MASTER'S DISSERTATION

Nonlinear Moving-horizon State Estimation for Hardware implementation and a Model Predictive Control Application

Rafael Koji Vatanabe Brunello

Brasília, March de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

MASTER'S DISSERTATION

Nonlinear Moving-horizon State Estimation for Hardware implementation and a Model Predictive Control Application

Rafael Koji Vatanabe Brunello

Master's Dissertation submetida ao Departamento de Engenharia Mecânica como requisito parcial para obtenção do grau de Master of Mechatronics Systems

Banca Examinadora

Prof. Dr. Carlos Humberto Llanos Quintero _____
Orientador

Prof. Dr. Helon Vicente Hultmann Ayala _____
Co-orientador

Prof. Dr. Adriano Todorovic Fabro _____
Examinador externo

Prof. Dr. Renato Coral Sampaio _____
Examinador externo

FICHA CATALOGRÁFICA

BRUNELLO, RAFAEL KOJI VATANABE
Nonlinear Moving-horizon State Estimationfor Hardware implementation and a Model Predictive Control Application [Distrito Federal] 2021.
xvi, 69 p., 210 x 297 mm (ENM/FT/UnB, Master, Engenharia Mecânica, 2021).
Master's Dissertation - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Mecânica

1. Moving-Horizon State Estimation	2. Embedded systems
3. Neural Networks	4. Model Predictive Control
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BRUNELLO, R.K.V. (2021). *Nonlinear Moving-horizon State Estimationfor Hardware implementation and a Model Predictive Control Application*. Master's Dissertation, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 69 p.

Rafael Koji Vatanabe Brunello
Depto. de Engenharia Mecânica (ENM) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Agradecimentos

Firstly, I would like to show my gratitude and appreciation to all the people that helped me all the way here, even those that I will not name here directly. From my research environment: my master's advisors, prof. Carlos Llanos and Helon Ayala, our colleagues from LEIA, Carlos Eduardo and Renato Sampaio, for their guidance and enlightenment in helping me find my way through this dissertation and for the opportunity. To my personal environment, my parents and my girlfriend Andressa, those without their emotional support and encouragement I would not be able to finish this work. I would also like to thank my friends, in special Edrysson, who helped me think and develop many of this work's ideas and solutions. Also would like to the National Council of Scientific and Technologic Development of Brazil - CNPq for its financial support of this work. Lastly, I would like to acknowledge and praise myself for enduring the depression and anxiety, and being able to see this work to the end. Thank you all so much.

Rafael Koji Vatanabe Brunello

RESUMO

Nesta dissertação, exploramos a aplicação de redes neurais artificiais de funções de base radial (RBFs) embutidas em hardware para estimação de estados e controle em tempo real utilizando os algoritmos de Moving-Horizon Estimation(MHE) e Model Predictive Control (MPC). Esses algoritmos foram posteriormente aproximados por RBFs e implementados em um Field Programmable Gate Array (FPGA), que tem mostrado bons resultados em termos de precisão e tempo computacional. Mostramos que a estimativa de estado usando a versão aproximada do MHE pode ser executada usando um kit em escala de laboratório de aproximadamente 500 kHz para um pêndulo invertido a uma taxa de clock de cerca de 110 MHz. A latência para fornecer uma estimativa pode ser reduzida ainda mais quando FPGAs com clocks mais altos são usados, pois a arquitetura da rede neural artificial é inerentemente paralela. Após uma inspeção mais detalhada, descobriu-se que era possível reduzir o custo da área de chip trocando a função de custo por uma com resultados mais facilmente representáveis. Ele poderia então utilizar uma representação em 32 bits e o módulo CORDIC poderia ser removido, usando apenas a aproximação mais simples da série de Taylor de 2ª ordem. Em seguida, expandimos isso, investigando a ideia de usar uma única rede neural para substituir tanto o controle quanto o estimatidor de estados. Comparado a um MPC com informações completas, sua versão utilizando o MHE não teve um bom desempenho contra ruídos de saída. A princípio não foi possível aproximar o controle e a estimativa do pêndulo com um bom resultado, porém ao separar o controle em duas partes obtivemos melhores resultados. Por fim, verificamos que tal rede neural foi capaz de estabilizar o sistema de pêndulo invertido, mas não de aproximar sua parte oscilante não linear. A solução aqui apresentada é encorajada a ser estendida para sistemas mais complexos e não lineares, uma vez que uma arquitetura com complexidade razoável é encontrada para a rede neural artificial para ser implementada.

Palavras-chave:Estimação e filtragem, Moving-horizon State Estimator, Nonlinear model predictive control, Redes neurais, Arquiteturas de computador embarcadas, Radial-basis function, FPGAs

ABSTRACT

In this dissertation, we explore the application of radial basis functions (RBFs) artificial neural networks embedded in hardware for real-time estimation and control algorithms as the Moving-Horizon Estimation (MHE) and the Model Predictive Control (MPC). These algorithms are then approximated using RBFs and implemented in a Field Programmable Gate Array (FPGA), which has shown good results in terms of accuracy and computational time. We show that the state estimate using the approximate version of the MHE can be run using a laboratory-scale kit of approximately 500 kHz for an inverted pendulum at a clock rate of about 110 MHz. The latency to provide an estimate can be further reduced when FPGAs with higher clocks are used as the artificial neural network architecture is inherently parallel. Upon further inspection, it was found to be possible to reduce the chip area cost by switching the cost function for one with more easily representable results. It could then utilize a 32-bits representation and the CORDIC module could be removed, using instead only the simpler 2º order Taylor approximation. We then expand upon this, probing at the idea of using a single neural network to substitute both the control and state-estimation. Compared to a MPC with full information, its version utilizing the MHE did not perform well against output noises. At first, it was not possible to approximate the pendulum control and estimation with a good result, however when separating the control in two parts we gained better outcomes. Lastly, we verify that such a neural network was capable of stabilizing the inverted pendulum system, but not of approximating the non-linear swing-up part of it. The solution herein presented is encouraged to be further extended for more complex and nonlinear systems, given that an architecture is found for the artificial neural network with reasonable complexity to be implemented.

Keywords:Estimation and filtering, Moving-horizon State Estimator, Nonlinear model predictive control, Neural networks, Embedded computer architectures, Radial-basis function, FPGAs

Sumário

List of Figures

List of Tables

LIST OF SYMBOLS

Greek Symbols

Δ	Variation rate of a variable
ξ	Additive disturbance affecting the system dynamic states
η	Additive disturbance affecting the measured output
μ	Weight of the confidence about the measurements with respect to an initial estimate of the state $\overline{x}_0$ provided by the user
Ω	The closed set of admissible state estimates
ν	Number of neurons of the artificial neural network
λ	Artificial neural network predicted output
ϕ	The output of the hidden layer activation function for each neuron of the ANN
σ	The width of the radial basis functions ANN
θ	Angular position of the pendulum, being its resting position defined as π
γ	Coefficient of viscous friction in N.m.s

Adimensional Constants

e	Euler's constant
π	The ratio of a circle's circumference to its diameter

Undescripts

max	Maximum
min	Minimum
t	At time t
i, t	Estimation made at time t for the value at time i
$t - N, t$	Estimation made at time t for the value at the beginning of the horizon window
0	Initial prediction
ss	The value on the steady-state of the system
ref	The reference that we desire the system to follow

Overscripts

n, g, q	Set dimensionality
$\overline{x}$	State prediction, found by the iterative process using the system function
$\hat{x}$	State estimation, found by minimizing the cost function
J^o	Optimal minimization result of cost function J
ϵ	Positive scalar used in algorithm E^ϵ
$\hat{a}$	Approximate estimation function
$\tilde{x}$	Output of the ANN approximate state estimation
$\tilde{a}$	Approximate state estimation function using ANN
(j)	The j_{th} state estimate output
T	Matrix transpose
u^{opt}	The optimal solution for the control signal found by minimizing cost function J

Hardware Architectures

FPexp	Floating-point exponential unit that uses CORDIC and Taylor series
FPadd	Floating-point Addition/subtraction unit
FPCordic	Floating-point CORDIC unit
FPdiv	Floating-point Division unit
FPfrac	Floating-point Fractional part
FPmul	Floating-point Multiplication unit

Acronyms

ADMM	Alternating Direction Method of Multipliers
ALM	Adaptive Logic Module
AMHEC	Approximate moving-horizon estimation and control
ANN	Artificial Neural Networks
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Blocks
CORDIC	COordinate Rotation DIgital Computer
COVID	Corona-virus
CPU	Central Processing Unit
DSPs	Digital Signal Processing
EKF	Extended Kalman Filter
FPGA	*Field-Programmable Gate Array*
FSM	Finite State Machine
GPP	General Purpose Processors
GPU	Graphics Processing Unit
HIL	Hardware-In-the-Loop

HPC	High Performance Computing
HW/SW	Hardware/Software
IEEE	Institute of Electrical and Electronics Engineers
IPMSM	Interior Permanent Magnet Synchronous Machine
KF	Kalman Filter
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MH	Moving-Horizon
MHE	Moving-Horizon Estimation
MHEC	Moving-Horizon Estimation and Control
MHSE	Moving-Horizon State Estimation
MLP	Multi-Layer Perceptron
MPC	*Model Predictive Control*
MPSoC	System-on-Chip Multi-Processor
MSE	Mean Squared Error
MUX	Multiplexer
NMPC	*Nonlinear Model Predictive Control*
PLL	Phase-Locked Loop
PSO	Particle Swarm Optimization
R2	Statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in aregression model.
RAM	Random Access Memory
RBF	Radial-Basis Function
REG	Registers
RNA	Rede Neural Artificial
ROM	Read Only Memory
RTL	Register Transfer Level
SoC	*System-on-a-chip*
SVM	Support Vector Machine
SVR	Support Vector Regression
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*
WoS	Web of Science

1 INTRODUCTION

The state space approach has had a central role in modern control methods (Bennett 1996), due to, among other reasons, its generality in dealing with complexity of system orders and number of inputs. The state space representation was seminal for the inception of optimal control (Lewis, Vrabie e Syrmos 2012) and filtering (Kalman 1960) methods. Since then, however, control scientists and practitioners have felt that the inherent ability to handle complexity lead to computationally intensive control laws. Model predictive control strategies are regarded as one of the most important recent developments in the control field (Raković e Levine 2019). Frequently new methods in nonlinear optimal control are derived according to the model predictive control philosophy, and so we met no decrease in computational complexity for methods that ideally run in a deterministic clock frequency. Many methods have been devised in order to embed such control laws, and very practical tools for C code generation, for example, are now available with vast documentation and flexible GPL-3 software licenses (GNU General Public License-3) (Andersson et al. 2019).

On the other hand, state estimation has close relation with the classical regulation problem, as it is in fact mathematically the same (dual) problem (Kalman 1960, Bennett 1996). An analysis on the dual problem for the model predictive control is analysed and used in (Goodwin et al. 2005, Alessandri, Baglietto e Battistelli 2008). The moving-horizon state estimation (MHE) paradigm uses a sliding window of most recent measurements and a state prediction to infer the state estimates (Rawlings, Mayne e Diehl 2017). This is conceptually different than the Kalman original formulation, which uses solely one set of measurements made at a given time instant in order to provide an estimate of the mean and covariance of the states of the system. The advantages in terms of accuracy is generally recognized for moving-horizon approaches for estimation when compared to its Kalman-based counterparts, in spite of the greater computational effort (Haseltine e Rawlings 2005). In the receding-horizon approach, another name given to the moving-horizon, the algorithm solves an optimization problem at each sampling instant, what makes possible to take into account many measurements and explicitly the bounds in the states. Naturally this advanced state estimation algorithm demands considerable amount of computational resources to run in real-time. This hinders the application of advanced control and estimation methods to embedded solutions and systems that may operate at high frequency rates such as piezoelectric micromanipulators (Ayala et al. 2015), (Ayala, Rakotondrahe e Coelho 2018) and vibration mitigation (Zorić et al. 2019).

In (Alessandri, Baglietto e Battistelli 2008) the authors propose a moving-horizon estimation scheme for nonlinear state space systems with asymptotic convergence property. The results presented in this paper include an approximate version of the filter, where the idea is that the filter may be faced as a nonlinear function mapping from the observations to the state estimates. Artificial neural networks have also been employed to approximate Kalman-based

state estimation (Jazaeri e Nasrabadi 2015), or ad-hoc estimation mapping (Chen et al. 2018). In (Alessandri et al. 2011) the authors thoroughly compare the optimal and approximate version of the filter with artificial neural networks, showing overall better results when compared to the standard extended Kalman filter. In fact, optimal control problems may be solved offline and then estimated with universal approximators such as artificial neural networks in many useful applications (Zoppoli et al. 2020). In the literature we can find works that address the real-time implementation of advanced receding-horizon methods for estimation and control in embedded platforms.

In (Stellato, Geyer e Goulart 2017) the authors implement a linear model predictive controller on FPGA with four states and three inputs with approximate dynamical programming, reaching a sampling time of 25 microseconds. (Abdollahpouri, Takács e Rohaľ-Ilkiv 2017) runs a nonlinear moving-horizon estimation scheme similar to the one in (Alessandri, Baglietto e Battistelli 2008) on a Raspberry Pi in a vibration test-bench. To this end, the authors compiled C code and obtained a sampling time of 10 ms for a system with 5 states and weak multiplicative nonlinearities. The implementation on embedded hardware with ad-hoc architectures of the nonlinear receding-horizon filter proposed in (Alessandri, Baglietto e Battistelli 2008, Alessandri et al. 2011) is still lacking in the literature. Among the options, the radial basis function artificial neural network is an universal approximator and with important results in terms of embedded hardware deployment, as we review next.

In (Fan e Hwang 2013) the authors implement a fuzzy c-means and batch least squares algorithm to train and run radial basis functions artificial neural networks on FPGAs. (Chou et al. 2013) propose the use of the aforementioned neural network on hardware in order to control electrical motors. In (Souza e Fernandes 2014) the authors investigate the implementation with fixed-point precision calculations of a radial basis function artificial network on FPGAs and test it with the classical XOR problem. (Kim e Jung 2015) propose an FPGA implementation for the backpropagation algorithm to tune the parameters of a radial basis function artificial neural network. In (Kung, Than e Chuang 2018) the authors use a radial basis function artificial neural network in order to adaptively identify a system in order to perform gain scheduling online for a 2-degrees-of-freedom positioning table. For a review of most recent work on deep learning deployments on FPGAs and stacked auto-encoders case, see (Coutinho, Torquato e Fernandes 2019). In the present work, in turn, we provide a novel architecture for radial basis function artificial neural network when compared to a previous work (Ayala et al. 2017), by improving the number of calculations in the output of the network and also optimizing the hardware usage in the calculations of the neuron. As the calculations of the neurons and the whole network are the most important, we provide improvements in the aspect of hardware implementation of artificial neural networks. It is worth noting that our architecture uses floating-point representation, which offers greater dynamic-range than fixed-point representation, and therefore being suitable for control and robotics applications. To do that we have used the parameterized library proposed in (Muñoz et al. 2010), which allows the user to set the numerical representation size depending on the precision required by the application.

1.1 PROBLEM DEFINITION

The moving-horizon state estimator was contrived primarily in academia, as opposed to its dual problem the MPC (Allgöwer et al. 1999), it is also less popular and has not been researched and applied as much as the MPC. Both have similar complexity and computational cost problems, however, while the MPC is studied and developed both in academia and the industry, the MHE is mostly disregarded for the simpler and faster Kalman filter and its variations. Fortunately, most of the breakthroughs for the MPC are also applicable to the MHE.

Moreover, the ones that do research on the topic focus on what is considered its main challenging issues, improving its optimization algorithm, the design of its arrival cost function, and the performance analysis of the estimation (Zou et al. 2020). The latter two are mostly theoretical and academic problems, and while the former hit upon the application problems it narrows it down too much. The enhancement desired from the optimization algorithm is mainly related to its speed, or its inability to run fast enough on most industrial online systems. As such, the problem this work tries to solve is to attain a real-time implementation of the moving-horizon approach, boosting its speed while retaining its results.

1.2 MOTIVATION

This work strives to overcome some of the shortcomings of the Moving-horizon Estimation method (MHE), mainly its high computational cost and time inefficiency for real-time applications. It also tries to promote and encourage creative thinking to circumvent this kind of theorical difficulties using more practical approaches. In this way, a version of the MHE can be used to solve other types of problems that require state estimators. Various applications in the areas of control, automation, and robotics can benefit from this purpose.

In this direction, in LEIA (*Laboratory of Embedded System and Integrated Circuit Applications*) belonging to GRACO (UnB Automation and Control Group), several research lines can take advantage of the results of this work.

1.3 OBJECTIVES

1.3.1 General Objective

In the present work, we evaluate the real-time implementation of the approximate filter proposed in (Alessandri, Baglietto e Battistelli 2008, Alessandri et al. 2011) for the first time, to the best of our knowledge, in FPGAs using new architectures developed to implement an artificial neural network. To this end, we improved a previous implementation of a radial basis function artificial neural network made by the authors in (Ayala et al. 2017), by minimizing the use of resources for

calculating the output of each neuron of the artificial neural network and also adapting it to the multiple-output case needed for the moving-horizon state estimation.

In order to take one step further to a real application and use of the MHE, it was initially intended to utilize it on an inverted pendulum didactic benchmark. However, with the COVID-19 pandemic happening we decided it was best to focus on another possibility that could be worked on from home. So, as an alternative, we employ it together with a Model Predictive Control (MPC), taking advantage of a previous work developed in the Group (Sampaio 2018).

Afterward, the same steps for making an approximate ANN is made, but this time trying to approximate the results of both estimation and control as a single network.

1.3.2 Main Objectives

The main objectives are divided into two central development cycles. In the first one, referring to the implementation of the approximate moving-horizon estimator, we have:

- Develop the algorithm for the neural network moving-horizon estimator in the high-level programming language (Matlab) for validation of the model and case studies;
- Devise training and tunning procedures for the radial-basis function that approximates the MHE behavior;
- Adapt and generate the RBF implementation in FPGA on (Ayala et al. 2017) for the multiple outputs case scenario;
- Validate the system's performance in software and Hardware-in-the-loop (HIL) simulations.

In the second cycle, where we attempt to do the same with a combination of the moving-horizon control and estimator, we have:

- Produce a code that implements the model-predictive control (MPC) used in (Ayala et al. 2016) with the MHE in the high-level programming language (Matlab) for validation of the model and case studies;
- Design tunning procedures for the MPC parameters;
- Develop the algorithm for the neural network approximation of the MPC with MHE in the high-level programming language (Matlab) for validation of the model and case studies;
- Devise training and tunning procedures for the RBF that approximates the MPC with MHE behavior as one singular neural network;
- Generate the RBF implementation files for the AMHEC;
- Validate the system's performance in software and Hardware-in-the-loop (HIL) simulations.

1.4 METHODOLOGY SUMMARY

During the research of this dissertation, the methodology was as follow:

- Development of the NNMHE in software;
- Tuning of RBF parameters for the NNMHE;
- Improvement of RBF hardware implementation;
- Solving implementation precision problem and enhance area usage;
- Using MPC with MHE in software;
- Tuning MPC and MHE parameters for better noise robustness;
- Tuning of RBF for approximating the MHEC;
- Improving and solving MHEC issues;
- Tuning two different MPC controllers for each stage of the pendulum motion;
- Training the RBF approximation of both stages of the MHEC;
- Testing other neural networks architectures and an SVM using the NIOTS program from (Santos et al. 2017).

1.5 CONTRIBUTIONS

In summary, the contributions of the present work are:

1. The implementation of the moving-horizon state estimation scheme in (Alessandri, Baglietto e Battistelli 2008, Alessandri et al. 2011) using artificial neural networks and floating-point representation in FPGAs, showing that it is possible to obtain very small sampling times by employing this strategy using ad-hoc hardware implementations and leveraging the inherent parallel architecture of this model class;
2. The improvement of the hardware implementation of the radial basis function artificial neural network detailed in (Ayala et al. 2017) with respect to the use of resources and adaption to the multiple output case;
3. A publication at an international congress (IFAC 2020) (Brunello et al. 2020) which its contents were included in this document in Chapter 3;
4. The development of an RBF neural network approximation on a two-step MHEC for the inverted pendulum case study.

1.6 DISSERTATION PLAN

The remainder of the dissertation is organized thusly. The mathematical and theoretical basis needed to understand the algorithms, theories, and hardware used are presented in Chapter 2, togheter with a brief state of art review on the subject. Then, Chapter 3 describes the step-by-step procedure employed in the development of the RBFMHE with its results and improvements. In Chapter 4 we show, compare, and discuss the methodology and results of the approximate moving-horizon estimation and control. Lastly, Chapter 5 ends the dissertation with its conclusions and perspectives for future works.

2 METHODS

In this chapter, we explain and describe all the theoretical foundations needed to understand and analyze this dissertation, starting with the MHE formulation, its approximate version, offline design, applications and main challenges. Followed by the NMPC formulation summary and its application with some machine-learning-based methods. Next, we explain the RBF ANN architecture and how we will evaluate its results. Then, an explanation of the choice of hardware used its structure and current market state. Lastly, we present a simple state of art research done on embedded MPC and MHE, also its implementations with ANN.

2.1 MOVING-HORIZON STATE ESTIMATION

In the present work we are concerned with the state estimation problem for system of the type

$$x_{t+1} = f\left(x_t, u_t\right) + \xi_t, \tag{2.1a}$$

$$y_t = h\left(x_t\right) + \eta_t, \tag{2.1b}$$

where $t \in \mathbb{Z}_+ \triangleq \{0, 1, \ldots\}$ denotes the time sample, $x_t \in \mathbb{R}^n$ describes the continuous state of the system, $u_t \in \mathbb{R}^q$ is the control input and $y_t \in \mathbb{R}^g$ is the system measured output. The quantities $\xi_t \in \mathbb{R}^n$ and $\eta_t \in \mathbb{R}^g$ are additive disturbances affecting the system dynamic states and the measured output, respectively. The former may represent any uncertainty in modeling the system equations and the later eventual disturbances affecting the measurements. Functions $f : \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^g$ are in general nonlinear. We employ the moving-horizon state estimation algorithm as described in (Alessandri, Baglietto e Battistelli 2008). The moving-horizon approach for state estimation, also termed receding-horizon, comprehend a set of estimation algorithms in which the most distinctive feature is the use of a sliding window containing a set of most recent measurements. This is in contrast with the Kalman filter-based solutions, which solely use the most recent measurement and a state estimation propagation from one time instant to the next.

The filter in (Alessandri, Baglietto e Battistelli 2008) has asymptotic convergence guarantees, given mainly that the system is state observable according to an ad-hoc definition for the case of moving-horizon state estimation. Briefly speaking, the moving-horizon estimation paradigm uses the most recent amount of information available and it is assumed that a nonlinear state-space model is available. As the estimation problem is termed as an optimization problem to be solved online once new measurements become available, it is possible to explicitly take into account the constraints of the system variables.

Consider the cost function J given by

$$J = \mu \parallel \hat{x}_{t-N,t} - \bar{x}_{t-N} \parallel^2 + \sum_{i=t-N}^{t} \parallel y_i - h(\hat{x}_{i,t}) \parallel^2, \tag{2.2}$$

where $N+1$ is the size of the window considered for calculating J, $\bar{x}_t$ is the state prediction at time t, and $\hat{x}_{i,t}$ is the state estimation of the state at time i made at time t, with $N - t \leq i \leq t$. The variable μ is used to weight the confidence we have about the measurements with respect to an initial estimate of the state $\bar{x}_0$ provided by the user. When the cost function J is minimized having $\hat{x}_{t-N,t}$ as the decision variable, that is called the *state estimation function*

$$\begin{aligned} &\hat{x}_{t-N,t} = a(\overline{x}_{t-N}, I_t) = \arg\min J, \\ &\text{s.t. } \hat{x}_{i,t} \in \Omega, \end{aligned} \tag{2.3}$$

where Ω denotes the closed set of admissible state estimates, and $I_t = [y_{t-N}, \ldots, y_t, u_{t-N}, \ldots, u_t]^\intercal$ is the information vector with all the information within a sliding window of the output and input vectors; we may obtain a state estimate at the beginning of the moving-horizon window. Note that for sake of simplicity, when solving Eq. (2.3) we focus on obtaining the state estimate at $t - N$, that is, at the beginning of the moving-horizon window. Great simplification is achieved in this process by obtaining the rest of the state estimates within the window by recursively iterating the dynamic state equation as

$$\overline{x}_{i+1,t} = f\left(\hat{x}_{i,t}, u_i\right). \tag{2.4}$$

Summing up, as the Figure 2.1 shows, the moving-horizon state estimation algorithm adds up to solving Eq. (2.3) to obtain the state estimate at the beginning of the window, presented as a red dot. While the rest of the state estimates in this window and time are obtained by the iterative application of Eq. (2.4). Note that the bounds in the state estimates are taken into account naturally in this process. Lastly, Figure 2.2 presents the program flow chart of a typical MH estimation algorithm.

2.1.1 Approximate moving-horizon estimator

So that it is explicitly checked for a potential error in the minimization of equation 2.2, we define that, given a generic pair $(\overline{x}_{t-N}, I_t)$, the associated value with the exact minimization is given in Equation 2.5. Employing this definition the algorithm E^ϵ can then be stated.

$$J^o(\overline{x}_{t-N,t}, I_t) \triangleq min_{\hat{x}_{t-N,t}} J(\hat{x}_{t-N,t}, \overline{x}_{t-N}, I_t). \tag{2.5}$$

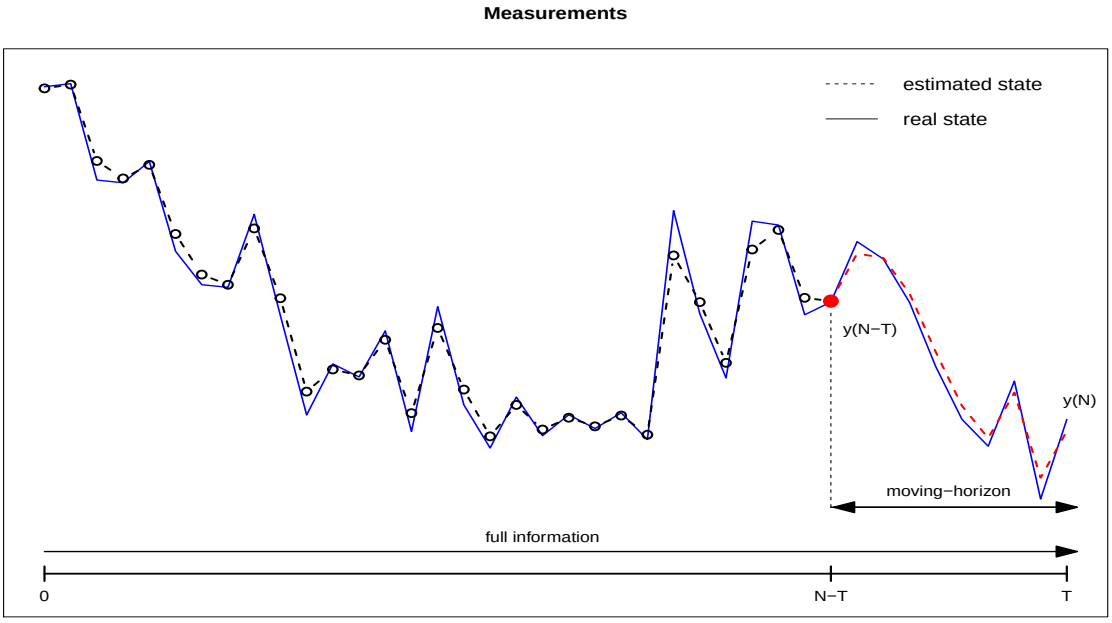


Figure 2.1: Moving-horizon estimation principle.

2.1.1.1 Algorithm E^{ϵ}

Given an a-priori prediction $\overline{x}_0^{\epsilon}$ and a positive scalar ϵ, at any time *t*, find an estimate $\hat{x}_{t_N,t}^{\epsilon}$ such that $\hat{x}_{t_N,t}^{\epsilon} \in \Omega$ and the prediction is propagated as:

$$J(\hat{x}_{t-N,t}^{\epsilon}, \overline{x}_{t-N}^{\epsilon}, I_t) - J^o(\overline{x}_{t-N}^{\epsilon}, I_t) \leq \epsilon, \tag{2.6a}$$

$$\overline{x}_{t-N+1}^{\epsilon} = f(\hat{x}_{t-N,t}^{\epsilon}, u_{t-N}). \tag{2.6b}$$

This algorithm calculates, at each instant *t*, a smoothed estimate $\hat{x}_{t_N,t}^{\epsilon}$ of the state at time *t-N*. But, in most feedback control applications, it is preferred to have the current state estimation at time *t*. This estimation is obtainable via the noise-free dynamics simulation, as done in Equation 2.4. However, this method may incur an erratic behavior of the estimate for high values of *N*. This issue is circumvented with many different approaches for online estimation, some of these are discussed in (Raff et al. 2005) and (Scala, Bitmead e James 1995). Though, as the focus of this dissertation is not on the online application it will not be further detailed here.

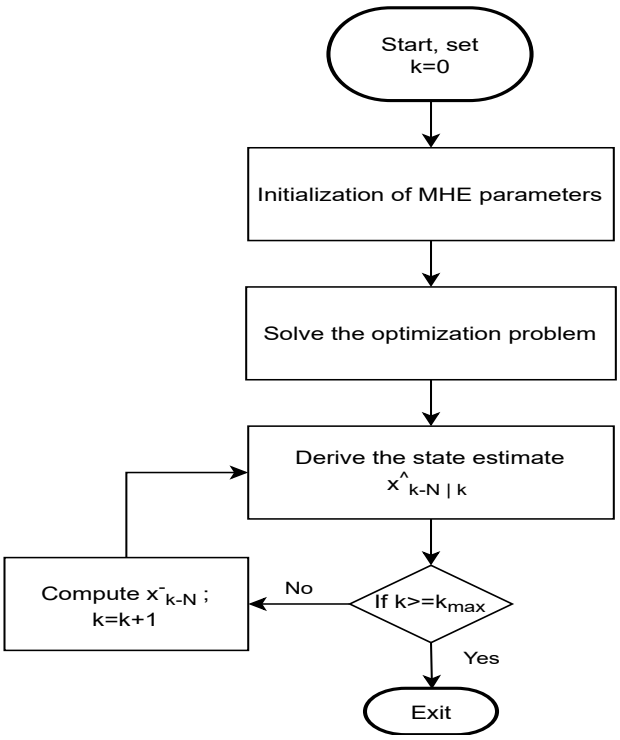


Figure 2.2: Moving-horizon estimation algorithm.

2.1.1.2 Offline design

In (Alessandri et al. 2011) an offline design of an approximate MHE was also developed for the solution of the minimization involved in the algorithm E^{ϵ} that is based on non-linear approximators. The method consists of constraining the Equation 2.3 to take a fixed form as:

$$\hat{a}(\bar{I}_t, w), t = N, N+1, ... \tag{2.7a}$$

$$\bar{I}_t \triangleq col(\overline{x}_{t-N}, I_t), \tag{2.7b}$$

where w is a parameter vector to be optimized offline in order to ensure that Equation 2.6 is always fulfilled. After w is determined, the approximate estimation function $\hat{a}(\bar{I}_t, w)$ can be employed online to obtain the state estimates with low computational effort. The focus in (Alessandri et al. 2011) was on approximating using a one hidden-layer feedforward neural network, in this work however we utilize the similar radial basis function artificial neural network. Many other choices are possible among fixed-structure non-linear approximators that benefit from density properties, some of these can be found in (Hornik, Stinchcombe e White 1989), (Leshno et al. 1993) and (Zoppoli, Sanguineti e Parisini 2002).

The inputs of the NN are the components of the moving-horizon prediction window $\overline{x}_{t-N}$ and the information vector I_t. And, the outputs $\tilde{x}^{(j)}_{t-N,t}$ correspond to the j_{th} state estimate in the beginning of the window. This means that the output vector has the form on Equation 2.8

$$\tilde{x}^{(j)}_{t-N,t} = \tilde{a}^{(j)}(\bar{I}_t, w) = \sum_{p=1}^{\nu} c_{pj} g(w_p^T \bar{I}_t + w_{0p}) + c_{0j}, \tag{2.8}$$

where ν is the number of neurons, *g(.)* is the activation function of the neuron, and the coefficients c and w are the components of the vector w. In the RBF neural network case, it can be linked to the centers and weights to be determined. It is also fundamental to note that the ν is sufficient to characterize the complexity of the network since the number of free parameters grows linearly with it.

$$w \triangleq col(c_{pj}, c_{0j}, w_p, w_{0p}; p = 1, 2, ..., \nu; j = 1, 2, ..., n). \tag{2.9}$$

Regrettably, employing a non-linear approximator cannot assure that the results will be within the system boundaries and defined constraints. This can be done by projecting its output to the desired set, the composition of the neural approximator and the projection operator is called *neural state estimation function*. However, applying the projection requires the solution of a minimization problem. To simplify the implementation this work applies only the incomplete version without the projection.

2.1.2 Applications

Due to the popularity of digital processing technologies, the MHE has been broadly utilized in practical applications. One of the most important applications of state estimation is fault detection and isolation for a dynamic system. Such as, robust fault detection (Tyler, Asano e Morari 2000), optimal dosing in chemotherapy (Chen, Kirkby e Jena 2012), estimation of states of a nanopositioner (Polóni et al. 2013) and a real-time fault-tolerant air data estimation scheme (Wan e Keviczky 2019). Other examples are, attitude estimation for navigation systems be it for spacecraft ((Huang, Zhao e Zhang 2017),(Qin e Chen 2013),(Vandersteen et al. 2013)), mobile robots (Liu et al. 2017) or humanoid ones (Bae e Oh 2017). It was also used in state of charge estimation technology for lithium batteries ((Shen et al. 2016),(Shen et al. 2019),(Hu, Cao e Egardt 2018)).

Another common application is parameter estimation (Kwon et al. 2015). From towed cables systems (Sun et al. 2015) to active cantilever beams (Abdollahpouri, Takács e Rohaľ-Ilkiv 2017) and enhancing robustness and decreasing sensibility of power systems (Chen 2017). It also has been employed combined with some sort of control, some examples are with the model predictive control for water level control (Segovia et al. 2019) and for blood glucose regulation (Copp, Gondhalekar e Hespanha 2018), in (Andersson e Thiringer 2018) it was combined with a motion sensor-less controller for an interior permanent magnet synchronous machine (IPMSM).

2.1.3 Main challenges

As (Zou et al. 2020), the most recent (up-to-date) overview paper on MHE, describes, there are three main challenging issues with respect to research topics on MH estimation. These problems are: how to design an optimization algorithm with good adaptability, design of the arrival cost for complex systems, and the performance analysis of the estimation. The first is the issue of designing the optimization algorithm to reduce the computational complexity while ensuring estimation performance. The second refers to the design of the MH estimator, it is of critical importance for the estimation scheme. It should be designed according to the system structure and parameter nature, the design of the MHE is also much more difficult compared to traditional estimators. Lastly, the parameter design of the MHE is always implemented based on the performance analysis, one of the most investigated issues in this is the stability and convergence properties. However, as the MHE is not written in an analytical form it leads to a difficult problem on deriving the estimation error dynamics.

2.2 MODEL PREDICTIVE CONTROL (MPC)

In this dissertation we are concerned with the utilization of the non-linear MPC employing a general representation of discrete time prediction model as the one presented in (Ayala et al. 2016),

$$x(k+1) = f(x(k), u(k)), y(k) = h(x(k)), \tag{2.10}$$

where $x(k) \in \mathbb{R}^n$ is the *n*-th state vector in the instant $k \in \mathbb{N}$, $u(k) \in \mathbb{R}^m$ is the input vector of size m and $y(k) \in \mathbb{R}^{n_y}$ is the output vector of size n_y. Moreover, the $f : \mathbb{R}^n \times \mathbb{R}^{m \to \mathbb{R}^n}$ function is the one that determines the vector state value on instant $k+1$, while h is the output state vector mapping function.

Now it is needed to determine the size of the prediction (N) and control (N_c) horizons. They define how many instants ahead will be predicted and controlled, respectively, by the MPC. These sizes affect the controller's capability to anticipate future events and optimize its control actions, however, it proportionally increases the computational cost making it slower and harder to implement in real-time applications. As such, it is usually employed a smallest horizon size that is capable of stabilizing the system within the desired control parameters.

With the prediction horizon, the current system state $x(k)$ and a set of input vectors $\tilde{u}(k)$ in Equation 2.11a, the one-step ahead prediction model presented in Equation 2.10 can be applied in an iterative manner to estimate the value of the future states ($\tilde{x}(k)$) and future outputs ($\tilde{y}(k)$).

$$\tilde{u}(k) = [u(k), u(k+1), ..., u(k+N-1)] \in \mathbb{R}^{N.m}, \tag{2.11a}$$

$$\tilde{x}(k) = [x(k+1), ..., x(k+N)] \in \mathbb{R}^{N.n}, \tag{2.11b}$$

$$\tilde{y}(k) = [y(k+1), ..., y(k+N)] \in \mathbb{R}^{N.n_y}. \tag{2.11c}$$

It must also be noted the constant N_c which defines the control window, this value must be less or equal to the prediction horizon ($N_c < N$). This means that the number of degrees of freedom to determine the sequence of control actions is limited to N_c and, if $N_c < N$, then the last value of the control signal in $u(N_c)$ is repeated for the rest of the horizon. Like the MHE, the MPC also allows a systematic inclusion of restrictions in its control law and controller design, this makes the controller safer, more predictable, and, by definition, that it will not pass its defined boundaries.

With Equations 2.11 it is then necessary to establish objective criteria for the controller performance, so that it can choose the optimal solution u^{opt} to be used. Therefore defining a cost function from a linear combination of these criteria to generate a scalar value to be minimized. The typically used criteria, including stability considerations from (Mayne et al. 2000) and (Rossiter 2013), are the position error ($e(k) = \tilde{y}(k) - y_{ref}(k)$), the input deviation ($\hat{u}(k) = u(k) - u_{ss}$ and the input variation rate ($\Delta u = u(k) - u(k-1)$. The first considers an estimate output vector in relation to a reference vector ($y_{ref}(k)$), while the second presents the difference between the current input and the one needed to keep the system stabilized in the steady-state (u_{ss}), the third being self-explanatory.

Equation 2.12 describes a possible cost function where a weighted sum of the squared indicators is used to obtain a convex function, which facilitates the calculation of a global minimum value. Where Q and R are diagonal weight matrices to influence the output.

$$J(x(k), u(k)) \triangleq \sum_{i=1}^{N} \| e(k) \|_Q^2 + \sum_{i=0}^{N_c} \| u(k) \|_R^2. \tag{2.12}$$

However, this cost function does not guarantee the system stability (Mayne et al. 2000). As such, one alternative is employing an additional term called *terminal cost*. It refers to a pondering specific to the state at the end of the prediction horizon ($x(k+N)$). In such case the cost function equation can be rewritten as Equation 2.13, where Q_f is another diagonal weight matrix and x_{ss} is the state value in its steady-state. From the minimization of this cost function, we can then determine the optimum output vector u^{opt} to be used in the system described in Equation 2.14.

$$J(x(k), u(k)) \triangleq \sum_{i=1}^{N-1} \| e(k) \|_Q^2 + \sum_{i=0}^{N_c-1} \| u(k) \|_R^2 + \| x(N) - x_{ss} \|_{Q_f}^2. \tag{2.13}$$

$$u^{opt} = min_{u_{k,k+N-1}} J(x(k), u(k, k+N-1)). \tag{2.14}$$

A graphical representation of the MPC method can be observed in Figure 2.3. The red curve is the reference trajectory that the controller wants to follow, the yellow and dark blue curves refer to the measured output and the past control input respectively. While the purple and the light blue curves are the predicted future outputs and control signal. it also illustrates that when the control

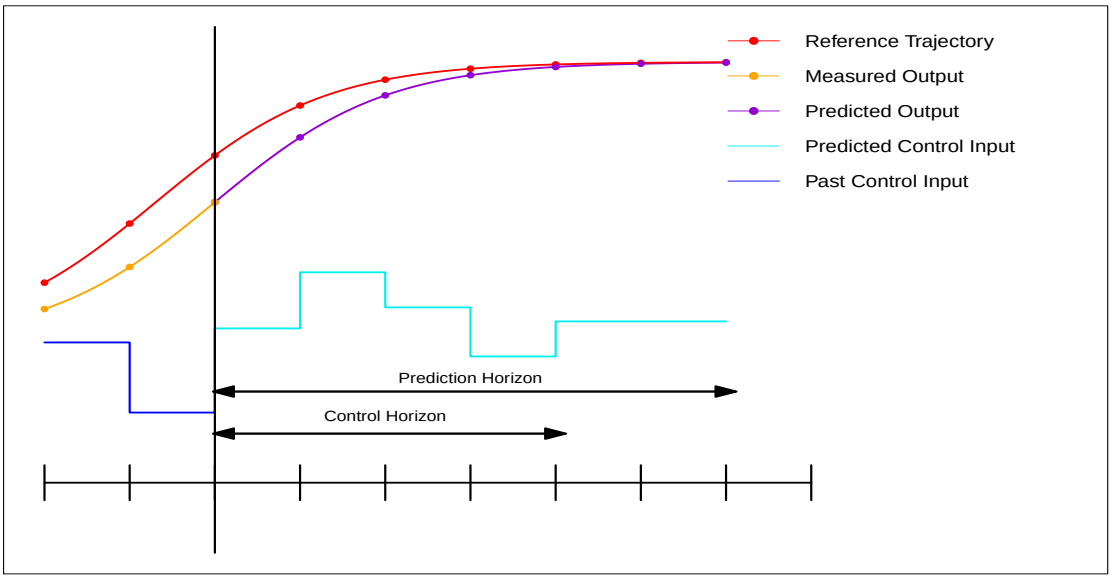


Figure 2.3: Model predictive control principle.

horizon is smaller than the prediction horizon the last predicted control input is maintained for the rest of the prediction horizon.

With this we can now define the steps of a general MPC as:

1. Measure or estimate the states $x(k)$ of the system;
2. Compute the optimal vector u^{opt} from the minimization of cost function **J**;
3. Use $u(k) = u^{opt}$ as the first term of the control sequence and apply it on the system;
4. Go to the next iteration ($k = k + 1$) and restart from step 1.

Lastly, to emulate the system in order to obtain its states it is necessary to solve non-linear differential equations. To do this we employ one of the most common numerical methods, the Runge-Kutta method that discretizes the continuous solution in function of the sampling time. Starting from a known initial value and the sampling time Δt it is possible to calculate an approximate estimative of the next state. To obtain better estimates it is possible to utilize higher orders, but in this dessertation, we are employing the 4_{th} order Runge-Kutta shown in Equation 2.15.

$$
\begin{aligned}
z_1 &= f(u(k), x(k)), \\
z_2 &= f(u(k), x(k) + 0,5\Delta t.z_1), \\
z_3 &= f(u(k), x(k) + 0,5\Delta t.z_2), \\
z_4 &= f(u(k), x(k) + \Delta t.z_3), \\
x(k+1) &= x(k) + \Delta t.(z_1 + 2z_2 + 2z_3 + z_4)/6.
\end{aligned} \tag{2.15}
$$

2.2.1 Machine learning based methods

One of the more promising approaches to implementing the MPC with high sampling frequencies (over 100Hz or even over 1kHz) is the machine learning-based methods, be it with artificial neural networks (ANNs) or support vector machines (SVMs).

Some of the first relevant papers on this area were made by Camacho and Ortega ((Ortega e Camacho 1996), (Gomez-Ortega e Camacho 1994)), approximating an NMPC used on trajectory control of a mobile robot with a multi-layer perceptron network, in this case, the whole network was trained offline. Two feedforwards ANN were developed in (Åkesson e Toivonen 2006) to approximate two different control systems, one of pH and the other of a liquid mixer, but as these systems are slow the sampling time is in seconds. Both of these works utilized the ANN to approximate the whole behavior of the NMPC, the following works of Kittisupakorn and Ławrynczuk try to approximate only the non-linear system prediction. In (Kittisupakorn et al. 2009) the ANN does the first state prediction and then the rest is derived by the iterative method. Expanding upon this method, ((Ławryńczuk 01 Jun. 2009),(Ławryńczuk 2011) and (Ławryńczuk 2009)) employs structured ANN models to avoid the accumulation of error by the iterative method.

The possibility of using RNAs to represent the behavior of non-linear systems is corroborated by research that demonstrates feedforward neural networks as universal approximators (Hornik, Stinchcombe e White 1989) and (Park e Sandberg 1991) . In (Ławryńczuk 2009), it presents a survey on the use of RNAs applied to MPC and highlights that among the most used architectures are the Multi-Layer Perceptron (MLP) and the Radial Basis Function (RBF), the latter being the one used in this dissertation.

2.3 RADIAL BASIS FUNCTION ARTIFICIAL NEURAL NETWORKS

The radial basis functions artificial neural network may be represented as

$$
\hat{\lambda} = F[r] = W \cdot \Phi(r, C, \sigma), \tag{2.16}
$$

where $\hat{\lambda} \in \mathbb{R}^p$ and $r \in \mathbb{R}^{n_r}$ are respectively the i-th network predicted output and the input vector, $C \in \mathbb{R}^{M,n_r}$ and $\sigma \in \mathbb{R}^M$ are respectively the centers and the widths of the radial basis functions

artificial neural network, $M \in \mathbb{N}^+$ is the number of neurons in the hidden layer, and the output weights are given by $W \in \mathbb{R}^{p,M}$. The outputs of the hidden layer are given by

$$\Phi(r, C, \sigma) = \begin{bmatrix} \phi(r, C^1, \sigma^1) \\ \vdots \\ \phi(r, C^M, \sigma^M) \end{bmatrix}, \tag{2.17}$$

where $\phi(r, C^m, \sigma^m)$ is the output of the hidden layer for each m-th neuron[1]. The Gaussian activation function is more frequently used in the case of radial basis functions artificial neural networks. They may be calculated as

$$\phi(r, C^m, \sigma^m) = \exp\left[-\frac{\|r - C^m\|^2}{2(\sigma^m)^2}\right]. \tag{2.18}$$

The squared vector norm calculation may be summarized as

$$\phi(r, C^m, \sigma^m) = \exp\left[-\frac{1}{2(\sigma^m)^2}\sum_{i=1}^{n_r}(r^i - C^{m,i})^2\right], \tag{2.19}$$

which is more conveniently used for calculation in hardware as will be seen next.

The radial basis function artificial neural network can be employed for state estimation according to the approximate strategy devised previously. We look for constructing a nonlinear mapping $F : \mathcal{X} \times \mathcal{I} \to \mathcal{X}$ which can be constructed by employing an artificial neural network to fit a dataset constructed offline by solving (2.3) in a simulation environment, where $\mathcal{I}, \mathcal{X}$ represent respectively the function domains for the information vector and the state estimate. Being so, the artificial neural network will deliver a state estimate readily once the information vector and the state prediction are updated. Thus if we set

$$\lambda_t = \hat{x}_{t,t}, r_t = \begin{bmatrix} \bar{x}_{t-N,t} \\ I_t \end{bmatrix}, \tag{2.20}$$

and build a dataset to train the artificial neural network using simulation data and the results of the moving-horizon which is rich enough to represent the system dynamics, it is possible to construct a proxy for the optimal moving-horizon estimator using an artificial neural network. In the following section, we devise hardware architectures to implement the radial basis functions artificial neural network as in (2.16).

2.3.0.1 Approximation Residual Evaluation

It is possible to compare λ_t and $\hat{\lambda}_t$ to evaluate the approximation capability of the artificial neural network. If we define the residual measure as $e_t^i = \lambda_t^i - \hat{\lambda}_t^i$, we can calculate the multiple

[1] We denote by A^k the vector composed by the k-th line of a matrix A. Similarly, the k-th component of a vector v is given by v^k.

correlation coefficient for each i-th output as (Schaible, Xie e Lee 1997)

$$R^{2,i} = 1 - \frac{\sum_{t=1}^{N} \left[e_t^i\right]^2}{\sum_{t=1}^{N} \left[\lambda_t^i - \bar{\lambda}^i\right]^2}, \tag{2.21}$$

where N is the total number of samples and the upper bar denotes the mean of a sequence. The closer R^2 is to 1, the better is the approximation capability of the artificial neural network. It is convenient to use this measure as it allows the comparison of outputs of different units and magnitudes, as we will evaluate the approximate estimation of translation/angular positions/velocities.

2.4 REPROGRAMMABLE SYSTEM ON CHIP

With a focus on the application of the MHE in embedded systems, it is necessary to explore the appropriate architectures that meet the requirements of real-time execution and low energy consumption. In this sense, the architecture of the computational system has a great influence on its performance. The most common architectures are general-purpose processors (GPPs), Digital Signal Processors (DSPs), Graphics Processing Units (GPUs), and architectures dedicated, being implemented in reconfigurable devices such as FPGAs (FieldProgrammable Gate Arrays) or manufactured in silicon, as is the case of an Application Specific Integrated Circuit (ASIC) that is a dedicated circuit for a specific purpose, a System-on-chip (SoC) that includes a processor or a System-on-chip Multiprocessor (MPSoCs) that includes multiple processors (usually heterogeneous) (Gajski et al. 2009).

General-purpose processors, whether embedded or not, are based on the von Neumann model (Burks, Goldstine e Neumann 1982) which defines an architecture where instructions and data are stored in memory and interconnected to a processing unit via a bus. This architecture widely used in the industry has limitations, the main two being called the Memory (Wulf e McKee 1995) and Power Walls (Hartenstein). The first derives from the fact that the central processing unit (CPU) has a higher frequency than the memories it uses, limiting its processing performance. The latter advents from the increase in the number of transistors together with an increase in frequency, then reaching a power consumption and heat dissipation limit. Coming to the adoption of multiple processing units, commonly known as multi-core. But, this is not always practical or beneficial as many algorithms are not easily parallelized and many of those have limitations in architectures.

In a comparable form, the GPUs have been becoming more popular and increasing their market, with their highly parallel processing unit architectures. Its downsides are that it is not suited to sequential algorithms and has an even higher energy consumption than CPUs. A possible alternative to overcome these limitations is the production of specific architectures that directly implements the algorithms in hardware. This enables the possibility to adjust between dedicated processing units for sequential and parallel applications. This middle-ground is where FPGAs provide adaptable architecture with fast prototyping and low energy expenditure when compared to

high-frequency embedded CPUs and GPUs. This flexibility can be utilized both as an end-product or as a way of validating a project to be done on a dedicated chip. These qualities are the reason for its adoption in this project.

2.4.1 Field Programmable Gate Arrays

A Field-Programmable Gate Arrays, more known as FPGA, is a chip consisting of an array of configurable logic blocks (CLB). Different from an Application Specific Integrated Circuit (ASIC), which, as its name says, can perform only a unique function for the lifetime of the chip, an FPGA can be reprogrammed as needed to serve various functions in little to no time. The potential to be reprogrammed of FPGAs are at the same time, a blessing and a curse, as it offers high flexibility in use, but vastly increases the difficulty of programming it when compared to software. Since early 2000 FPGAs have begun to contain enough resources (logic cells/IO) to make the High-Performance Computing (HPC) community interested (Wain et al. 2006). But despite these disadvantages, FPGAs are a compelling option for digital system implementation due to their less time to market and low volume cost (Farooq, Marrakchi e Mehrez 2012).

Normally FPGAs comprise of:

- Programmable logic blocks which implement logic functions.
- Programmable routing that connects these logic functions.
- I/O blocks that are connected to logic blocks through routing interconnect and that make off-chip connections.

A generalized example of an FPGA is shown in Fig. 2.4 where configurable logic blocks (CLBs) are arranged in a two-dimensional grid and are interconnected by programmable routing resources. I/O blocks are arranged at the periphery of the grid and are also connected to the programmable routing interconnect. The "programmable/reconfigurable" term in FPGAs indicates their ability to implement a new function on the chip after its fabrication is complete. The reconfigurability/programmability of an FPGA is based on an underlying programming technology, which can cause a change in behavior of a pre-fabricated chip after its fabrication (Farooq, Marrakchi e Mehrez 2012).

There are multiple discrepancies between traditional software and hardware design flow for FPGAs. Subsequently, to the design and implementation of a hardware design, there is a multistep process that needs to be done before it can be used in the hardware. Step 1 is Synthesis, which translates HDL code into a textual description of a circuit diagram or schematic named netlist. Step 2 is using register-transfer level simulation to verify that the design described in the netlist functions accordingly. Thereafter, the netlist is translated into a binary format (Translate), the components and connections that it defines are mapped to CLBs (Map), and the design is arranged and routed to fit onto the FPGA (Place and Route). Another simulation is then executed to establish

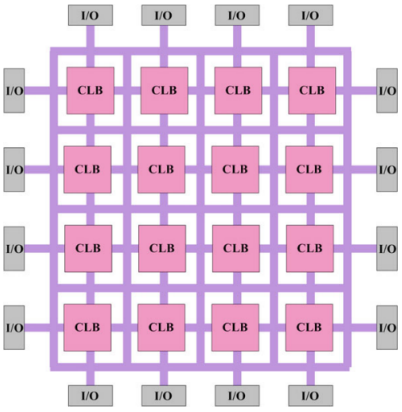


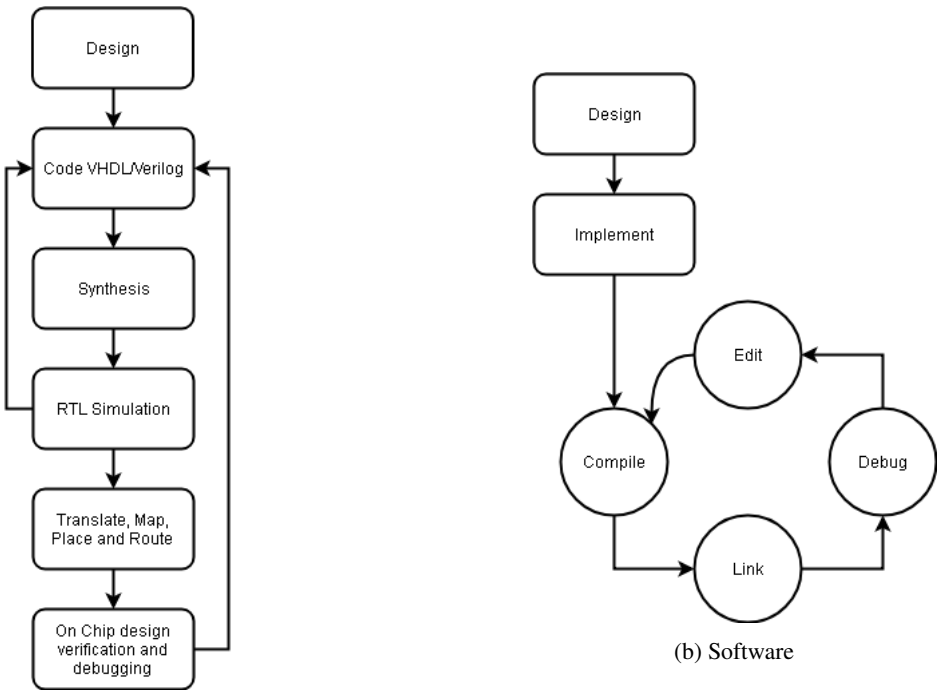
Figure 2.4: Overview of an FPGA architecture (Farooq, Marrakchi e Mehrez 2012)

how the design has been arranged and routed. Lastly, a file is created to load the design on the hardware. This file is a configuration file used to program all the resources in the FPGA. Utilizing tools, such as Xilinx Chipscope, it is possible to analyze and debug the design while it is running on the hardware. The software design flow has no such requirements for the pre-implementation simulation step (Wain et al. 2006).

> Compile times for software are much shorter than implementation times for hardware designs so it is practical to recompile code and perform debugging as an iterative process. In hardware, it is very important to establish that a design is functionally correct before implementation as a broken design could take a day or more to place and route and could potentially cause damage to system components. Figures 2.5b and 2.5a illustrate the differences between software and hardware design flows (Wain et al. 2006).

The use of FPGAs in real-time applications already have a formal verification in (Jabeen, Srinivasan e Shuja 2017) and are used for industrial processes (Ibañez, Ocampo-Martinez e Gonzalez 2017), predictive control (Iplikci e Bahtiyar 2016), aircraft control (Hartley et al. 2013), and even some metaheuristics implementation on control (Xu et al. 2015) and robotics (Huang 2012). Lastly, we quote (Sampaio 2018) on the current state of the FPGA market and techonology:

> In terms of the configuration of processors embedded internally in the FPGA, we can see examples from the two largest manufacturers, Altera and Xilinx. Altera offers in its FPGAs the possibility of configuring the Nios II processor which is a Soft Core Processor or softcore (the reconfigurable processor that uses the logical elements of the FPGA) and Xilinx offers the Soft Processor MicroBlaze. Hardcore processors (SoC) like ARM are also available in the Cyclone V family from Intel / Altera (Cyclone V Device Handbook - Intel) and in the Zynq family from Xilinx (Zynq-7000 SoC Data Sheet: Overview), which makes co-design solutions more efficient (Noguera e Badia 2002).



(a) Hardware

(b) Software

Figure 2.5: Design Flowchart of Hardware and Software

Although FPGAs have historically offered a limited number of logic cells, low frequency, and higher power consumption compared to dedicated hardware solutions (ASIC), recent releases from both manufacturers using fabrication with 16 and 14 nm transistors provide solutions with a high number of logic cells, in the millions and high frequencies, in the 1 GHz band. Two practical examples are the FPGAs of the Virtex-7 family from Xilinx (Virtex-7 FPGAs 2012) with up to 2,000,000 logic elements and the Intel / Altera (Kenny e Watt 2016) Stratix 10 family FPGA with up to 5,510,000 logic elements. Both of these features combined with manufacturing technologies are capable of providing a reduction in energy consumption of up to 70% compared to previous devices. These features make these devices increasingly attractive for large-scale use.

2.5 STATE OF THE ART

In this section, we present a simple bibliographic study done to illustrate the relevance and innovation factors of this work. We intend to do this with the help of the data collected from the Web of Science (WoS) Core Collection and shown here in Tables 2.1 and 2.2. Table 2.1 refers to the number of related papers by topic, the intersections are related to the number found when searching utilizing both topics with the **AND** logic connector. While Table 2.2 exhibits the number of the related papers found by search topic in a number of years. Although most search topics are self-explanatory, the topic here referred to as *Embedded* actually represents the results of a

group of search topics, these are embedded, microcontrollers, GPUs, FPGAs. All these topics together are collectively referred to as embedding, as such the results found in the FPGA column are a subset of it.

Initially, we can observe in Table 2.1 that the MHE has much fewer total related papers than both its dual, the MPC, and its primary competitor, the Kalman filter, showing its smaller popularity and use. One important thing to point out is that while this data gives an overview of the popularity of each topic, it is not completely accurate. As the related articles found in the intersections $MHExRBF$ and $MHExNNembedded$ do not in fact related to the moving-horizon estimator, but actually are about moving-horizon neural network methods (Dewasme 2019) and (Fernando et al. 2019). On the other hand, the paper found by the intersection $MHExFPGA$ is indeed about an FPGA implementation of an alternating direction method of multipliers (ADMM) which solves QP problems arising from Moving Horizon Estimation (Dang e Ling 2014). The intersection $MHExNN$ mainly refers to works where both were employed together, not using the neural-network to approximate the MHE behavior as done in this dissertation.

The other main point of Table 2.1 is to show that there are still few papers that approach both the MPC and the MHE together, while the KF is much more used as it is simpler and easier to use. The combination of MPC, MHE, and NN brings only two results, the first being the (Alessandri et al. 2011) paper much-cited and discussed here in this dissertation, and the other is (Kamesh e Rani 2017) which discuss, in fact, an MPC formulation based on EKF employing an adaptive ANN model for supervisory control and is illustrated for setpoint tracking of reactor temperature of an industrial multiproduct semi-batch polymerization reactor challenge problem.

However, the combination of MPC, MHE, and embedded systems, brings about 9 different results, from which 3, (Seenivasan, Olivares e Staffetti 2020),(Englert et al. 2019) and (Frey et al. 2019) does not have relation to MHE and is only about embedded MPC. Another one is a solver for constrained trajectory optimization based on a sequential operator splitting framework and is potentially suitable for nonlinear model predictive control and moving horizon state estimation in embedded systems (Sindhwani, Roelofs e Kalakrishnan 2017). The rest combine MPC, MHE and an embedded implementation, varying from uses in induction motors (Frick et al. 2012), delivery drones (Mehndiratta e Kayacan 2019), communication delays and losses (Zeng e Liu 2015), fault identification (Peng et al. 2015) and lastly, an overview of its opportunities and challenges (Findeisen, Graichen e Monnigmann 2018).

If Table 2.1 confirms this dissertation novelty factor, by displaying that its intended results have very little to no existing similar papers results. Then, Table 2.2 serves to endorse these topics' relevance, as research continues to increase and grow, especially in the last decade.

Table 2.1: Number of related papers by topic from Web of Science

Search Topic	Total	RBF	Approximation	Embedded	NN	NN embedded	FPGA
MHE	1088	1	154	91	142	1	1
MPC	58076	296	1394	991	3817	88	229
NN	425373	12347	13134	15729	425373	15729	2868
FPGA	52238	64	687	6859	2863	2863	52238
RBF	25527	25527	4363	559	12347	559	64
MPC with MHE	247	0	14	9	2	0	0
Kalman Filter	56642	262	2064	1490	2906	90	242
KF MPC	1054	6	39	20	102	2	1

Table 2.2: Number of related papers by topic and year according to Web of Science

Topic\Publication Year	Until 2000	2000-2005	2006-2010	2011-2015	2016-2020
MHE	58	45	158	350	465
MPC	2941	4767	8005	14880	27389
NN	43199	47002	62157	76358	195452
FPGA	1550	5674	11569	15560	17840
RBF	2071	3472	5097	6215	6996
MPCMHE	1	13	33	80	121
MHE FPGA	0	0	0	1	0
MPC FPGA	0	3	21	72	133
MPCMHE FPGA	0	0	0	0	0
Kalman Filter	5028	5656	9928	14858	21015
KF MPC	59	98	153	272	467

2.6 CHAPTER FINAL CONSIDERATIONS

All of the methods, architectures and hardware that will be employed in this dissertation have been explained in this chapter, along with their combined relevance and state of art. The next chapter is one amalgamation of most of these theories put in practice, approximating the MHE with an RBF ANN and embedding it on an FPGA.

3 EFFICIENT HARDWARE IMPLEMENTATION OF NONLINEAR MOVING-HORIZON STATE ESTIMATION WITH ARTIFICIAL NEURAL NETWORKS

This chapter discloses the RBFMHE implementation, mainly the same content as the published paper (Brunello et al. 2020), and some improvements and corrections made after its publication. The first section describes its design and the methodology employed in its development in a broad manner, while the second section shows its results, the case study used and specifies some parameters and experiments from the first section.

3.1 DESIGN OF APPROXIMATE MOVING-HORIZON STATE ESTIMATION

The optimization problem in Eq. (2.3) may be performed to obtain the state estimates within a sliding window. Being so, it is necessary to solve at each sampling instant an optimization problem, which in many cases may be impractical or demand significant computational resources. In (Alessandri, Baglietto e Battistelli 2008) the authors propose also an approximate version of the algorithm described in Eqs. (2.3) and (2.4), which was further analyzed for the case of artificial neural networks in (Alessandri et al. 2011).

The method for obtaining an approximate version of the moving-horizon estimation algorithm as given in Section2.1 can be obtained if the mapping from the state prediction and information vector, which is performed offline solving an optimization problem, is done by any function approximation regression model. In other words, we want to construct an approximate nonlinear mapping for the mathematical operation performed in the optimization in Eq. (2.3), which will be much less computationally intensive and indicated for an embedded computation platform. This can be achieved by obtaining the state estimates offline and constructing a nonlinear mapping offline. For details see (Alessandri, Baglietto e Battistelli 2008, Alessandri et al. 2011).

To perform the nonlinear mapping from observations to state estimates, we may employ any

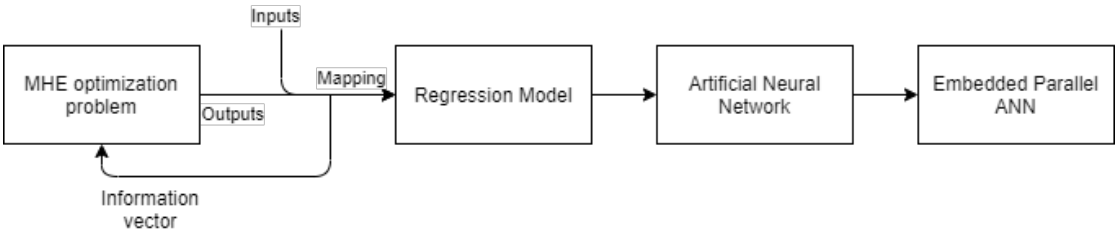


Figure 3.1: Design of the embedded approximate MHE using NN.

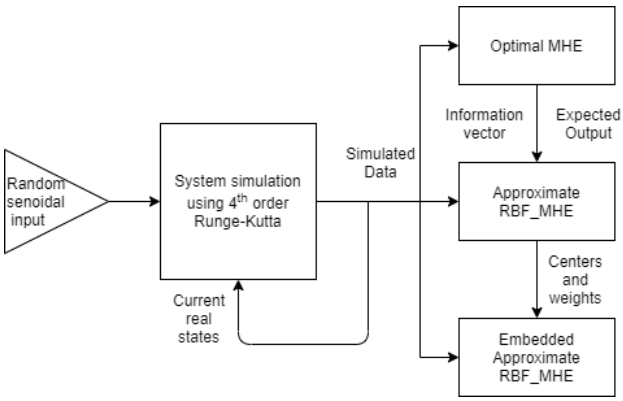


Figure 3.2: Dataflow of the RBFMHE implementation.

nonlinear approximation function for regression as we have many examples from machine learning. One of the cases is the artificial neural network, which is particularly interesting in real-time applications due to its inherent parallel architecture. To best take advantage of this parallel architecture, we decided to embed it on reconfigurable hardware, in this case, an FPGA. Figure 3.1 shows a simplified overview of the whole theoretical design of the embedded approximate MHE using neural networks.

3.1.1 Software implementation

Using what was previously explained as the analytical basis we now try to describe the procedures done to make its software implementation.

At first, we sketch how each macro part of the design will communicate with each other, defining the dataflow presented in Fig.3.2. In this image, it discloses that the model system will be approximated via a 4th order Runge-Kutta. It is fed by the system's current states and its control input is substituted by a sinusoidal input with varying amplitude. The system simulation is done in a whole, single batch, then deliver all the inputs, outputs, and states to the next phase. The system that is used in this dissertation as a case study is the inverted pendulum and is detailed in Subsection **??**. Its outputs are the linear and angular positions in centimeters and radians, combined with a random white noise which means are 0,05 and 0,1 respectively.

In the optimal moving-horizon estimation stage we now do a one-step at a time simulation traversing the data batch initiating when there is an information window one size larger than the moving-horizon defined. To solve the optimization problem at each iteration we use Matlab's *lsqnonlin* function, it is a nonlinear least-squares problems solver also accepting the use of lower and upper boundaries for its solution, one of MHE's better points over other estimation methods. The cost function used is the one represented in Eq.2.2 and the constant μ was 0,01 denoting that it does not have confidence in the initial estimate provided by the user. The solution of the optimization problem brings the $\hat{x}_{t-N,t}$ of the present MHE window, we then can finally use the same method in Eq.2.4 to obtain the rest of the estimates until the current moment.

When the MHE is finalized we carry the information vector and the estimated states to train the RBF neural network, the former being its input and the latter being its expected output. The main reason that the real states are not employed in the training is that we are assuming that they will not be readily available in many practical cases, that being the case we then train it to obtain the closest results to those of the optimal MHE. To utilize the information vector as the input it is first needed to format it in a configuration where each column has the current and the window with the horizon information. To do so means that the number of inputs in the NN will be the same as Eq.2.20. That is immediately followed by a normalization of all data, each one by its maximum presented value, this is done so that the neural network does not have to deal with multiple types of range of numbers, making it have weights values closer to each other.

After the normalization, it starts the training by finding the RBF's centers using Matlab's *kmeans* function, which uses the k-means clustering algorithm and the squared Euclidean distances as default to find the locations of the centroids of the provided data. This program receives the NN input and the number of neurons the network will have, this number is the same as the numbers of centroids the program must find. Next, it uses the centers, inputs and the σ to calculate the radial basis function ϕ previously determined in Eq.2.19 using a 2_{nd} order Taylor polynomial. This function can be exchanged with any other type of radial basis function best fitted to each type of problem. Lastly, it operates with the ϕ and the expected output to retrieve the weights, finalizing the training and securing all information needed to employ the RBF neural network. To apply the trained NN in a system loop or iteration it simply needs to insert the current input data on the RBF to obtain the current phi and multiply it by the weights to achieve the current output estimate, in this work, it is the state's estimation.

$$NN_{inputs} = horizon * (inputs_{MHE} + outputs_{MHE}). \tag{3.1}$$

As we already have the NN and know how to use it, now we need a form of evaluating its results. In Fig.3.3 it can be observed a brief step-by-step of all the main points in the RBFMHE implementation. In it is also presented how we decided to assess the results is using the R-squared residual evaluation shown in Eq.2.21, we also determine that the NN must obtain an R-squared result of approximately 0,95 in each of the estimates for it to be considered a good enough approximation of the optimal MHE. The means-squared error is also calculated, however it is not used as a fixed threshold that needs to be achieved. If the R-squared does not pass the test, then we return to the phase of defining the parameters of the RBF neural network. To better analyze the results of the network and expand the range of possible choices, it was initially made more than 150000 combinations of the number of neurons, σ, and random seed to observe how the network reacted at each of the parameters. The parameters were varied from 1 to 30, 1 to 1000, and 1 to 5 respectively. The greater variation of the $sigma$ parameter derives from the fact that it is the only parameter that we can control in reality without it affecting too much the energy consumption and the area cost on the embedded hardware. We then take all the R-squared data and put it into a table for each random seed, where each cell of the table is the lowest scoring R2 between the four

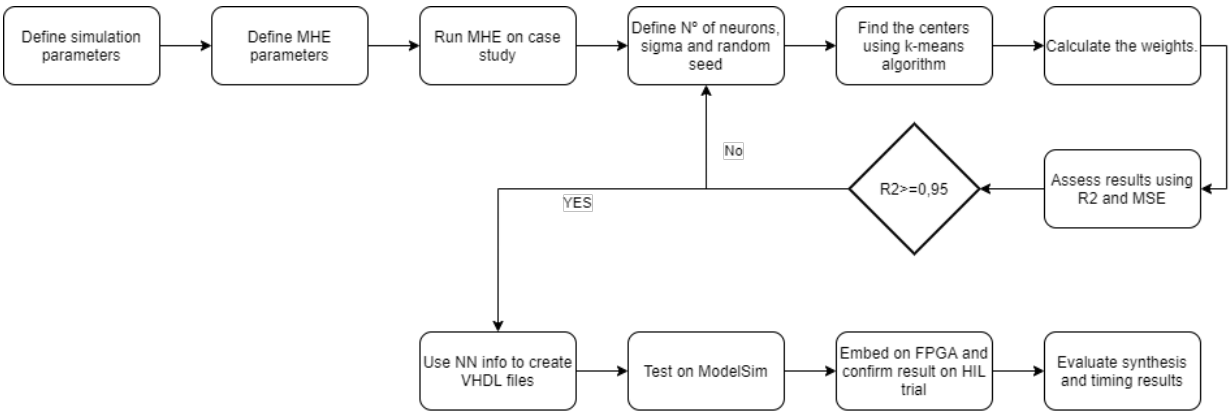


Figure 3.3: Step-by-step of the RBFMHE implementation.

states estimated. Finally, we find in these tables the lowest number of neurons needed to pass the R-squared threshold and the other parameters with it, this is the chosen RBF network to later be implemented.

With this, we then need to adapt the hardware architecture observed in (Ayala et al. 2017) to accept multiple outputs. To do so it was necessary to change the output design to a more modular one, where it could be easily increased or decreased the number of desired outputs. Then, it is firstly tested in the ModelSim software to verify if its results coincide with the expected. It is in this step that was encountered a problem forced the use of 64 bits representation instead of the initially desired 32 or less. This problem was solved and is unraveled in subsection 3.1.3 with more details. After the architecture was functioning on the ModelSim we embed it on an FPGA and confirm the results with a hardware-in-the-loop (HIL) trial. We then conclude the experiment by comparing the results with the expected ones and evaluating its synthesis and timing results.

In the next subsection, we describe its hardware implementation which may be leveraged to implement the state estimation algorithm on hardware.

3.1.2 Hardware Architectures for Radial Basis Function Artificial Neural Network

The hardware architecture of the radial basis function artificial neural network is implemented in Very High Speed Integrated Circuit Hardware Description Language (VHDL) using parallel modules for both the hidden layer neurons as well as the output layer neurons. All modules use Finite State Machines (FSMs) as their controllers for data processing synchronization and all arithmetic operations are done using customizable floating-point libraries for better precision and a large. These libraries are described in (Muñoz et al. 2009) and (Muñoz et al. 2010) and are based on the IEEE 754 standard (IEEE 1985). The main floating-point modules used in this work are the addition/subtraction, multiplication and exponential units called *FPadd*, *FPmul* and *FPexp*, respectively.

The generalized neural network architecture is presented in Fig. 3.4 which shows the variable

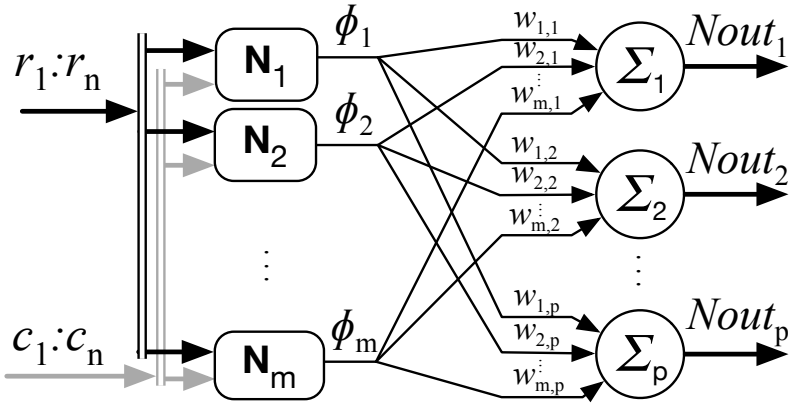


Figure 3.4: Radial basis function artificial neural network hardware architecture. (Brunello et al. 2020)

inputs ($r_1 : r_n$), the constant Gaussian centers ($c_1 : c_n$), obtained from the training process, all as inputs of the hidden layer neurons (**N**). The hidden layer neurons initiate their computations in parallel with a *start* signal and when their output values are done, yield a *ready* signal which initiates the output layer ($\sum$) compute process which uses the hidden layer outputs (Φ) and constant weights ($w_{m,p}$). Finally, the output layer raises a *ready_all* signal when the output values are available.

The hidden neuron module described in Fig. 3.5 is responsible for implementing Eq. 2.19 and uses two *FPadd*, a *FPmul* and a *FPexp* units to implement the logic. The process is controlled by an FSM using the start and ready signals for the *FPadd*, *FPmul* and *FPexp* units and synchronizes them using two MUXES with their respective switch signals (sel_1, sel_2) to efficiently minimize hardware area reusing each FP unit.

The output layer neurons follow a similar architecture each using one *FPadd* and one *FPmul* unit and is represented in Fig. 3.6.

Finally, a hardware description configuration MATLAB script was developed to easily be able to test multiple configurations of this architecture based on these neural network models. The script automatically generates the VHDL code for the radial basis function artificial neural network based on the number of inputs, the number of hidden and output layer neurons, and all constants resulting from the training process. The script can also generate each floating-point unit with a varying bit-width representation for a trade-off study of a more precise or a lower resource consumption architecture.

3.1.3 Precision representation problem

Now we return to the previously presented problem, which is that when using representation lower than 64-bits the network results were all completely wrong. After a thorough investigation, it was found that at the ϕ calculation phase all its values were very close to 1, producing numbers with nines for many decimal places like 0,999995. As it is known the floating-point representation

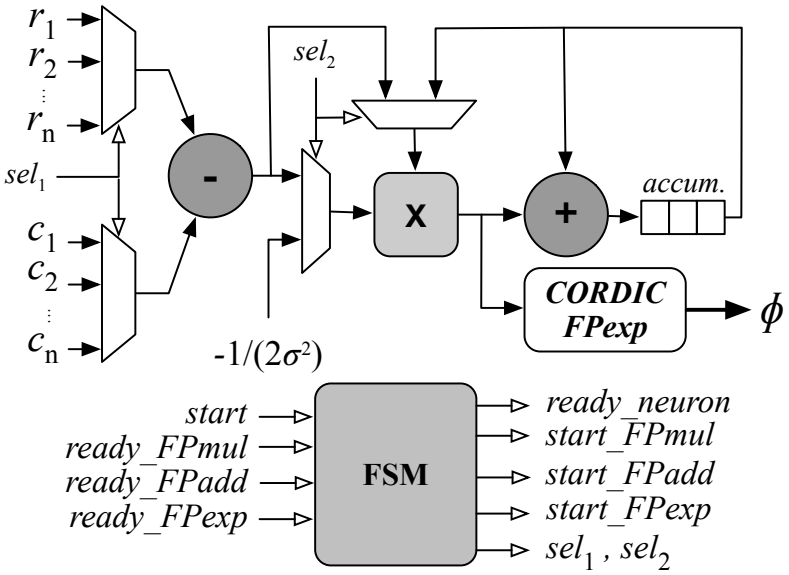


Figure 3.5: Gaussian radial basis function neuron architecture on hardware, denoted by **N** in Fig. 3.4.(Brunello et al. 2020)

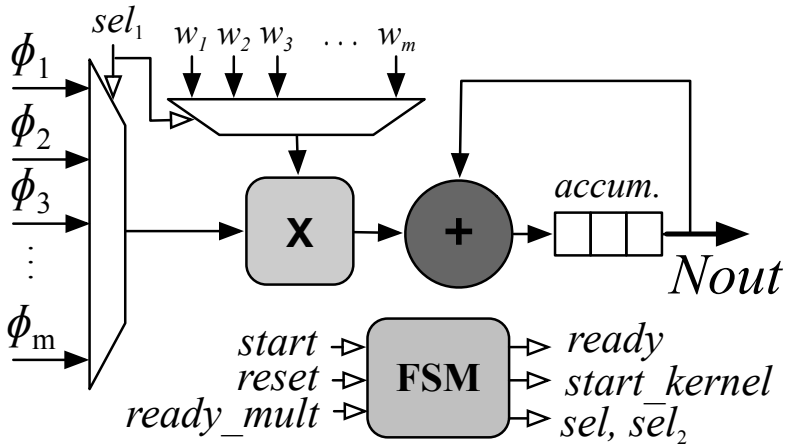


Figure 3.6: Radial basis function output layer neuron architecture on hardware, denoted by $\sum$ in Fig. 3.4.(Brunello et al. 2020)

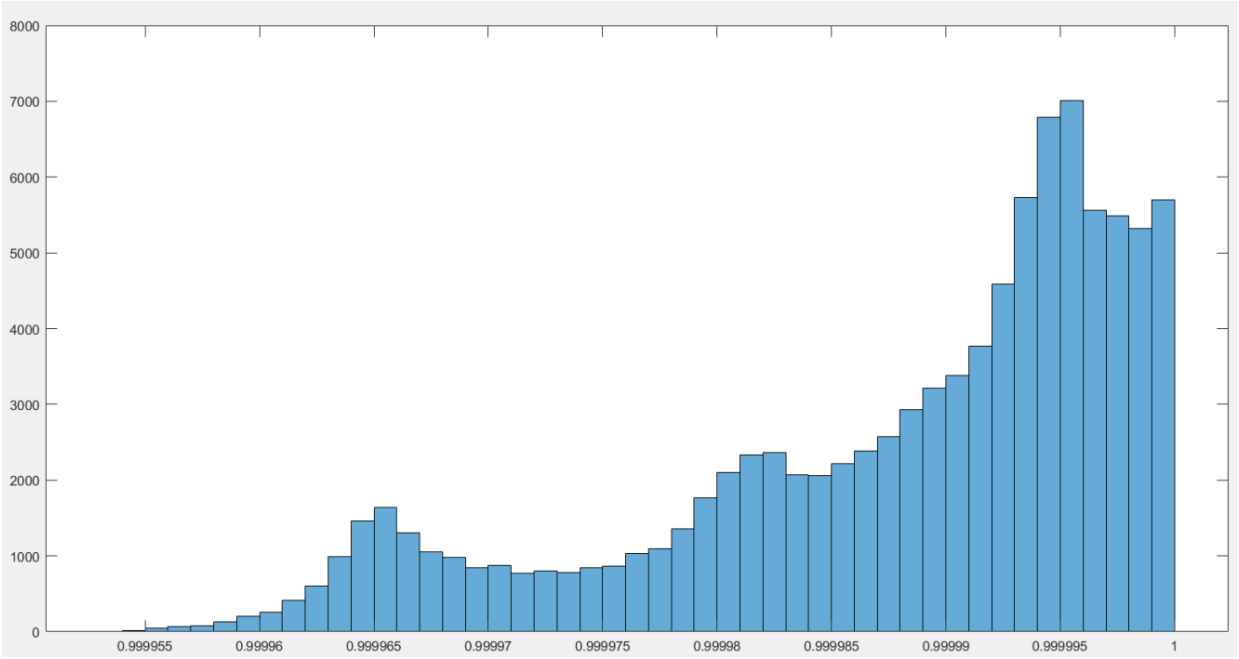


Figure 3.7: Histogram of phi values from a simulation

has trouble representing this kind of value and in many cases it rounds it up to 1 when doing arithmetics, losing all the meaning it has in the calculation. This problem was initially found during the ModelSim simulation, being later replicated in Matlab using the *cast32* function which forces computations to be done using 32 bits representation. In Figure 3.7 we can observe a histogram showing the value window that the ϕ variable has.

This problem was unfortunately deeply related to the radial basis function used, the estimation data values, and the σ parameter. As we increase σ to improve the result of the estimation, more precision is needed to represent ϕ and its subsequent calculations. The method needed to solve this conundrum was to change the cost function in a way that solved the precision problem but didn't force big alterations to the hardware design. This is achieved by using an ingenious approach, simply subtracting the radial basis function by 1. It accomplishes all the stated requirements, solving the precision problem by changing its value from numbers close to 1 to numbers with many left-side zeroes, which floating-point representation has no difficulty representing with few bits. The only adjustment needed in the hardware design is that it excludes the first element of the Taylor series. However, the other process that computes the radial basis function still needed to be addressed.

3.1.4 CORDIC removal from RBF neuron implementation

After the changes were done to correct the precision problem the design was once again tested in ModelSim, this time it worked without a problem. This though created more questions, how was it working when there have not been any changes in the CORDIC module. After inspecting all the

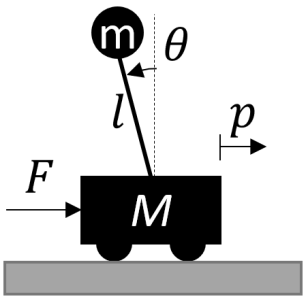


Figure 3.8: Representation of the inverted pendulum system, which is used as a case study for state estimation in the present paper.

simulation iterations it was then confirmed that the CORDIC was not utilized a single time. This happens because the values to be computed in the FP-CORDIC-TAYLOR module are all close to 0, where was shown in (Muñoz et al. 2010) that the CORDIC algorithm has poor performance, the module then only uses the Taylor series approximation. With this knowledge, it is then possible to remove the CORDIC completely without lowering the performance and reducing the hardware area needed for the RBF neural network. Finally, all the tuning, tests, and simulations were remade with this new architecture.

3.2 RESULTS ON RBFMHE

In the present section, we provide results for the MHSE implementation in FPGA using approximate solutions described in this dissertation. To this end, we use a simulated case study and analyze its real-time implementation using an Intel Arria 10 FPGA.

3.2.1 Case Study

In order to test the moving-horizon state estimation algorithm implementation on hardware, we use an inverted pendulum. See Fig. 3.8 for details. Let p, θ be respectively the translation of the cart and angular position of the pendulum, m, M be each of the concentrated masses, l and J are respectively the length and moment of inertia of the pendulum and F represents an exogenous force which is the input of the system. If we set the continuous-time state as $z = \begin{bmatrix} p & \theta & \dot{p} & \dot{\theta} \end{bmatrix}$ and the input as $w = F$, we can describe the equations of motion in nonlinear continuous-time state space form as (Aström e Murray 2008)

$$\dot{z} = f_c(z, w) = \begin{bmatrix} \dot{p} & \dot{\theta} & \ddot{p} & \ddot{\theta} \end{bmatrix}^{\mathsf{T}}, \tag{3.2}$$

with the dynamic equations of the transition of the state in continuous time are given by

$$
\begin{aligned}
f_c^3(z,w) &= \frac{-mls_\theta\dot{\theta}^2 + m^2gl^2s_\theta c_\theta/J_t - c\dot{p} - mlc_\theta\gamma\dot{\theta}/J_t}{M_t - m^2l^2c_\theta^2/J_t}, \\
f_c^4(z,w) &= \frac{-ml^2s_\theta c_\theta\dot{\theta}^2 + M_tgls_\theta - clc_\theta\dot{p} - \gamma\dot{\theta}M_t/m + lc_\theta u}{J_t(M_t/m) - ml^2c_\theta^2},
\end{aligned}
\tag{3.3}
$$

where $s_\theta = \sin\theta, c_\theta = \cos\theta$, $M_t = m + M$, and $J_t = J + ml^2$. We consider to measure both positions of the cart and the pendulum, that is, $y = \begin{bmatrix} p & \theta \end{bmatrix}^\intercal$. The inverted pendulum and its close variants have been used in many recent works with respect to control and state estimation (e.g., to cite a few recent contributions, (Dwivedi, Pandey e Junghare 2017, Messikh, Guechi e Benloucif 2017, Su et al. 2018), due to its switching stable/unstable regimes. As it is a well-known case study, it is a good choice for testing new implementations on hardware for state estimation algorithms. It will also ease the reproduction for the interested reader.

3.2.2 Simulation results

Data is generated using a 4th order Runge-Kutta solver with sampling time of 10 ms, so that the discrete-time model formulation as in (2.1) can be employed. Simulated data is generated with a sinusoidal input of 10 rad/s and 50 N amplitude, for 20 seconds in total, and the measurements are corrupted by a white noise signal, so that the mean is the true measurement and with covariance matrix as $\text{diag}(0.05, 0.1)$. The parameters of the physical system used in the simulations are $M = 10$ kg, $m = 80$ kg, $c = 0.1$ N s/m, $J = 100$ kg.m^2/s^2, $l = 1$ m, $\gamma = 0.01$ N.m.s, $g = 9.8$ m/s^2, all with proper units.

The optimal moving-horizon state estimator is then run in order to establish the state estimations as required. After some trial and error, we set $N + 1 = 11$ as the moving-horizon length and $\mu = 0.01$. We use this for this example no state prediction and a window containing the 10 most recent measurements. The size of the window has been set so as to guarantee constrained error as $t \to \infty$. By running the optimal estimator, we were able to generate all the inputs necessary for the approximate version of the state estimator. The required data to construct the approximate version are, according to Eq. (2.20), the time history of the information vector and the state estimate at the beginning of the moving-horizon window, as we are not using the state prediction for sake of simplicity as the focus in on the comparison of execution time on different heterogeneous platforms as we will see in the next subsection.

An approximate filter has been obtained for the optimal moving-horizon estimator, using a radial basis function artificial neural network as in Eq. (2.16). We set the inputs of the network as the information vector (I_t) containing all inputs and outputs within the receding-horizon window. We trained the network using the data from the optimal estimation procedure using a 2-steps procedure (Haykin 2009), by selecting the centers using the k-means algorithm, fine-tuning the spread by testing many different values, and getting the output layer weights with the Penrose-Moore pseudo-inverse with QR factorization (Moody e Darken 1989).

The training has been done for a range of $1/\sigma^2$ in $[0.1, 100]$ with a step of 0.1 between two consecutive values. The impact of the number of neurons on the estimation accuracy was also evaluated by varying it in the range $[1, 30]$. As the k-means algorithm depends on random initial solutions, we tested all these configurations 5 times. Testing 5 times, for the range of $1/\sigma^2$ and the number of neurons set amounts to 150,000 different artificial neural networks tested.

In Figures 3.9 and 3.10 we see examples where we maintain one of the parameters constant and manipulate the other evaluating the resulting R-squared, with these we can infer that the angular velocity state is the more difficult state to estimate. The former shows that the lower the value of the delta, or the higher the sigma, the better are the results for $\dot{\theta}$, the other states presents a similar pattern however we can observe a limit being reached on $\dot{p}$ close to 0,95. As this parameter does not, at first, results in a variation in the area and computational cost we utilize its value with the best results. Whereas, the latter display that there are inflection points on the curves where increasing the number of neurons does not significantly improve its results, sometimes even making them worse. A complete surface of the variation of all parameters and states along with a summarized version of the chosen tables can be found in the Appendix I Figure I.1 and I.2.

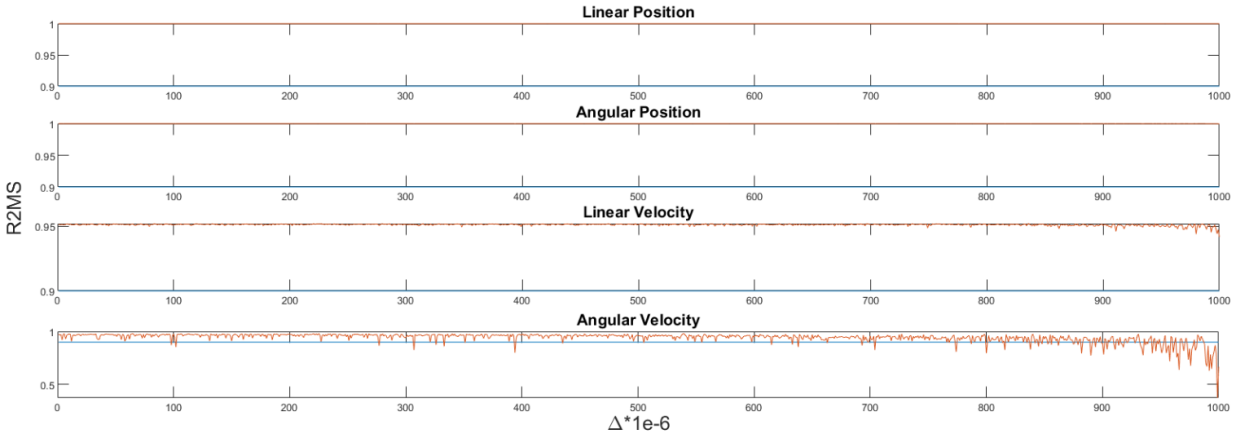


Figure 3.9: R-squared of all states by the variation of delta.

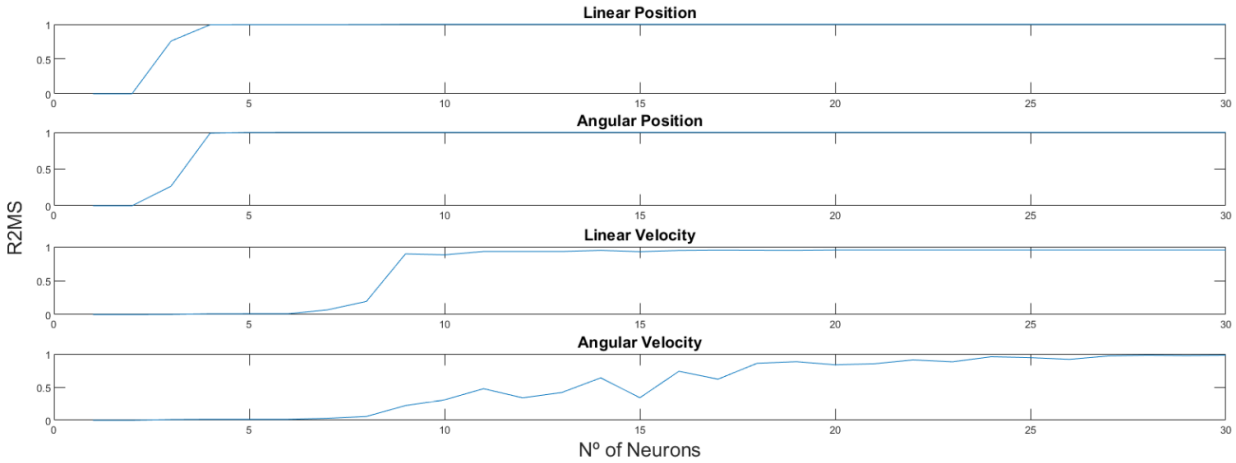


Figure 3.10: R-squared of all states by the variation of number of neurons.

This information is then summarized in Figure 3.11, which depicts the multiple correlation

coefficient as in (2.21) for the least accurate state (among the four) evaluated for a given number of neurons in the artificial neural network. It can be seen that there is a point in which augmenting the number of neurons does not improve significantly the accuracy of the artificial neural network in estimating the states. As the hardware resource use, energy consumption, and latency are directly affected by the number of neurons, which represent the computational complexity of the state estimator, we favor the solution that represents the best compromise between accuracy and complexity. We then chose the artificial neural networks around the 14 neurons solution, which is the inflection point of the curve with the greatest R^2.

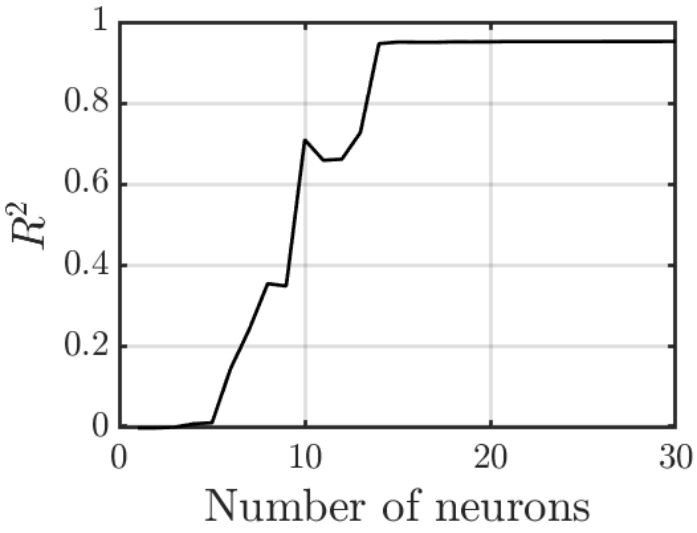


Figure 3.11: Multiple correlation coefficient for the worst state estimate obtained for the radial basis functions artificial neural networks varying the number of neurons. We can see that there is a point which represents a good trade-off for accuracy and complexity.

Employing the configuration of 14 neurons, sigma equal 1000 and the random seed 3, we acquire the data on Figure 3.12 that compares the results of the approximate MHE with the optimal one. As both results are close to each other Figure 3.13 serves to better display that the MSE of the approximation is higher than the optimal solution, but still within a reasonable margin. Next, we will present the results from the FPGA synthesis and the HIL simulation.

3.2.3 FPGA synthesis results

The radial basis function artificial neural network as reported in Subsection 3.1.2 is implemented using VHDL code. This code was simulated using ModelSim for logic validation synthesized on Intel® Quartus Prime 17.1 for the FPGA implementation. The target FPGA was an Intel Arria 10 (10AS066N3F40E2SG). The mean squared errors (MSE) obtained by the estimation schemes implemented are summarized by using a modified mean square error metric for the case of state

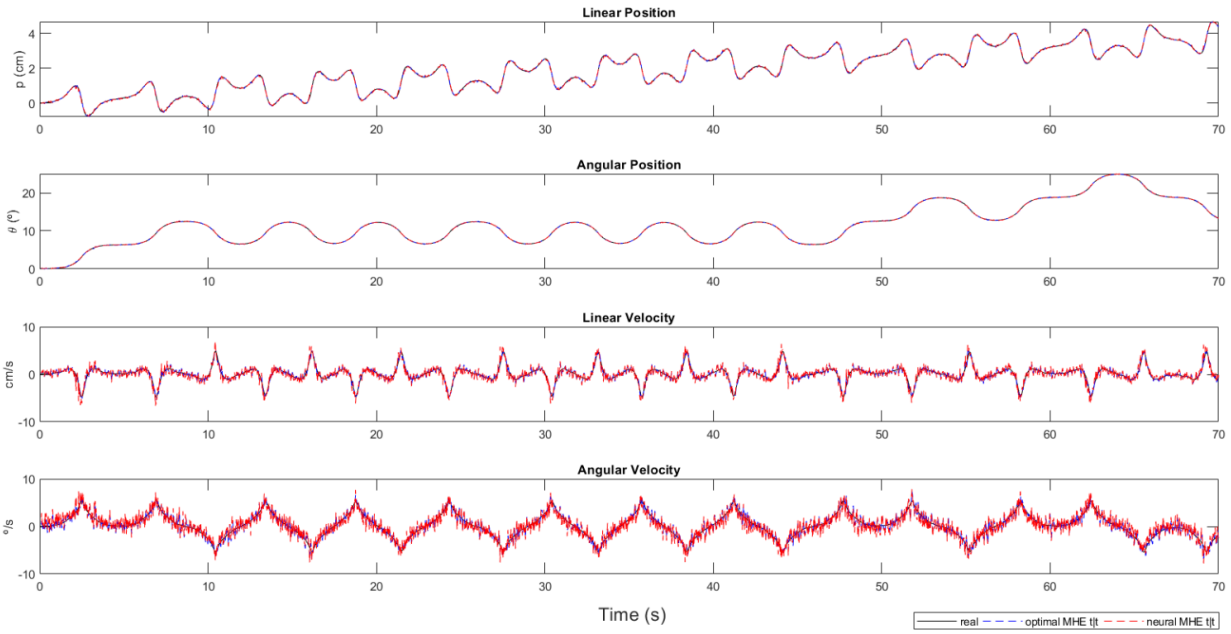


Figure 3.12: Comparison between all the real states and its estimates by the RBFMHE at t—t.

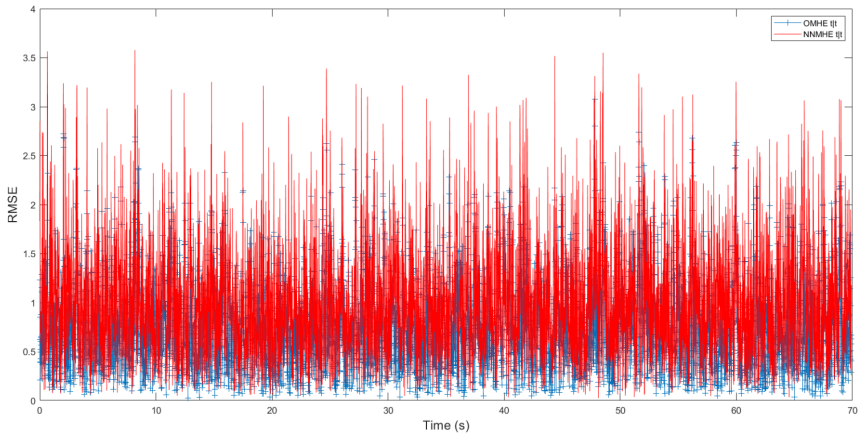


Figure 3.13: Comparison between the mean squared error of the optimal MHE and its NN counterpart.

vectors, as

$$\text{MSE}_{\text{dB}}\left(x^{[1]}_{t-N,t}, x^{[2]}_{t-N,t}\right) = 20\log\left[\frac{1}{N_t - N}\sum_{t=N+1}^{N_t}\|\hat{x}^{[1]}_{t-N,t} - \hat{x}^{\text{FPGA}}_{t-N,t}\|^2\right], \tag{3.4}$$

where N_t is the total amount of samples and $x^{[1]}_{t-N,t}, x^{[2]}_{t-N,t}$ are two different state estimates at t made at $t-N$ one wishes to compare. Being so, the MSE_{dB} presents an overall metric of how well the estimates at the beginning of the sliding window are made. Minimizing MSE_{dB} means that the difference of $x^{[1]}_{t-N,t}, x^{[2]}_{t-N,t}$ are minimized. For example, if we compare the optimal solution with

the approximate implemented in hardware, minimizing MSE_{dB} means improving the accuracy of the FPGA solution. Table 3.1 summarizes the MSE_{dB} for the cases in the inflection point of the R_2 curve shown in Fig 3.11, namely the solutions around the 14 neurons case. In this table we see the comparison of estimates made on the FPGA, the artificial neural network run offline, and the optimal moving-horizon state estimation. From the results presented in the table, we can see that (i) for a number of neurons greater than 13 the estimates do not present great improvements in terms of accuracy, what corroborates the information given in Fig 3.11 and (ii) that the implementation on hardware is accurate when compared to the MATLAB offline solution, as the errror is virtually zero (please note that the values are given in dB). Thus, we look further for the synthesis results for number of neurons greater than 13, as we shall analyze in the following, as they present a good compromise in terms of accuracy, hardware consumption.

Table 3.1: Accuracy of the hardware implementation of the moving-horizon state estimation implemented on hardware. In the table below, the state estimates of the optimal filter performed offline, the FPGA implementation, and artificial neural network implemented on MATLAB are described respectively as $\hat{x}^{\text{opt}}_{t-N,t}$, $\hat{x}^{\text{FPGA}}_{t-N,t}$, and $\hat{x}^{\text{ANN}}_{t-N,t}$.

Metric \ Nº of neurons	10	11	12	13	14	15	16
$\text{MSE}_{\text{dB}}\left(\hat{x}^{\text{opt}}_{t-N,t}, \hat{x}^{\text{FPGA}}_{t-N,t}\right)$	7.53	11.12	9.82	5.27	-6.87	-8.18	-9.84
$\text{MSE}_{\text{dB}}\left(\hat{x}^{\text{FPGA}}_{t-N,t}, \hat{x}^{\text{ANN}}_{t-N,t}\right)$	-178.26	-170.36	-167.38	-157.00	-170.90	-183.05	-184.52

Table 3.2 shows the synthesis results including the number of Adaptive Logic Modules (ALMs), Registers (REGs), dedicated hardware multipliers (Mult.), maximum frequency (Freq.), and the number of cycles needed to compute each solution. Three architectures with a varying number of neurons from 14 to 16 were synthesized to illustrate the trade-offs between accuracy and hardware use. All floating-point operations are synthesized with double precision arithmetic (64 bits) since lower precision operators incur high computation errors for these radial basis function artificial neural network configurations.

In Table 3.3 we see the timing results of the approximate moving-horizon state estimator. The time to process a state estimate of the artificial neural network has been calculated by $T_i\times$ no. of cycles, using information from the last two columns of Table 3.2 (T_i is the respective period in each artificial neural network). It is possible to see that the approximate filter implemented on FPGAs achieves considerably high-frequency rates, in the order of 500 kHz. The result is compared to a software solution running on an ARM processor of a Raspberry Pi 3 to showcase the speed up that can be achieved by the dedicated hardware solution. A speed-up rate higher than 31 times is achieved. This result is important as it enables the application of such advanced filtering methods to systems with fast dynamics (Zorić et al. 2019) or with complex structure, (Christofides et al. 2013) on low power embedded systems.

Table 3.2: FPGA Synthesis Results.

Neurons	ALMs (251.680)	REGs	Mult. (156)	Freq. (MHz)	Cycles
Cyclone V		(56.480)		(156)	
13	45,656	13,548	68	67.5	173
14	47,981	14,509	72	66.2	178
15	50,121	15,470	76	65.6	179
16	54,578	16,435	80	64.1	182

Table 3.3: FPGA Timing Results.

Neurons	FPGA (μs)	ARM (μs)	Speed up
13	2.56	46.98	18.3 x
14	1.60	50.58	31.6 x
15	1.64	54.19	33.0 x
16	1.67	57.80	34.6 x

3.2.4 Improved RBFMHE

After improving the approximate MHE changing the cost function to Gauss-1 we redo all the previous simulations. Figure 3.14 confirms that the software simulation and results are on par with the ones found previously, while Table 3.4 show that the number of, adaptive logic modules (ALMs), registers, multipliers used are less than half of that of the previously 64-bit implementation even if the time taken has remained the same. This decrease of the area made it possible to embed it on a smaller, simpler, and more compact FPGA which are all improvements desired to a real-time application.

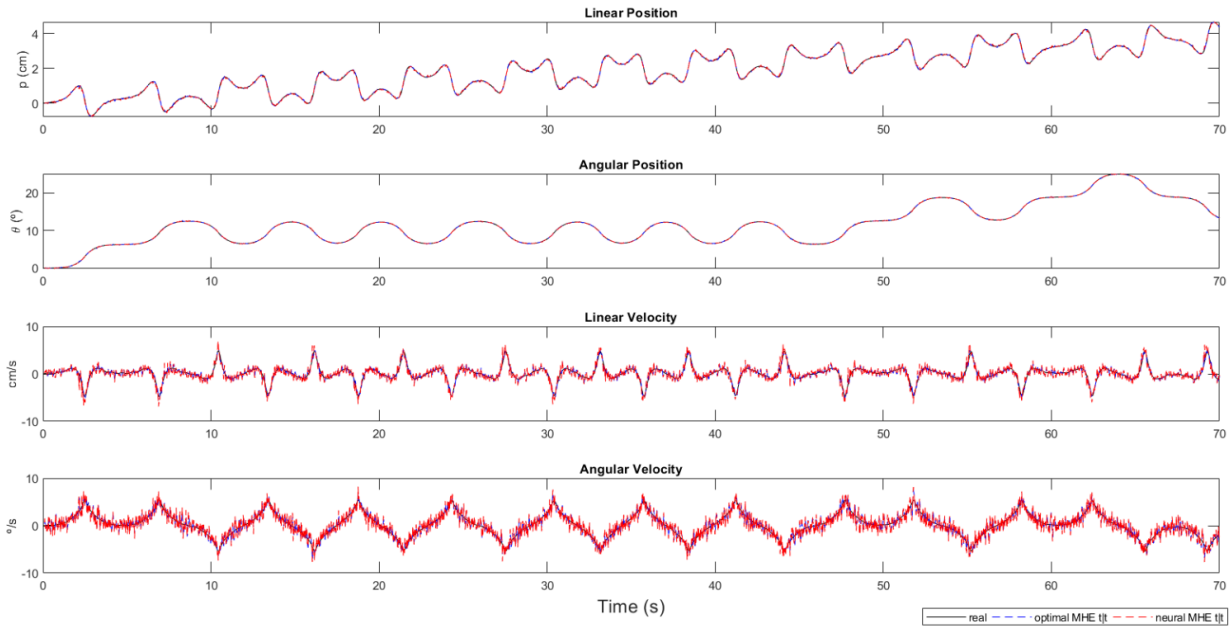


Figure 3.14: Comparison between the state results and mean squared error of the optimal MHE and its NN counterpart with G-1 as cost function and 32-bits representation. State results at t/t.

Table 3.4: FPGA Synthesis and Timing Results for the improved RBFMHE

Family				Device		
Cyclone V				5CSXFC6D6F31C6		
ALMs (41,910)	REGs	Mult. (112)	Freq. (MHz)	Neurons	FPGA (μs)	Cycles
19,429	6519	18	100	14	1.63	163

3.3 CHAPTER FINAL CONSIDERATIONS

The efficient hardware implementation of nonlinear MHE with ANN presented on the paper accepted on IFAC 2020 (Brunello et al. 2020) was described in-depth, showing that its FPGA 64-bits 14 neurons architecture has a speed-up of approximately 30 times that of its ARM counterpart. This implementation is then further improved by reducing its representation to 32-bits and removing the CORDIC exponential module while maintaining almost the same speed-up as before. The next chapter will describe the attempt made to further expand upon the idea of moving-horizon neural approximation, approximating the combination of both the MHE and the MPC as a single ANN.

4 APPROXIMATING MOVING-HORIZON ESTIMATION AND MODEL PREDICTIVE CONTROL

In this chapter, we will present a new contribution on approximating a combination of both the MHE and the MPC using neural networks, specifically the RBF. Following the same pattern as Chapter 3, we first characterize the idea, its conception, software implementation, failures, and iterations. In the second section, we present the results of the main iterations and compare the noise each one can handle, also specifying the parameters used in each experiment.

4.1 METHODOLOGY

It was initially planned to progress the approximate MHE one step closer to real-time implementation by running it on a didactic inverted pendulum testbench. However, this plan was foiled by unpredictable circumstances such as the COVID-19 pandemic. We then needed a different final contribution that can be done in simulations and further improves the possibilities and usability of the MHE in real applications and industry.

Some of the potential contributions conceived were: using a metaheuristic algorithm in place of the current optimization one or the k-means algorithm; switch the model system for one that has stricter timing requirements; try other neural network architectures. While the first and third ideas are interesting they do not have much to improve on the present design. The first idea is something that is researched to make the optimization faster in a real embedded MHE, since we take the offline optimization and approximate MHE approach it is better to use the best optimization and clustering algorithms as the time it takes is not an issue. The third idea has similar problems, as the decision to change the RBF for other NN architectures would most likely mean an NN with more complexity and number of neurons.

On the other hand, the second idea was the addition most recommended from the article peer review. The reviews suggested tests on more complex systems, such as a drone. Nevertheless, the current input would not be enough to test the state estimation in the drone case, it is then mandatory to implement a control system to stabilize and move the drone. Hence, we consider the MPC since it is one of the most used and researched controllers, both by itself and more recently utilized in conjunction with the MHE as shown in Chapter 2.

The MPC presented in (Sampaio 2018) is used as a basis along with its inverted pendulum case study. It is then needed to combine and adapt both the MPC and MHE programs to work on a step-by-step basis instead of the full simulation batch previously employed as shown by Fig 4.1. After successfully combining both programs and verifying that they were working individually and also capable of communicating with each other, some crucial problems were found and among

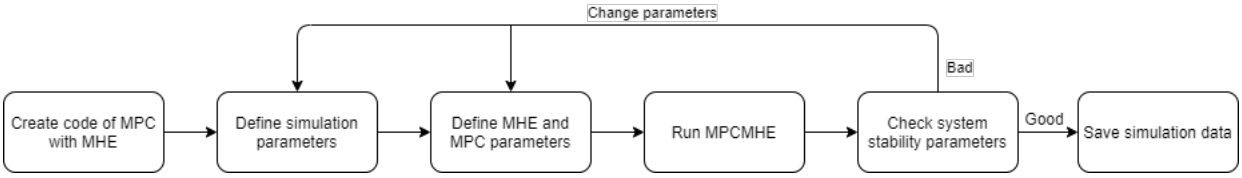


Figure 4.1: Step-by-step of the MHEC implementation.

those, there were two main issues. The first one is that the control was not capable of stabilizing the system when using the MHE to estimate the missing states with some noise. And the second is that the control system stopped trying to stabilize after some unsuccessful tries.

Many attempts were made to resolve these problems, such as tunning the MPC parameters, changing the noise levels, varying the horizons of both the MPC and the MHE, decrease sampling time, and adjusting the MPC cost function. Tunning the MPC parameters greatly altered the results, however even with an extensive search varying all 4 parameters in Q, 4 in Q_f and R, it was not capable of doing the swing-up motion and stabilizing the pendulum. It was then decreased the noise input on the system state reading to evaluate at which point it would work, at each alteration of noise it was necessary to re-tune the MPC, as certain weights obtained better results in noise-rich data while others did the opposite. Not even the MPC without the MHE was capable of the swing-up and stabilizing the system with the initially tested noise of 0,02 and 0,1 for the linear and angular positions, although it can resist more noise.

Another situation observed when combining the MPC and MHE was that when increasing the horizon size of the MHE the results worsened for the MPC. To counter that a new parameter for the MHE was created, a weight that indicates how much the cost function will value newer estimations over older ones. This parameter was varied between 1 and 1,5, the first being the standard MHE.

After expending a lot of effort trying to resolve this issue without success, it was in our best interest to comply with extremely low noise or forgo the noise entirely. Nevertheless, the second main problem was solved by altering the way the cost function dealt with the angles, a program was made to limit the read data of the angular position between $-\pi$ and π. This problem happened because as the control tried to swing-up the pendulum to then stabilize, the angular position kept increasing and as such, the MPC stopped working altogether as doing differently would only increase the error.

Thereafter, it was possible to find a tunning that, with low noise and the angle issue fixed, was able to stabilize the system for some seconds, but not forever. Taking it as an initial *good enough* result we decided to take it to the next level shown in Fig 4.2, which is the approximate version using an RBF neural network. However, this time it will approximate the result obtained by both the MPC and MHE as one single network, having its input be the one from MHE and the expected output by the control signal from the MPC. The construction and training of the NN are the same as previously discussed in subsection 3.1.1, the only changes being using the control signal as the expected output and that the R-squared is not a good indicator anymore. This happens

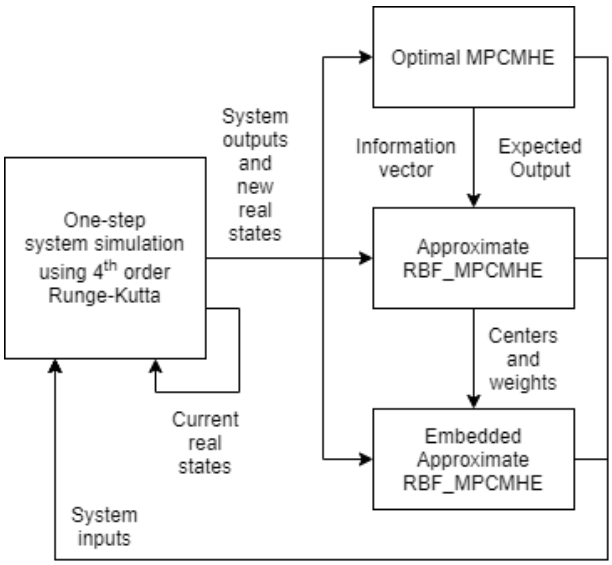


Figure 4.2: Dataflow of the AMHEC implementation.

because while the NN is approximating the control signal given by the MPC, our real intention is to stabilize the system. In (Sampaio 2018) they show that even with an R-squared result better than 0,95 the system can still not be stabilized, and sometimes with values worse than that they can. So, to evaluate the NN, it will be indispensable to test it in the model system and analyze its results. To aid in the evaluation we employ two methods, one is the MSE between the angular position of the expected stabilization by the approximated one. The other is called **stable time** and as the name previews it indicates how much time the system was able to be stabilized, it starts with the same value as the number of samples in the simulation, and for each iteration that the system is considered stable it decreases its value. Yet, even after thousands of different tunings, it was not possible to find an RBF neural network capable of swinging-up and stabilizing the inverted pendulum, all of the results were not even close to being capable of doing the swing-up, the control signal was all too small for the pendulum to oscillate enough.

4.1.1 Switch MHEC

After contemplating the results and some failed test trials, the next considerable idea was the prospect of dividing the problem and consequently the controller into two main parts, the non-linear swing-up, and the stabilization. The basis for this decision is that the weights desired on the swing-up were different from the ones needed on the stabilization phase, the former dealing with large variations on angular position while the latter must maintain this variation to a minimum. With this plan in mind, we must have two different tunings for each control, the swing control must be capable of oscillating the pendulum from rest and stabilizing it for a bit, and the stabilizing control must be capable of maintaining the pendulum inverted while the cart moves side-to-side. Next, we assess the results from each of these tunning methods and choose the seemingly best controls. Then, we implement a *switch* function that changes the weights of the MPC during the

simulation when the angular position reaches a certain threshold. Finally, we run some simulations with this switching-control to check if there were improvements in the results.

As we now have two MPC controls to tune, the time needed to do this also doubles. To decrease the time spent tuning we further inspected the effects of each of the MPC parameters. In this manner, we realized that the way we were employing the Q and Q_f one of them was made redundant, as such we removed the Q_f parameter to simplify tuning the control. The Q_f is used as an extra weight pulling the trajectory to the desired endpoint, but this becomes redundant because the trajectory being used is a step from π to 0, which is the desired endpoint. Another parameter that needed consideration was the MHE horizon size, as a smaller window is better for the MPC but gives less information for the NN training. As such, we initially employed a horizon size of 5 however, to gain better results on the NN training phase size of 10 was reached after experimentations and evaluation on both results of the MPC and the NN.

With this new configuration, the MHEC was able to endure some more noise, but still not enough, and stabilize the system for an undetermined time duration. With these promising results, we once again try to construct and train the approximate NN, but now it will need what initially seems to be two different networks. To keep the initial concept of low complexity and computational cost we plan to have both neural networks possess the same number of neurons, this way with only altering the value of the weights and centers it is possible to dynamically switch the NN being used. We then train the NN by following the steps shown in Fig 4.3. These steps are very much like the ones in Fig 3.3 but here we train two neural networks and use different indicators to assess them.

The NN that maintains the system stabilized was possible to train, but the non-linear component of the swing-up motion was too complex for the RBF neural network chosen by this method. We then adjust our method to first try to find an RBF network capable of successfully doing the swing-up motion and then using its configuration train another for the stabilization. This, however, was also a failure. Most results were similar in that they could not accompany the vast oscillation that was needed in the input signal, meaning that while the NN oscillate the control signal it did not have the necessary amplitude to have the desired effect. Some of the configurations were capable of making the swing-up motion but at the cost of extrapolating the pre-determined input and state limits, which immediately backfires by having the network and the system going out of control.

4.1.1.1 Other neural networks and SVM

To serve as a comparison we decided to try and approximate the swing-up by using other types of neural networks. In this case we tested in some of the already implemented types on Matlab, using functions as *feedforwardnet*, *fitnet*, *cascadenet*. Initially, we tested by using a single hidden layer similar to the RBF composition and complexity, this was unsuccessful. Next, we tested using multiple hidden layers and several neurons, though not in an extensive manner, this test was also met with failure. The same problems observed on the RBF were also present on these NN architectures.

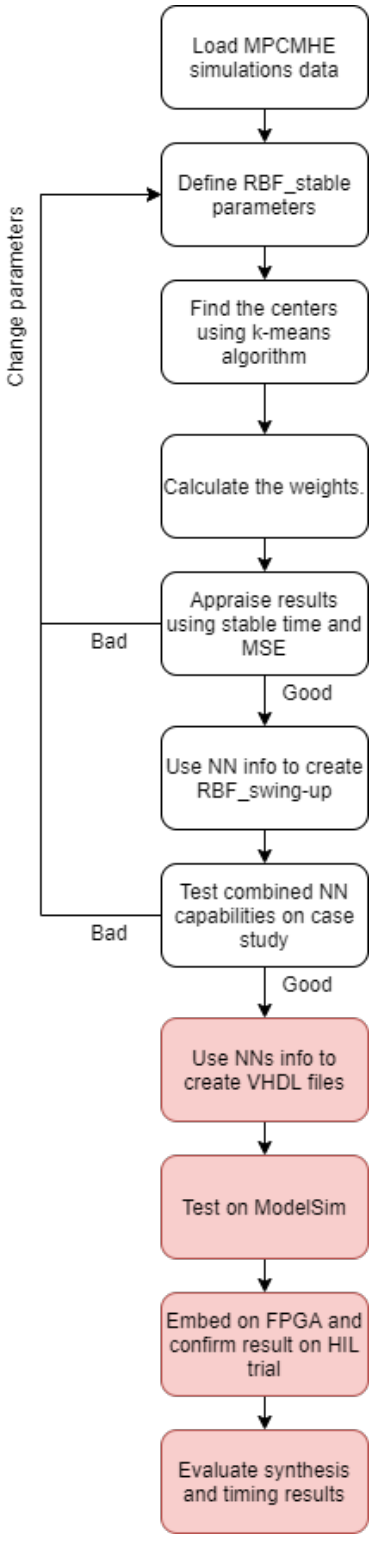


Figure 4.3: Step-by-step of the AMHEC implementation.

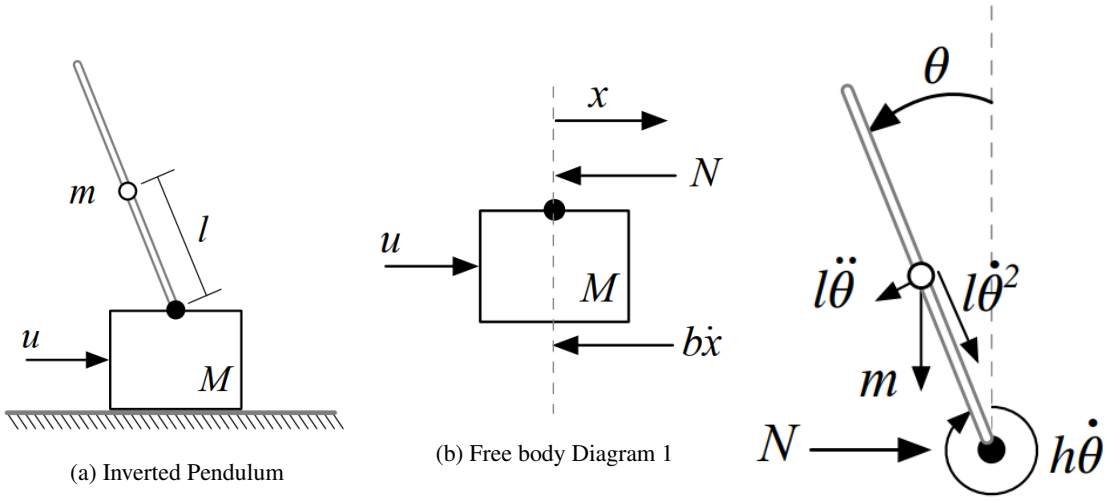
As a last resort, we tested Support Vector Machines (SVMs) trained using the NIOTS program used in (Santos et al. 2017). Fortunately, they were able to fit the swing-up function and stay within the pre-determined boundaries. This proves the concept that it is possible to approximate the non-linear swing-up motion by machine learning, however, due to time constraints this work was not capable of finding the solution to solving the problems observed on the RBF networks. It was also not possible to finalize the steps marked as red on the Fig 4.3, leaving it as a future work to be finished.

In the next section, we will show the most relevant graphics, tables, and data obtained through the realization of this methodology, as well as discuss and analyze its results.

4.2 RESULTS AND ANALYSIS

In the present section, we display the results for the MHEC implementation in Matlab using the same approximate solutions of the RBFMHE. To this end, we use a simulated case study and analyze its results.

4.2.1 Case Study



(a) Inverted Pendulum

(b) Free body Diagram 1

(c) Free body Diagram 2

Figure 4.4: MHEC case study

This case study employed was described in (Alaniz 2004) and then used by (Mercieca e Fabri 2012) to test many techniques of predictive control. The optimal MPC code applied and modified in this dissertation is from (Sampaio 2018), which also adopts this model system as a case study to illustrate its development. Figure 4.4 presents the model system along with the free body diagrams, while the Equations 4.1 describe its non-linear dynamics. The equations are expressed in terms of the state variables x, $\dot{x}$, θ, and $\dot{\theta}$, which are, respectively, the

cart position on the rail, its linear velocity, the pendulum inclination angle, and its angular velocity. The constants present in Table 4.1 are M the mass of the cart, m the uniformly distributed mass of the pendulum's body, l is half the length of the pendulum, b is the surface friction, h is the rotation friction damping coefficient, g is the gravity's acceleration, and u represents the control force applied to the cart.

$$\begin{aligned} \ddot{x} &= \frac{1}{M+m} * (u - b\dot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta), \\ \ddot{\theta} &= \frac{3}{4ml^2} * (mgl\sin\theta - ml\ddot{x}\cos\theta - h\dot{\theta}). \end{aligned} \tag{4.1}$$

The simplified cost function utilized is:

$$J(\mathbf{x}(k), \mathbf{u}(k)) = \sum_{i=1}^{N-1} \parallel \tilde{\mathbf{y}}(k+i) - \mathbf{y}_{ref}(k+i)(k) \parallel_Q^2 + \sum_{i=0}^{N_c-1} \parallel \tilde{\mathbf{u}}(k+i) - \mathbf{u}_{ss} \parallel_R^2 + \tilde{\mathbf{x}}(N) - \mathbf{x}_{ss} \parallel_{Q_f}^2 . \tag{4.2}$$

To be able to control the system it is imperative to define the parameters related to its dynamics, such as the sampling time (T_a), the physical boundaries of the states (x_{max}, x_{min}) or of the control signal (y_{max}, y_{min}) and the limits of the actuator (u_{min}, u_{max} and Δu_{max}). The parameters specific for the control must also be defined, such as the prediction horizon (N), the control horizon (N_c), the weighting matrix of states (Q), the weighting matrix of steady-states (Q_f), and the weighting matrix of the control signal (R). These are also shown in the Table 4.1.

Table 4.1: Case study 2 parameters.

Control parameters	Value
T_a	100 ms
x_{min}	-0.5 m
x_{max}	0.5 m
u_{min}	-50 N
u_{max}	50 N
Δu_{max}	50 N
N	20
Nc	10
System constants	**Value**
m	7.3 Kg
M	14.6 Kg
g	9.81 m/s^2
l	1.2 m
b	14.6 Kg/s
h	0.0136 $Kg * m^2/s$

4.2.2 Optimal MHEC

After defining the model system that will be used, now we start testing and iterating upon the model predictive controller that will be combined with the moving horizon estimator. Table 4.2 presents the parameters with the best results found among those tested, they also will be the ones utilized in the following figures. In Appendix I there are figures which show how the system behaves when altering some of the individual parameters such as the MHE's horizon and weight, and the MPC's prediction and control horizons. For a more in-depth analysis on how the matrices Q and Q_f affect the system look upon (Mayne et al. 2000).

Table 4.2: Controller parameters.

Controller	MPC	MHEC	MHEC_stable	MHEC_swingup
Q	1 400 875 10	1 400 875 10	100 100 700 700	700 400 1000 100
Qf	1 100 1000 100	1 100 1000 100	-	-
R	1	1	1	1
H	-	10	10	10
w	-	1.1	1	1

The testing also made it possible to identify the maximum noise value that each controller employ can handle without becoming unstable, we call this the limit noise. In Table 4.3 it is stated the value for each of the noises that will appear on the following figures, the noise considered as high is one that not one of the controllers was capable of stabilizing and it is the same previously used on the RBFMHE section of this dissertation. The Figures 4.5 will show the resulting angle θ that the controller obtained for each level of noise, followed by the Figures 4.7 with two figures demonstrating visually the amplitude of each state's level of noise.

Table 4.3: Noise table.

Type of noise		High noise	Limit noise MPC	Limit noise MHEC	Limit noise Switch MHEC
Noise value	Position (cm)	5e-2	2e-2	8,59e-4	5e-3
	Angle (rad)	1e-1	3e-2	8,59e-4	5e-3

First we notice in Figures 4.5, 4.6 and 4.7 that the higher the noise, bigger the delay on the control to stabilize the system. When the system has no noise all the controllers have no problem stabilizing it seamlessly. However, in the optimal MPC's limit noise we note that even when the pendulum is near its unstable equilibrium point the control is struggling to maintain it. For the high noise, the MPC was capable of preserving the stability for more than a few iterations, but the Switch MPCMHE tried to arrive at the stability more times quicker. The graphs in Figure 4.5 compare the results of each controller for each level of noise, while Figure 4.6 does the opposite and compares the results from all the controllers for each type of noise, remembering that the limit noise for each controller is different.

In Table 4.3 and Figure 4.7 we can also recognize that the system and control handle the noise on the linear position better than in its angular position. When analyzing the high noise mean

on percentile the linear noise becomes 10% of its maximum possible value, while the angular noise becomes close to 3%. It is important to reiterate that these values are the mean of a normal probability curve, that is why there are values much higher than those in Figure 4.7. This figure also shows that together with Table 4.3 that there is a difference of an order of magnitude between the limit noise of the MPC's to the Switch MHEC and two orders of magnitude to the MHEC, clearly demonstrating an improvement in Switch MHEC's part on handling noise better than the pure MPCMHE.

4.2.3 Approximate MHEC

The first attempt to train and approximate was made using the MHEC, which its best results were not promising and are presented in Figure 4.8. From this attempt, we learned two main points, that a bigger MHE's horizon improved the training as there is more information. And that the non-linear swing-up and the stabilization were almost polar opposites, they first need to employ high amplitude signal in sequence while the latter requires a more fine and low amplitude signal control. This brought the idea of separating the control in two for each part of the process, thus creating the version Switch MHEC here applied.

We then tune and create a training dataset for each of the controls. At first, we tried to train the stabilizing control, and using its configuration, train the swing-up approximation. Employing the same steps from the approximate MHE to determine, we arrived at the architecture presented in Figure 4.9, with 24 neurons and 34 inputs, the new input refers to the control linear position reference. With this NN we acquire the Figures 4.10 and 4.11, the former displaying the results of the stabilization approximation moving the cart from one position to another maintaining the pendulum stable. This NN approximation gives a smoother control signal curve and its angular position has a greater variation than expected, but it is still capable of keeping the pendulum stable during the whole motion. While the latter figure exhibits the results of the approximate swing-up control with its expected results, though this is not capable of its desired function, which is to swing the pendulum until it arrives close to $\theta = 0$. From Figure's 4.11 that the approximation's control signal is not capable of dealing with the high amplitude and frequency variation, and as a result, its output is smoothed too much to decrease the mean-squared error.

As this method of training did not succeed, we decided to capitulate on the idea of it working on any position on the rail. Thus, the training data is made from only a swing-up simulation from the center, trying to force the neural network to overfit the dataset. With this attempt, we attained the results seen in Figures 4.12 and 4.13 which are the best for the mean-squared error and the stable time cost function, respectively. Both produced better results that come closer to the expected behavior, yet both also, unfortunately, extrapolate the system limits in both control signal and cart position. This illustrates one of the down points of this type of approximation, the loss of the capability to impose the system restrictions on the controller.

Table 4.4: SVM parameters.

Modelo	Kernel	C	Gama	Epsilon	MSE	Total SV	RHO
LibSVM	RBF_kernel	8.960187	0.240270	1.102487	21.961317	145	-0.959801

4.2.4 Other neural networks and SVM

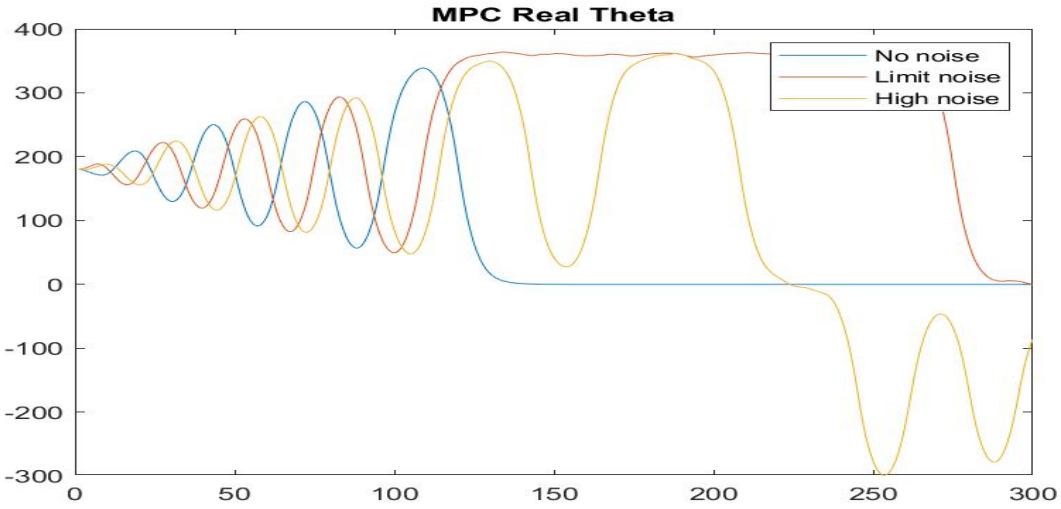
Finally, we tested other architectures of neural networks as an alternative to the RBF, to verify if the difficulty steemed from the architecture. First, we confirm that Matlab's feedforward is capable of approximating the stabilizing control using the same parameters as Fig 4.9. Figure 4.14 verifies this, but it has a worse result than its RBF counterpart, its control signal is rougher and has a wider variation in θ during its movement. Training and testing it for swing-up control also presents similar results to the RBFs and Figure's 4.8.

Next, we analyze it with multiple hidden layers and a varying number of neurons on each of them. One example can be seen in Figure 4.15 which has 3 hidden layers and [31,15,5] neurons on each layer. Using this configuration and the functions for training a feed-forward and a fitnet, we acquire Figures 4.16 and 4.17. However, both of these contain the same problems verified on the RBF's results, extrapolation of the pre-determined limits, and they also have a worse θ swing variation.

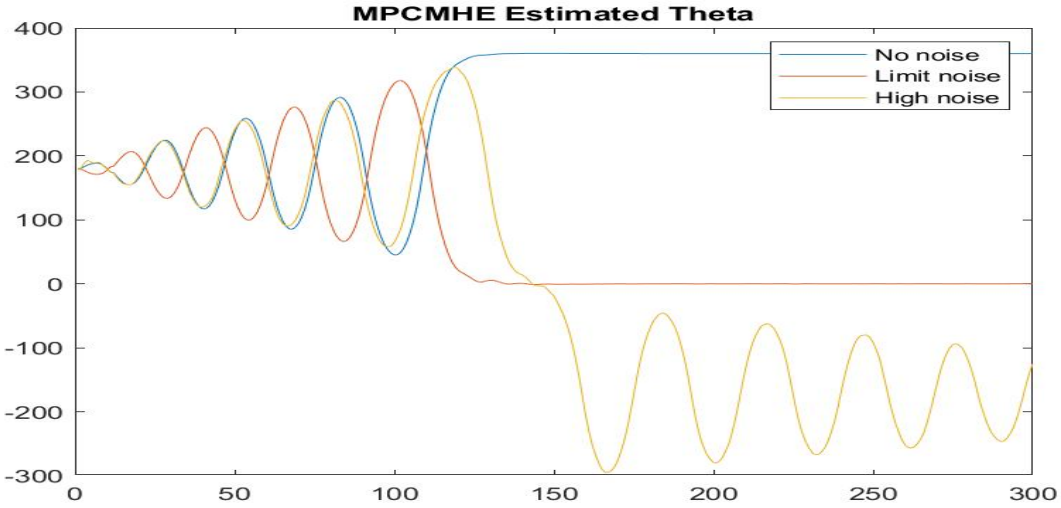
As a last resort to check that a neural network is capable of fitting this curve, we employ the **NIOTS II** program for training support vector machines described in (Santos et al. 2017). The program is executed following the configuration shown in Figure 4.18 and trained using the overfit dataset. It is then necessary to test and choose the best fit from the results obtained from its Pareto's frontier. With one of its results, with the parameters presented in Table **??** we obtain Figure 4.19. It shows that the SVM was capable of overfitting the desired signal, it also stayed within the desired boundaries. As such, it serves as a proof of concept that neural networks can in fact fit this signal, just that the RBF architecture used is still lacking something unknown.

4.3 CHAPTER FINAL CONSIDERATIONS

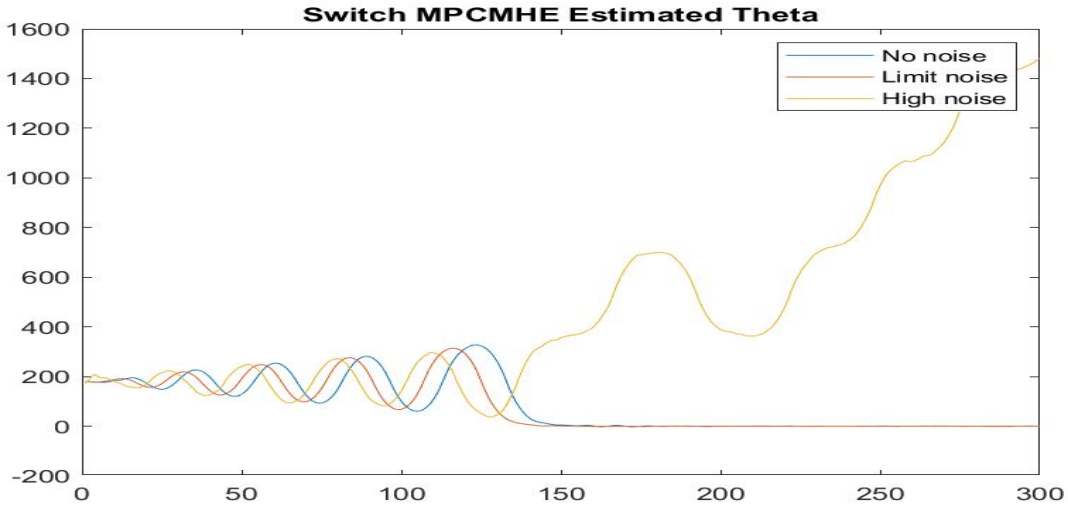
The implementation of the MHEC was successful and it was possible to increase its noise robustness by separating the control into two differently weighted parts and switching between them when necessary. The results from the neural approximation were terrible for the simple MHEC, but with the switching mode, it was possible to approximate the stabilization part of the control. On the other hand, the swing-up control faced many difficulties, even with a purposefully overfitted ANN it was not capable of properly approximating the control function without overstepping its boundaries. Lastly, an SVM approximation attempt was made successfully for the swing-up control, which suggests it is a promising alternative, but still needs more research. The next chapter will congregate all of this dissertation results and conclusions, suggesting future improvements and alternatives paths to develop from.



(a) MPC

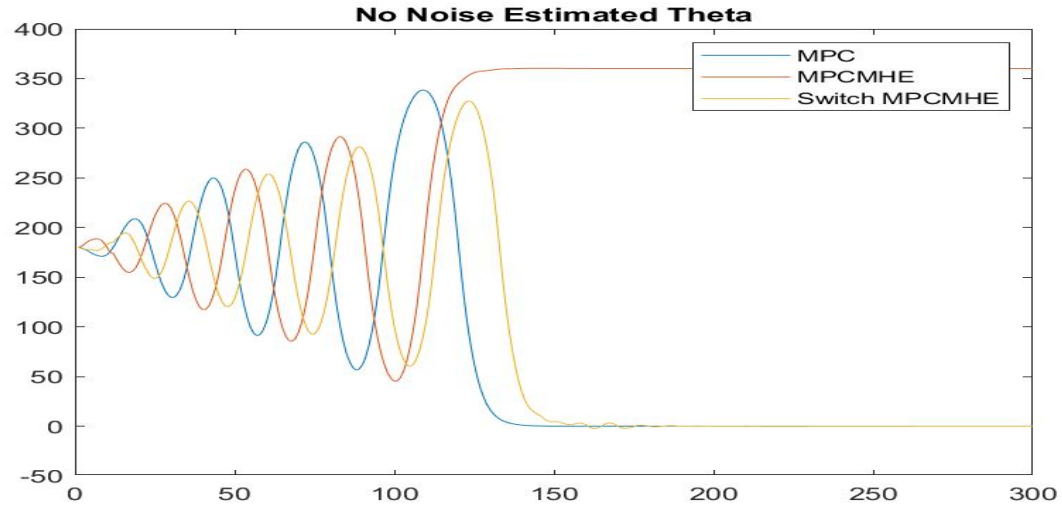


(b) MHEC

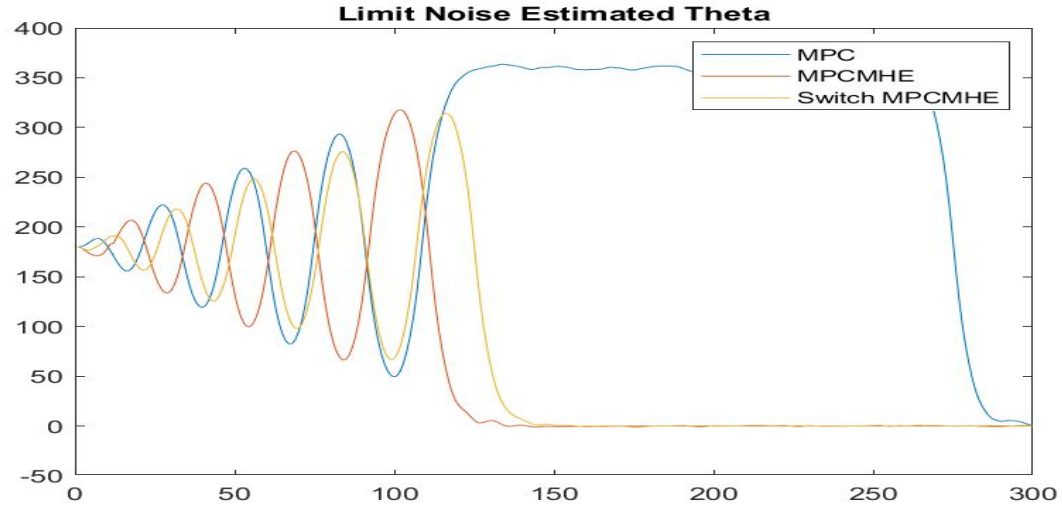


(c) Switch MHEC

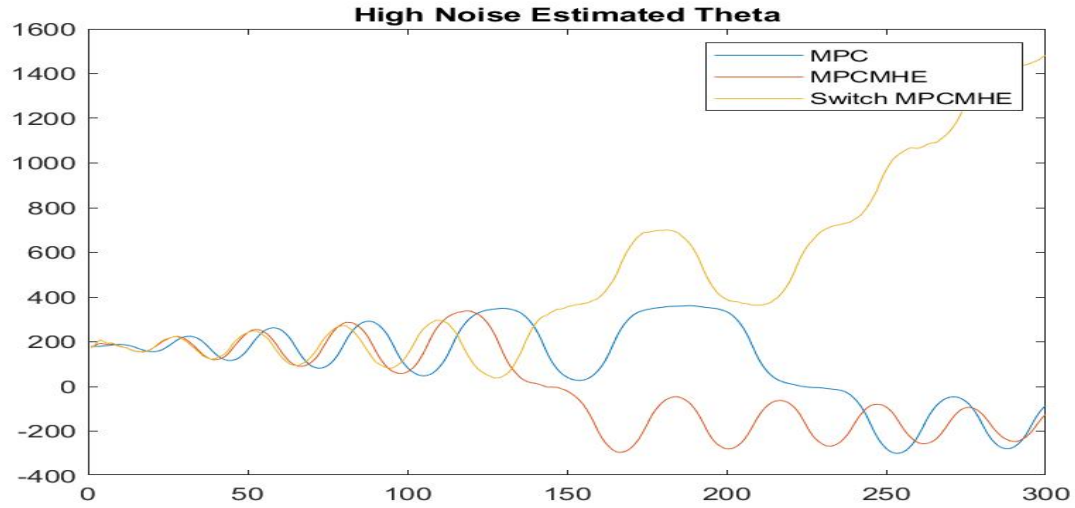
Figure 4.5: Theta value of the system for multiple noises.



(a) No noise

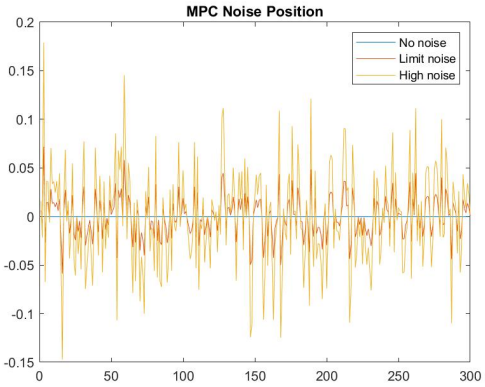


(b) Limit noise

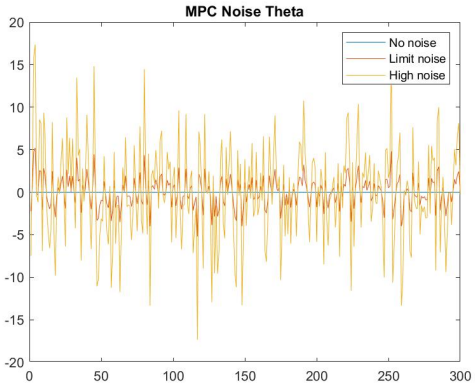


(c) High noise

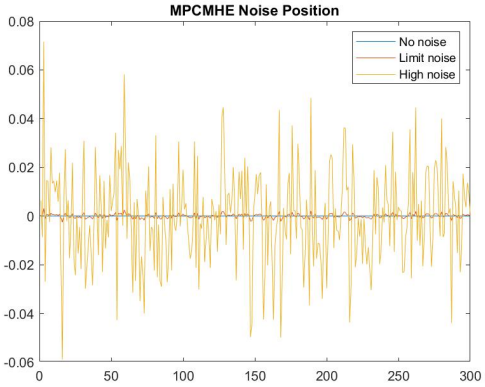
Figure 4.6: Theta value of the system for multiple controls.



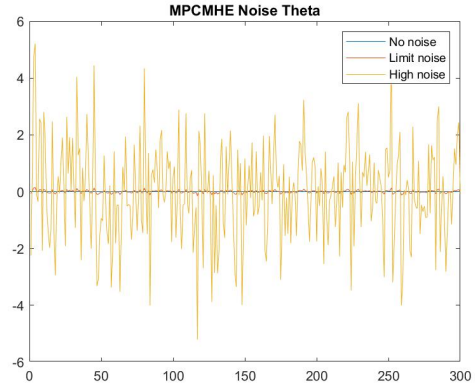
(a) On MPC's Position



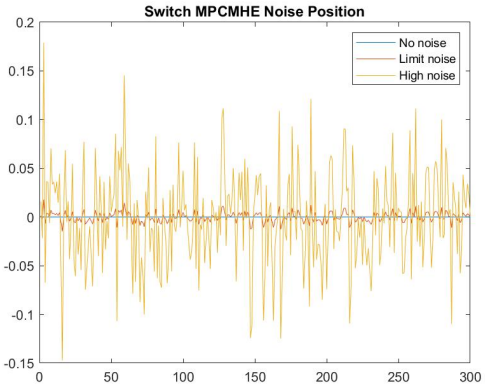
(b) On MPC's Theta



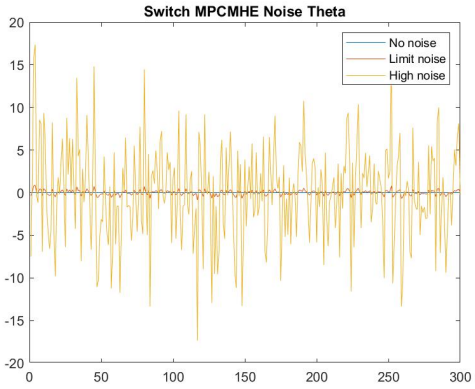
(c) On MHEC's Position



(d) On MHEC's Theta

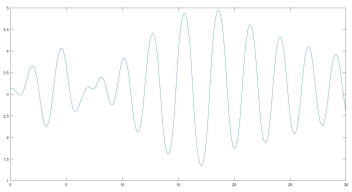


(e) On Switch MHEC's Position

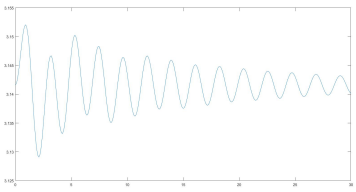


(f) On Switch MHEC's Theta

Figure 4.7: System noise



(a) Using SE



(b) Using ST

Figure 4.8: Best results on AMHEC

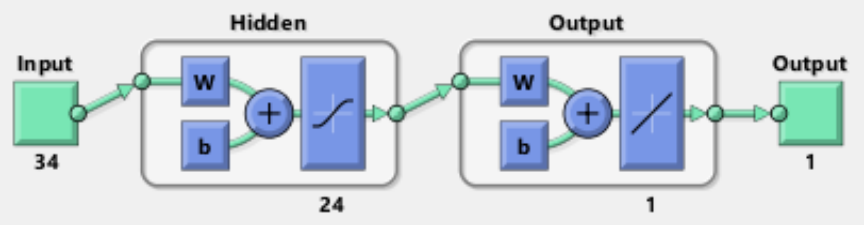
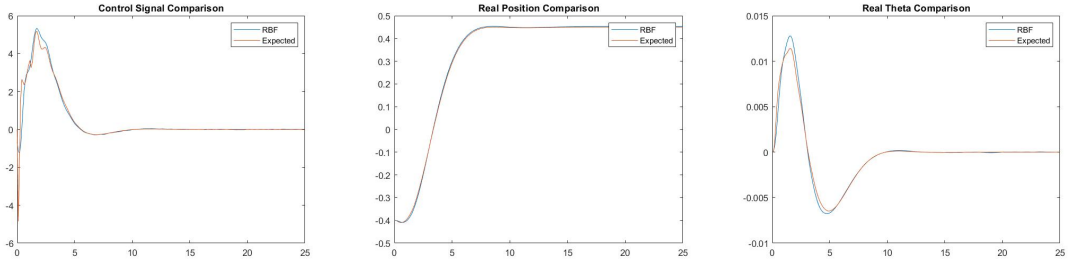
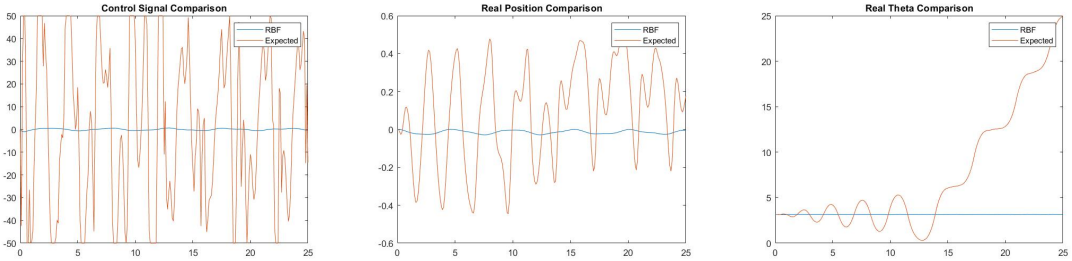


Figure 4.9: Example of Matlab neural network architecture used.



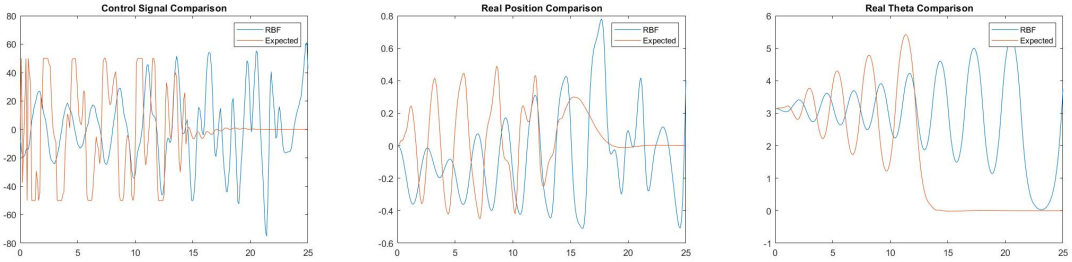
(a) Control (b) Position (c) Theta

Figure 4.10: Best results on using the RBF on approximating the stabilization control.



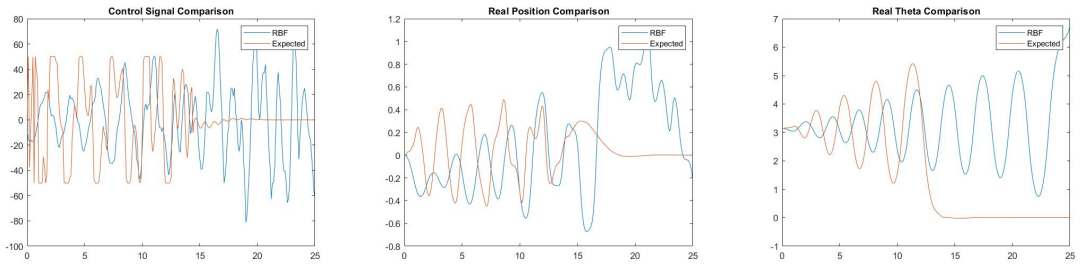
(a) Control (b) Position (c) Theta

Figure 4.11: Results on using same configuration on approximating the swing-up control.



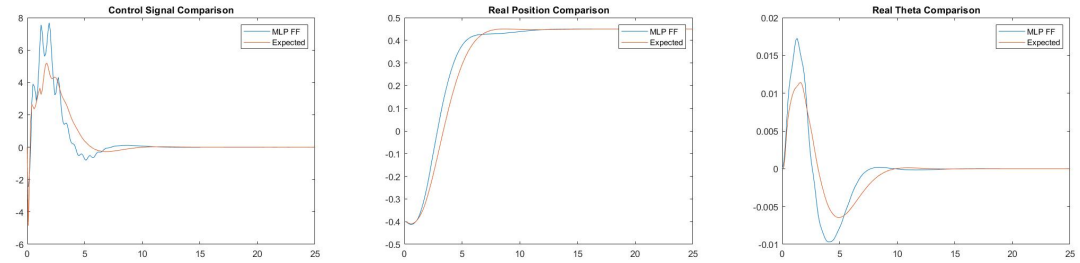
(a) Control (b) Position (c) Theta

Figure 4.12: Results on using the best SE swing-up control training for overfit.



(a) Control (b) Position (c) Theta

Figure 4.13: Results on using the best ST swing-up control training for overfit.



(a) Control (b) Position (c) Theta

Figure 4.14: Example of results on using the feed-forward on approximating the stabilization control.

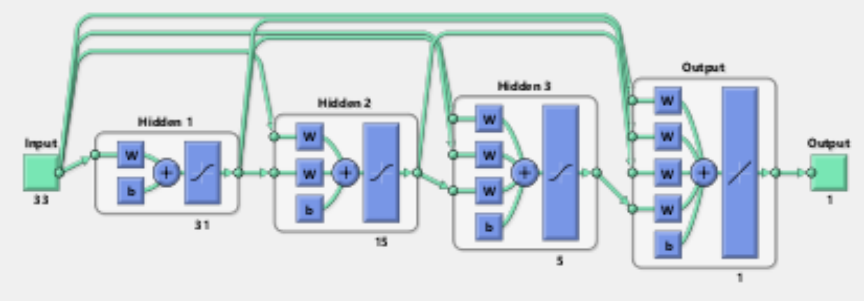
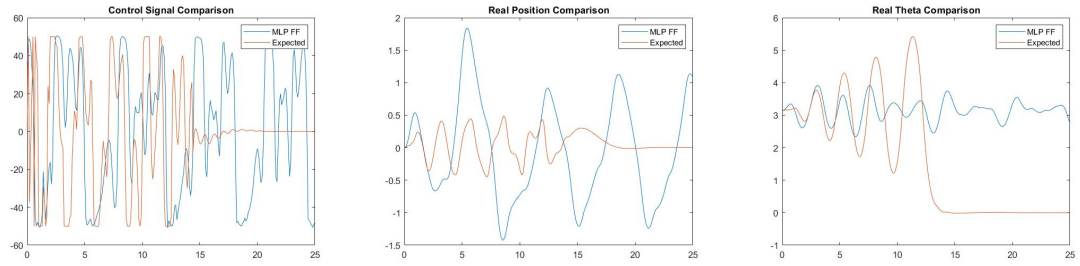
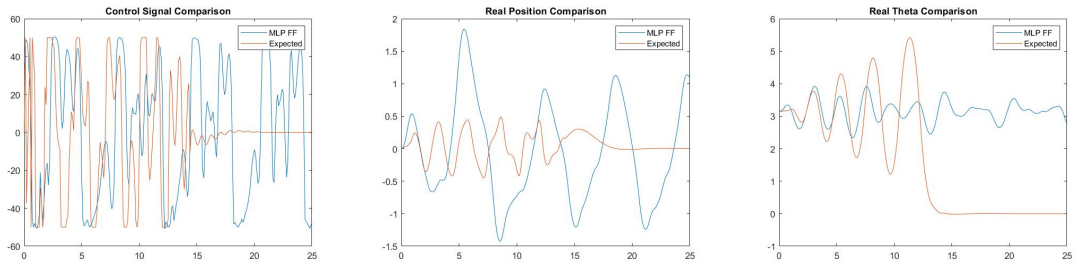


Figure 4.15: Example of Matlab neural network architecture used.



(a) Control (b) Position (c) Theta

Figure 4.16: Example of results on using the feed-forward on approximating the swing-up control.



(a) Control (b) Position (c) Theta

Figure 4.17: Example of results on using the fitnet on approximating the swing-up control.

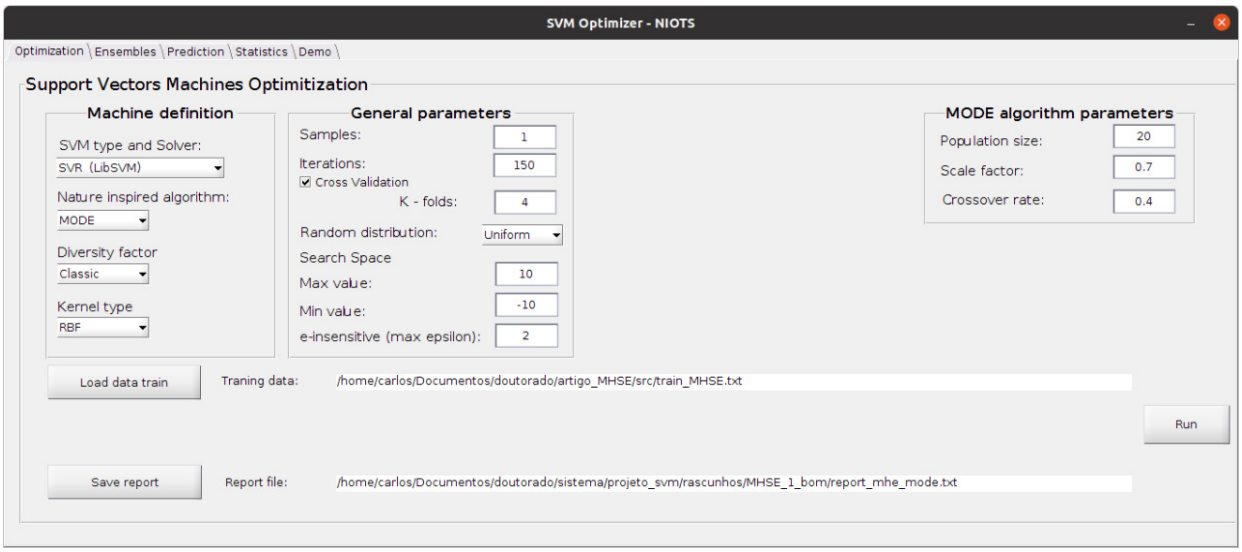


Figure 4.18: NIOTS II configuration.

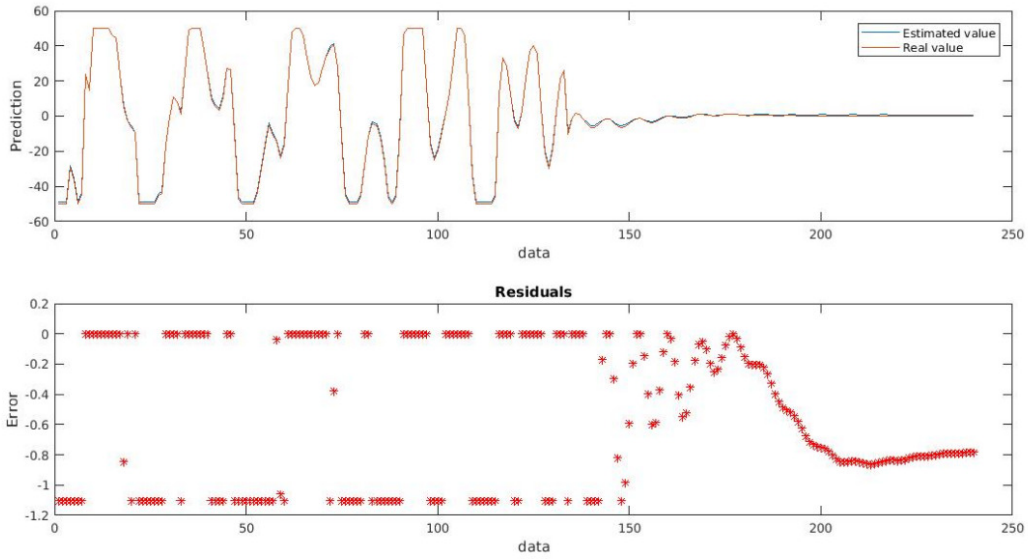


Figure 4.19: Control signal using an SVM compared to the expected result.

5 CONCLUSION AND FUTURE WORK

In this dissertation we presented two main approaches to accelerate the MHE method and facilitate its usage in real-time applications. Both approaches involve approximating the function results by employing RBF ANNs. Also using the inherent parallel structure of ANNs and FPGAs, what is a synergistic combination since the FPGAs allows the ad-hoc implementation of fast parallel computations directly on hardware. However, while one approach cover only the MHE, the other encompass both MHE and MPC.

In the first case, the implications of the work herein presented confirm the expectations of the approximate filter presented in (Alessandri, Baglietto e Battistelli 2008) and (Alessandri et al. 2011), for the first time implemented in hardware in the present work: it is possible to calculate the estimation offline and accurately approximate the state estimates by means of an ANN online, which performs very fast due to its inherent parallel architecture and enabled by its direct hardware implementation. Being further improved since its article publication (Brunello et al. 2020) by solving the precision problem and reducing its representation from 64 to 32-bits, also further simplifying it by removing the CORDIC module from the floating-point unit as it was using only the Taylor series approximation method.

In spite of this, the second case was not as successful. The combination of MHE and MPC worked despite its inherent loss of robustness to noise from having to estimate two states. However, it was not possible to obtain a ANN that approximates both its response to the non-linear and linearizable parts of the system. This resulted in a development of two separated controls, one for each stage with the vision of a dynamic weight switchin implemetation on hardware. The new approach produced better results, being capable of approximating the control response for the linearizable step with 24 RBF neurons. Though, the approximations of the non-linear step did not work, be it due to overstepping the system limits or its control signal being too smoothed out. Also, as it was not found a RBF approximation for the non-linear step, we could not implement and test the dynamic weight switching in hardware. We tested the possibility of the problem being the RBF architecture and tested two already implemented ANN architectures on Matlab, the feed-forward and the fitnet, yet both still did not correctly approximate the non-linear control. As a last attempt, we utilized the NIOTS program (Santos et al. 2017) to train SVMs to approximate the non-linear control signal. We finally obtained a result that did not exceed the system boundaries. Unfortunately, due to time constraints it was not possible to further develop and research these results.

5.1 FUTURE WORK

Even though the methods and architectures shown in this dissertation have been applied to some degree of success, there are still many improvements and variations possible. Some possibilities and suggestions with no specific order or ranking are:

5.1.1 Approximate Moving-Horizon Estimation

- Test other types of clusterization algorithms beyond the k-means;
- Compare the optimization algorithm used on the MHE with metaheuristics approaches;
- Test many types of radial basis functions;
- Test other types of ANNs architectures;
- Propose a didactic test-bench with the inverted pendulum (Magana e Holzapfel 1998) for receding-horizon estimation and control in FPGAs with approximate solutions and ANNs (Zoppoli et al. 2020);
- Test the approximate MHE implemented in FPGAs in real-world systems with fast dynamics (Ayala, Rakotondrahe e Coelho 2018).

5.1.2 Approximate MHEC

- Solve the non-linear RBF approximation problem;
- Embed and simulate the system in the FPGA;
- Implement and test the dynamic weight switching in a HIL simulation;
- Explore ways to impose the system restrictions on the ANN training, one possibility being the set projection discussed in (Alessandri et al. 2011);
- Test other types of RBF cost functions;
- Approximate the whole MHEC function using SVMs.

BIBLIOGRAPHY

Abdollahpouri, Takács e Rohaľ-Ilkiv 2017 ABDOLLAHPOURI, M.; TAKáCS, G.; ROHAĽ-ILKIV, B. Real-time moving horizon estimation for a vibrating active cantilever. *Mechanical Systems and Signal Processing*, v. 86, p. 1–15, 2017. ISSN 0888-3270. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0888327016303636⟩.

Alaniz 2004 ALANIZ, A. *Model predictive control with application to real-time hardware and guided parafoil*. Tese (Doutorado) — Massachusetts Institute of Technology, 2004.

Alessandri, Baglietto e Battistelli 2008 ALESSANDRI, A.; BAGLIETTO, M.; BATTISTELLI, G. Moving-horizon state estimation for nonlinear discrete-time systems: New stability results and approximation schemes. *Automatica*, v. 44, n. 7, p. 1753–1765, 2008.

Alessandri et al. 2011 ALESSANDRI, A.; BAGLIETTO, M.; BATTISTELLI, G.; GAGGERO, M. Moving-horizon state estimation for nonlinear systems using neural networks. *IEEE Transactions on Neural Networks*, v. 22, n. 5, p. 768–780, May 2011.

Allgöwer et al. 1999 ALLGÖWER, F.; BADGWELL, T. A.; QIN, J. S.; RAWLINGS, J. B.; WRIGHT, S. J. Nonlinear predictive control and moving horizon estimation — an introductory overview. In: FRANK, P. M. (Ed.). *Advances in Control*. London: Springer London, 1999. p. 391–449. ISBN 978-1-4471-0853-5.

Andersson e Thiringer 2018 Andersson, A.; Thiringer, T. Motion sensorless ipmsm control using linear moving horizon estimation with luenberger observer state feedback. *IEEE Transactions on Transportation Electrification*, v. 4, n. 2, p. 464–473, 2018.

Andersson et al. 2019 ANDERSSON, J. A. E.; GILLIS, J.; HORN, G.; RAWLINGS, J. B.; DIEHL, M. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, v. 11, n. 1, p. 1–36, 2019.

Aström e Murray 2008 ASTRÖM, K. J.; MURRAY, R. M. *Feedback systems: an introduction for scientists and engineers*. Princeton, New Jersey: Princeton university press, 2008.

Ayala et al. 2016 Ayala, H.; Sampaio, R.; Muñoz, D. M.; Llanos, C.; Coelho, L.; Jacobi, R. Nonlinear model predictive control hardware implementation with custom-precision floating point operations. In: *2016 24th Mediterranean Conference on Control and Automation (MED)*. [S.l.: s.n.], 2016. p. 135–140.

Ayala et al. 2015 AYALA, H. V. H.; HABINEZA, D.; RAKOTONDRABE, M.; KLEIN, C. E.; COELHO, L. S. Nonlinear black-box system identification through neural networks of a hysteretic piezoelectric robotic micromanipulator. *IFAC-PapersOnLine*, v. 48, n. 28, p. 409 – 414, 2015.

Ayala et al. 2017 AYALA, H. V. H.; MUñOZ, D. M.; LLANOS, C. H.; COELHO, L. S. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. *Control Engineering Practice*, v. 60, p. 124 – 132, 2017.

Ayala, Rakotondrahe e Coelho 2018 Ayala, H. V. H.; Rakotondrahe, M.; Coelho, L. d. S. Modeling of a 2-DOF piezoelectric micromanipulator at high frequency rates through nonlinear black-box system identification. In: *American Control Conference*. Milwaukee, Wisconsin, USA: [s.n.], 2018. p. 4354–4359.

Bae e Oh 2017 BAE, H.; OH, J.-H. Humanoid state estimation using a moving horizon estimator. *Advanced Robotics*, Taylor Francis, v. 31, n. 13, p. 695–705, 2017. Disponível em: ⟨https://doi.org/10.1080/01691864.2017.1326317⟩.

Bennett 1996 Bennett, S. A brief history of automatic control. *IEEE Control Systems Magazine*, v. 16, n. 3, p. 17–25, June 1996.

Brunello et al. 2020 BRUNELLO, R. K. V.; COELHO, L. d. S.; SAMPAIO, R. C.; AYALA, H. V. H.; LLANOS, C. H. Efficient hardware implementation of nonlinear moving-horizon state estimation with artificial neural networks. In: *21th IFAC World Congress*. [S.l.: s.n.], 2020. (IFAC 2020).

Burks, Goldstine e Neumann 1982 BURKS, A. W.; GOLDSTINE, H. H.; NEUMANN, J. V. Preliminary discussion of the logical design of an electronic computing instrument. In: *The Origins of Digital Computers*. [S.l.]: Springer, 1982. p. 399–413.

Chen et al. 2018 Chen, J.; Ouyang, Q.; Xu, C.; Su, H. Neural network-based state of charge observer design for lithium-ion batteries. *IEEE Transactions on Control Systems Technology*, v. 26, n. 1, p. 313–320, Jan 2018.

Chen 2017 CHEN, T. Robust state estimation for power systems via moving horizon strategy. *Sustainable Energy, Grids and Networks*, v. 10, p. 46–54, 2017. ISSN 2352-4677. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S2352467716300935⟩.

Chen, Kirkby e Jena 2012 CHEN, T.; KIRKBY, N. F.; JENA, R. Optimal dosing of cancer chemotherapy using model predictive control and moving horizon state/parameter estimation. *Computer Methods and Programs in Biomedicine*, v. 108, n. 3, p. 973–983, 2012. ISSN 0169-2607. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S016926071200137X⟩.

Chou et al. 2013 CHOU, H.-H.; KUNG, Y.-S.; QUYNH, N. V.; CHENG, S. Optimized FPGA design, verification and implementation of a neuro-fuzzy controller for PMSM drives. *Mathematics and Computers in Simulation*, v. 90, p. 28 – 44, 2013.

Christofides et al. 2013 CHRISTOFIDES, P. D.; SCATTOLINI, R.; PEñA, D. M. de la; LIU, J. Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, v. 51, p. 21 – 41, 2013.

Copp, Gondhalekar e Hespanha 2018 COPP, D. A.; GONDHALEKAR, R.; HESPANHA, J. P. Simultaneous model predictive control and moving horizon estimation for blood glucose regulation in type 1 diabetes. *Optimal Control Applications and Methods*, v. 39, n. 2, p. 904–918, 2018. Disponível em: ⟨https://onlinelibrary.wiley.com/doi/abs/10.1002/oca.2388⟩.

Coutinho, Torquato e Fernandes 2019 Coutinho, M. G. F.; Torquato, M. F.; Fernandes, M. A. C. Deep neural network hardware implementation based on stacked sparse autoencoder. *IEEE Access*, v. 7, p. 40674–40694, 2019.

Cyclone V Device Handbook - Intel CYCLONE V Device Handbook - Intel. Intel Corporation. Disponível em: ⟨https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyclone-v/cv_5v3.pdf⟩.

Dang e Ling 2014 DANG, T. V.; LING, K. V. Moving Horizon Estimation on a Chip. In: *2014 13TH INTERNATIONAL CONFERENCE ON CONTROL AUTOMATION ROBOTICS & VISION (ICARCV)*. [S.l.: s.n.], 2014. (International Conference on Control Automation Robotics and Vision), p. 431–437. ISBN 978-1-4799-5199-4. ISSN 2474-2953. 13th International Conference on Control Automation Robotics & Vision (ICARCV), Singapore, SINGAPORE, DEC 10-12, 2014.

Dewasme 2019 DEWASME, L. Neural network-based software sensors for the estimation of key components in brewery wastewater anaerobic digester: an experimental validation. *WATER SCIENCE AND TECHNOLOGY*, 80, n. 10, p. 1975–1985, NOV 15 2019. ISSN 0273-1223.

Dwivedi, Pandey e Junghare 2017 DWIVEDI, P.; PANDEY, S.; JUNGHARE, A. S. Stabilization of unstable equilibrium point of rotary inverted pendulum using fractional controller. *Journal of the Franklin Institute*, v. 354, n. 17, p. 7732 – 7766, 2017.

Englert et al. 2019 ENGLERT, T.; VOELZ, A.; MESMER, F.; RHEIN, S.; GRAICHEN, K. A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC). *OPTIMIZATION AND ENGINEERING*, 20, n. 3, p. 769–809, SEP 2019. ISSN 1389-4420.

Fan e Hwang 2013 FAN, Z.-C.; HWANG, W.-J. Efficient VLSI architecture for training radial basis function networks. *Sensors*, v. 13, n. 3, p. 3848–3877, 2013.

Farooq, Marrakchi e Mehrez 2012 FAROOQ, U.; MARRAKCHI, Z.; MEHREZ, H. *Tree-based heterogeneous FPGA architectures: application specific exploration and optimization*. [S.l.]: Springer Science & Business Media, 2012.

Fernando et al. 2019 FERNANDO, T.; DENMAN, S.; SRIDHARAN, S.; FOOKES, C. Neighbourhood Context Embeddings in Deep Inverse Reinforcement Learning for Predicting Pedestrian Motion Over Long Time Horizons. In: IEEE; IEEE Comp Soc; CVF. *2019 IEEE/CVF INTERNATIONAL CONFERENCE ON COMPUTER VISION WORKSHOPS (ICCVW)*. [S.l.], 2019. (IEEE International Conference on Computer Vision Workshops), p. 1179–1187. ISBN 978-1-7281-5023-9. ISSN 2473-9936. IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, SOUTH KOREA, OCT 27-NOV 02, 2019.

Findeisen, Graichen e Monnigmann 2018 FINDEISEN, R.; GRAICHEN, K.; MONNIGMANN, M. Embedded optimization in control: an introduction, opportunities, and challenges. *AT-AUTOMATISIERUNGSTECHNIK*, 66, n. 11, p. 877–902, NOV 2018. ISSN 0178-2312.

Frey et al. 2019 FREY, J.; QUIRYNEN, R.; KOUZOUPIS, D.; FRISON, G.; GEISLER, J.; SCHILD, A.; DIEHL, M. Detecting and Exploiting Generalized Nonlinear Static Feedback Structures in DAE Systems for MPC. In: Unvi Naples Federico II; Univ Sannio Benevento. *2019 18TH EUROPEAN CONTROL CONFERENCE (ECC)*. [S.l.], 2019. p. 2756–2762. ISBN 978-3-907144-00-8. 18th European Control Conference (ECC), Naples, ITALY, JUN 25-28, 2019.

Frick et al. 2012 FRICK, D.; DOMAHIDI, A.; VUKOV, M.; MARIETHOZ, S.; DIEHL, M.; MORARI, M. Moving Horizon Estimation for Induction Motors. In: IEEE. *2012 IEEE SYMPOSIUM ON SENSORLESS CONTROL FOR ELECTRICAL DRIVES (SLED)*. [S.l.], 2012. (Symposium on Sensorless Control for Electrical Drives). ISBN 978-1-4673-2966-8; 978-1-4673-2965-1. ISSN 2166-6725. IEEE Symposium on Sensorless Control for Electrical Drives (SLED), Milwaukee, WI, SEP 21-22, 2012.

Gajski et al. 2009 GAJSKI, D. D.; ABDI, S.; GERSTLAUER, A.; SCHIRNER, G. *Embedded system design: modeling, synthesis and verification*. [S.l.]: Springer Science & Business Media, 2009.

Gomez-Ortega e Camacho 1994 GOMEZ-ORTEGA, J.; CAMACHO, E. Neural network mbpc for mobile robot path tracking. *Robotics and Computer-Integrated Manufacturing*, v. 11, n. 4, p. 271–278, 1994. ISSN 0736-5845. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/0736584595000038⟩.

Goodwin et al. 2005 GOODWIN, G. C.; DONá, J. A. D.; SERON, M. M.; ZHUO, X. W. Lagrangian duality between constrained estimation and control. *Automatica*, v. 41, n. 6, p. 935 – 944, 2005.

Hartenstein HARTENSTEIN, R. Xputer: the alternative machine paradigm for energy-efficient computing.

Hartley et al. 2013 HARTLEY, E. N.; JEREZ, J. L.; SUARDI, A.; MACIEJOWSKI, J. M.; KERRIGAN, E. C.; CONSTANTINIDES, G. A. Predictive control using an fpga with application to aircraft control. *IEEE Transactions on Control Systems Technology*, IEEE, v. 22, n. 3, p. 1006–1017, 2013.

Haseltine e Rawlings 2005 HASELTINE, E. L.; RAWLINGS, J. B. Critical evaluation of extended kalman filtering and moving-horizon estimation. *Industrial & Engineering Chemistry Research*, v. 44, n. 8, p. 2451–2460, 2005.

Haykin 2009 HAYKIN, S. S. *Neural networks and learning machines*. 3rd. ed. Upper Saddle River: Prentice Hall, 2009.

Hornik, Stinchcombe e White 1989 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/0893608089900208⟩.

Hu, Cao e Egardt 2018 Hu, X.; Cao, D.; Egardt, B. Condition monitoring in advanced battery management systems: Moving horizon estimation using a reduced electrochemical model. *IEEE/ASME Transactions on Mechatronics*, v. 23, n. 1, p. 167–178, 2018.

Huang 2012 HUANG, H.-C. Fpga-based hybrid ga-pso algorithm and its application to global path planning for mobile robots. *Przeglad elektrotechniczny*, v. 88, n. 7B, p. 281–284, 2012.

Huang, Zhao e Zhang 2017 HUANG, J.; ZHAO, G.; ZHANG, X. Mems gyroscope/tam-integrated attitude estimation based on moving horizon estimation. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, v. 231, n. 8, p. 1451–1459, 2017. Disponível em: ⟨https://doi.org/10.1177/0954410016652920⟩.

Ibañez, Ocampo-Martinez e Gonzalez 2017 IBAÑEZ, C.; OCAMPO-MARTINEZ, C.; GONZALEZ, B. Embedded optimization-based controllers for industrial processes. In: IEEE. *2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC)*. [S.l.], 2017. p. 1–6.

IEEE 1985 IEEE. *IEEE standard for binary floating-point arithmetic*. NY, 1985.

Iplikci e Bahtiyar 2016 IPLIKCI, S.; BAHTIYAR, B. A field-programmable gate array implementation of a real-time nonlinear runge–kutta model predictive control. *Transactions of the Institute of Measurement and Control*, SAGE Publications Sage UK: London, England, v. 38, n. 5, p. 555–564, 2016.

Jabeen, Srinivasan e Shuja 2017 JABEEN, S.; SRINIVASAN, S.; SHUJA, S. Formal verification methodology for real-time field programmable gate array. *IET Computers & Digital Techniques*, IET, v. 11, n. 5, p. 197–203, 2017.

Jazaeri e Nasrabadi 2015 JAZAERI, M.; NASRABADI, M. T. A new fast and efficient artificial neural network based state estimator incorporated into a linear optimal regulator for power system control enhancement. *Electric Power Components and Systems*, Taylor and Francis, v. 43, n. 6, p. 644–655, 2015.

Kalman 1960 KALMAN, R. On the general theory of control systems. *IFAC Proceedings Volumes*, v. 1, n. 1, p. 491 – 502, 1960. 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.

Kalman 1960 KALMAN, R. E. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, v. 82, n. 1, p. 35–45, 1960.

Kamesh e Rani 2017 KAMESH, R.; RANI, K. Y. Novel Formulation of Adaptive MPC as EKF Using ANN Model: Multiproduct Semibatch Polymerization Reactor Case Study. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, 28, n. 12, p. 3061–3073, DEC 2017. ISSN 2162-237X.

Kenny e Watt 2016 KENNY, R.; WATT, J. The breakthrough advantage for fpgas with tri-gate technology. *URL: https://www. altera. com/en_US/pdfs/literature/wp/wp-01201-fpga-tri-gate-technology. pdf (: 12.10. 2017)*, 2016.

Kim e Jung 2015 KIM, J.; JUNG, S. Implementation of the RBF neural chip with the back-propagation algorithm for on-line learning. *Applied Soft Computing*, v. 29, p. 233 – 244, 2015.

Kittisupakorn et al. 2009 KITTISUPAKORN, P.; THITIYASOOK, P.; HUSSAIN, M.; DAOSUD, W. Neural network based model predictive control for a steel pickling process. *Journal of Process Control*, v. 19, n. 4, p. 579–590, 2009. ISSN 0959-1524. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0959152408001388⟩.

Kung, Than e Chuang 2018 KUNG, Y.-S.; THAN, H.; CHUANG, T.-Y. Fpga-realization of a self-tuning pid controller for x–y table with rbf neural network identification. *Microsystem Technologies*, v. 24, n. 1, p. 243–253, 2018.

Kwon et al. 2015 KWON, J. S.-I.; NAYHOUSE, M.; ORKOULAS, G.; NI, D.; CHRISTOFIDES, P. D. A method for handling batch-to-batch parametric drift using moving horizon estimation: Application to run-to-run mpc of batch crystallization. *Chemical Engineering Science*, v. 127, p. 210–219, 2015. ISSN 0009-2509. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0009250915000536⟩.

Ławryńczuk 2009 ŁAWRYŃCZUK, M. Neural networks in model predictive control. In: ____. *Intelligent Systems for Knowledge Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 31–63. ISBN 978-3-642-04170-9. Disponível em: ⟨https://doi.org/10.1007/978-3-642-04170-9_2⟩.

Leshno et al. 1993 LESHNO, M.; LIN, V. Y.; PINKUS, A.; SCHOCKEN, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, v. 6, n. 6, p. 861–867, 1993. ISSN 0893-6080. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0893608005801315⟩.

Lewis, Vrabie e Syrmos 2012 LEWIS, F. L.; VRABIE, D.; SYRMOS, V. L. *Optimal control*. Hoboken, New Jersey: John Wiley & Sons, 2012.

Liu et al. 2017 Liu, A.; Zhang, W.; Chen, M. Z. Q.; Yu, L. Moving horizon estimation for mobile robots with multirate sampling. *IEEE Transactions on Industrial Electronics*, v. 64, n. 2, p. 1457–1467, 2017.

Magana e Holzapfel 1998 Magana, M. E.; Holzapfel, F. Fuzzy-logic control of an inverted pendulum with vision feedback. *IEEE Transactions on Education*, v. 41, n. 2, p. 165–170, May 1998.

Mayne et al. 2000 MAYNE, D.; RAWLINGS, J.; RAO, C.; SCOKAERT, P. Constrained model predictive control: Stability and optimality. *Automatica*, v. 36, n. 6, p. 789–814, 2000. ISSN 0005-1098. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0005109899002149⟩.

Mehndiratta e Kayacan 2019 MEHNDIRATTA, M.; KAYACAN, E. A constrained instantaneous learning approach for aerial package delivery robots: onboard implementation and experimental results. *AUTONOMOUS ROBOTS*, 43, n. 8, p. 2209–2228, DEC 2019. ISSN 0929-5593.

Mercieca e Fabri 2012 MERCIECA, J.; FABRI, S. G. A metaheuristic particle swarm optimization approach to nonlinear model predictive control. John Wiley and Sons Ltd., 2012.

Messikh, Guechi e Benloucif 2017 MESSIKH, L.; GUECHI, E.; BENLOUCIF, M. Critically damped stabilization of inverted-pendulum systems using continuous-time cascade linear model predictive control. *Journal of the Franklin Institute*, v. 354, n. 16, p. 7241 – 7265, 2017.

Moody e Darken 1989 MOODY, J.; DARKEN, C. J. Fast learning in networks of locally-tuned processing units. *Neural Computation*, v. 1, n. 2, p. 281–294, 1989.

Muñoz et al. 2009 MUñOZ, D.; SANCHEZ, D.; LLANOS, C.; AYALA-RINCON, M. Tradeoff of FPGA design of floating-point transcendental functions. In: *17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*. [S.l.: s.n.], 2009. p. 239–242.

Muñoz et al. 2010 MUñOZ, D.; SANCHEZ, D.; LLANOS, C.; AYALA-RINCON, M. FPGA based floating-point library for cordic algorithms. In: *Programmable Logic Conference (SPL), VI Southern*. Ipojuca, Brazil: [s.n.], 2010. p. 55–60.

Muñoz et al. 2010 MUñOZ, D. M.; SáNCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCóN, M. Tradeoff of fpga design of a floating-point library for arithmetic operators. *JICS - Journal of Integrated Circuits and Systems*, v. 5, n. 1, p. 42–52, 2010.

Noguera e Badia 2002 NOGUERA, J.; BADIA, R. M. Hw/sw codesign techniques for dynamically reconfigurable architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, IEEE, v. 10, n. 4, p. 399–415, 2002.

Ortega e Camacho 1996 ORTEGA, J.; CAMACHO, E. Mobile robot navigation in a partially structured static environment, using neural predictive control. *Control Engineering Practice*, v. 4, n. 12, p. 1669–1679, 1996. ISSN 0967-0661. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0967066196001840⟩.

Park e Sandberg 1991 PARK, J.; SANDBERG, I. W. Universal approximation using radial-basis-function networks. *Neural Computation*, v. 3, n. 2, p. 246–257, Jun 1991. ISSN 0899-7667. Disponível em: ⟨https://doi.org/10.1162/neco.1991.3.2.246⟩.

Peng et al. 2015 PENG, D.; EL-FARRA, N. H.; GENG, Z.; ZHU, Q. Distributed data-based fault identification and accommodation in networked process systems. *CHEMICAL ENGINEERING SCIENCE*, 136, n. SI, p. 88–105, NOV 2 2015. ISSN 0009-2509.

Polóni et al. 2013 POLÓNI, T.; EIELSEN, A. A.; ROHAL'-ILKIV, B.; JOHANSEN, T. A. Adaptive model estimation of vibration motion for a nanopositioner with moving horizon optimized extended kalman filter. *Journal of Dynamic Systems, Measurement, and Control*, v. 135, n. 4, May 2013. ISSN 0022-0434. 041019. Disponível em: ⟨https://doi.org/10.1115/1.4024008⟩.

Qin e Chen 2013 QIN, H.; CHEN, W. Application of the constrained moving horizon estimation method for the ultra-short baseline attitude determination. *Acta Geodaetica et Geophysica*, v. 48, n. 1, p. 27–38, Mar 2013. ISSN 2213-5820. Disponível em: ⟨https://doi.org/10.1007/s40328-012-0004-2⟩.

Raff et al. 2005 RAFF, T.; EBENBAUER, C.; FINDEISEN, R.; ALLGÖWER, F. Remarks on moving horizon state estimation with guaranteed convergence. In: ____. *Control and Observer Design for Nonlinear Finite and Infinite Dimensional Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 67–80. ISBN 978-3-540-31573-5. Disponível em: ⟨https://doi.org/10.1007/11529798_5⟩.

Raković e Levine 2019 RAKOVIĆ, S. V.; LEVINE, W. S. *Handbook of model predictive control*. Cham, Switzerland: Birkhäuser, 2019.

Rawlings, Mayne e Diehl 2017 RAWLINGS, J. B.; MAYNE, D. Q.; DIEHL, M. *Model predictive control: theory, computation, and design*. Madison, WI: Nob Hill Publishing, 2017. v. 2.

Rossiter 2013 ROSSITER, J. A. *Model-Based Predictive Control A Practical Approach*. [S.l.]: CRC Press, 2013.

Sampaio 2018 SAMPAIO, R. C. *ARQUITETURAS DE HARDWARE PARA ACELERAÇÃO DE ALGORITMOS DE CONTROLE PREDITIVO NÃO-LINEAR*. Tese (Doutorado), 2018.

Santos et al. 2017 SANTOS, C. E.; COELHO, L. d. S.; SAMPAIO, R. C.; JACOBI, R.; AYALA, H.; LLANOS, C. H. A svm optimization tool and fpga system architecture applied to nmpc. In: *Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands*. New York, NY, USA: Association for Computing Machinery, 2017. (SBCCI '17), p. 96–102. ISBN 9781450351065. Disponível em: ⟨https://doi.org/10.1145/3109984.3110007⟩.

Scala, Bitmead e James 1995 SCALA, B. F. L.; BITMEAD, R. R.; JAMES, M. R. Conditions for stability of the extended kalman filter and their application to the frequency tracking problem. *Mathematics of Control, Signals and Systems*, v. 8, n. 1, p. 1–26, Mar 1995. ISSN 1435-568X. Disponível em: ⟨https://doi.org/10.1007/BF01212364⟩.

Schaible, Xie e Lee 1997 SCHAIBLE, B.; XIE, H.; LEE, Y.-C. Fuzzy logic models for ranking process effects. *IEEE Transactions on Fuzzy Systems*, v. 5, n. 4, p. 545–556, 1997.

Seenivasan, Olivares e Staffetti 2020 SEENIVASAN, D. B.; OLIVARES, A.; STAFFETTI, E. Multi-aircraft optimal 4D online trajectory planning in the presence of a multi-cell storm in development. *TRANSPORTATION RESEARCH PART C-EMERGING TECHNOLOGIES*, 110, p. 123–142, JAN 2020. ISSN 0968-090X.

Segovia et al. 2019 SEGOVIA, P.; RAJAOARISOA, L.; NEJJARI, F.; DUVIELLA, E.; PUIG, V. Model predictive control and moving horizon estimation for water level regulation in inland waterways. *Journal of Process Control*, v. 76, p. 1–14, 2019. ISSN 0959-1524. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0959152418303470⟩.

Shen et al. 2019 Shen, J.; Shen, J.; He, Y.; Ma, Z. Accurate state of charge estimation with model mismatch for li-ion batteries: A joint moving horizon estimation approach. *IEEE Transactions on Power Electronics*, v. 34, n. 5, p. 4329–4342, May 2019. ISSN 1941-0107.

Shen et al. 2016 SHEN, J.-N.; HE, Y.-J.; MA, Z.-F.; LUO, H.-B.; ZHANG, Z.-F. Online state of charge estimation of lithium-ion batteries: A moving horizon estimation approach. *Chemical Engineering Science*, v. 154, p. 42–53, 2016. ISSN 0009-2509. Recent Advances in Energy Conversion and Storage Devices. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0009250916303578⟩.

Sindhwani, Roelofs e Kalakrishnan 2017 SINDHWANI, V.; ROELOFS, R.; KALAKRISHNAN, M. Sequential Operator Splitting for Constrained Nonlinear Optimal Control. In: GE Global Res; Mitsubishi Elect Res Labs; Eaton; United Technologies Res Ctr; MathWorks; Amer Automat Control Council. *2017 AMERICAN CONTROL CONFERENCE (ACC)*. [S.l.], 2017. (Proceedings of the American Control Conference), p. 4864–4871. ISBN 978-1-5090-5992-8. ISSN 0743-1619. American Control Conference (ACC), Seattle, WA, MAY 24-26, 2017.

Souza e Fernandes 2014 SOUZA, A. C. D. de; FERNANDES, M. A. C. Parallel fixed point implementation of a radial basis function network in an FPGA. *Sensors*, v. 14, n. 10, p. 18223, 2014.

Stellato, Geyer e Goulart 2017 Stellato, B.; Geyer, T.; Goulart, P. J. High-speed finite control set model predictive control for power electronics. *IEEE Transactions on Power Electronics*, v. 32, n. 5, p. 4007–4020, 2017.

Su et al. 2018 SU, X.; XIA, F.; LIU, J.; WU, L. Event-triggered fuzzy control of nonlinear systems with its application to inverted pendulum systems. *Automatica*, v. 94, p. 236 – 248, 2018.

Sun et al. 2015 Sun, L.; Castagno, J. D.; Hedengren, J. D.; Beard, R. W. Parameter estimation for towed cable systems using moving horizon estimation. *IEEE Transactions on Aerospace and Electronic Systems*, v. 51, n. 2, p. 1432–1446, 2015.

Tyler, Asano e Morari 2000 TYLER, M. L.; ASANO, K.; MORARI, M. Application of moving horizon estimation based fault detection to cold tandem steel mill. *International Journal of Control*, Taylor Francis, v. 73, n. 5, p. 427–438, 2000. Disponível em: ⟨https://doi.org/10.1080/002071700219605⟩.

Vandersteen et al. 2013 VANDERSTEEN, J.; DIEHL, M.; AERTS, C.; SWEVERS, J. Spacecraft attitude estimation and sensor calibration using moving horizon estimation. *Journal of Guidance, Control, and Dynamics*, v. 36, n. 3, p. 734–742, 2013. Disponível em: ⟨https://doi.org/10.2514/1.58805⟩.

Virtex-7 FPGAs 2012 VIRTEX-7 FPGAs. San Jose, CA: [s.n.], 2012.

Wain et al. 2006 WAIN, R.; BUSH, I.; GUEST, M.; DEEGAN, M.; KOZIN, I.; KITCHEN, C. *An overview of FPGAs and FPGA programming-Initial experiences at Daresbury*. [S.l.], 2006.

Wan e Keviczky 2019 Wan, Y.; Keviczky, T. Real-time fault-tolerant moving horizon air data estimation for the reconfigure benchmark. *IEEE Transactions on Control Systems Technology*, v. 27, n. 3, p. 997–1011, 2019.

Wulf e McKee 1995 WULF, W. A.; MCKEE, S. A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, ACM New York, NY, USA, v. 23, n. 1, p. 20–24, 1995.

Xu et al. 2015 XU, F.; CHEN, H.; GONG, X.; MEI, Q. Fast nonlinear model predictive control on fpga using particle swarm optimization. *IEEE Transactions on Industrial Electronics*, IEEE, v. 63, n. 1, p. 310–321, 2015.

Zeng e Liu 2015 ZENG, J.; LIU, J. Distributed Moving Horizon Estimation Subject to Communication Delays and Losses. In: Amer Automat Control Council; IFAC; Adaptics Inc; Altair; dSPACE; Eaton Corp; Elsevier; Int Journal Automat & Comp; Journal Franklin Inst; Plexim Inc; Soc Ind & Appl Math; Springer; CRC Press Taylor & Francis Grp Cogent OA; United Technologies Res Ctr; Wiley; Boeing; Ford Motor Co; GE Global Res; Honeywell; MathWorks; Mitsubishi Elect Res Lab; Quanser. *2015 AMERICAN CONTROL CONFERENCE (ACC)*. [S.l.], 2015. (Proceedings of the American Control Conference), p. 5533–5538. ISBN 978-1-4799-8684-2. ISSN 0743-1619. American Control Conference, Chicago, IL, JUL 01-03, 2015.

Zoppoli et al. 2020 ZOPPOLI, R.; PARISINI, T.; BAGLIETTO, M.; SANGUINETI, M. *Neural Approximations for optimal control and decision*. Cham, Switzerland: Springer, 2020.

Zoppoli, Sanguineti e Parisini 2002 ZOPPOLI, R.; SANGUINETI, M.; PARISINI, T. Approximating networks and extended ritz method for the solution of functional optimization problems. *Journal of Optimization Theory and Applications*, v. 112, n. 2, p. 403–440, Feb 2002. ISSN 1573-2878. Disponível em: ⟨https://doi.org/10.1023/A:1013662124879⟩.

Zorić et al. 2019 ZORIć, N. D.; TOMOVIć, A. M.; OBRADOVIć, A. M.; RADULOVIć, R. D.; PETROVIć, G. R. Active vibration control of smart composite plates using optimized self-tuning fuzzy logic controller with optimization of placement, sizing and orientation of pfrc actuators. *Journal of Sound and Vibration*, v. 456, p. 173 – 198, 2019.

Zou et al. 2020 ZOU, L.; WANG, Z.; HU, J.; HAN, Q.-L. Moving horizon estimation meets multi-sensor information fusion: Development, opportunities and challenges. *Information Fusion*, v. 60, p. 1–10, 2020. ISSN 1566-2535. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S1566253519310024⟩.

Zynq-7000 SoC Data Sheet: Overview ZYNQ-7000 SoC Data Sheet: Overview. Xilinx Inc. Disponível em: ⟨https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf⟩.

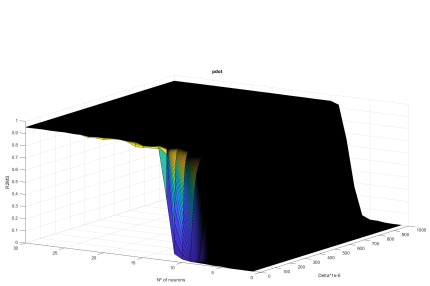
Åkesson e Toivonen 2006 ÅKESSON, B. M.; TOIVONEN, H. T. A neural network model predictive controller. *Journal of Process Control*, v. 16, n. 9, p. 937–946, 2006. ISSN 0959-1524. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0959152406000618⟩.

Ławryńczuk 01 Jun. 2009 ŁAWRYńCZUK, M. Efficient nonlinear predictive control based on structured neural models. *International Journal of Applied Mathematics and Computer Science*, Sciendo, Berlin, v. 19, n. 2, p. 233 – 246, 01 Jun. 2009. Disponível em: ⟨https://content.sciendo.com/view/journals/amcs/19/2/article-p233.xml⟩.

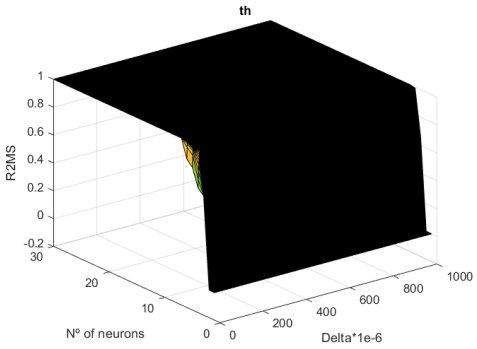
Ławryńczuk 2011 ŁAWRYńCZUK, M. On improving accuracy of computationally efficient nonlinear predictive control based on neural models. *Chemical Engineering Science*, v. 66, n. 21, p. 5253–5267, 2011. ISSN 0009-2509. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0009250911004763⟩.

APÊNDICES

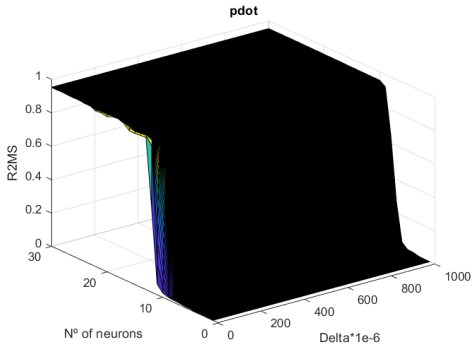
I RADIAL BASIS FUNCTION MOVING-HORIZON ESTIMATOR



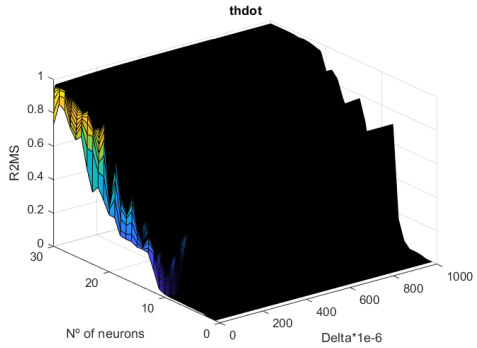
(a) Position



(b) Theta



(c) Pdot



(d) Theta dot

Figure I.1: R-squared surface of all states of RBFMHE variation by nº of neurons and delta.

Nº of Neurons/ Delta(1e-3/Iter)	Mínimo da linha	1	100	200	300	400	500	600	700	800	900	1000
1	-1,5579371E-06	-0,001610763	-1,55841E-05	-7,79074E-06	-5,19354E-06	-3,89504E-06	-3,11598E-06	-2,59662E-06	-2,22566E-06	-1,94744E-06	-1,73105E-06	-1,55794E-06
2	3,5262751E-07	-0,001589685	3,526275E-07	3,498623E-07	3,489402E-07	3,484791E-07	3,482024E-07	3,480179E-07	3,478861E-07	3,477873E-07	3,477104E-07	3,476489E-07
3	0,0018051241	2,979009E-05	0,0017864	0,001797372	0,001800714	0,001802325	0,001803273	0,001803896	0,001804338	0,001804667	0,001804921	0,001805124
4	0,0075309257	0,002633187	0,007530926	0,007519188	0,007514431	0,007511894	0,007510322	0,007509252	0,007508478	0,007507892	0,007507432	0,007507063
5	0,011053514	0,0046582	0,011053514	0,011043894	0,011040023	0,011037963	0,011036688	0,011035821	0,011035193	0,011034718	0,011034346	0,011034047
6	0,13213954	0,00714478	0,017384605	0,023885324	0,033324865	0,045121275	0,058624776	0,073193582	0,088251543	0,10332172	0,11803781	0,13213954
7	0,21960659	0,011554719	0,022542508	0,04059982	0,065076755	0,09237012	0,11962046	0,14505009	0,1678188	0,18771569	0,20487954	0,21960659
8	0,23333982	0,011589668	0,099311683	0,15776994	0,18080532	0,19265591	0,20084076	0,2077496	0,21423298	0,22060501	0,226969	0,23333982
9	0,30560567	0,015038737	0,21342873	0,22462538	0,23788974	0,25171024	0,26455652	0,27576677	0,28526127	0,29322557	0,29992231	0,30560567
10	0,6472769	0,028628472	0,52360395	0,57592611	0,59580603	0,60926723	0,6195527	0,62760738	0,63401648	0,63922397	0,64356286	0,6472769
11	0,4489778	0,076606016	0,37796118	0,39457326	0,40823853	0,41963649	0,42845634	0,43507715	0,44003603	0,44379392	0,44669362	0,4489778
12	0,53283554	0,026458841	0,42559751	0,45002956	0,4617283	0,47144398	0,48110628	0,49106279	0,50131085	0,51176135	0,52230568	0,53283554
13	0,72802571	0,082287891	0,5124122	0,57410639	0,62021328	0,65207997	0,67444184	0,69084723	0,70340032	0,7133289	0,72137833	0,72802571
14	0,9469373	0,25155508	0,76838613	0,86883407	0,90622245	0,92314926	0,93223901	0,93772443	0,94131152	0,94379621	0,94559316	0,9469373
15	0,95080931	0,082030819	0,77278331	0,86764689	0,90967131	0,92958199	0,94022472	0,94652243	0,94995297	0,95033847	0,9506107	0,95080931
16	0,95046692	0,24506357	0,68014674	0,85926404	0,91588276	0,93914864	0,94811491	0,94891721	0,94949113	0,94991139	0,9502262	0,95046692
17	0,95029109	0,22690326	0,6926312	0,86155844	0,91641238	0,93927832	0,9481952	0,94891545	0,94942943	0,94980325	0,95008082	0,95029109
18	0,9501424	0,25021926	0,7680783	0,87905605	0,9214615	0,94056024	0,94930675	0,94957019	0,94977245	0,94992778	0,9500481	0,9501424
19	0,94821257	0,30575933	0,81344914	0,88184212	0,91048284	0,92500265	0,93336993	0,9386643	0,94225585	0,94482418	0,94673802	0,94821257
20	0,95145621	0,3352152	0,82089816	0,89308361	0,92780615	0,94545087	0,95144766	0,95145287	0,95145514	0,95145602	0,95145621	0,95145603
21	0,9513148	0,41882755	0,93643087	0,94583017	0,95081489	0,95095009	0,95104937	0,95112662	0,95118834	0,95123859	0,95128009	0,9513148
22	0,95190717	0,40978507	0,95142016	0,95175261	0,95187137	0,95190599	0,95190717	0,95189605	0,9518813	0,95186639	0,95185259	0,9518403
23	0,95204693	0,72093148	0,95160808	0,95197739	0,95204693	0,95204131	0,95201842	0,95199359	0,95197065	0,95195029	0,95193231	0,95191637
24	0,95201427	0,74784647	0,95198236	0,95201427	0,95200585	0,95199696	0,9519908	0,95198645	0,95198312	0,9519804	0,95197803	0,95197588
25	0,95206583	0,74451652	0,95198801	0,95205767	0,95206557	0,95206583	0,95206354	0,95205964	0,95205453	0,95204852	0,9520419	0,95203489
26	0,95195009	0,58390752	0,95177891	0,95185076	0,95190339	0,95193033	0,95194296	0,95194837	0,95195009	0,95194991	0,95194873	0,95194705
27	0,95198257	0,46472968	0,95166546	0,95187115	0,95194939	0,95197647	0,95198257	0,95198013	0,95197445	0,95196782	0,95196122	0,95195503
28	0,95201452	0,87566478	0,95201452	0,95196583	0,95194703	0,95194233	0,95194229	0,95194306	0,95192577	0,95177217	0,95176466	0,95175914
29	0,95207105	0,92095252	0,95207105	0,95207009	0,95206563	0,95205747	0,95204844	0,95203961	0,95203128	0,95202345	0,95201603	0,95200901
30	0,95213331	0,72935837	0,95213331	0,95212685	0,95212259	0,95211694	0,95210891	0,9520988	0,95208741	0,95207551	0,95206371	0,95205242

Figure I.2: R-squared RBFMHE table.

Nº of Neurons/ Delta(1/2^Iter)	Mínimo da linha	16	20	25	30	35	40	45
6	0,0092028975	0,008129418	0,009146094	0,009201169	0,009202898	0,009201944	0,00920254	0,00920254
7	0,29051977	0,017754138	0,19980413	0,29047769	0,29046363	0,29045403	0,29051977	0,29051977
8	0,27909702	0,062645555	0,20581466	0,27869046	0,27897066	0,27909702	0,27900934	0,27849668
9	0,35668242	0,20204711	0,30415463	0,35645407	0,35668242	0,3565647	0,35639739	0,35627466
10	0,64359391	0,48924154	0,63067937	0,64356935	0,64339036	0,64359391	0,64354032	0,64319432
11	0,46342087	0,32468367	0,44814658	0,46188027	0,46325099	0,46281254	0,46342087	0,46259373
12	0,74363267	0,39129764	0,53713256	0,7402218	0,74363267	0,74182332	0,74357796	0,74286419
13	0,80094117	0,43714869	0,72793925	0,79560441	0,80001509	0,79770505	0,79970455	0,80094117
14	0,95010328	0,67333126	0,93454212	0,94992584	0,950023	0,94996834	0,94999158	0,95010328
15	0,95150059	0,7180565	0,94619066	0,95150059	0,951478	0,95146614	0,9514761	0,95147526
16	0,95162463	0,41794783	0,95060253	0,95162463	0,95156437	0,9515903	0,95154506	0,95154905
17	0,95137334	0,48009181	0,95043117	0,95137334	0,95133752	0,95129734	0,95129406	0,95131356
18	0,95091069	0,64795411	0,95022541	0,95091069	0,95086926	0,95073658	0,95084679	0,95082217
19	0,95129979	0,73660493	0,94062209	0,95123756	0,95113564	0,95129979	0,9512713	0,95123321
20	0,95146376	0,78644621	0,95130706	0,95140654	0,95146376	0,95145667	0,95145082	0,95145303

Figure I.3: R-squared RBFMHE with cost function Gauss-1 table.

I MODEL PREDICTIVE CONTROL WITH MOVING HORIZON ESTIMATOR

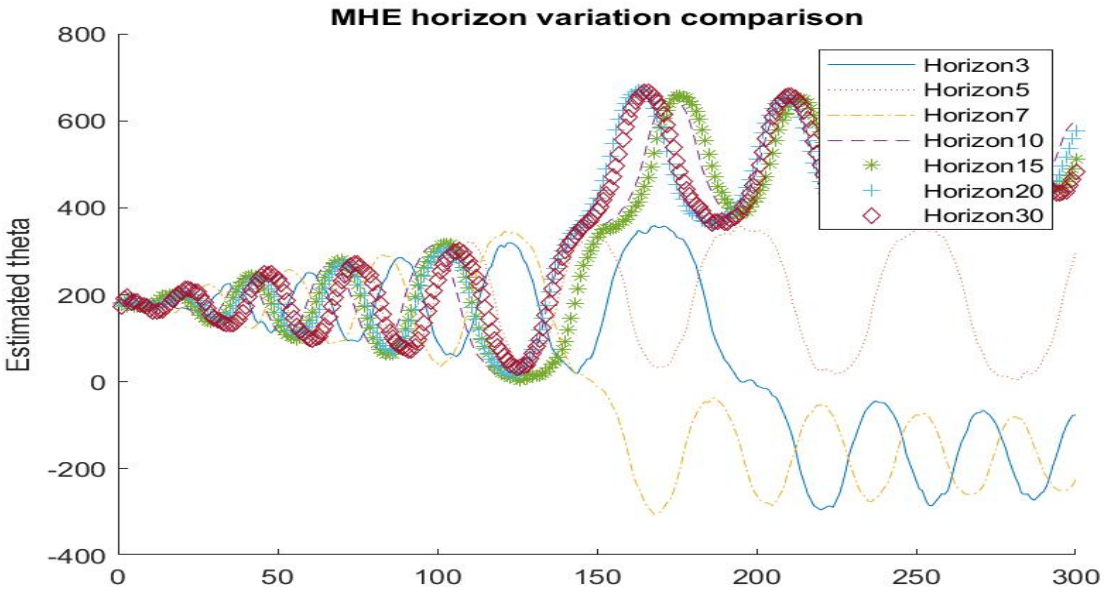


Figure I.1: Result comparison of multiple MHE horizon sizes for the MHEC.

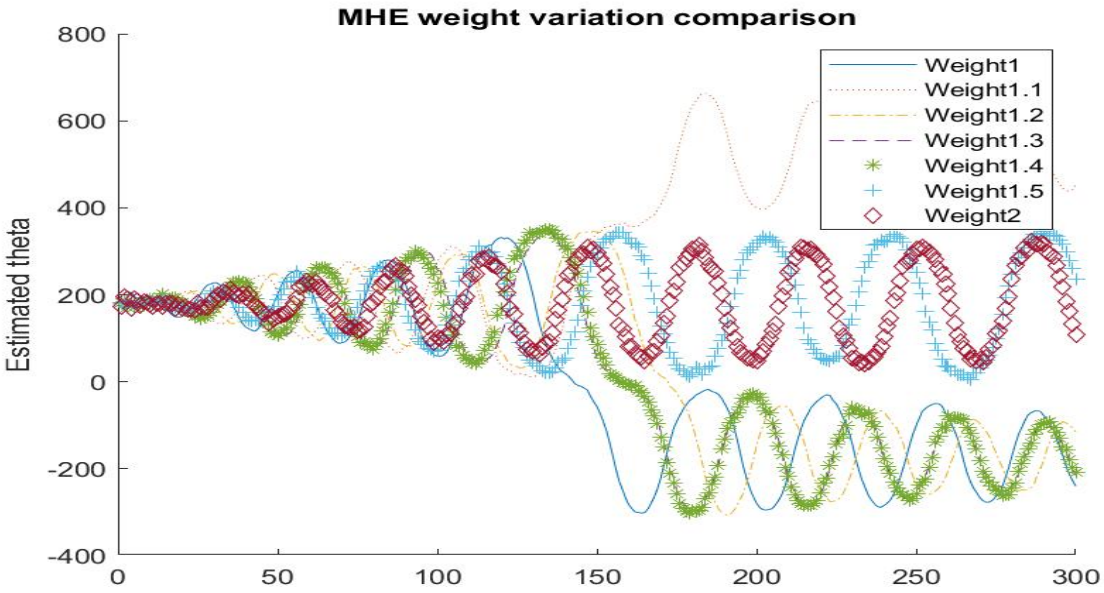


Figure I.2: Result comparison of multiple MHE weights for the MHEC.

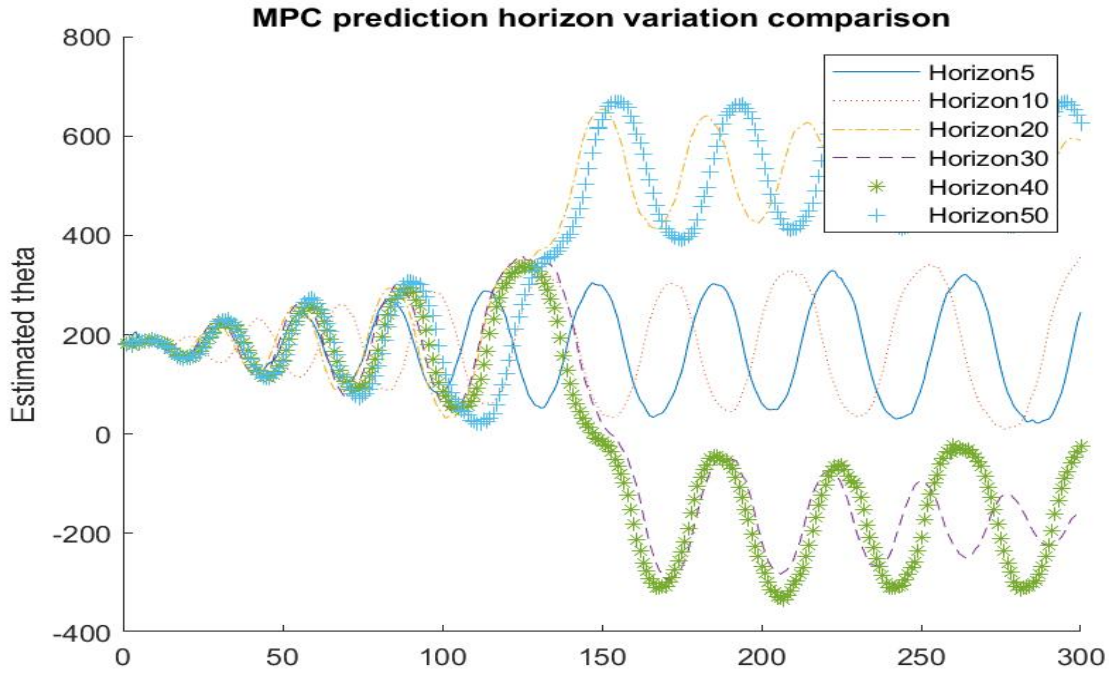


Figure I.3: Result comparison of multiple MPC prediction horizon sizes for the MHEC.

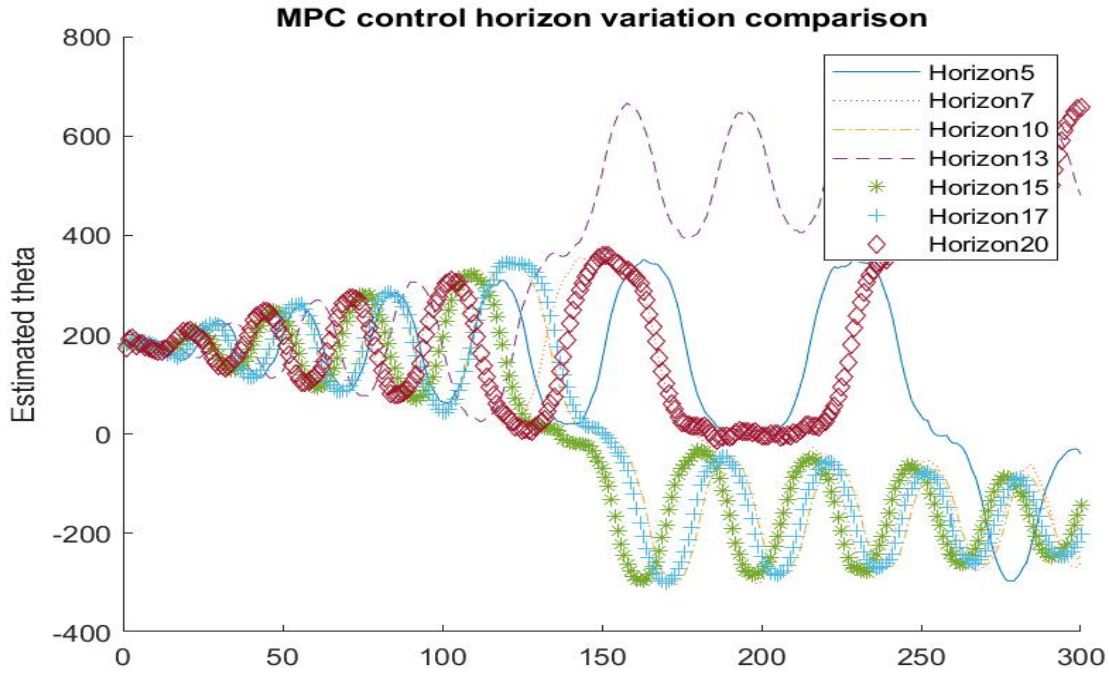


Figure I.4: Result comparison of multiple MPC control horizon sizes for the MHEC.