博士論文

Deep Neural Evolution of Inverse Models for Predictive Control

(予測制御における逆モデルのためのディープニューロ進化)

Morales Perez Edgar Ademir

モラレス　ペレス　エドガル　アデミル

指導教員 伊庭斉志 教授

東京大学大学院 情報理工学系研究科 電子情報学専攻

This dissertation is submitted for the degree of

Doctor of Philosophy

December 2021

ABSTRACT

The Control Theory is a branch of applied mathematics and engineering that forces physical dynamic systems using feedback information with specific and strict boundaries of time and magnitude. Often, the problem is simplified to force stability on physical problems, but most applications involve regulation and close monitoring of measurable variables.

In this dissertation, we propose a set of control algorithms based on Deep-learning and the concept of Inverse dynamics. Moreover, an evolutionary-based optimization serves as a foundation for the proper tunning required by complicated problems, such as nonlinear cases.

Beyond applying deep-learning architectures that directly learn how to control a system, we bound such solutions and combine them with a conventional predictive control setting. Our objective is to provide a realistically applicable intelligence-based algorithm that solves the control problem without the black-box nature concerns typically found in Deep-learning methods.

Model Predictive Control (MPC) is a collection of algorithm techniques designed to regulate and manage the dynamic operation of Nonlinear Systems by using future predictions and optimization methods. Rather than a specific set of rules, MPC is a closed-loop concept of sequential optimization in a moving horizon in the foreseeable future. To calculate future information of a controlled system, MPC utilizes a mathematical representation called Predictor Model. The Predictor model computes an open-loop response, and with that information, the rest of the problem relies on optimizing a signal to drive the system state to the desired value.

Acting based on future predictions only makes the MPC a reliable method if such forecasted values are correct. Thus, the Predictor Model is a fundamental element in MPC implementations. Furthermore, the Predictor Model accuracy presents a significant problem in MPC and the Control Theory in general.

Hence, this work proposes a combination of data-based methods and conventional predictive control schemes. The Predictor Model capabilities are enhanced by Deep-learning models, which are proven excellent nonlinear function approximators. Moreover, being MPC a highly computational complex algorithm, an Inverse model initialization is employed to increase the system's quality while reducing the convergence times.

Fast MPC implementations that comply with the strict industrial requirements are our ultimate goal. We prove through numerical simulations that the use of optimally evolved Inverse models dramatically reduces the necessary steps to compute optimal control signals, creating a possible scenario of practical implementations of predictive control systems to the previously prohibited fast dynamics systems.

要旨

制御理論は応用数学と工学の一分野であり、時間と空間に特定の境界を有するフィードバック情報を使用して物理力学系の制御を試みる。多くの場合に問題は物理的な安定性を強制するため単純化されるが、ほとんどの応用例では測定可能な変数の調整と綿密な監視が必要とされる。

本論文では、深層学習と逆ダイナミクスの概念を統合した制御アルゴリズムを提案する。さらに、進化計算に基づく最適化により、非線形系などの複雑な問題に必要な調整機能を的確に実現する。

この研究では、深層学習をシステムの制御に直接適用するだけでなく、そこで得られた解を従来の予測制御設定と組み合わせるフレームワークを構築する。その目的は、一般に深層学習手法で見られるブラックボックス的な性質を回避し、制御問題を解決するために現実的に適用可能な知的システムの基盤を提供することである。

モデル予測制御（Model Predictive Control: MPC）は、将来の予測と最適化の方法を使用して、非線形システムの動的な動作を調整および管理するために設計されたアルゴリズム手法の 1 つである。MPC はある決まった範囲を予測するための特定のルールセットではなく、逐次最適化の閉ループであり、将来の可変領域を予見可能にするものである。

制御システムの将来情報を導出するために、MPC は予測モデルと呼ばれる数学的表現を利用する。予測モデルは開ループ応答を計算し、その情報を使用して制御信号を最適化し、目的とする状態に系を近づける。

将来予測に基づいて行動するので、予測値が正しい場合にのみ MPC は信頼できる方法となる。つまり、MPC の基本要素である予測モデルの精度は、MPC および一般的な制御理論において非常に重要である。

したがって、本論文では、的確な訓練データベースの構築法と従来の予測制御との統合フレームワークを提案する。予測モデルの機能は、優れた非線形関数近似器である深層学習モデルによって強化される。一般に、MPC は非常に計算が複雑なアルゴリズムである。そのため本研究では、逆モデルの初期化を利用することで、収束時間を短縮しつつ予測制御の品質を向上させることに成功した。

以上要するに、厳格な産業要件に準拠した高速 MPC の実現が本研究の目標である。学習で進化した逆モデルを利用すると、制御信号を計算するために必要な手順が大幅に削減され、従来は困難であった動的システムに対する予測制御の構築が可能なことを実験的に検証した。

Acknowledgements

Firstly, my deepest gratitude to my advisor, Professor Hitoshi Iba, for the guidance, trust, and mentorship over all these years. The experiences and the warmth I found at Iba-lab were the most rewarding of my life in Japan.

I would like to mention my gratitude for the mentoring, comments, revision and the time of Professors Toshihiko Yamasaki, Yoshimasa Tsuruoka , Yoshihiko Hasegawa and Gouhei Tanaka. The present work was greatly improved by their advice and wisdom. Thank you very much.

I would also like to extend my appreciation to the University of Tokyo staff. The patience the hardworking people showed every day made my academic life so much easier. Also, I would like to thank the Ministry of Education, Culture, Sports, Science and Technology, and the government of Japan for one of the most incredible opportunities in my life. Thank you very much.

I would like to express my gratitude to my dearests colleagues, Adam, Ysmaldo, Oscar, and Surya, for their advice, camaraderie, discussions, and above all, their friendship. There are no words to express how grateful I am.

In addition, I would like to thank my parents Miguel and Leticia, for their counsel, love, and caring words, even from so far away. Furthermore, for their constant support, my brothers Omar, Micky, Beto, Manzana, my cousins Martha and Alberto.

Finally, to the love of my life, Nathaly, I dedicate this work. Without your aid, none of this would have been possible. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Control Problem

To talk about the Control Problem is to discuss the broad area of dynamic systems analysis. For example, mathematical representations can describe physical systems, which formulate the relationship between causes and effects in an input and output setting. These models describe the overall evolution of observed variables in a dynamic system with a certain amount of detail, and as the management motto says: "What gets measured gets managed."

Controlling a system requires a deep understanding of the physical problem. Thus, the standard workflow begins with building a mathematical representation. The dynamic system analysis is the area that started the generation of mathematical models representing physical systems, with methods ranging from algebraic analysis, going through differential equations, to stochastic models. The first control elements emerged with the model concept and knowledge of the causal variables involved.

However, it was until the emergence of one of the most revolutionary ideas in engineering that the shape and branch of Control theory commenced: The negative feedback. Measuring observable variables and using their information to compute the difference between the desired quantities; so a control element can correct the system is known as a control system in the general sense.

With automatic control development, negative feedback, and closed-loop systems, instability phenomena, transitory shapes, amplified noise, and stationary errors became the object of study, and the focus of control algorithms expanded.

It is an ongoing research problem to modify and force the physical systems' evolution through time in the desired manner, not only in magnitude but also in transitory shapes and stability. The design of control algorithms directly impacts all areas of engineering: From management systems or electrical energy generation to space exploration and robotics. Any system requiring automatic reactions or adjustments requires a control system setup and a control algorithm.

The general description of an automatic control system is the continuous adjustment of an observed variable. The algorithm calculates such adjustments in real-time, using the negative feedback measurements from the observed variables, redirecting the system's internal and observable variables to desired states. Conversely, as systems become more complex in time, the necessity of additional mathematical representations and increases in dimensionality generate the need for more robust approaches rather than the usual proportional response to an error calculation.

As artificial intelligence methods offered alternatives to conventional and analytical techniques in many areas of engineering, the control of dynamic systems became one of the first successful applications of deep learning, at least in the theoretical context.

1.1.1 Control Theory

The following definitions simplify the understanding of the Control Theory to continue discussing the dynamic systems we aim to manipulate with the proposed algorithms. Additionally, they will serve as the departure point for the rest of our contributions:

Dynamic System. The abstract representation of a physical system through parametric realizations. Often, these systems are described by differential equations with time derivatives, able to compute future internal values or states from their current states. In other words, they are functions that map the evolution through time of a relationship between dependent and independent variables in time.

State variables. The internal variables of a dynamic system, represented as a vector in the geometric space, are known as state variables or state vectors. These variables describe the system and are not necessarily accessible for measurement. In some cases, especially in complex systems, these values must be estimated.

Observable variables. All measurable variables of a dynamic system. These values are accessible for negative feedback and constitute the source of information from which the control algorithm will base its decisions. From these observable variables, internal information of the dynamic system can be estimated, such as the state vector.

Input variables. The system states and observable variables change as a function of time, but an excitation signal starts this mechanism. Such signals are inputs or manipulable variables since they are the direct access users have to the system. Control algorithms compute the necessary values that will serve as inputs to the controlled systems.

Disturbances. Input variables induce the evolution of the dynamic systems through time, changing their internal state vector and generating observable, measurable outputs. However, since interactions between a system and its environment exist, all direct or indirect elements affecting the progression of a system are known as disturbances. Although most disturbances are related to the dynamic system's interaction with its environment, measurement noise and uncertainties are also considered disturbances in the control theory context.

Closed-loop and Open-loop. A system with a continuous comparison between desired values of the state vector and the measurable, observable variables ultimately used to control a dynamic system is a closed-loop setup. On the contrary, a controlled system without measured feedback is known as an open-loop. Closed-loop setups are the majority of control systems since the increased accuracy achieved greatly overcomes the stability issues. Open-loop controls are easier to design, but since no information is measured in the system's execution, their reliability makes them unsuitable for most tasks. In this dissertation, we work on closed-loop control systems exclusively.

Finally, any system within a closed-loop with continuous regulation is called a controlled system. Some authors refer to controlled systems as *plants* or *feedback systems*. Similarly, it is a common occurrence to use these terms interchangeably. Nonetheless, in this dissertation we use the conventional term: **controlled system**.

1.1.2 Intelligent Control

Most conventional control algorithms are designed based on the premise that the dynamic system model is known and well-understood. Additionally, a simplistic consideration of the model uncertainty is added only to selected methods, such as the robust control. Furthermore, the more the general advances of engineering expand, the more the complexity of automatic systems increase.

Consequently, modeling and developing control systems is an ever-increasing task, both in complexity and difficulty. Moreover, industry requirements have become more demanding. These reasons constitute the superficial point where data-based methods started to gain importance. Learning the inner dynamics buried in data and nonlinear information often overlooked by the systems designers has proven an effective alternative to the conventional analytical approach.

One of the most successful methods is known as Fuzzy Logic. This technique is a formalization and condensation of a set of expert rules, real-life experience, and in some instances, common sense. Generally, *linguistic* variables and IF-THEN clauses describe these points. This level of artificial intelligence can model system dynamics and control them. The main difference between Fuzzy controls is its task-oriented nature rather than a set-point oriented.

Fuzzy systems have a long history of successful implementations. The most prevalent techniques are the Takagi-Sugeno[103], and Mandani [64] algorithms. As this work studies a specific point of Intelligent Control, the Deep Learning-based, we will not expand the Fuzzy logic knowledge. However, it is essential to note its relevance in the field, and the reader is encouraged to consider Fuzzy-methods and the review papers [74, 95, 18].

The evolution of data-based methods moved towards Neural Networks (NN). Some of the first successful applications of NN were in the Control Theory field, and physical system modeling [120]. Since this type of nonlinear model can approximate complex functions, the first usages assisted in identifying complex dynamics. The combination of tools to define and generate mathematical models from data is known as **System Identification** [93].

Generating a model from data using NN is also referred to as Deep System Identification in more contemporary settings. Nevertheless, the main components of the learning process remain the same. The identification process is refactored into a supervised learning approach, where the labeled data is the state vector, and the features are the input vector.

Beyond identifying nonlinear systems, supervised learning algorithms can also be used to learn control signals directly. Such approaches have a powerful premise consisting of the combination of adaptive and optimal control features. For instance, a trending method at the moment of writing is **Reinforcement Learning** (RL). A random exploration of a system provides an iterative method that learns the internal dynamic states with the required control signals to drive it successfully to desired values.

RL methods are heavily influenced by the first iterations of adaptive and optimal controls [57]. In fact, these methods solve the optimal control problem online in time-varying systems, for which there are solid arguments for the future of Control Theory and RL. However, regardless of the effectiveness of such methods in many areas, the control of dynamic systems has specific and strict requirements that may be prohibitive for random online explorations of *in production* systems. Moreover, black-box methods have small guarantees for continuous operation in all required ranges of physical systems. Our approach to this matter combines techniques to exploit the ability to learn plus the certainty of well-proven control methods.

Another powerful technique referred to as **Inverse Model** control proposes perfect opposite signals to those of a dynamic system. The transition of a time step to another would encounter the opposite value, causing a *cancellation*. Since canceled variables in the algebraic context mean, in reality, a unity result, the observable variables of the dynamic system will be the same as the input variables, without any transitory changes. Such interaction is purely ideal; hence this kind of method is called *perfect control.*

As mentioned earlier, dynamic systems have an erratic, nonlinear, and in some instances chaotic nature. Therefore, the existence of an Inverse of a describing nonlinear function is not guaranteed. Furthermore, even if there is an inverse function, the causality, stability, and other factors may be beyond the reasonable application zone. Nevertheless, considering the approximation features of Deep NN (DNN), an Inverse model may be learned from data.

1.2 Related works

The field of intelligent control applications is extensive and has a long history, dating from the first solutions provided by neural networks. In addition, the universal approximation feature of neural networks made them suitable for control problems. As the Inverse model of a system is considered the *ideal* control, several approaches have attempted to emulate an ideal control, from analytical attempts to data-based methods.

As pointed in recent survey compendiums [32], the nonlinear MPC challenges lie in the effectiveness and quality of the predictions for the ever-increasing requirements of modern control systems, especially for the nonlinear systems. Furthermore, as data-based solutions expand their limited applications to more broad areas, opportunities and new considerations arise in the control theory [40].

The initial approach to include artificial intelligence was to substitute the analytical models in the prediction stages with DL-based architectures with the objective of improving the predictions quality, mainly in complex systems, where models were not readily available. As a result, several successful applications have been reported [63, 29, 34] offering DL-based predictions where analytical models have been substituted by data-based ones.

In [112], authors considered the computational load problem for fast systems using MPC. They suggested a neural network model that mimics MPC solutions. In other words, the time-prohibitive control calculations were simulated offline to generate training data for the neural network. Such an approach generated a NN that behaves as an MPC algorithm in fast systems. Although the training may generate a good enough controller, the central characteristic of MPC is the ability to forecast future events and unforeseen disturbances; For the method to properly ensure a robust execution, it would require an online adaptive feature, specifically in the Prediction model of the algorithm. Moreover, the optimization problem is no longer solved online, meaning unforeseen system states are not considered in future calculations.

The Inverse modeling approach as direct control has an extensive history [122, 123] because it has been widely regarded as an ideal controller. Still, research on the matter has continued along with advancements in neural networks. Recent applications, such as [4] trained a DL model capable of approximating the inverse dynamics of vibrational effects in a building structure to design a compensating mechanisms with an attenuation feature. Previously, fast dynamics and complex systems such as vibration effects were considered off-limits for IM application due

to the causality issues often found. However, using the approximation provided by the DL model achieved good results, and the authors applied it as a direct method. Nevertheless, the control signals produced by an IM have no long-term guarantees; For instance, in an unforeseen scenario[1], the DL model output is unpredictable.

Bounded methods have proven effective when employing artificial intelligence algorithms for control. For example, in the paper, [97] the use of an inverse approximation was combined with a predictive control for a rapid attitude maneuver system of an aircraft. The analytic derivation of that system's inverse model is a common practice in the area. Thus the contribution was the addition of an MPC setting, improving the overall results and control. For particular systems where an analytical derivation is possible, such as this case, the inverse model has been applied successfully.

Recently, [38] implemented an inverse MPC for an autonomous driving control system. Such a novel presentation was introduced to capture the human interaction to the vehicle, learn it, and propose it as an initial solution to the MPC algorithm. The benefits of having inverse dynamics data are notable, however, limited to particular applications. Additionally, in [100] the authors proposed learning the Inverse dynamics of systems for the implementation in MPC settings. This was one of the first works to suggest a similar approach to the best of our knowledge. The authors employed a radial basis neural network and analytically solved the optimization stage to use the inverse solutions. The numeric experiments presented were based on a linear version of the inverse pole on a cart system, with empiric-tuning of the control parameters. Although an excellent method for linear systems, the nonlinear cases remain an open-problem.

This dissertation details our approach to the general use of Inverse models in MPC settings based on DL architectures and their evolutionary optimization. We differ from previous proposals in the sense of directly learning the Inverse model, being linear or nonlinear, without further assumptions than the physical boundaries of the system. Moreover, we propose no linearization procedures nor other limitations or assumptions for the systems. We aim to provide a framework for the complete range of controllable dynamic systems.

[1] Common in the control of nonlinear systems, such as vibration control

1.3 Contents and Chapters descriptions

The present dissertation is focused on the analysis and control of nonlinear systems through Deep learning approaches, specifically with the use of Inverse Model approximations for optimal control solutions based on Predictive Control frameworks. Our initial procedure is the model and identification of nonlinear systems using Deep learning methods, focusing on the heuristic search of optimal parameters via evolutionary computation. Then we proceed to improve the predictions and optimization of conventional MPC settings by including data-based learned predictions and evolutionary optimization. As one of the ongoing concerns of model-based control and MPC, in general, is the computational load, we present the possibility and empirical approach to a feasible reduction via Inverse Models and the subsequent implementation on relatively fast systems.

Furthermore, we provide the necessary foundations and experimental assessment for the implementation of Deep learning-based predictive controllers. Finally, Inverse Modeling will be detailed and expanded to enhance the Predictive Control paradigm, the conventional control theory and ultimately close the gap between research and application of artificial intelligence mechanisms in control systems. Indeed, the main concern regarding intelligent control is the uncertainty that black-box methods present; Therefore, we propose a classical method enhanced with the power of an Inverse Model via Deep Learning as our main contribution.

The remaining chapters are structured and described as follows:

Chapter 2: We introduce and expand the Model Predictive Control methodologies set. We commence with a theoretical definition, including the algorithm description and implementation, describing the general stages, prediction and optimization, and their execution in real-time. In addition, we present a numerical simulation to show the performance and general behavior of the algorithm.

Chapter 3: The essential information regarding Deep Learning methods and their scope in the control theory context. We detail the system identification theory and the application framework to nonlinear systems. Moreover, the details for generating predictor models in MPC schemes are expanded with NARX modeling.

Chapter 4: The Inverse modeling of nonlinear systems. We present detailed analyzes regarding the Inverse approximation with means of control. The Inverse Model Predictive Control, using the control signals produced by a Deep learning-based Inverse model, we present a framework to reduce the computational load while improving the overall control system execution quality.

Chapter 5: We extensively use the heuristic search power of evolutionary optimization to generate optimal Inverse models in the control of dynamic systems context. Additionally, we provide the training framework for the optimal selection of parameters in the control of dynamic systems context.

Chapter 6: The Inverse Model Predictive Control will control nonlinear systems in numerical simulations. We develop an assessment evaluation methodology and present the workflow for automated testing to validate our claims. A detailed comparison with the proposed benchmarks is highlighted in this chapter. The evaluation metrics are displayed, and one on one comparisons with other methods, conventional and intelligent, are presented.

Chapter 7: Comparative analysis and results. We address the impact of our contribution to the control of nonlinear systems.

Chapter 2

Model Predictive Control of Nonlinear Systems

2.1 Introduction

The nonlinear control problem is a generalization of the algorithm design for the automatic regulation of dynamic systems. As most problems are considered and reconstructed as linear systems for an analytic approach, nonlinear methods deal with systems in their most accurate representation. Consequently, nonlinear controls are precise and better than their linear counterparts. However, since the design of linear controllers is a mature, robust, and proven toolset, the applications of nonlinear controllers are scarce. Nevertheless, the most popular method to solve the nonlinear control problem is the Model Predictive Control (MPC), a technique based on online optimization and forecasting.

2.2 Algorithm

The Model Predictive Control, commonly known as MPC, is an optimal, discrete control technique. Its development started as an alternative of single control loops for a simultaneous approach to multiple input/output systems. Moreover, since the formulation of the control problem became closely related to optimization techniques, MPC proposed an optimal online control. The consideration of optimization constraints within the online MPC generated increasing popularity for the method.

MPC is better understood as an idea rather than a direct method. It is a combination of model-based optimization and continuous execution through time also called receding horizon. As the system's observable variables evolve through time, the receding optimization changes its initial parameters to solve an optimal control problem online. The model-based optimization algorithm utilizes a Predictor model. Meaning, the Predictor model evaluates the proposed solutions given by the optimization algorithm based on an arbitrary cost function. To properly visualize the MPC extension, let us define the dynamic system.

Definition 2.2.1 (Dynamic System)**.** *Mathematical formalization of a descriptive rule for the time transition between an input and output variables of a physical system. The nonlinear relationship is defined such that:*

$$\dot{\boldsymbol{x}}_t = f(\boldsymbol{x}_t, \boldsymbol{u}_t, t), \quad \boldsymbol{y}_t = h(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad t \in \mathbb{R} \geq 0 \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^M$ is the state vector, $\boldsymbol{u} \in \mathbb{R}^N$ is the system's input vector, and $\boldsymbol{y}$ is the system observable output, all time-dependent. The system states or internal variables of size M and the input number of variables N are $M, N \neq 1$ for multiple MIMO systems. Assuming state and input vectors are bounded such that $\boldsymbol{x} \in \boldsymbol{X}$ and $\boldsymbol{u} \in \boldsymbol{U}$, both subspaces of $\mathbb{R}^M$ and $\mathbb{R}^N$ respectively, the nonlinear mapping function f describes the complete evolution of $\boldsymbol{x}$.

Here, the main goal of MPC is to minimize the difference between the state vector $\mathbf{x}$ and a reference value $\mathbf{r}$ by finding the optimal $\mathbf{u}$ as:

$$\arg\min_{u} e(t) = \mathbf{r}(t) - [f(\mathbf{x}(t), \mathbf{u}(t), t)], \quad t \in \mathbb{R} > 0, \tag{2.2}$$

for the duration of the system operation. The solution of (2.2) is calculated each time step t_i. Hence operations must be performed within a sample time.

The online optimization problem is solved considering the boundaries as follows:

$$\begin{aligned} \mathbf{u}_k &\in [\mathbf{u}_l, \mathbf{u}_h] \\ \mathbf{x}_k &\in [\mathbf{x}_l, \mathbf{x}_h] \\ \mathbf{r}_k &\in [\mathbf{r}_l, \mathbf{r}_h] \end{aligned} \tag{2.3}$$

where all variables subscripts u_l, x_l, r_l are the lower bounds and all u_h, x_h, r_h are the upper bounds for all k during the system execution. Generally, bounded optimization problems are difficult to solve, especially within nonlinear or nonconvex problems, system-dependent characteristics.

MPC has two main components: The model-based optimization algorithm and the Predictor model.

2.2.1 Optimization

The system input $\mathbf{u}$ manipulates the system. Therefore it is also known as a manipulable or control signal depending on the application area. In this work, we selected the term control signal. As the main objective of MPC is to minimize the error vector, as shown in (2.2), the optimization stage is the critical element. For the system execution at each time t, the algorithm minimizes the arbitrary cost function with the general form:

$$J = \sum_{i=0}^{T_p} w_{x_i}(r_i - x_i)^2 + \sum_{i=0}^{T_c} w_{u_i}(u_i - u_{i-1})^2, \tag{2.4}$$

where the cost J is simplified into a quadratic function with two cost sections, the system's state and the control signal. The weighting coefficients w_{x_i} and w_{u_i} are proportional to the importance given, i.e., are problem dependent. The cost function (2.4) contains two important MPC parameters: Prediction and Control windows, T_p and T_c respectively.

The algorithm requires information $i \in [i = 0, T_p]$ to solve the optimization problem (2.4), which in other words would imply knowledge of the future, assuming a starting point of zero. Instead of the actual $x_i \in [i = t, i = (t + T_p)]$, the predictor model provides the remaining values within a relatively small window $[t, t+T_p]$. The selection of T_p is important because it directly affects the algorithm performance.

The second section of (2.4) involves the control signal cost. Since the calculation only considers previous control signals, there is no need to predict solutions in the standard case. However, the selection of T_c directly affects the control quality, meaning the amount of the signal referred to the system. Both Prediction and Control windows have a crucial effect on MPC designs. Therefore they must be carefully selected, especially for systems with relatively fast dynamics.

The numerical optimization is performed iteratively by the algorithm. At each time step t, a set of future predictions $\hat{\mathbf{x}}_i \in [i_0, i+T_p]$ are produced by the Predictor model, which then complete the information required to solve (2.4). Therefore, the computational complexity results in:

$$\mathcal{O}(KT_p), \tag{2.5}$$

where K is the worst-case iteration number required by the algorithm times the prediction window T_p. To summarize, the solution to the optimal control problem

defined by (2.4) is:

$$u^0(x) = u^0(0;x), u^0(1;x), ..., u^0(T_p - 1;x); \tag{2.6}$$

For all the given states and windows. Finally, following the implicit MPC, the first sequence of the optimal $\mathbf{u}$ is transferred to the controlled system as:

$$U_N(x) = U^0(0,x), \tag{2.7}$$

and the process repeats for the duration of the controlled system execution.

2.2.2 Prediction

The optimization algorithm numerically minimizes (2.4) along with the prediction vector $\hat{\mathbf{x}}_i$ and the difference of previous control signals $u_i - u_{i-1}$. The prediction vector is assumed precise enough, such as:

$$\epsilon = \mathbf{x}_i - \hat{\mathbf{x}}_i, \tag{2.8}$$

where ϵ stands for the prediction error; however, in the practical case, it entirely depends on the precision of the constructed model. Most dynamic systems have a nonlinear nature with a relatively small range of operations where they behave linearly. Therefore, it is common to find the linear MPC implementation where a linear model generates the prediction vector. Predictions within the range of linearity are considered precise and robust. Moreover, they simplify the optimization problem and reduce it to an analytical problem with simple computational costs. Nevertheless, outside the linear interval, the linearized version of predictors does not ensure the necessary level of accuracy. As for nonlinear systems in general, data-based solutions or carefully crafted nonlinear expressions constitute Nonlinear MPC (NMPC) algorithms.

In this work, the primary consideration is using data-based methods for modeling or identifying dynamic systems—specifically, Deep Learning (DL) architectures. Since DL function approximations abilities can model dynamic systems, these are considered excellent nonlinear predictor models. Thereby, the prediction is defined by:

$$x_{i+1|k} = f(x_{i|k}, u_{i|k}, i + k, w_{i|k}, \epsilon), \tag{2.9}$$

where $k \in \mathbb{Z}$ is the discrete time variable, and $i|k$ is the i-th prediction given a time-step. The variable ϵ describes the parametric uncertainty or prediction quality as given by (2.8). While the true dynamics of a given system f can be fairly complex, a DL training scheme provides a powerful approximation, thus reducing ϵ, increasing the level of certainty of predictions (2.9).

2.2.3 Implementation

MPC is widely used in industry in its linear, implicit version, with a more significant presence in process control.

The practical approach is a well-understood class of control algorithms which the reader can consult in the technical review [88] for detailed implementation. As for the nonlinear MPC, its action area lies in complex systems, with multiple input and output settings, time-delays, and uncertainties. The review papers [2, 85, 130] summarize the overwhelming presence of MPC in-process controls, mainly due to the MIMO capabilities, optimally bounded control signals.

Attempts to reduce the computational load in MPC have been suggested since the initial formalization of the method. For instance, the works [116, 99] specifically deal with reducing iterations or complexity, aiming at the subsequent implementation to fast systems. However, as noted by the author, fast MPC remains to be an open problem.

With the re-emergence of artificial intelligence methods, the Control Theory's parallel development of new approaches using DL appeared. Although the use of neural networks for control is not a new idea [72], the ongoing research and practical attempts show the potential solutions to specific problems. Nevertheless, as shared in nonlinear systems, there is no general solution, even with DL approaches. As mentioned earlier in this chapter, although the concept of MPC is more of a set of control algorithms, there is a general step format for the nominal or explicit case. Such steps are stated as follows:

- Read $\mathbf{x}_i, \mathbf{u}_{i-1}$ from the system
- Determine the system conditions at time k
- Compute optimization of (2.4)
- Output optimal solution, $U_N(x) = U^0(0, x)$ to the system.

Nominal MPC is a well-studied method for the convex and time-invariant cases, i.e., linear dynamics. For the general case, the recursive feasibility property of MPC [40] states that if the optimization problem of (1.3) has a solution for the state vector $\mathbf{x}_k$, it is therefore feasible for all $\mathbf{x}_{k+1}$ and all future time steps.

Regarding stability, while there are carefully detailed analyses, the general approach is the Lyapunov arguments related to the optimization function (2.4) and

the inherent nature of the solutions and predictions. In other words, the guarantee of stability is given by the optimization decision and the recursive feasibility property of MPC, analytically ensured at each time step. Conversely, for data-based cases, numeric simulations and assessment evaluation methods give stability analysis. This work bases our implementations on entire data-based systems; large-scale simulations offer oversight for stability and feasibility requirements.

2.3 Control

Real-time execution is the first strict requirement of control systems since the observable variables must be regulated at each moment to avoid error and instability. Let us consider a dynamic system in the form of definition 2.2.1, where the control signal is denoted by u_t. The control algorithm's primary directive is to compute the appropriate u_t to drive the system state vector x_t to the desired value. Since MPC is a closed-loop setting, the feedback information collected from the observable variables is used for two central purposes – to establish the Prediction model's initial conditions and compute the error in the function of the actual state of the system.

The Predictor model –which is assumed to be as close as possible to the actual system – produces a future $\hat{x}_t$ with a length of the prediction window. The optimization algorithm generates a set of possible control signals, evaluated using the Predictor model in an embedded simulation. After several iterations depending on the optimization algorithm, the optimal control signal –which guarantees stability and local optimality in the finite interval of the prediction window – is driven to the physical system. Thus, the observable variables are collected through a sensor element and used as feedback information for the process to repeat.

Because the control objective depends on the particular requirements of the controlled system, the essential factor is to consider the desired state vector r_t as a reference point for all calculations. Moreover, if the control problem holds information of future desired values – i.e., r_t is known in advance – it may be used in the Prediction computation, such as:

$$\hat{x}_{t+w_p} = \varphi(x_t, r_t) \tag{2.10}$$

which would guarantee a more precise future output. In the other case, where the desired values are determined during the control system execution, for example, in wind power generation with variable demand and supply, the reference vector r_t is considered constant for the duration of the prediction window. Similarly, for

regulation tasks where the system states must remain within the desired value despite disturbances or the systems' nature, r_t remains constant.

2.4 Performance

As the system evolves in time, the optimal control signal is generated at each time step k. This results in a highly complex algorithm regarding execution time, where the minimization problem must be solved within t_s, the system sampling time. As mentioned before, the root of MPC popularity within process systems is the relatively slow nature of some scenarios where time constraints are negligible [1]. Conversely, for fast systems, time constraints are an ongoing issue.

To visualize MPC performance, let us consider the numeric simulation based on an example nonlinear system defined as:

$$x[k+1] = \frac{x[k]}{1+x[k]^2} + u[k]^3 \tag{2.11}$$

Which is a discrete representation of a generic nonlinear system. System (2.11) evolves in the discrete-time at $k \in [0, 1, 2..., N]$. The system variable x_t is considered fully observable, meaning the complete set of internal variables is known. The control objective is to design an algorithm that drives the system state vector x_k to the desired value of an arbitrary reference signal given by $r_k = 0.2 \sin k$. Thus, the MPC algorithm computes an optimal u_k at each instant k solving the following cost function, derived from the general form (2.4) and a constant prediction window W_p:

$$J = \arg\min u \sum_{k=0}^{W_p} (r[k] - x[k])^2 \tag{2.12}$$

The figure shows the multiple components involved in the entire system execution, highlighting the amount of predictions made for each control signal. Indeed, for a simple nonlinear system such as (2.11) the computations amount is relatively high.

[1] For instance, a reactor system time constant is on the order of hours

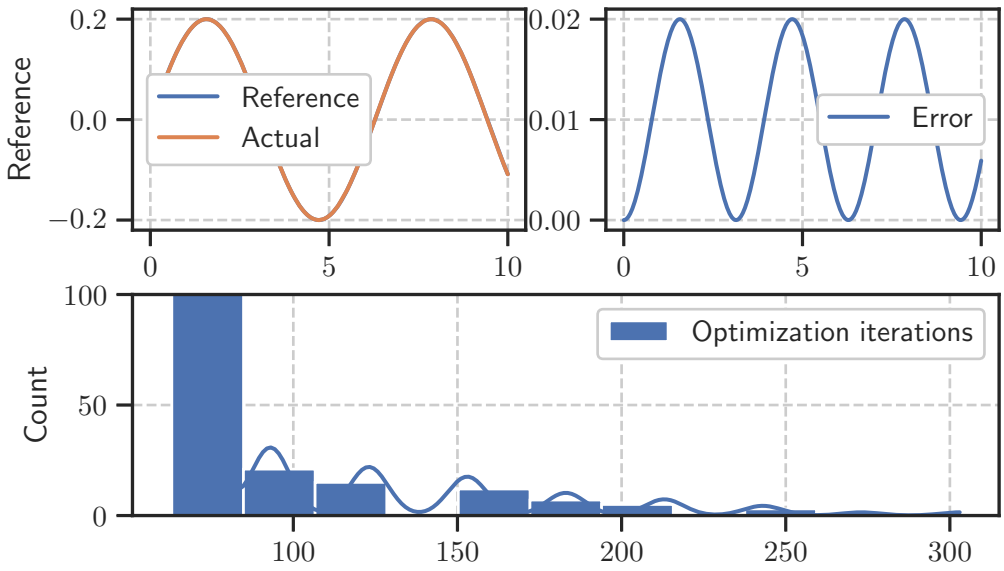


FIGURE 2.1: MPC Simulation. The first plot shows the actual and reference signals. The error plot shows the difference between the desired value and the real system output. Note the error is minimum. Below, the iterations count per time-step. Note the highest amount for the optimization algorithm takes place at the start of the simulation, wiht a steady decrease.

2.5 Summary

The MPC algorithm serves multiple purposes by producing optimal control signals in a multivariable setting. Because of the nonlinear nature of most real-life systems, the prediction models must accurately emulate the existing conditions to ensure an accurate MPC execution; however, as noted in the numeric simulation, performance rapidly becomes an issue where a large number of functions validations are performed.

Thus, beyond the optimization stage and prediction accuracy, the underlying situation of a computationally high cost restricts MPC to slow systems. Therefore, our aims go to the performance increase and the overall quality of the algorithm output. We propose a decrease in computational complexity and overall enhancement by adding some elements and including DL architectures as initial solutions and predictor models.

Chapter 3

Deep learning based Predictive Control

3.1 Introduction

Deep Learning-based controls refer to a group of nonlinear methods based on data that solve the control problem. DL architectures are excellent function approximators; therefore, the uses range from dynamics identification to learning control signals. Since dynamic systems are causal phenomena in which evolution happens through time, DL architectures must also have a dynamic execution. The training algorithms differ in this sense to other DL applications, where problems may be considered *static*. In this chapter, we will expand the notion of DL involvement in the intelligent control field. Specifically, learning dynamics and learning to control.

3.2 System Identification

System Identification (SI) is a compilation of tools and methods that construct dynamic system models from measured input-output data. SI has three major components:

1. Data
2. Model selection
3. Estimation method

This chapter will detail the concept and application of DL-based SI methods to construct nonlinear models from data for Prediction in MPC settings. Moreover, such methods are the foundation of the Inverse Modeling aspect of the thesis.

Identifying linear models using measured input-output data pairs is a well-studied field with an extensive history [60]. The first constructed models from linear phenomena became the basis of the conventional control designs and applications. After that, however, the SI methods moved towards a nonlinear identification philosophy because real-world problems are nonlinear and time-varying.

There are situations where linear models successfully capture the general aspects of problems; however, the nonlinearities cannot be ignored for other applications. For instance, in mechanical systems, the presence of multiple interconnected components generates energy transfers, being thermal or vibrational, both undesired effects with potentially damaging consequences. Also, in electrical systems, active components have different behaviors with the temperature rise, such as resistance and capacitance. Moreover, in power generation, active regions of energy production change with time, depending on the source. Many of these aspects are essential in the design of models and controls, and linear models are imprecise or insufficient to provide the vast amount of information nonlinearities contain.

Beyond the dynamic systems and control, nonlinear models serve other areas of science, such as simulations. Again, the level of detail overcomes the simplicity and relatively low difficulty of the linear model SI. These are the primary motivations for nonlinear identification.

The elements of the nonlinear SI toolbox go beyond the scope of this work. However, there is extensive literature with technical approaches and general implementations the reader can consult [61, 82, 35]. We focus on the use of DL methods for the nonlinear SI of dynamic systems.

As mentioned at the beginning of this chapter, the SI consists of three main parts, data, model selection, and estimation method. The amount and quality of data is the most critical element. In dynamic systems, the data collection is a non-trivial task since the underlying information lies in the magnitude of the values and their temporal relationship. Therefore, the methods for data collection are many and are still an ongoing research.

The model building from data involves specific requirements in the dynamic system context. For example, from the dynamic system of Equation (1.1), where the nonlinear function f and input vector $\mathbf{u}_k$ map the state vector $\mathbf{x}_k$ transition, general response characteristics are measured and quantified. Figure 3.1 shows the

general aspects of the dynamic response to a test signal:

$$\mathbf{u}_k = \sum_{k=0}^{n} \alpha_i \chi_{A_k} k, \tag{3.1}$$

where χ_{A_k} is 1 for all $k \geq 0$, and 0 for all $k \leq 0$, also known as unity step [77].

We can observe from the test input unity step that several points are relevant in the transitory response of a dynamic system. These characteristics are applicable in the SI process; therefore, they must be present in data. In linear systems, the response features to a test input are the same for all $k \geq 0$ due to the superposition theorem. However, the amount of necessary data dramatically increases in the nonlinear case, where the response features differ depending on multiple internal or external factors. Figure 3.2 shows a comparison between intervals of testing inputs to linear and nonlinear systems.

The main characteristics that must be present in the training data are the transitory times and magnitudes. Essentially, the data collection must ensure that the domain of interest brings out all relevant values of interest [93].

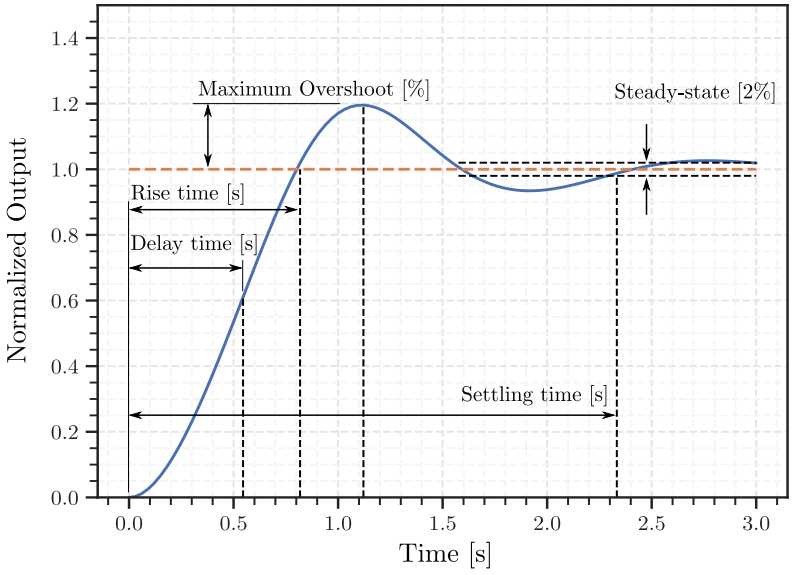


FIGURE 3.1: Dynamic System's response to the test input Eq. (2.1). The parameters shown are common landmarks in the analysis and design of automatic controllers. These points are also relevant to the identification through data-based models.

In the remaining chapter, we will introduce specific methodologies for the data collection in the control of dynamic systems context, with the specifics and parameter selection that will cover the necessary information for proper training.

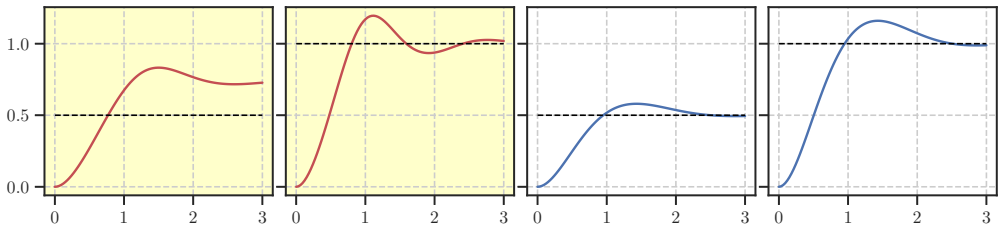


FIGURE 3.2: Nonlinear (Left) versus Linear (Right) systems with two test inputs: 0.5 and 1.0. Note that in nonlinear systems, the superposition theorem is not respected, i.e., a change in the input magnitude is not proportional. On the contrary, linear systems' magnitude change is proportional

3.3 Nonlinear Auto Regressive with Exogenous inputs (NARX)

As the second of the three elements in SI (data, model selection, estimation methods), the model selection is a trivial step for linear systems. In linear cases, SI takes an approach to physical characteristics of the system, such as an energy balance equation or input-output analytic expression.

While for the nonlinear systems, the general case must estimate a future value x_{k+1} from the currently available information. Thus, past input-output values become the data source as we lack information on the full range of possible dynamics. Therefore, we define the selected model that considers previous information as input as:

$$\varphi_t = [x_{t-1}, x_{t-2}, ..., x_{t-n_a}, u_{t-1}, u_{t-2}, ..., u_{t-n_b}]^T, \tag{3.2}$$

$$\hat{x}_i = h(\varphi_t), \tag{3.3}$$

where φ_t is the past input-output data vector, and the nonlinear function h describes the system's dynamics. The lagged terms, $x_t \in \mathbb{R}^X$ and $u_t \in \mathbb{R}^U$ are the system's observable states and its input vectors. Lastly, constants n_a and n_b are delay parameters arbitrarily selected (Particular dynamics influence the selection of these values, discussed further in this chapter). The model (3.3) is known as NARX (Nonlinear Auto Regressive with Exogenous inputs).

NARX models acquired the concept from signal processing and FIR models (Finite Response Impulse); Where an impulse is used to characterize a full range of dynamics in a frequency domain dynamic system. NARX models with $n_a = 0$ are nonlinear FIR models [31].

There are several ways to parametrize the nonlinear function h, but all involve solving an optimization problem. This point made NARX models a popular method for SI since it requires no iterative optimization procedure but the definition of h only. To formalize the objective of the SI, the cost function:

$$\hat{\theta}_N = \arg\min_{\theta} \sum_{t=1}^{N} \|x_t - \hat{x}_t|\theta\|^2, \tag{3.4}$$

where θ is the model that best describes the input-output relationship from data N, therefore, the learned model $\hat{\theta}_N$ serves as an accurate representation of a dynamic system, assuming the quality and representation of the training data.

3.3.1 Polynomial NARX

The construction of θ_N NARX models involves the optimization of the cost function (2.4). The conventional approach of NARX models is an optimal selection of polynomial constants, considering (3.2) and defined as:

$$\hat{x}_k = \Theta_0 + \sum_{i=1}^{n} \Theta_i \varphi_t, \tag{3.5}$$

where Θ_i are learnable parameters that compose a linear combination, along with φ_t (lagged values from 3.2). Although the selection of Θ_i is a linear problem, the presence of the lagged components formulates (3.5) as a nonlinear approximation framework.

The solution of (3.5) involves the selection of: $n_a \geq 0$, $n_b \geq 0$ and $n \geq 1$ through an optimization algorithm, minimizing cost function (2.4). The amount and characteristics of the training data are problem dependent. The figure 3.3 summarizes the steps for the design of a polynomial NARX model.

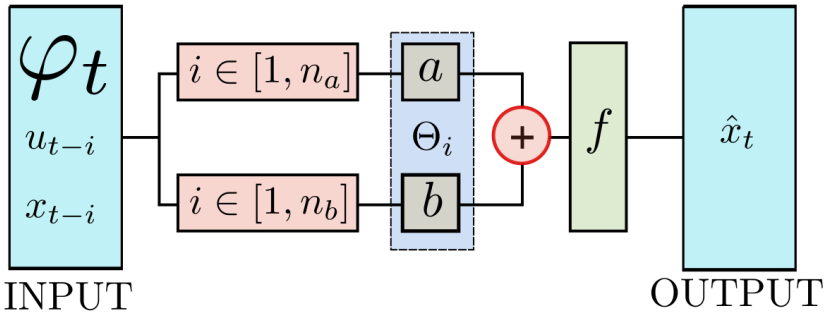


FIGURE 3.3: Polynomial NARX. The model is a linear combination of the regressor vector φ_t and constant vector Θ_i. The activation function f is nonlinear.

3.3.2 Deep leaning NARX

Although selecting the constant parameters in the polynomial version of the NARX model is relatively simple, the problem becomes more complex for multiple input-output systems. Moreover, the degree of potential numerical errors due to process noise and the nonlinearities included in the training data may affect the final accuracy and certainty of the model.

Another approach to model NARX expressions is to approximate the nonlinear element h with a DL architecture. The process is equivalent to a supervised training method, where the set of lagged information φ_t becomes *features* and the known $\mathbf{x}_t$ vectors become the *labeled* data.

Learning from data can be reformulated to an approximation problem, such that:

$$y(t) = F(x_t) + \epsilon, \quad t = 1, 2, ..., N \tag{3.6}$$

where $y(t)$ is the estimation, F is the function to infer, x_t is the training data, and ϵ is the process noise. Thus, learning can be summarized as adjusting F with the difference $y(t) - x_t$ as the training evolves.

Considering the NARX model (3.3) and constructing the nonlinear mapping as to a Feedforward Neural Network (FNN):

$$\hat{\mathbf{x}}_t = \Psi(\varphi_t), \tag{3.7}$$

where the prediction $\hat{\mathbf{x}}_t$ is estimated by the NN Ψ using the lagged information vector φ_t.

From the lagged vector φ_t components, the input vector $\mathbf{u}_k$ applied to the NN produces a hidden layer response as:

$$H_i(t) = f_h[\sum_{n_1}^{n_a} w_{in_1} x_{t-n_1} + \sum_{n_2}^{n_b} w_{in_2} u_{t-n_2} + b_i], \tag{3.8}$$

where f_h is the layer activation function, w_{in_2} is the connection weight between the input neuron and the i-th hidden neuron. Similarly, w_{in_1} weights the output neuron to its i-th hidden neuron.

Subsequently, the NN output layer is defined as:

$$\hat{\mathbf{x}}_j = f_o[\sum_{i=1}^{n_h} w_{ji} H_i + b_j], \tag{3.9}$$

where the output layer activation function is f_o, w_{ji} is the weighting value, n_h is the number of hidden neurons and H_i is the i-th hidden neuron output.

Equations (3.7) to (3.9) formalize the NARX model using a simple NN, the FNN. The training algorithms require to optimize the weighting values to the difference between the *actual* state values $\mathbf{x}_k$ and those predicted by the output layer (3.9). In addition, the activation functions must provide a nonlinearity for adequately estimating the nonlinear elements contained in the training data. An experimental evaluation showed that the best activation function to model nonlinear dynamic systems is the hyperbolic tangent, defined as:

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{3.10}$$

where e is the euler's number. Different activation functions can also be employed, however, our experimental framework showed this to be the most effective. More

about the training and optimal NARX will be discussed in Chapter 5.

Despite FNN's overwhelming capacity for learning dynamic systems, the most difficult ones benefit from more advanced DL architectures. To summarize the DL-NARX, the figure shows a simplified version of the mentioned equations.

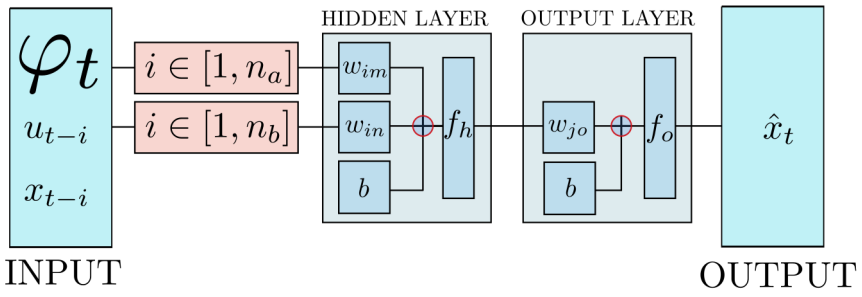


FIGURE 3.4: Deep Learning-based NARX simulation model, based on Equations (3.7) to (3.9).

3.4 Architectures

Feedforward NN generates accurate mappings from the regression vector φ_t to a single step in the future, $\mathbf{x}_{k+1}$. Using the feedback interconnections, the NN works as a recurrent neural network. However, feedback connections may present instability issues as the vector φ_t is fed back to the input layer and the process noise element.

An additional issue is that older data is replaced in training as new regressive information enters the feedforward network.

Remark 3.4.1. *All of our DL-NARX are simulation models. Simulation models act in real-time, meaning that new information enters the input layer as it becomes available. For instance,* $\varphi_{t=0} = [x_{t-1}, x_{t-2}, ..., u_{t-1}, u_{t-2}]$ *updates at* $t = 1$ *as:* $\varphi_{t=1} = [x_{t-2}, x_{t-3}, ..., u_{t-2}, u_{t-3}]$. *Note that the regression vector changes one step at a time, in a real-time scenario. On the opposite kind, forecasting models compute their predictions with complete sequences of time, in a sequence-sequence fashion.*

3.4.1 LSTM

The new information supplied to the *recurrent* NN formed by the DL-NARX model may create an effect of memory decay. The solution proposed to solve this phenomenon is called Long-Short Term Memory (LSTM) [41]. The problem of vanishing gradients is resolved by a set of activation *gates* that control the learning memory and pace of the internal sequences. It has two main gates, *cell gate* and *hidden gate*, for long-term and short-term memories management, respectively.

The gate composition is called *cell*, and deep LSTM networks possess a fixed number of stacked layers. LSTM outputs are often linear layers, similarly to the feedforward architecture. To formalize the LSTM cell, we name its cell components, such that:

$$i_t = \sigma_g(w_i x_t + U_i h_{t-1} + b_i), \tag{3.11}$$
$$o_t = \sigma_g(w_o x_t + U_o h_{t-1} + b_o), \tag{3.12}$$
$$f_t = \sigma_g(w_f x_t + U_f h_{t-1} + b_f), \tag{3.13}$$
$$\tilde{c}_t = \sigma_c(w_c x_t + U_c h_{t-1} + b_c), \tag{3.14}$$

where $x_t \in \mathbb{R}^d$ is the cell's input vector, f_t, is the *forget gate*, c_t is the cell activation vector and i_t, o_t are the input and output vectors, respectively. The trainable weights, $w_x, \in i, o, f$ are constant parameters affected only during training phase.

The memory control gates have activation functions σ_g performing the *tanh* (3.10) and *sigmoid* as:

$$\sigma_b = \frac{1}{1 + e^{-x}}, \tag{3.15}$$

where e is the euler's number. The activation functions arguments perform element-wise operations. LSTM technical guidelines and implementations can be consulted in [96, 36, 15].

LSTM architectures consider the memory elements of the training data; consequently, successful time-series forecasting models have recently appeared in research papers [117, 118, 79]. Furthermore, the use of LSTM in SI tasks has also increased since vanishing gradients problems are common in long sequences. Furthermore, for dynamic systems, changes after long sequences are common since nonlinearities and other uncertainties can appear at any moment of the execution. Moreover, LSTM is excellent DL architecture for producing dynamic systems simulation models, specifically NARX methods.

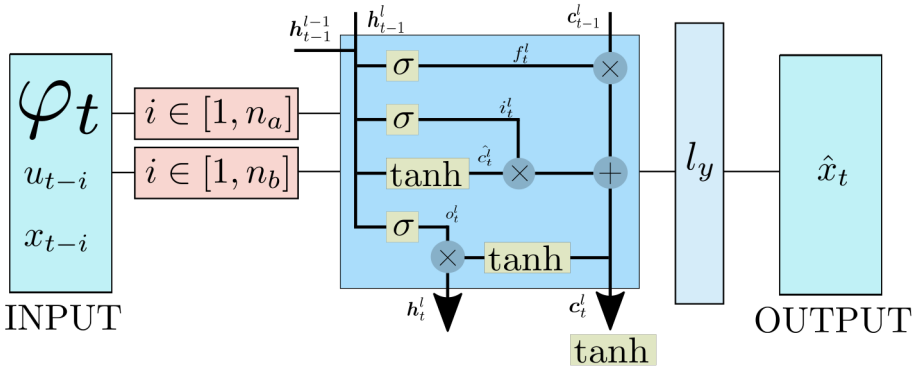


FIGURE 3.5: LSTM-based NARX simulation model. The center block represents the LSTM cell. An LSTM NN is constituted by a number of cells. The linear layer l_y serves as output.

3.4.2 CNN

Convolutional Neural Networks are a famous DL architecture with feature extraction capabilities, popular in image recognition tasks, video sequencing, and application where structural patterns lie hidden in data. The application field is vast, ranging from NLP problems, speech recognition and computer vision.

CNN excels in parameter sharing, sparse interactions, and equivalent representations. Moreover, CNN can process information in a multi-dimensional setting, for example, 2D of image data. This characteristic made it suitable for the mentioned tasks. However, in time-series forecasting and SI, it has been utilized as a classifier instead. Besides robust pattern recognition and classification skills, CNN structures can also be reformed into NARX models.

The way CNN operates differs from the FFN and LSTM in some particular ways. First, the execution follows several *convolutional* layers, along with subsampling methods, intending to extract and analyze sectors of the data. Subsequently, the patterns are filtered through several layers and, finally, a linear output. Input data has the multidimensional shape m, m, r where m is the height and width of the input vector, and r is the depth for the *channels* value. We formalize the CNN as the following:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n), \tag{3.16}$$

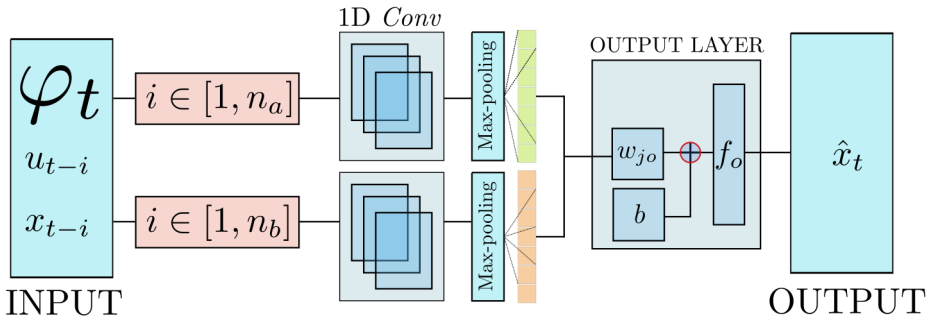


FIGURE 3.6: CNN-based NARX simulation model. Note the 1D-Convolution operations extract relevant features from the regression vector. The output is a linear layer.

where $*$ represents the convolution operation and m, n are multidimensional data. In other words, the convolution operation in a CNN is commutative, meaning that several operations can be stacked. To summarize the CNN development, the Figure 3.6 displays a NARX setting using a deep CNN.

3.4.3 Autoencoders

An autoencoder NN is an advanced DL method that combines two stages called encoder and decoder. Initially, decoder networks attempt to map the encoder's result as an output. This operation performs an internal modification and extraction similar to the convolution operation but with improved outcomes for dynamic data. Next, the primary encoder network generates a conversion of the input data, also known as *code*. Finally, the subsequent decoder layer attempts to reconstruct the code.

In this setting, the counter-intuitively learning manner is to expect a difference in the decoder interpretation of data. Instead, it is expected a certain degree of approximation, forcing the decoder network to optimize and *focus* on relevant patterns.

The relevant part of autoencoders, particularly from the dynamic system's analysis, is the latent space created in the pre-decoder zone. Such hidden information is called *latent space* and is used, among other applications, to learn apparent dynamics from data. The use of autoencoder frameworks in identifying dynamic

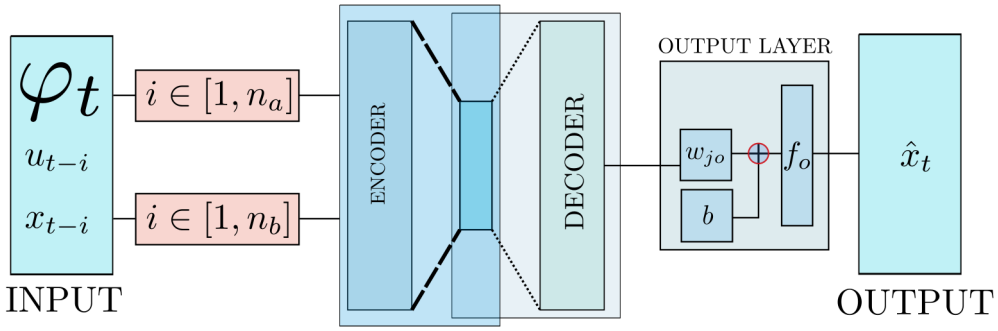


FIGURE 3.7: Autoencoder-based NARX Simulation model. The encoder-decoder networks could be LSTM, CNN or FFN.

systems has resulted in several new ways of SI. For instance, the works [80, 66] generate state-space models from the latent space generated by the encoder network.

In the DL field, learning the most relevant information from data using encoder-decoder setups is applicable using the under-complete form. Moreover, prioritization is greatly improved by shaping the encoder with a larger number of parameters and the decoder network with fewer. We consider this fact in optimizing this architecture and take it as a natural constrain for the training algorithm.

The Figure 3.7 shows a generic setting of autoencoder configurations. An encoder network is selected with more parameters to decode, i.e., with a more significant number of outputs. In the same manner, the decoder reshapes those inputs into the expected outcome of the network. Additionally, the autoencoder architecture can also be formed as a NARX simulation model.

3.4.4 Attention Mechanism

The attention mechanism was introduced to autoencoder frameworks as an attempt to improve machine translation tasks. The objective is to allow the decoder network to select the most relevant features flexibly with a weight layer [8].

Different lengths in the input sequences affect the performance of the decoder network; The attention layer allows a flexible solution to the particular cases, where such differences forced outputs in the networks.

The attention mechanism follows a set of steps:

Alignment scores: This step quantifies the level of alignment between input sequences and the current position. It is defined by:

$$e_{t,i} = a(s_{t-1}, h_i), \tag{3.17}$$

where $e_{t,i}$ is the alignment, a is a nonlinear function approximated by a FFN, s_{t-1} is the previous decoder output and h_i is the hidden states, product of the encoder.

Weights calculation: The weight level requires to be $\mathbb{R}^K \rightarrow [0,1]^K$, which is achieved by:

$$\alpha_{t,i} = \sigma_z(e_{t,i}), \tag{3.18}$$

where $\alpha_{t,i}$ is the weights vector and σ_z is the *softmax* activation function such that:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{3.19}$$

that produces a probability distribution $\in [0,1]$ given the alignment scores vector, $e_{t,i}$ as z.

Context vector: The context generalizes the priority of the decoder network into a set of weighted encoder hidden states as:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i, \tag{3.20}$$

thus finalizing a forward encode-decoder pass with attention.

The general attention case makes use of machine-translation concepts due to the initial proposal of the method [111]. The concepts are namely: *queries* q, *keys* K and *values* V. The queries vector q is processed in an element-wise product with the keys K vector, as:

$$e_{q,K_i} = q_i \cdot K_i, \tag{3.21}$$

and the scores e_{q,K_i} are nominalized by a *softmax* function (3.19) to generate the weighting coefficients, α_{q,K_i}.

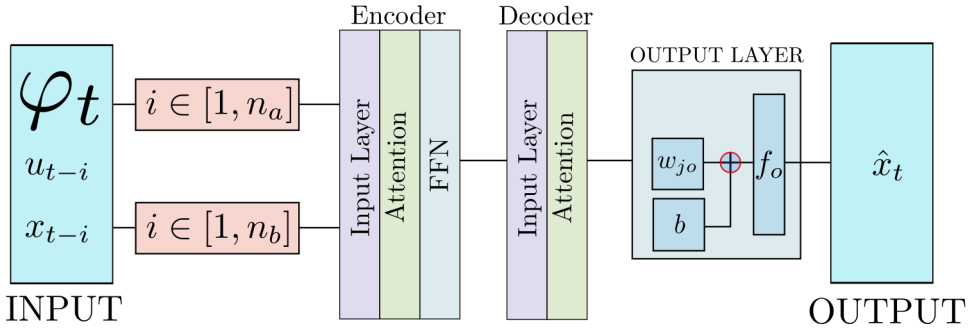


FIGURE 3.8: Attention mechanism-based NARX simulation model.

Finally, the general attention mechanism is represented such that:

$$att(q, K, V) = \sum_{i=0} \alpha_{q,K_i} V_{k_i}. \tag{3.22}$$

In the context of dynamic systems, we define the q, K, and V vectors as components of a time-series sequence, where information follows a similar pattern with words in machine translation, being the main difference the comparison against a *database* of meanings. Attention mechanisms have performed well for NLP tasks, and in dynamic systems as well [86, 39, 65]. Figure 3.8 summarizes the attention mechanism as a NARX simulation model.

3.5 Performance

Designing a NARX model involves data collection that covers the relevant system dynamics, i.e., information such as the stabilization or rise times to recreate a supervised learning problem. The mentioned methodology is exemplified with a numeric simulation involving the nonlinear system (2.11) shown in Chapter 2.

First, let us consider system (2.11) in its discrete form, excited by a uniformly distributed random signal $u[k]$. The produced output $x[k]$ corresponds to the target data in our supervised learning framework, such that $\Psi_t = \varphi(x, y)$ is the DNN emulating the system after training.

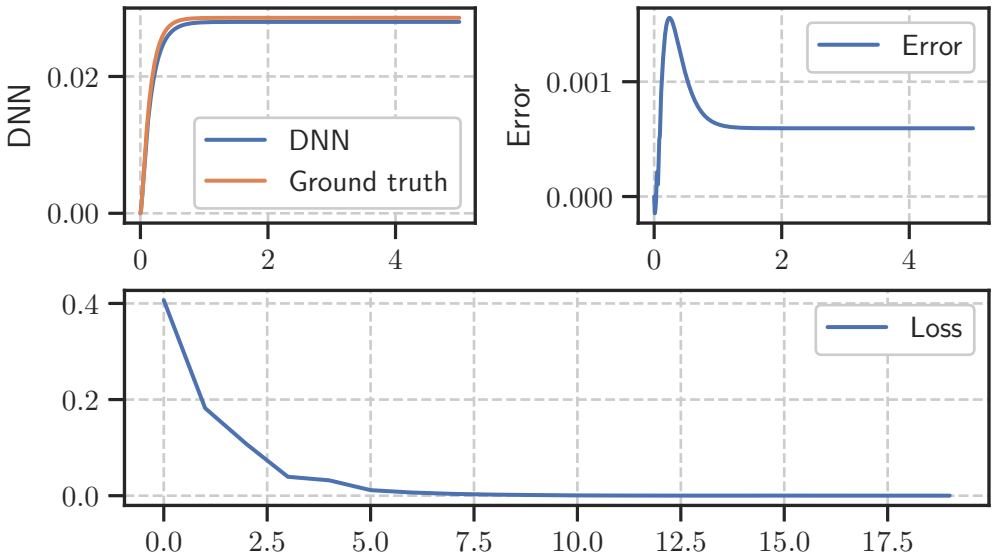


FIGURE 3.9: Training summary for the NARX model of system (2.11). The system response to a step signal is shown in the first plot. The error between the ground truth and the model is 0.0005 on average. This model is used by the DMPC to generate future predictions. Below, the loss over batch training (Epochs).

The NARX model Ψ_t design and training follows Algorithm 1. Thus the training data pairs are formed as the regressor vector φ_t with empirically selected delay values $n_a = n_b = 2$. Therefore, the regressor vector becomes $\varphi_t = [x[t-1], x[t-2], u[t-1], u[t-2]]$.

The numeric simulation generates $n = 1000$ samples where $u[k] \in [-2, 2]$ producing n pairs (u, x) defined as *features* and *targets* in the machine learning context. Performing Algorithm 1 results in a DNN model that closely behaves as the study case, capturing most nonlinear components from data as well as the general dynamics. The training is summarized in Figure 3.9.

3.6 Learning to Control

Thus far, our analysis of dynamic systems has been reduced to the conventional, negative feedback case. The assumptions for this kind of control system are a complete knowledge of the general dynamics. Although a data-based SI framework adds detail to the system's model, for the cases where data is not readily available or the dynamics are entirely unknown, model-based systems do not func-

tion with the expected correctness. Moreover, disturbance quantities may diminish the accurate knowledge of the dynamic behavior of a control system.

Conventional control algorithms are designed offline with a heavy focus on minimizing an arbitrary function, most commonly *error*. However, different conditions arise as the controlled system's state variables transition through time, not considered in the design stage. Such time-varying conditions are the foundation of *adaptive* controllers, which modify their constant parameters according to as new information becomes available. This subfield of variable controllers is known as *adaptive control*.

Because of the focus on minimizing error functions during the system's execution, the control problem can be rephrased into an optimization problem. The collection of analysis and tools for the design of optimal controllers is known as *optimal control*.

For the time-varying, nonlinear cases, the *adaptive* and *optimal* schemes have proposed interesting solutions, with complete stability guarantees. However, the several assumptions made, such as the overall conditions along with the complete dominion of the system, continuity, and general stability [5, 24, 10] are questionable in real-life systems.

These points summarize the contemporary control problem, as methodologies for model-based problems are powerful enough; the remaining issues lie in the modeling and approximations. Nevertheless, beyond that, the foundation assumes *unvariant* systems which data-based models, such as DL-NARX, can approximate better. Still, in online settings, disturbances and other unforeseen events may change the entire theatre, for which *online*, *adaptive* and *optimal* solutions are required.

3.6.1 Reinforcement learning

Reinforcement Learning (RL) is a machine learning of online control and adaptation by exploring an environment or system. The method can also be called an *action-based* learning framework because through small excitations obtains feedback. The RL methods are heavily inspired by training methods conducted in biology and behavioral studies [70].

RL basic algorithm performs a small control signal to excite an environment. In the machine-learning context, the term environment is used conversely to the

control theory, controlled system. Thus, when an excitation signal enters the system, the produced output is used as feedback information by the RL agent. Here, the tern *agent* corresponds to the term *controller* in the control theory context. In this fashion, the RL controller is similar to the conventional closed-loop setting. However, the main difference lies in the *actor-critic* formula proposed in the field [53, 12].

The actor-critic algorithms are methods that generate solutions and evaluations in a parallel manner. The actor component serves as *action* and the critic component as an *assessment*. These terms are closely related to the Predictor model and controller in the MPC methodology discussed earlier in this chapter. The objective is to adapt an appropriate actor-controller based on the observations performed by the Predictor-critic.

Nevertheless, similarities between MPC and RL methods mainly are in the closed-loop setting only. Moreover, the methodologies to generate the control and assessment components vary. As MPC seeks to generate bounded optimal control signals at each step, RL aims to learn the complete range of signals for every situation. We show in this work a detailed comparison between methods, assessment, and suggested application zones.

Therefore, in this work, we consider RL methods purely for comparison purposes. Specifically, the Deep Deterministic Policy Gradient (DDPG) for the actor-critic set up as an online controller for the nonlinear systems. Finally, to summarize our introduction to RL and its similarities with MPC, Figure 3.10 shows a diagram with the actor-critic reward system as a closed-loop feedback controller.

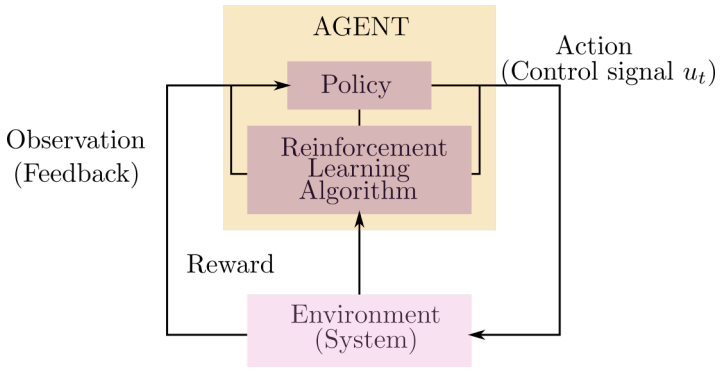


FIGURE 3.10: Reinforcement learning general setup. The agent works as a control element, tuned by training using a reward system. The control signals are expected to follow an optimal sequence.

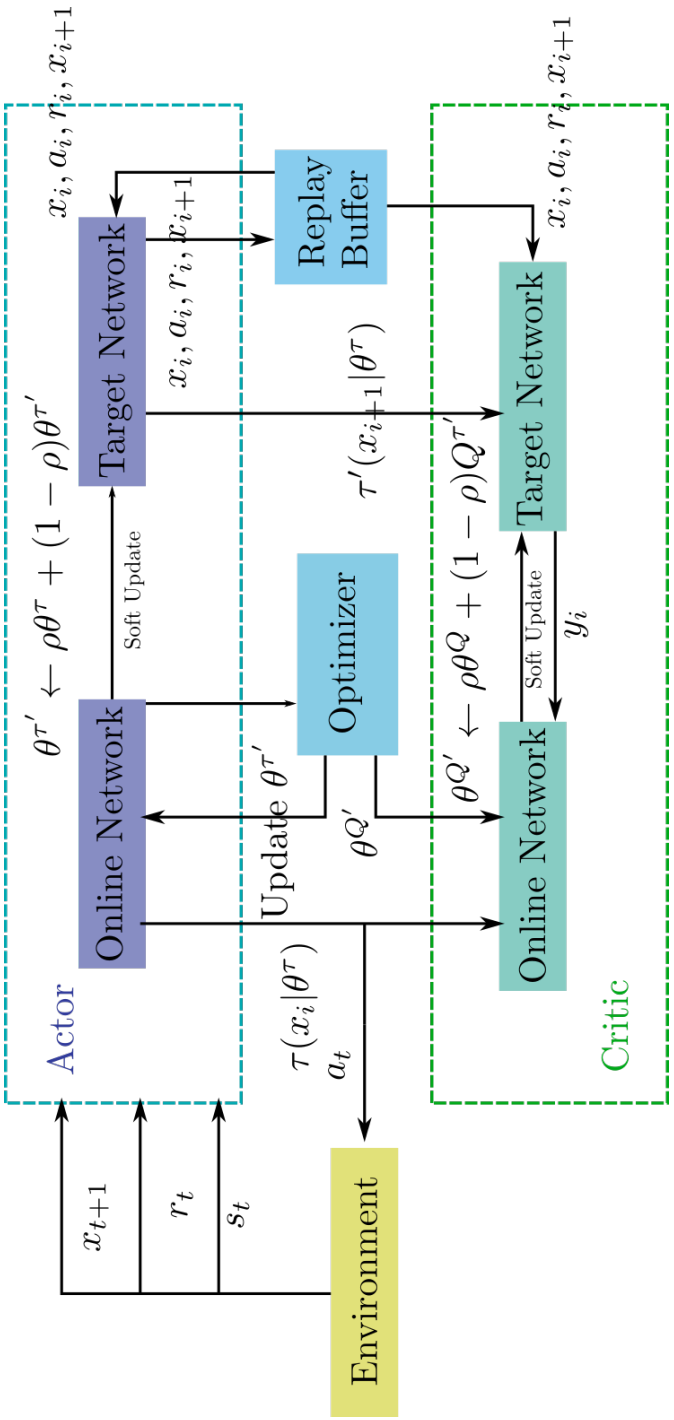


FIGURE 3.11: Deep Deterministic Policy Gradient Algorithm Overview.

3.7 Summary

By adding a DNN as a predictor, MPC improves its overall response quality at the expense of an increase in the system complexity. Additionally, we observed a slight reduction over a conventional model since the GPU performs such computations. Nevertheless, this slight improvement is neglected due to its hardware dependency and small magnitude. Moreover, design work has been reduced because the model is produced from data instead of a detailed analytical approach. This may be considered beneficial from the accuracy and production point of view, whereas less knowledge of the system is held from the control practitioner's stance.

Data-based learning mechanisms ensure accurate function approximation solutions, therefore well-suited for the design of control algorithms. However, demanding requirements may be too strict for a learning mechanism to ensure, as a degree of error and uncertainty is always present. For such a specific reason, learning mechanisms are not widely used in practical control designs beyond research and experimental cases. For instance, the RL algorithms require access to a random exploration that has proven highly successful in simulated environments. Conversely, results are directly proportional to the simulated environment accuracy for the practical implementation generating the mentioned uncertainty and lacking reliability.

Since data-based methods provide more remarkable accuracy by considering multiple patterns within data, its incorporation into the control field cannot be neglected. Therefore, we propose a parallel action by combining the MPC paradigm with a data-based learning mechanism with the accuracy and performance increase as a primary goal. Furthermore, combining such tools allows taking advantage of the multiple features available without losing the conventional control theory bounds, generating optimal control solutions without the uncertainty of a DNN model.

Chapter 4

Deep Inverse Predictive Control

4.1 Introduction

From the use of DNN as Predictor models, we observe an improvement in the quality of the forecasted values, making the optimization stage of the MPC algorithm a more reliable component. Furthermore, the available data permits an increase in the accuracy for future predictions, despite the nonlinear behavior shown by most dynamic systems. Nevertheless, the computation complexity remains an issue, where optimal signals are calculated within a sampling time.

This chapter details our main control algorithm and contribution, the Inverse Model-based auxiliary computation method for the MPC scheme, the Deep Inverse Model Predictive Control (DIMPC). An evolutionary DNN produces the IM approximation used as a starting point for the MPC algorithm for the real-time control of multiple input-output systems with nonlinear characteristics. The optimal control signals are computed by an optimization stage, tested with a DNN Predictor model. A DNN IM provides the initial solutions for the optimization, reducing convergence time and improving the quality of the responses.

We commence with the definition of Inverse modeling, approximation, and subsequential use as a control method. Then, we detail a learning framework based on DL and training data for the derivation and validation of Inverse models. Finally, we compose the complete algorithm, showing the initialization step and methods with the Predictive Control scheme.

4.2 Inverse Dynamics

The dynamic systems representations provide helpful, general information about the evolution of internal state variables through time. Still, dynamic models are only approximations nonetheless. However, an exciting and promising characteristic found in these realizations was the Inverse Model (IM).

IM are opposite functions to the nonlinear mapping of a dynamic system. Consider the dynamic system of (2.2.1), rewritten here for simplicity to the reader, $\dot{x}_t = f(x_t, u_t, t)$. Internal state variables x_t and input variables u_t are driven by the nonlinear function f that we assume fully describes the system's dynamics. The IM is an approximation of f^{-1}. Figure 4.1 shows a block diagram of an IM placed as a direct controller.

From Figure 4.1 we can derive the error signal as the difference between our reference-desired value r_t and the actual system output y_t defined as $y_t = Cx_t$ where C is a constant matrix. The error function is:

$$e_t = r_t(1 - f(x_t, u_t, t)f'(x_t, u_t, t)) - H_t\epsilon_t, \tag{4.1}$$

where H_t is an unknown time-dependent, external disturbance signal, and ϵ_t is the measurement, normally distributed noise. Then, assuming in (4.1) that the IM is ideal, the resulting e_t is a function of the unknown disturbance H_t and the measurement noise, such that:

$$e_t = H_t\epsilon_t \tag{4.2}$$

leaving the controlled system in closed loop affected only by external factors, excluding the internal dynamics.

Assumption 4.2.1 (Inverse existence)**.** *Nonlinear function f is not necessarily stable. However, the inverse function f^{-1} is assumed to exist, to be causal, and not necessarily stable.*

For a nonlinear system, such as 2.2.1, the IM identification problem is reformulated as:

$$\hat{\Upsilon}_\theta = \arg\min_\theta \|r_t - x_t|\theta_t\|^2, \tag{4.3}$$

where θ_t is the model that best defines the IM using the training data given by the regressor vector φ_t and the reference value r_t.

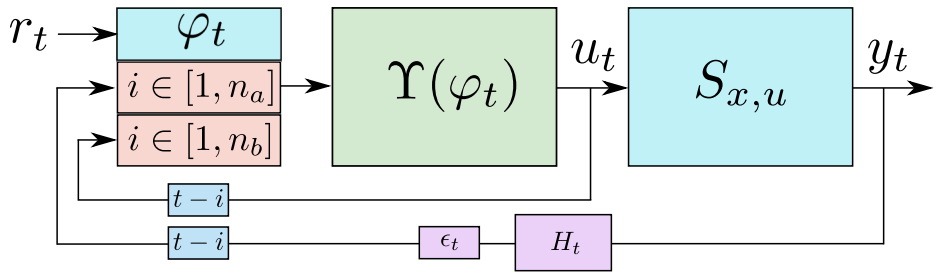


FIGURE 4.1: Inverse Model Υ_t used as Direct Inverse Control. The control signal u_t is computed using the regression vector φ_t.

A parametrization of the system is needed to identify the IM fully, meaning a dynamic model is required. To put it in another way, a SI procedure must be performed beforehand to create a simulation model from data. Therefore, the identification for IM can be divided into two generalized approaches.

- A model $\hat{f}(x, u, t)$ is characterized from input-output data or analytical methods, to be inverted analytically to generate $\hat{f}^{-1}$ [21, 108, 109]

- An IM approximation is computed directly from data [48, 14].

The first method involves an inherent error in design. In order to analytically invert a model approximation $\hat{f}$, the selected method must be a linear, simpler model. In other words, in this manner, we trade simplicity over accuracy. While this may work for simpler, linear systems, it may not suit more complex problems, as stated in Chapter 2.

The second method aligns with data-based SI. Assuming training data pairs, that is u_t, x_t, measured from the controlled system, the nonlinear IM is approximated as follows:

$$\Upsilon_t = \Psi(\varphi_t), \tag{4.4}$$

where Υ_t is the IM of (1.1), Ψ is DL-NN, and φ_t is the regression vector used in NARX models. This work is based on the data-based approximation of Υ through the second method.

4.3 Direct Inverse Control

The realization of IM has the potential to become a closed-loop feedback controller on its own. However, in order to design IM controllers, special considerations have to be analyzed beforehand: The IM of nonminimum phase dynamic systems is considered noncausal. Therefore its direct implementation is almost unexistent.

Since IM outputs and internal variables may contain high gain elements, the use of direct controllers may lead to instability due to the inversion step. Aside from the previous statements, which are only formally applicable to analytical derivations of the IM, the approximation computed by a NN is expected to perform under strict boundaries, included in the design stage. Thus, the IM controller based on NN is a causal and stable component under limited constraints.

To formalize the previous statements consider the nonlinear dynamic system to be controlled and its approximated IM:

$$y_t = f(x, u, t)\Upsilon_t + H_t\epsilon_t \tag{4.5}$$

where y_t is the system output, the observable variable combination of one or more states x_t. Since the IM Υ_t is an approximation to the true inverse dynamics, after the algebraic cancellation, a residual error namely ϵ_Υ remains such that the system output:

$$y_t = \epsilon_\Upsilon + H_t\epsilon_t, \tag{4.6}$$

is a composition of the inherent disturbances, measurement errors, and the IM approximation error.

The direct use of IM in control is known as the *perfect* control. However, as noted in (4.6), the regulation achieves the dynamic cancellation only, leaving disturbances and noise. In order to overcome this problem, IM as direct control is used in conjunction with other conventional closed-loop methods, designed to reject the remaining components.

IMs are popular in advanced control techniques such as *feedforward control* and *internal model-based* control, where the IM acts to reject disturbances, assisting a general-purpose closed-loop algorithm. It is relevant to note that due to the intricate nature of IM dynamics, most applications are limited by the analytical solution of such, meaning no general approach is feasible at the moment of writing.

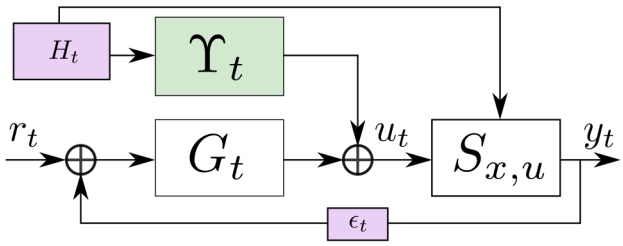


FIGURE 4.2: Feedforward Control scheme. The IM Υ_t provides disturbance rejection while the main controller G_t regulates the system. In this configuration, IM serves as an auxiliar control.

4.4 Inverse Models learning

The IM Υ_t approximation utilizing a DNN is composed as a supervised problem. The selected architecture in this dissertation for approximating all the dynamic structures, direct or inverse, is the NARX identification model. The process for the SI for an IM of a given dynamic system problem with aims to control is:

1. Data collection
2. Regressor vector φ_t construction
3. Inverse training dataset arrangement
4. DL architecture selection
5. Training
6. Evaluation and Optimization

To detail, consider the nonlinear dynamic system (1.1) in the expanded form that contains the observable vector $\boldsymbol{y}_t$:

$$\dot{x}_t = f(\mathbf{x}_t, \mathbf{u}_t, t), \quad \mathbf{y}_t = h(\mathbf{x}_t, \mathbf{u}_t, t), \quad t \geq 0 \tag{4.7}$$

where the nonlinear function f describes the state vector evolution through time, and the function h maps the observable variables in function with the state variables. Usually, function h has a one to one transformation, i.e., one or more state

variables are the system output $\mathbf{y}_t$. For simplicity, we assume that the transformation is such that, $\mathbf{y}_t = \mathbf{x}_t$.

Data collection: The specific requirements for the dynamic systems identification have the prerequisites mentioned in Chapter 2.1. To summarize it, data must contain the full range of relevant dynamics in the system. This work uses a uniform random distribution input with σ^2 variance and mean with a normalized value. In other words, our excitation signal has a probability distribution $p(x) = 1/b - a$, with the interval $a, b \in [0, 1]$. The resulting observable variables from the controlled system form the state vector $\mathbf{x}_t$.

Regressor vector construction: By the generation of input-output pairs and to reshape the data into a supervised learning scheme, vectors $\mathbf{u}_t$ and $\mathbf{y}_t$ are collected as: $\mathbf{u}_t = p(x)$ and $\mathbf{y}_t = \mathbf{x}_t$ for a sufficiently large number of iterations T, $t \in [0, T]$ using a simulator or a real-world system. After collection, the regressor vector forms:

$$\varphi_t = [y_{t-i}, u_{t-j}], \quad i \in [0, n_a], \quad j \in [0, n_b] \tag{4.8}$$

with n_a and n_b as arbitrarily selected constant values.

Inverse training dataset arrangement: The training data is formed by *features* and *labels*. For the sake of simplicity and to differentiate between inputs-outputs in the control theory context, the inputs to a DL architecture will be called features: $X \in \mathbb{R}^{n_x m}$, and the outputs to a DL architecture will be called labels: $Y \in \mathbb{R}^{n_y m}$. The IM rearrangement is defined such that:

$$X^{n_x m} = \varphi_{y,u} = [y_{t-0}, y_{t-1}, ..., yt - n_a, u_{t-1}, u_{t-2}, ..., u_{t-n_b}] \tag{4.9}$$

$$Y^{n_y m} = \varphi_u = [u_{t-0}] \tag{4.10}$$

where n_x being the features size, n_y the labels size and m as the number of samples (input-output pairs). As in most DNN cases, results are closely related to the data availability. Such issue

DL Architecture selection: By assuming the existence of Υ_t, and after considering the analysis performed in this chapter, the IM is considered a complex, nonlinear function. Therefore, it is expected that more creative DL architectures would perform better in the mapping. However, as demonstrated by our experiments in further chapters, the solution is not straightforward. Our initial attempts were with selecting FFN as the baseline of comparisons, moving towards LSTM, CNN, Autoencoders, and Attention mechanisms, such as the Transformer.

Training: Training algorithms are closely related to the selected DL architectures. In this work, we used for all of our models the *backpropagation through time* or BPTT [121], a gradient-based method for DL architectures with feedback. The algorithm and specifications are detailed in Chapter 5, experimental framework.

Evaluation: We utilized the direct control method and a mean squared error (mse) cost function for the IM assessment. Placing the IM Υ_t as a series controller to the controlled system, minimizing the error function (3.1), we validated the correct results of the training. As for the residual elements, the unknown disturbances and the process noise were considered a random variable distribution, i.e., white noise.

Optimization: Our experimental setup is designed to find optimal settings in IM with aiming to control. As mentioned earlier, parameter selection has a profound impact on the overall execution of the controlled system. Therefore, we applied an optimization search method to optimize DL architecture and parameters based on neuroevolution. The details regarding the algorithm will be expanded in Chapter 4.

4.5 Adaptive Inverse Control

The offline derivation of IM generates a powerful assistant technique for more elaborate control schemes. When the IM is characterized by a NN, however, an additional capability can be exploited: the ability to adapt online. Online adaptation refers to the modification of the model's internal parameters using new *production* information. Initially, since the model is trained offline, we have high expectations regarding its control performance, but as the controlled system is expected to be time-variant, different zones of operation may start to appear, rendering the IM unsuitable for these new changes.

As for the online *adaptation*, the model must collect a set of new training data with a minimum length n_x and n_y before performing a new iteration [1]. In addition, the internal weights of the NN must provide an overall better response according to the cost function to continue with the update.

[1] Epoch in the machine learning context

Assumption 4.5.1 (Stability). *All controlled system of the form:* $\hat{x}_t = f(x, u, t)$, *are asymptotically stable or make stable through closed-loop even when their non-linear functions* f *are unknown.*

If the controlled system's dynamics are unknown, the adaptive control makes the assumption 4.5.1. Conversely, the system must be stabilized by empirical methods beforehand. Thus, stability concerns constitute a considerable drawback for the adaptive inverse control.

Assuming an adaptive execution and corresponding online updates, an error mismatch, defined as ϵ_a, is present at all t during the system's execution. The sources for ϵ_a include measurement noise, unknown disturbances H_t, and the offline training error.

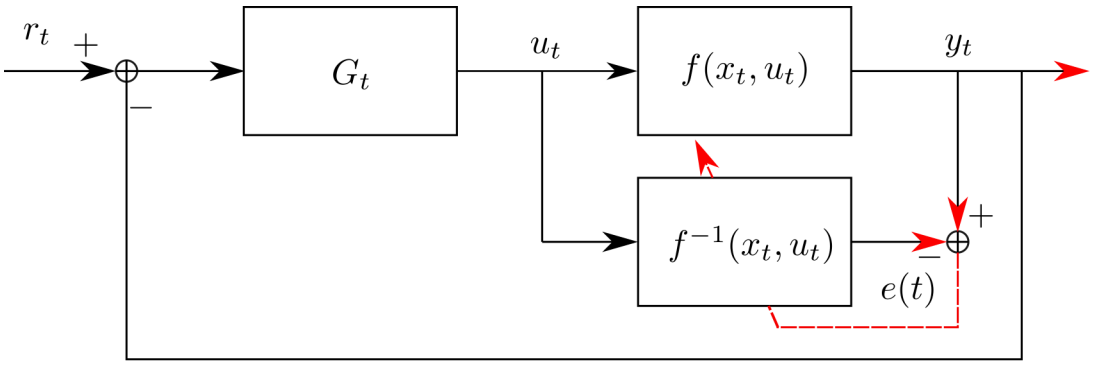


FIGURE 4.3: Adaptive Inverse Control scheme. The Inverse Model is constantly updated online using the difference between the model and the controlled system. An additional controller G_t working as a filter, attenuates measurement noise and additional disturbances.

4.6 Design roadmap

For the successful design of control systems, some parameters must be established, either arbitrarily or given by the system. For the cases where dynamics are not well-understood, limitations must be set according to the limited knowledge. Hence, at least the boundaries are required.

Assumption 4.6.1. *Boundaries* $\boldsymbol{x}_t \in [x_a, x_b]$ *and* $\boldsymbol{u}_t \in [u_a, u_b]$ *are known. The nonlinear dynamics* f *of (1.1) may not be known or stable.*

The limitations of assumption 4.6.1 serve multiple purposes. Firstly, such boundaries allow the application of the excitation signal used for training data collection. Then, the optimization algorithm would search solutions in a limited space on the MPC scheme, even though it is a nonconvex problem. Finally, for the cases of unknown dynamics, the process is slower but secure and reliable.

Our objective is to design an optimal Inverse Predictive Control (IPC) to control a nonlinear dynamic system of the general form (1.1) through the toolset defined in previous chapters.

To proceed with the design, we follow the next steps:

1. Predictor model construction
2. Optimization algorithm setup
3. Control system settings
4. IPC optimization
5. Analysis and Simulation

4.6.1 Predictor Model construction.

IPC relies on accurate predictions for the correct solutions to the online optimization problem. Therefore, accuracy concerns are the top priority for the design of predictive controllers, for the algorithms are solving the problem based on an approximation model and measurements. The predictions will be generated by the NARX model (2.3).

Let us denote the control problem system S_t based on (1.1) as:

$$S_t := \mathbf{y}_t = \mathbf{x}_t, \quad \dot{\mathbf{x}}_t = f(\mathbf{x}_t, \mathbf{u}_t), \quad t \geq 0, \tag{4.11}$$

with $\mathbf{x}_t \in [\mathbf{x}_a, \mathbf{x}_b]$ and $\mathbf{u}_t \in [\mathbf{u}_a, \mathbf{u}_b]$. The system S_t is then *excited* by the signal $\mathbf{u}_t = \mathcal{R}$ with a normal distribution over $[\mathbf{u}_a, \mathbf{u}_b]$, to generate the training dataset $\mathcal{D} = \{x_{i,t}, u_{i,t} | t < m\}$, with size m.

Remark 4.6.1. *The dataset $\mathcal{D}$ of size m must contain the most relevant information to approximate the nonlinear dynamics properly. This applies to the magnitude and transitory values. On the other hand, real-world datasets are expected to improve results by adding actual conditions beyond those created with a simulation.*

Having dataset D, the regressor vector φ_t defined in (2.4), constituted by lagged values of $\mathbf{y}_t$ and $\mathbf{u}_t$ of arbitrary size n_a, n_b respectively, is used as an input to a DNN $\Phi_{\varphi_t, w}$ such that, $\hat{\mathbf{y}}_t = \Phi(\varphi_t, w)$.

The weight values w corresponding to the selected DNN are calculated using the BPTT algorithm, with *batch* training. The delay length of both *features* and *labels* is closely related to the system dynamics. The optimal values are, however, computed by an evolutionary algorithm, described later in this chapter. Dataset $\mathcal{D}$ is a matrix of m-rows and columns:

$$D_{columns} = n_a + n_b + 1 \tag{4.12}$$

The training aims to minimize the difference between sequences of the measured variable $\mathbf{y}_t$ and DNN predictions $\hat{\mathbf{y}}_t$[2]. Finally, to summarize the creation of the Predictor model based on NARX architectures, Algorithm 2 describes the training and evaluation procedures.

4.6.2 Optimization algorithm setup

The MPC computes an optimal control signal using an optimization algorithm with a numeric approach, testing the potential solutions with a predictor model.

[2]Detailed numeric simulations are provided in Chapter 6.

Algorithm 1 DL-NARX Predictor training

▷ Initialize DL Model
$m \leftarrow$ MLP, LSTM, AutoEncoder, etc.
$x \rightarrow$ Targets
$u \rightarrow$ Features
for batch training **do**
  for epochs number **do**
    ▷ Construct NN input matrix from training data
    $a_{i,j} \leftarrow$
    $[x[k-1], ..., x[k-m], u[k-1], ..., u[k-n]]$
  $i \rightarrow$ Batch size
  $j \rightarrow (m+n)+1$ ▷ State and Input vector Delay values
    $\hat{x} \leftarrow m(a)$
    ▷ Compute loss function
    $e \leftarrow \frac{1}{i}\sum_{k=1}^{i}(x[k] - \hat{x[k]})^2$
    ▷ Update Network Weights
  end for
end for

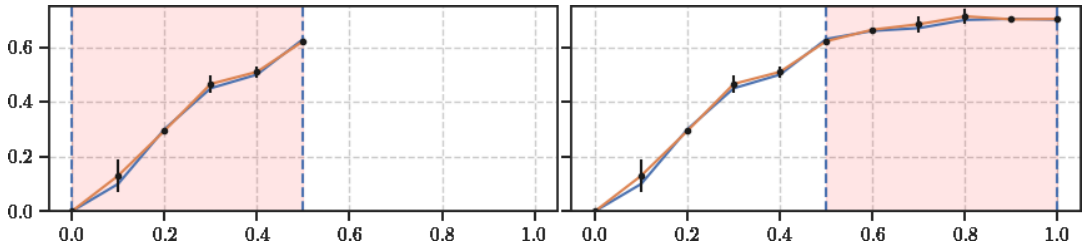


FIGURE 4.4: Predicting x_t example. The receding horizon approach can be seen in the subsequent calculations in the future with a fixed window. The larger the window, the more information about the future behavior of the system. With these values, the rest of the algorithm proposes optimal control signals.

The optimization algorithm must contain specific characteristics to be helpful in a predictive control scheme. In essence, any population-based method could suffice. However, it is crucial to keep the computational effort in mind since calculations are performed *online*, at every time-step [3].

The optimization algorithm solves cost function J (1.4) considering the MPC parameters, prediction w_p, and control w_c windows, i.e., the calculations are limited to a finite, relatively small future window. Therefore, an important consideration

[3]For the nominal MPC. Other methods, such as *skipping*, compute solutions assuming no relevant changes in an arbitrary relatively small future.

is the number of calculations performed by the algorithm, which means selecting the most time-efficient. We considered the following algorithms:

1. Gradient-descent-based Broyden-Fletcher-Goldfarb-Shanno (BFGS)[131]
2. Differential Evolution (DE)[83]
3. Particle Swarm Optimization (PSO)[50]

The numerical approach to this problem means that in sequence form, a control signal solves the cost function limited by the prediction and control windows. In other words, the optimal solution is valid only in the range $[t, t + wp]$. Next, the MPC algorithm uses the first element of the optimal sequence u_0 to control system S_t, discarding the remainder because solutions must be found at each step.

4.6.3 Control System settings

The essential parameters in MPC are the prediction and control windows. These values directly impact the algorithm's performance and overall quality of the response, i.e., the control signal. Therefore, the selection must be carefully made since too large values yield better responses but may be prohibitive for fast dynamics systems.

By considering such a tradeoff, popular values decide the window of potential disturbances presence. While this requires previous knowledge of the system, experimentation and observation can also compute optimal window values. Figure 4.1 shows the difference in prediction and control windows selection, noting the many points affecting the optimal solution convergence.

4.6.4 IPC Optimization

The optimal selection of Predictive Controllers depends on the actual system execution in a closed-loop. However, since most disturbances are unknown beforehand, active, online updates can be considered for an adaptive implementation, especially for the complex systems in which dynamics are expected to change or are unknown.

On the offline preparation and design, window parameters can be adjusted through simulations. Therefore, the strict requirement for a simulation benchmark is an accurate approximation of the system and its environment. Since our designs include a DL-NARX model, the system's S_t accuracy is considered sufficient.

For the optimization of IPC parameters, we propose a heuristic search optimization algorithm [4] based on evolutionary computation. Experimentation showed an overall improvement in the offline parameters selection. However, these are problem-dependent, and a specific implementation was required for each numeric simulation. The precise details for each specific numeric simulation are defined in Chapter 5.

4.6.5 Simulation

Figure 4.2 displays the DMPC simulation model based on the DL-NARX predictor and an optimization algorithm, with optimally selected prediction and control windows, given a cost function. The control system is expected to automatically and optimally drive the system's observable variables $\mathbf{y}_t$ and internal states $\mathbf{x}_t$ to desired values $\mathbf{r}_t$ despite the presence of noise and disturbances. Moreover, with the adaptive capability of the DL-NARX model, the predictions can be updated from new information, should the system S_t internal parameters vary with time.

4.7 Inverse Model seeding

The control problem solutions and adaptive mechanisms provided by the predictive control setting require a fast computation within a time step. The system dynamics, given by S_t (4.1), dictate the time step length, following the Nyquist-Shannon sampling formula where the rate must be greater than $2B$, such that $B < f_s/2$, being B the highest frequency measured in the system and f_s the selected sampling rate. Such frequencies imply that relative fast systems have less time margin of operation, which may cause Predictive Control methods to be unsuitable.

It is under such considerations that the need for fast MPC implementations remains an ongoing research problem. Therefore, an Inverse Model control signal is used as the initial solution for the optimization method to overcome the problem.

[4] Which is different from the optimization stage of the MPC execution.

Remark 4.7.1. *The Predictive Control system has a f_s sampling rate to compute an optimal signal. However, the system S_t (4.1) disturbances are unknown; therefore, a warm-starting*[5] *method is impossible.*

Assumption 4.7.1. *The nonlinear control problem described by S_t is an online optimization problem is defined as non-convex, such that,*

$$f(y) =\leq f(x) + (\nabla f(x), y - x) + \frac{\beta}{2}\|x - y\|_2^2, \tag{4.13}$$

$$f(y) =\geq f(x) + (\nabla f(x), y - x) - \frac{\beta}{2}\|x - y\|_2^2, \tag{4.14}$$

$$\beta = sup_{x\in\mathbb{R}^d}\|\nabla^2 f(x)\,| \tag{4.15}$$

where $x, y \in \mathbb{R}^d$ whereas for convex has a linear lower bound, such that:

$$y(t) > f(x) + (\nabla f(x), y - x). \tag{4.16}$$

Theorem 4.7.1. *The control signal u_t computed by the Inverse Model approximation $\Upsilon_{\varphi,t}$ based on a DL-NARX approximation is the near-optimal solution of the control problem.*

Proof. Considering assumptions 4.7.1 and given $\Upsilon_{\varphi,t}$ is the optimal approximation of f^{-1}, system S_t output y_t equals to:

$$y_t = f(x_t, u_t)f^{-1}(\varphi_t) + H_t\epsilon_t, \tag{4.17}$$

whereafter the shown operations, the final output yields:

$$y_t = H_t\epsilon_t, \tag{4.18}$$

as described in Chapter 2, leaving only the unknown and noise elements. □

Since the dynamics are solved by the inverse model $\Upsilon_{\varphi,t}$ the cost function J takes optimal $\mathbf{u}_t$ as initial solution. Therefore, calculations to find the optimal

[5]Warm-starting method: To utilize previous solutions, skipping calculations when known disturbances are not expected in the foreseeable future.

$\mathbf{u}_t$ are greatly reduced in the practical sense. Ideally, convergence is achieved in one iteration; however, considering (4.8), the optimal control signal must contain elements to compensate the remainders.

Assuming the iterations number is significantly reduced, the possible applications range increases. Thus far, the most relevant limitation of predictive controllers, aside from the precision of the predictions, is the computational cost derived from the optimization stage. Therefore, the predictive control mechanism can handle faster dynamics by adding the IM as the initial solution.

Furthermore, optimization algorithms based on populations, such as the evolutionary inspired, can be added into the control scheme. There are numerous benefits from doing so; for instance, complete sequences are added as initial solutions. Moreover, the search space in this method is significantly larger. At the same time, this results in an efficiency problem; it also guarantees better overall solutions [6]. Finally, algorithm 3 summarizes the DL-NARX training for an Inverse Model.

Algorithm 2 DL-NARX Inverse Model training

▷ Initialize DL Model
$m \leftarrow$ MLP, LSTM, AutoEncoder, etc.
$x \rightarrow$ **Features**
$u \rightarrow$ **Targets**
for batch training **do**
  for epochs number **do**
    ▷ Construct NN input matrix from training data
    $a_{i,j} \leftarrow$
    $[x[k-1], ..., x[k-m], u[k-1], ..., u[k-n]]$
  $i \rightarrow$ Batch size
  $j \rightarrow (m+n)+1$ ▷ State and Input vector Delay values
    $\hat{x} \leftarrow m(a)$
    ▷ Compute loss function
    $e \leftarrow \frac{1}{i}\sum_{k=1}^{i}(x[k] - \hat{x[k]})^2$
    ▷ Update Network Weights
  end for
end for

[6] Naturally, the time-dependencies are dictated by the controlled systems, meaning slower systems benefit the most from population-based methods. Numeric assessments on these claims are effected in Chapter 5.

4.8 Performance

Control systems applications are divided into two general problems, *regulation*, and *servomechanism*. Depending on the reference signal r_t, also known as *desired* value, the control system acts regulating, stabilizing, and maintaining the system output to the closest to the reference as possible, ideally with zero error. On both problems, rejecting disturbances and maintaining stability are the priorities. Therefore, regulation problems maintain the system output y_t in a constant value while *servomechanism* problems are about tracking a moving signal.

DIMPC execution happens at a frequency rate of f_s, in accordance with the mentioned time requirements. The primary source of information is the regressor vector φ_t constructed with previous, lagged values of the observable $\mathbf{y}_t$ and input $\mathbf{u}_t$ variables. With such vector defined, the optimization algorithm solves the cost function J iteratively, whose solution and performance directly depends on the prediction w_p and control w_c windows. The optimal control signals $\mathbf{u}_t$ is a sequence of length w_c, and its valid only in $[t, t + T]$ with the current system feedback.

At the same time, the regressor vector is used by the DL-NARX IM to compute an *ideal* control signal $\mathbf{u}_t$, that instead of being used to drive the system, it is the initial solution of the optimization algorithm [7]. Our claim is based on improving the overall performance and response quality, enhanced by the DL-NARX Predictor model and the DL-NARX IM optimization stage. Algorithm 3 summarizes the DIMPC methodology.

4.9 Summary

The addition of Inverse Models to the MPC paradigms serves as an initial solution for the optimization stage. However, the existence of such IM is a particular problem addressed in this dissertation using deep learning architectures. Therefore, we assumed an approximation rather than the analytical solution to the inverse modeling and employed them as auxiliary solutions rather than direct controllers. We discussed the use of direct inverse controllers, which produce perfect controllers by assuming a perfect inverse. Nevertheless, we propose a realistic approach by using it as an initial solution, with a performance improvement and overall quality

[7] Or initial population for population-based algorithms.

Algorithm 3 DL-IMPC Execution

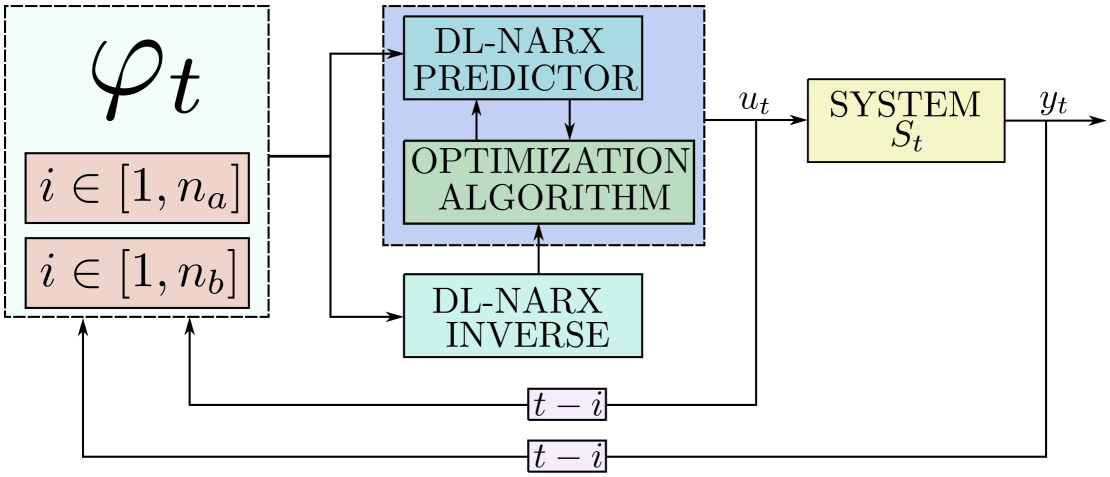
▷ Initialize DL-IMPC Controller
$m \leftarrow$ DL Predictor Model
$im \leftarrow$ DL Inverse Model
$OA \leftarrow$ Optimization algorithm (BFGS, DE, PSO)
for K-iterations **do**
    ▷ Measure current state vector, $x[k]$
    ▷ Find Optimal $u[k]$
    ▷ Compute Inverse Model Control signal
    $u_0 \leftarrow im(x[k], u[k-n])$
    ▷ Initialize optimizer with u_0 NOTE: $u_0 \approx u_o pt$
    $OA \leftarrow u_0$
    while u_{opt} not found **do**
        $u[k] = OA(\hat{x[k]}, u[k-1])$
        ▷ Compute Predictions
        $\hat{x} \leftarrow m(a)$
    end while
    ▷ Return control signal u_{opt}
end for

of the control system, tested by numeric experiments.

Using DNN as a predictor model improves the quality of the predictions, thereby producing better control signals. Moreover, the performance achieved by adding the IM as an initial solution, assumed optimal, dramatically reduces the average number of evaluations done by the optimization stage, thus reducing the overall computational load.

The remaining issues for the proposed framework are the sensitivity of the parameter selections, where different values affect the system output—in other words, selecting the correct number of hyperparameters for the prediction and inverse models directly impacts the general execution of the system. Furthermore, the particular aspects of nonlinear systems make the selection a complex task, where a random or manual search fails to find optimal values.

We addressed these uncertainties by including an evolutionary-based paradigm for the entire selection of values. Considering the nonlinear aspects of the controlled systems, the nature of DNN, the complexity of Inverse Models, and the delicate interaction between all these elements, we reformed the problem as an optimization task, with a global solution search based on Evolutionary optimization.



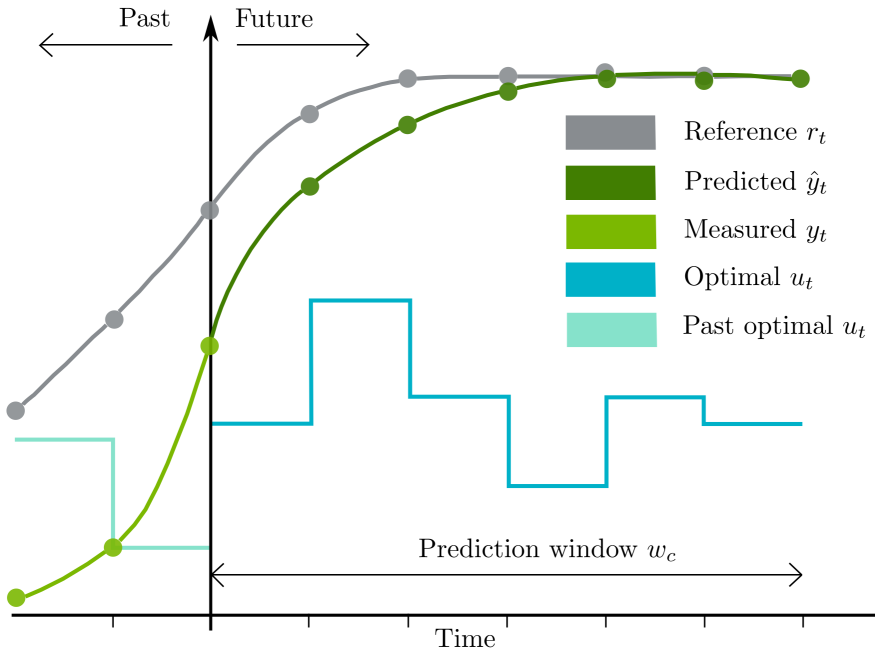


FIGURE 4.5: Complete Deep-learning based Inverse Predictive Control. On top, an overview schematic of the internal components and its interconnection with the controlled system. Below, an example of the execution of the algorithm with the predicted trajectory of the system and its subsequent online optimal computation.

Chapter 5

Deep Neural Evolution

5.1 Introduction

Deep neural networks training is performed via gradient-descent variants of optimization algorithms, fitting training data while minimizing a cost function. Although recent results are increasingly promising in several areas, such as computer vision and natural language processing, other fields contain substantial obstacles for implementing such algorithms. For example, as discussed earlier, dynamic systems and control algorithms have specific and strict requirements closely related to the training and model hyperparameters. Moreover, the nonlinear artifacts found in some applications may alter the already stochastic nature of deep neural networks. For instance, batch training and its relative size depend on the time constant of the given system.

The process of learning dynamic systems response through time involves a minimization not only of a cost error function but also on the time-dependent information buried in the training data. Furthermore, precise hyperparameter selection produces accurate representations, reduced training times, and generalizations when dealing with real-time inference in simulation models.

This chapter expands our idea of specific training designed to learn dynamic systems and control algorithms. The hyperparameter and architecture selection is based on *evolutionary* optimization, a methodology inspired by natural selection and evolution. Its application to deep learning signifies a more realistic approach

to the natural conception of learning mechanisms, focusing on a set of goals and a population of possible solutions. The natural selection foundation generates a specific deep neural network model optimized for the control problem and the dynamic system learning.

5.2 Related work

The application of optimization methods for selecting deep neural networks and their parameters has been extensively studied because of their intrinsic stochastic nature. For example, the works [11, 43] pose the difference between the simple selection method known as *grid* search and cost-based techniques, showing an improvement over the "brute-force" random selection or "manual" tunning found in several DNN applications. However, it is essential to note that implementation requires a significant amount of experimentation and manual tunning.

Additionally to conventional optimization, it has been proposed [98] to use evolutionary-based algorithms for several reasons. Firstly, such methods solve nonconvex problems, finding the optimal global solution. Secondly, assuming a sufficiently large population, the resulting DNN potentially has a better performance in addition to the learning problem. Finally, the initial population set may be seeded with known values or experience-based guesses.

The works [33, 110] dealt with the general state of the neuroevolution field, designs, and encodings, highlighting with numeric experiments the excel over a naive selection and training. The relatively simple concept of evolution holds the potential of global optimally DNN production. However, the computational load resulting from this methodology cannot be neglected.

Nevertheless, designing DNN using evolutionary algorithms is a more direct approach to emulating the natural execution of intelligence. Therefore, our implementation and application to dynamic systems and their controls are based on the premise of an optimal selection associated with the specific and strict variables found in physical problems.

5.3 Algorithm

Learning dynamic systems through DNN training involves the following components:

1. Response time (time constant)
2. Steady-state response time [1]
3. Delay time [2]
4. Response magnitude

Although such components may vary with time, a general distribution is expected to be understood beforehand. In other words, general aspects of the problem at hand must be available for the correct assumptions. However, by having an evolutionary-based method, an appropriate cost function may override the necessity of such knowledge and instead infer it from training data.

The deep evolution-based algorithm utilizes the concept of direct encoding, which means translating the variable parameters into a set of interpretable values. This set is also known as *chromosome* for the contents resemble those of genetic structures. Each solution within the population holds a particular chromosome whose values are dictated by a random function that generates individuals within constraints.Additionally, known possible solutions can be included in the initial set, for example, those used in a grid search.

The constraints vary depending on the system, but they can be selected within the system boundaries. Although a deep understanding of the system dynamics is not a strict requirement for data-based approaches, the system boundaries are included in the data collection. Therefore, such a condition is realistically applicable, especially the limits of the system operation.

[1] Non-aplicable to unstable systems.
[2] Time for certain systems to respond to an excitation input.

Thus, the constraints can be summarized as follows:

1. Response time $t_r \in [t_s + 2\tau, t_s + n\tau]$ where t_s is the sample time and τ is the system's time constant.
2. Steady-state time $t_{ss} \in [5\tau, n\tau]$.
3. Delay time $t_d \in [t_d + \tau, t_d + n\tau]$.
4. Response magnitude $y_t \in [y_a, y_b]$.

Following the mentioned constraints, an initial population set of potential solutions is generated via a random distribution function, with a strong focus on diversity. Diversity states that the number of different individuals in the population is proportional to the likelihood of achieving an optimal solution without falling in locally optimal solutions, a common occurrence in nonconvex problems.

Nevertheless, an increase in the population length, i.e., diversity, results in a quadratic complexity, where each potential solution must be evaluated at least once per iteration. Although this method is employed in this dissertation in an offline fashion, the computational load must be taken into consideration, especially for the strongly nonlinear cases, where training takes a considerable amount of resources. For a summary of the direct encoding, refer to Figure 5.1.

Network	Architecture	Activation	Batch size	u_d	y_d
LSTM	$\{32, 64, 32\}$	$\{relu, tanh\}$	64	2	2

FIGURE 5.1: DNN hyperparameters encoding example. The sequence serves as *chromosome* or *individual* in the evolutionary-based algorithm.

The evolutionary approach considers iterations as *generations* emulating the natural process of fitness and gradual progress to a defined goal. Our method uses this procedure to close the potential solutions to the global optima in a sequential combination fashion. Each capable solution has the ability to *reproduce* and transfer its genes to a new generation of potential solutions. Given enough generations, much as in nature, an optimal global individual is found, which thrived and adapted to the arbitrary condition of the given problem.

For dynamic system learning, each DNN represents an individual with the potential to recreate the actual dynamics accurately. Based on gradient descent, the internal elements are tuned to fit the provided data through training. Additionally, the parameters are selected through evolution, meaning the DNN will fit the training along with fitting the global goal.

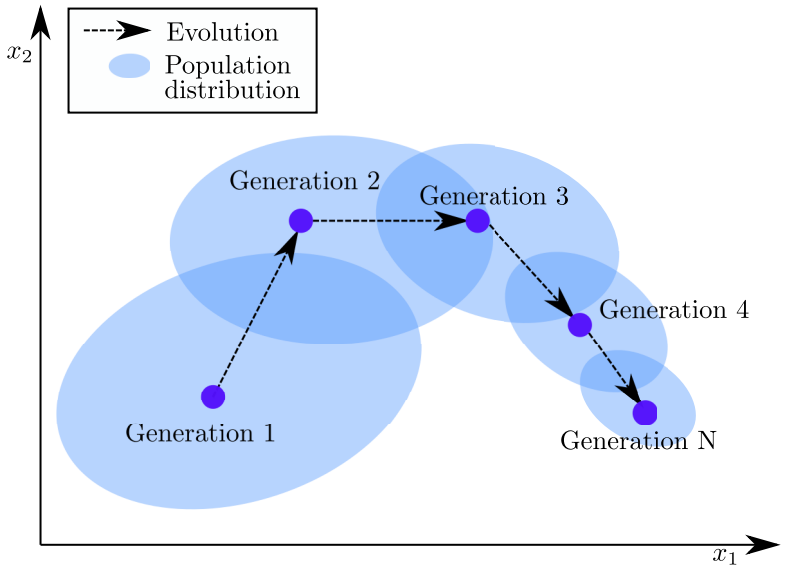


FIGURE 5.2: Evolution and distribution of populations of solutions though generations. The optimal solution and the search space are closer the longer the number of generations.

The complexity of DL architectures works simultaneously as an advantage and disadvantage because it allows approximating nonlinear systems dynamics for the price of a complicated design. It has been suggested that the most relevant attribute regarding training DNN is the amount of data; however, architecture plays a significant role as well.

The selection of hyperparameters is crucial because an incorrect set may lead to undesired results or slow training convergence, reducing the method's effectiveness and confidence. On the other hand, a proper selection involves considerable experimentation with more extensive sessions of testing and validations.

By having a cost function, we can rephrase the training as an optimization problem, modifying not only parameters but internal structures and interconnection between neurons to achieve the desired accuracy or performance.

Let us denote the set of candidate hyperparameters, including those related to the system dynamics, $\mathbf{x} \in \mathbb{R}^n$. The objective is to minimize a cost function J_T (1.4), which is defined by the problem. In the SI case, especially for a complex issue such as IM identification, error minimization is wanted.

Such candidate solutions to J_T are evaluated multiple times due to the stochastic nature of the training, and best candidates are passed to subsequent iterations known as *generations* with a combination of the best elements in the set. This

process motivates a constant adaptation to the cost function, solving the problem as in nature and the evolution process.

Although evolution requires a considerable amount of evaluations and experimentation, numeric problems such as SI greatly benefit from the optimal selection of hyperparameters. In addition, early-stopping training methods prevent an overfitting scenario. Moreover, cost functions may be proposed to reduce the architecture size for lower-powered hardware or the most performance-demanding cases.

The evolution approach taken in this dissertation is based on the Differential Evolution (DE) algorithm [76, 102]. A member x_i of the candidate solutions $\mathbf{x}$ is randomly selected [3] along with three others, to form a comparison subset $[a, b, c]$ where each member is different from each other. An operation R of recombination is performed to the comparison subset to calculate the candidate's new position, randomly modifying the structure. Afterward, the comparison subset substitutes x_i if it approaches the optimal solution better. The process is repeated for all candidate solutions, which must be ≥ 4. The number of evaluations depends on the number of candidate solutions, also referred to as a population.

Algorithm 4 Deep Neural Evolution for NARX models.

```
▷ Initialize NARX Architecture
n_a ← Input delays
n_b ← Output delays
for K-generations do
    ▷ Perform forward pass
    ŷ_t = Ψ(φ_t)
    ▷ Compute loss function
    J ← (1/N)‖y_t − ŷ_t‖²
    ▷ Perform training and weights optimization
    while J not improved do
        ▷ Differential Evolution()
        x ∈ [Network, Architecture, Activation, Batch, n_a, n_b]
        ▷ Compute Predictions
        ŷ ← Ψ(φ_t)
        ▷ Update variables
    end while
    ▷ Return optimal sequence x
end for
```

[3] A member is considered the set of hyperparameters and structures for a single DL model.

5.4 Numeric experiments

Let us consider the identification problem by training a DNN as a NARX model to learn the example system's dynamics of equation 2.11. First, the data collection is performed as shown in algorithm 1, producing a regressor vector φ_t being updated each time step in the real-time execution. For the training dataset, the system was excited, as shown in Chapter 3. Therefore, the optimal parameter selection is the main difference between conventional training and the current evolutionary-based algorithm.

The initial set of candidates, i.e., potential solutions, is randomly generated. The greater the diversity, the more range is covered within the search space, with no guarantees of a global solution. However, the boundaries selection is defined by the dynamic nature rather than arbitrary values for the dynamic systems. Thus, the problem becomes strictly constrained since optimal solutions exist within a

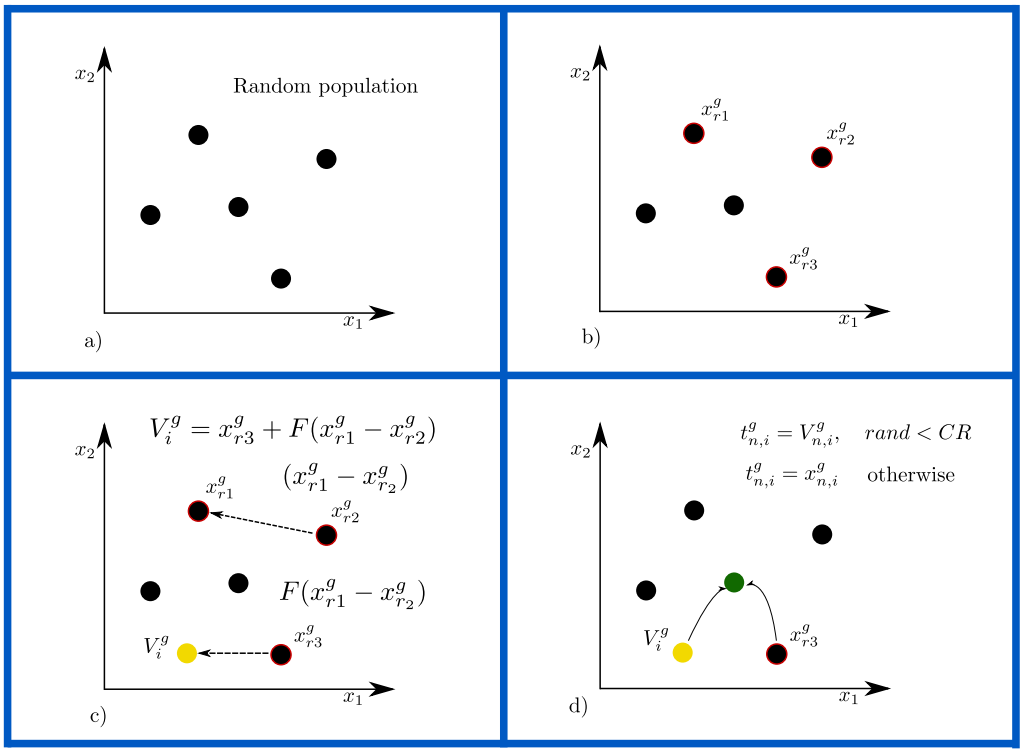


FIGURE 5.3: Differential Evolution algorithm. a) An initial population of random candidate solution is proposed. b) A random selection is performed. c) Mutation operation. d)Crossover. Steps a) to d) are repeated for every base candidate for λ times.

limited space.

With the initial set, the algorithm proceeds to evaluate the model using a defined cost function, focusing on the error between the expected output and current predictions. In order to avoid overfitting scenarios, our training parameters have an *early stop* feature, capable of terminating the training in cases where no improvements occur. Moreover, multiple training sequences are considered due to the stochastic nature of the networks and the training algorithm.

A single step in the DNE method involves the evaluation of the trained model with a single set of possible solutions, called the *individual*. The algorithm trains and tests each individual then orders them by a score –minimal error– and proceeds to select and reproduce, closely resembling natural selection. This process is called generation. The algorithm is expected to converge to an optimal solution through multiple generations.

Although a large number of generations and possible candidates increases the chances of finding the global optimum configuration, the number of function evaluations increases in parallel and the training process. Moreover, the training stage is computationally costly for the more complex cases, reducing the algorithm performance. Nevertheless, this process is performed offline during the design stages; therefore, computational concerns are more flexible.

The numeric example was performed to visualize the impact of function evaluations considering a randomly initialized, bounded set of an initial population of candidate solutions. The DNN is a feedforward neural network trained using the BPTT algorithm. A simulation procedure generated the training and evaluation dataset. Each individual was evaluated $n = 10$, and the computed error was the average.

Considering n as the arbitrary number of evaluations, and P the population size, where the algorithm evaluates each individual, the computational complexity is such that:

$$\mathcal{O} = nP^2, \tag{5.1}$$

where the larger values of n and P increase the likelihood of optimization convergence, the numeric example took x seconds to converge, yielding an average error of x.

5.5 Summary

We observed particular results regarding the batch sizes, mainly due to the relation between the training data and the time parameters intrinsic in the system dynamics. Nonetheless, the remaining parameters serve as components for the overall training, and the DNN size does not affect the output considerably from the error point of view. As for the network performance, decisions depend entirely on the intended hardware for production. For our limited experiments and the scope of our work, more constrained hardware was not considered in our experiments but is highly suggested for future iterations of this work.

Finally, to summarize the complete algorithm, the Figure 5.4 shows a flow chart containing the essential elements discussed in this chapter.

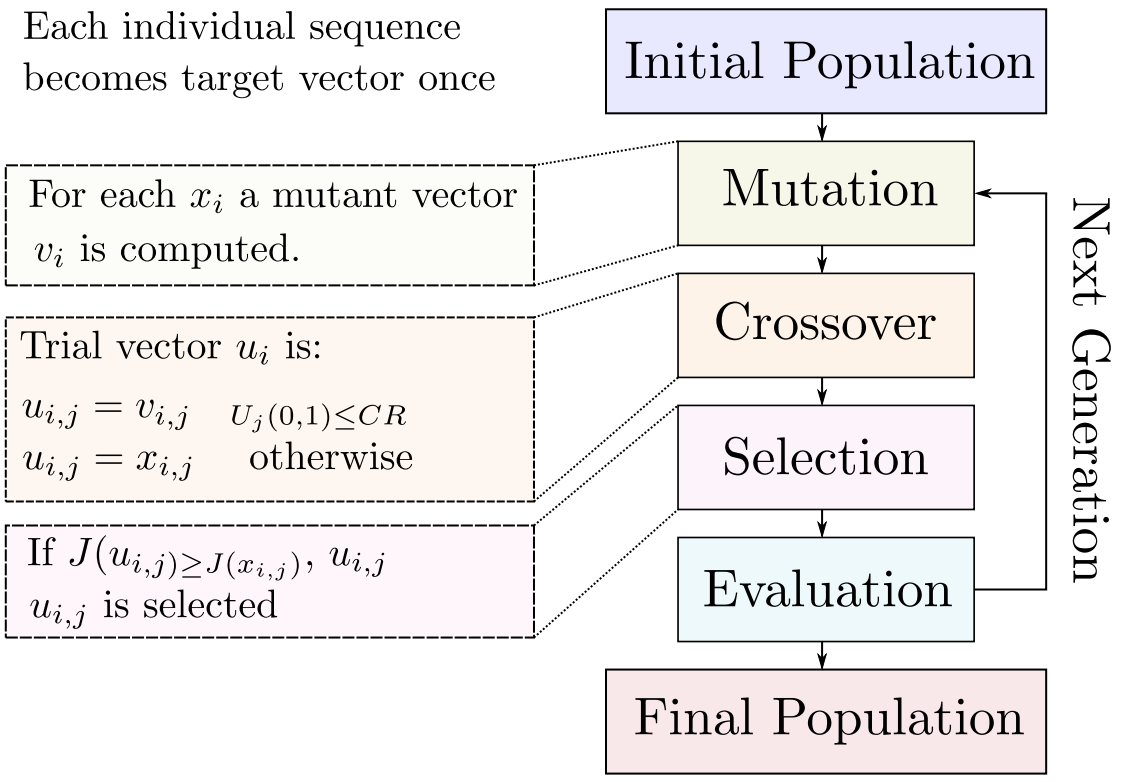


FIGURE 5.4: DNE Algorithm summary. The DE algorithm performs multiple function evaluations to decide the next generation. It is expected that the best individual contains an optimal solution. Due to the stochastic nature of DNN models, each training session is performed multiple times. (Ex. $n = 10$.)

Chapter 6

Numerical Simulation Framework

6.1 Introduction

This chapter presents the framework of the numerical experiments that deal with dynamic systems numerical and control algorithms simulations. We show the numerical approximation of the ordinary differential equation (ODE), time discretization through specific intervals and show the approximation to the solution of ODE through one-step numerical methods.

We then continue to simulate the MPC scheme, showing the parallel numerical approximations involving the studied elements: The predictor model and optimization stage, along with the controlled system. Since DL-based methods are included in the complete simulation, we show the numeric packages and the wrapping settings to achieve comprehensive results.

This chapter shows an overview of the simulation methods, software packages and particularities regarding the numeric approaches. As complementary material, Appendix A and B contain the systems equations, DNN hyperparameters and numeric results.

Finally, we show the Inverse Model control signals addition as the initial solution to the optimization algorithm, detailing a method for the parallel execution of all elements within the control system. Moreover, we describe the assessment criteria tools selected for the fair evaluation in control theory.

6.2 Dynamic Systems simulation

Let us define the ODE such that:

$$x_t = f(x), \quad x(0) = X, \tag{6.1}$$

for $x(t) \in \mathbb{R}^p$ and the time discretization though the interval $t_n = n\Delta t$. The approximation to (5.1) is defined by one-step numerical methods of the form

$$X_{t+1} = F(X_n; \Delta t), \quad X_0 = U, \tag{6.2}$$

where $X_n \in \mathbb{R}^p$ is considered an approximation to $x(t_n)$. The numerical algorithms such as the *Runge-kutta* are characterized as (5.2) assuming the overall stability [101].

Remark 6.2.1 (Benchmark systems)**.** *The dynamic systems considered in this dissertation as benchmark problems are characterized as nonlinear,non-stiff based on systems of differential equations,except when indicated otherwise.*

With (5.1) and (5.2), the dynamic systems are simulated by the numerical approximations based on one-step methods with constant time-steps $n\Delta t$. Hence, variable-step methods are not considered since control actions computed by MPC and IM are of constant frequency. Moreover, one-step methods are more stable yet slower but within the boundaries of our computational capabilities, i.e., dynamic systems numerical approximations times are negligible.

Therefore, the benchmark systems of the form (1.1) are formatted such that

$$\dot{\mathbf{x}}_t = f(x, u, t), \quad \mathbf{x}_0 = X0, \tag{6.3}$$

for a 1-D computational array time $t \in [t_0, t_0 + T]$ evenly and monotonically increasing, N-D computational array being with strict boundaries $x_t \in [x_a, x_b]$, M-D computational array with strict boundaries $u_t \in [u_a, u_b]$ and initial conditions $X0$ representing x_t values at t_0. Function f is defined by the nonlinear system case of study.

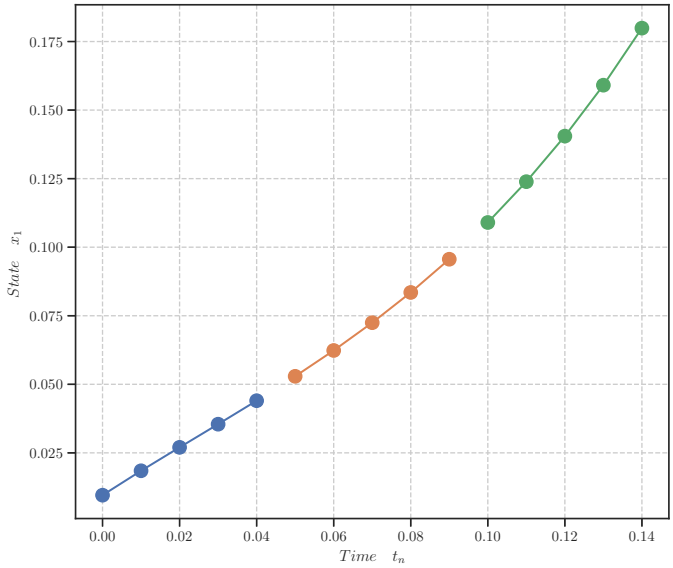


FIGURE 6.1: Numeric simulation example. The nonlinear state vector values are calculated with a numeric approach and a detailed dynamics system of equations.

6.3 Model Predictive Control simulation

MPC components act parallel within a time step t_n based on feedback data from the regressor vector φ_t. The initial simulation point t_0 assumes all conditions to zero. This assumption is not strict, meaning that an execution-ready system can also be numerically approximated. However, for the sake of simplicity, all considered benchmark problems have initial conditions set to zero.

6.3.1 Predictor Model

The Predictor Model of the form $\mathbf{x}_t = \Psi(\varphi_t)$ drove by the DL architecture Ψ is simulated following the NARX models. The training and simulation stages are performed differently as the following:

- Training algorithm 1 conducts batch learning, i.e., pieces of sequences are fed into the DL model in a feature-label fashion.
- Simulation model (Figure 2.4) acts in one time-step fashion, with newly produced predictions $\hat{\mathbf{x}}_t$ updating the regressor vector φ_t.

This distinction is important because training assumes complete knowledge of the dataset. In other words, having labeled data infers knowledge of future values of the involved variables, whereas predictions are fed back into the system for the simulation case, updating the simulation online. Furthermore, simulation models can include disturbances at any point of the simulation, which is unconsidered by our framework, therefore proving by simulation its effectiveness for the rejection.

6.3.2 Optimization algorithm

Numerical optimization algorithms are iterative methods beginning with an initial guess of the relevant variable that minimizes or maximizes a given cost function, generating improved estimations until a satisfactory solution is achieved [75]. The optimization methods use current cost function values, previous values, and boundaries to approximate a solution. While there are methods based on derivatives, our approach takes the pure numerical approach.

The primary considerations regarding optimization methods in the MPC context are based on efficiency more than on accuracy because one of the open issues remains the computational complexity. Analytically solving the optimization problem is possible only with an infinite prediction window, i.e., linear problems. As for our benchmark cases, the iterative approach method is applied.

To summarize the online optimization problem, given the MPC parameters: Prediction window w_c, control window w_c and sampling time t_n, the cost function J (1.4) is determined by the weighting coefficients $w_{x,i}$ and $w_{u,i}$ which add or reduce values to the output difference and control difference. In other words, the amount of contribution of the control signal can also be used for the cost calculation for the cases where the control effort must economize.

Table 6.1: Python 3.9.2 utilized modules for the simulations.

Module	**Version**	**Application**
scipy-odeint	1.7.1	Numerical integration for Dynamic System simulation
Tensorflow-Keras	2.6.0	DL-NARX Prediction training and simulation model
scipy-optimize	1.7.1	Numerical optimization algorithms
Tensorflow-Keras	2.6.0	DL-NARX Inverse Model training and simulation model

Remark 6.3.1. *Boundaries, weighting coefficients, and prediction and control windows constitute the basis of MPC implementations. Moreover, the significant popularity of these control schemes in industrial applications lies in the ability to produce optimal control signals from such constrained scenarios, yielding realistic and applicable optimal control.*

6.3.3 Test cases simulation

The complete control system simulation involves four elements, the controlled system S_t, the D-NARX Predictor Model, the optimization algorithm, and the DL-IM initial solution. All of these elements are simulated in parallel following a set of tools depicted as follows:

- Dynamic system numeric integration
- DL-NARX Prediction
- Optimization algorithm
- DL-NARX IM

The MPC scheme is numerically simulated using the programming scripting language Python 3.9.2. Table 5.1 specifies the particular modules used in the simulation. As for DL architectures, the Python-based libraries Tensorflow and Keras are the basis of training and simulation models.

Appendix A presents the numeric simulations product of our main algorithm, DL-IMPC. To detail the control systems utilized, we start with a description of the case study, system model, and numerical implementation. Firstly, we proceed to generate training and validation data to create the DL-NARX Predictor and Inverse models. Then, we continue with the overall algorithm simulation, highlighting the necessary parameters and constraints. Moreover, we present the comparison and evaluation with the assessment criteria presented in this Chapter. Finally, we perform evolution-based optimization for all models and algorithms, showing an increase in performance and response quality, specially designed to control nonlinear systems.

The study cases shown in Table 6.2 and A.1 are nonlinear dynamic systems with a variety of particular elements that render them *difficult* to control. Such

Table 6.2: Nonlinear Benchmark systems

Dynamic System	**Input** u_t	**Output** y_t	**States** x_t	**Ref**	**Particularities**
DC Machine	1	1	2	[69, 126]	Nonlinear
Mass spring	1	1	2	[128, 59]	Nonlinear, high frequency, noise sensibility
Mass spring with Inverse Pole	1	2	4	[17, 19]	Nonlinear, unstable
Cart-Pole	1	2	4	[73, 54]	Nonlinear, unstable
Microgrid	1	1	4	[115, 58]	Nonlinear, high frequency, noise sensibility
Lorenz Attractor	1	3	3	[62, 49]	Nonlinear Chaotic, noise sensibility
Robotic Manipulator	6	6	12	[90, 67]	Nonlinear, high frequency
Wind Farm	6	6	24	[78]	Nonlinear, multiple elements

particularities are an additional complexity to the control problem and simulations alike. Nevertheless, we selected such systems to validate our claims and show our contribution's capabilities and potential.

Furthermore, such systems are recognized benchmarks and baselines for numerous control algorithms proposals, to the degree to be considered *classic* problems, making our comparisons easier to visualize against previous work and also work as a potential basis for future ideas.

6.4 Assessment criteria

Control systems regulate and actuate over the transitory components of the time response, therefore, evaluations beyond the error are required. Conventionally, to assess the control system performance, the transitory error and steady-state error are considered as a mean of the squared difference (MSE). To exhibit a fairer comparison, the metric MSE, ITAE, and IAE [94] are employed in all control simulations. Such metrics consider the time component as well as the steady-state error. Thus, the metrics are defined as:

$$MSE = \frac{1}{T}\sum_{i=0}^{T}(r_i - y_i)^2, \tag{6.4}$$

$$ITAE = \int_{i=0}^{T} t\|r_i - y_i\|dt, \tag{6.5}$$

$$IAE = \int_{i=0}^{T} \|r_i - y_i\|dt, \tag{6.6}$$

where the error values defined as $(r_i - y_t)$ are the difference between the reference vector and the controlled output in a simulation time T.

6.5 Numeric experiments

The simulated systems allow the generation of training data aiming to emulate real-world scenarios; with such, DL-NARX models are constructed to serve as predictor elements in MPC schemes. Our aim is to approximate real-world scenarios with the addition of noise and practical considerations regarding the benchmark systems as much as possible.

As a complementary section to this chapter, Appendix B contains the numeric results considering the systems of Table A.1. To perform the experiments, we train NARX models with different DL architectures, following a process of neuroevolution, intending to find the optimal configuration that best describes the system. Then, we follow a similar approach to train models with the Inverse of the system. Finally, we construct the complete algorithm and conduct control experiments with numeric simulations. The results are compared to other control techniques as well as artificial intelligence-based methods such as a learning-to-control NN and RL DDPG. To summarize DL-IMPC application to the benchmark systems, the following steps are followed sequentially:

1. DL-NARX Predictor model optimal training
2. DL-NARX Inverse model optimal training
3. IMPC configuration
4. Control experiments and validation

6.6 Inverse Model initialization

DL-NARX IM simulation scheme is updated and executed at each t_n. Sequences of length w_c are obtained in a mini-batch simulation. From $t, t + w_p$, the predictor and inverse model initial conditions are the same, based on the current output in the regressor vector φ_t.

At the time t_0, regressor vector φ_{t_0} is constituted by the lagged values of the observable variables and input variables, for this case, where the previous t_{0-i}, the elements are considered zero.

The inverse model Υ_t depends on the feedback information similarly to the predictor model. Both models are updated with the same regressor vector at time t but with a different procedure. While the prediction model generates future responses provided by the optimization algorithm, the Inverse model directly computes control signals.

As mentioned in Chapter 4, direct control signals created by Inverse models are ideal control laws; however, since we are dealing with an approximation for the practical cases, there is not enough guarantee for the direct use. Instead, we propose to use such as *starting point* or *initial solution* to the optimization algorithm.

The optimization method converges faster with a closer candidate to the ideal solution, reducing the computational load. We propose two methods to initialize the system with Inverse signals:

1. Use the complete sequence as a starting point. This applies to all optimization algorithms that allow an initial guess.

2. Generate a sub-population of uniformly distributed sequences over the Inverse model produced. This applies to population-based optimization algorithms.

A simplification of this concept can be appreciated in Figure 6.2, where the IM control signals are added to the set of candidate solutions generated by the optimization algorithm.

Let us define the initial candidate solution as:

$$x_0 = \Upsilon(\varphi_t) \tag{6.7}$$

generated by IM υ_t. For the first method, simply $X0 = x_0$ suffices. Conversely, for the second method, the initial sub-population is given by:

$$X0 = P(x_0) \in N, \quad x_0 = \Upsilon(\varphi_t) \tag{6.8}$$

where $P(x_0)$ denotes a random distribution centered over x_0 produced by the IM.

6.7 Summary

The algorithm DIPC was formally introduced in this chapter. We showed the generation of predictions, the candidate solutions, the evaluations over a cost function, and concluded with the starting points provied by the IM. In the appendix section, several numeric simulations provide evidence of the capabilities discussed thus far.

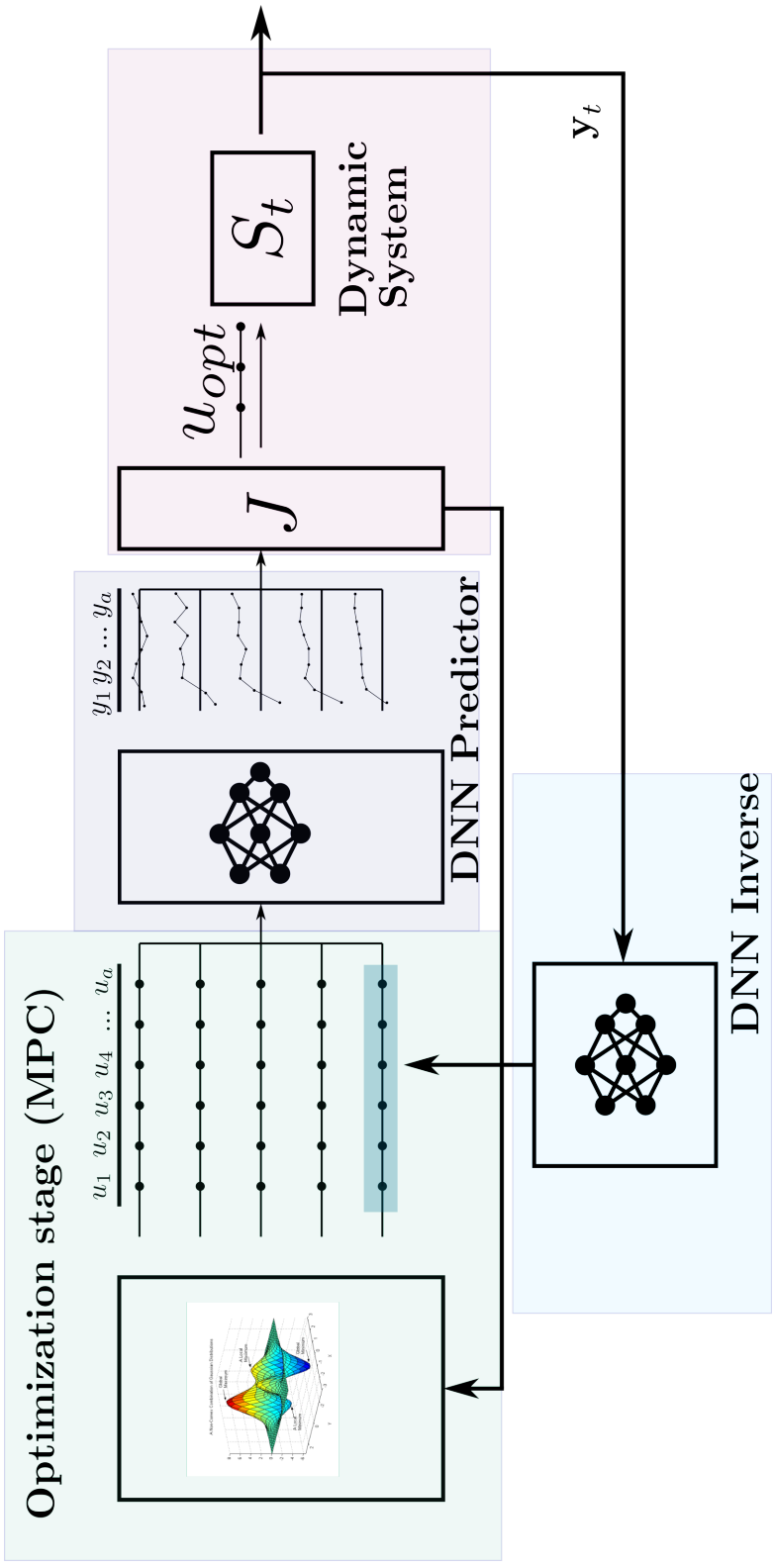


FIGURE 6.2: Full DIPC algorithm. The optimization algorithm proposes control signals evaluated by the predictor model and a cost function J. The optimal sequence is sent to the system and feedback information updates the Predictor and Inverse models. The Inverse Model provides the algorithm with an initial solution, expected to be closer to the optimal solution.

Chapter 7

Conclusion

7.1 Deep-learning evolution

The optimal derivation of DL models is a complex problem mainly due to the computational complexity involved. However, the optimal architecture and parameters selected greatly improved accuracy in our experiments, especially in the Predictor models within DL-IPC. This is a strong argument for using optimization methods in the design and training of DL methods, particularly within the dynamic systems field.

From the nonlinear test cases, we observe that nonlinear behaviors tend to make more difficult the learning stage. Most evolved networks took longer convergence times the more complex the system.

The DC Machine case is the simplest, nonlinear model. However, the model architecture became significantly complex in several layers and sizes over its linear counterpart.

For the mass-spring system and its inverse pole version, the case was similar but with the additional issue of instability. Unstable systems grow faster with time, yet the closed-loop stabilization prior to the data collection improved the final learning stage.

The cart-pole system is a simple problem with a particularly unstable and tiny

region of controllability. This challenge makes it popular with the reinforcement learning field. Conversely, the evolved NARX model had no significant issues learning its dynamics.

For the special test cases, the robot manipulator and wind farm, the accurate simulators provided sufficient data. Therefore no significant issues were noted in the evolutionary stage of optimal training of NARX models. However, it is essential to note the significance of data abundance in quality and quantity.

7.2 Deep learning Predictive Control

The application of DL-MPC facilitated the satisfaction of the control requirements at the expense of increased computation complexity. The numeric optimization algorithm had more space to cover, with greater detail, but a small time frame to deliver.

We can observe the increases in quality over other methods. Indeed using these algorithms is not a novel approach for a DC Machine, but it is for a Wind Farm. We observe that the more interconnected elements exist within a nonlinear system, the more accuracy is required for future predictions. Our experiments showed that model-based control algorithms are still the most trustworthy over direct-learning methods, such as DDPG.

Moreover, the potential adaptive features of DL-based predictors make the algorithm suitable for unknown dynamics systems. We noted in the experiments that, although performance did not increase much over long periods, modifications in the system simulation affected little the control scheme.

The experiments showed that DL-based predictors are suitable when sufficient data is available. Such data can be obtained from offline simulators, such as the robot manipulator or the wind farm cases, to supply the DL models.

7.3 Deep Inverse Predictive Control

Learning Inverse Models is a challenging topic. The resulting evolved NARX models are notably more complex than the direct dynamics version. Moreover, the

mse values could not get lower for any of the test cases. Indeed, the ideal Inverse Model is still out of reach, but the approximation was proved to be sufficient to assist the Predictive algorithm.

In the test case DC Machine, we observed an overall reduction of 59.1% in the iterations taken. This value corresponds to the nominal predictive control version, where the optimal solution is calculated every step. The reduction shows that a potential practical application is feasible.

For the test case mass-spring, the overall reduction reached 70.8%. In more complex cases, such as the Lorenz attractor, the reduction was 10.5%. We can observe that reduction values increase with the accuracy of the Inverse Model. Hence, we can state that Inverse models provide the optimal control solution. For the most complex cases, Inverse Models are not accurate enough for a more significant reduction. However, the benefits are present, and the method is applicable.

7.4 Summary and Contributions

This dissertation presented an alternative scheme for controlling dynamic systems involving a data-based learning mechanism and online optimization. Our control algorithm works in essence as a conventional method, limited by strict boundaries that guarantee a stable and reliable operation, but with the advantages of Deep-learning architectures. Intelligence-based control methods have shown excellent results in the past but lack the certainty required to be applicable in real-world scenarios. We showed the benefits of training agents capable of suggesting optimal control actions while maintaining confidence and stability in the control signals.

Our main contribution is the addition of Inverse models to the Predictive Control. While the inverse dynamics are considered ideal controllers, their derivation is not possible in most cases, yet the universal approximation feature of Deep Neural Networks provides a close estimate of the ideal control. Furthermore, as initial intuition would be the direct use of inverse models to control, we showed that using them as a starting point in the online optimization stage decreases the search time and increases the system's overall quality.

The Model Predictive Control method is a proven technique, yet its application is limited to slow dynamic systems; Thus, reducing the computational load of the online optimization stage, we expand the number of systems that can benefit from this optimal control method. As technology changes, the complexity of industrial

systems increases, and with it, more powerful control systems are needed. Numerous disturbances appear, more strict requirements, noisy environments, and overall lack of precision in complex systems are a continuous motivation for the ongoing research of the control theory.

We introduced in Chapter 1 the basic etymology in Control Theory, giving an overview of the conventional setting regarding the manipulation of dynamic systems.

In Chapter 2, we introduced the concept and algorithm of Model Predictive Control, definition, components and stages, predictions and optimization algorithms.

In Chapter 3, we commented on using artificial intelligence methods as solutions to the control problem, specifically Deep-Learning methods. We started with system identification techniques and expanded to the reinforcement learning approach. Moreover, we formalized the DL use within control theory and its combination with the Predictive Control scheme.

In Chapter 4, we introduced the Inverse Model as a concept and potential control method. A methodology for the overview and the understanding within the context of the dynamic system was provided.

In Chapter 4, our main contribution, the Deep Learning-based Inverse Predictive Control, was introduced. Details on the execution, design, and optimization via evolutionary computation were given.

In Chapter 5, we presented the evolutionary approach for the training and development of optimal DNN used in the control theory setting. The offline training concept was introduced as means of finding the optimal configuration required for the complex systems learning, specifically, the Inverse Model.

Chapter 6 provided a detailed selection of nonlinear control systems, with numeric simulations and extensive derivation for benchmarking our algorithm and future comparisons. Moreover, we showed the results of implementing our method against the conventional controllers. The numeric experiments are detailed in the appendix sections, A and B.

7.5 Future work

This dissertation expanded optimal Deep-learning methods to the dynamic systems control aiming at practical and applicable solutions. However, there are several challenges not addressed enough in this work and others.

Practical experimentation While simulated environments provide enough data for a solid approach, real-world data is still a significant limitation. Beyond our method, most deep-learning control mechanisms are limited to numeric implementations, lacking that final step to practical solutions.

Fast systems Although our method, significantly decreased the number of iterations required by an online optimization solver, the reality of the faster systems remains a big challenge. Moreover, even with the modern computational resources, the degree of mistrust in these techniques in industrial settings is a big challenge.

Inverse model The Inverse dynamics of a system may be a noncausal relationship between the input and the output. We showed the potential of accurate representations based on simulations, but testing involving real-world data could expand such systems' knowledge and proper derivation. Furthermore, our approach was based on the NARX architecture; investigation with directly learning inverse models, deep learning mechanisms, or other data-based methods is of great interest.

The study of Inverse dynamics is not limited sorely to its potential applications to automatic control. It also holds a deeper understanding of the full view and analysis of physical systems.

Appendix A

Deep Inverse Control application: Test cases

A.1 Introduction

This appendix presents the numeric simulations product of our main algorithm, DL-IMPC. To detail the control systems utilized, we start with a description of the case study, system model, and numerical implementation. Firstly, we proceed to generate training and validation data to create the DL-NARX Predictor and Inverse models. Then, we continue with the overall algorithm simulation, highlighting the necessary parameters and constraints. Moreover, we present the comparison and evaluation with the assessment criteria presented in the previous Chapter. Finally, we perform evolution-based optimization for all models and algorithms, showing an increase in performance and response quality, specially designed to control nonlinear systems.

The study cases explained in the following lines are nonlinear dynamic systems with a variety of particular elements that render them *difficult* to control. Such particularities are an additional complexity to the control problem and simulations alike. Nevertheless, we selected such systems to validate our claims and show our contribution's capabilities and potential.

Table A.1: Nonlinear Benchmark systems

Dynamic System	**Input** u_t	**Output** y_t	**States** x_t	**Ref**	**Particularities**
DC Machine	1	1	2	[69, 126]	Nonlinear
Mass spring	1	1	2	[128, 59]	Nonlinear, high frequency, noise sensibility
Mass spring with Inverse Pole	1	2	4	[17, 19]	Nonlinear, unstable
Cart-Pole	1	2	4	[73, 54]	Nonlinear, unstable
Microgrid	1	1	4	[115, 58]	Nonlinear, high frequency, noise sensibility
Lorenz Attractor	1	3	3	[62, 49]	Nonlinear Chaotic, noise sensibility
Robotic Manipulator	6	6	12	[90, 67]	Nonlinear, high frequency
Wind Farm	6	6	24	[78]	Nonlinear, multiple elements

Furthermore, such systems are recognized benchmarks and baselines for numerous control algorithms proposals, to the degree to be considered *classic* problems, making our comparisons easier to visualize against previous work and also work as a potential basis for future ideas.

A.2 Benchmark systems

Nonlinear benchmark problems and specifics are highlighted in this subsection. We commence with a simple definition, description, and numeric simulation of each one. Table A.1 contains information about the benchmark systems.

A.2.1 DC Machine

Direct Current (DC) Machines in the motor configuration are a type of electrical machinery that transforms electrical energy into rotary mechanical energy. DC motors actively use the force of a magnetic field (either constant or variable) and the internal currents to regulate their speed, and torque variables [84]. These devices are widely used in industrial applications as prime movers for more complex machinery such as servo-mechanisms from automated belts to robotic manipulators.

Specific requirements depend on the motor application. However, most systems rely on torque and speed control [42]. Therefore, regulating the speed magnitude despite the presence of undesired disturbances is a fundamental control problem for DC Motors [119]. DC Motors have two general parts: Field and Armature. Two separate electrical circuits establish the field, defined ast_1 and t_2 and another two, the armature defined as T_1 and T_2. The *series* setting involves connecting all

circuits with those of the armature. In other words, each terminal is connected to the corresponding one to make all currents (armature, field) the same such that, $i_a = i_f = i$. This configuration allows that terminal voltage to control the system's output, being speed in this case. Therefore, to denote the series dynamics of a DC Motor, considering the field and armature circuits connections, such that

$$L\frac{di}{dt} = -Ri - K_m L_f i\omega + V \tag{A.1}$$

$$J\frac{d\omega}{dt} = K_m L_f i^2 - D\omega - \tau_l \tag{A.2}$$

where L is the overall electrical inductance, R is the overall electrical resistance, K_m is the back electromagnetic force constant, L_f is the field electrical inductance, i is the electrical current, ω is the rotor angular speed, V is the terminal voltage, J is the rotor moment of inertia, D is the viscous friction coefficient, and τ_l denotes the rotor torque.

Numeric simulation

To denote the DC Motor dynamics from equations (A.1) and (A.2) in the general form of definition 2.2.1, we consider the following

$$u_t = V, \quad V \in [-12, 12] \tag{A.3}$$

$$y_t = \omega \tag{A.4}$$

$$x_t = [i, \omega]^T \tag{A.5}$$

Therefore, the DC Motor dynamic system S_t becomes:

$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\frac{Ri}{L} & \frac{K_m L_f \omega}{L} \\ \frac{K_m L_f i}{J} & \frac{D-\tau_l}{J} \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u_t \tag{A.6}$$

$$y_t = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} \tag{A.7}$$

Finally, Figure A.1 shows a numeric simulation example.

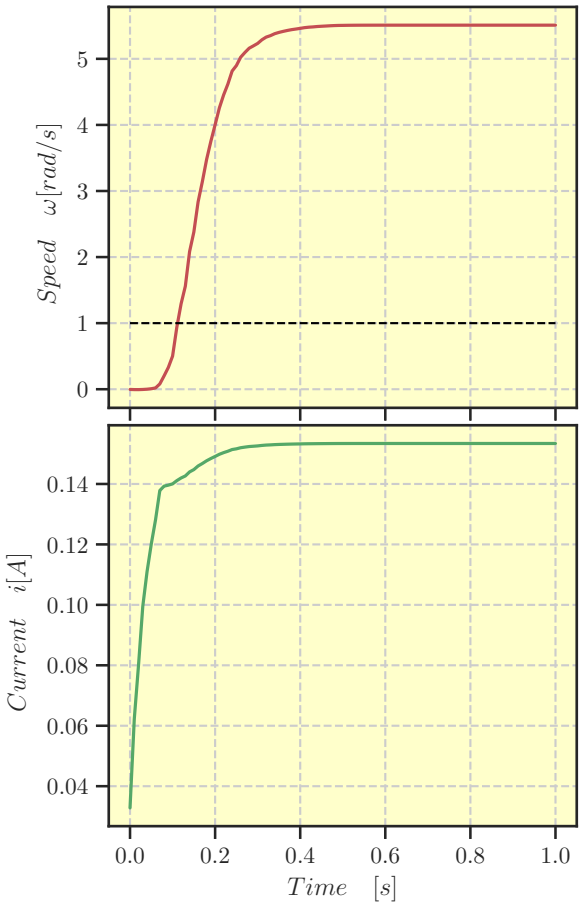


FIGURE A.1: DC Machine open-loop numeric simulation. A voltage test signal u_t of magnitude 1.0 was applied to the system producing states $[i, \omega]^T$. The observable variables y_t are the same as states in this particular case, but we are focusing on the speed ω.

A.2.2 Mass-spring

The presence of vibrational effects during the energy exchange in mechanical systems of industrial application is a severe problem for the control area. This type of physical resonance exists in virtually all components. In some cases, engineers may ignore vibrations because their bandwidth does not interfere with the regular operation. However, in other cases, active vibration rejection devices must be included in the control loops to avoid damage and instability resulting from the permanent presence of oscillations.

Mass-spring models were created under physical assumptions and analogies with energy-storage components to recreate the vibration and interaction between mechanical devices. As a result, mass-spring models have become a benchmark in the control theory field and mechanical engineering. We selected mass-spring systems because of their nonlinear nature, the high sensitivity to noise, and the high frequency of operation.

Generally speaking, the mass-spring system is not a *complicated* system to control, but it provides meaningful insight into the frequency capacity of the designed controllers.

A Mass-spring system is defined as the interconnection of mass elements, interacting through the presence of a kinetic energy-storage element called spring in a single linear axis. The basic system consists of one mass and one spring, but it is not limited to those [59]. To define its dynamics, let us consider the mass m and spring with coefficient k, with the kinetic energy as $\frac{1}{2}mv^2$, where v stands for velocity. The horizontal displacement result of an external force F applied to the system is denoted by x_i for each mass i. To visualize the active component in this study case, see Figure A.2. Ultimately, Figure A.3 display the simulated response.

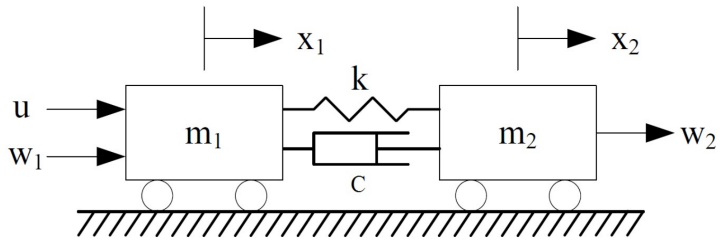


FIGURE A.2: Mass-spring diagram. The system is described as the interaction between two cars of mass m_1,m_2 with displacement x_1 and x_2 produced by an external force u. As interconnection element, a spring with coefficient k and a damping effect c provides an attenuation effect generating oscillations.

Numeric simulation

Mass-spring dynamics are derived from energy balance equations [128]. Denoted as the general case where:

$$u_t = F \tag{A.8}$$
$$y_t = [x_1, x_2]^T \tag{A.9}$$
$$x_t = [x_1, x_2, \dot{x}_1, \dot{x}_2]^T = [x_1, x_2, x_3, x_4]^T \tag{A.10}$$

constructed as the form of definition 2.2.1, such that:

$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k}{m_1} & \frac{k}{m_1} & -\frac{c}{m_1} & \frac{c}{m_1} \\ \frac{k}{m_2} & -\frac{k}{m_2} & \frac{c}{m_2} & -\frac{c}{m_2} \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_1} \\ 0 \end{bmatrix} (u_t + w_1) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{m_1} \end{bmatrix} w_2 \tag{A.11}$$

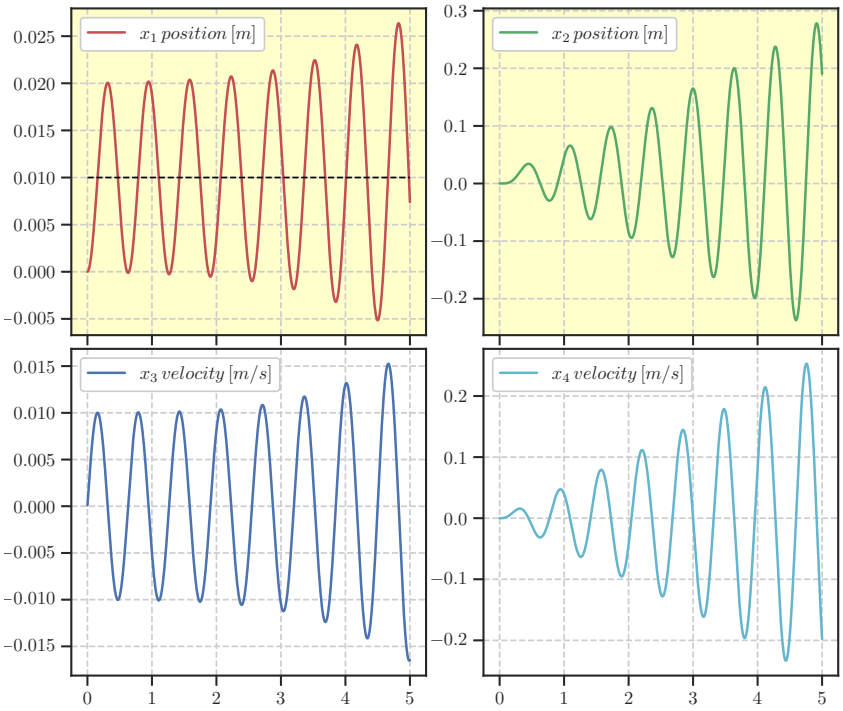


FIGURE A.3: Mass-spring system open-loop numeric simulation. The system is excited by a constant external force u_t of 0.01. Note the state variables instability, increasing in magnitude with time. Observable variables y_t are the cart positions, $x1$ and $x2$.

A.2.3 Mass-spring with Inverse pole

The experimental nonlinear control problem Mass-spring with an Inverted Pendulum (MSIP) [17] is considered. It is a benchmark problem that combines the dynamics of mass-spring energy exchange with the oscillations of an inverted hanging element. The model was initially developed as a simplified version of a dual-spin aircraft to emulate the resonance capture [87], and currently, as a study for translational motion stability [113, 19].

The system is composed of a mass-spring system connected to a cart that can provide translational displacement. It is also constrained to have one-dimensional movement only. The disturbance force moves the complete system, restricted in the x-axis only, and the inverse pole with mass oscillates in response. The system is best explained in Figure A.4.

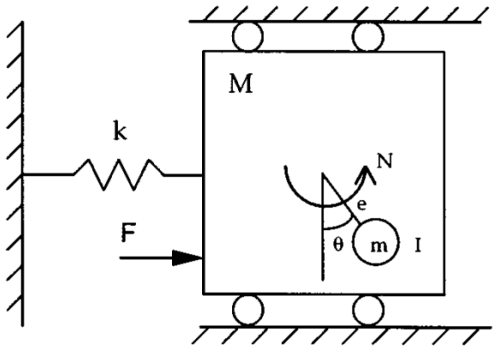


FIGURE A.4: Mass-spring with Inverse pole system. The model is used in translational motion studies that involves mechanical energy exchange and vibrations. The disturbance force F excites the system and the pole with mass m oscillates in consequence.

Through the analysis of motion with energy exchange, the dynamics equations result in the following:

$$(M+m)\ddot{q} + kq = -me(\ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta) + F \tag{A.12}$$

$$(I+me^2)\ddot{\theta} = -me\ddot{q}\cos\theta + N \tag{A.13}$$

where k is the spring coefficient, M and m are the cart and pole masses respectively, e is the pole length, I is the moment of inertia, N is the torque control input, q is the horizontal displacement and θ is the pole angle.

The torque force N serves as input to manipulate the entire system where different objectives can be considered depending on the problem; for our test case, we will consider the attenuation of the displacement despite the external disturbances.

Numeric simulation

The system defined in the general form of definition 2.2.1 is obtained through a normalization step and arranges developed in [20]. All constant parameters are equal to 1, that is $M = m = e = I = 1$, and the relevant vectors are:

$$u_t = N \tag{A.14}$$
$$y_t = [q, \theta]^T \tag{A.15}$$
$$x_t = [q, \dot{q}, \theta, \dot{\theta}]^T \tag{A.16}$$

that conform the simulation system S_t such that:

$$\begin{bmatrix} q \\ \dot{q} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{-x_1+\epsilon x_4^2 \sin x3}{\alpha} \\ x_4 \\ \frac{\epsilon \cos x_3 (x_1 - \epsilon x_4^2 \sin(x_3))}{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-\epsilon \cos x_3}{\alpha} \\ 0 \\ \frac{1}{\alpha} \end{bmatrix} u_t + \begin{bmatrix} 0 \\ \frac{1}{\alpha} \\ 0 \\ \frac{-\epsilon \cos x_3}{\alpha} \end{bmatrix} F \tag{A.17}$$

where α and ϵ are given by:

$$\alpha = 1 - \epsilon^2 \cos {x_3}^2, \quad \epsilon = \frac{me}{\sqrt{(I + me^2)(M + m)}} \tag{A.18}$$

Figure A.5 shows the numerical simulation of the system S_t.

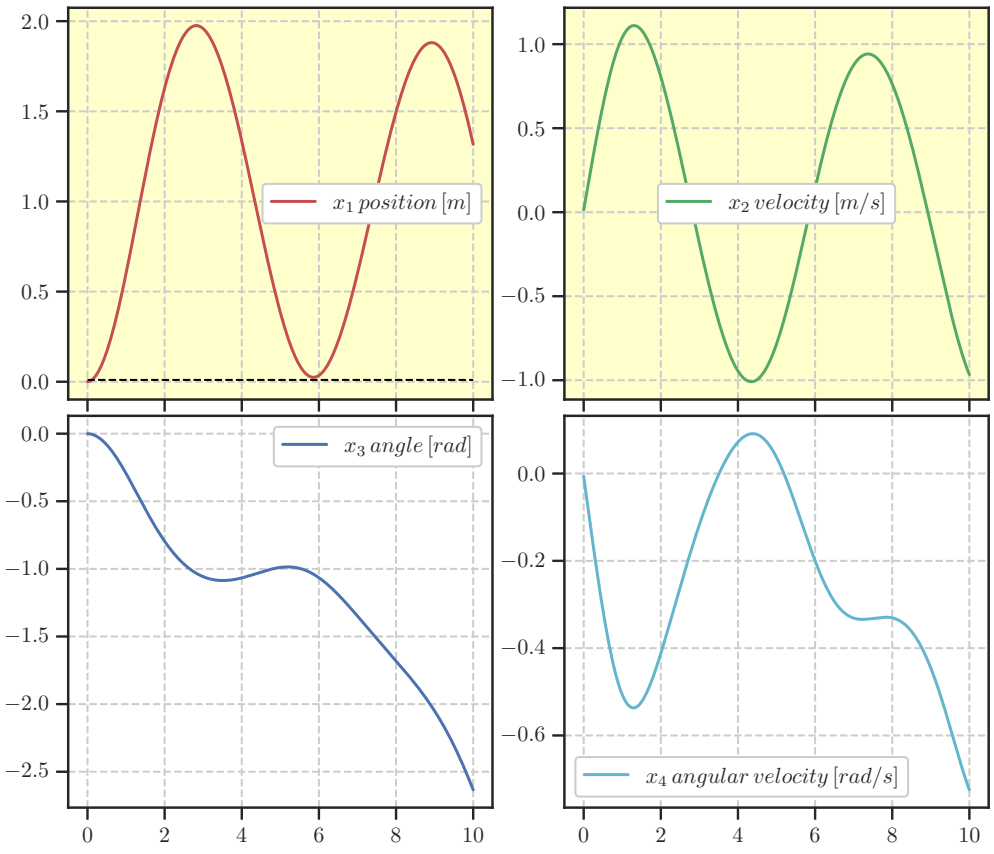


FIGURE A.5: Mass spring with Inverse pole open-loop numeric simulation. The system was excited by the torque signal $N = 0.01$. Note the unstable nature of the system.

A.2.4 Cart Pole

The benchmark problem of the inverse pendulum has an extensive history as a baseline test method for nonlinear controls, mainly focused on stabilization criteria. The improved problem is *cart-pole*, where a horizontal displacement device, limited to one dimension, performs longitudinal movement while balancing an inverse pole with a mass on top [73]. The gravity forces and the inertia product of the same movement of the cart cause angular accelerations that move the pole mass to face south in a 2-D geometric space, making the problem a challenging one. This complexity makes the benchmark attractive, not only to the control theory field but also to other approaches such as RL [54].

Figure A.6 displays a schematic of the cart-pole arrangement. The cart is excited by an external force F, producing a lateral movement that accelerates the pole with mass m and angle θ. The cart system is limited by the surface of length L, and as a disturbance force, we have gravity g.

The control objective is to manipulate the cart position, velocity, and acceleration through an internal force product of the torque generated by a motor to place the pole position in a stable state. Since the invert pole has only two stable positions, 0^o and 270^o in the vertical axis, the automatic controller will place the pole upwards.

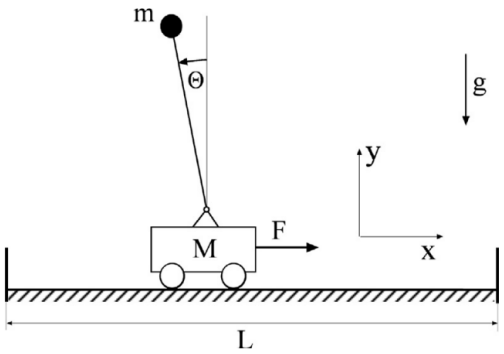


FIGURE A.6: Cart-pole system. The mass on top op the inverse pole adds an additional disturbance to the system. The horizontal displacement is used to regulate the position upwards, to a stable state.

The system setup is defined by the following equations describing its dynamics:

$$\ddot{\theta} = \frac{1}{2l - mla\cos\theta^2}\{g\sin\theta - mla\dot{\theta}^2\cos\theta\sin\theta - a\cos\theta(u - F_c) - d_m\dot{x}\} \quad \text{(A.19)}$$

$$\ddot{x} = \frac{2}{2 - ma\cos\theta^2}\{ml\dot{\theta}^2 - \frac{1}{2}mg\cos\theta\sin\theta + (u - F_c) + \frac{1}{2l}\cos\theta d_m\dot{x}\} \quad \text{(A.20)}$$

where m is the pole mass, M is the cart mass, l is the pole length, g is the gravity force, x the cart position, θ the pole angle, $F_c = f_c sgn(\dot{x})$ is the coulomb friction and $a = 1/(M + m)$.

Numeric simulation

Let us denote the general form vectors from definition 2.2.1 as the following

$$u_t = u, \quad u \in [-1, 1] \quad \text{(A.21)}$$

$$y_t = [x, \theta]^T \quad \text{(A.22)}$$

$$x_t = [x, \dot{x}, \theta, \dot{\theta}]^T \quad \text{(A.23)}$$

Therefore, the general form of the cart-pole system is rearranged such that,

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{\alpha} & \frac{m^2gl^2}{\alpha} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{\alpha} & \frac{mgl(M+m)}{\alpha} & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{\alpha} \\ 0 \\ \frac{ml}{\alpha} \end{bmatrix} u_t, \quad \text{(A.24)}$$

where α is given by:

$$\alpha = I(M + m) + Mml^2 \quad \text{(A.25)}$$

Figure A.7 displays the numeric simulation.

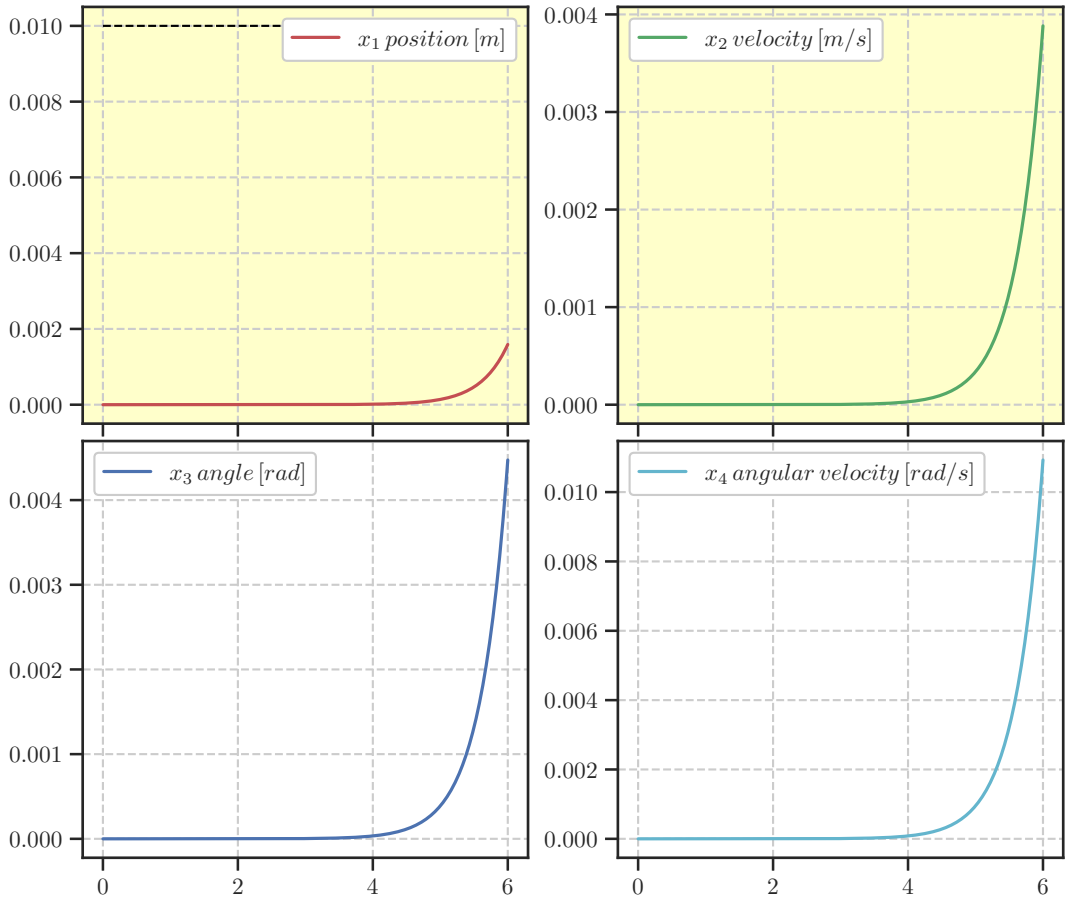


FIGURE A.7: Cart-pole numeric open-loop simulation. The system is excited by an external force u_t producing an horizontal displacement. The system states are unstable, meaning that slight changes causes divergence.

A.2.5 Microgrid

Electric power generation is a complex system of several interconnected energy sources that contribute to all systems on the planet. Therefore, the execution of the power network is critical in every situation, mainly to fulfill the load demands of quality and reliability. It is a complex situation where not robust enough souces, such as renewable generation, find a complex operation, with constant risks of instability due to their intermittent nature. Consequently, subsystems are created in an attempt to isolate the operation of different energy sources while providing their power production. These subsystems are known as *microgrids* [56].

Microgrids have the advantage of isolating their operation; with this, several issues are solved locally, for instance, stability. On the other hand, the power systems control is a large area of the control theory, with regulation systems at every variable, such as current, voltage, frequency, and power flow [9]. Therefore, locally solving these issues is a powerful idea.

Furthermore, microgrids can contemplate the contribution of different energy sources, for instance, receive and manage energy production from wind-solar plants, fossil-based generators, and more. Additionally, as in power systems control in general, locally tuned algorithms participate in the overall regulation and output of the grid.

The microgrid is modeled as a small-scale power network to define its dynamics. A popular small-scale network model is the IEEE-4 bus test [45, 13] constituted by four power generation branches. Let us consider the benchmark microgrid [115] such that:

$$\dot{\Delta f} = -\frac{1}{T_p}\Delta f + \frac{k_p}{T_p}\Delta P_d \tag{A.26}$$

$$\dot{\Delta P_d} = -\frac{1}{T_t}\Delta P_d + \frac{1}{T_t}\Delta P_g \tag{A.27}$$

$$\dot{\Delta P_g} = -\frac{1}{k_s T_g}\Delta f - \frac{1}{T_g}\Delta P_g - \frac{1}{T_g}\Delta E + \frac{1}{T_g}u_t \tag{A.28}$$

$$\dot{\Delta E} = k_e \Delta f \tag{A.29}$$

where Δf is the frequency deviation, ΔP_d is the power output change of diesel-nature, ΔP_g is the governor deviation, ΔE change in the integral parameter, T_p, T_t, T_g are time constants denoting generator, turbine and governor dynamics; k_p, k_t, k_g are the system, speed, and regulation gains. For the sake of simplicity,

as in [58] we consider $T_p = T_t = T_g = k_p = k_t = k_g = 1$ as our interest is the stabilization of the changes, i.e., controlling their convergence to zero.

Numeric simulation

Considering equations (6.27)-(6.30) and the values of the parameters set to the unity, we construct the general system form of definition 2.2.1, given that:

$$u_t = u, \quad \in [-1, 1] \tag{A.30}$$
$$y_t = [\Delta f, \Delta P_d, \Delta P_g, \Delta E]^T \tag{A.31}$$
$$x_t = [\Delta f, \Delta P_d, \Delta P_g, \Delta E]^T = [x_1, x_2, x_3, x_4]^T, \tag{A.32}$$

the system model S_t is:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} -\frac{1}{T_p} & \frac{k_p}{T_p} & 0 & 0 \\ 0 & -\frac{1}{T_t} & \frac{1}{T_t} & 0 \\ -\frac{1}{k_s T_g} & 0 & -\frac{1}{T_g} & -\frac{1}{T_g} \\ k_e & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{T_g} \\ 0 \end{bmatrix} u_t \tag{A.33}$$

where $T_p = T_t = T_g = k_p = k_t = k_g = 1$.

Finally, Figure A.8 displays a numeric simulation of the deviation effects in a microgrid subjected to disturbances.

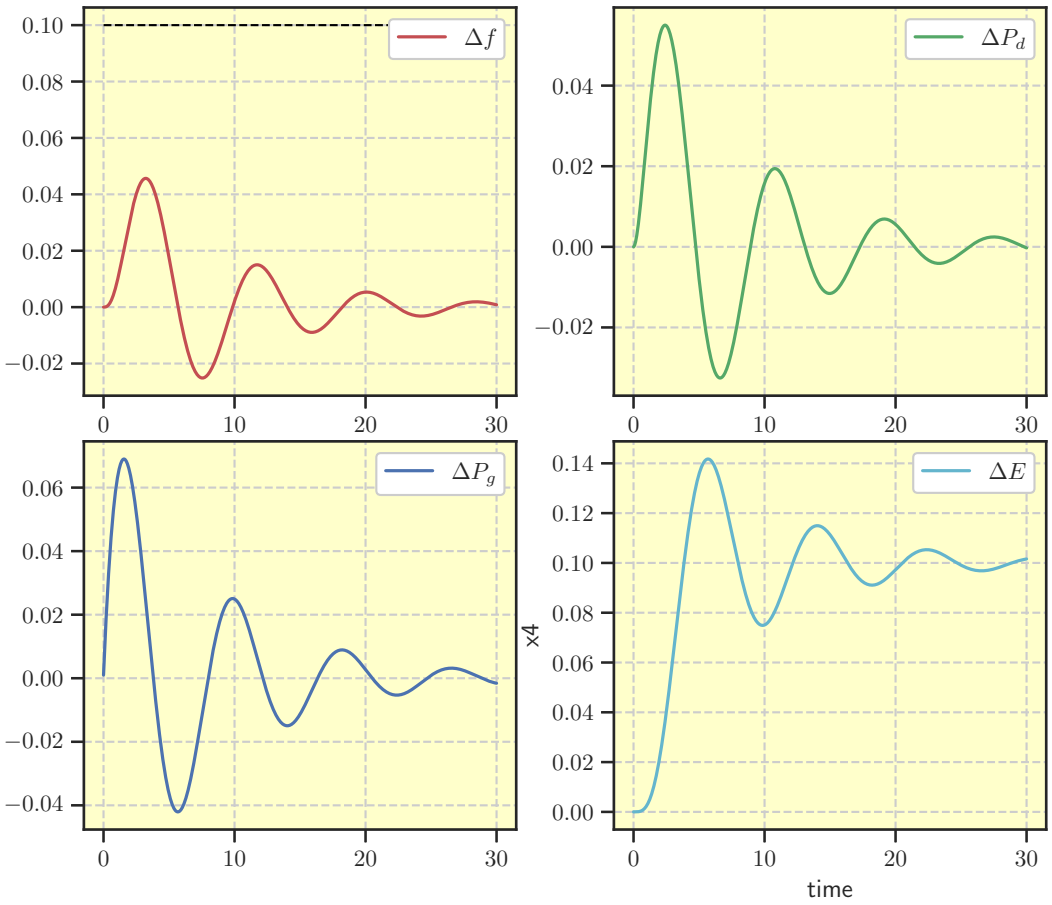


FIGURE A.8: Microgrid power system open-loop numeric simulation. The system states x_t tend to converge to zero after and event occurred (disturbance). The control objective is to ensure the convergence is done in timely and stable manner.

A.2.6 Lorenz Attractor

The Lorenz attractor is a system of differential equations with the particularity of being oversensitive to initial conditions, where even a minimal change produces significant changes in the magnitude of the variable, to the point of being *unpredictable* [62]. The Lorenz attractor is a famous characterization of the *chaotic* systems and their exciting behavior, for being an utterly causal system yet exhibiting a chaotic nature.

The Lorenz attractor has become a benchmark problem for nonlinear control algorithms for the dynamic system's field and the control theory. The controllable

version contains an input signal u_t that affects the entire system [89, 49]. The nonlinear nature of the system and the oversensitivity to changes and noise make it suitable for testing nonlinear control algorithms.

The system dynamics are based on the Lorenz equations[62], defined as:

$$\dot{x} = \sigma(y - x) \tag{A.34}$$
$$\dot{y} = x(\rho - z) - y \tag{A.35}$$
$$\dot{z} = xy - \beta z \tag{A.36}$$

based on the simplified model of atmospheric convection.

The Lorenz attractor system describes chaotic responses when the parameters $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 28$ are selected. Although the dynamics exhibit such an erratic response, their geometry is simple enough to be regulated by an arbitrary input u_t, specifically the x variable.

Numeric simulation

The notation for the general form as definition 2.2.1 as described as:

$$u_t = u, \quad u \in [-1, 1] \tag{A.37}$$
$$y_t = [x, y, z]^T \tag{A.38}$$
$$x_t = [x, y, z]^T = [x_1, x_2, x_3]^T \tag{A.39}$$

composing the system S_t such that:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} -\sigma & \sigma & 0 \\ \rho & -1 & -x_1 \\ x_2 & 0 & -\beta x_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u_t \tag{A.40}$$

where $\sigma = 10$, $\beta = \frac{8}{3}$ and $\rho = 28$.
Figure A.9 shows the numeric simulation.

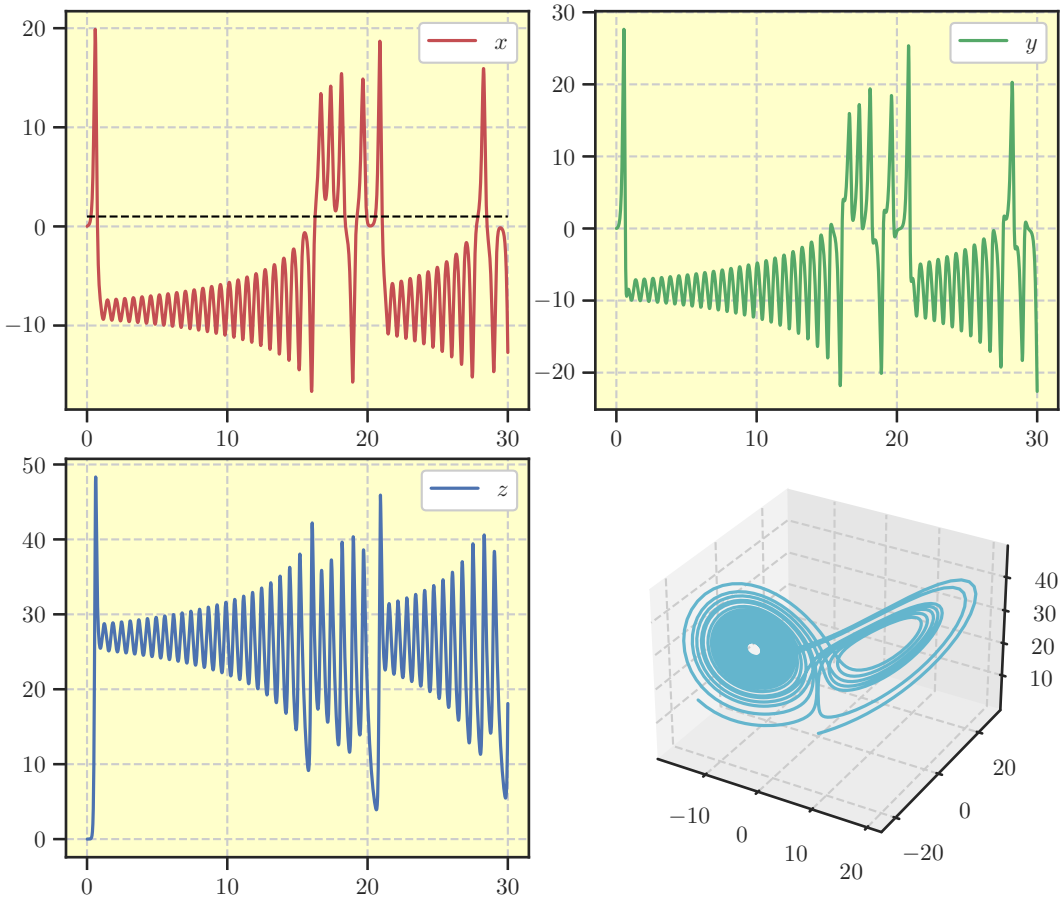


FIGURE A.9: Lorenz Attractor system of equations open-loop simulation. The system is excited by an input x_t to the x variable description. Note that initial conditions in this simulation are set to zero, however, any change produces a different form. The four figure (row=2, column=2) shows the 3D version of the system.

A.3 Benchmark systems: Special cases

Our dissertation algorithms are also proven to be effective with dealing with more advanced numeric simulations. Two systems following this fashion are included in this subsection with a general overview and details about the numerical simulator employed.

A.3.1 Robotic arm

Robot manipulators are a set of actuation mechanisms for general-purposes tasks, with automatic translation movements referenced to an end-effector device, such as a robot hand or tools. The main application area lies in industrial and automated production tasks due to their precision and overall low costs. The control of these devices has extensive literature, ranging from conventional algorithms to intelligent learning-based methods [91, 7, 47].

The control problem related to robot manipulators is present in many applications that require precision and exactitude. The main control variables are the so-called *joint-angles*, since the position of the end-effector can be estimated using the matrix of angles in a process called direct-kinematics[71]. However, in the opposite case, computing the required angles to move the end-effector to an arbitrary position known as the inverse-kinematics problem is very challenging. As a result, several attempts to solve the problem have been proposed for decades [105] with only numeric approximations available today[30].

To define the robot dynamics, in this particular case, we resorted to a numeric simulator. The main motivations are the degree of fidelity a professional simulator provides additionally to the general complexity of the system to emulate with differential equations. Therefore, we selected the open-source simulator based on the ROBOTIS open-hardware robot manipulator and MATLAB Simulink [90, 67].

The robotic manipulator is divided into several components by the simulator. Specifically, the direct and inverse kinematics are coded as subsystems, while the robot manipulator sections are composed of mechanical models of the library *simscape*. The reader is encouraged to visit the simulator online documentation for specific details. Our focus deals with the optimal control and computation of the joint-angles for a specific yet complex task.

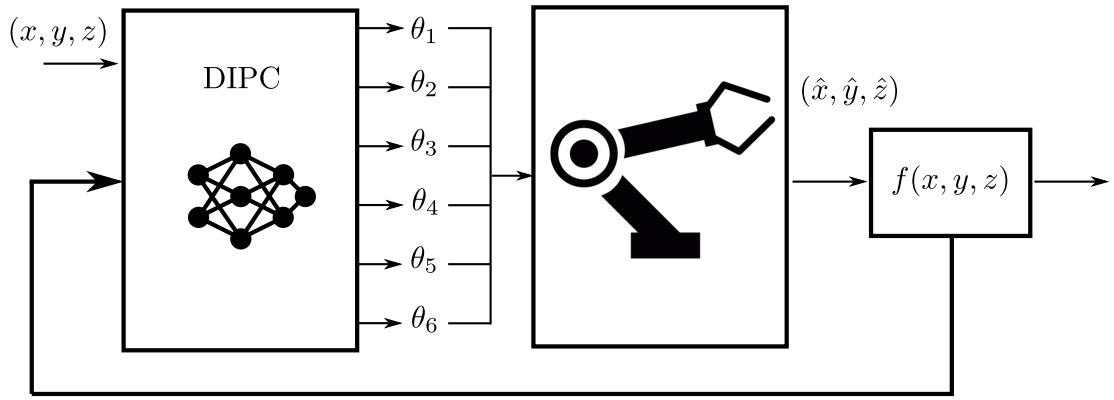


FIGURE A.10: Robot manipulator closed-loop DIPC. The six joint angles are optimally computed by our proposed method. The feedback angles are calculated by a function f known as *forward kinematics* provided by the MATLAB simulator.

The control problem is summarized as computing the optimal joint angles accurately and within the time constraints to move the end effector and meet a payload. Figure A.10 displays the general schematic with the control algorithm actuating over the robot manipulator system.

A.3.2 Wind Farm

The electric power demand has been steadily increasing in recent decades, causing the already complex power network to face additional issues [107]. Moreover, the dangerous emissions produced by the fossil-based power generation facilities pose a direct threat to all human aspects. To overcome such difficulties, recent interests have reemerged regarding renewable energy sources, specifically solar and wind-based [125]. Nevertheless, including these sources is a challenging task since their intermittent nature limits their robustness and reliability. Fortunately, there are several proposals to increase the quality and amount of power from these sources [55].

Wind farms are arrays of wind-based turbine generation devices placed on high-speed wind zones to satisfy the power demand [127]. The very nature of the wind, which is erratic and difficult to predict, makes the primary source the variable of concern. Moreover, the interaction between several wind turbines produces different phenomena related to fluid dynamics, such as the wake-effect [37]. The wake effect is created by the cooperative interaction between the numerous blades and the wind flow. Frontline turbine receives a relatively unaffected wind source, but

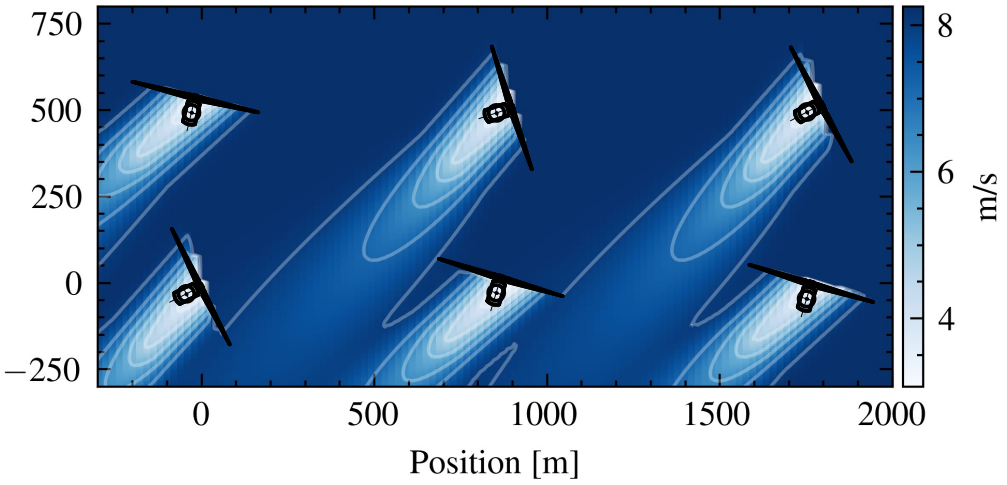


FIGURE A.11: Farm layout simulation (Top view) with wake effects. The wind velocity is shown with the intensity scale. Note the resulting wake effects of the wind contact with each turbine.

subsequent devices must deal with ripples, and residual phenomena [3]. However, by adjusting the *face* of wind turbines, the average power output of the complete wind farm may be optimized. This method is known as *yaw* control.

The analysis of wind farms is a complex, multidimensional problem. From the control theory perspective, along with the standard controllers found in power facilities, the additional yaw and pitch control poses a challenging issue. In addition, the nature of the wind makes the control designs complex, but more so the testing phases. Nevertheless, the study of wind nature is a mature field that has produced numeric tools that assist with the design and testing controls [129].

We based our numeric simulations and control algorithms on the wake-effect and optimal yaw control for this particular study case. Wake effects are complex phenomena addressed in numerous research contributions, with particular attention to the emulation of wind farm facilities design.

The selected simulator is FLORIS [78], a tool designed by NREL to study wake effects employing multiple models. The tool is based on Python and provides multiple study variables, mainly for our research, the mean and individual power outputs, yaw angles, and emulated environment. Individual wind turbines are off the scope of this dissertation but should it interest the reader, refer to the review papers [25, 114, 16].

A.3.3 Wake Effect

The wake effect is a phenomenon generated from fluid dynamics, i.e., swirl effects produced at the contact with the blades. In turn, subsequent turbines receive wind with **less speed** and more **turbulence**; Such increases are undesired due to the notable reduction of power produced under more turbulent, low-speed conditions.

As the wind stream touches the frontline turbines, an angle adjustment may change the course of the wake effects, resulting in a controlled degree of disturbance to the subsequent generators. The adjustment of the axial position of a turbine is known as *yaw control*, and its application is a convenient solution in wind-based production facilities.

Figure A.12 displays a wind stream touching the wind farm from a longitudinal axis. The effect is shown as a modified stream, highlighting that subsequent generators do not receive an ideal input.

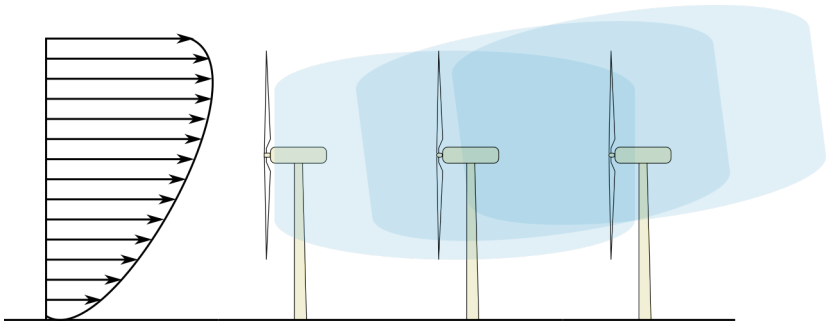


FIGURE A.12: Wind downstream. The wind components, direction, and speed touch the front line turbines, producing the wake effect and altering the subsequent generators' wind input.

A.3.4 Yaw Control

The axial adjustment, known as yaw control, adjusts the wind components' position. Such ability permits a fine adjustment allowing an utterly closed-loop system, using the generated power as feedback data. On the other hand, the real-time correction using yaw-controls requires precise information about the generated power and, more importantly, the wind components, which depend on forecasting models. Nevertheless, closed-loop control dramatically improves the power generation being the *greedy control* the most popular method.

Greedy control stands for maximum production from an individual point of view, i.e., each turbine generator seeks an appropriate yaw angle to capture the most input to produce maximum power. However, although this method successfully overcomes wake effects, it does not consider the collective work of all elements. Therefore, optimal production is not achieved. Nevertheless, the maximum power output is reached at the individual level, as shown in Figure A.13.

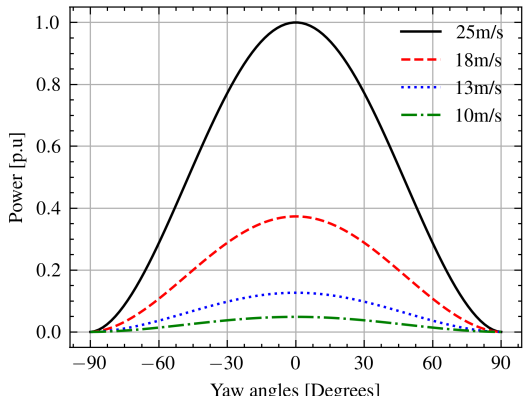


FIGURE A.13: Maximum power generation at zero degrees (Facing the wind direction) at different speeds.

A.3.5 Control experiments

In order to simulate a real-time control employing the numerical simulator for wake effects, we designed a set of control algorithms, as mentioned in Chapters 3 and 4. The reinforcement learning approach is based on the DDPG algorithm, shown in Figure A.14, where a random exploration with a reward system of maximizing the power output interacts with the simulation in an offline environment. After that, the trained agent regulates the angles of each turbine element in a real-time simulation.

The DIPC algorithm discussed in Chapter 4 and the main topic of this dissertation was used as a closed-loop controller. The **Prediction** model is based on an AutoEncoder DNN trained offline, forecasting the behavior of the complete wind farm. The **Inverse** model is based on an AutoEncoder DNN trained offline, emulating the inverse dynamics of individual generators. The optimization method selected is a gradient-based numeric approach to maximize power production. The prediction window is fixed to 20 seconds in the future, which is an arbitrary value

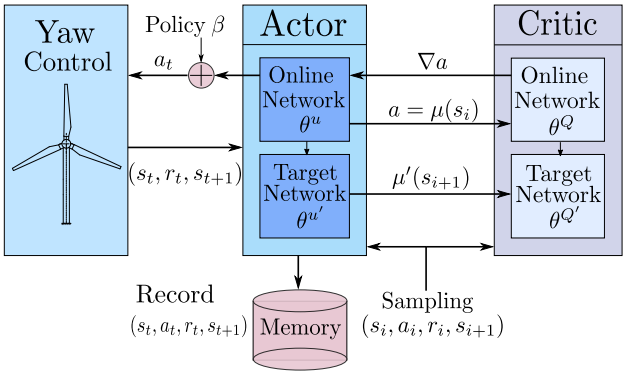


FIGURE A.14: DDPG algorithm as yaw controller. The agent learns the control signals (action) as the yaw angles, aiming to maximize the power production.

based on experimentation of the turbine dynamics. In other words, the algorithms generate a control signal every 20 seconds. The DIPC algorithm is shown in Figure A.15.

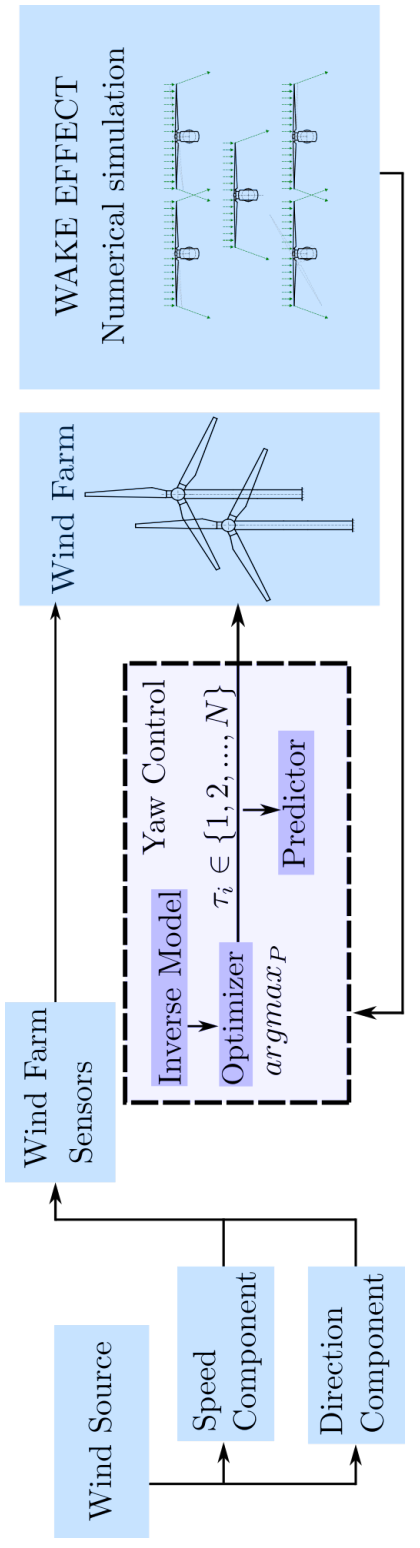


FIGURE A.15: DIPC algorithm as yaw control within a closed-loop controlled, simulated wind energy production facility. The Prediction model forecasts the future behavior of the wind farm, while the Inverse model generates initial solution for the optimal control.

Appendix B

Numeric simulations

B.1 Control simulation

B.1.1 DC Machine control

The control objective for the DC Machine system is to regulate the speed defined as $y_t = x_1$, despite the presence of unknown disturbances and measurement noise with terminal voltage u_t. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.1}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the terminal voltage $\in [-1, 1]$ normally distributed random signal. The observable variable y_t is the motor speed, and the feedback states are x_1 and x_1.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [LSTM, 221, 120, \{\text{"}relu\text{"}, \text{"}tanh\text{"}\}, 56, 2, 2] \end{aligned} \tag{B.2}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [Autoencoder, \{136, 62\}, \{"tanh", "tanh"\}, 56, 2, 4] \tag{B.3}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}, x_{t-4}]$.

The table B.1 summarizes the DL-NARX Predictor and Inverse models parameters. Note such values were optimally computed by the evolutionary approach described in Chapter 5.

Table B.1: DL-NARX DC Motor models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	LSTM
Layers	2
Cells	221, 120
Activation	relu, tanh
Epochs	111 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	Autoencoder
Layers	2 LSTM
Cells	136, 62
Activation	relu, tanh
Epochs	198 (Early stopping)
Cost	mse

DC Machine controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the DC Machine are the regulation of speed with minimal error, minimal time response, and minimal overshoot, ideally zero. Table B.2 summarizes the control algorithms involved in this experiment.

Table B.2: DC Machine Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 10\, k_i = 1\, k_d = 0.1$	[6]
LQR	Conventional	Linear Quadratic Regulator	Q=I R=[100 10]	[23]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[104]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[106]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

DC Machine control simulation

The control simulations run for 10 seconds with $t_s = 0.01$ or 10 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.2 Mass-spring control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.4}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [FNN, 343, \{"tanh"\}, 24, 2, 3] \end{aligned} \tag{B.5}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [LSTM, \{128, 62\}, \{"tanh", "tanh"\}, 30, 3, 3] \tag{B.6}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.3 summarizes the DL-NARX Predictor and Inverse models parameters.

Table B.3: DL-NARX Mass-spring models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	FNN
Layers	2
Cells	343
Activation	tanh
Epochs	23(Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	LSTM
Layers	2 2
Cells	128,62
Activation	relu, tanh
Epochs	198 (Early stopping)
Cost	mse

Mass-spring controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time response, and minimal overshoot, ideally zero. Table B.4 summarizes the control algorithms involved in this experiment.

Table B.4: Mass-spring Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 23\, k_i = 100\, k_d = 0.01$	[1]
LQR	Conventional	Linear Quadratic Regulator	Q=I R=[1000, 0]	[92]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[22]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[52]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Mass-spring control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.3 Mass-spring with Inverse pole control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.7}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [LSTM, 33, 12, \{"tanh"\}, 12, 2, 2] \end{aligned} \tag{B.8}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [LSTM, \{512, 587\}, \{"tanh", "tanh"\}, 8, 2, 2] \tag{B.9}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.5 summarizes the DL-NARX Predictor and Inverse models.

Mass-spring with Inverse pole controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time

Table B.5: DL-NARX Mass-spring with Inverse Pole models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	LSTM
Layers	2
Cells	33, 12
Activation	tanh
Epochs	89 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	LSTM
Layers	2
Cells	512, 587
Activation	relu, tanh
Epochs	485 (Early stopping)
Cost	mse

response, and minimal overshoot, ideally zero. Table B.6 summarizes the control algorithms involved in this experiment.

Table B.6: Mass-spring with Inverse Pole Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 2.5\, k_i = 0.25\, k_d = 0$	[6]
LQR	Conventional	Linear Quadratic Regulator	Q=I R=[32 10]	[23]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[22]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[52]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Mass-spring with Inverse pole control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.4 Cart-Pole control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.10}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [CNN, 8, 4, 2, \{"tanh"\}, 8, 2, 2] \end{aligned} \tag{B.11}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [Autoencoder, \{512, 512\}, \{"tanh", "tanh"\}, 8, 2, 2] \tag{B.12}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.7 summarizes the DL-NARX Predictor and Inverse models parameters.

Cart-Pole controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-

Table B.7: DL-NARX Cart Pole models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	CNN
Operations	4
Values	8, 4, 2
Activation	tanh
Epochs	786 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	Autoencoder
Layers	2
Cells	512, 512
Activation	tanh, tanh
Epochs	762 (Early stopping)
Cost	mse

Table B.8: Cart Pole Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 100\, k_i = 1\, k_d = 20$	[81]
LQR	Conventional	Linear Quadratic Regulator	Q=I R=[32 10]	[81]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[46]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[52]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

spring are the regulation of the mass position with minimal error, minimal time response, and minimal overshoot, ideally zero. Table B.8 summarizes the control algorithms involved in this experiment.

Cart-Pole control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.5 Microgrid control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.13}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [FNN, 24, 48, \{"tanh"\}, 64, 8, 8] \end{aligned} \tag{B.14}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [FNN, \{74, 28\}, \{"tanh", "tanh"\}, 64, 8, 8] \tag{B.15}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.9 summarizes the DL-NARX Predictor and Inverse models.

Microgrid controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time

Table B.9: DL-NARX Microgrid models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	FNN
Layers	2
Neurons	24, 48
Activation	tanh
Epochs	65 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	FNN
Layers	2
Neurons	74, 28
Activation	tanh, tanh
Epochs	125 (Early stopping)
Cost	mse

response, and minimal overshoot, ideally zero. The Table B.10 summarizes the control algorithms involved in this experiment.

Table B.10: Mass-spring with Inverse Pole Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 2.5\, k_i = 0.25\, k_d = 0$	[6]
LQR	Conventional	Linear Quadratic Regulator	Q=I R=[32 10]	[23]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[22]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[52]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Microgrid control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.6 Lorenz Attractor control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.16}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [Autoencoder, 543, 523, \{"tanh"\}, 256, 8, 8] \end{aligned} \tag{B.17}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [Autoencoder, \{745, 945\}, \{"tanh", "tanh"\}, 256, 8, 8] \tag{B.18}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.11 summarizes the DL-NARX Predictor and Inverse models.

Lorenz Attractor controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time

Table B.11: DL-NARX Lorenz Attractor system models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	Autoencoder
Layers	2 LSTM
Cells	543, 523
Activation	relu, tanh
Epochs	1258 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	Autoencoder
Layers	2 LSTM
Cells	745, 945
Activation	relu, tanh
Epochs	3590 (Early stopping)
Cost	mse

response, and minimal overshoot, ideally zero. Table B.12 summarizes the control algorithms involved in this experiment.

Table B.12: Lorenz Attractor system Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 1\, k_i = 0.1\, k_d = 0.001$	[26]
LQR	Conventional	Linear Quadratic Regulator	Q= R=	[68]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[124]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	[44]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Lorenz Attractor control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.7 Robot manipulator control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.19}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [LSTM, 256, \{\text{"}tanh\text{"}\}, 12, 2, 2] \end{aligned} \tag{B.20}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [LSTM, \{256\}, \{\text{"}tanh\text{"}, \text{"}tanh\text{"}\}, 12, 2, 2] \tag{B.21}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.13 summarizes the DL-NARX Predictor and Inverse models.

Robot manipulator controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time

Table B.13: DL-NARX Robot manipulator system models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	LSTM
Layers	1
Cells	256
Activation	tanh
Epochs	22 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	LSTM
Layers	1
Cells	256
Activation	tanh
Epochs	213 (Early stopping)
Cost	mse

response, and minimal overshoot, ideally zero. Table B.14 summarizes the control algorithms involved in this experiment.

Table B.14: Robot manipulator system Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	$k_p = 10\, k_i = 1\, k_d = 0.1$	[27]
LQR	Conventional	Linear Quadratic Regulator	Q= R=	[51]
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	[104]
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 0.9\, \tau = 0.005$, E=100	This work
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Robot manipulator control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

B.1.8 Wind Farm control

The control objective for the Mass spring system is to regulate the position defined as $y_t = x_2$, despite the presence of unknown disturbances and measurement noise with the external force $u_t = F$. Following our sequential steps, the DL-NARX is defined as:

$$\Psi(\varphi_t) = f[u_t, u_{t-1}, ..., u_{t-u_a}, x_{t-1}, x_{t-2}, ..., x_{t-x_a}] \tag{B.22}$$

where $f \in$ [FNN, LSTM, CNN, Autoencoder, Attention], $u_a \in [2, 10]$ and $x_a \in [2, 10]$ selected by the evolution-based optimization algorithm. The training data is generated by u_t being the external force $\in [0, 1]$ normally distributed random signal. The observable variable y_t is the mass position, and the feedback states are positions x_1, x_2 and velocities x_3, x_4.

After the neuroevolution algorithm execution, the optimal sequence found is described as:

$$\begin{aligned} g_{opt} &= [Network, \{Architecture\}, \{Activation\}, Batchsize, u_a, x_a] \\ M_{opt} &= [LSTM, \{256, 256\}, \{"tanh"\}, 24, 2, 3] \end{aligned} \tag{B.23}$$

Thus constructing DL-NARX Predictor simulation model $\Psi(\varphi_t)$ as (7.2) with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, x_{t-1}, x_{t-2}, x_{t-3}]$.

Following the similar NARX (7.1) with inverted features and labels, the optimal sequence for the Inverse models is such that:

$$I_{opt} = [Autoencoder, \{720, 64\}, \{"tanh", "tanh"\}, 30, 3, 3] \tag{B.24}$$

with regressor vector as $\varphi_t = [u_t, u_{t-1}, u_{t-2}, u_{t-3}, x_{t-1}, x_{t-2}, x_{t-3}]$.

The table B.15 summarizes the DL-NARX Predictor and Inverse models.

Wind Farm controls configuration

The DL-IMPC algorithm performance and execution are compared to conventional controls and with the RL DDPG. The particular control requirements for the Mass-spring are the regulation of the mass position with minimal error, minimal time

Table B.15: DL-NARX Wind Farm system models hyperparameters and training details.

DL-NARX Predictor	
Parameter	Values
Architecture	LSTM
Layers	2
Cells	256, 256
Activation	relu, tanh
Epochs	321 (Early stopping)
Cost	mse

DL-NARX Inverse	
Parameter	Values
Architecture	Autoencoder
Layers	2 LSTM
Cells	720, 64
Activation	relu, tanh
Epochs	652 (Early stopping)
Cost	mse

response, and minimal overshoot, ideally zero. Table B.16 summarizes the control algorithms involved in this experiment.

Table B.16: Wind Farm system Control algorithms

Algorithm	**Type**	**Description**	**Parameters**	**Reference**
PID	Conventional	Proportional Integral Derivative of the error	NA	
LQR	Conventional	Linear Quadratic Regulator	NA	
MPC	Conventional	Model Predictive Control	$w_p = 10 w_c = 10$	This work
RL DDPG	Intelligent	Deep Deterministic Policy Gradient	$\gamma = 1.0\, \tau = 0.001$, E=1000	[28]
DL-MPC	Intelligent	Deep Learning Model Predictive Control	$w_p = 10, w_c = 10$	This work
DL-IMPC	Intelligent	Deep Learning Inverse Predictive Control	$w_p = 10, w_c = 10$	This work

Wind Farm control simulation

The control simulations run for 60 seconds with $t_s = 0.02$ or 5 sample times per second. Random disturbances H_t and measurement noise ϵ_t are emulated at each time instant. The regulation point is the unity step, meaning the speed must be maintained within the interval. The performance and accuracy are validated with the assessment criteria metrics.

TABLE B.17: Experimental results summary

Experiment	**ITAE**	**IAE**	**MSE**	**Worst Control**	**Best Control**
DC Machine	12.484	12.5	0.045	PID	DIPC
Mass spring	11.548	11.5	0.012	PID	DIPC
Mass spring with Inverse Pole	13.842	10.98	0.048	MPC	DIPC
Cart Pole	15.484	15.81	0.135	MPC	DIPC
Microgrid	9.841	9.002	0.002	PID	DIPC
Lorenz Attractor	24.485	22.84	0.457	PID	DIPC
Robotic Manipulator	10.985	10.25	0.281	PID	DDPG
Wind Farm	NA	NA	NA	NA	DIPC

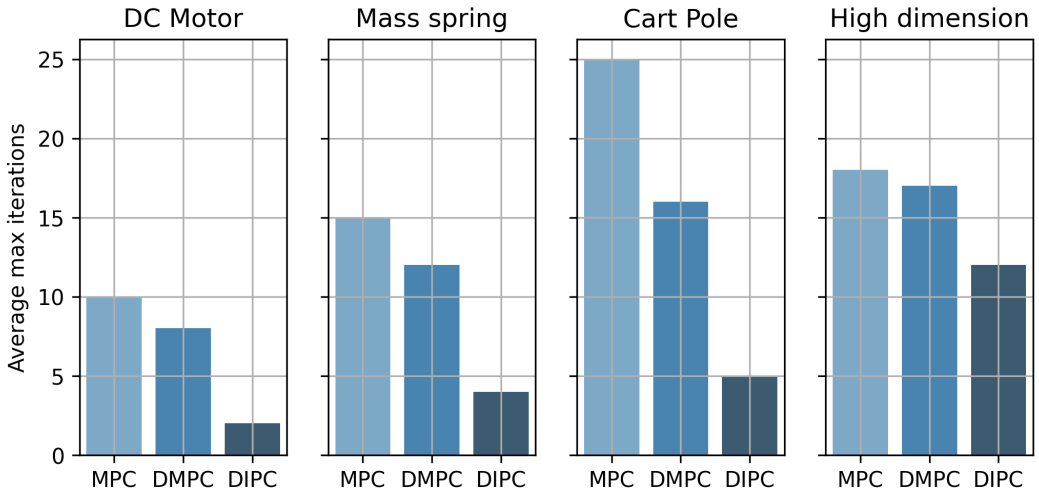


FIGURE B.1: Detailed performance comparison. The decrease in iterations utilized to find the optimal control sequence is shown for the tested nonlinear benchmarks (DC Machine, Mass Spring, Cart Pole, High dimension) using MPC, DMPC and DIPC.

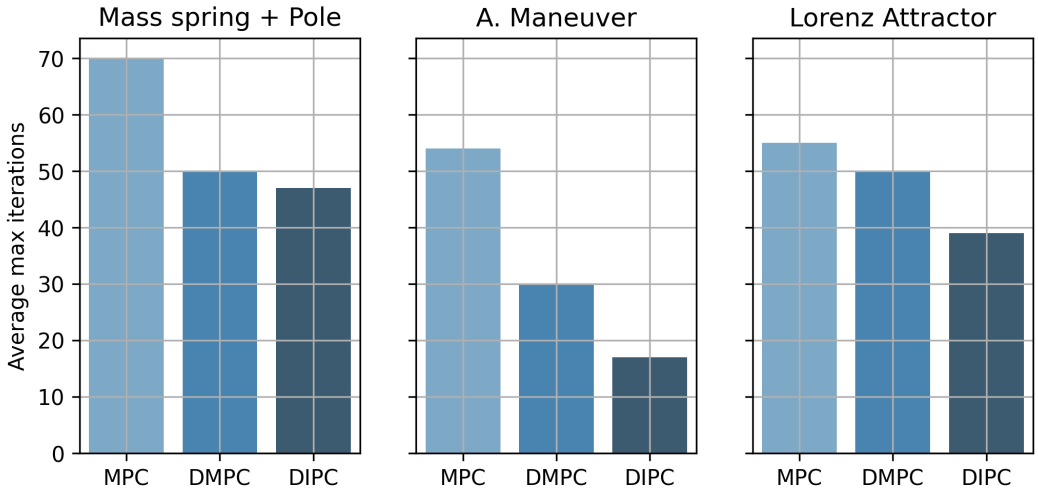


FIGURE B.2: Detailed performance comparison. The decrease in iterations utilized to find the optimal control sequence is shown for the tested nonlinear benchmarks (Mass Spring + Pole, A. Maneuver, Lorenz Attractor) using MPC, DMPC and DIPC.

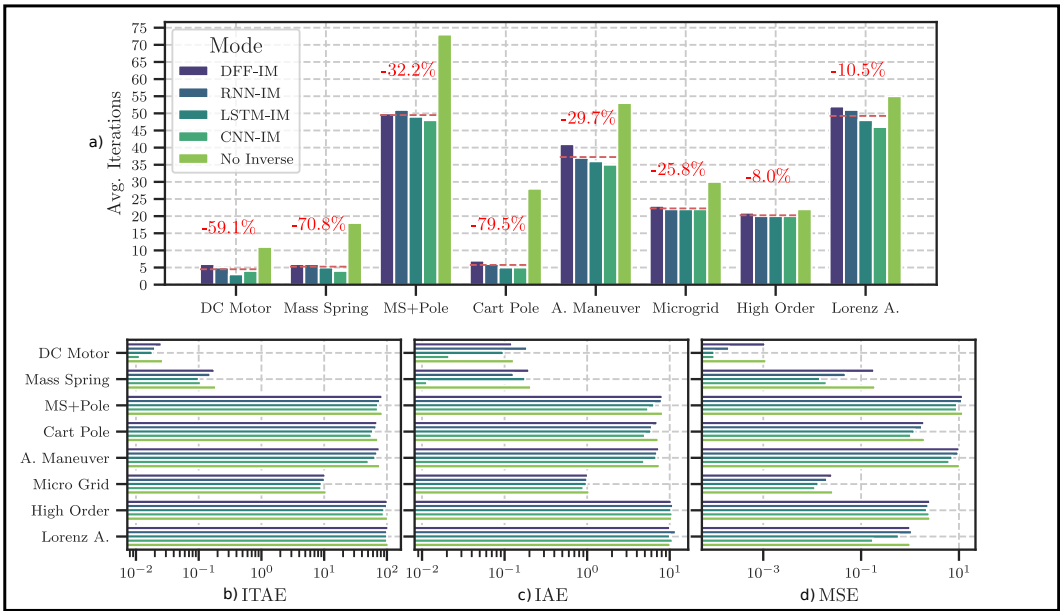


FIGURE B.3: Performance comparison, as presented in the publication [1]. The decrease observed in the average iterations used to compute a control signal is significant. Moreover, a slight increase in the response quality was achieved, as pointed by the assessment criteria considered. For more complex systems, the reduction was not as drastic as with simpler ones, yet considerable.

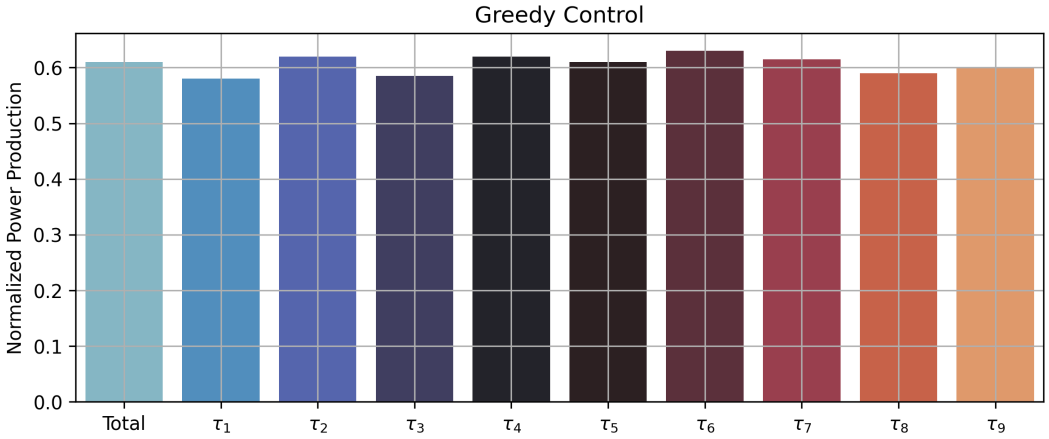


FIGURE B.4: Greedy yaw control results. The average power produced by each turbine element is shown. The first column, Total, represents the average production of the complete wind farm.

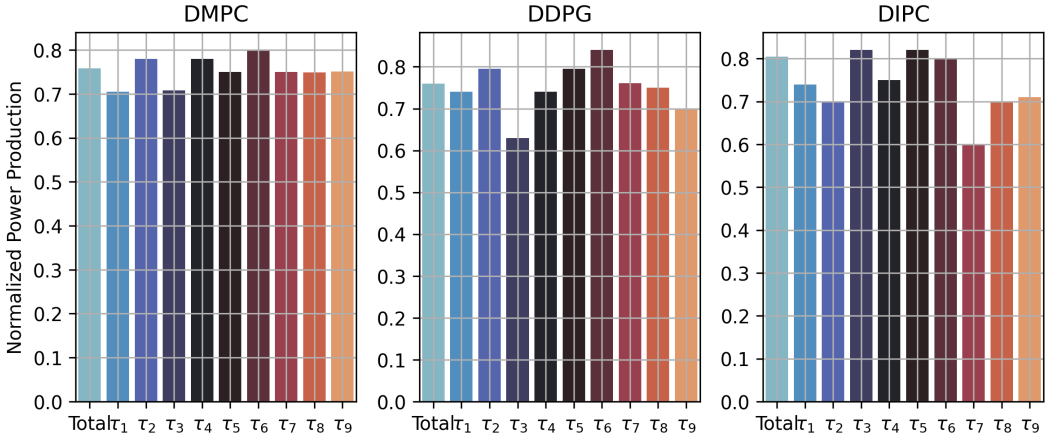


FIGURE B.5: Power output comparison between the used methods. The first column of each plot, Total, represents the average production of the complete wind farm. Note that DIPC generates more power in average against the other control algorithms.

References

[1] Wameedh Riyadh Abdul-Adheem and Ibraheem Kasim Ibraheem. From pid to nonlinear state error feedback controller. *International Journal of Advanced Computer Science and Applications*, 8(1):312–322, 2017.

[2] Frank Allgöwer, Thomas A Badgwell, Joe S Qin, James B Rawlings, and Steven J Wright. Nonlinear predictive control and moving horizon estimation an introductory overview. *Advances in control*, pages 391–449, 1999.

[3] Jennifer Annoni, Peter Seiler, Kathryn Johnson, Paul Fleming, and Pieter Gebraad. Evaluating wake models for wind farm control. In *2014 American Control Conference*, pages 2517–2523. IEEE, 2014.

[4] William Camilo Ariza-Zambrano and Alberto Luiz Serpa. Direct inverse control for active vibration suppression using artificial neural networks. *Journal of Vibration and Control*, 27(1-2):31–42, 2021.

[5] Karl J Åström and Björn Wittenmark. *Adaptive control*. Courier Corporation, 2013.

[6] Karl Johan Åström, Tore Hägglund, and Karl J Astrom. *Advanced PID control*, volume 461. ISA-The Instrumentation, Systems, and Automation Society Research Triangle ⋯, 2006.

[7] Dang Xuan Ba, Manh-Son Tran, Van-Phong Vu, Vi-Do Tran, Minh-Due Tran, Nguyen Trong Tai, and Cong-Doan Truong. A neural-network-based nonlinear controller for robot manipulators with gain-learning ability and output constraints. In *2021 International Symposium on Electrical and Electronics Engineering (ISEE)*, pages 149–153. IEEE, 2021.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Mahmuda Begum, Li Li, Jianguo Zhu, and Zhen Li. State-space modeling and stability analysis for microgrids with distributed secondary control. In *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*, pages 1201–1206. IEEE, 2018.

[10] Mouhacine Benosman. Model-based vs data-driven adaptive control: an overview. *International Journal of Adaptive Control and Signal Processing*, 32(5):753–776, 2018.

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[12] Shalabh Bhatnagar, Richard S Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[13] Stefano Bifaretti, S Pipolo, Alessandro Lidozzi, Luca Solero, L Tarisciotti, and Pericle Zanchetta. Predictive control for active split dc-bus 4-leg inverters. In *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 1–6. IEEE, 2016.

[14] Frank Boeren, Tom Oomen, and Maarten Steinbuch. Iterative motion feedforward tuning: A data-driven approach based on instrumental variable identification. *Control Engineering Practice*, 37:11–19, 2015.

[15] Stefan Braun. Lstm benchmarks for deep learning frameworks. *arXiv preprint arXiv:1806.01818*, 2018.

[16] S-P Breton, J Sumner, Jens Nørkær Sørensen, Kurt Schaldemose Hansen, Sasan Sarmast, and Stefan Ivanell. A survey of modelling methods for high-fidelity wind farm simulations using large eddy simulation. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2091):20160097, 2017.

[17] Robert T Bu and Vincent T Coppola. Benchmark problem for nonlinear control design: Problem statement, experimental testbed, and passive nonlinear compensation.

[18] James J Buckley. Theory of the fuzzy controller: A brief survey. *Cybernetics and applied systems*, pages 293–307, 2018.

[19] Robert T Bupp, Dennis S Bernstein, and Vincent T Coppola. Vibration suppression of multi-modal translational motion using a rotational actuator. In *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, volume 4, pages 4030–4034. IEEE, 1994.

[20] Robert T Bupp, Dennis S Bernstein, and Vincent T Coppola. A benchmark problem for nonlinear control design. 1998.

[21] Jeffrey A Butterworth, Lucy Y Pao, and Daniel Y Abramovitch. Analysis and comparison of three discrete-time feedforward model-inverse control techniques for nonminimum-phase systems. *Mechatronics*, 22(5):577–587, 2012.

[22] S Di Cairano, Alberto Bemporad, Ilya V Kolmanovsky, and Davor Hrovat. Model predictive control of magnetically actuated mass spring dampers for automotive applications. *International Journal of Control*, 80(11):1701–1716, 2007.

[23] Yongcan Cao and Wei Ren. Optimal linear-consensus algorithms: An lqr perspective. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(3):819–830, 2009.

[24] VV Chalam. *Adaptive control systems: Techniques and applications*. Routledge, 2017.

[25] Antonio Crespo, J Hernandez, and Sten Frandsen. Survey of modelling methods for wind turbine wakes and wind farms. *Wind Energy: An International Journal for Progress and Applications in Wind Power Conversion Technology*, 2(1):1–24, 1999.

[26] Saptarshi Das, Anish Acharya, and Indranil Pan. Simulation studies on the design of optimum pid controllers to suppress chaotic oscillations in a family of lorenz-like multi-wing attractors. *Mathematics and Computers in Simulation*, 100:72–87, 2014.

[27] H Delavari, Reza Ghaderi, NA Ranjbar, S Hassan HosseinNia, and Shaher Momani. Adaptive fractional pid controller for robot manipulator. *arXiv preprint arXiv:1206.2027*, 2012.

[28] Hongyang Dong, Jincheng Zhang, and Xiaowei Zhao. Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations. *Applied Energy*, 292:116928, 2021.

[29] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *Conference on Robot Learning*, pages 133–142. PMLR, 2017.

[30] Ahmed El-Sherbiny, Mostafa A Elhosseini, and Amira Y Haikal. A comparative study of soft computing methods to solve inverse kinematics problem. *Ain Shams Engineering Journal*, 9(4):2535–2548, 2018.

[31] Douglas F Elliott. *Handbook of digital signal processing: engineering applications*. Elsevier, 2013.

[32] Rolf Findeisen, Frank Allgöwer, and Lorenz T Biegler. *Assessment and future directions of nonlinear model predictive control*, volume 358. Springer, 2007.

[33] Edgar Galván and Peter Mooney. Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence*, 2021.

[34] Morgan T Gillespie, Charles M Best, Eric C Townsend, David Wingate, and Marc D Killpack. Learning nonlinear dynamic models of soft robots for model predictive control with neural networks. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 39–45. IEEE, 2018.

[35] Fouad Giri and Er-Wei Bai. *Block-oriented nonlinear system identification*, volume 1. Springer, 2010.

[36] Jesús Gonzalez and Wen Yu. Non-linear system modeling using lstm neural networks. *IFAC-PapersOnLine*, 51(13):485–489, 2018.

[37] Francisco González-Longatt, P Wall, and V Terzija. Wake effect in wind farm performance: Steady-state and dynamic behavior. *Renewable Energy*, 39(1):329–338, 2012.

[38] Longxiang Guo and Yunyi Jia. Inverse model predictive control (impc) based modeling and prediction of human-driven vehicles in mixed traffic. *IEEE Transactions on Intelligent Vehicles*, 2020.

[39] Adrián Hernández and José M Amigó. Attention mechanisms and their applications to complex systems. *Entropy*, 23(3):283, 2021.

[40] Lukas Hewing, Kim P Wabersich, Marcel Menner, and Melanie N Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:269–296, 2020.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Yunfeng Hu, Wanli Gu, Hui Zhang, and Hong Chen. Adaptive robust triple-step control for compensating cogging torque and model uncertainty in a dc motor. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(12):2396–2405, 2018.

[43] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[44] Junya Ikemoto and Toshimitsu Ushio. Application of deep reinforcement learning to networked control systems with uncertain network delays. *Nonlinear Theory and Its Applications, IEICE*, 11(4):480–500, 2020.

[45] PK Iyambo and Raynitchka Tzoneva. Transient stability analysis of the ieee 14-bus electric power system. In *AFRICON 2007*, pages 1–9. IEEE, 2007.

[46] Andrzej Jezierski, Jakub Mozaryn, and Damian Suski. A comparison of lqr and mpc control algorithms of an inverted pendulum. In *Polish Control Conference*, pages 65–76. Springer, 2017.

[47] Long Jin, Shuai Li, Jiguo Yu, and Jinbo He. Robot manipulator control using neural networks: A survey. *Neurocomputing*, 285:23–34, 2018.

[48] Ylva Jung and Martin Enqvist. Estimating models of inverse systems. In *52nd IEEE Conference on Decision and Control*, pages 7143–7148. IEEE, 2013.

[49] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.

[50] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[51] Mohammad Khairudin, Zaharuddin Mohamed, and Abdul Rashid Husain. Dynamic model and robust control of flexible link robot manipulator. *Telkomnika*, 9(2):279–286, 2011.

[52] Seong-Jae Kim, Hyun-Soo Kim, and Dong-Joong Kang. Vibration control of a vehicle active suspension system using a ddpg algorithm. In *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, pages 1654–1656. IEEE, 2018.

[53] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[54] Andreas Kroll and Horst Schulte. Benchmark problems for nonlinear system identification and control using soft computing methods: Need and overview. *Applied Soft Computing*, 25:496–513, 2014.

[55] Andrew Kusiak and Zhe Song. Design of wind farm layout for maximum wind energy capture. *Renewable energy*, 35(3):685–694, 2010.

[56] Robert H Lasseter and Paolo Paigi. Microgrid: A conceptual solution. In *2004 IEEE 35th Annual Power Electronics Specialists Conference (IEEE Cat. No. 04CH37551)*, volume 6, pages 4285–4290. IEEE, 2004.

[57] Frank L Lewis, Draguna Vrabie, and Kyriakos G Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105, 2012.

[58] Tie Li, Junyou Yang, and Dai Cui. Artificial-intelligence-based algorithms in multi-access edge computing for the performance optimization control of a benchmark microgrid. *Physical Communication*, 44:101240, 2021.

[59] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[60] Lennart Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

[61] Lennart Ljung. Perspectives on system identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

[62] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[63] Sergio Lucia and Benjamin Karg. A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20):511–516, 2018.

[64] Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

[65] Mohamed Massaoudi, Ines Chihi, Lilia Sidhom, Mohamed Trabelsi, Shady S Refaat, Haitham Abu-Rub, and Fakhreddine S Oueslati. An effective hybrid narx-lstm model for point and interval pv power forecasting. *IEEE Access*, 9:36571–36588, 2021.

[66] Daniele Masti and Alberto Bemporad. Learning nonlinear state–space models using autoencoders. *Automatica*, 129:109666, 2021.

[67] Mathworks. MATHWORKS robotics manipulator algorithms, year = 2021, url = https://github.com/mathworksrobotics/designingrobotmanipulator-algorithms, urldate = 2021-11-30.

[68] John Mckernan, James F Whidborne, and George Papadakis. Minimisation of transient perturbation growth in linearised lorenz equations. *IFAC Proceedings Volumes*, 38(1):342–347, 2005.

[69] Samir Mehta and John Chiasson. Nonlinear control of a series dc motor: theory and experiment. *IEEE Transactions on industrial electronics*, 45(1):134–141, 1998.

[70] JM Mendel and RW McLaren. 8 reinforcement-learning control and pattern recognition systems. In *Mathematics in science and engineering*, volume 66, pages 287–318. Elsevier, 1970.

[71] J-P Merlet. Direct kinematics of parallel manipulators. *IEEE transactions on robotics and automation*, 9(6):842–846, 1993.

[72] W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.

[73] Shozo Mori, Hiroyoshi Nishihara, and Katsuhisa Furuta. Control of unstable mechanical system control of pendulum. *International Journal of Control*, 23(5):673–692, 1976.

[74] Anh-Tu Nguyen, Tadanari Taniguchi, Luka Eciolaza, Victor Campos, Reinaldo Palhares, and Michio Sugeno. Fuzzy control systems: Past, present and future. *IEEE Computational Intelligence Magazine*, 14(1):56–68, 2019.

[75] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[76] Nasimul Noman and Hitoshi Iba. Enhancing differential evolution performance with local search for high dimensional function optimization. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 967–974, 2005.

[77] Robert D Nowak. Nonlinear system identification. *Circuits, Systems and Signal Processing*, 21(1):109–122, 2002.

[78] NREL. Floris. version 2.4, 2021.

[79] Olalekan Ogunmolu, Xuejun Gu, Steve Jiang, and Nicholas Gans. Nonlinear systems identification using deep dynamic neural networks. *arXiv preprint arXiv:1610.01439*, 2016.

[80] Samuel E Otto and Clarence W Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[81] Jyoti Ranjan Pati. *Modeling, identification and control of cart-pole system*. PhD thesis, 2014.

[82] RK Pearson. Selecting nonlinear model structures for computer control. *Journal of process control*, 13(1):1–26, 2003.

[83] Kenneth V Price. Differential evolution. In *Handbook of optimization*, pages 187–214. Springer, 2013.

[84] Juha Pyrhonen, Tapani Jokinen, and Valeria Hrabovcova. *Design of rotating electrical machines*. John Wiley & Sons, 2013.

[85] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[86] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[87] Richard H Rand, RJ Kinsey, and D Lewis Mingori. Dynamics of spinup through resonance. *International Journal of Non-Linear Mechanics*, 27(3):489–502, 1992.

[88] James B Rawlings. Tutorial overview of model predictive control. *IEEE control systems magazine*, 20(3):38–52, 2000.

[89] Hendrik Richter. Controlling the lorenz system: combining global and local schemes. *Chaos, Solitons & Fractals*, 12(13):2375–2380, 2001.

[90] ROBOTIS. ROBOTIS Open hardware manipulator, year = 2021, url = https://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/, urldate = 2021-11-30.

[91] HG Sage, MF De Mathelin, and E Ostertag. Robust control of robot manipulators: a survey. *International Journal of control*, 72(16):1498–1522, 1999.

[92] Yahaya Md Sam, Mohd Ruddin Hj Ab Ghani, and Nasarudin Ahmad. Lqr controller for active car suspension. In *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No. 00CH37119)*, volume 1, pages 441–444. IEEE, 2000.

[93] Johan Schoukens and Lennart Ljung. Nonlinear system identification: A user-oriented road map. *IEEE Control Systems Magazine*, 39(6):28–99, 2019.

[94] WC Schultz and VC Rideout. Control system performance measures: Past, present, and future. *IRE transactions on automatic control*, (1):22–35, 1961.

[95] Kaushik Das Sharma, Amitava Chatterjee, and Anjan Rakshit. Intelligent adaptive fuzzy control. In *Intelligent Control*, pages 3–21. Springer, 2018.

[96] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[97] Chao Song, Genaro Islas, and Klaus Schilling. Inverse dynamics based model predictive control for spacecraft rapid attitude maneuver. *IFAC-PapersOnLine*, 52(12):111–116, 2019.

[98] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[99] Lorenzo Stella, Andreas Themelis, Pantelis Sopasakis, and Panagiotis Patrinos. A simple and efficient algorithm for nonlinear model predictive control. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1939–1944. IEEE, 2017.

[100] Marios Stogiannos, Alex Alexandridis, and Haralambos Sarimveis. Model predictive control for systems with fast dynamics using inverse neural models. *ISA transactions*, 72:161–177, 2018.

[101] Andrew M Stuart. Numerical analysis of dynamical systems. *Acta numerica*, 3:467–572, 1994.

[102] Bidyadhar Subudhi and Debashisha Jena. Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification. *Neural Processing Letters*, 27(3):285–296, 2008.

[103] Michio Sugeno. On stability of fuzzy systems expressed by fuzzy rules with singleton consequents. *IEEE Transactions on Fuzzy systems*, 7(2):201–224, 1999.

[104] Arief Syaichu-Rohman and Raphael Sirius. Model predictive control implementation on a programmable logic controller for dc motor speed control. In *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, pages 1–4. IEEE, 2011.

[105] Sreenivas Tejomurtula and Subhash Kak. Inverse kinematics in robotics using neural networks. *Information sciences*, 116(2-4):147–164, 1999.

[106] Arne Traue, Gerrit Book, Wilhelm Kirchgässner, and Oliver Wallscheid. Toward a reinforcement learning environment toolbox for intelligent electric motor control. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[107] Carlos E Ugalde-Loo, Enrique Acha, and Eduardo Licéaga-Castro. Multi-machine power system state-space modelling for small-signal stability assessments. *Applied Mathematical Modelling*, 37(24):10141–10161, 2013.

[108] Jurgen van Zundert, Joost Bolder, Sjirk Koekebakker, and Tom Oomen. Resource-efficient ilc for lti/ltv systems through lq tracking and stable inversion: Enabling large feedforward tasks on a position-dependent printer. *Mechatronics*, 38:76–90, 2016.

[109] Jurgen van Zundert and Tom Oomen. On inversion-based approaches for feedforward and ilc. *Mechatronics*, 50:282–291, 2018.

[110] Gustavo A Vargas-Hákim, Efrén Mezura-Montes, and Héctor-Gabriel Acosta-Mesa. A review on convolutional neural networks encodings for neuroevolution. *IEEE Transactions on Evolutionary Computation*, 2021.

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[112] Yannic Vaupel, Nils C Hamacher, Adrian Caspari, Adel Mhamdi, Ioannis G Kevrekidis, and Alexander Mitsos. Accelerating nonlinear model predictive control through machine learning. *Journal of Process Control*, 92:261–270, 2020.

[113] Chih-Jian Wan, Dennis S Bernstein, and Vincent T Coppola. Global stabilization of the oscillating eccentric rotor. *Nonlinear dynamics*, 10(1):49–62, 1996.

[114] Chen Wang, Liming Wang, Libao Shi, and Yixin Ni. A survey on wind power technologies in power systems. In *2007 IEEE Power Engineering Society General Meeting*, pages 1–6. IEEE, 2007.

[115] Chengshan Wang, Yang Mi, Yang Fu, and Peng Wang. Frequency control of an isolated micro-grid using double sliding mode controllers and disturbance observer. *IEEE Transactions on Smart Grid*, 9(2):923–930, 2016.

[116] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267–278, 2009.

[117] Yu Wang. A new concept using lstm neural networks for dynamic system identification. In *2017 American Control Conference (ACC)*, pages 5324–5329. IEEE, 2017.

[118] Yu Wang. A new concept using lstm neural networks for dynamic system identification. In *2017 American Control Conference (ACC)*, pages 5324–5329. IEEE, 2017.

[119] Tigo Wati et al. Simulation model of speed control dc motor using fractional order pid controller. In *Journal of Physics: Conference Series*, volume 1444, page 012022. IOP Publishing, 2020.

[120] Paul J Werbos. Neural networks for control and system identification. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 260–265. IEEE, 1989.

[121] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[122] Bernard Widrow. Adaptive inverse control. In *Adaptive Systems in Control and Signal Processing 1986*, pages 1–5. Elsevier, 1987.

[123] Bernard Widrow. Adaptive inverse control. In *Applications of Artificial Neural Networks*, volume 1294, pages 13–21. International Society for Optics and Photonics, 1990.

[124] Andrew Wynn, Milan Vukov, and Moritz Diehl. Convergence guarantees for moving horizon estimation based on the real-time iteration scheme. *IEEE Transactions on Automatic Control*, 59(8):2215–2221, 2014.

[125] Lei Yazhou. Studies on wind farm integration into power system [j]. *Automation of Electric Power Systems*, 8(17):12–17, 2003.

[126] Yun-Hua Yin, Bin Zheng, and Hao-Xin Zheng. A method for modeling and simulation of brushless dc motor control system based on matlab [j]. *Journal of System Simulation*, 2(008), 2008.

[127] Ziying Yu, Xing Zheng, and Qingwei Ma. Study on actuator line modeling of two nrel 5-mw wind turbine wakes. *Applied Sciences*, 8(3):434, 2018.

[128] Han Zhang, Shen Zhao, and Zhiqiang Gao. An active disturbance rejection control solution for the two-mass-spring benchmark problem. In *2016 American Control Conference (ACC)*, pages 1566–1571. IEEE, 2016.

[129] Jincheng Zhang and Xiaowei Zhao. A novel dynamic wind farm wake model based on deep learning. *Applied Energy*, 277:115552, 2020.

[130] Yinyan Zhang, Shuai Li, and Liefa Liao. Near-optimal control of nonlinear dynamical systems: A brief survey. *Annual Reviews in Control*, 47:71–80, 2019.

[131] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

Publications

Publications related to the thesis

Journal paper

[1] Morales Perez Edgar Ademir and Iba Hitoshi. Deep learning-based Inverse modeling for Predictive Control. *IEEE Control Systems Letters.* vol. 6, pp. 956-961, 2022.

International conference

[2] Morales Perez Edgar Ademir and Iba Hitoshi. Inverse Model Optimization by Differential Evolution to improve Neural Predictive Control. *IEEE, Joint 11th International Conference on Soft Computing and Intelligent Systems and 21st International Symposium on Advanced Intelligent Systems (SCIS-ISIS)* Proceedings. pp. 1-8, Tokyo, Japan, December, 2020.

[3] Morales Perez Edgar Ademir and Iba Hitoshi. LSTM Neural Network-based Predictive Control for a Robotic Manipulator. *ACM, International Symposium on Electrical, Electronics and Information Engineering* Proceedings. pp. 128-135. Seoul, Republic of Korea. February, 2021.

[4] Rikitaka Kinoyama, Edgar Ademir Morales Perez and Hitoshi Iba. Preventing Overfitting of LSTMs using Ant Colony Optimization. *10th International Congress on Advanced Applied Informatics 2021* Proceedings. pp. 341-350. Tokyo, Japan. July 2021.

[5] Morales Perez Edgar Ademir and Iba Hitoshi. Deep learning-based Inverse modeling for Predictive Control. *IEEE 60th Conference on Decision and Control (CDC 2021)* Proceedings. ThA02.4. Texas, USA. December 2021.

[6] Morales Perez Edgar Ademir and Iba Hitoshi. Evolutionary Optimization of Multi-step Dynamic Systems Learning. *IEEE 8th International Conference on Mechatronics and Robotics Engineering (ICMRE 2022)* Proceedings. Munich, Germany. February 2022.