresponding to the then-applied input trajectory. Once the resulting controller is deployed, the

quantity $\bar{E}_k$ is instead chosen as the shifted optimal sequence computed at the previous time step. This finally results in trying to learn a controller of the form

$$\hat{U}_T = G_T(x_k) + F_T(x_k, E_k^T)E_k^T \tag{5.11}$$

which is still an approximation of a first-order Taylor expansion of H_T, but it does not require the design of $\bar{E}_k$. At the same time, this structure is sensibly more expressive than the one in (5.10) as it can better represents systems with non-affine input characteristics. We point out that, compared to a standard sensitivity-based approach, evaluating $F_T(x_k, E_k^T)$ comes at no additional cost, making the computational requirements of the resulting controller comparable to a classical Linear Time Varying (LTV) solution.

5.3.1 The design of the reference governor

Under the assumption that a data-driven controller designed to match the reference model $\mathcal{M}$ is feasible for Σ_P, then in closed loop it holds that $y_{k+1}^{\mathcal{M}} \approx y_{k+1}$. This also means that $e_{k+i} \approx e_{k+i}^{\mathcal{M}} = r_{k+i} - y_{k+i}^{\mathcal{M}}$. This approximation can be exploited to design the external reference governor.

Suppose that a fixed term predictor (5.11) has been trained to reproduce u_k and that the reference model $f_{\mathcal{M}}$ can be expanded into the same predictive form considered previously, namely

$$\hat{Y}^{\mathcal{M}} = \begin{bmatrix} y_{k+1}^{\mathcal{M}} \\ \vdots \\ y_{k+T+1}^{\mathcal{M}} \end{bmatrix} = G^{\mathcal{M}}(x_k^{\mathcal{M}}) + F_T^{\mathcal{M}}(x_k^{\mathcal{M}}, \bar{R}_k^T)R_k^T \tag{5.12}$$

with

$$\begin{aligned} R_k^T &= [r_k, \dots, r_{k+T}]', \\ x_k^{\mathcal{M}} &= [r_{k-1}, \dots, r_{k-M+1}, y_k^{\mathcal{M}}, \dots, y_{k-M+1}^{\mathcal{M}}] \end{aligned}$$

for some choice of $\bar{R}_k^T$. Then, the data-driven reference governor can be

parameterized at each control step by solving

$$\begin{aligned}
\min_{R_k^T} \quad & LRG(\hat{Y}^{\mathcal{M}}, U_T, R_k^T, \bar{Y}) \\
\text{subject to} \quad & \hat{Y}^{\mathcal{M}} = G^{\mathcal{M}}(x_k^{\mathcal{M}}) + F_T^{\mathcal{M}}(x_k^{\mathcal{M}}, \bar{R}_k^T) R_k^T \\
& U_T = G_T(x_k) + F_T(x_k, \bar{E}_k^T) E_k^T \\
& E_k^T = [r_k - y_k, r_{k+1} - y_{k+1}^{\mathcal{M}}, \dots, \\
& \qquad r_{k+T} - y_{k+T}^{\mathcal{M}}]' \\
& \bar{E}_k^T = [\bar{r}_k - y_k, \bar{r}_{k+1} - \bar{y}_{k+1}^{\mathcal{M}}, \dots, \\
& \qquad \bar{r}_{k+T} - \bar{y}_{k+T}^{\mathcal{M}}]' \\
& U_T \in \mathcal{X}, \hat{Y}^{\mathcal{M}} \in \mathcal{Y}.
\end{aligned} \tag{5.13}$$

where $\bar{Y}^{\mathcal{M}} = G^{\mathcal{M}}(x_k^{\mathcal{M}}) + F_T^{\mathcal{M}}(x_k^{\mathcal{M}}, \bar{R}_k^T)\bar{R}_k^T$ is the output of the reference model computed for $\bar{R}_k^T$. Note that, compared to the initial design problem in Equation (5.5), we have now separated $\bar{x}_k$ into two distinct quantities, namely the information coming from feedback x_k and $x_k^{\mathcal{M}}$, which depend on the reference model that approximately represents the inner closed-loop. We further highlight that while the fixed horizon predictor for U_T algebraically depends on r_k, the predictions $\hat{Y}^{\mathcal{M}}$ are obtained by using a strictly proper law. Indeed, we cannot expect the closed-loop system to be "delay-free" and, at the same time, we want to exploit the most recent feedback information to compute the current control action. This also ensures that the controller is not subject to mismatches, at least on the first step of the prediction horizon.

Remark 2 *If $\mathcal{X}, \mathcal{Y}$ are polytopes and LRG is either linear or quadratic, Problem* (5.13) *can be then reliably solved within real-time constraints [158], [159].*

Offset rejection

The presented architecture does not include an integral action and it is naturally prone to experience offset when tracking constant references. From a practical point of view, this offset can be seen as a constant disturbance $q \in \mathbb{R}^{N_y}$ that affects the output of the system. A way to overcome this issue would then be to recursively estimate the magnitude of such a disturbance and compensate the reference model accordingly. Indeed, for a classical single-output case, a recursive estimator for q can be

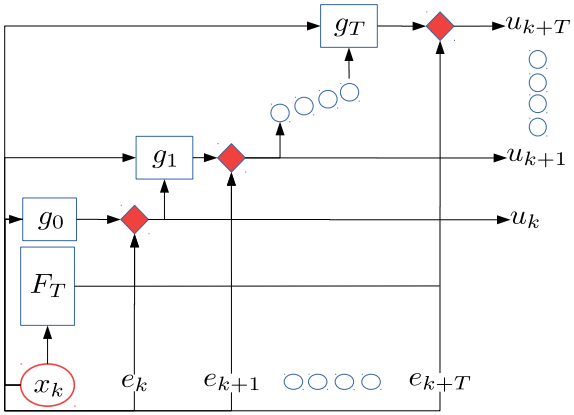


Figure 14: Illustrative scheme of the employed ANN structure, adapted from [19]. © 2020 IEEE

designed as

$$\hat{q}_k = \hat{q}_{k-1} - \alpha(\hat{y}_k^{\mathcal{M}} - y_k), \tag{5.14}$$

where $\hat{y}_k^{\mathcal{M}}$ is the predicted evolution of the closed-loop system at time $k-1$, y_k is the feedback information at time k and $\alpha \in \mathbb{R}$. The estimate $\hat{q}_k$ can then be integrated in a reference governor scheme, which will now aim to solve

$$\begin{aligned} \min_{R_k^T} \quad & LRG(\hat{Y}^{\mathcal{M}}, U_T, R_k^T, \bar{Y}) \\ \text{subject to} \quad & \hat{Y}^{\mathcal{M}} = G^{\mathcal{M}}(x_k^{\mathcal{M}}) + F_T^{\mathcal{M}}(x_k^{\mathcal{M}}, \bar{R}_k^T) R_k^T + \hat{q}_k \mathbf{1} \\ & \text{as in Equation (5.13)} \end{aligned} \tag{5.15}$$

where $\mathbf{1} \in \mathbb{R}^T = [1, \ldots, 1]'$. Similar correction will likewise be applied to $\bar{Y}^{\mathcal{M}}$. ■

5.3.2 ANNs for controller parameterization

As already noted in Chapter 4, directly fitting a set of discrete maps like the ones in Equations (5.6) can be quite an inefficient process, since no information from the predictions of the earlier terms along the horizon

is necessarily re-used to predict the later ones. At best, this would make the training procedure more difficult to carry out. More likely, it will require the use of more powerful regression techniques to achieve the desired fitting performance, especially if we use approximators like Artificial Neural Networks (ANNs), that naturally benefit from the stacking of nonlinear layers/components.

A possible way to leverage the knowledge obtained from the approximators employed in shorter-term predictions is to use their outputs $\hat{u}_k$ as additional input for the regressor involved in the prediction of u_{k+n}, $n = 1, \ldots, T$. In particular, when training the controller within our structure, we obtain

$$\begin{aligned} \hat{u}_{k+n} &= f_n(x_k) + g_n(x_k, \bar{E}_k^n) \left[\frac{E_k^n}{\hat{u}_{k+n-1}} \right] \\ &= \tilde{f}_n(x_k) + \tilde{g}_n(x_k, \bar{E}_k^n) E_k^n \end{aligned} \tag{5.16}$$

for some choice of $\bar{E}_k^n$. This process can be repeated all over the horizon and might also include, when applicable, additional terms other than the one immediately preceding the n-th term. We stress that the number of past linear terms is a *hyper-parameter* of the approach, that can be tuned to trade-off between the accuracy of the predictor and the sparsity pattern of the resulting constraint structure.

The problem of learning *in parallel* a set of maps like (5.11) can be then recast into solving the following optimization problem:

$$\begin{aligned} \min_{\substack{f_0, \ldots, f_T \\ g_0, \ldots, g_T}} \quad & \sum_{k=M+1}^{\mathcal{N}-T-1} \mathcal{L}_{\text{train}}(\hat{O}_k, O_k) \\ \text{s.t.} \quad & \end{aligned} \tag{5.17}$$

$$
\begin{aligned}
&\hat{O}_k = \begin{bmatrix} \hat{u}_k \\ \vdots \\ \hat{u}_{k+T} \end{bmatrix} O_k = \begin{bmatrix} u_k \\ \vdots \\ u_{k+T} \end{bmatrix} \\
&\hat{u}_k = f_0(x_k) + g_0(x_k, \bar{e}_k) e_k^\star \\
&\hat{u}_{k+i} = f_i(x_k) + \\
&\qquad g_i \left(x_k, \begin{bmatrix} \bar{e}_k \\ \vdots \\ \bar{e}_{k+i} \\ \hat{u}_{k+i-1} \end{bmatrix} \right) \begin{bmatrix} e_k^\star \\ \vdots \\ e_{k+i}^\star \\ \hat{u}_{k+i-1} \end{bmatrix} \\
&\text{for } i = 1, \ldots, T
\end{aligned}
$$

where $\mathcal{L}_{\text{train}} : \mathbb{R}^{((T+1)\times N_U)} \times \mathbb{R}^{((T+1)\times N_U)} \to \mathbb{R}$ is a suitable loss function. Without loss of generality, we select $\mathcal{L}_{\text{train}}$ as the Mean Absolute Error (MAE) [60]. Speaking about the loss figure $\mathcal{L}_{\text{train}}$, in principle, there are no limitations on its nature. While in practice a user might want to restrict its choices to figures that are compatible with classical derivative-based optimization frameworks, this allows fine-tuning the learning process to the physical meaning of the analyzed quantities, i.e.: using a cosine distance for an angular quantity.

The peculiar structure of the considered training problem restricts the pool of the functional approximators that can be used to represent the maps f_i and g_i. Instead of developing application-specific solutions, here we exploit the nature of ANNs as directed acyclic computational graphs to build the structure of the constraints in (5.17) into the topology of the network itself. Choosing neural networks is also convenient due their flexibility to be trained with any differentiable loss function, their theoretical universal approximation power [58], and the availability of well maintained frameworks [160] that can aid the design. This approach also allows us to reduce the whole learning procedure to a standard regression problem, while retaining the capability to easily access the outputs of f_i, g_i as intermediate results of the single sub-components of the network, whose overall scheme is reported in Figure 14.

We stress that the topology of the involved ANNs closely follows the structure highlighted in (5.16). In particular, all the components f_i are grouped in a single discrete sub-network, while each g_i is a stan-

dalone object. For simplicity, we restrict our analysis to a densely connected feed-forward topology for each subnetwork, meaning that each sub-ANN is comprised by a stack of nonlinear hidden layers connected to a final linear output layer with appropriate output shape.

Remark 3 *Although one might use the proposed fixed horizon predictor as a stand-alone controller by applying at each step the first component of $\hat{U}_T$, we remark that no precautions are taken to make the resulting controller reliable for such a use case.*

5.4 Simulation case studies

In this section, we present the preliminary results obtained by employing the proposed technique on a variation of two nonlinear benchmarks already used in the previous Chapters, *i.e.*, the discrete-time approximation of the well-known nonlinear two-tanks system

$$\Sigma_{\text{tank}} = \begin{cases} x_{1,k+1} = x_{1,k} - k_1\sqrt{x_{1,k}} + k_2 v_k \\ x_{2,k+1} = x_{2,k} + k_3\sqrt{x_{1,k}} - k_4\sqrt{x_{2,k}} \\ v_k = \begin{cases} 1.5 & \text{if } u_k \geq 1.0 \\ -0.5 & \text{if } u_k \leq -1.0 \\ u_k + 0.5 & \text{otherwise} \end{cases} \\ y_k = x_{2,k} \end{cases} \tag{5.18}$$

where $x_{i,t}$ denotes the i-th component of $x_t \in \mathbb{R}^2$ and $k_1 = 0.5$, $k_2 = 0.4$, $k_3 = 0.2$, $k_4 = 0.3$, and the following Hammerstein-Wiener model

$$\Sigma_{\text{HW}} = \begin{cases} x_{k+1} = \begin{bmatrix} 0.7555 & 0.25 \\ -0.1991 & 0 \end{bmatrix} x_k + \\ \qquad \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} v_k \\ v_k = \begin{cases} 1 & \text{if } u_k \geq 1.0 \\ -1 & \text{if } u_k \leq -1.0 \\ \text{sign}(u_k)\sqrt{|u_k|} & \text{otherwise} \end{cases} \\ w_k = [0.6993 \ \ -0.4427] x_k \\ y_k = w_k + 5\sin(w_k) \end{cases} \tag{5.19}$$

Note that both models exhibit a saturation-like nonlinearity on the input.

As reference model $\mathcal{M}$ for Σ_{tank} we considered the following Linear Time-Invariant (LTI) model

$$\mathcal{M}_{\text{tank}}(z) = \frac{0.4z^{-2}}{1 - 0.6z^{-1}}$$

where z^{-1} is the delay operator. For Σ_{HW} we instead use the reference model

$$\mathcal{M}_{\text{HW}}(z) = \frac{0.65z^{-1}}{1 - 0.35z^{-1}}.$$

We stress that both the reference models are linear so that in closing the loop we aim at compensating the plant nonlinearity, while fully exploiting its operating regime. We remark that the reference model is a hyperparameter of the VRFT framework and its optimal choice is linked to the nature of the target process. We refer the interested reader to [147], [161] for further reading on the topic.

For both these benchmark systems, we collected a training dataset $Z_{\mathcal{N}}$ of 20,000 samples, generated by exciting the system with a sequence of variable-amplitude step signals of period 7 steps, with amplitudes drawn from a Gaussian distribution with zero mean and standard deviation $\sigma = 1$ and then clipped to be within the interval $[-1, 1]$. After the experiment, the output signal was empirically normalized using the empirical mean and standard deviation computed from the dataset, before being used for training. After the normalization, all the signals were superimposed with a Gaussian white noise drawn from a distribution with zero mean and standard deviation $\sigma_w = 0.02$.

With these data, we trained a fixed-horizon predictor with $T = 6$, in which every sub-network is composed of 4 Rectified Linear Units (ReLU) [79] hidden layers with 20 neurons followed by a final linear output layer. The state size was set to $M = 7$. The implementation was carried out using Keras [80]. A Lasso [67] penalization was also added to all neurons to regularize the resulting models.

The reference governor was tuned by setting

$$LRG = \|\hat{Y}^{\mathcal{M}} - \bar{Y}\|_2^2$$

and by imposing a constraint on the input, namely that $|u_k| \leq 1, \ \forall k$.

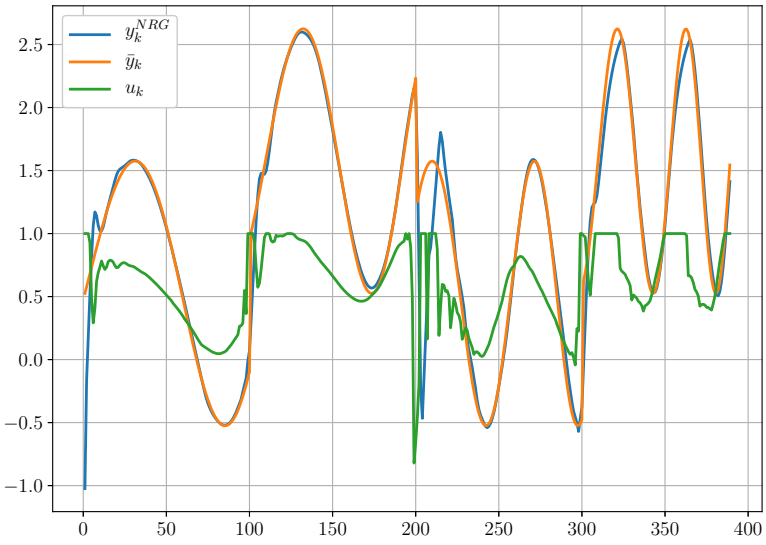


Figure 15: Closed-loop performance (Σ_{tank}). © 2020 IEEE

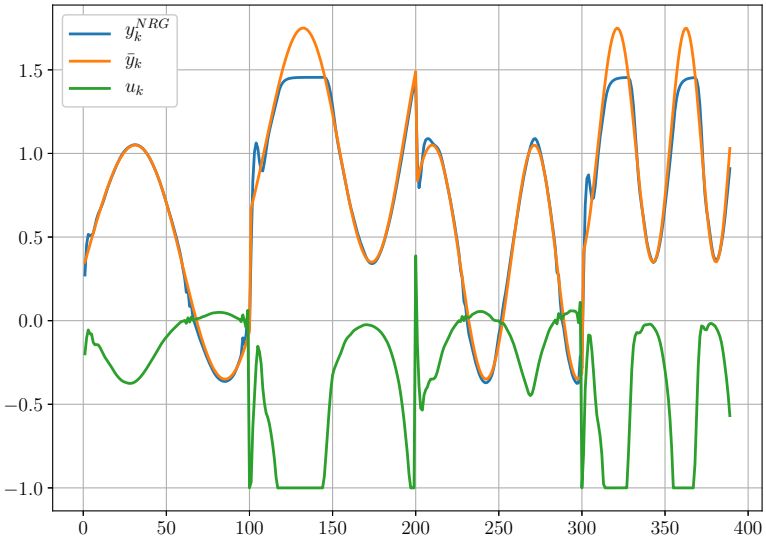


Figure 16: Closed-loop performance (Σ_{HW}). © 2020 IEEE

The reference trajectory at time k is set by shifting the optimal sequence computed at the previous step $k-1$. We also assumed that a preview of the future references was available and further embedded in our structure, together with the recursive disturbance compensator in (5.14), with gain α set to 0.1.

In Figure 15 we report the closed-loop performance obtained on Σ_{tank} by the proposed approach when tracking a square wave superimposed with a sine-sweep reference signal. The proposed approach, while not being that effective during the initial steps of the transient, is able to properly track also high-frequency reference signals.

Similar considerations can be drawn for the closed-loop performance associated to the system Σ_{HW} shown in Figure 16. In there the proposed approach achieves satisfactory performance, showing good capabilities to deal with the strong nonlinearity of the input characteristic of the system when u_k crosses 0. In both Figures 15 and 16 we also appreciate how the disturbance observer does not cause any windup issue, that would severely affect closed-loop performance.

Remark 4 (Computational insights) *As a whole, the employed predictor has $\approx 10^4$ parameters. The training procedure was carried out in a few minutes using an Intel Core i5-6200U CPU laptop with 16GB of RAM. The controller, which relied on a general-purpose solver [84], required around 10^{-2} seconds to compute the control action at each time step. Considering the general-purpose nature of the employed libraries, computational requirements can probably be significantly reduced by using more specialized solvers, such as the one in [162], and more performance-oriented deep learning frameworks.* ■

5.5 Conclusions

We presented a variation of the approach presented in [130] to directly synthesize nonlinear constrained controllers within the model-reference framework. In designing the proposed neural reference governor, we exploit results from control-oriented system identification already explored in Chapter 4 to directly learn short-term fixed horizon ANN-based predictors of the virtual internal control law from data. The resulting scheme

has shown good performance on two nonlinear benchmarks, ensuring good closed-loop reference tracking performance, and suggesting that it can be competitive with more traditional control design scheme.

These results, albeit preliminary, jointly with the modest computational resources required by the approach, suggest its possible use in embedded fast-sampling applications, which will be the focus of future works together with an in detail comparison with control-aware system-identification and constrained data-driven control schemes.

Chapter 6

NAW-NET: neural anti-windup control for saturated nonlinear systems

6.1 Introduction

Due to the increasing complexity of control systems and thanks to the ever-growing availability of large sets of data, control strategies are more and more commonly designed with the aid of data-driven approaches, which provide invaluable help to either compute/update a model of the plant to be controlled or to directly tune the parameters of the controller. Indeed, this is not a novel trend: several *System Identification* (SI) techniques have been proposed over the years to learn approximate laws that describe the behavior of otherwise unknown processes [1]. However, as we have discussed in the previous Chapters, these classical approaches are agnostic regarding the intended applications of the learnt models and thus usually aim only to obtain the best open-loop simulation accuracy. This is, however, often an overreach as many control techniques can attain satisfactory performance even with a crude approximation of the real plant dynamics. Moreover, the quest for models minimizing the simulation error usually results in large-dimensional models,

which in turn may require the use of model-reduction techniques to be made compatible with control synthesis procedures and with resource-constrained hardware platforms. Last but not least, as we have seen in Chapter 2, identification techniques can require quite some expertise to be used, thus making them out of reach for many control practitioners.

As an alternative to such a classical two-stage design paradigm, where a model of the plant is first identified, and then a model-based controller is designed, *direct data-driven control* strategies have been proposed. In this case, data collected from the plant are directly used to synthesize a controller without the involvement of an explicit model of the system. These methods range from data-driven optimal control approaches such as *Reinforcement Learning* (RL) [10], [11], [163], [164] to schemes exploiting the *model reference* paradigm, such as the *Virtual Reference Feedback Tuning* (VRFT) approach [151], [165]. As we have already shown in Chapter 5, the latter approaches have the advantage of being one-shot techniques, in that they do not require additional data to refine/tune the control law. However, they also provide little to no flexibility to ease the integration of constraint handling methods. Since, in practice, the authority of actuators is always limited, the use of these techniques in industrial applications has been thus restricted to those limited cases in which such an issue is not problematic.

Traditionally, input saturations are managed either by using predictive control strategies [69] or by pairing a controller with an *anti-windup compensator*, which aims at preserving performance when the system operates within actuator bounds and trying to guarantee the asymptotic recovery of unconstrained behavior after saturation occurs [166]–[168]. These approaches not only require the knowledge of the saturation limits, but they generally rely on additional insights on the plant. However, the level of accuracy needed to attain satisfactory performance might not be attainable when the model for the plant is retrieved from data. As an alternative to classical model-based strategies, some attempts have been recently made to incorporate constraint-handling methods within the data-driven control framework. For example, in [130] a reference governor is used on top of a direct data-driven controller for *Linear Parameter*

Varying (LPV) systems [152] to impose constraint satisfaction. Another comparable procedure, which also suffers from the need for two successive design phases, is the one reported in Chapter 5. Nevertheless, both approaches still require the solution of an optimization problem in real-time for each control step.

Inspired by *anti-windup* architectures, we propose a one-shot learning scheme to design a *Neural NETwork for Anti-Windup control* (NAW-NET) directly from data. The proposed approach seeks to allow one to retrieve both a controller and an anti-windup compensator for an *unknown* plant from a set of input/output data that can handle actuator constraints effectively. Inspired by the principle behind the VRFT approach, the synthesis procedure is fully carried out off-line.

In order to account for the innate nonlinear nature of constrained control strategy and, possibly, of the plant to be controlled, we parameterize the controller and the saturation compensator through two different *Artificial Neural Networks* (ANNs), due to their flexibility and excellent function approximation properties [169]. The modular nature of the chosen controller parameterization further allows us to easily convert the proposed 2 *Degree-Of-Freedom* (DOF) control architecture into a simpler 1 DOF structure, where the anti-windup action is *embedded* within the control input.

Remark 5 *For simplicity, in this Chapter, we focus only on the more traditionally structured 2-DOF architecture, but please note that there are no a-priori (dis)advantages of using this structure rather than a 1-DOF one.* ■

The rest of the Chapter is organized as follows. Section 6.2 formally states the control problem of interest. Section 6.3 is devoted to the formulation of the NAW-NET learning task. The proposed structure and the proposed anti-windup compensator are described in Section 6.3.2. A strategy to handle non-informative data is proposed in Section 6.3.4. Simulation results are shown in Section 6.4, while conclusions and possible directions for future work are discussed in Section 6.5.

6.2 Setting and Goal

Let Σ be a *discrete-time* nonlinear *Single-Input Single-Output* (SISO) plant, whose dynamics is

$$\Sigma : \begin{cases} x_{k+1} = f(x_k, u_k), \\ y_k^{\circ} = h(x_k), \end{cases} \tag{6.1}$$

where $u_k \in \mathbb{R}$, $x_k \in \mathbb{R}^{n_x}$ and $y_k^{\circ} \in \mathbb{R}$ are the commanded input, the state and the noiseless output of the plant at the instant $k \in \mathbb{N}$, respectively, and $f : \mathbb{R}^{n_x} \times \mathbb{R} \to \mathbb{R}^{n_x}$, $h : \mathbb{R}^{n_x} \to \mathbb{R}$ are *unknown* maps in the input and the state of Σ. Suppose that the system is *Bounded-Input Bounded-Output* (BIBO) stable so that an open-loop experiment on a the system can be carried out without the need to resort on a baseline controller. Suppose also that the actuator has limited range, i.e., that the actual input received by the plant is $\tilde{u}_k \triangleq g_{\text{sat}}(u_k)$, where

$$g_{\text{sat}}(u_k) \triangleq \begin{cases} \underline{u} & \text{if } u_k < \underline{u}, \\ v_k & \text{if } u_k \in [\underline{u},\ \bar{u}], \\ \bar{u} & \text{if } u_k > \bar{u}, \end{cases} \tag{6.2}$$

with $\underline{u},\ \bar{u} \in \mathbb{R}$ being the *known* limits of the actuator. Accordingly, the dynamics (6.1) can be recast as

$$\Sigma : \begin{cases} x_{k+1} = \tilde{f}(x_k, u_k), \\ y_k^{\circ} = h(x_k). \end{cases} \tag{6.3}$$

where $\tilde{f}(x, u) = f(x, \tilde{u})$. Since no prior knowledge is available on Σ, other than the actuator bounds, and output sensors are noisy, we assume that open-loop experiments are carried out by providing a proper sequence of inputs u_k to the plant and record

$$y_k = y_k^{\circ} + v_k$$

for $k = 1, \dots, N$, where $v_k \in \mathbb{R}$ is measurement noise.

6.2.1 Direct data-driven control design

Given the *open-loop* data collection $Z_N = \{u_k, y_k\}_{k=1}^{N}$, our aim is to design a *nonlinear* feedback controller C_θ belonging to a pre-defined class

$\mathcal{C}_\theta$ so that: (i) some desired reference-to-output tracking performance is attained, if allowed by the control authority limitations; (ii) a safe behavior of the controlled system is possibly promoted even if the actuator saturates.

Throughout the rest of the Chapter, we focus on the following input-output parametrization $\mathcal{C}_\theta$ of the controller, described as

$$\mathcal{C}_\theta:\ \nu_{k+1} = C_\theta(\nu_k, \ldots, \nu_{k-Q}, e_{k+1}, \ldots, e_{k-Q}), \tag{6.4}$$

where $e_k = r_k - y_k$ is the tracking error attained at time k and $Q > 0$ is the order of the controller, which is fixed a priori by the designer.

In this chapter we focus on controller able to deal with saturation like the one in Equation 6.2. To do so, we enchant the control architecture with an *anti-windup* block [166] $H_\varphi \in \mathcal{H}_\varphi$ that embeds the known saturation, defined as

$$\mathcal{H}_\varphi:\ a_{k+1} = H_\varphi(\nu_{k+1}, \nu_k, \ldots, \nu_{k-H}, e_{k+1}, \ldots, e_{k-H}), \tag{6.5}$$

whose output is subtracted to the one generated by the controller C_θ, so that the overall control architecture can be represented as in Figure 17.

Remark 6 *To guarantee a clear distinction between C_θ and H_φ, one can slightly modify the structure of the compensator as follows:*

$$\tilde{\mathcal{H}}_\varphi:\ a_{k+1} = H_\varphi(\nu_{k+1}, \nu_k, \ldots, \nu_{k-H}),$$

so that the anti-windup action depends only on the collection of corrected and past outputs of C_θ. This leads to a structure more similar to the one adopted in standard anti-windup schemes [166]. On the other hand, it reduces the generality in the structure of the compensator. ■

Here we explore various approaches to parameterize C_θ and H_φ using *Artificial Neural Networks* (ANNs). Without loss of generality we further assume that they share the same order $H = Q$. We stress that the knowledge of the saturation is embedded within the chosen parametrization of C_θ and H_φ, as explained in Section 6.3.2.

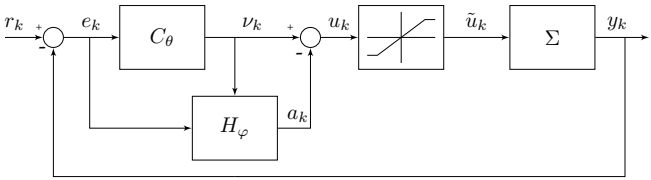


Figure 17: The proposed direct data-driven anti-windup control scheme. © 2020 IEEE

6.3 NAW-NET: training

To avoid modeling the plant Σ, we rely only on the open-loop dataset Z_N and exploit the rationale of the *Virtual Reference* approach, originally described in [151]. Therefore, we construct a *virtual* closed-loop using the open-loop data collection, such that the reference-to-output relationship is exactly $\mathcal{M}$, as shown in Figure 18.

Similarly to the approach presented in Chapter 5, we start from the available input and output measurements, and we compute what would have been the *virtual* reference $\tilde{r}_k$ and the *virtual* error signal $\tilde{e}_k = \tilde{r}_k - y_k$ of such a fictitious closed-loop, by noticing that the former corresponds to the signal that would produce the measured output y_k when feeding the reference model $\mathcal{M}$, *i.e.*,

$$y_{k+1} = f_{\mathcal{M}}(y_k, \dots, y_{k-M}, \tilde{r}_k, \dots, \tilde{r}_{k-M}).$$

The computation of $\tilde{r}_k$ requires the inversion of $\mathcal{M}$, which can be explicitly obtained for relatively simple classes of reference models, such as linear transfer functions with asymptotically stable zeros [130]. In case of more complex reference models $\mathcal{M}$, more generally the fictitious reference can be obtained by solving the fitting problem

$$\begin{aligned} \min_{\{\tilde{r}_k\}} \ & \frac{1}{N-M} \sum_{k=M+2}^{N} \ell_M(\hat{y}_k^M, y_k) \\ \text{s.t. } & \hat{y}_{k+1}^{\mathcal{M}} = f_{\mathcal{M}}(y_k, \dots, y_{k-M}, \tilde{r}_k, \dots, \tilde{r}_{k-M}), \end{aligned} \tag{6.6}$$

where $\ell_M : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a properly defined loss function.

Once the fictitious reference sequence $\{\tilde{r}_k\}$ is computed, the latter can be used to design the desired blocks (see equations (6.4) and (6.5), respectively), by trying to match the measured input sequence $\{u_k\}_{k=1}^N$ with the predicted input from the controller C_θ.

6.3.1 Training the anti-windup block

While the controller C_θ is asked to track as closely the recorded *command signal*, in our scheme the burden of ensuring that the actual *control signal* will not cause issues falls on H_φ. As these two components are intimately linked, a practical option is to train both of them at the same time. Formally, this means solving the following learning problem

$$\begin{aligned} &\min_{\substack{C_\theta \in \mathcal{C}_\theta \\ H_\varphi \in \mathcal{H}_\varphi}} \mathcal{J}(\hat{\nu}_k(\theta), u_k, \hat{a}_k(\varphi)), \\ &\text{s.t. } \hat{\nu}_{k+1}(\theta) = C_\theta(u_k, \dots, u_{k-Q}, \tilde{e}_{k+1}, \dots, \tilde{e}_{k-Q}), \\ &\quad \hat{a}_{k+1}(\varphi) = H_\varphi(\hat{\nu}_{k+1}, u_k, \dots, u_{k-Q}, \tilde{e}_{k+1}, \dots, \tilde{e}_{k-Q}), \end{aligned} \tag{6.7}$$

where

$$\begin{aligned} &\mathcal{J}(\hat{\nu}(\theta), u, \hat{a}(\varphi)) = \\ &\qquad \frac{1}{N-Q} \sum_{k=Q+2}^{N-1} \ell_C(\hat{\nu}_k(\theta), u_k) + \gamma \ell_S(\hat{\nu}_k(\theta) - \hat{a}_k(\varphi)), \end{aligned} \tag{6.8}$$

and $\gamma > 0$ is a hyper-parameter to be tuned that trades off between fitting the input samples and violate saturation limits. Note that the order of the controller Q dictates the number of samples that are discarded in the learning phase.

We adopt the *Mean Absolute Error* (MAE) loss figure [60] to weight the fitting error on the input (other losses could be used too), while the error due to unfeasible control actions is accounted for by defining ℓ_S as follows:

$$\ell_S(\hat{u}_k(\theta, \varphi)) = \|\hat{u}_k(\theta, \varphi) - g_{\text{sat}}(\hat{u}_k(\theta, \varphi))\|_1,$$

with $\hat{u}_k(\theta, \varphi) = \hat{\nu}_k(\theta) - \hat{a}_k(\varphi)$.

Remark 7 *As mentioned in the introduction, The learning problem in* (6.7)*, along with the proposed anti-windup control scheme, can be readily adapted*

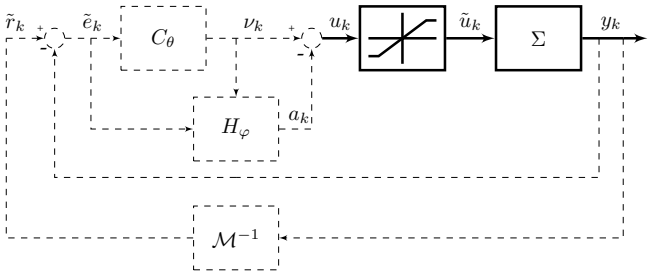


Figure 18: The *Virtual Reference* rationale of [151] used within our setting. The thick line denotes the real plant on which the experiment has been performed, while the dashed lines illustrate the "virtual" remainder of the loop. © 2020 IEEE

to a 1-DOF architecture, by setting $\hat{a}_k = 0$, $\forall k$, *so that the learning problem becomes*

$$\min_{C_\theta \in \mathcal{C}_\theta} \frac{1}{N-Q} \sum_{k=Q+2}^{N-1} \ell_C(\hat{\nu}_k(\theta), u_k) + \gamma \ell_S(\hat{\nu}_k(\theta))$$
$$s.t.\ \ \hat{\nu}_{k+1}(\theta) = C_\theta(u_k, \dots, u_{k-Q}, \tilde{e}_{k+1}, \dots, \tilde{e}_{k-Q}).$$

Although not explicitly exploiting an anti-windup block, this optimization problem still allows one to train a controller that accounts for saturation limits, thanks to the structure of C_θ *(see Section 6.3.2).*

Remark 8 *Since the solution of problem* (6.7) *allows us to concurrently train the controller and the anti-windup block, a possible strategy for the selection of* γ *is described next. By solving problem* (6.7) *multiple times iteratively, one can initially choose* γ *low to promote better fitting performance. The value of* γ *can then be gradually increased so to enforce satisfaction of the actuator bounds.*

Remark 9 *Alternatively to the cost in* (6.8)*, one can also minimize the following objective:*

$$\mathcal{J}(\hat{u}(\theta, \varphi), \tilde{u}) = \frac{1}{N-Q} \sum_{k=Q+2}^{N-1} \ell_C(\hat{u}_k(\theta, \varphi), \tilde{u}_k) + \gamma \ell_S(\hat{u}_k(\theta, \varphi)),$$

with $\hat{u}_k(\theta, \varphi) = \hat{\nu}_k(\theta) - \hat{a}_k(\varphi)$*. In this case,* C_θ *and* H_φ *are designed so that the input to* Σ *corresponds to the saturated control actions* $\{\tilde{u}_k\}_{k=1}^N$*. We stress that* ℓ_S *has still to be included in the cost if one wants the compensator to act only when the control action* ν_k *exceeds the saturation bounds, as in standard anti-windup schemes.*

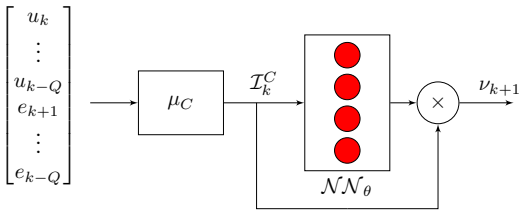


Figure 19: The selected controller structure. © 2020 IEEE

6.3.2 NAW-NET parameterization

It is clear that the chosen parameterizations for the maps $C_\theta \in \mathcal{C}_\theta$ and $H_\varphi \in \mathcal{H}_\varphi$ are crucial to achieve good control performance, namely to attain the desired closed-loop behavior $\mathcal{M}$ while capitalizing on the whole operating range of the actuators. Among possible alternatives for both the controller and the anti-windup block parameterizations, we use *Artificial Neural Networks* (ANNs) [66] for their well-known effectiveness as maps approximators and the availability of well-maintained tools for easily training them.

Specifically, the controller is parameterized as the input/output law

$$\nu_{k+1} = \mathcal{NN}_\theta(\mathcal{I}_k^C)'\mathcal{I}_k^C, \tag{6.9}$$

where $\mathcal{I}_k^C \in \mathbb{R}^{n_\mathcal{I}}$ is the feature vector fed to the parametric part of the controller, namely the ANN $\mathcal{NN}_\theta : \mathbb{R}^{N_\mathcal{I}} \to \mathbb{R}$, whose internal structure is schematically represented in Figure 19. The feature vector $\mathcal{I}_k^C$ is extracted via a map $\mu_C : \mathbb{R}^{2Q+1} \to \mathbb{R}^{n_\mathcal{I}}$ on the measured inputs and tracking errors, namely

$$\mathcal{I}_k^C = \mu_C(u_k, \dots, u_{k-Q}, \tilde{e}_{k+1}, \dots, \tilde{e}_{k-Q}), \tag{6.10}$$

through which we embed in our controller any available prior knowledge on Σ. Since in our setting the only prior information on the plant resides in the known saturation function g_{sat} in (6.2), we select μ_C as

$$\begin{aligned} &\mu_C(u_k, \dots, u_{k-Q}, \tilde{e}_{k+1}, \dots, \tilde{e}_{k-Q}) = \\ &\quad = \left[u_k, \cdots, u_{k-Q}, g_{\text{sat}}(u_k), \cdots, g_{\text{sat}}(u_{k-Q}), e_{k+1}, \cdots, e_{k-Q}\right], \end{aligned} \tag{6.11}$$

so to also include the projections $g_{\text{sat}}(u_{k-j})$ on the admissible input range of past control samples u_{k-j} as additional regressive terms, $j = 0, \dots, Q$.

We remark that, in principle, μ_C can be learned from data along with $\mathcal{NN}_\theta$. However, this may require a much larger amount of training data and would result in a more complex architecture, eventually making the approach too demanding for simple embedded applications. From a practical stand point, C_θ is implemented through a feed-forward neural network fed by $\mathcal{I}_k^C$. Nonetheless, the network is used to predict a set of intermediate coefficients that produce the control action once they are multiplied by $\mathcal{I}_k^C$.

The chosen structure of C_θ allows us to trade-off between the expressiveness of the parameterization and the ease of inspection of the learned controller. Moreover, if a *linear time invariant* (LTI) parameterization is sufficient, it is fairly simple to downgrade the neural controller to an LTI one, by exploiting parameter shrinkage strategies on the non-bias weights of the network in the training phase [67]. Indeed, if after training the non-bias weights all result zero, this implies that the network itself can be replaced by its (constant) output. In such a case, the control law as whole falls back into a more well understood "LTI controller + anti-windup" architecture, which may be helpful to pursue further analysis on the obtained closed-loop.

Differently from the controller, for the anti-windup block we resort to a standard input-output feed-forward structure, with the ANN parameterizing H_φ directly mapping its input into the corrective action a_k. This choice is due to the fact that the anti-windup block is nonlinear, thus making more complex architectures like the one chosen for C_θ quite useful, at the price of a more complex optimization problem to be solved during the training phase. For the compensator to share similar characteristics to standard anti-windup blocks, we do still exploit a preliminary feature extraction map $\mu_H = \mu_C$.

6.3.3 Improving NAW-NET performance via Truncated Back Propagation Through Time

Despite its appealing simple structure, a design problem based on one-step-ahead predictions like the one defined in (6.7) might result in a

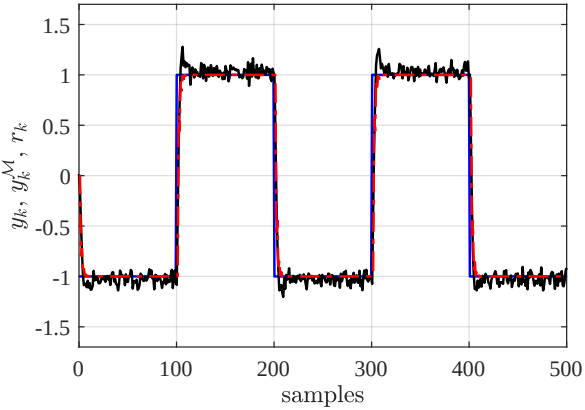


Figure 20: Reference (blue) vs desired (dashed red) and attained (black) closed-loop response. Desired and achieved closed-loop response when tracking the set point in (6.15). The reference signal and the desired output are almost always overlapped. © 2020 IEEE

data-driven control scheme that, albeit good over short horizons, may not be suited to be repeatedly iterated over time. A possible approach to overcome this limitation is to exploit *Back Propagation Through Time* in the learning scheme, which has already been used for direct control applications in [61], [169]. This approach allows one to evaluate the controller+anti-windup block *simulation* accuracy, by concatenating predictions at two successive sampling steps. However, it concurrently involves the optimization of loss functions that are computationally expensive to evaluate and hard to minimize [63].

To avoid over-complicating the learning scheme, while retaining some of the desirable features of a BPTT architecture, we resort like in Chapter 2 on the *truncated* BPTT approach [64].Accordingly, predictions are propagated for a limited number of steps $F \ll N$, with F being a strictly positive additional tuning parameter of the approach. Let then

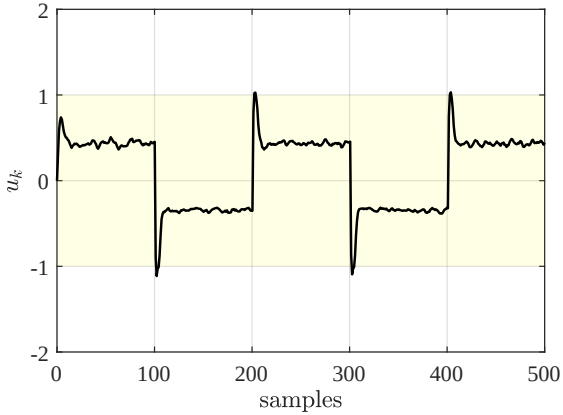


Figure 21: Control input (black) and linear operating region of the actuator (yellow area). © 2020 IEEE

v_{k+h} be defined as

$$v_{k+h} = \begin{cases} u_{k+h} \text{ if } h \leq 0, \\ \hat{\nu}_{k+h} \text{ otherwise,} \end{cases} \tag{6.12a}$$

and

$$\tilde{\mathcal{J}}(\hat{\nu}(\theta), u, \hat{a}(\varphi)) = \frac{1}{N-Q} \sum_{k=Q}^{N-F-1} \sum_{j=1}^{F} \ell_C(\hat{\nu}_{k+j}(\theta), u_{k+j}) + \gamma \ell_S(\hat{\nu}_{k+j}(\theta) - \hat{a}_{k+j}(\varphi)). \tag{6.12b}$$

The design problem in (6.7) is thus recast as:

$$\begin{aligned} &\min_{\substack{C_\theta \in \mathcal{C}_\theta \\ H_\varphi \in \mathcal{H}_\varphi}} \tilde{\mathcal{J}}(\hat{\nu}(\theta), u, \hat{a}(\varphi)) \\ &\text{s.t. } \hat{\nu}_{k+j}(\theta) = C_\theta(v_{k+j}, \ldots, v_{k-Q+j}, \tilde{e}_{k+j}, \ldots, \tilde{e}_{k-Q+j}), \\ &\quad \hat{a}_{k+j}(\varphi) = H_\varphi(\hat{\nu}_{k+j}, v_{k+j-1}, \ldots, v_{k-Q+j}, \\ &\qquad\qquad \tilde{e}_{k+j}, \ldots, \tilde{e}_{k-Q+j}), \\ &\qquad\qquad j = 1, \ldots, F. \end{aligned} \tag{6.12c}$$

It is clear that the quality of the control scheme obtained via the solution of the optimization problem in (6.12c) heavily depends on the choice of

the length of the prediction horizon F. We stress that F should be chosen so as to compromise of the available computing capabilities, the simulation accuracy of the learned controller, and the ability of the optimization solver to reach a good quality solution of (6.12).

6.3.4 Data augmentation

Envisioning a training scheme for the antiwindup H_φ block is a trickier process. The first difficulty arises from the experiment design phase. Indeed, in most common situations, we cannot expect that the dataset will encompass many situations in which the actuators saturate as this would cause safety/reliability concerns for both the actuator and the system itself, meaning that one cannot arbitrarily select the inputs that excite the plant.

While this condition might be good for a prospective identification of Σ in (6.3) since it would reduce the overall non-linearity between input and output data, having a training set that only consists of input/output sequences that already belong to the feasible region, there is little hope that the constraint enforcement module H_φ obtained by solving (6.12c) will have satisfactory performances. However, If one can assume that during the experiment the actuators were used (at least) close to their limit, an effective and rather general strategy to handle this problem is to augment the original dataset with *fictitious samples* that would violate the saturation bounds. The generation of these data is quite a delicate task, which could also deteriorate the quality of the original dataset with samples that the actual plant could never generate. To bypass these limitations, we rely on the following heuristic by accounting for the fact that the saturation bounds act on the input only: we augment the learning procedure, by superimposing a suitably defined white noise sequence on the inputs v_k of the two involved blocks when evaluating the antiwindup action. In such a case, a possible choice for such a disturbance is an additive noise with Gaussian distribution, so that the each input v_k of the involved ANNs is substituted by

$$\rho(v_k) = v_k + w, \quad w \sim \mathcal{N}(0, \sigma^2), \tag{6.13}$$

Table 16: Performance indexes obtained with different controllers.© 2020 IEEE

	$\text{RMSE}_{\mathcal{M}}$	$\mathcal{OB}_{\%}$
Unbounded actuator (ideal case)	0.154	N/A
Controller without anti-windup action ($\gamma = 0$)	0.110	4.8
Controller + anti-windup compensator (2-DOF, $\gamma = 6$)	0.098	2.0

Table 17: Indexes obtained when using less informative training data.© 2020 IEEE

	$w_k = 0, \forall k$ in (6.13)	$\sigma^2 = 0.4$ in (6.13)
$\text{RMSE}_{\mathcal{M}}$	0.092	0.097
$\mathcal{OB}_{\%}$ [%]	1.4	0.4

and where the variance $\sigma^2 \in \mathbb{R}$ is a new parameter to be tuned. We stress that the quality of the results obtained by exploiting this approach strongly depends on the characteristics of the chosen noise sequence.

6.4 Simulation results

To show the effectiveness of the proposed data-driven scheme, we consider the problem of controlling the following two-dimensional Hammerstein-Wiener plant:

$$\Sigma_{\text{HW}} : \begin{cases} x_{k+1} = \begin{bmatrix} 0.755 & 0.250 \\ -0.199 & 0 \end{bmatrix} x_k + \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} \text{sign}(\tilde{u}_k)\sqrt{|\tilde{u}_k|}, \\ z_k = \begin{bmatrix} 0.699 & -0.443 \end{bmatrix} x_k, \\ y_k^{\circ} = z_k + 5\sin(z_k), \end{cases}$$

where $\tilde{u}_k$ is generated according to (6.2), with $\underline{u} = -1$ and $\bar{u} = 1$. In order to attain the closed-loop behavior described by the reference model

$$\mathcal{M} : \quad y_{k+1}^{\mathcal{M}} = -0.1 y_k^{\mathcal{M}} + 0.3 y_{k-1}^{\mathcal{M}} + 0.8 r_{k-1},$$

we learn a controller and an anti-windup compensator of order $Q = 4$ by using a set Z_N of $N = 10,000$ samples. These are generated by exciting the system with a sequence u_k obtained by superimposing a normally distributed white noise sequence with a periodic step signal with variable amplitude and period equal to 7. The output is corrupted by a white noise sequence $\{v_k\}_{k=1}^{N}$ with zero-mean Gaussian distribution and variance equal to 0.2, and the resulting signal is normalized using the empirical mean and standard deviation computed on the available data before starting the learning procedure. All the involved ANNs are chosen as compact networks featuring 4 nonlinear hidden layers of 15 neurons each and a final output layer of proper dimensions. We resort to the well-known *Rectified Linear Unit* (ReLU) maps [79] as the activation function of the neurons. The implementation of all the networks is carried out in Keras [80] with $F = 5$. In the learning phase, we initially set $\gamma = 0$ in (6.12c). Once this optimization procedure converges, we retrain both C_θ and H_φ by imposing $\gamma = 6$ and using the weights resulting from the previous phase as initial guesses for the new instance of the learning scheme. All the computations were carried out using a laptop equipped with a 2.8-GHz Intel Core i7 with 16 GB of RAM.

Figures 20 and 21 show the response of the designed scheme when we consider a reference sequence $\{r_k\}_{k=1}^{\tilde{N}}$ of length $\tilde{N} = 500$, where r_k is the piecewise-constant signal

$$r_k = \begin{cases} -1, \text{ if } 100(i-1)+1 \leq k \leq 100 \cdot i, \\ 1, \text{ otherwise,} \end{cases} \tag{6.15}$$

for $i = 1, 3, 5, \ldots$, and the output is corrupted by a noise with the same distribution of that acting on the training set. It is clear that the output obtained in closed-loop tracks the desired response and that the control input feeding the plant rarely exceeds the saturation bounds. Since our aim is to track the set point as similarly as possible to the reference model $\mathcal{M}$, while devising control inputs that exceeds as little as possible the saturation limits, we quantitatively assess the performance of the scheme by introducing the following indexes:

$$\text{RMSE}_{\mathcal{M}} = \sqrt{\frac{1}{\tilde{N}}\sum_{k=1}^{\tilde{N}}(y_k - y_k^{\mathcal{M}})^2}, \tag{6.16a}$$

$$\mathcal{OB}_{\%} = \frac{\#\{k \in [1,\ \tilde{N}] : u_k \notin [\underline{u},\ \bar{u}]\}}{\tilde{N}} \cdot 100\%, \tag{6.16b}$$

with $\text{RMSE}_{\mathcal{M}}$ and $\mathcal{OB}_{\%}$[1] respectively indicating how well the desired closed-loop behavior is tracked and how many times the actuator bounds are exceeded. The obtained results are reported in Table 16, along with the ones retrieved by considering the ideal case of unbounded actuators and a scheme designed by neglecting input saturation, *i.e.*, by setting $\gamma = 0$ in (6.12b) and not introducing the saturated inputs as regressive terms. It is clear that the achieved performances are comparable when considering the tracking capabilities of the different configurations, with the proposed 2-DOF architecture allowing us to reduce the number of times the actuator bounds are exceeded and slightly improve tracking performance.

We finally test the proposed approach in the presence of less informative data, by assuming that only saturated inputs are available to learn the controller with anti-windup. In this case, we compare the performance attained when the proposed 2-DOF architecture is designed with and without the additional data augmentation presented in Section 6.3.4. The obtained quality indexes are reported in Table 17, showing that the exploited strategy still allows us to obtain comparable closed-loop performance in terms of tracking, while resulting in a control input that exceeds the saturation bounds even less than the one obtained by learning the controller+anti-windup block with unsaturated inputs.

6.5 Conclusions

We presented a data-driven design method of neural controllers with anti-windup, by relying on a model-reference architecture approach. The

[1]Given a set $\mathcal{A}$, $\#\mathcal{A}$ indicates its cardinality.

proposed learning strategy only requires the knowledge of the actuator limits, and does not involve the identification of a full open-loop model of the plant to be controlled. As shown by the promising initial numerical results, the approach allows us to attain satisfactory performance in terms of reference tracking, in spite of bounded actuators. Future research will be devoted to study data-driven strategies for the selection of proper reference models, the development of systematic strategies to tune the relevant hyperparameters of the data augmentation strategy, and to assess the performance of the approach in more challenging scenarios.

Chapter 7

Conclusion

7.1 Summary of contributions

The objective of this thesis was to collect some results regarding the application of machine learning techniques to control theory, both with the objective of synthesizing novel techniques and of lessening taxing properties of already well-tested methods.

We first presented a system identification approach based on deep artificial neural networks that can learn state-space models of nonlinear systems. The approach is especially inviting for "control-oriented" applications due to its ability to produce models that can be immediately used in a Linear Time-Varying MPC scheme without computing sensitivities. A heuristic approach to tune the learning technique is also provided.

We then proposed a data-driven virtual sensor synthesis approach, inspired by the MMAE framework for reconstructing normally unmeasurable quantities such as scheduling parameters in parameter-varying systems, hidden modes in switching systems, and states of nonlinear systems. The main idea of this work was to verify if the classical model-based MMAE framework could be adapted to a data-driven setting. Furthermore, great attention has been devoted to making the approach as computationally affordable as possible in order to improve its compatibility with embedded and fast-sampling applications.

In the remaining Chapters, we focused on data-driven control applications. In Chapter 4 we presented an approach to learn a neural-networks-based short-term predictor of the behavior of nonlinear dynamical systems, which allows applying a standard model predictive control scheme to the identified plant without neither the need to learn a recursive Markovian model nor to compute sensitivities. In particular, a selling point of this approach is that it allows one to employ a standard LTV structure for the MPC controller, which can be very cheap from a computational point of view. Comparison with a state-of-the-art competing technique has shown the competitiveness of the approach.

Such an idea was further developed in Chapter 5, where the approach was employed to provide constraints handling capability to the well-known VRFT framework. Compared to the solution presented in Chapter 4, the short-term predictor employed in this Chapter also provides an improved capability to handle systems with non-affine input characteristics.

Finally, in Chapter 6, we proposed an alternative approach to synthesize VRFT controllers for constrained systems called "NAW-NET". Compared to the solution in Chapter 5, NAW-NET does not require to perform online any optimization, greatly simplifying its implementation on resource-constrained environments. The price to pay was to restrict its capability to a particular kind of constraint, namely actuator saturations, which are anyhow the most common kind of constraints found in real-world applications.

7.2 Open questions and future research directions

All the proposed ideas present a margin of improvement. Regarding the AE-based approach presented in Chapter 2, the most promising avenue is to explore its weaknesses and strengths compared to both Variational and Koopman-based approaches. This may allow assessing better its limitations and also provide more insightful guidelines to balance the performance and complexity of the resulting models. Another possible

extension is to assess the performance of the proposed approach when dealing with more complex kinds of sensor data, such as the ones presented in [29], [53]. A final avenue would be to investigate the stability and performance properties of the proposed LTV MPC scheme.

A natural extension to the virtual sensor proposed in Chapter 3 is to explore its capability in real-life prognostic and predictive maintenance applications. Moreover, considering its already limited memory and CPU footprint, its implementation in an embedded-friendly software package would also enable its deployment into a real world experimental setup. Another possibility would be to adapt the virtual sensor to closed-loop applications. For example, a possible idea would be to envision a scheme where the virtual sensor selects the correct controller from a set in a gain-scheduling-like scheme [93].

For the data-driven control techniques, the most compelling questions regard the overall stability of the proposed schemes and how to tune their relevant hyper-parameters. An attractive approach to the latter problem may be the use of global optimization methods, which have already found application in many learning and optimization contribution for exactly such purpose (see, for instance, [9]). Another possible research direction concerning such resource-aware techniques is, like for the virtual sensor, to develop the approaches into a library meant to be compatible with embedded platforms. This would allow one to verify the limits of their applicability in resource constrained environments and, if necessary, to address them.

Bibliography

[1] L. Lennart, "System identification: Theory for the user," *PTR Prentice Hall, Upper Saddle River, NJ*, pp. 1–14, 1999.

[2] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York, 2009.

[4] C. Andersson, A. H. Ribeiro, K. Tiels, N. Wahlström, and T. B. Schön, "Deep convolutional networks in system identification," in *2019 IEEE 58th Conference on Decision and Control*, IEEE, 2019, pp. 3670–3676.

[5] L. Ljung, C. Andersson, K. Tiels, and T. B. Schon, "Deep learning and system identification," in *Proceedings of the 21st IFAC World Congress*, 2020.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," *arXiv preprint arXiv: 1603.04467*, 2015, Software available from `http://www.tensorflow.org`.

[7] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA: ACM, 2016, pp. 2135–2135.

[8] A. Paszke, S. Gross, S. Chintala, *et al.*, *Automatic differentiation in PyTorch*, 2017.

[9] A. Bemporad, "Global optimization via inverse distance weighting and radial basis functions," *Computational Optimization and Applications*, vol. 77, pp. 571–595, 2020, Code available at `http://cse.lab.imtlucca.it/~bemporad/glis`.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[11] L. Ferrarotti and A. Bemporad, "Synthesis of optimal feedback controllers from data via stochastic gradient descent," in *2019 18th European Control Conference (ECC)*, 2019, pp. 2486–2491.

[12] S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2020.

[13] D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in *Proceedings of European Control Conference*, Naples, Italy, 2019, pp. 1494–1499.

[14] A. Cauligi, P. Culbertson, B. Stellato, D. Bertsimas, M. Schwager, and M. Pavone, "Learning mixed-integer convex optimization strategies for robot planning and control," in *Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC)*, Jeju Island, Republic of Korea, 2020, pp. 1698–1705.

[15] M. Klaučo, M. Kalúz, and M. Kvasnica, "Machine learning-based warm starting of active set methods in embedded model predictive control," *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 1–8, 2019.

[16] D. Masti and A. Bemporad, "Learning nonlinear state–space models using autoencoders," *Automatica*, vol. 129, p. 109 666, 2021.

[17] D. Masti, D. Bernardini, and A. Bemporad, "A machine-learning approach to synthesize virtual sensors for parameter-varying systems," *European Journal of Control*, vol. 61, pp. 40–49, 2021, ISSN: 0947-3580.

[18] ——, "A machine-learning approach to synthesize virtual sensors for parameter-varying systems," *arXiv preprint arXiv:2103.12324*, 2021.

[19] D. Masti, F. Smarra, A. D'Innocenzo, and A. Bemporad, "Learning affine predictors for MPC of nonlinear systems via artificial neural networks," in *Proceedings of the 21st IFAC World Congress*, 2020.

[20] D. Masti, V. Breschi, S. Formentin, and A. Bemporad, "Direct data-driven design of neural reference governors," in *Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC)*, Jeju Island, Republic of Korea, 2020, pp. 4955–4960.

[21] V. Breschi, D. Masti, S. Formentin, and A. Bemporad, "NAW-NET: Neural anti-windup control for saturated nonlinear systems," in *Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC)*, Jeju Island, Republic of Korea, 2020, pp. 3335–3340.

[22] D. Masti, D. Bernardini, and A. Bemporad, "Learning virtual sensors for estimating the scheduling signal of parameter-varying systems," in *27^{th} Mediterranean Conference on Control and Automation (MED)*, Akko, Israel: IEEE, 2019, pp. 232–237.

[23] D. Masti, T. Pippia, A. Bemporad, and B. De Schutter, "Learning approximate semi-explicit hybrid MPC with an application to microgrids," in *Proceedings of the 21st IFAC World Congress*, 2020.

[24] D. Masti, M. Zanon, and A. Bemporad, "Tuning LQR controllers: A sensitivity-based approach," *IEEE Control Systems Letters*, vol. 6, pp. 932–937, 2022. DOI: 10.1109/LCSYS.2021.3087556.

[25] J. Schoukens and L. Ljung, "Nonlinear system identification: A user-oriented road map," *IEEE Control Systems Magazine*, vol. 39, no. 6, pp. 28–99, 2019.

[26] G. Pillonetto, F. Dinuzzo, T. Chen, G. D. Nicolao, and L. Ljung, "Kernel methods in system identification, machine learning and function estimation: A survey," *Automatica*, vol. 50, no. 3, pp. 657–682, 2014, ISSN: 0005-1098.

[27] V. Breschi, D. Piga, and A. Bemporad, "Piecewise affine regression via recursive multiple least squares and multicategory discrimination," *Automatica*, vol. 73, pp. 155–162, Nov. 2016.

[28] Y. Wang, "A new concept using LSTM neural networks for dynamic system identification," in *2017 American Control Conference*, IEEE, 2017, pp. 5324–5329.

[29] Q. Lu and V. M. Zavala, "Image-based model predictive control via dynamic mode decomposition," *arXiv preprint arXiv: 2006.06727*, 2020.

[30] T. Simpson, N. Dervilis, and E. Chatzi, "On the use of nonlinear normal modes for nonlinear reduced order modelling," *arXiv preprint arXiv:2007.00466*, 2020.

[31] C. Wehmeyer and F. Noé, "Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics," *The Journal of chemical physics*, vol. 148, no. 24, p. 241 703, 2018.

[32] D. P. Kingma and M. Welling, *An Introduction to Variational Autoencoders*. Now Foundations and Trends, 2019.

[33] M. Karl, M. Soelch, J. Bayer, and P. Van der Smagt, "Deep variational bayes filters: Unsupervised learning of state space models from raw data," *arXiv preprint arXiv:1605.06432*, 2016.

[34] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Advances in neural information processing systems*, 2015, pp. 2746–2754.

[35] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep Kalman filters," *arXiv preprint arXiv:1511.05121*, 2015.

[36] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep state space models for nonlinear system identification," *arXiv preprint arXiv:2003.14162*, 2020.

[37] R. K. Kandukuri, J. Achterhold, M. Möller, and J. Stückler, "Learning to identify physical parameters from video using differentiable physics," *arXiv preprint arXiv: 2009.08292*, 2020.

[38] M. Okada and T. Taniguchi, "Dreaming: Model-based reinforcement learning by latent imagination without reconstruction," *arXiv preprint arXiv: 2007.14535*, 2020.

[39] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2016, pp. 3928–3934.

[40] Q. Li, F. Dietrich, E. M. Bollt, and I. G. Kevrekidis, "Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 10, p. 103 111, 2017.

[41] Y. Xiao, X. Zhang, X. Xu, X. Liu, and J. Liu, "A deep learning framework based on Koopman operator for data-driven modeling of vehicle dynamics," *arXiv preprint arXiv:2007.02219*, 2020.

[42] Z. Ping, Z. Yin, X. Li, Y. Liu, and T. Yang, "Deep Koopman model predictive control for enhancing transient stability in grids," *International Journal of Robust and Nonlinear Control*, 2020.

[43] N. Takeishi, Y. Kawahara, and T. Yairi, "Learning Koopman invariant subspaces for dynamic mode decomposition," in *Advances in Neural Information Processing Systems*, 2017, pp. 1130–1140.

[44] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature communications*, vol. 9, no. 1, pp. 1–10, 2018.

[45] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019.

[46] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in neural information processing systems*, 2018, pp. 7785–7794.

[47] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 3601–3610.

[48] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, 2015, pp. 2980–2988.

[49] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[50] S. E. Otto and C. W. Rowley, "Linearly recurrent autoencoder networks for learning dynamics," *SIAM Journal on Applied Dynamical Systems*, vol. 18, no. 1, pp. 558–593, 2019.

[51] D. Masti and A. Bemporad, "Learning nonlinear state-space models using deep autoencoders," in *Proceedings of 2018 57th Conference on Decision and Control (CDC)*, Miami Beach, FL, USA, 2018, pp. 3862–3867.

[52] G. Beintema, R. Toth, and M. Schoukens, "Nonlinear state-space identification using deep encoder networks," *arXiv preprint arXiv: 2012.07697*, 2020.

[53] G. I. Beintema, R. Toth, and M. Schoukens, "Non-linear state-space model identification from video data using deep encoders," *arXiv preprint arXiv: 2012.07721*, 2020.

[54] J. Drgona, A. R. Tuor, V. Chandan, and D. L. Vrabie, "Physics-constrained deep learning of multi-zone building thermal dynamics," *arXiv preprint arXiv: 2011.05987*, 2020.

[55] R. L. Williams and D. A. Lawrence, "Minimal realizations," in *Linear State-Space Control Systems*. John Wiley Sons, Inc., 2007, pp. 185–197, ISBN: 9780470117873.

[56] Y. Baram, "Minimal order representation, estimation and feedback of continuous-time stochastic linear systems," in *Mathematical Theory of Networks and Systems*, Springer, 1984, pp. 24–41.

[57] I. Guyon and A. Elisseeff, "An introduction to feature extraction," in *Feature extraction*, Springer, 2006, pp. 1–25.

[58] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

[59] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[60] "Mean absolute error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., Springer US, 2010, pp. 652–652, ISBN: 978-0-387-30164-8.

[61] P. J. Werbos *et al.*, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[63] M. Forgione and D. Piga, "Model structures and fitting criteria for system identification with neural networks," in *Proceedings of 2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, 2020, pp. 1–6.

[64] G. V. Puskorius and L. A. Feldkamp, "Truncated backpropagation through time and Kalman filter training for neurocontrol," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, IEEE, vol. 4, 1994, pp. 2488–2493.

[65] M. Forgione and D. Piga, "Continuous-time system identification with neural networks: Model structures and fitting criteria," *European Journal of Control*, 2021, ISSN: 0947-3580.

[66] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[67] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[68] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, " Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[69] A. Bemporad, "Model-based predictive control design: New trends and tools," in *2006 IEEE Conference on Decision and Control and European Control Conference*, San Diego, CA, 2006, pp. 6678–6683.

[70] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[71] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[72] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear MPC: Bridging the gap via the real-time iteration," *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020.

[73] M. Schoukens and J. P. Noël, "Three benchmarks addressing open challenges in nonlinear system identification," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 446–451, 2017.

[74] J. Wang, A. Sano, T. Chen, and B. Huang, "Identification of Hammerstein systems without explicit parameterisation of non - linearity," *International Journal of Control*, vol. 82, no. 5, pp. 937–952, 2009.

[75] The MathWorks, Inc., *Nonlinear modeling of a magneto-rheological fluid damper*, `https://mathworks.com/help/ident/ug/nonlinear-modeling-of-a-magneto-rheological-fluid-damper.html`, 2020.

[76] ——, *Two tank system: C MEX-file modeling of time-continuous siso system*, `https://it.mathworks.com/help/ident/ug/two-tank-system-c-mex-file-modeling-of-time-continuous-siso-system.html`, 2020.

[77] T. Wigren and J. Schoukens, "Three free data sets for development and benchmarking in nonlinear system identification," in *2013 European control conference (ECC)*, IEEE, 2013, pp. 2933–2938.

[78] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and S. H. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, p. 947, 2000.

[79] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[80] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[81] S. K. Sashank J. Reddi Satyen Kale, "On the convergence of Adam and beyond," *Internation Conference on Learning Representations*, 2018.

[82] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 2951–2959.

[83] The MathWorks, Inc., *System identification toolbox*, Natick, Massachusetts, United State, 2019.

[84] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[85] N. Saraf and A. Bemporad, "A bounded-variable least-squares solver based on stable QR updates," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1242–1247, 2020.

[86] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[87] F.-R. López-Estrada, D. Rotondo, and G. Valencia-Palomo, "A review of convex approaches for control, observation and safety of linear parameter varying and Takagi-Sugeno systems," *Processes*, vol. 7, no. 11, p. 814, 2019.

[88] R. Tóth, *Modeling and Identification of Linear Parameter-Varying Systems*. Springer, Berlin, Heidelberg, 2010, ISBN: 9783642138119.

[89] F. D. Torrisi and A. Bemporad, "HYSDEL — A tool for generating computational hybrid models," *IEEE Trans. Contr. Systems Technology*, vol. 12, no. 2, pp. 235–249, Mar. 2004.

[90] D. Rotondo, V. Puig, J. Acevedo Valle, and F. Nejjari, "FTC of LPV systems using a bank of virtual sensors: Application to wind turbines," in *Proceedings of Conference on Control and Fault-Tolerant Systems*, 2013, pp. 492–497.

[91] M. Witczak, *Fault diagnosis and fault-tolerant control strategies for non-linear systems*. Springer, 2014, vol. 266.

[92] J. Guzman, F.-R. López-Estrada, V. Estrada-Manzo, and G. Valencia-Palomo, "Actuator fault estimation based on a proportional-integral observer with nonquadratic Lyapunov functions," *International Journal of Systems Science*, pp. 1–14, 2021.

[93] H. Khalil, *Nonlinear Systems*. Prentice Hall, 2002.

[94] M. Misin and V. Puig, "LPV MPC control of an autonomous aerial vehicle," in *2020 28th Mediterranean Conference on Control and Automation (MED)*, IEEE, 2020, pp. 109–114.

[95] M.-H. Do, D. Koenig, and D. Theilliol, "H_∞ observer design for singular nonlinear parameter-varying system," in *Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC)*, Jeju Island, Republic of Korea, 2020, pp. 3927–3932.

[96] K. J. Keesman, *System identification: an introduction*. Springer-Verlag London, 2011.

[97] M. Milanese, C. Novara, K. Hsu, and K. Poolla, "The filter design from data (FD2) problem: Nonlinear set membership approach," *Automatica*, vol. 45, no. 10, pp. 2350–2357, 2009.

[98] T. Poggi, M. Rubagotti, A. Bemporad, and M. Storace, "High-speed piecewise affine virtual sensors," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1228–1237, 2012.

[99] E. Marchi, F. Vesperini, F. Eyben, S. Squartini, and B. Schuller, "A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015, pp. 1996–2000.

[100] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–A decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[101] M. D. Morse and J. M. Patel, "An efficient and accurate method for evaluating time series similarity," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, ACM, 2007, pp. 569–580.

[102] A. Akca and M. Ö. Efe, "Multiple model Kalman and particle filters and applications: A survey," *IFAC-PapersOnLine*, vol. 52, no. 3, pp. 73–78, 2019.

[103] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[104] B. N. Alsuwaidan, J. L. Crassidis, and Y. Cheng, "Generalized multiple-model adaptive estimation using an autocorrelation approach," *IEEE Transactions on Aerospace and Electronic systems*, vol. 47, no. 3, pp. 2138–2152, 2011.

[105] X.-R. Li and Y. Bar-Shalom, "Multiple-model estimation with variable structure," *IEEE Transactions on Automatic control*, vol. 41, no. 4, pp. 478–493, 1996.

[106] J. Feng and Z.-H. Zhou, "Autoencoder by forest," in *Proceedings of the AAAI conference on artificial intelligence*, 2018.

[107] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[108] A. Bemporad, V. Breschi, D. Piga, and S. P. Boyd, "Fitting jump models," *Automatica*, vol. 96, pp. 11–21, 2018.

[109] P. Mellodge, *A Practical Approach to Dynamical Systems for Engineers*. Woodhead Publishing, 2015.

[110] S. M. Shinners, *Modern Control System Theory and Design*, 2nd. New York, NY, USA: John Wiley & Sons, Inc., 1998.

[111] "Generative and discriminative learning," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer US, 2010, pp. 454–455.

[112] P. D. Hanlon and P. S. Maybeck, "Multiple-model adaptive estimation using a residual correlation Kalman filter bank," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 2, pp. 393–406, 2000.

[113] Y. Gao, L. Zhu, H.-D. Zhu, Y. Gan, and L. Shang, "Extract features using stacked denoised autoencoder," in *Intelligent Computing in Bioinformatics*, Springer International Publishing, 2014, pp. 10–14, ISBN: 978-3-319-09330-7.

[114] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, Springer, 2003, pp. 63–71.

[115] T. L. Vincent, C. Galarza, and P. P. Khargonekar, "Adaptive estimation using multiple models and neural networks," *IFAC Proceedings Volumes*, vol. 31, no. 29, pp. 149–154, 1998.

[116] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.

[117] R. L. Marchese Robinson, A. Palczewska, J. Palczewski, and N. Kidley, "Comparison of the predictive performance and interpretability of random forest and linear models on benchmark data sets," *Journal of Chemical Information and Modeling*, vol. 57, no. 8, pp. 1773–1792, 2017.

[118] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[119] V. Breschi and M. Mejari, "Shrinkage strategies for structure selection and identification of piecewise affine models," in *Proceedings of 2020 59th IEEE Conference on Decision and Control (CDC)*, Jeju Island, Republic of Korea, 2020, pp. 1626–1631.

[120] X. R. Li, X. Zwi, and Y. Zwang, "Multiple-model estimation with variable structure. iii. model-group switching algorithm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 35, no. 1, pp. 225–241, 1999.

[121] L. Buitinck, G. Louppe, M. Blondel, *et al.*, "API design for machine learning software: Experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[122] "Mean squared error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 653–653, ISBN: 978-0-387-30164-8.

[123] D. Liberzon, *Switching in systems and control*. Springer Science & Business Media, 2003.

[124] Y. Sasaki *et al.*, "The truth of the F-measure," *Teach Tutor mater*, vol. 1, no. 5, pp. 1–5, 2007.

[125] D. Ali, S. Mukhopadhyay, H. Rehman, and A. Khurram, "UAS based Li-ion battery model parameters estimation," *Control Engineering Practice*, vol. 66, pp. 126–145, 2017.

[126] H. Abdi, B. Mohammadi-ivatloo, S. Javadi, A. R. Khodaei, and E. Dehnavi, "Energy storage systems," in *Distributed Generation Systems*, G. Gharehpetian and S. M. Mousavi Agah, Eds., Butterworth-Heinemann, 2017, pp. 333–368, ISBN: 978-0-12-804208-3.

[127] B. Anderson and J. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[128] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[129] J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith, "Dynamic systems identification with gaussian processes," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 11, no. 4, pp. 411–424, 2005.

[130] D. Piga, S. Formentin, and A. Bemporad, "Direct data-driven control of constrained systems," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1422–1429, 2017.

[131] A. Jain, F. Smarra, and R. Mangharam, "Data predictive control using regression trees and ensemble learning," in *Proceedings of the 56th Annual Conference on Decision and Control*, IEEE, 2017, pp. 4446–4451.

[132] A. Jain, F. Smarra, M. Behl, and R. Mangharam, "Data-driven model predictive control with regression trees-An application to building energy management," *ACM Transactions on Cyber-Physical Systems*, vol. 2, no. 1, p. 4, 2018.

[133] A. Afram, F. Janabi-Sharifi, A. S. Fung, and K. Raahemifar, "Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system," *Energy and Buildings*, vol. 141, pp. 96–113, 2017.

[134] M. Behl, F. Smarra, and R. Mangharam, "Dr-advisor: A data-driven demand response recommender system," *Applied Energy*, vol. 170, pp. 30–46, 2016.

[135] F. Smarra, A. Jain, T. de Rubeis, D. Ambrosini, A. D'Innocenzo, and R. Mangharam, "Data-driven model predictive control using random forests for building energy optimization and climate control," *Applied Energy*, vol. 226, 2018.

[136] E. Terzi, L. Fagiano, M. Farina, and R. Scattolini, "Learning-based predictive control for linear systems: A unitary approach," *Automatica*, vol. 108, p. 108 473, 2019.

[137] G. Liu and V. Kadirkamanathan, "Predictive control for non-linear systems using neural networks," *International Journal of Control*, vol. 71, no. 6, pp. 1119–1132, 1998.

[138] F. Smarra, A. Jain, R. Mangharam, and A. D'Innocenzo, "Data-driven switched affine modeling for model predictive control," in *IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'18)*, IFAC, 2018, pp. 199–204.

[139] F. Smarra, G. D. Di Girolamo, V. De Iuliis, A. Jain, R. Mangharam, and A. D'Innocenzo, "Data-driven switching modeling for mpc using regression trees and random forests," *Nonlinear Analysis: Hybrid Systems*, vol. 36, 2020.

[140] J. A. Suykens, "Deep restricted kernel machines using conjugate feature duality," *Neural computation*, vol. 29, no. 8, pp. 2123–2163, 2017.

[141] F. Bünning, B. Huber, P. Heer, A. Aboudonia, and J. Lygeros, "Experimental demonstration of data predictive control for energy optimization and thermal comfort in buildings," *Energy and Buildings*, vol. 211, p. 109 792, 2020.

[142] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[143] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[144] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[145] L. Ljung, *System Identification Toolbox for MATLAB – User's Guide*. The Mathworks, Inc., 2001.

[146] M. H. Beale, M. T. Hagan, and H. B. Demuth, *Deep Learning Toolbox – User's Guide*. The Mathworks, Inc., 2018.

[147] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven MPC design," *IEEE control systems letters*, vol. 3, no. 3, pp. 577–582, 2019.

[148] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, pp. 3–35, 2013.

[149] S. Formentin, K. Van Heusden, and A. Karimi, "A comparison of model-based and data-driven controller tuning," *International Journal of Adaptive Control and Signal Processing*, vol. 28, no. 10, pp. 882–897, 2014.

[150] S. Formentin, S. Savaresi, and L. Del Re, "Non-iterative direct data-driven controller tuning for multivariable systems: Theory and application," *IET control theory & applications*, vol. 6, no. 9, pp. 1250–1257, 2012.

[151] M. Campi, A. Lecchini, and S. Savaresi, "Virtual reference feedback tuning: A direct method for the design of feedback controllers," *Automatica*, vol. 38, no. 8, pp. 1337–1346, 2002.

[152] S. Formentin, D. Piga, R. Tóth, and S. M. Savaresi, "Direct learning of LPV controllers from data," *Automatica*, vol. 65, pp. 98–110, 2016.

[153] V. Breschi and S. Formentin, "Direct data-driven design of switching controllers," *International Journal of Robust and Nonlinear Control*, 2019.

[154] M. Radac, R. Precup, and R. Roman, "Data-driven model reference control of MIMO vertical tank systems with model-free VRFT and Q-learning," *ISA transactions*, vol. 73, pp. 227–238, 2018.

[155] A. Sadeghzadeh and H. Momeni, "Virtual closed loop identification: A new method for low-order H_∞ controller design," *IFAC Proceedings Volumes*, vol. 42, no. 10, pp. 314–319, 2009.

[156] A. Bemporad, "Reference governor for constrained nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 3, pp. 415–419, 1998.

[157] M. Farina and L. Piroddi, "Simulation error minimization identification based on multi-stage prediction," *International Journal of Adaptive Control and Signal Processing*, vol. 25, no. 5, pp. 389–406, 2011.

[158] G. Cimini and A. Bemporad, "Exact complexity certification of active-set methods for quadratic programming," *IEEE Transactions on Automatic Control*, vol. 62, no. 12, pp. 6094–6109, 2017.

[159] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2013.

[160] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5595–5637, 2017.

[161] V. Breschi and S. Formentin, "Virtual reference feedback tuning with data-driven reference model selection," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, vol. 120, PMLR, 2020, pp. 37–45.

[162] A. Bemporad, "A numerically stable solver for positive semidefinite quadratic programs based on nonnegative least squares," *IEEE Transactions on Automatic Control*, vol. 63, no. 2, pp. 525–531, 2017.

[163] B. Recht, *A Tour of Reinforcement Learning: The View from Continuous Control*, 2018. eprint: `arXiv:1806.09460`.

[164] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using Reinforcement Learning: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2042–2062, 2018.

[165] S. Formentin, M. C. Campi, A. Carè, and S. M. Savaresi, "Deterministic continuous-time virtual reference feedback tuning (VRFT) with application to PID design," *Systems & Control Letters*, vol. 127, pp. 25–34, 2019.

[166] L. Zaccarian and A. Teel, *Modern Anti-windup Synthesis: Control Augmentation for Actuator Saturation.* Princeton University Press, 2011.

[167] S. Formentin, F. Dabbene, R. Tempo, L. Zaccarian, and S. M. Savaresi, "Robust linear static anti-windup with probabilistic certificates," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1575–1589, 2016.

[168] F. Todeschini, S. Formentin, G. Panzani, M. Corno, S. M. Savaresi, and L. Zaccarian, "Nonlinear pressure control for bbw systems via dead-zone and antiwindup compensation," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 4, pp. 1419–1431, 2015.

[169] P. Yan, D. Liu, D. Wang, and H. Ma, "Data-driven controller design for general MIMO nonlinear systems via virtual reference feedback tuning and neural networks," *Neurocomputing*, vol. 171, pp. 815–825, 2016.