FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

DIPLOMA THESIS

AN AI-BASED SOLUTION FOR WIRELESS CHANNEL INTERFERENCE PREDICTION AND WIRELESS REMOTE CONTROL

Author	Christian Padilla
Supervisor	Sumudu Samarakoon
Second examiner	Samad Ali

March 2023

Padilla C. (2022) An AI-Based solution for Wireless Channel Interference Prediction and Wireless Remote Control Faculty of Information Technology and Electrical Engineering, Degree Programme in Wireless Communications Engineering. Master's Thesis, 48 pages.

ABSTRACT

Most control systems rely on wired connectivity between controllers and plants due to their need for fast and reliable real-time control. Yet the demand for mobility, scalability, low operational and maintenance costs call for wireless networked control system designs. Naturally, over-the-air communication is susceptible to interference and fading and therefore, enabling low latency and high reliability is crucial for wireless control scenarios. In this view, the work of this thesis aims to enhance reliability of the wireless communication and to optimize the energy consumption while maintaining low latency and the stability of the controller-plant system. To achieve this goal, two core abstractions have been used, a neural wireless channel interference predictor and a neural predictive controller. This neural predictor design is motivated by the capability of machine learning in assimilating underlying patterns and dynamics of systems using the observed data. The system model is composed of a controller-plant scheme on which the controller transmits control signals wirelessly. The neural wireless predictor and the neural controller predict wireless channel interference and plant states, respectively. This information is used to optimize energy consumption and prevent communication outages while controlling the plant. This thesis presents the development of the neural wireless predictor, the neural controller and a neural plant. Interaction and functionality of these elements are demonstrated using a Simulink simulation. Results of simulation illustrate the effectiveness of neural networks in both control and wireless domain. The proposed solution yields about 17% reduction in energy consumption compared to state-of-the-art designs by minimizing the impact of interference in the control links while ensuring plant stability.

Keywords: neural controller, long short term memory cells, model reference adaptive controller, interference prediction, non autoregressive neural networks

CONTENTS

PREFACE

This thesis was aimed at the development of an AI-based solution for future systems than involve both wireless communications and control.
I would like to express my gratitude to my supervisor Sumudu for his exceptional technical guidance and to my close friends Diana and Xavier who supported me in both academic and personal aspects during my studies.

Christian Padilla

1 INTRODUCTION

Control systems mostly rely on wired connections to send control signals to a plant of interest. This is due to the fact that real-time control requires low latent and reliable communications to ensure plant stability and observability. In contrast to the wired connectivity, wireless transmissions are susceptible to interference and fading, and therefore, are not the default choice for real-time control. Nevertheless, the benefits of versatility offered within wireless connectivity can be reaped for control applications if the drawbacks of wireless transmissions are addressed. latency and reliability. For an example, in manufacturing plant, installation and maintenance costs can be reduced due to the wireless networking being utilized to enhance the production process.

This work presents a solution to tackle one major inconvenient of wireless-networked real-time control applications. This proposal aims to boost reliability of the wireless communication to ensure stability of the controller-plant system. To achieve this goal, two core abstractions will be used, a neural wireless channel interference predictor and a neural predictive controller/plant. The system model is composed of a controller-plant scheme on which the controller transmits control signals wirelessly. This information is used to send "future" control commands as a queue before the interference levels make real-time control ineffective. The plant uses this cached future control commands until the interference levels drop down enough to switch back to real-time control. Calculating future commands can be challenging since there is also a need to predict future plant states and every state depends on the previous plant state and current input. For this reason, the neural predictive controller/plant has been developed.

1.1 Background and motivation

Increasing the control coverage of a manufacturing line with wireless communications can be very advantageous, an increased control coverage makes possible to adjust parameters of the manufacturing line dynamically to accommodate new requirements. With wireless links, adjustments are not restricted to any movement or hazards involving rewiring existing infrastructure. Consider the expansion of production capacity in a factory, the ability of remote configuration and control facilitates deployment of new actuators while avoiding the inherent hazards of installing new wires.

The productivity of a manufacturing line is mainly affected by downtime, which is not only caused by failures in machinery but also by broken cables that are attached to moving parts. Finding and repairing a broken cable is a time-consuming and costly task [1]. In other words, a broken cable causes downtime, material costs and maintenance. The advantages of wireless technologies in an industrial environment are not limited to wiring costs and improved flexibility, wireless communications also make easier to add redundancy and add support for mobile systems in industrial environments.

1.2 Research Problem

According to [2], when wireless networks are deployed, often factories do not meet requirements of reliability, resilience and scalability. Wireless transmissions are susceptible to fading and interference from other transmitters; these issues affect reliability of the communication, and therefore wireless networks are not the default choice to establish communication between a controller and actuators; in [3], it is stated that wireless communication is not within the

first ten most popular communication protocols. Another key challenge regarding wireless communications is the stringent latency requirement for industry environments, usually 0.5-1ms [4]. Methods based on artificial intelligence (AI) are proposed in this work to overcome these challenges.

1.3 Scope

The targets of this work can be summarized in overcoming reliability and latency issues with maximum energy efficiency in a wireless industrial environment. To tackle these challenges, interference prediction is needed; in this work, AI and more specifically, deep learning is used to predict interference. Having knowledge of future interference power allows to adjust transmitting power (at the wireless transceiver at the controller) to avoid outage and ensure latency. Another advantage of having this knowledge is that is possible to choose to transmit in advance a set of control commands if the maximum transmit power at the controller cannot meet the requirements to keep the plant stable. This feature can also be exploited to reduce transmit energy consumption by sending in advance control commands that according to predictions might have to overcome high interference in the future, transmitting them now while the interference is low is better that transmitting in the future when the interference has risen.

Sending future control commands require knowledge of future states of the plant. These states must be calculated recursively since the next state depends on the current state and control command applied, i.e. if the control command needed for ten steps in the future is needed, the previous nine steps need to be calculated. In addition, all plants are different and require different mathematical modelling. As a novel method, this work presents development of neural controllers and neural plant both which can be trained using historical data. This implies that they can be used on a myriad of scenarios and do not require specific design as long as the training data is available.

In conclusion, this work proposes methods based on deep learning to achieve reliability, latency and energy efficiency in wireless industrial environments.

1.4 Methodology

This work consists mainly on developing three neural networks: neural wireless predictor, neural controller and neural plant.

For the neural wireless predictor, this work started by generating the interference power data in simulations. Later, this data was used to train and test the neural network, since this is a prediction this problem, a time series forecasting neural network was needed; performance of various neural network architectures were tested to find the most suitable one to forecast interference power.

For the neural plant, the goal was to mimic the behaviour of a real plant. Doing this, required choosing the dynamics of the plant and using random numbers and inputs applied to these dynamics to obtain outputs. The input-output pairs were used to train and validate the neural plant.

For the neural controller, a reference model with the desired behaviour of the controller-plant system was used. In this case, the inputs were made of random input references for the controller and the desired outputs obtained from the reference model. This is known as a Neural Model Reference Adaptive Controller. More details will be further explained in the following chapters.

There is one last piece of the system: an optimizer that was developed to gather all neural

networks and reduce power consumption by choosing which command needed to be transmitted instead of transmitting every command. To achieve this an optimization problem was formulated and solved.

1.5 Contribution

The main contribution of this thesis is to propose a novel wireless interference prediction method and an innovative controller design. The proposed solution focuses on maximizing energy efficiency while ensuring reliability and latency. A key advantage of this proposal is that the whole system only requires historic data for training making deployment relatively easy. In addition, the neural plant is made of a shallow neural network and the controller has only 4 layers and around 20 neurons in total, its simplicity can help keep the complexity of implementation low while reducing computational cost, the Neural Wireless predictor does use LSTM cells.

Currently, wired communications are the most common in industrial environments, but given the promising advantages of wireless technologies in a factory like improved flexibility, redundancy, safety and cost reduction, AI methods have been developed in this work in the favor of wireless communications.

1.6 Thesis Structure

This thesis consists of four chapters, it is organized in the following manner: Chapter 2 explores AI principles and its applications in wireless communications, Chapter 3 explores relevant control theory principles and AI-aided control. Chapter 4 showcases the system model and the proposed solution and exposes results and benchmarking and Chapter 5 shows conclusions and future directions.

2 ARTIFICIAL INTELLIGENCE FOR INTERFERENCE PREDICTION

AI, machine learning and deep learning have great potential for improving efficiency of a wireless system; these technologies can aid resource allocation, scheduling, system monitoring, optimization, and other applications [5]; Figure 2.1 shows how Artificial Intelligence, Machine learning and Deep Learning are related. Deep learning is a subset of machine Learning, and at the same time machine learning is a subset of AI.

In [5], it is stated that AI is a term used to describe intelligence demonstrated by a machine by mimicking some human-related mental processes as comprehension, learning and decision making. In the context of wireless communications, machine learning is the research area that aims to provide a networked system with autonomous performance improvement capabilities by interacting with the real world [5]. In other words, machine learning enables the design of self-optimizing systems.

Deep learning is inspired by the human brain, for this reason, a neural network mimics the human brain structure [5]. Predictions of future behavior of systems have fundamental importance in this thesis work; deep learning can be utilized to obtain models with forecasting capabilities. Prediction ability of several model architectures will be exploited in Chapter 2 and Chapter 3. Note that applications for machine learning are not limited to prediction; classification, clustering and regression are other in-demand use cases.

2.1 Neural Networks

Neural networks are made of numerous nodes. These nodes are called neurons because they simulate the behaviour and functions of a biological neuron [6]. Neural networks replicate the neuron's association process by using weights [7]. The structure of a neuron with only one input is shown in Figure 2.1. The scalar input x is multiplied by the weight w to obtain the product wx, which is added to the value of bias b and then fed to the transfer (also called activation) function f, which produces the output of the neuron y. Weights and bias are adjusted during training to obtain a desired specific behaviour [7]. Regarding the activation function, there is a vast assortment of functions available but two specific types were commonly used when designing neural networks in this work: linear and log-sigmoid activation functions. Usually, a linear function is used in the output layer neurons while log-sigmoid is used in the hidden layers. The linear function is expressed by

$$f(n) = n, \tag{2.1}$$

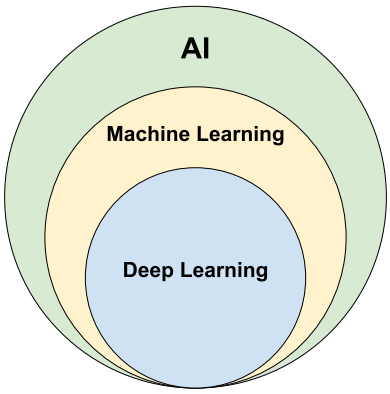

Figure 2.1. Relationship between AI, machine learning and deep learning.

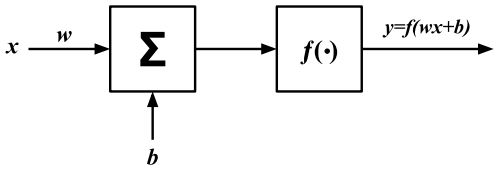

Figure 2.2. Neuron architecture.

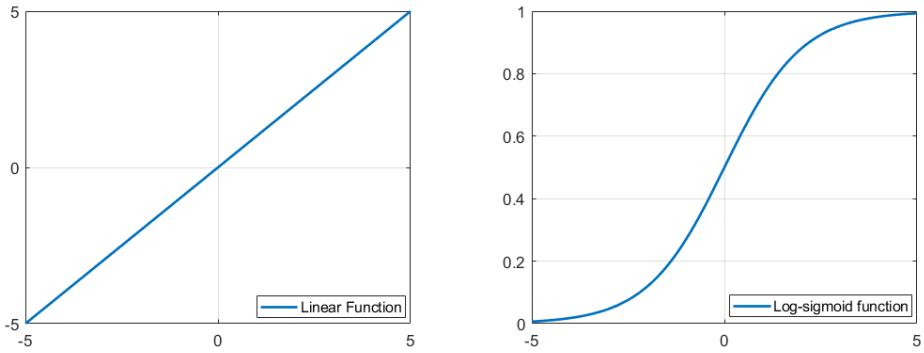

Figure 2.3. Linear (left) and log-sigmoid (right) activation functions.

where n is the result of vector operations $wx + b$, and $f(n)$ is the output of the neuron. This function is plotted in Figure 2.3.

The log-sigmoid activation function is expressed by

$$f(n) = \frac{1}{1 + e^{-n}}, \tag{2.2}$$

where n is the result of operations $wx + b$ and $f(n)$ is the output of the neuron. This function is plotted in Figure 2.3.

In real-life scenarios, neurons have more than one input. A neuron with multiple inputs is depicted in Figure 2.4 inside the red dashed lines. The elements in the input vector $\mathbf{x} = [x_1, x_2, ..., x_N]$ are weighted respectively by elements of vector $\mathbf{w} = [w_1, w_2, ..., w_N]$; bias $\mathbf{b} = [b_1, b_2, ..., b_N]$ is added to $\mathbf{wx}$, the result is passed as an argument to the activation function f.

2.2 Layers of Neural Networks

Neural networks have a layered structure as shown in Figure 2.4 [6]. Every multi-input neuron is represented by a circle and input nodes are depicted as squares. Layers are widely classified in bibliography as input, hidden and output layers. Input layer passes input signals to neurons in the hidden layer. The input layer does not perform the weighted sum, bias and it does not apply activation function, these operations are applied on hidden and output layer neurons. The layers between input and output are known as the hidden layers. A neural network with two or more hidden layers is called a deep neural network.

Consider the multi-input neuron in Figure 2.4. If a second neuron is added after the output

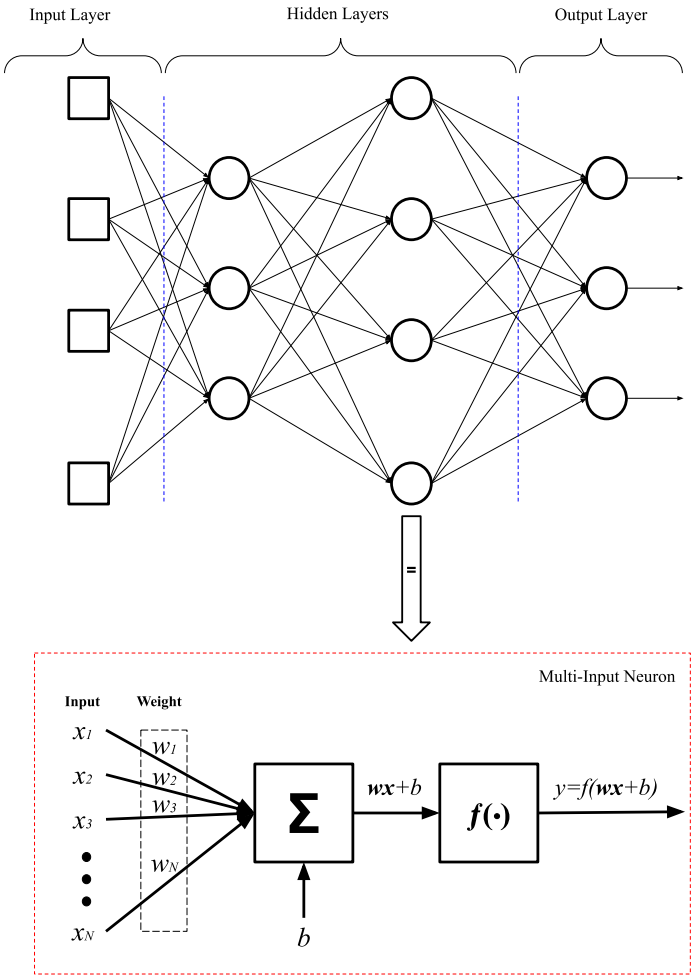

Figure 2.4. Neural network architecture and multi-input neuron.

of the first neuron, the output of this shallow neural network is given by

$$y^{(2)} = f^{(2)}(W_L^{(2,1)} f(W_I^{(1,1)} x + b^{(1)}) + b^{(2)}), \tag{2.3}$$

where ${W_L}^{(2,1)}$ represents the layer weight matrix of connections starting at layer 1 and ending at layer 2. The second index indicates the source, and the first index indicates the destination. ${W_I}^{(1,1)}$ is the matrix corresponding to the input weights, it is implied that both its indices are 1 since they connect the input with neurons at the first layer. An index above the bias b has also been used to indicate their respective layer; $b^{(1)}$ and $b^{(2)}$ represent biases for first and second layer respectively.

2.3 Training a Neural Network

There are two main learning paradigms for neural networks: unsupervised and supervised learning. In supervised learning, neural networks are trained using labeled input and output

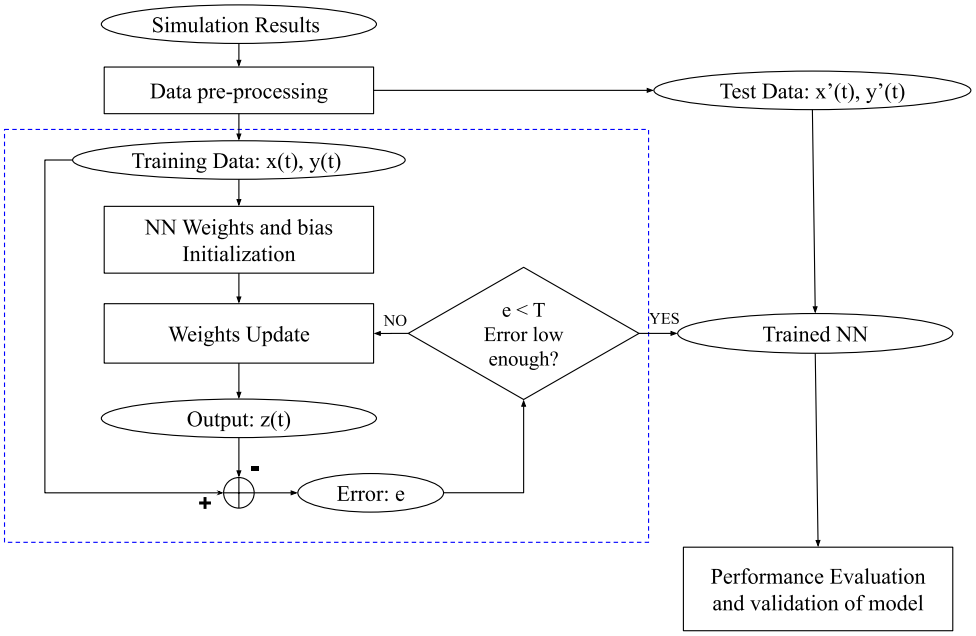

Figure 2.5. Neural network training procedure.

data. Unsupervised learning algorithms do not require someone to understand and label inputs and outputs. This work was developed using supervised learning only. Supervised learning consists of reducing the difference between the correct output and the neural network output in an iterative manner. To achieve this, first the weights of the neural network are initialized, second the input data set must be fed to the neural network, its output is compared to the output data set to calculate the error. The third step consists of adjusting the weights and bias to try to reduce the error. The second and third steps should be repeated until the error has been decreased enough to consider the training is effective. The neural network should be tested against a dataset that has not been used in training to verify the outputs are accurate and to make sure that the model is useful regardless of the data source. This procedure is depicted in Figure 2.5 and it is based on the description in [8].

To update the weights during training a technique called backpropagation is needed. Backpropagation is performed to reduce the error between the output and the reference output. Backpropagation consists of propagating a value delta, which is a function of the error and the activation function. Propagation starts at the output layer and ends at the input layer hence the name backpropagation. A notorious learning algorithm is the Levenberg-Marquardt back propagation (LMBP); its key features are its fast convergence and accuracy [9]. This work was developed using the LMBP algorithm for every neural network implementation. The following subsections will explore backpropagation in more detail.

2.3.1 Standard Backpropagation Algorithm

To obtain the output of the m-th layer, (2.3) can be extended [10], which is given by

$$\boldsymbol{y}^{(m+1)} = \boldsymbol{f}^{(m+1)}(\boldsymbol{W}^{(m+1)}\boldsymbol{y}^{(m)} + \boldsymbol{b}^{(m+1)}), \tag{2.4}$$

where $\mathbf{y}^{(\mathbf{m+1})}$ and $\mathbf{y}^{(\mathbf{m})}$ are the outputs of the $(m+1)$-th and m-th layers respectively. The initial condition for (2.4) is constituted by the initial input vector x. This is shown in (2.5). Note that

the matrix W is a generic term for both layer and input weights W_L and W_I respectively.

$$\boldsymbol{y}^{(\mathbf{0})} = \boldsymbol{x}. \tag{2.5}$$

To train a neural network a data set with input-output pairs $(\mathbf{x_1}, \mathbf{t_1})$, $(\mathbf{x_2}, \mathbf{t_2})$, ..., $(\mathbf{x_k}, \mathbf{t_k})$ is needed [7] [6] [10] ; $\mathbf{x_k}$ and $\mathbf{t_k}$ represent the input vector and the target output respectively. The main goal is to find the weights that minimize the mean square error between the outputs of the neural network $\mathbf{y_k}$ and the target output $\mathbf{t_k}$. The mean square error $G(x)$ can be calculated as

$$G(\boldsymbol{x}) = E[(\boldsymbol{t} - \boldsymbol{y})^T(\boldsymbol{t} - \boldsymbol{y})], \tag{2.6}$$

where $\mathbf{t}$ contains elements $\mathbf{t_1}, \mathbf{t_2}, ..., \mathbf{t_k}$ and $\mathbf{y}$ contains elements $\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_k}$.

According to [10], (2.6) can be approximated using the approximate steepest descent rule. The expectation of the squared error is replaced by the squared instantaneous error resulting in

$$\hat{G}(\boldsymbol{x}) = (\boldsymbol{t}(k) - \boldsymbol{y}(k))^T(\boldsymbol{t}(k) - \boldsymbol{y}(k)). \tag{2.7}$$

The steepest descent algorithm is given by

$$w_{i,j}(k+1) = w_{i,j}(k) - \alpha\frac{\partial\hat{G}}{\partial w_{i,j}}, \tag{2.8}$$

$$b_i(k+1) = b_i(k) - \alpha\frac{\partial\hat{G}}{\partial b_i}, \tag{2.9}$$

where α factor represents the learning rate; (2.8) and (2.9) describe how the biases and weights of the network should be updated in every training iteration.

After calculating the derivatives in (2.8) and (2.9), the approximate steepest descent algorithm is obtained, and is given by

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W^m(k)} - \alpha\boldsymbol{s^m}(\boldsymbol{a}^{m-1})^T, \tag{2.10}$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - \alpha\boldsymbol{s}^m, \tag{2.11}$$

where the sensitivity parameter $\boldsymbol{s}^m$ is defined as

$$\boldsymbol{s}^m \equiv \frac{\partial\hat{G}}{\partial\boldsymbol{n}^m} = [\frac{\partial\hat{G}}{\partial n_1^m}, \frac{\partial\hat{G}}{\partial n_2^m}, ..., \frac{\partial\hat{G}}{\partial n_i^m}], \tag{2.12}$$

and

$$n = \boldsymbol{W}\boldsymbol{p} + b. \tag{2.13}$$

Note that m represents the layer and i is related to the i-th element in that specific layer (every layer has a number of neurons larger than one). Note that $\boldsymbol{s}^m$ is given by (2.14).

$$\boldsymbol{s}^m = \boldsymbol{G}^m(\boldsymbol{n}^m)(W^{m+1})^T \boldsymbol{s}^{m+1}. \tag{2.14}$$

To execute backpropagation, first the input is propagated in the forward direction through the whole network, second the sensitivities must be propagated in the backward direction and third, the weights and biases are updated according to the approximate steepest descent rule in (2.8) and (2.9) [10]. Since the backpropagation starts at the final layer, sensitivity should be initialized with (2.15).

$$\boldsymbol{s}^M = -2\boldsymbol{G}^M(\boldsymbol{n}^m)(\boldsymbol{t} - \boldsymbol{a}). \tag{2.15}$$

2.3.2 Levenberg-Marquardt Back Propagation

Levenberg-Marquardt backpropagation is an algorithm based in Newton's method [11] [12] [13]. Newton's method is designed to minimize sum of squares of nonlinear functions [13]. Assume we have a function $F(x)$ that requires minimization with respect to parameter x. The Newton's method is given by

$$\Delta \boldsymbol{x} = -[\nabla^2 F(\boldsymbol{x})]^{-1} \nabla F(\boldsymbol{x}), \tag{2.16}$$

where $\nabla^2 F(x)$ is the Hessian matrix and $\nabla F(x)$ is the gradient. Assume that $F(x)$ is a sum of squares function, then $F(x)$ is calculated as

$$F(x) = \sum_{i=1}^{N} e_i^2(x) = \boldsymbol{e}^T(\boldsymbol{x})\boldsymbol{e}(\boldsymbol{x}). \tag{2.17}$$

According to [11] the gradient and the Hessian matrix are given by

$$\nabla F(\boldsymbol{x}) = 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{e}(\boldsymbol{x}), \tag{2.18}$$

$$\nabla^2 F(\boldsymbol{x}) = 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x}) + 2\boldsymbol{S}(\boldsymbol{x}), \tag{2.19}$$

where $\boldsymbol{J}(\boldsymbol{x})$ is known as the Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial e_1(\mathbf{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_N(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial e_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{2.20}$$

and

$$\boldsymbol{S}(\boldsymbol{x}) = \sum_{i=1}^{N} e_i(\boldsymbol{x})\nabla^2 e_i(\boldsymbol{x}). \tag{2.21}$$

If we assume that $\boldsymbol{S}(x)$ is very small (Gauss-Newton method), that means that (2.16) can be approximated as

$$\Delta \boldsymbol{x} = [\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})]^{-1}\boldsymbol{J}^T\boldsymbol{e}(\boldsymbol{x}). \tag{2.22}$$

The LMBP algorithm makes a subtle modification to (2.22) [11], it is given by

$$\Delta \boldsymbol{x} = [\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x}) + \mu \boldsymbol{I}]^{-1}\boldsymbol{J}^T\boldsymbol{e}(\boldsymbol{x}), \tag{2.23}$$

where $\boldsymbol{I}$ is the identity matrix

$$I = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{2.24}$$

2.4 One-Step ahead Interference Prediction

Interference has long been considered a deleterious factor that limits the wireless system capacity [14]. At the same time, interference has always been a challenge in the design of a wireless system [15]. Having the ability to predict future interference is beneficial to tackle the effects of interference [14]. Obtaining accurate forecasts of the interference power is a challenging task due to the time varying nature of wireless channels [8]. Various researchers have proposed different interference prediction methods in [9], [15] and [16].

According to [8], the interference power prediction can be classified into statistical and AI methods. Statistical interference power prediction implies that some statistical properties of the interference power, such as mean interference, or even full probability distributions are used to estimate future interference values [8]. On the other side, AI-based interference prediction methods use a machine learning algorithm to approximate the behavior of the channel with a model that uses past interference values as inputs and future interference values as outputs [8]. In this thesis, one of the methods that is going to be explored is the use of the Nonlinear Autoregressive Neural Network Architecture (NARNN) for interference power prediction.

2.4.1 Nonlinear Autoregressive Neural Networks

Interference power in a time series is difficult to forecast accurately due to the random nature of the wireless channel [8]. Linear models are often used but a nonlinear approach was utilized in [8] as a novel technique to model the random variations in the wireless channel. A NARNN was used as part of this work for effective time series prediction of the interference power.

The NARNN is a recurrent dynamic neural network with feedback connections enclosing layers of the network; this implies that the current output depends on the values of the past outputs [17]. The NARNN is defined as [18]

$$y(t) = f(y(t-1), y(t-2), y(t-3), ...y(t-n)) + \epsilon(t), \tag{2.25}$$

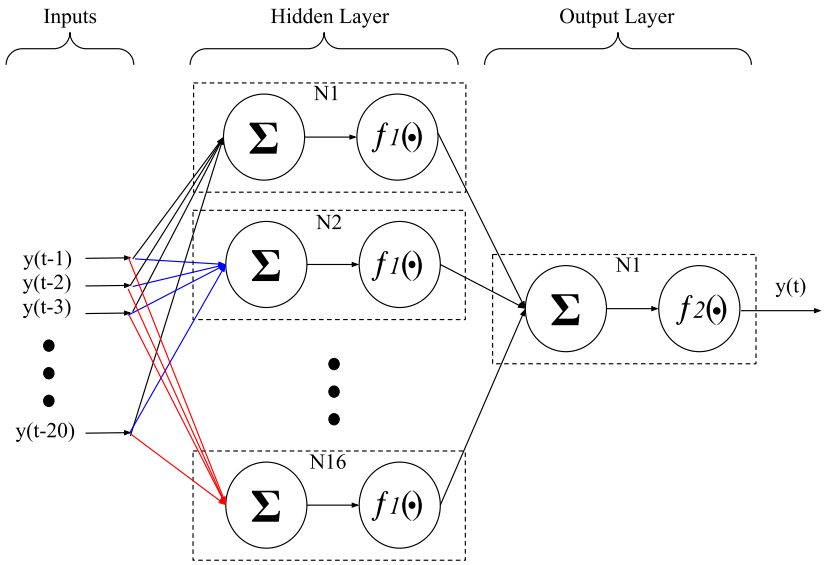

Figure 2.6. NARNN architecture.

where y is the data series for the combined interference of N-1 interferers [8], $f_1(\cdot)$ and $f_2(\cdot)$ represent the log-sigmoid and linear activation functions, n is the input delay of the interference time series and ϵ is the error due to the approximations of the neural network. The architecture of the NARNN is shown in Figure 2.6. The NARNN requires an specific number of delays, hidden nodes, activation functions and an efficient training algorithm [8] [19]. According to [20], the aforementioned parameters can be found by trial and error. Regarding the training algorithm, LMBP was utilized in [8]; LMBP has fast convergence and accuracy [11] [21].

2.5 Multiple-Step ahead Interference Prediction

In [22] a performance comparison of several deep learning models for time series prediction is presented. The main architectures presented in this article are convolutional neural networks, long short-term memory cells (LSTM), bidirectional LSTM networks and encoder-decoder LSTM (ED-LSTM). The main finding in the article is that the ED-LSTM had the best performance in regards of time series prediction. For this reason, LSTM and ED-LSTM architectures will be explored in Subsection 2.5.1 and Subsection 2.5.2.

2.5.1 Long Short-Term Memory model

Recurrent neural networks are difficult to train, their main drawback is the exploding/vanishing gradient problem [23], which is common when learning long-term dependencies [24]. To overcome this problem, the LSTM architecture was introduced in [25].

The basic LSTM cell is called vanilla LSTM cell. It is composed of an input gate, an output gate, a forget gate and a cell [26]. The forget gate is used to allow the network to reset its state [27]. A key feature of LSTM neural networks is that they are capable of remembering values while the gates are used to regulate the flow of information related with the cell. LSTM cells are trained using supervised techniques using an adaptation of the BPTT (Backpropagation through time) algorithm that considers the respective gates [25]. The vanilla LSTM cell architecture is

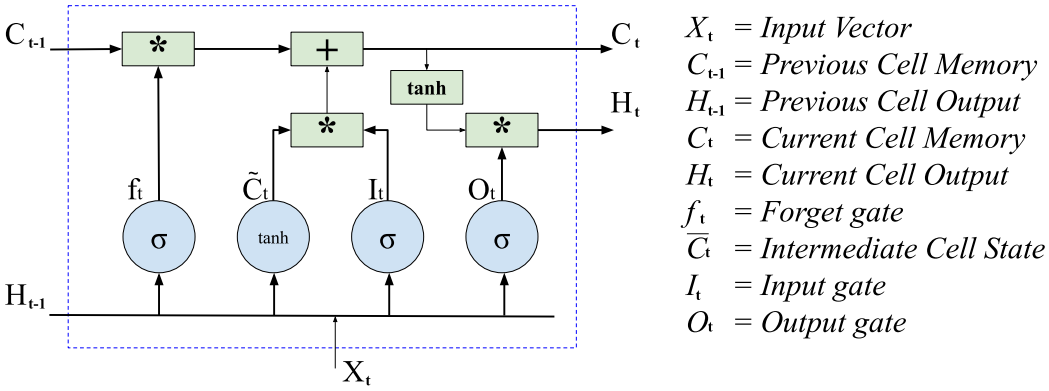

Figure 2.7. Long Short Term Memory Cell.

depicted in Figure 2.7.

According to [22], LSTM networks calculate a hidden state h_t with (2.26). In this set of equations, i_t, f_t and o_t are related to the input, forget and output gates respectively. W and U are the weight matrices. C_t is the internal memory and $\tilde{C}_t$ is a candidate hidden state.

$$\begin{aligned} i_t &= \sigma(x_t U^{(i)} + h_{t-1} W^{(i)}), \\ f_t &= \sigma(x_t U^{(f)} + h_{t-1} W^{(f)}), \\ o_t &= \sigma(x_t U^{(o)} + h_{t-1} W^{(o)}), \\ \tilde{C}_t &= \tanh(x_{(t)} U^{(g)} + h_{(t-1)} W^{(g)}), \\ C_t &= \sigma(f_t C_{t-1} + i_t \tilde{C}_t), \\ h_t &= o_t \tanh(C_t). \end{aligned} \tag{2.26}$$

2.5.2 Encoder-Decoder LSTM networks

In [28], Sutskever et al. presented a novel neural network architecture: the ED-LSTM. They demonstrated remarkably accurate translations from English to French. It is important to note that this architecture allows mapping of sequences with different lengths. It is possible to have as an input sequence an audio recording and as output sequence the text corresponding to the conversation in the recording, this is useful to generate closed captions automatically. According to [22], mapping sequences of different lengths, is equivalent to estimating the conditional probability of an output sequence $[y_1, ...y_q]$ given an input sequence $[x_1, ..., x_n]$. The ED-LSTM architecture is shown in Figure 2.8. In the context of the wireless communications, n inputs can be used for the past samples and q samples as the forecasts of any variable of interest.

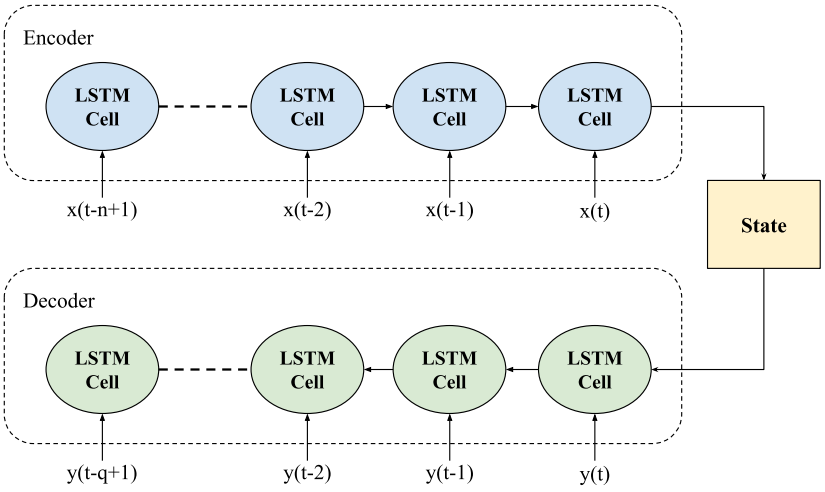

Figure 2.8. ED-LSTM network.

3 ARTIFICIAL INTELLIGENCE FOR NONLINEAR CLOSED-LOOP CONTROL

Control systems are designed to modify the behavior of a system to perform in a desired way [29]. Examples of day-to-day life are: air conditioner (temperature control), steering control of a car, aircraft flight path control system, etc. Considering the air conditioner, elements to modify the temperature are needed, known as *actuators*. To make sure the desired temperature has been reached, *sensors* are needed. The last required element is the control software/logic. It is important to note that the air conditioner is going to keep the desired temperature even if a perturbation has been introduced in the room, this is a key feature of control systems.

3.1 Basic Concepts in Control

According to [30] a control system is an interconnection of components forming a system configuration that will provide a desired system response. In control theory, a cause-effect behavior is assumed for components of a system [30]. This behavior is represented in Figure 3.1. There are two types of control systems if we sort them by the existence of a feedback loop: open-loop and closed-loop control systems. An open-loop control system is shown in Figure 3.2; the controller is used to obtain a desired response but there is no mechanism (sensors and feedback loop) to verify the state of the process. In other words, there is no way to verify that the actual output of the process is the same as the desired output response.

A closed-loop control system is shown in Figure 3.3, the feedback mechanism allows the control system to compare the actual output with the desired output. In addition, having a feedback mechanism implies that perturbations in the process are continuously managed by the controller [29]. A feedback control system often uses a function of a prescribed relationship between the output and reference input to control the process [30].

3.2 PID Controller

The most popular controller used in process industries for closed loop is the PID controller [31]. PID stands for Proportional, Integral and Derivative. According to [32], 97% of the regulatory controllers in industry use the PID algorithm. The PID controller reacts proportionally to the error, integral of the error and derivative of the error [29]; hence it is known as the three term controller. The PID controller in the time domain is defined by

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right), \tag{3.1}$$

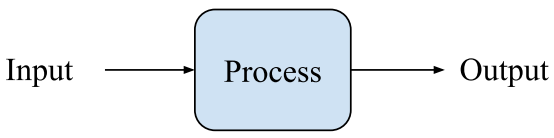

Figure 3.1. Process under control.

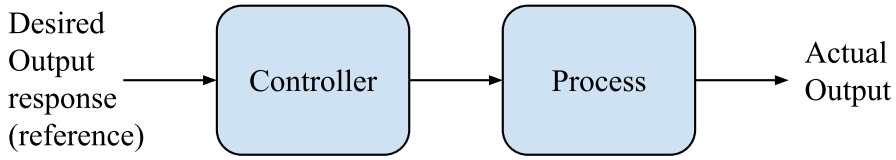

Figure 3.2. Process under control.

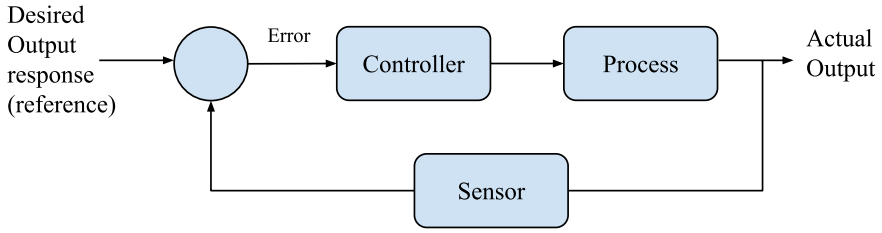

Figure 3.3. Process under control.

where $u(t)$ is the control signal, K_p is a proportionality factor, $e(t)$ is the difference between the plant output and the reference, T_i is integration time and T_d is derivative time.

The expression in (3.1) can be rewritten as [29]

$$\begin{aligned} u(t) &= K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}, \\ K_i &= \frac{K_p}{T_i}, \\ K_d &= K_p T_d. \end{aligned} \tag{3.2}$$

Consider (3.2), after simple manipulations the transfer function G_{PID} of a PID controller is obtained and it is equal to

$$G_{\text{PID}}(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + K_d s. \tag{3.3}$$

From (3.3) it can be noted that a PID controller adds one pole and two zeros to the plant. An example of a PID controller and plant is shown in Figure 3.4. The PID controller is depicted inside the blue dashed rectangle; note the presence of a feedback connection and how the error signal is fed into the terms of PID controller. An integrator is represented as $\frac{1}{s}$ and a derivator is expressed as s in frequency domain.

The main goal of a controller is to make the plant reach a desired reference. Figure 3.5 shows the input reference signal $u(t)$ and the plant output $y(t)$ for the example of Figure 3.4. The figure demonstrates it takes around $t = 5$s for the plant output to reach the reference.

3.3 Model Reference Adaptive Controller

Adaptive control is a technique used for adjusting parameters in real-time in order to maintain a desired level of performance when the parameters of the system are unknown and/or change with

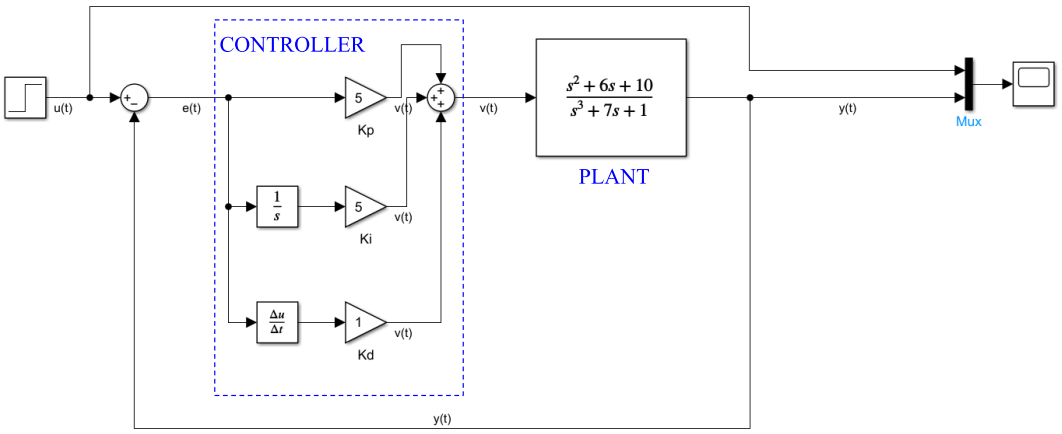

Figure 3.4. PID controller and plant.

time [33]. Model reference adaptive control (MRAC) offers considerably higher performance when compared to conventional closed-loop feedback control [34]. The block diagram of an MRAC system is depicted in Figure 3.6, the MRAC system comprises a reference model, controller, plant and adjustment mechanism. The controller and plant form an inner loop, and the reference model and adjustment mechanism form an outer loop [34]. The outer loop provides the adaptive capabilities of an MRAC controller and the inner loop is typical in conventional feedback control. MRAC controllers are capable of adjusting the variables of the system dynamically by comparing the plant output with the reference response (obtained with the reference model) [35].

Note that inner and outer loops in in Figure 3.6 have been highlighted in blue and orange respectively. A reference signal r is sent to the controller which produces a control signal U for the plant, then the plant output Y_p is compared to the reference signal Y_m to obtain the output error which is sent to the outer loop for the adaptive law/mechanism to adjust dynamically controller parameters. In the following subsection a technique to solve MRAC with MIT rule will be explained.

3.3.1 MIT Rule

MRAC was first proposed by Whitaker in 1950's [33]. MIT developed a rule for the adaptive law of an MRAC controller namely the MIT rule. This technique became the main method to solve MRAC.

Let's define the output error e as the difference between the plant output Y_p and and the model output Y_m, this definition is consistent with Figure 3.6. The output error is then given by

$$e = Y_p - Y_m. \tag{3.4}$$

The main goal is to minimize the error, a cost function $J(\theta)$ (where θ is the adaptable parameter) has to be defined. The cost function is expressed as

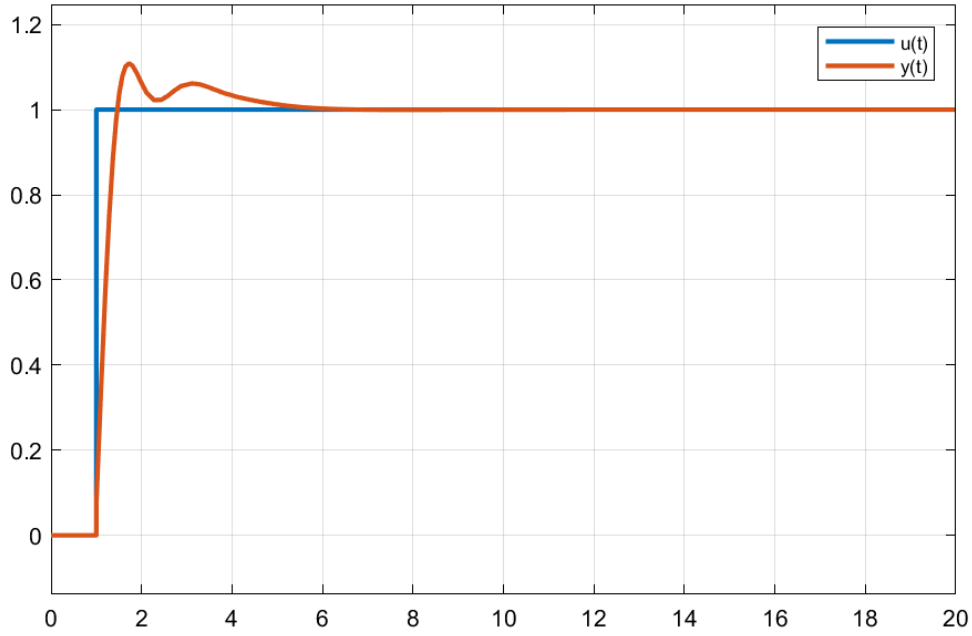

Figure 3.5. Reference and plant output.

$$J(\theta) = \frac{1}{2}e^2(\theta). \tag{3.5}$$

Parameter θ needs to be adjusted so that the cost function is minimized, this implies that changes in parameter θ are made in the direction of the negative gradient of $J(\theta)$ [36], that is

$$\frac{\partial \theta}{\partial t} = -\gamma \frac{\partial J}{\partial \theta}. \tag{3.6}$$

After replacing (3.5) in (3.6) we obtain

$$\frac{\partial \theta}{\partial t} = -\gamma e \frac{\partial e}{\partial \theta}, \tag{3.7}$$

where the term $\frac{\partial e}{\partial \theta}$ is known as the sensitivity derivative, which describes changes in error with respect to the adjustable parameter θ. The γ factor represents the adaptation gain of the controller. Figure 3.7 shows a Simulink® example of MRAC controller presented in [36]. Simulation results of MRAC controller in Figure 3.7 are depicted in Figure 3.8. Note how the plant response is equal to the model reference output after 25 seconds. The current adaptation rate in this example is 0.5. Increasing adaptation rate value would ensure faster convergence but might cause plant instability.

3.4 Neural Network Control Systems

Hagan M. and Demuth H. presented several neural control architectures in [37]. The three main architectures discussed in this work are: Model Predictive control (MPC), Feedback

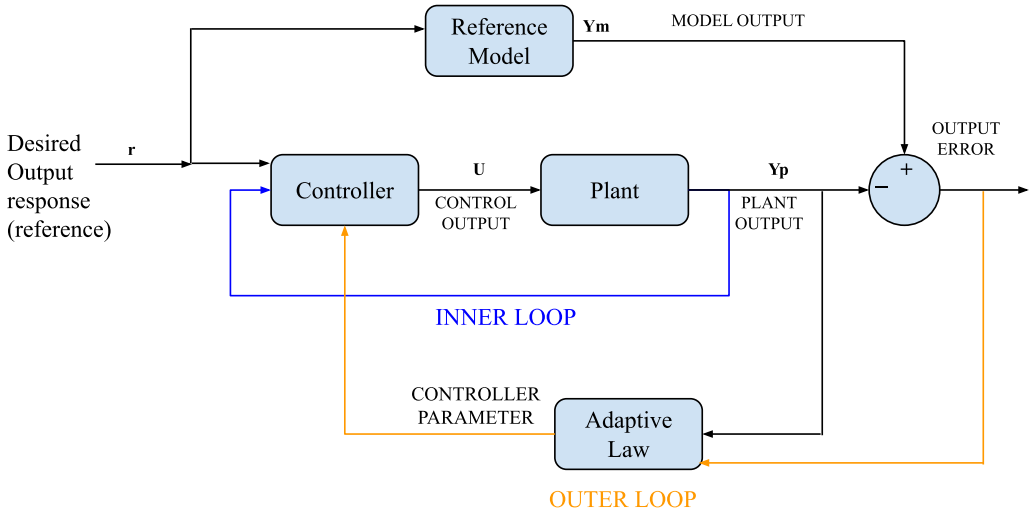

Figure 3.6. Block diagram of MRAC controller.

Linearization control and Model Reference Adaptive Control.

There are two steps required for neural control training: Plant identification and control design [37]. In the plant identification, a neural network is trained to mimic the behavior of the plant we are intended to control (neural plant), the control design stage depends on the neural controller architecture of choice. According to [37], MPC requires a plant model (obtained in plant identification) to predict future states of the plant, an optimization algorithm is also required to determine which control command optimizes future performance. Feedback linearization requires to rearrange the neural plant after it has been trained. In MRAC a neural plant is used to train the controller.

3.4.1 Neural Predictive Control

A predictive controller is depicted in Figure 3.9. The predictive controller is a combination of a neural plant and an optimization algorithm. The neural plant is used to predict future behavior of the real plant, the optimization algorithm determines the control input that gets the best performance out of the plant.

Regarding system identification, neural networks are utilized since they have the capacity of capturing nonlinear dynamics [38]. It is recommended to use a neural network architecture that uses previous control inputs and plant outputs to predict the future plant responses, that is a dynamic neural network. Training must be performed offline using historical data. Although many neural network architectures are suitable to be trained as a neural plant, the NARX architecture was chosen due to its simple structure and reliable performance.

Regarding the predictive control, the optimization algorithm is key for a high-performing controller. A promising optimization algorithm implementation is presented by Soloway and Haley in [39]; this work shows how a Generalized Predictive control algorithm is derived using the Newton-Raphson method as optimization algorithm. The goal of a predictive controller is to minimize a cost function over a finite prediction horizon. The cost function is shown in (3.8) [39].

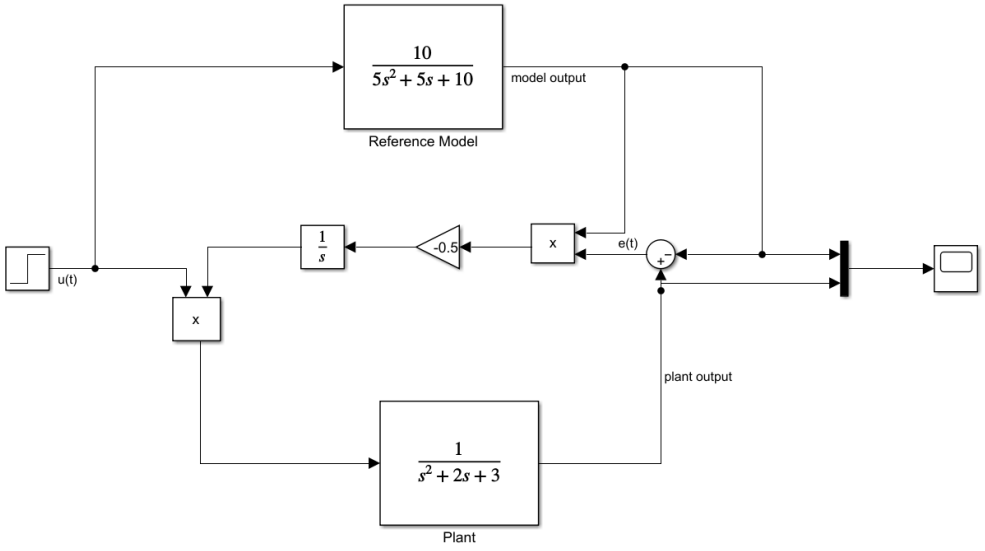

Figure 3.7. Simulink model of MRAC controller.

$$J = \sum_{j=N_1}^{N_2} [y_m(n+j) - y_n(n+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\Delta u(n+j)]^2 + \sum_{j=1}^{N_u} [\frac{s}{u(n+j) + \frac{r}{2} - b} + \frac{s}{\frac{r}{2} + b - u(n+j) - \frac{4}{r}}], \tag{3.8}$$

where N_1 is the minimum costing horizon, N_2 is the maximum costing horizon, N_u is the control horizon, y_m is the reference, y_n is the predicted response of the neural network, λ is the control input weighting factor, $\Delta u(n+j)$ represents variation in u which is $u(n+j) - u(n+j-1)$, s is the sharpness of the corners of the constraint function, r is the range of the constraint, and b is an offset to the range. It is important to note that N_1 and N_u must be less or equal to N_2. The first summation represents the error between the predicted output and the reference. The second summation is necessary to make sure we perform as little variations in the control input as possible. The third summation refers to the shape of constraint function. In [38] a simplified version of (3.8) was used; the reduced version is

$$J = \sum_{j=N_1}^{N_2} [y_m(n+j) - y_n(n+j)]^2 + \lambda \sum_{j=1}^{N_u} [\Delta u(n+j)]^2. \tag{3.9}$$

Note that λ in (3.9) is no longer a function of time step. The cost function J in (3.9) should be minimized with respect to the sequence of control commands $[u(n+1), u(n+2), ..., u(n+N_u)]^T$ namely $\mathbf{U}$ as specified by the Newton-Rhapson algorithm [39]. J is minimized in an iterative manner to find optimal $\mathbf{U}$. This iterative process produces intermediate values of J, namely $J(k)$, where k represents the iteration number. Every iteration of $J(k)$ has a corresponding $\mathbf{U}(k)$ which is denoted by

$$\mathbf{U}(k) = [u(n+1), u(n+2)..., u(n+N_u)]^T, k = 1, 2..., N_{\text{iter}}, \tag{3.10}$$

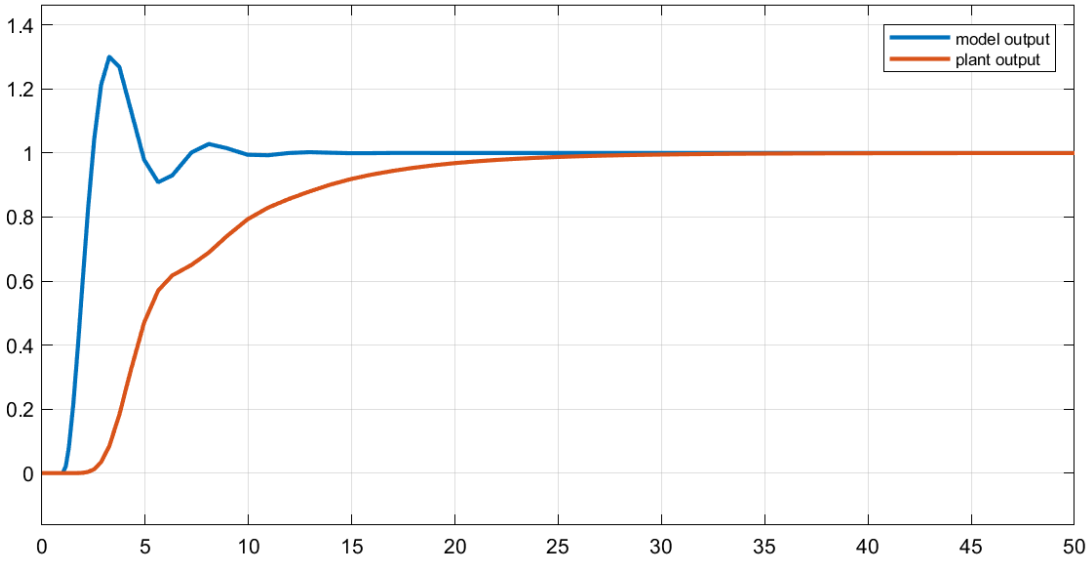

Figure 3.8. Reference and Plant output.

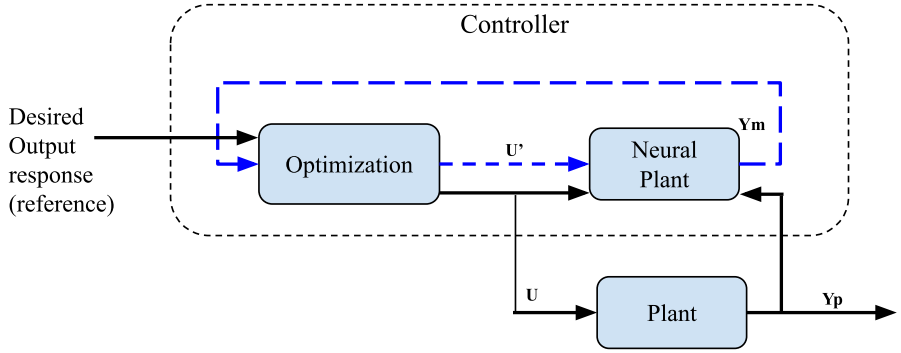

Figure 3.9. Neural Predictive controller.

where N_{iter} represents the number of iterations. The Newton-Raphson method consists mainly of an update rule that in this case must be applied to $\mathbf{U}$ as

$$\mathbf{U}(k+1) = \mathbf{U}(k) - \left(\frac{\partial \mathbf{J}^2}{\partial \mathbf{U}^2}(k)\right)^{-1} \frac{\partial \mathbf{J}}{\partial \mathbf{U}}(k). \tag{3.11}$$

The most challenging task in the iterative computations of (3.11) is obtaining the inverse of the Hessian function, the $m-$ th and $h-$ th elements of the Hessian were derived in [38], they can be calculated as

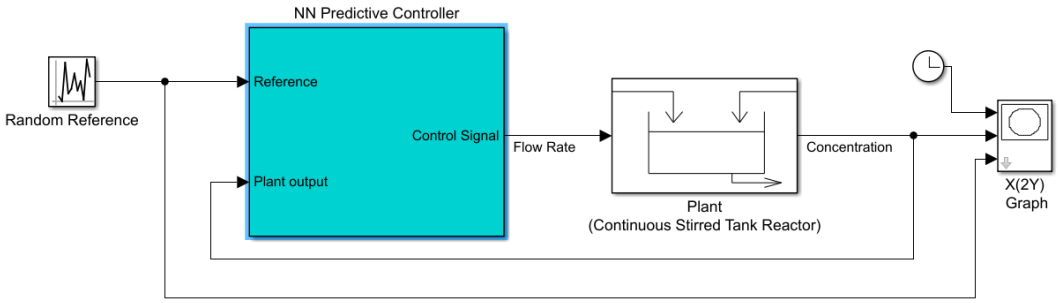

Figure 3.10. Simulink model of a neural predictive controller.

$$\begin{aligned}\frac{\partial^2 J}{\partial u(n+m)\partial u(n+h)} &= 2\sum_{j=N_1}^{N_2}\{\frac{\partial y_n(n+j)}{\partial u(n+m)}\frac{\partial y_n(n+j)}{\partial u(n+h)} \\ &- \frac{\partial^2 y_n(n+j)}{\partial u(n+m)\partial u(n+h)}[y_m(n+j)-y_n(n+j)]\} \\ &+ 2\sum_{j=1}^{N_u}\lambda(j)\{\frac{\partial \Delta u(n+j)}{\partial u(n+m)}\frac{\partial \Delta u(n+j)}{\partial u(n+h)} \\ &+ \Delta u(n+j)\frac{\partial^2 \Delta u(n+j)}{\partial u(n+m)\partial u(n+h)}\}.\end{aligned} \tag{3.12}$$

The Figure 3.10 shows a Simulink® model of a neural predictive controller. For this example, a catalytic continuous stirred tank reactor has been used.

For simplicity, a mask has been used at the neural predictive controller. A detailed description of the neural predictive controller is shown in Figure 3.11. The neural network model is a clone of the real plant, it is not capable of producing control commands. Control commands are generated in the block *predopt* which makes use of plant model predictions to calculate the future performance to choose the best possible control command using the Newton-Raphson method. This example demonstrates a neural predictive controller, plant output and reference are shown in Figure 3.12, the plant has an oscillatory response but recall it is a nonlinear plant, stabilizing such plant is a complex task; the neural predictive controller reduces the difficulties of the controller design to the training of neural plant with plant inputs and responses.

3.4.2 Feedback Linearization Control

The key feature of NARMA-L2 control, is the ability to transform nonlinear dynamics into linear dynamics. This is done by cancelling nonlinearities [40]. The NARMA model is a discrete representation of a nonlinear dynamical system in neighborhood of the equilibrium state [40]. According to [41], for an $n-$ th order nonlinear SISO system, the comapnion form of NARMA is written as

$$\begin{aligned}y(k+d) = F[&y(k), y(k-1), \ldots y(k-n+1), \\ &u(k), u(k-1), \ldots, u(k-n+1)],\end{aligned} \tag{3.13}$$

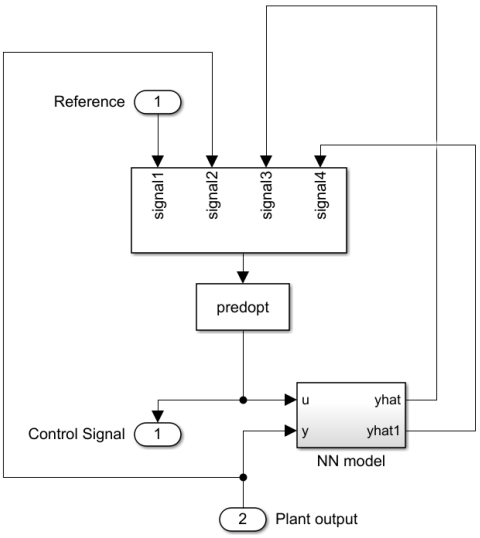

Figure 3.11. Simulink subsystem of neural predictive controller.

where $u(k)$ is the system input (control signal), $y(k)$ is the system output (plant output) and d is the degree, which in turn represents the delay between the system input and the system output; (3.13) is useful for system identification but according to [40], model in (3.13) is not useful to compute the system input/control signal because it is a solution of the inverse dynamics problem. There is a solution to this problem presented in [41], using linear approximations of NARMA, namely NARMA L2. To design a NARMA L2 controller, consider the reference trajectory $y_r(k+d)$ and the system output $y(k+d)$, ideally $y(k+d) = y_r(k+d)$, that means our system (the plant) is following the reference trajectory. Then, a nonlinear controller must have the form

$$\begin{aligned} u(k) = G[&y(k), y(k-1), ...y(k-n+1), y_r(k+d) \\ &u(k-1), ..., u(k-m+1)]. \end{aligned} \tag{3.14}$$

One of the goals of this thesis is to build a neural controller. A neural network needs to be trained with dynamic backpropagation to mimic the function G in (3.14) [42]; using dynamic backpropagation is a time consuming process, for this reason an approximation of the NARMA-L2 model is proposed in [43] by Narendra et al. The approximate NARMA-L2 model (companion form) is

$$\begin{aligned} \hat{y}(k+d) = &f[y(k), y(k-1), ...y(k-n+1), u(k-1), ..., u(k-m+1)] \\ &+ g[y(k), y(k-1), ..., y(k-n+1), u(k-1), ..., u(k-m+1)]u(k). \end{aligned} \tag{3.15}$$

(3.15) can be solved for the control command $u(k)$ [44]. Then, the controller that follows the reference $y_r(k+d)$ is described by (3.16).

$$u(k) = \frac{y_r(k+d) - f[y(k), y(k-1), ..., y(k-n+1), u(k-n+1), u(k-1), ..., u(k-n+1)]}{g[y(k), y(k-1), ..., y(k-n+1), u(k-1), ..., u(k-n+1)]} \tag{3.16}$$

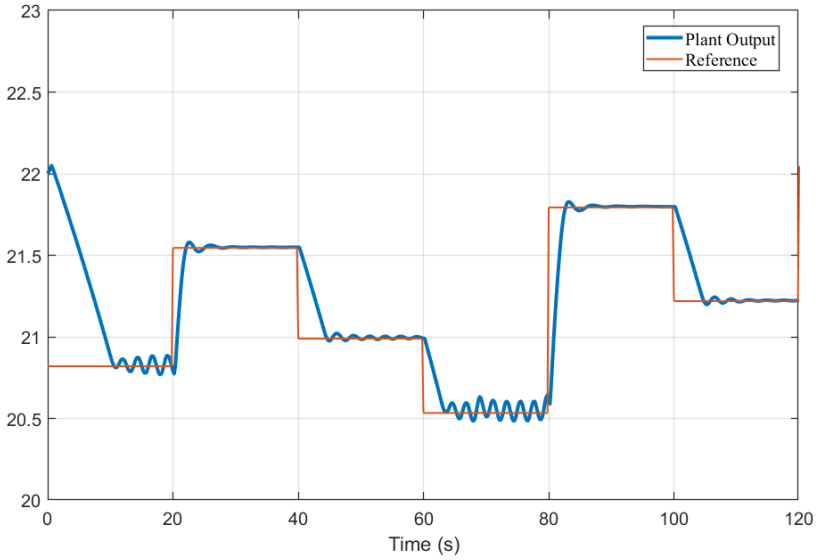

Figure 3.12. Plant output and reference after neural predictive control.

From (3.15) and (3.16), we can see that the NARMA-L2 controller requires two functions to be computed: $f(u, y)$ and $g(u, y)$. For this reason, a neural implementation of the NARMA-L2 controller requires in turn two neural networks to approximate both f and g functions. Implementation of NARMA-L2 controller is shown in Figure 3.13. Note that neural approximations of both f and g functions are enclosed in dashed blue lines and they do not require a deep neural network. Previous outputs of the plant ($Y(t+1)$), previous control commands ($u(t+1)$) and the reference trajectory (Y_n) are used to compute the next control command $u(t+2)$.

As demonstration, a magnetic levitation System was presented in [44] to test efficacy of neural controllers. The goal of this system is to control the height of a magnet using an electromagnet. The equation of motion is given by

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M}\frac{i^2(t)\text{sgn}[i(t)]}{y(t)} - \frac{\beta}{M}\frac{dy(t)}{dt}, \tag{3.17}$$

where $y(t)$ is the height of the magnet above the electromagnet, M is the mass of the magnet, g is the gravity constant and i(t) is the current flowing into the electromagnet. β is a friction coefficient and α is the field strength constant. The Simulink system model for simulation is shown in Figure 3.14 and response of the example plant is shown in Figure 3.15.

3.4.3 Neural MRAC

Model Reference Adaptive Controllers were introduced in Section 3.3. In this section, a neural implementation of MRAC will be showcased. For a neural MRAC system, two key elements are needed; a neural plant replicating the real plant and a neural controller to produce the required control input to achieve the desired reference output by the real plant. The block diagram for an MRAC system is shown in Figure 3.16. A key feature of neural MRAC control systems is the ability to model both the plant and the controller as neural networks. This is remarkably useful for the purposes of this thesis since the neural plant can be used to predict future plant outputs

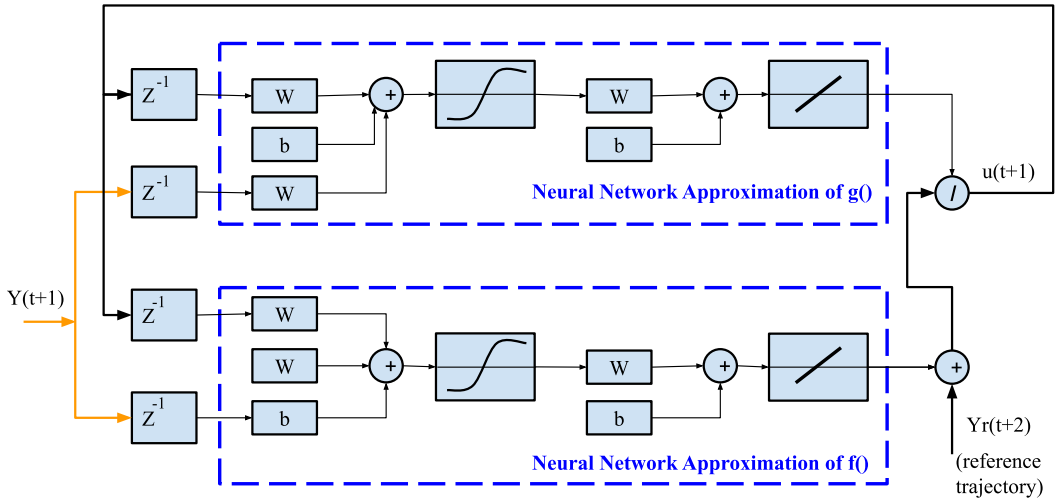

Figure 3.13. Neural implementation of NARMA-L2.

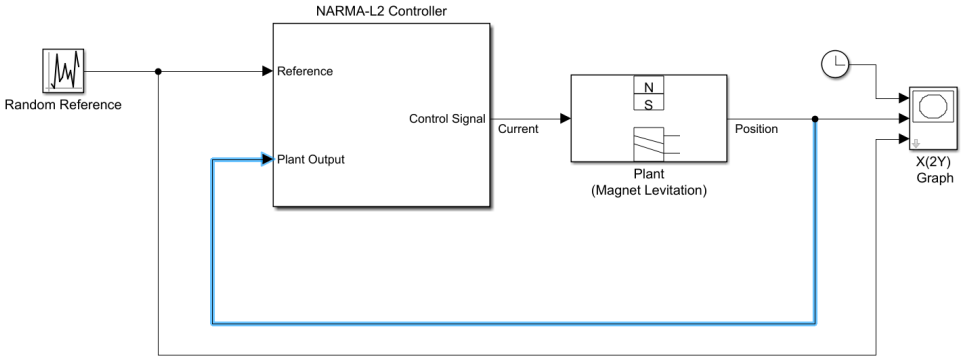

Figure 3.14. Simulink model for NARMA-L2 simulation

and therefore control commands can be prepared in advance.

Regarding the neural controller, a feedforward dynamic neural network can be used. To model the plant, a NARX neural network is commonly utilized.

As an illustration example, a neural MRAC controller is used with a robot arm presented by Hagan et al. in [44]. The equation that describes the motion of the arm is

$$\frac{d^2\phi}{dt^2} = -10\sin\phi - 2\frac{d\phi}{dt} + u, \tag{3.18}$$

where u is the torque provided by the electric motor of the arm and ϕ is the angle of the arm. The objective is to control the angle of the robot arm. The reference model in use is

$$\frac{d^2y_r}{dt^2} = -9y_r - 6\frac{dy_r}{dt} + 9r, \tag{3.19}$$

where r is the reference signal and y_r is the output of the reference model. This reference model was also developed in [44].

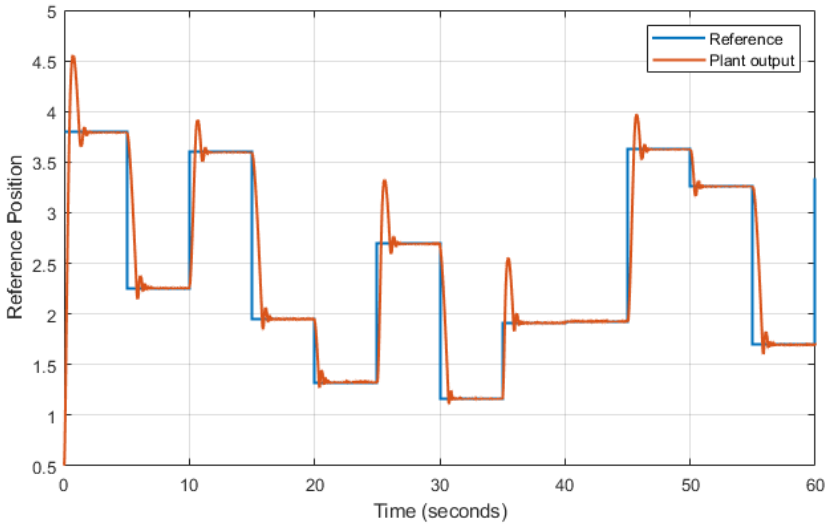

Figure 3.15. Plant output and reference using NARMA-L2 controller.

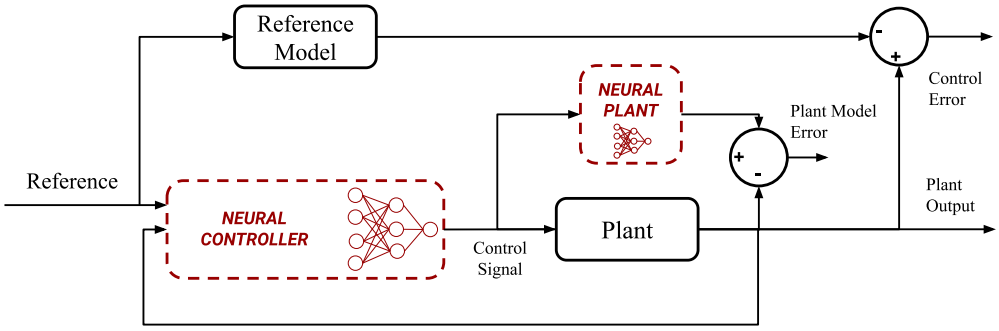

Figure 3.16. Neural implementation of NARMA-L2.

Plant identification has to be performed before training the controller. For this example, the plant inputs and outputs were obtained by sending random reference signals to the real plant and recording both for the neural plant training. The set of random inputs of the real plant and their corresponding plant responses are plotted in Figure 3.17.

To train the controller, the trained neural plant is used once the plant identification has been completed. It is also needed to generate data with the reference model using random inputs and their plant responses. The combined operation of neural controller and neural plant has to be considered, both should be combined in series as depicted in Figure 3.18. The actual controller consists of the first 2 layers which correspond to a feedforward neural network while the neural plant comprises the 2 subsequent layers. Note that the neural plant segment should be kept intact during training, learning for this segment must be disabled. After successful training, the neural controller and neural plant must be separated. In normal scenarios only the neural controller is used, but for the purposes of this thesis, the neural plant will be re-used to obtain predictions of the future plant states. Figure 3.19 shows the plant output after neural control. Note the absence of spikes and oscillatory behavior; the MRAC controller was chosen for the final design due to its smooth plant responses and the fact that its training process delivers neural networks for both controller and plant.

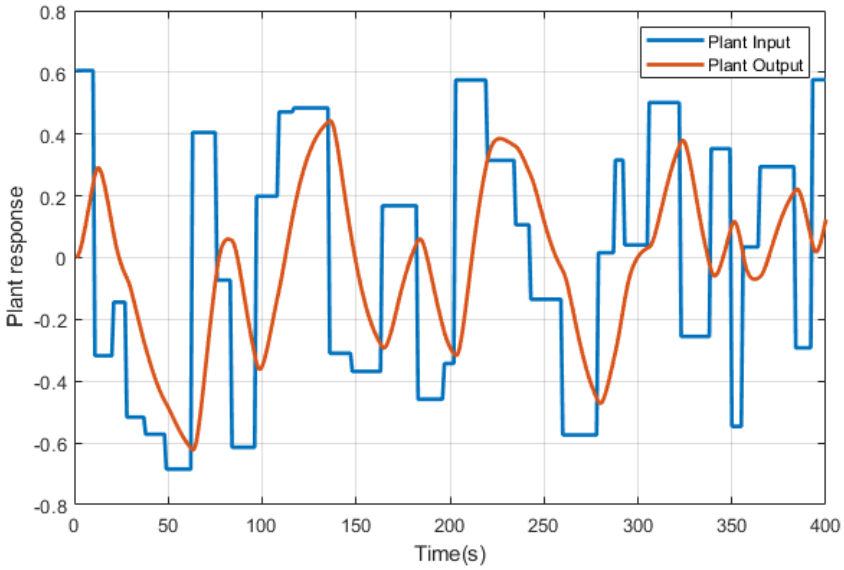

Figure 3.17. Random plant inputs and real plant response.

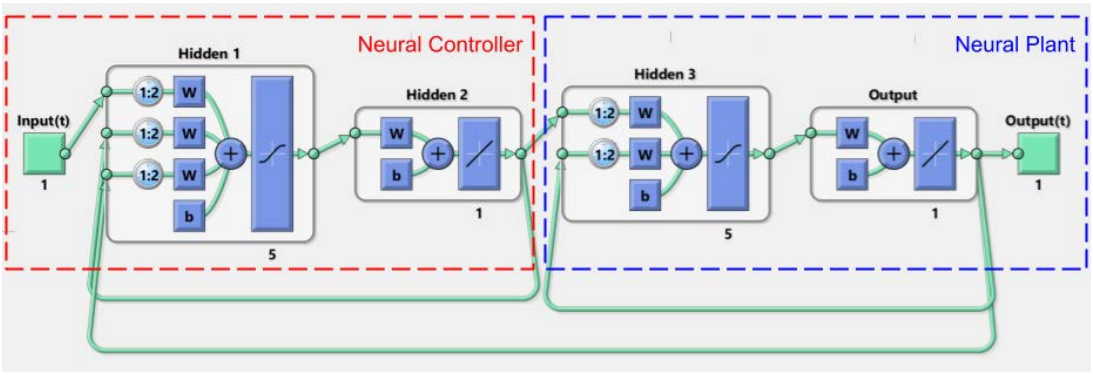

Figure 3.18. MRAC system structure for training in MATLAB.

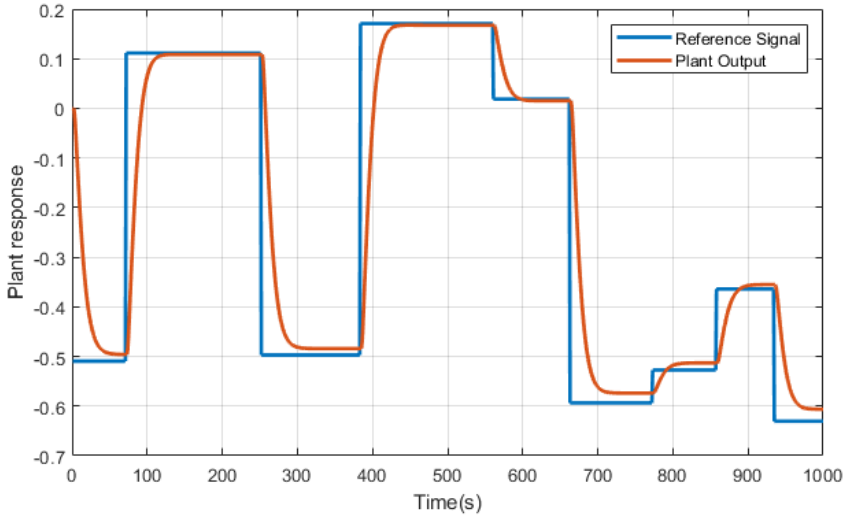

Figure 3.19. Plant output after neural control.

4 AI-BASED PROPOSED SOLUTION AND RESULTS

This chapter presents the core structure of the proposed solution at the system level and the description of its components. The interaction and working principle of these blocks are also explored.

4.1 System Model Description

Consider the Figure 4.1, there is a controller, a plant, and a sensor in a widely known feedback control configuration. Although, there is a difference with the classical configuration: a wireless link is used instead of a wire to interface the controller and the plant. This wireless link is represented by dashed lines. In this proposal, the controller-plant communication is considered to be unreliable due to its wireless nature. On the other hand, the plant-sensor communication is wired and therefore it is reliable. The wireless channel is Rayleigh-faded with N interferers and to keep the latency as low as possible, a single-shot transmission scheme is used.

The purpose of this thesis is to design a system capable of wireless control with two key features: ensuring plant stability in the presence of interference and maximizing energy efficiency. These targets cannot be reached with a classical PID controller, to tackle the challenges imposed by the interference in the wireless channel, an innovative system based on deep learning is presented. An MRAC neural controller, a neural plant and a neural interference predictor comprise the proposal for a wireless channel interference predictor and wireless remote control. These neural components were gathered as a replacement of a classical controller. The proposed system is shown in Figure 4.2; note that wireless sensing capabilities are also present at the controller.

The neural plant consists of a nonlinear autoregressive neural network with exogenous input, its function is to mimic behaviour of the real plant. It is used to make predictions of future plant responses based on the current plant state and latest control signal fed into the real plant. The neural plant only predicts one-step ahead, that is, it produces one prediction per time step. It might seem inconvenient to get only 1 prediction per time step, but this is rather advantageous because every plant response must have a corresponding control command; after a plant response prediction is completed, the corresponding control command is computed at the neural controller, this control command is saved in cache and sent as input to the neural plant which produces a forecast and the process starts over again. Using a neural network for the neural plant requires historical data but allows the designer to avoid approximations, linearization, or complex mathematical modelling for nonlinear plants. The neural controller replaces the normal function of a controller by tracking a reference signal, the advantage of using a neural version

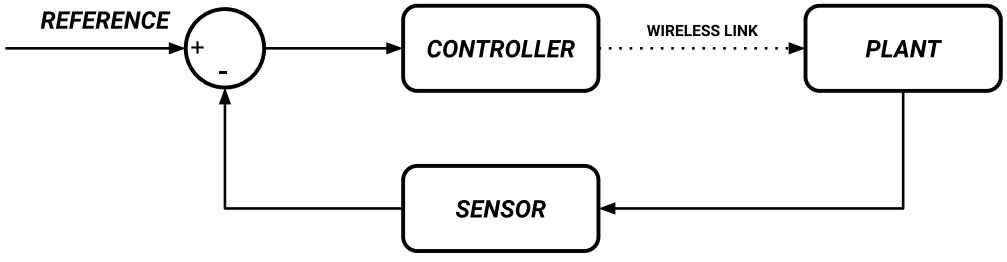

Figure 4.1. System Model.

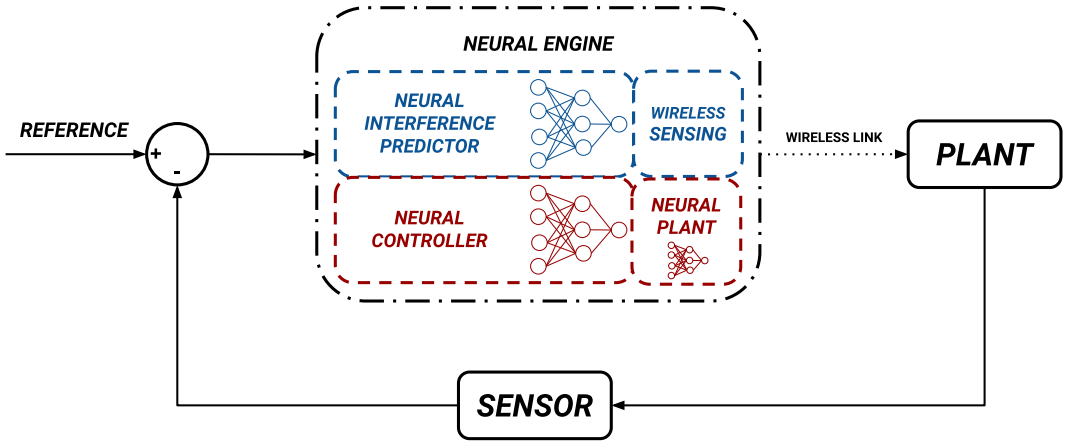

Figure 4.2. System Model.

that is also an MRAC is that input and outputs required for training are defined by the desired behaviour of the plant, i.e., the reference model of the plant. Regarding the wireless capabilities, a neural network based on LSTM cells in an encoder-decoder architecture was used to predict the future interference power. These predictions depend on 5 past interference power values to predict the interference for the next 10 samples; for this reason, wireless sensing is needed. This neural network is referred as neural interference predictor. The module formed by the neural interference predictor and wireless sensing is called the neural wireless module. The neural controller and neural plant comprise the neural control module. Both the neural wireless module and neural control module compose a bigger system, the Neural Engine (NE).

4.2 Setting up simulation

There are two main stages related to the simulation: energy optimization and Simulink model. The first stage consists of optimizing energy consumption by choosing the future control commands that are transmitted at the beginning of the prediction window. Future control commands are computed by the neural controller as functions of neural plant predictions. Recall that this design utilizes a 10-step ahead prediction window for both control and interference forecasts; choosing the future control commands requires solving an optimization problem that takes forecasts in the window as input. The optimization problem is further discussed in Subsection 4.2.1.

For the second stage, Simulink is used due to its capacity for modelling neural networks, controllers and plants. The Simulink model encompasses the neural control system depicted in Figure 4.2, real and predicted interference and the results of optimization of the first stage. Details of the Simulink model are further explored in Subsection 4.2.2

4.2.1 Optimization Problem

Consider a prediction window of interference values $\mathbf{I} = i_1, i_2, ..., i_N$, predicted control commands $\mathbf{C} = c_1, c_2, ..., c_N$ and time steps $\mathbf{T} = t_1, t_2, ..., t_N$, where N is the length of the window; also the transmission of a control command is considered to be successful if the target SINR

is reached. The set of transmission power values required to achieve the target SINR in an entire prediction window is represented by $\mathbf{P} = P_1, P_2, ..., P_N$. Energy efficiency is guaranteed by minimizing the total power needed for successful transmissions $P_T = \sum_1^N P_n$. Interference predictions allow computation of $\mathbf{P}$ at time t_0. The ED-LSTM neural network that composes the interference predictor produces forecasts for 10 time steps, i.e $N = 10$ for this design (see Subsection 4.3.4 for a technical justification of this value). The set of commands that ensure minimal power consumption is $\mathbf{C_s}$ and it is transmitted at t_1. $\mathbf{C_s}$ is calculated as

$$\mathbf{C_s} = \mathbf{C}^T\mathbf{Y}, \tag{4.1}$$

where $\mathbf{Y}$ is a vector with elements $0 \leq Y_n \leq 1$. If Y_3 and Y_{10} are equal to 1 then the third and tenth elements of $\mathbf{C}$ are transmitted at t_1, in other words, only the non-zero elements of $\mathbf{C_s}$ are transmitted at t_1. The remaining elements of $\mathbf{C}$ are transmitted in real time, if $Y_4 = 0$ the fourth element in $\mathbf{C}$ is transmitted at t_4. The vector $\mathbf{C}$ in (4.1) is computed at the neural controller; the remaining question is how to compute $\mathbf{Y}$. Calculating $\mathbf{Y}$ requires an optimization problem that considers the power needed to compute the control commands in addition to transmission power, for this reason the vector $\mathbf{X}$ is defined where the n-th element of $\mathbf{X}$ is called X_n. If X_6 is equal to 1, that implies the sixth element of $\mathbf{C}$ is computed. Control commands are calculated recursively at the neural controller, this has a very important implication; a control command C_n, requires previous commands $C_{n-k}, C_{n-k+1}, ..., C_{n-2}, C_{n-1}$ to be computed. All these considerations are presented in the optimization problem

$$\begin{aligned} \min_{X_n, Y_n} \quad & (P_s\mathbf{1})^T\mathbf{X} + (P_1\mathbf{1})^T\mathbf{Y} + \mathbf{P_n}^T(\mathbf{1} - \mathbf{Y}) \\ \text{s.t.} \quad & 0 \leq X_n \leq 1 \\ & 0 \leq Y_n \leq 1 \\ & Y_n \leq X_n \\ & X_{n+1} \leq X_n \end{aligned} \tag{4.2}$$

where,
X_n: Value of 1 to enable computation of n-th plant state (and n-th control command)
$\mathbf{X}$: Sequence of X_n values. Length of this vector is 10.
Y_n: Value of 1 to transmit the n-th control command. This vector is telling us which control commands are going to be transmitted to reduce power consumption.
P_s: Power needed to make a neural plant prediction (One Step).
P_1: Power needed to transmit a message at time step 1 (First Prediction). This value is obtained taking in consideration the SINR needed to achieve the target rate.
P_n: Power needed to transmit a message at time step n. This value is obtained taking in consideration the SINR needed to achieve the target rate. To calculate this, wireless channel predictions (ED-LSTM) are used.

This problem is a linear program, due to the amount of constraints it is recommended to solve it using numerical methods. In this work, the solution to the problem was found using MATLAB *linprog* command.

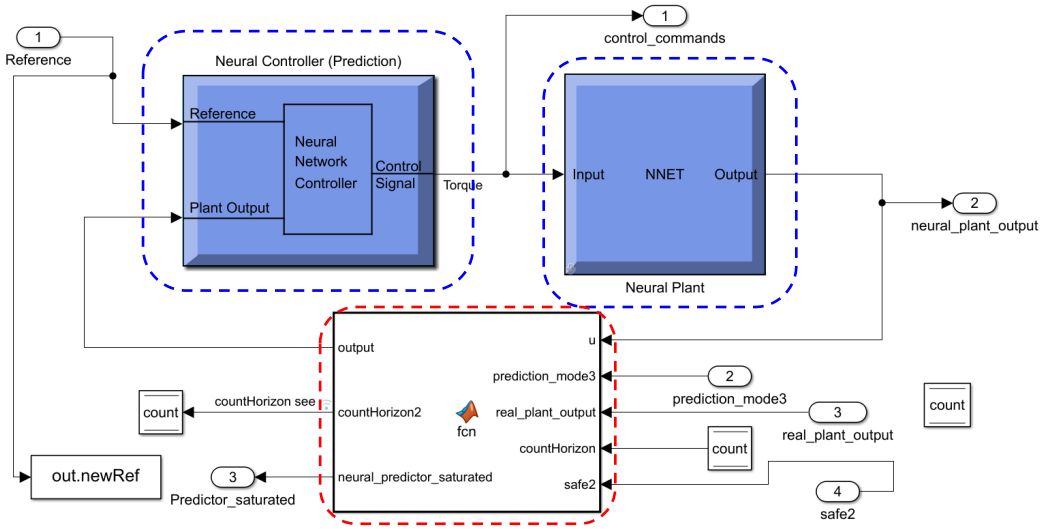

Figure 4.3. Simulink model for neural predictor.

4.2.2 Simulink Model

The Simulink model is composed of 3 main building blocks: a predictive control block, a neural controller for real time control and the real plant. These elements are bonded by MATLAB script blocks that handle switching operations between the real and neural plant and inter-block communication.

The predictive control block is shown in Figure 4.3, it consists of a switch, a neural controller (prediction) and the neural plant. The switch is implemented by the *fcn* block which is a custom-made script inside a Simulink block. The switch is used to select either the output of the real or the neural plant as input of the neural controller. Real plant responses are selected by the switch during real time control to get updated information on the current state of the plant; when computing future control commands, the neural plant is used instead for recursive calculations of control commands. Note that the real plant is not inside this block. The real and neural plant responses are captured through the ports *real_plant_output* and *u* respectively. The choice depends on the boolean variable *prediction_mode3*, if true, then the output of the neural plant is fed as input for the neural controller and the output of the real plant is used otherwise. Note that the *fcn* block also considers prediction saturation (*Predictor_saturated*). The predictor is considered to be saturated if the prediction horizon (*countHorizon*) is reached. Experimentally, this prediction horizon has been found to be 60, in other words, 60 control commands can be predicted without causing considerable deviation of the plant output with respect to the reference signal. Exceeding the threshold of 60 caused severe prediction errors at the neural plant, this finding is further discussed in Subsection 4.3.4. Regarding the neural controller, the MRAC type is used. Its structure was demonstrated at the functional block level in Figure 3.18. In addition, the neural plant uses the NAR architecture.

For the real plant, the dynamics for a ship steering system are given by

$$\dot{y} = 9u(t) - y^3 - 0.8y, \tag{4.3}$$

where $y(t)$ is the output signal of the plant and $u(t)$ is the input signal of the plant. These dynamics can be represented in a Simulink model as depicted in Figure 4.4.

According to [45], for this ship-steering system, it is recommended to use the reference model

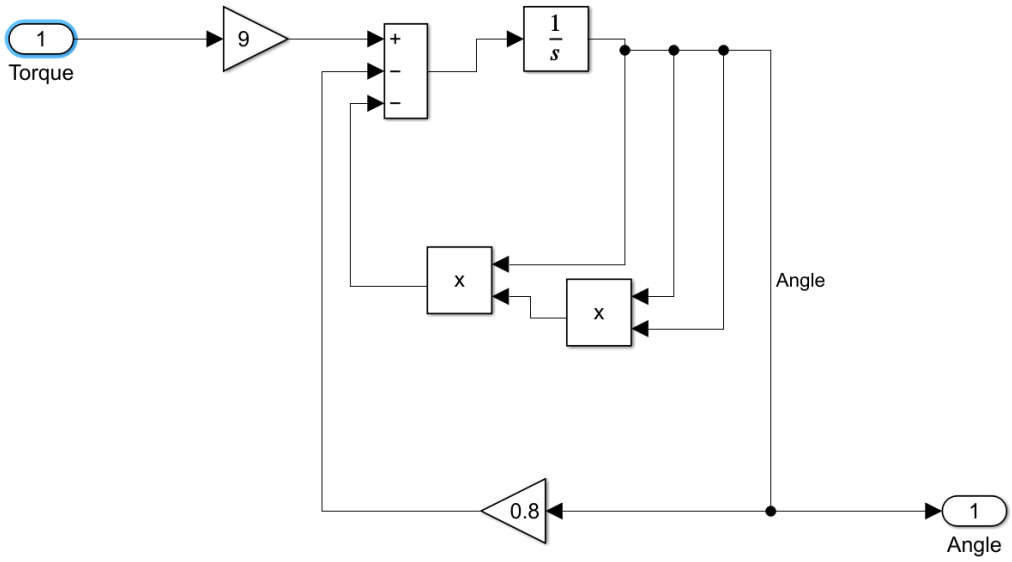

Figure 4.4. Simulink model for the real plant.

$$\dot{y_m} + 2y_m(t) = 2r(t), \tag{4.4}$$

where $r(t)$ is a reference input signal between $[-1, 1]$ and y_m is the output of the plant.

4.3 Results

This section is divided in two subsections. In the first one, the performance of the wireless interference predictor will be explored. In the latter one the combined performance of the wireless module and the control system will be showcased.

4.3.1 Interference Prediction Results

Recall that the interference predictor takes five past samples of measured interference power as inputs to forecast up to ten upcoming interference values. Figure 4.5 provides an insight of the real and predicted values using a prediction horizon of 10. The performance of the neural interference predictor was evaluated by measuring accuracy of predicted interference power using mean squared error (MSE) and mean absolute percentage error (MAPE). Mean square error is given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{4.5}$$

and mean absolute percentage error is calculated as

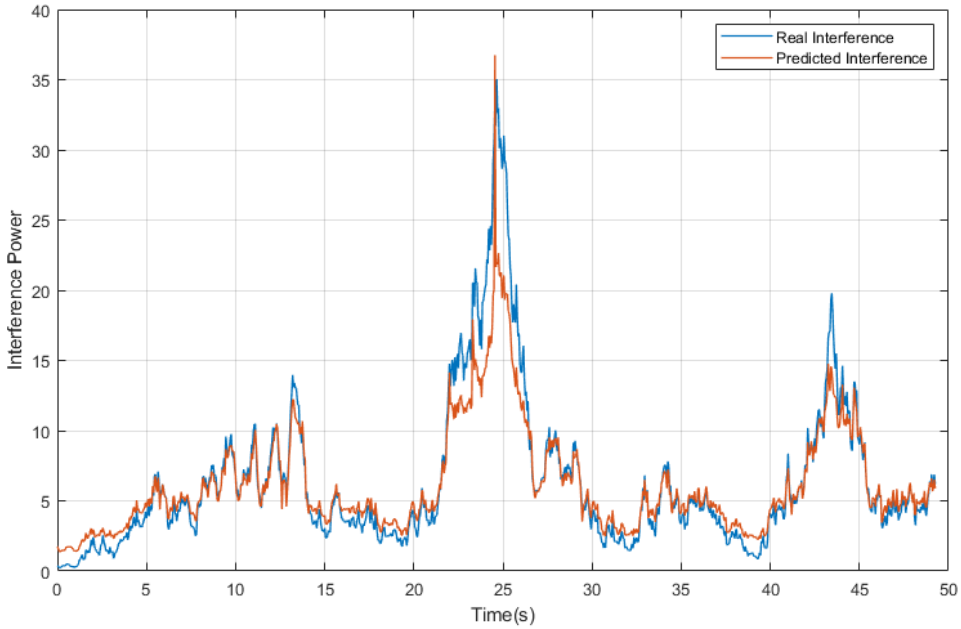

Figure 4.5. Real and predicted interference.

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \times 100, \tag{4.6}$$

where y_i and $\hat{y}_i$ represent the real and predicted interference values respectively [8].

The MSE and MAPE for different prediction horizons are shown in Table 4.1. Note that higher prediction horizons lead to increased prediction errors. This behavior is expected since predicting farther in the future is more effortful than predicting immediate events. To realize the positive impact of reducing the prediction horizon in the prediction errors see the forecasts made with a prediction horizon of 1 in Figure 4.6. While there is a noticeable improvement in performance when using a small prediction horizon, it is preferable to have a large prediction horizon as will be demonstrated in Subsection 4.3.4. Predicting interference farther in the future allows the controller to compute and send future control commands; this is crucial to achieve power savings and to ensure plant stability amidst interference.

For the final version of the wireless-control system a neural interference predictor with ED-LSTM architecture and a prediction horizon of 10 steps was used. The sampling period of the simulation is 0.05 seconds, which for a prediction horizon of 10 is equivalent to 0.5 seconds available for computation and transmission of future control commands; this period of time makes the transmission feasible but fast computations and short delays are needed.

4.3.2 Performance of Alternative Interference predictors

The neural interference predictors presented in this subsection are not part of the final design. However, they have many potential applications in the wireless communication domain that might be of interest for the reader. The first architecture that was evaluated is the NAR presented in Section 2.4. This architecture was utilized in [8] as a novel method for interference prediction.

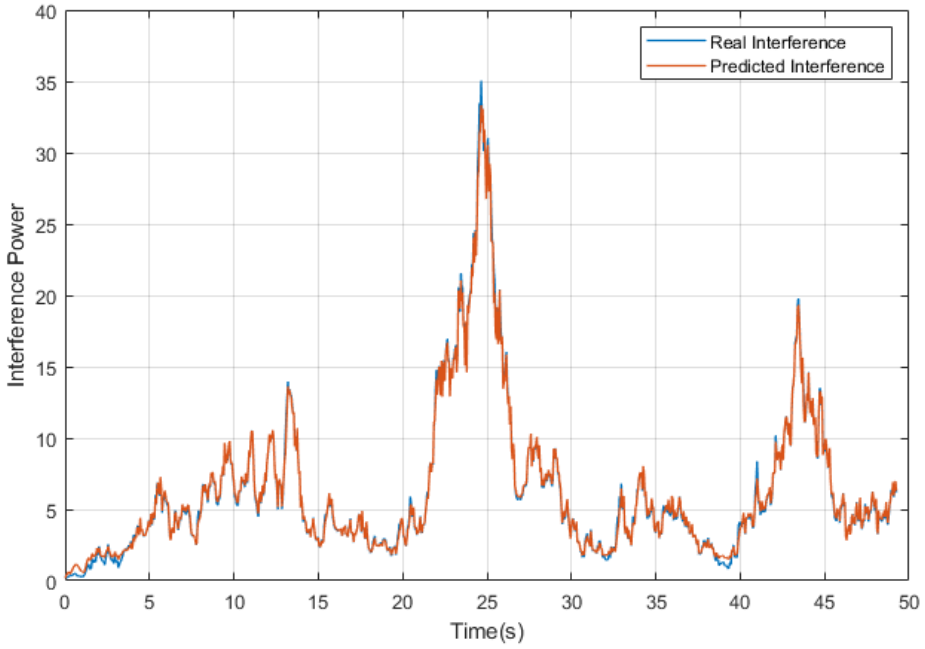

Figure 4.6. Real and predicted interference with a prediction horizon of 1.

Results in [8] were replicated as part of the selection process of the neural predictor for the final design. Figure 4.7 depicts the real interference power and predictions made by a NAR neural network. These results were achieved with a prediction horizon of 1 step. The MAPE for this experiment was 7.83% which makes the NAR performance very close to the 7.07% obtained by ED-LSTM in Subsection 4.3.1 for the case of one-step ahead prediction. Although both architectures have similar performances, the NAR architecture is far less complex. The NAR neural network consists of a shallow neural network with 17 neurons while the ED-LSTM has many dozens of neurons making the NARNN the least computationally expensive prediction method. The NAR arquitecture was abandoned for the reason that there is no use for NAR model in the context of multi step ahead prediction of interference; this model only produces one output which is ahead in time n steps. In contrast, the ED-LSTM produces predictions for every n_i step in the prediction horizon, which is a richer set of information that allows scheduling transmission of control commands for maximum energy efficiency.

To evaluate the performance of the wireless predictor, instantaneous error $E_i(n)$ is used with the definition

$$E_i(n) = \frac{|I_p(n) - I_r(n)|}{|I_r|} \times 100, \tag{4.7}$$

Table 4.1. MSE and MAPE for different number of prediction steps

Metric	1	2	3	4	5	6	7	8	9	10
MSE	0.09	0.54	0.82	0.9161	1.66	2.26	2.75	3.13	3.41	3.57
MAPE	7.07	9.31	11.96	16.95	19.82	23.22	25.96	27.57	28.57	29.32

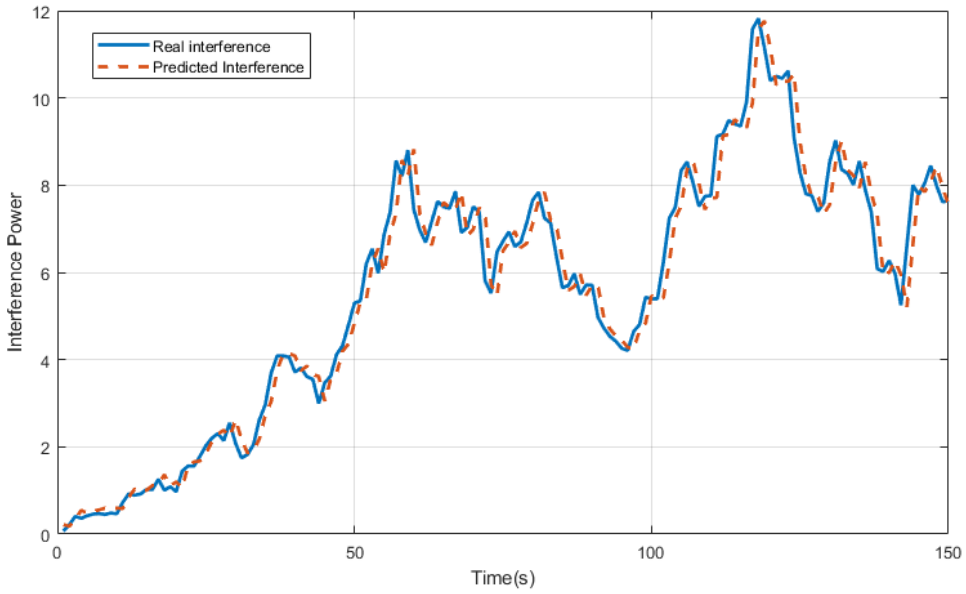

Figure 4.7. Real and predicted interference with a NAR model.

where $I_p(n)$ is the predicted interference power and $I_r(n)$ is the real interference. Figure 4.8 illustrates the CDF of the prediction errors. It can be noted that 90% of samples have a percentage error E_i of 7.5% or less for a control horizon of 1 and E_i of up to 50% for a control horizon of 10. In addition, the highest values for E_i are 242% and 550% for control horizons of 1 and 10 respectively.

Another relevant prediction architecture is a variant of the NAR model, the Nonlinear Autoregressive with Exogenous input model (NARX). In this model, a time series is used for one of the inputs and the output; but the exogenous input should be selected to be a variable related to the time series. For this experiment, the output of the neural network was fed back to the exogenous input. This might seem counter intuitive to the norm but doing this ensures that the neural network uses information of both past and new samples. The NARX predictor achieved a MAPE of 7.28% for one-step ahead prediction which is on par with the 7.83% of the NAR predictor. Experiments showed that the NARX predictor produced forecasts with MAPE of 29.6% for 2-step ahead prediction. For this reason, the NARX architecture is not suitable for the final version of the system.

4.3.3 Comparison of Interference Predictors

Three neural architectures were analyzed to find the most suitable model for interference prediction. ED-LSTM, NAR and NARX architectures were compared based on their prediction accuracy, complexity and maximum acceptable prediction horizon. Table 4.2 sets these architectures side by side. Regarding the prediction accuracy of these architectures, the ED-LSTM exhibited the best performance of the three with a MAPE of 7.07% for one-step ahead prediction, NAR and NARX displayed a MAPE of 7.83% and 7.28% which are nearly as low as ED-LSTM errors. In addition, NAR and NARX are less complex than ED-LSTM; they were designed to have 14 and 16 neurons respectively while the ED-LSTM was devised with 150 neurons. Up to this point, NAR and NARX seem to be a more suitable choice for interference prediction but there is a key advantage of ED-LSTM over NAR/NARX. The ED-LSTM model used as interfer-

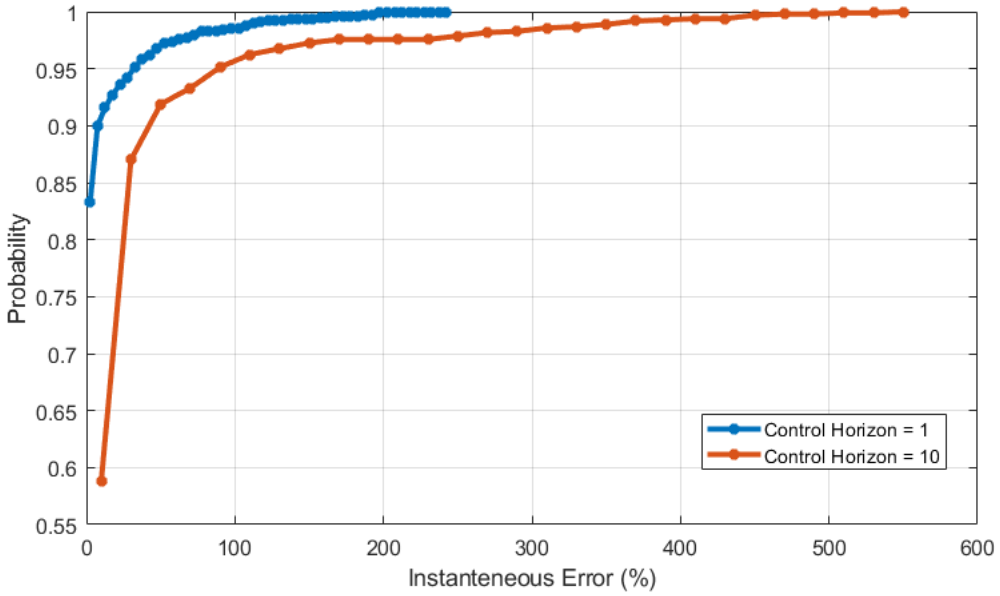

Figure 4.8. Cumulative distribution function for interference prediction errors.

ence predictor in this thesis produces forecasts $f(t_1), f(t_2), ...f(t_N)$ with a prediction horizon N at time t_0 while the NAR/NARX models are capable of providing only $f(t_1)$. Autoregresssive models produce only 1 output regardless of the number of delays used at the input nodes, this is a disadvantage since forecasts for the complete horizon are essential for optimizing energy consumption.

The ED-LSTM was selected as the ultimate choice for the neural interference predictor because of its accuracy and ability to produce forecasts for the complete prediction window. In addition, it is capable of providing outputs with a different size than the input (in this design 5 inputs produce 10 outputs). Nevertheless, it is important to mention that both NAR and NARX models might be suitable in other scenarios, if low processing power or a prediction horizon of 1 are needed, nonlinear autoregressive models should be preferred, their low complexity makes them very appealing for such applications.

Table 4.2. Comparison of interference predictors

Architecture	NAR	NARX	ED-LSTM
# of Neurons	14	16	15 cells /150 neurons
# of layers	2	2	4
Complexity	Low (shallow NN)	Low (shallow NN)	High (Deep NN)
Past samples	3	3	5
Prediction Horizon	1 step	1 step	10 steps
MAPE	7.83%	7.28%	7.07 %(1 step)/ 29.32% (10 steps)

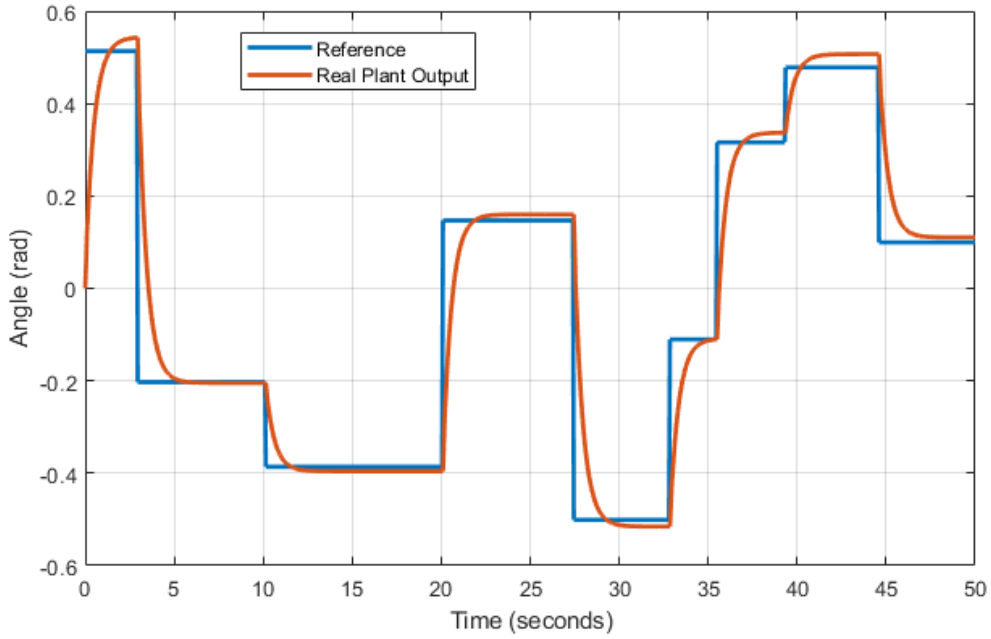

Figure 4.9. Reference and response of real plant.

4.3.4 Neural Engine Results

Wireless and control predictors were tested using Simulink. The response of the real plant in the simulated system can be seen in Figure 4.9. The plant is able to follow the reference trajectory when the neural controller is in use, it takes around two seconds to reach the reference signal, this period of time is reasonable considering the plant model corresponds to a ship steering system. Faster convergence in other applications is possible, it all depends on the reference model and limitations imposed by the plant dynamics. Note the plant response has no oscillatory behavior but there is a slight deviation from the reference when the reference signal reaches its top value (0.5). These results demonstrate effective switching between the real and neural plant since there isn't any considerable deviation from the reference.

Regarding power consumption of the system, to measure the reduction in energy consumption after optimization, a metric called η has been defined and it is given by

$$\eta = 1 - \frac{(P_1\mathbf{1})^T\mathbf{Y} + \mathbf{P}_n^T\mathbf{Y} + \mathbf{P}_n^T\overline{\mathbf{Y}}}{\mathbf{P}_n^T\mathbf{1}}, \tag{4.8}$$

where P_1 is the power needed to transmit a message at time step 1 (first prediction), P_n is the power needed to transmit a message at time step n, $\mathbf{Y}$ is a vector of length 10, same as the prediction horizon of the system. Every element is actually an element of Y_n, which needs to be 1 in order to transmit the n-th control command at time step 1. η values range from 0 to 1 and provide information on the percentage of improvement in energy consumption using optimization compared to not using any optimization that is, if a value of 0.3 is obtained, that means 30% improvement in energy usage achieved with optimization. This metric verifies performance in a per-window basis. The experiments revealed an average η of 0.8322 which implies that power consumption has been reduced in roughly 17%.

It is also important to consider the impact of wireless communications failure on control

and plant states. Loss of control messages is equivalent to not providing a control signal in a wired controller-plant system, this can cause the plant output to deviate from the reference. To quantify deviation of the plant response with respect to the reference, a metric that stores the accumulated difference between the reference and the real plant response has been defined as

$$E_c = \sum_{n=0}^{N} |Y_m(n) - Y_r(n)|, \tag{4.9}$$

where E_c is the accumulated error, $Y_m(n)$ is the actual output of the plant, $Y_r(n)$ is the reference and N is the total number of time steps considered in the experiment. The lower the E_c, the better the controller is. Every experiment in the final design was executed under different channel conditions and using the same neural controller; the controller effectiveness can be impacted by an abrasive wireless environment, the accumulated error E_c might increase if there is low transmission power available or if there is a high number of interferers.

The first set of results show the impact of the transmission power in the accumulated error,these results are presented in Figure 4.10. Note that different curves for predictive control horizons have been considered; it is observed that the higher the control horizon window, the lower the accumulated control error E_c. This is explained by the fact that some control messages might not be delivered correctly specially in situations where the interference reaches peak values, increasing the control horizon has one key implication: a higher number of control messages is sent in advance (at time step t_1) this messages act as a backup in case of communication failure. Note that not all control messages are sent at t_1 and interference predictions are not perfect, there is also the possibility of loss of communication for some time steps for which corresponding commands were not sent at t_1, when this happens, E_c increases. Although this design was targeted at energy efficiency, predictions have an additional benefit, they mitigate the effects of the unreliable wireless channel in transmission of control commands.

From Figure 4.10 can also be seen that E_c is reduced as the transmission power available is increased. This behavior can be explained by the fact that the transmitter is able to send successfully a higher number of control commands when the target SINR is easier to achieve. Another important finding is that the control horizon is more relevant for low power regime. While there is a difference of 20 units in error between control horizons 10 and 1 at around 8 mW power, at 35 mW this difference is almost negligible. Experiments presented in Figure 4.10 consider 5 interferers.

Next, the impact of the number of interferers in the plant output was explored using the same metric E_c as reference. Consider Figure 4.11, curves for three different control horizons have been drawn in a plot that depicts increase of E_c when there is a high number of interferers. For this experiment, transmission power was fixed; this implies reduction in SINR values as the amount of interferers is increased. As a consequence of reduced SINR, the amount of control commands that are sent succesfully is reduced and the accumulated error is increased. It is clear that a higher control horizon has a smaller E_c. Also note that the error difference between control horizons 5 and 10 are not considerable until interferers population reaches 12. Based on these results, If the target is to optimize computational resources, limiting the control horizon to 5 could help to save computational resources while keeping a similar control error.

Limitations of the neural controller-plant system were also studied by making control predictions at the neural plant without feedback from the real plant. The same controller interacts with each of the plants (real and neural) and for that reason, the difference lies in how good is the neural plant at replicating the real plant behavior. Responses of both the real and neural plant were compared computing the MSE for different prediction horizons. See Figure 4.12, it was observed that the MSE is almost zero for every prediction horizon that is smaller than 60

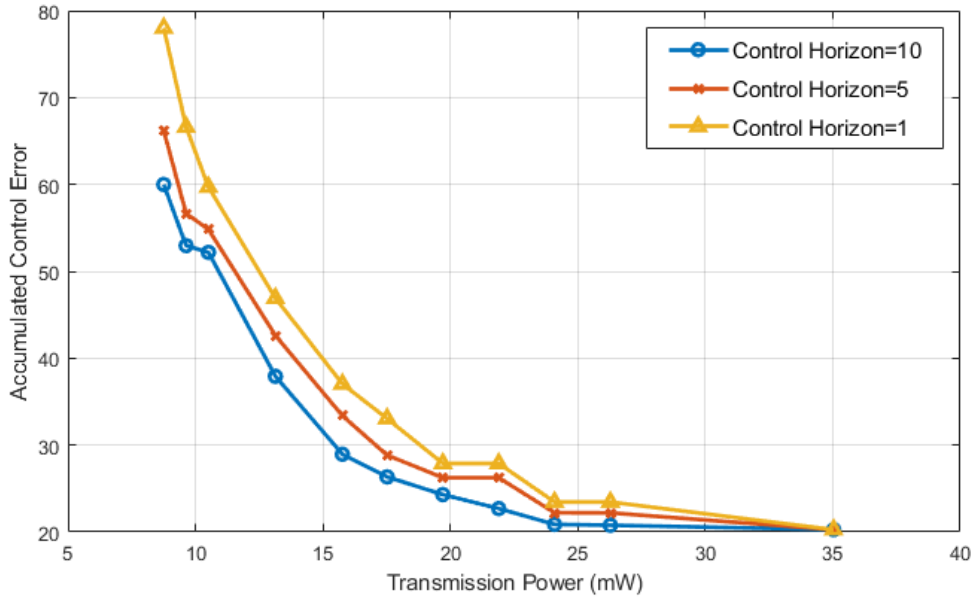

Figure 4.10. Accumulated Control Error vs Transmission Power.

time steps. The errors increase almost linearly for horizon higher than 60; this is an outstanding finding since this horizon represents 3 seconds in real time for a simulation with sampling time of 0.05 seconds. Also note that the wireless prediction horizon of the predictor presented in this work is 10; the neural controller-plant system exceeds by far the wireless prediction capabilities. It might seem counter intuitive to predict control commands for 60 time steps if the highest horizon for interference prediction is 10, but having these prediction capabilities opens the door for future work; for example, in a wireless scenario with very high uncertainty regarding the channel state, it might be better to compute 60 predictions and send them all if accurate wireless channel predictions are not available; also, communication might take longer than predicted to be recovered because of interference not dropping to acceptable levels before all the predicted control commands have been used. In other words, for very harsh environments it is better to use the full capacity of the control predictor. Note that the scenario presented in this thesis does not require the use of the full capacity of the neural control predictor, its limitations were presented for illustration purposes. Control predictions were limited to 10 time steps to align with wireless predictions. Using the same prediction horizon for interference and control prediction is needed for energy consumption optimization.

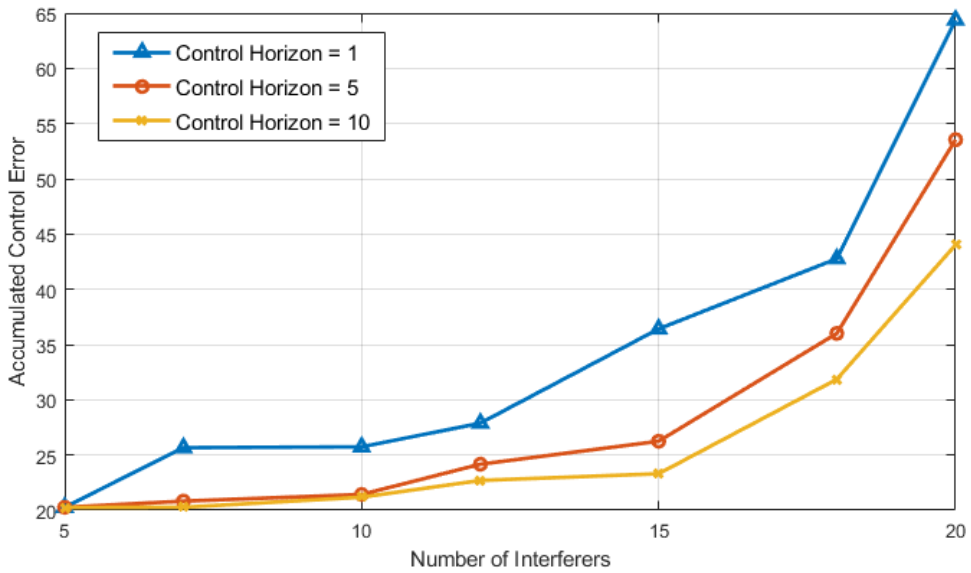

Figure 4.11. Accumulated Control Error vs Number of Interferers.

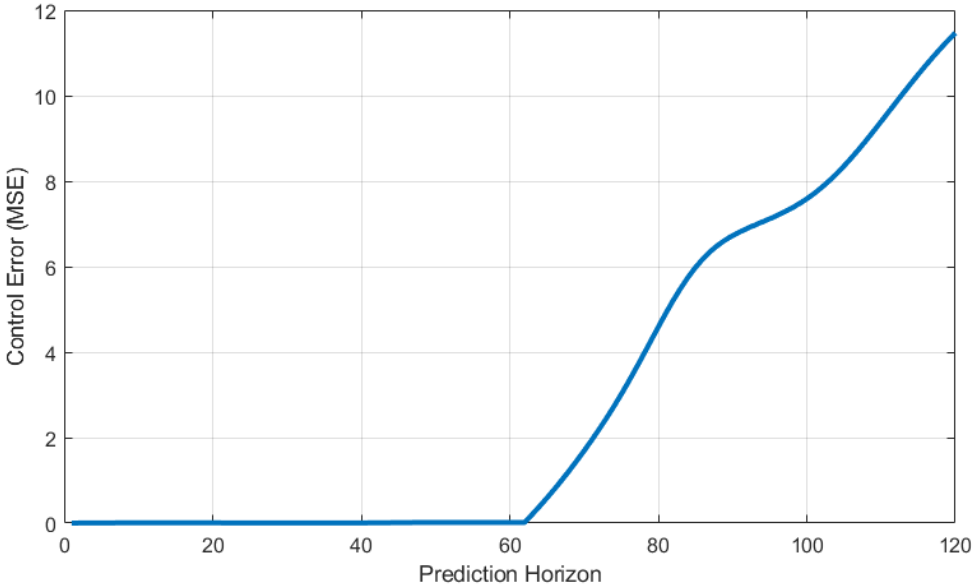

Figure 4.12. Control error (MSE) vs control horizon.

5 CONCLUSIONS AND FUTURE DIRECTIONS

The use of wired connections for transmission of control commands impose some technical difficulties and limitations when it comes to repairs, safety and scalability; these constraints make wireless connections very appealing, but wireless control faces other challenges that come with the unreliable nature of a wireless channel. This thesis work focused on finding and implementing effective neural architectures for prediction of the interference and control of a plant in a wireless environment. The goal of this design consists of overcoming drawbacks of wireless links in controller-plant communication while ensuring plant stability and transmission power efficiency. The final design comprises two main building blocks, the neural interference predictor and the neural controller/plant.

Simulation results demonstrated that the best architecture for the neural interference predictor is the ED-LSTM with a MAPE of 7.07%; other architectures like the NAR and NARX were also tested. It was found that NAR and NARX models are very effective in scenarios that require one-step ahead prediction; forecasts of interference had a MAPE of 7.83% for NAR and 7.28% for NARX. The autoregressive architectures comprise shallow neural networks, their low complexity makes them convenient for applications that require low computational power. Although the ED-LSTM architecture is complex and requires high computational resources, it has a key advantage over the NAR/NARX models: It is capable of multiple-step ahead prediction. The ED-LSTM model produces predictions $f(t_1), f(t_2), ...f(t_{10})$ at time t_0 whereas the NAR/NARX models will produce only $f(t_1)$. The multiple-step ahead prediction was the primary factor for selecting the ED-LSTM architecture for the neural interference predictor. Having multiple interference forecasts enables power optimization and allows the system to compute future control commands.

Regarding the controller, a neural implementation was chosen due to the potential of neural networks of predicting future behavior, forecasting responses of the real plant is key for computation and transmission of future control commands. There were three controller architectures evaluated in this thesis work: predictive controller, L2-NARMA controller and MRAC. The predictive controller consisted of a neural plant with an optimization algorithm that acted as a controller. The algorithm was used to find the optimal control inputs recursively which makes the computation of future control commands very strenuous in addition to requiring high computational power. The L2-NARMA consisted of a rearrangement of the neural plant, this architecture is not convenient for the final design because it lacks the neural controller-plant structure required for computation and caching of future control commands; in addition, experiments showed that plants under L2-NARMA control exhibited an oscillatory response. Having oscillatory behavior is inconvenient from the stability point of view since the goal was to reduce the accumulated control error. The MRAC consists of a neural plant and a neural controller, this split structure is optimal for the final design because it allows the controller to compute and store future control commands.

The interference predictor and the controller had to be interfaced together; this was achieved by proposing and solving an optimization problem depicted in Section 4.2.1; the problem consisted in a linear program, its solution is comprised of vectors indicating the best time step for transmitting control commands. The design presented in this thesis reduced energy consumption in 17% after optimization.

To test effectiveness of the final design in an unreliable wireless channel, changes in the amount of interferers and maximum transmission power for different control horizons were introduced in simulations. It was observed that the number of interferers had impact in the wireless control causing deviations of the plant output from the reference; the higher the number of interferers, the higher the accumulated control error; for the same experiments, control horizons of 5 and 10 exhibited similar accumulated control errors for 10 or less interferers;

if the goal is to save computational resources and the amount of interferers is considerably small (10 or less), it is worth considering to reduce control horizon to reduce computational load. Regarding the transmission power, it was observed that increasing the transmission power reduces the accumulated control error and it also makes the choice for a control horizon less relevant. In other words, larger control horizons are more beneficial when power available for transmission is low.

Limitations in control predictions were also studied; feedback from the real plant response to the neural plant was interrupted to deduce the maximum control horizon. Experiments revealed that the neural plant is capable of predicting plant responses for 60 future time steps, this is equivalent to 3 seconds in simulation, which make neural plants (and controllers) a very enticing approach for controlling a system where communication outage can last for hundreds of miliseconds. This is feasible if the predictions of the maximum control horizon of 60 are used during outage. In this work the control horizon has been limited to 10 to match the prediction horizon of the interference predictor.

For future works, exploiting the full prediction capacity of the neural plant is intended, a system with increased robustness due to the use of the maximum attainable prediction horizon. Results also brought to light that there is a lot of potential for predictive control using nonlinear autoregressive neural networks; their low complexity implies low computational resources are required and therefore are suitable for IoT applications.

The interference predictor can be further improved with adaptive capabilities, the models developed for this thesis are better suited for scenarios where mean interference does not vary in time, in a real life this might not be the case and more sophisticated techniques like deep reinforcement learning should be considered.

6 BIBLIOGRAPHY

[1] De Beelde B., Plets D. & Joseph W. (2021) Wireless sensor networks for enabling smart production lines in industry 4.0. Applied Sciences 11.

[2] Candell R. & Kashef M. (09 2017) Industrial wireless: Problem space, success considerations, technologies, and future direction. pp. 133–139.

[3] Gogolák L. & Fürstner I. (2021) Wireless sensor network aided assembly line monitoring according to expectations of industry 4.0. Applied Sciences 11.

[4] Orfanus D., Indergaard R., Prytz G. & Wien T. (09 2013) Ethercat-based platform for distributed control in high-performance industrial applications. pp. 1–8.

[5] Kim H. (2020) Design and Optimization for 5G Wireless Communications, John Wiley and Sons, Ltd, pp. 397–400.

[6] Demuth H.B., Beale M.H., De Jess O. & Hagan M.T. (2014) Neural Network Design. Martin Hagan, Stillwater, OK, USA, second edition.

[7] Kim P. (2017) MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence. Apress, USA, first edition.

[8] Padilla C., Hashemi R., Mahmood N.H. & Latva-Aho M. (2021) A nonlinear autoregressive neural network for interference prediction and resource allocation in urllc scenarios. In: *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 184–189.

[9] Haenggi M. (2011) Mean interference in hard-core wireless networks. IEEE Communications Letters 15, pp. 792–794.

[10] Lv C., Xing Y., Zhang J., Na X., Li Y., Liu T. & Cao D. (11 2017) Levenberg-marquardt backpropagation training of multilayer neural networks for state estimation of a safety critical cyber-physical system. IEEE Transactions on Industrial Informatics PP, pp. 1–1.

[11] Hagan M. & Menhaj M. (1994) Training feedforward networks with the marquardt algorithm. IEEE Transactions on Neural Networks 5, pp. 989–993.

[12] Bishop C.M. (2006) Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[13] Dreyfus G. (01 2005) Neural Networks: Methodology and Applications.

[14] Zheng G., Krikidis I., Masouros C., Timotheou S., Toumpakaris D.A. & Ding Z. (nov 2014) Rethinking the role of interference in wireless networks. IEEE Communications Magazine 52, pp. 152–158.

[15] Schmidt J.F., Schilcher U., Atiq M.K. & Bettstetter C. (2021) Interference prediction in wireless networks: Stochastic geometry meets recursive filtering. IEEE Transactions on Vehicular Technology 70, pp. 2783–2793.

[16] Varma S. (2015) A machine learning algorithm for interference removal from a signal. In: *2015 National Conference on Recent Advances in Electronics Computer Engineering (RAECE)*, pp. 211–215.

[17] da Costa Lopes F., Watanabe E.H. & Rolim L.G.B. (2015) A control-oriented model of a pem fuel cell stack based on narx and noe neural networks. IEEE Transactions on Industrial Electronics 62, pp. 5155–5163.

[18] Nyanteh Y.D., Srivastava S.K., Edrington C.S. & Cartes D.A. (2013) Application of artificial intelligence to stator winding fault diagnosis in permanent magnet synchronous machines. Electric Power Systems Research 103, pp. 201–213.

[19] Taherdangkoo R., Tatomir A., Taherdangkoo M., Qiu P. & Sauter M. (2020) Nonlinear autoregressive neural networks to predict hydraulic fracturing fluid leakage into shallow groundwater. Water 12.

[20] Ruiz L.G.B., Cuéllar M.P., Calvo-Flores M.D. & Jiménez M.D.C.P. (2016) An application of non-linear autoregressive neural networks to predict energy consumption in public buildings. Energies 9.

[21] Marquardt D.W. (1963) An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics 11, pp. 431–441.

[22] Chandra R., Goyal S. & Gupta R. (2021) Evaluation of deep learning models for multi-step ahead time series prediction. IEEE Access 9, pp. 83105–83123.

[23] Kolen J.F. & Kremer S.C. (2001) Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 237–243.

[24] Hochreiter S. & Schmidhuber J. (1997) Lstm can solve hard long time lag problems. In: *Advances in Neural Information Processing Systems 9*, MIT Press, pp. 473–479.

[25] Hochreiter S. & Schmidhuber J. (11 1997) Long Short-Term Memory. Neural Computation 9, pp. 1735–1780.

[26] Van Houdt G., Mosquera C. & Nápoles G. (12 2020) A review on the long short-term memory model. Artificial Intelligence Review 53.

[27] Gers F.A., Schmidhuber J. & Cummins F. (10 2000) Learning to Forget: Continual Prediction with LSTM. Neural Computation 12, pp. 2451–2471.

[28] Sutskever I., Vinyals O. & Le Q.V. (2014), Sequence to sequence learning with neural networks.

[29] Canete J., Galindo C. & Moral I. (01 2011) Introduction to Control Systems, pp. 137–165.

[30] Dorf R.C. & Bishop R.H. (2000) Modern Control Systems. Prentice-Hall, Inc., USA, 9th edition.

[31] Gopi Krishna Rao P.V., Subramanyam M.V. & Satyaprasad K. (2014) Study on pid controller design and performance based on tuning techniques. In: *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, pp. 1411–1417.

[32] Desborough L. & Miller R. (01 2002) Increasing customer value of industrial control performance monitoring -honeywell's experience. AIChE Symposium Series 98.

[33] Shekhar A. & Sharma A. (2018) Review of model reference adaptive control. In: *2018 International Conference on Information , Communication, Engineering and Technology (ICICET)*, pp. 1–5.

[34] jie Su S., yuan Zhu Y., rong Wang H. & Yun C. (2019) A method to construct a reference model for model reference adaptive control. Advances in Mechanical Engineering 11, pp. 1687814019890455.

[35] Nguyen N.T. (2018) Model-Reference Adaptive Contro. Springer Cham., USA, first edition.

[36] Jain P. & Nigam M.J. (2013) Design of a model reference adaptive controller using modified mit rule for a second order system 1.

[37] Hagan M. & Demuth H. (1999) Neural networks for control. In: *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, volume 3, pp. 1642–1656 vol.3.

[38] Chidrawar S.K. & Patre B.M. (2008) Generalized predictive control and neural generalized predictive control.

[39] Soloway D. & Haley P. (10 1996) Neural generalized predictive control. pp. 277 – 282.

[40] Srakaew K., Sangveraphunsiri V., Chantranuwathana S. & Chancharoen R. (02 2010) Design of narma l2 neurocontroller for nonlinear dynamical system.

[41] Middleton R. & Goodwin G. (1988) Adaptive computed torque control for rigid link manipulations. Systems Control Letters 10, pp. 9–16.

[42] Narendra K. & Parthasarathy K. (02 1991) Learning automata approach to hierarchical multiobjective analysis. Systems, Man and Cybernetics, IEEE Transactions on 21, pp. 263 – 272.

[43] Narendra K. & Mukhopadhyay S. (1997) Adaptive control using neural networks and approximate models. IEEE Transactions on Neural Networks 8, pp. 475–485.

[44] Hagan M., Demuth H. & Jesús O. (09 2002) An introduction to the use of neural networks in control systems. International Journal of Robust and Nonlinear Control 12, pp. 959 – 985.

[45] Patiño H. & Liu D. (02 2000) Neural network-based model reference adaptive control system. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society 30, pp. 198–204.