Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Sondre Bø Hernes

Practical NMPC of Electrical Submersible Pumps based on Echo State Networks

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland and Eduardo Camponogara
Co-Supervisor: Eric Antonelo
July 2020

NTNU
Norwegian University of Science and Technology

Sondre Bø Hernes

Practical NMPC of Electrical Submersible Pumps based on Echo State Networks

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland and Eduardo Camponogara
Co-Supervisor: Eric Antonelo
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Summary

This dissertation aims to control an Electric Submersible Pump (ESP) using the Practical Nonlinear Model Predictive (PNMPC) based on an Echo State Network (ESN). The control of a nonlinear dynamic process can be a challenging task since the dynamic process might not be fully known. Another aspect is that such complex systems may suffer from modeling errors when applying nonlinear model predictive control. This dissertation uses an efficient data driven scheme that overcomes some of the drawbacks of the baseline PNMPC. In the PNMPC approach the system is divided in two responses, a free response that is kept nonlinear and a forced response that is partially linearized. This dissertation will use an echo state network, which is a recurrent neural network used for system identification. There are several advantages for using this approach: one is fast system identification, the other is the analytical computation of derivatives from the data driven model (ESN) for the forced response. This will make the computational complexity for calculating the derivatives unquestionably low. Also a correction filter is implemented to improve the robustness of the controller. The resulting ESN-PNMPC approach will be implemented to control an ESP, which is one of the most widely used methods for artificial lifting in the oil industry.

In this dissertation, an ESP pump model was developed using CasADi, an echo state network that predicts the pump's dynamics with good accuracy, and an ESN-PNMPC controller to follow a reference on the state p_{bh} while maximizing the flow (q). The result concluded with a controller demonstrating good results in being able to follow a reference with a smooth trajectory while optimizing the flow, without any errors like oscillations or overshooting.

Sammendrag

Dette prosjektet går ut på å kontrollere en ESP (electric submersible pump) ved bruk av ESN-PNMPC (Echo state network- practical nonlinear model predictive control) . Å kontrollere en ikke lineær dynamisk prosess kan være utfordrende siden den dynamiske prosessen ikke nødvendigvis er kjent. Et annet aspekt er at det kan oppstå modelleringsfeil i slike komplekse systemer, ved bruken av en kontroller som blir kalt NMPC (nonlinear model predictive control). I denne oppgaven benyttes en effektiv datadreven løsning som fjerner noen av problemene som kommer med PNMPC (Practical nonlinear model predicive control). I PNMPC løsningen er systemet delt inn i to responser, en fri respons som holdes ikke lineært og en tvungen respons som er delvis linearisert. Denne oppgaven bruker et ESN (Echo State Network), som er et tilbakevendende nevralt nettverk som brukes til systemidentifikasjon. Det er flere fordeler for å bruke denne tilnærmingen. Det gir en rask systemidentifisering, og gjør analytiske beregninger av derivater fra den datadrevene-modellen (ESN). Dette brukes til å beregne den tvungne responsen og vil gjøre beregningskompleksiteten for å beregne derivatene utvilsomt lavere. Et korreksjonsfilter er også implementert for å øke kontrollerens robusthet. Denne ESN-PNMPC-tilnærmingen vil bli implementert på en ESP, som er en av de mest benyttede metodene for olje pumping i oljeindustrien.

I denne oppgaven er det utviklet en modell for pumpen ESP med bruk av CasADi, et echo state network som predikerer pumpens dynamikk med god nøyaktighet, og en kontroller (ESN-PNMPC) for å følge en refereanse på den dynamiske tilstanden p_{bh}, samtidig som den maksiemere flyten (q). Resultatet endte med en kontroller som viste gode resultater med å kunne følge en referanse samtidig som den maksimerte flyten, uten noen feil som svingninger og overskriding.

Preface

This dissertation was a collaboration between my university NTNU and UFSC (Federal University of Santa Catarina). Therefore I had the luxury to live in Brazil the first few months of 2020. Unfortunately I had to leave Brazil because of the corona epidemic.

I would like to thank my supervisors Prof. Eduardo Camponogara, Eric Antonelo and Lars Imsland for giving me this opportunity. Your contribution, and quick responses to all my questions has been priceless.

I would also like to thank Jean Panaioti Jordanou for helping me with the code for developing the ESN-PNMPC, and sharing his previously work and allowing me to use his libraries.

Also a great thanks to Iver Osnes who was my collaborator for this dissertation and traveling partner in Brazil.

Finally a big thanks to all the friends I made in Brazil, I genuinely had a fantastic time in this beautiful country, and I have gotten some incredible memories I will adore for the rest of my life.

Contents

List of Figures

Acronyms

ANN Artificial neural networks.

CasADi Computer algebra system for automatic differentiation.

ESN Echo state network.

ESN-PNMPC Echo state network Practical nonlinear model predictive control.

ESP Electric submersible pump.

ML Machine learning.

MPC Model predictive control.

NLP Non-linear programming.

NMPC Nonlinear model predictive control.

NN Neural Network.

Oger OrGanic Environment for Reservoir computing.

PNMPC Practical nonlinear model predictive control.

QP Qudratic programming.

SI System identifaction.

Chapter 1

Introduction

1.1 Background and Motivation

The control of complex industrial processes where the models are initially unknown, can be a very difficult task. Such problems are widespread in the world and the control of these processes demands efficient data driven models. The data driven model of a unknown process is obtained by a system identification method.

There are many different ways to control an unknown process [1]. One control method that has been standard for multi-variable control, and which has been used in the oil and gas industry with good success [2] is Model Predictive Control (MPC). The MPC principle is to employ a prediction model, then solving a optimization problem over a time horizon at each sample time to find a control input.

The implementation of an MPC is challenging when the process is highly nonlinear, because the optimization that needs to be solved at each iteration is a nonlinear problem (NLP). To avoid the complexity involved with solving a nonlinear problem at each iteration, an approach called Practical Nonlinear Model Predictive Control (PNMPC) can be applied. PNMPC is based on the principle of spliting the process into two responses, a free response and a forced response. The free response contains all the nonlinearities and the forced response is linearized by a first order Taylor expansion. The PNMPC has achieved good results in control of diverse systems, particularly in gas and oil processes[3]. One drawback of using the PNMPC approach is that the gradients are very expensive to obtain, which are required to calculate the derivative terms involved in the method. This imposes a high computational cost.

This dissertation proposes to employ a data driven model NMPC approach for controlling an unknown nonlinear process, using the PNMPC method splitting the two responses with a trained Echo State Network (ESN) as the dynamic process based on [4]. ESN is a recurrent neural network which has proven to be effective for system identification and which relies on the principle of supervised learning. This approach has a big benefit when it comes to computational cost, since computational complexity of the derivative computation is reduced when using an Echo State Network as the predictive part of the PNMPC, making an ESN-PNMPC [4].

In this dissertation the control approach ESN-PNMPC will be implemented to operate an Electric Submersible Pump (ESP). ESP is one of the most widely used

methods for artificial lift in the oil industry [5].

1.2 Contributions

The main contributions of this dissertation are:

- A model of an ESP using CasADi.
- An ESN that predicts the behavior of the ESP with good accuracy.
- An application of ESN-PNMPC that controls the bottom-hole pressure p_{bh} to follow a reference while maximizing the flow (q).

1.3 Objectives

A short summary of the objectives follows below:

- Create a model for the ESP, and find the right parameters for the pump.
- Design an Echo State Network that models the ESP pump with high accuracy.
- Control the ESP using an ESN-PNMPC approach.

1.4 Outline

This document consist of six chapters, a short description of each chapter is described below.

Chapter 1 gives a short background and motivation for the project.

Chapter 2 provides information about the pump.

Chapter 3 presents relevant theory used kin the development of the dissertation.

Chapter 4 will describe the implementation for the experiments.

Chapter 5 shows the results.

Chapter 6 provides a conclusion and directions for future work.

Chapter 2

Problem statement

Controlling nonlinear processes can be a difficult task, mainly because such processes are hard to model correctly and the application of nonlinear predictive control (NMPC) imposes high computational costs. An alternative is to apply a controller called Practical Nonlinear Model Predictive Control (PNMPC) that overcomes some of the issues that arise when using the traditional NMPC. The PNMPC consists of two responses: a free response and a forced response, where the free response is the nonlinear system and the forced response is linearized. To lower the computational complexity of the derivative computation for the forced response, an echo state network is proposed as the predictive part of the PNMPC, giving rise to the ESN-PNMPC. In this dissertation the ESN-NMPC is put to the test in an Electric Submersible Pump (ESP). To create the ESN-NMPC three elements need to be fulfilled:

- Model of the ESP, that will act as the actual system.
- Create the ESN, with good results for predicting the ESPs behavior.
- Controlling the pump with the ESN-PNMPC, by finding the correct hyper parameters.

The problem statement for this dissertation is:

"How will an ESP perform when an ESN-PNMPC approach drives the pump to its desired references?"

A research model for this dissertation is shown in Figure 2.1, where the ESN-PNMPC finds the optimal input for reaching the reference for the bottom-hole pressure p_{bh} while maximizing the produced flow (q).

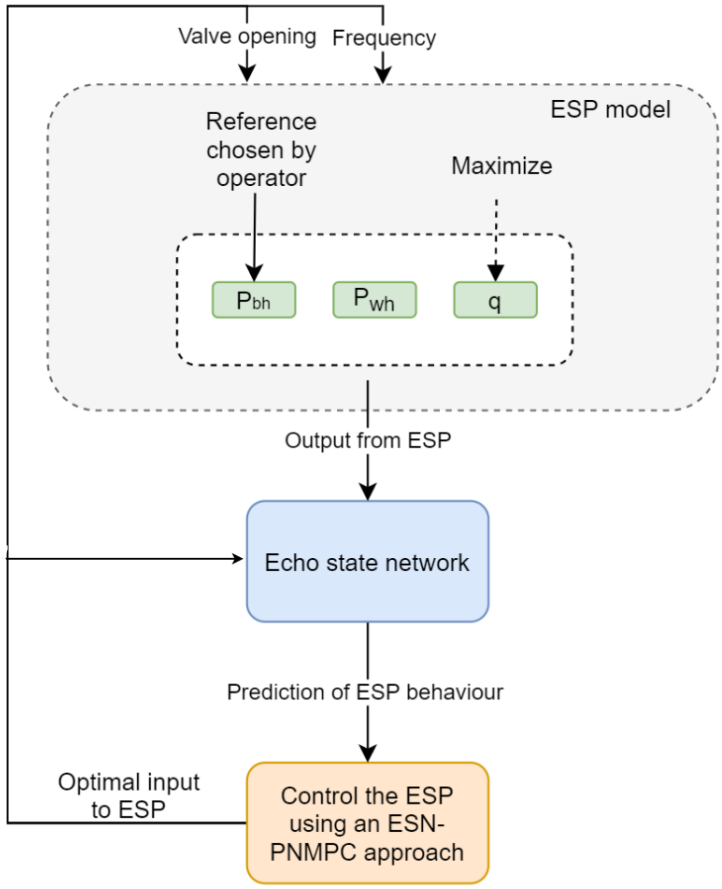


Figure 2.1: Research model

This chapter gives a description of the Electric Submersible Pump (ESP), address the issues regarding the synthesis of the echo state network and the ESN-PNMPC. It should be mentioned that section 2.1, Section 2.2.1, Section 2.3, Section 2.3.1 were written in a collaboration with Iver Osnes.

2.1 Oil wells and artificial lifting

In order to bring oil from a reservoir to the surface, enough pressure is essential. If a well has enough natural pressure to push fluid to the surface, we call it a flowing well. Flowing wells have a natural lift, which means that the pressure at the bottom of the well is strong enough to overcome the pressure loss through the pipeline on its way to the surface. However, most oil wells do not have enough pressure in their reservoir to rely on natural lift alone. Some wells might have a

natural lift in their early years of production, but the pressure will decrease over the lifetime. A well with insufficient pressure will leave valuable hydrocarbons deposited in the reservoir.

To overcome the problem of non-flowing wells, it is common practice to resort to artificial lifting. Artificial lifting is a method used to increase the pressure inside the well to boost oil production and to increase the lifetime of the well. There are mainly two different methods of artificial lifting: gas lift and pumping systems. The choice of method depends on multiple variables such as the volume of the well, depth, location (onshore or offshore) and the condition of the well [6]. A commonly used method for artificial lifting is the Electric Submersible Pump (ESP), which will be presented in the next section.

2.2 ESPs

An Electric Submersible Pump (ESP) is a multistage centrifugal pump installed several hundred meters under the sea surface in an oil well [7]. ESP will contribute to a boost in production and increase the recovery for a well. The pump inside the ESP works on a dynamic principle. Firstly, the kinetic energy of the liquid is increased, then, it is partly converted into pressure energy which will move the fluid through the pump [6].

ESP is primarily used in oil wells with high flow rates because of its high cost. They are therefore limited to high volume applications either offshore or onshore where the high cost can be justified.

ESP's have greater lifting power than most of other artificial lifting methods. According to [6], a set of advantages and disadvantages for the ESP areas follows.

Advantages:

- Suited for lifting high liquid volumes from medium depths.
- Efficient as long as the production is higher than 1000 bpd.
- Works well in deviated wells.
- Potentially low maintenance if properly designed and operated.
- Suited for offshore installations because of its low space requirements.

Disadvantages:

- Demand high electric power with high voltage.
- Low flexibility if it is run on a constant electrical frequency.

- Free gas at suction can harm the efficiency of the pump and even stop liquid production.
- Abrasive material as e.g. sand will increase the equipment wear.
- Expensive to purchase, repair and operate.
- High velocity will increase power usage and reduce productivity.

2.2.1 Lifespan

The lifespan of ESP's depends on multiple factors. The length of operation is an important factor, but would not cause a failure alone. ESPs do not normally wear out, it is often a sudden catastrophic event that causes the failure. Temperature, flow rate, vibration and power consumption can all affect the lifespan of an ESP. It is therefore normal to set constraints on these variables to increase the lifespan. A replacement of an ESP will cause a huge economic impact due to the cost of the replacement pump and the loss of production [8].

According with available statistics, 23% of all ESP failures are due to operator mistakes. When ESPs were first introduced, this number were as high as 80% [9]. A set of constraints has been introduced in later years to decrease the number of failures.

2.3 DAE modeling of wells with ESPs

The mathematical model of the system is based on a model developed by Statoil in [7], and additional equations from [10] are added to include viscosity. The system model consists of an ESP and a production choke valve. Perfect system knowledge is assumed for the model. The simulator is implemented in Python with CasADi as a tool for solving DAEs. This dynamic model works as a foundation for the development of further control and optimization strategies. A schematic picture of the model can be seen in Figure 2.2 and a description of the associated variables in Section 2.3.

The principles in this system is fairly simple. A mixture of liquid (oil, water and possibly gas) are flowing into the well from the reservoir (q_r). It will reach the ESP pump which will generate additional pressure and then raise the fluids to the production choke at the top of the well. An operator can control the ESP speed and production choke opening to reach a desired production or optimization target. The model assumes constant fluid properties to avoid an overly complex controller. Additional constraints are added to increase the lifespan of the ESP [7].

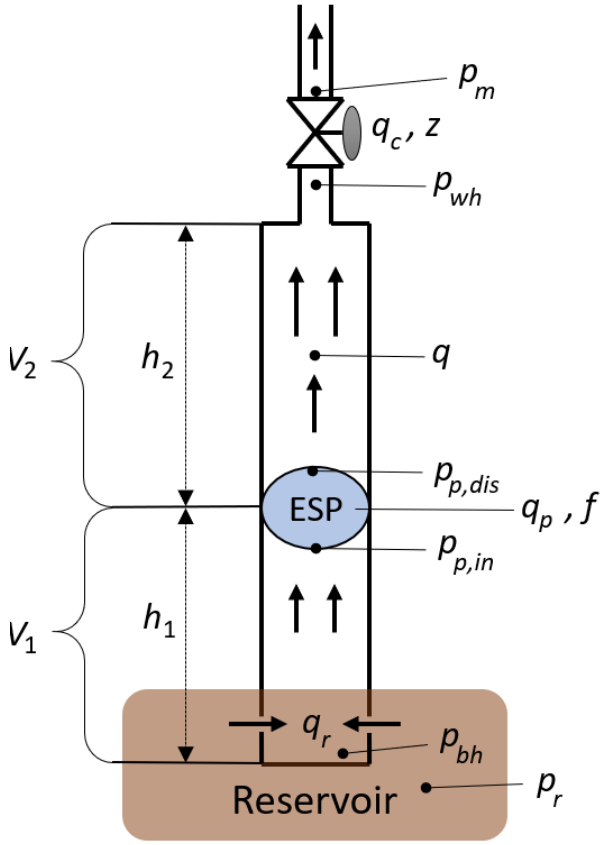


Figure 2.2: ESP lifted well, recreated from [10].

Control inputs	
F	ESP frequency
z	Choke valve opening

ESP data	
p_m	Production manifold pressure
p_{wh}	Wellhead pressure
p_{bh}	Bottomhole pressure
$p_{p,in}$	ESP intakepressure
$p_{p,dis}$	ESP discharge pressure
p_r	Reservoir pressure

Parameters from fluid analysis and well tests	
q	Average liquid flow rate
q_r	Flow rate from reservoir into the well
q_c	Flow rate through production choke

Table 2.1: Model variables

2.3.1 Model equations and parameters

The model of the ESP is divided into reservoir inflow, production pipe volumes, ESP and production choke. Despite excluding complexity such as effects due to gas and change of viscosity, the model will still represent the well dynamics quite accurately [7]. The system has three states: bottomhole pressure p_{bh}, wellhead pressure p_{wh} and average flow rate q. Their differential equations are as follows:

$$\dot{p}_{bh} = \frac{V_1}{\beta_1}(q_r - q) \tag{2.1a}$$

$$\dot{p}_{wh} = \frac{V_2}{\beta_2}(q - q_c) \tag{2.1b}$$

$$\dot{q} = \frac{1}{M}(p_{bh} - p_{wh} - \rho g h_w - \Delta p_f + \Delta P_p) \tag{2.1c}$$

where Δp_f are the pressure loss due to friction and ΔP_p are pressure loss due to the ESP dynamics. The differential equations come with a set of constraints, which can be described as the following algebraic equations:

Flow:

$$q_r = PI(p_r - p_{bh}) \tag{2.2a}$$
$$q_c = C_c\sqrt{p_{wh} - p_m}\ z \tag{2.2b}$$

Friction:

$$\Delta p_f = F_1 + F_2 \tag{2.3a}$$
$$F_i = 0.158\ \frac{\rho L_i q^2}{D_i A_i^2}\left(\frac{\mu}{\rho D_i q}\right)^{\frac{1}{4}} \tag{2.3b}$$

ESP:

$$\Delta p_p = \rho g H \tag{2.4a}$$
$$H = C_H(\mu)\left(c_0 + c_1\left(\frac{q}{C_Q(\mu)}\ \frac{f_0}{f}\right) - c_2\left(\frac{q}{C_Q(\mu)}\ \frac{f_0}{f}\right)^2\left(\frac{f}{f_0}\right)^2\right) \tag{2.4b}$$
$$c_0 = 9.5970 \cdot 10^2 \tag{2.4c}$$
$$c_1 = 7.4959 \cdot 10^3 \tag{2.4d}$$
$$c_2 = 1.2454 \cdot 10^6 \tag{2.4e}$$

The parameters used in this model are based on the parameters from [10]. These parameters can be found in Table A.1 in Appendix A, which includes the fixed parameters such as well dimensions and ESP parameters, and parameters found from analysis of fluid such as bulk modulus β_i and density ρ [10]. Parameters such as the well productivity index PI, viscosity μ and manifold pressure p_m are assumed constant in this dissertation. An illustration of how the different states behaves with a constant input given in the ESP can be seen in Figure 2.3.

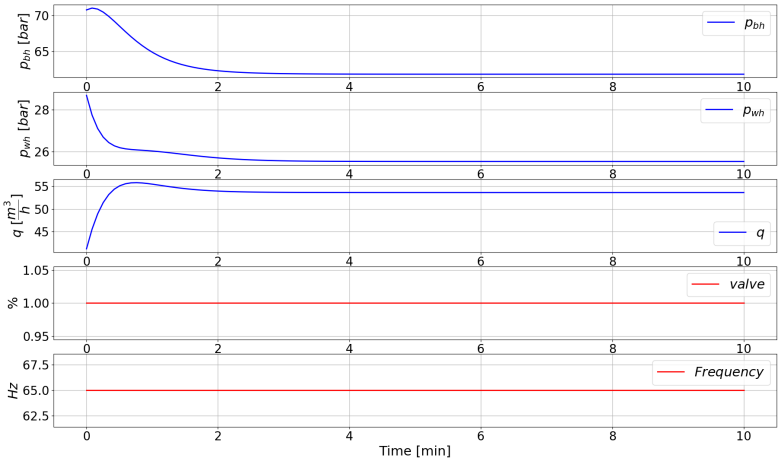


Figure 2.3: Illustration of the pumps behaviour

2.4 ESN of the pump

For this dissertation two networks were created, one using the Oger toolbox and one created from scratch. This is because it is possible to validate the second network using the results from the same data set and parameters, also because the Oger toolbox has a grid search program, making it easier to find the right parameters for the network.

To train the echo state network a data set is needed from the pump. It is important to find the correct sampling rate for the data set. The sampling rate has to be fast enough so that the system exhibits rich nonlinear dynamics, while having a sampling rate that is plausible for the sensors inside an ESP. Another important aspect is to create a random input to the ESP model that is adequate. Having multiple frequencies, it is instrumental that the data set is randomly generated in order to optimise the training of the ESN. To make the data set more realistic noise should be added to the data set.

When using an Echo State Network it is crucial to choose the right parameters to have a well functioning network. This can often be a difficult task or at least time consuming.

2.5 PNMPC of the pump

The purpose of this dissertation is to control the ESP using ESN-PNMPC. It is desired to control the state p_{bh} by adding a chosen reference point for this state, while maximizing the flow for the pump. The goal is to optimize the controller so that it reaches the desired reference with a smooth trajectory. The problems presented in this is the implementation, finding realistic constraints, finding right parameters to regulate the system and choosing reference points that are desired for the pump.

Chapter 3

Theory

This chapter presents relevant theory and some previously published work, starting with some basic system identification and general optimization problems. Later the framework of neural networks and Echo State Networks will be presented, then the controllers MPC, PNMPC and ESN-PNMPC. Section 3.1, Section 3.3, Section 3.5 are adapted from a previous project (TTK4551) written in the Spring Semester 2019 at NTNU.

3.1 System identification

System identification is an approach for creating a mathematical model of a dynamic system. This is done by finding a function that describes the input and output relationship. Finding this function makes it possible to predict future states with different inputs, which becomes useful for controlling the system later. Figure 3.1 below is an illustration of a relationship between the input and output in a dynamic model.

Figure 3.1: Illustration of the model with a mathematical representation of the relationship between input and output of a dynamic system.

There are different algorithms and tool boxes to use when creating the function for the system identification. To choose which method to use depends on how much information is available about the system. To classify the system there are three ways:

1. **Black Box** models only depend on the input and output data, since the dynamics of the system is not available. Artificial neural networks is a black box approach [11].
2. **Grey Box** models are also referred to as hybrid models. This is a model when some knowledge of the system is available and can be used in a

combination with black models. This means that the model is created with available data and theory about the system [11].

3. **White Box** models are based on the theory of the system. It is modeled based on physics equations, and are fully comprehensible [11].

3.2 DAE modeling and solution

Dynamic system simulation is used to examine system behavior over time. Most dynamic systems are described by differential equations or differential algebraic equations. The purpose of modeling and simulation is to imitate how a system will behave in the real world. Having the opportunity to simulate the system gives a big advantage since experiments on a real process might be both expensive and dangerous [12]. These simulations can be used to analyze the system behavior over time, and to predict system behavior with different input values.

In this dissertation differential algebraic equations (DAEs) will be used, which is a kind of equation with derivatives and an unknown function. The DAE can be seen as an ODE with additional algebraic constraints on the dynamic variable. The DAE standard form is given in (3.1) that follows below [13].

$$\dot{x} = f(x, y, t) \tag{3.1a}$$
$$0 = g(x, y, t) \tag{3.1b}$$

3.3 Optimization

Optimization means to find a set of feasible values that either maximize or minimize a cost function. Optimization is used in many fields from finance to engineering systems [14]. The standard formulation of an optimization problem is shown in (3.2) below [15].

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0,\ i = 1, \ldots, m \\ & h_j(x) = 0,\ j = 1, \ldots, p \end{aligned} \tag{3.2}$$

In (3.2) f is the objective function, the one that should be minimized, g holds the inequality constraints, and h holds the equality constraints.

An important aspect of optimization problems is convexity. If the problem is convex, it means that any local minimizer is also the global minimizer. The definition of convexity is stated below in (3.3) [16].

$$f[\lambda y + (1 - \lambda)z] \leq \lambda f(y) + (1 - \lambda) f(z) \tag{3.3a}$$

for all $y, z \in \operatorname{dom} f$ and $\lambda \in [0, 1]$. There are many different kinds of optimization problems, the most common are linear programming, quadratic programming and non linear programming, a short description is stated below.

1. **Linear programming** consists of the problems with the objective function and the constraints being linear. These problems are convex and are usually easy to solve with e.g. the simplex algorithm [15].
2. **Quadratic programming** is when the objective function is quadratic, but all the constraints are linear. This may or may not be easy to solve, depending if the objective is convex or not. Typical algorithms to solve quadratic programs are active set methods [15].
3. **Nonlinear programming** is where either the constraints or the objective function is nonlinear. Typical algorithms to solve these problems are sequential quadratic programming (SQP) algorithms [15].

3.4 Linear regression

Linear regression will find find the linear relationship between the input and a continuous output. Linear regression methods has been around since 1795, the method is called Ordinary least squares (OLS) and was developed by Carl Friedrich Gauss. Linear regression will find the optimal weights as long as some assumptions are given [17]. The equation for linear regression is given below in (3.4).

$$y = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \tag{3.4}$$

Where β_i are the coefficients and x is the independent variable.

For all regression methods, a common factor is that these methods become a optimization problem that minimizes the error between the true value and the estimated value through the regression model.

3.5 Introduction to Neural Networks

Artificial neural networks are a type of machine learning technique inspired by how the human brain works. This method is used when its difficult or impossible to express the issue in an algorithmic way. Fields that artificial neural networks are used include the recognition of handwritten digits, object identification, self-driving cars, banking and scores of others [18]. As this approach relies on a set of training data that is fed through the network, then the network can learn to recognize patterns.

The structure of a neural network is made with a set of neurons that are constructed layer by layer. A neuron is a mathematically function that sums up values from the previous layer and adds a bias. The information flow of each neuron is described in Figure 3.2.

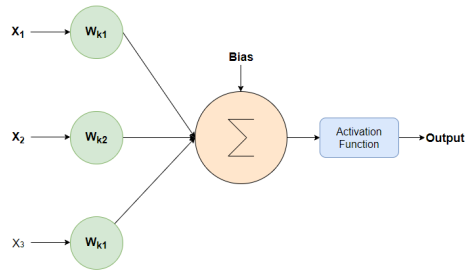


Figure 3.2: Figure showing information flow for a neuron

Each neuron sums up the input data multiplied with its unique weight plus some bias. The equation for this is shown in (3.5).

$$\sum_{i=1}^{n} x_i w_i + b \tag{3.5}$$

where n is the number of inputs to the neuron, $x = (x_1, \ldots, x_n)$ is the vector with the input values, w is the vector with weights, and b is the bias.

Before the neuron sends its output value to the next neuron, the value goes through an activation function. There are many different activation functions to use and the most common ones are Sigmoid, tanh and ReLU. The purpose of the activation function is to make the output of each neuron nonlinear and to scale the output. A schematic figure of deep neural networks is shown in Figure 3.3.

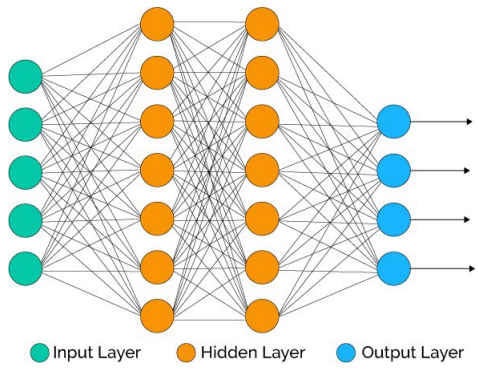


Figure 3.3: Diagram of a deep neural network [19]

The training process learns by the mistakes and uses a method called backprop-

agation to change the weights and biases in the network. Backpropagation is an algorithmic way of computation of derivatives back to the input layer. This method is based on what happens to the objective function if some of the weights and biases are changed. The objective of the artificial neural network is to minimize the error between the predicted output and the actual output. The objective function is shown in (3.6), when using mean square error.

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \left(p_i - \widehat{p}_i(w,b)\right)^2 \tag{3.6}$$

where p_i is the correct value and $\widehat{p}_i(w,b)$ is the predicted value produced by the neural network, for each example i, as a function of the network parameters (w,b). To minimize this function the problem becomes a nonlinear optimization problem. It is not possible to calculate all the weights and biases optimally, the problem is simply too large and complex for practical applications.

There are many classes that build on the artificial neural network method. One of these classes are recurrent neural networks (RNNs). This is a class of artificial neural networks that have a memory. Recurrent neural networks contains a loop, which makes the network to have a sequential memory, meaning that it uses past information to effect the future information.

3.6 Echo State Network (ESN)

Echo state network (ESN) is a type of recurrent neural network based on the principle of supervised learning, which has been used in fields like system identification [20]. ESNs have a memory as they are RNNs. This approach is used to create a model of a dynamic system dependent on its past behavior. ESNs can learn the dynamics of a system to a relatively low computational cost. In this project ESN is used as a black box approach to model the dynamic system.

3.6.1 Structure

The echo state network consists of an input layer, a hidden layer, and an output layer. The input layer is expressed by inputs of the data sets the network is learning on; the input layer is connected to the hidden layer with weights. The hidden layer is also known as the reservoir; this layer can be very sparsely connected. The number of nodes in the hidden layer is much larger that the number of network inputs. This layer can contain as many nodes as needed to replicate the desired system behavior, but at the expense of computation time. The output layer consists of the estimated value for the given input. Figure 3.4 shows a schematic of how the ESN looks like [21].

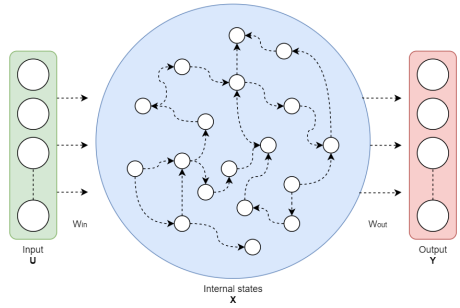


Figure 3.4: Structure of an echo state network

The output weights W_{out} are the ones that are trainable, which are changed so that the network can imitate specific patterns.

The notation used for the ESN in Figure 3.4 is described below:

- W_{in} are the weights between the input and the reservoir.
- X is the internal states given by a vector x.
- W is the weights between the internal states.

- W_{out} are the weights connected to the output nodes.

3.6.2 Workflow

In this section the workflow of the ESN will be described [21].

- Initialize all weights at random, except W_{out} and choose appropriate network parameters. These parameters are explained in Section 3.6.4.
- Run the network using the training input data $u(n)$ and gather the corresponding reservoir activation states.
- Calculate the linear output weights (W_{out}) using linear regression, then linear regression solves the minimization between the actual output and the target output.

3.6.3 Equations

The equations that describe how the Figure 3.4 works are shown in (3.7a). These equations shows how the network updates the internal states X and the output [21].

$$x[k] = (1-\alpha)x[k-1] + \alpha f(\mathbf{W_{in}}[k] + \mathbf{W}x[k-1] + \mathbf{W_b} + \mathbf{W_{fb}}y[k-1]) \quad (3.7a)$$

$$y[k] = \mathbf{W_{out}}x[k] \quad (3.7b)$$

where α is the leak rate described in Section 3.6.4, $x[k]$ is the reservoir state at time k, $\mathbf{W_{in}}$ is the matrix with weights from the input nodes to the reservoir, $\mathbf{W}$ is the matrix with weights between reservoir units, $\mathbf{W_b}$ is the bias, and $\mathbf{W_{fb}}$ is the weight matrix that introduces feedback from the output y, the feedback is not implemented in this dissertation. The function f is the activation function for the neuron. There are many different activation functions to use, and the most common ones are sigmoid, tanh and arctan. Figure 3.5 shows the activation functions which are commonly used. The activation functions forces the output value to lie between -1 and 1.

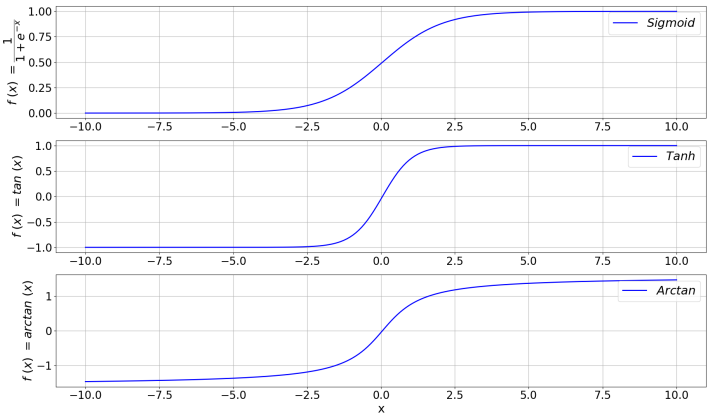


Figure 3.5: Different activation functions

The training of the Echo State Network becomes an optimization problem. The concept is about minimizing the error between the predicted output and the actual output. The weight matrices $\mathbf{W_{in}}$, $\mathbf{W}$ and $\mathbf{W_{fb}}$ are set arbitrarily according with an uniform distribution. The weights that are trained in an echo state network appear in the matrix $\mathbf{W_{out}}$. Echo state networks follow the principle of supervised learning, meaning that training data is needed. A training set consist of inputs ($\mathbf{U}$) and with their respectively outputs ($\mathbf{Y}$). With the training set, the network then iterates through (3.7a). Given the inputs, the estimated states ($\mathbf{X}$) and outputs $\widehat{\mathbf{Y}}$ are collected. Then the learning task reduces to minimize the error between the true value for the output and the predicted output [21]. The training is done by using linear regression.

3.6.4 Parameters

For creating an echo state network it is crucial to choose the right value for each parameter. Below is a description of the most important parameters and how they influence the network.

Size of the reservoir

The number of neurons in the reservoir plays an important role on how the network is performing. Usually the more neurons in the reservoir, the better the performance of the network will be. However, despite the computations of an ESN being relatively cheap, it is not uncommon to see big reservoir [22]. When the reservoir is large it becomes easier to find a linear combination to minimize

the error between the predicted value and the real value.

Leak rate

To choose the right leak rate depends on the dynamics of the system. The leak rate value regulates the speed of the dynamics in the internal states X. This means that a low leak rate value will slow down the dynamics and increase the memory of the network. In (3.7a) the leak rate α is defined as a number between $(0, 1)$.

Spectral Radius

Spectral radius is the scaling factor, the maximum absolute eigenvalue of W for the reservoir weights, which effects the dynamical behavior. The spectral radius is between 0 and 1. The value should be higher when the memory of the network needs to be longer and the task is more nonlinear.

Input scaling

Input scaling regulates the influence of the input weights to the network. Meaning that the input scaling regulates how nonlinear the hidden layer is. Choosing the input scaling value depends on how linear the task is, for a linear system the input scaling should be low.

Warm-up

The initial states of the echo state network is usually zero. Therefore it is desired to neglect the first iterations, since in the first iterations contain no history to the system. It is therefore desired to ignore the first iterations until the system has acquired enough history and washed out the initial conditions.

3.7 Model Predictive Control (MPC)

This section gives an introduction to the model predictive control, followed by a section describing the practical nonlinear model predictive control that is used in this dissertation.

A model predictive controller is an advanced method for controlling a process. This controller predicts the future outputs based on the change of the dependent variables caused by the manipulated variables, and chooses the optimal control input for each time-step, while satisfying a set of constraints. These kinds of controllers have been applied in oil refineries since the 1980s [23]. It is important to have a controller with the property of satisfying constraints, since many processes have limited inputs and if states go over their limitations it can be harmful for the system. Figure 3.6 illustrates the scheme of how an MPC controller works.

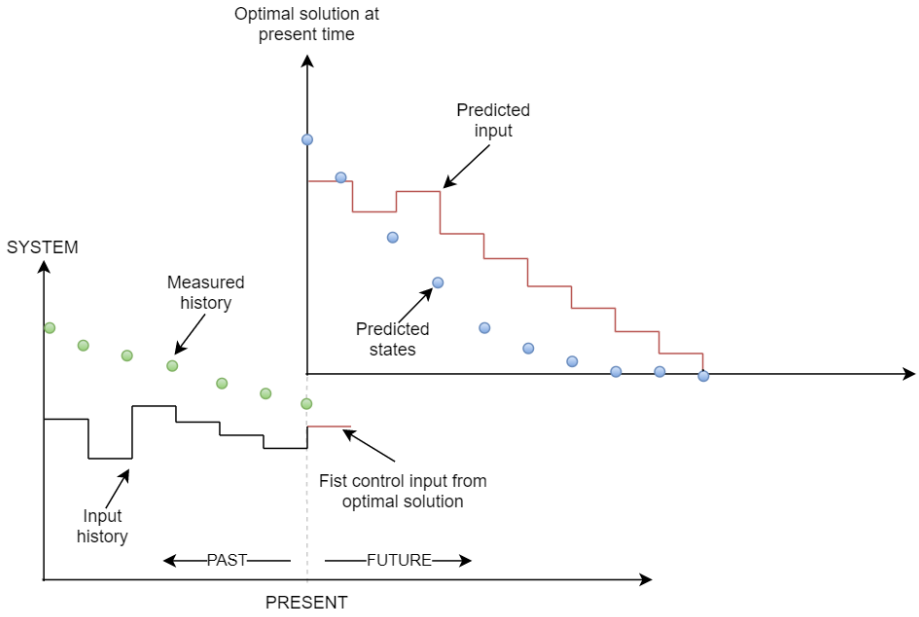


Figure 3.6: Scheme of how the MPC works

Figure 3.6 explains the workflow of the MPC. The y-axis shows where the current state of the process is at, to the left of the y-axis are the previous states and inputs, and to the right of the y-axis are the predicted values. At the present time a cost minimizing control strategy is calculated over a finite horizon, which is shown right of the y-axis. Then the current control input is implemented, and a new control strategy is computed at the next time step.

At each time step of the MPC an optimization problem is solved, which is usually a QP problem. The formulation is described in (3.8a), in which (3.8a) is the objective function, and (3.8b) and (3.8c) are the constraints.

$$\underset{x}{\text{minimize}}\ f(x) = \frac{1}{2}x^T\mathbf{G}x + x^T c \tag{3.8a}$$

$$\text{subject to}: a_i^T x = b_i,\ i \in \mathcal{E} \tag{3.8b}$$

$$a_i^T x \geq b_i,\ i \in \mathcal{I} \tag{3.8c}$$

In (3.8a), G is a symmetric positive-definite matrix, $\mathcal{E}$ and $\mathcal{I}$ represents sets of indices for constraints, and c and x are vectors.

3.8 Practical Nonlinear Model Predictive Control (PNMPC)

This controller works on the same principle as the MPC, only now the controller is based on separating the nonlinear dynamic model into two responses, a free response and a forced response. This is done by a first order Taylor expansion. A Taylor series is used to estimate how a function looks like in the current time-step. In (3.9) below the definition of the Taylor series is given [24]:

$$f(x) = f(a) + \frac{f^{'}(a)}{1!}(x-a) + \frac{f^{''}(a)}{2!}(x-a)^2 + \frac{f^{''}(a)}{3!}(x-a)^3 + \cdots \tag{3.9}$$

for a function $f : \Re \to \Re$ about a point a.

The advantage of separating the model into two responses is the computational advantage, since the problem now per iteration is a quadratic program (QP), as explained in Section 3.3. Below follows a compilation of the equations used in the PNMPC, first assuming the system on the form:

$$\mathbf{x}[k+i] = \mathbf{f}(x[k+1-1], \mathbf{u}[k+i-1]) \tag{3.10a}$$

$$\mathbf{y}[k+i] = \mathbf{g}(x[k+i]) \tag{3.10b}$$

$$\mathbf{u}[k+i-1] = \mathbf{u}[k-1] + \sum_{j=0}^{i-1} \Delta\mathbf{u}[k+j] \tag{3.10c}$$

Below is the equations for the PNMPC and a explanation of each parameter [25].

$$\widehat{\mathbf{Y}} = \mathbf{G} \cdot \Delta\mathbf{U} + \mathbf{F} \tag{3.11}$$

where $\widehat{\mathbf{Y}}$ is a vector with the predictions of the output, $\mathbf{F}$ is the free response, and $\mathbf{G} \cdot \Delta\mathbf{U}$ is the forced response over the prediction horizon (N_y).

Notice that the Jacobian of the state equation for the free response is given by

$$\mathbf{G} = \begin{bmatrix} \frac{\delta y[k+1]}{\delta u[k]} & 0 & \dots & 0 \\ \frac{\delta y[k+2]}{\delta u[k]} & \frac{\delta y[k+2]}{\delta u[k+1]} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta y[k+N_y]}{\delta u[k]} & \frac{\delta y[k+N_y]}{\delta u[k+1]} & \dots & \frac{\delta y[k+N_y]}{\delta u[k+N_u-1]} \end{bmatrix} \tag{3.12}$$

where all the derivatives being calculated inside the $\mathbf{G}$ matrix are carried out with respect to $\Delta\mathbf{u}[k+1] = 0$ and u is the manipulated variable. Thus each line in the G matrix is the linearization with first order Taylor series at a certain time. N_y is the prediction horizon and N_u is the control horizon.

The control increment $\Delta\mathbf{U}$ is given by

$$\Delta\mathbf{U} = \begin{bmatrix} \Delta\mathbf{u}[k] \\ \Delta\mathbf{u}[k+1] \\ \vdots \\ \Delta\mathbf{u}[k+N_u-1] \end{bmatrix} \tag{3.13}$$

which contains each control increment used in the calculation of the predictions. The free response $\mathbf{F}$ is given by

$$\mathbf{F} = \begin{bmatrix} \mathbf{g}(\mathbf{f}(x[k], u[k-1])) \\ \mathbf{g}(\mathbf{f}(x[k+1], u[k-1])) \\ \vdots \\ \mathbf{g}(\mathbf{f}(x[k+N_y-1], u[k-1])) \end{bmatrix} \tag{3.14}$$

(3.12) is derived from the first-order Taylor series, as mentioned at the start of this chapter. The difference between the free response and the forced response is that the free response contains all the non linearities and the forced response is linerarized, making the control increment calculated through a quadratic program.

When multiple variables are implemented the model suffers from a high computational cost. Because of this, it could be advantageous to implement the ESN into the PNMPC. This enables faster computation of linearized models. This method is explained in Section 3.9.

3.9 ESN-PNMPC

The echo state network will reduce the computation time because the analytical calculation of derivatives, since the solution becomes a QP problem. Figure 3.7 shows a block diagram of how the ESN integrates with the PNMPC.

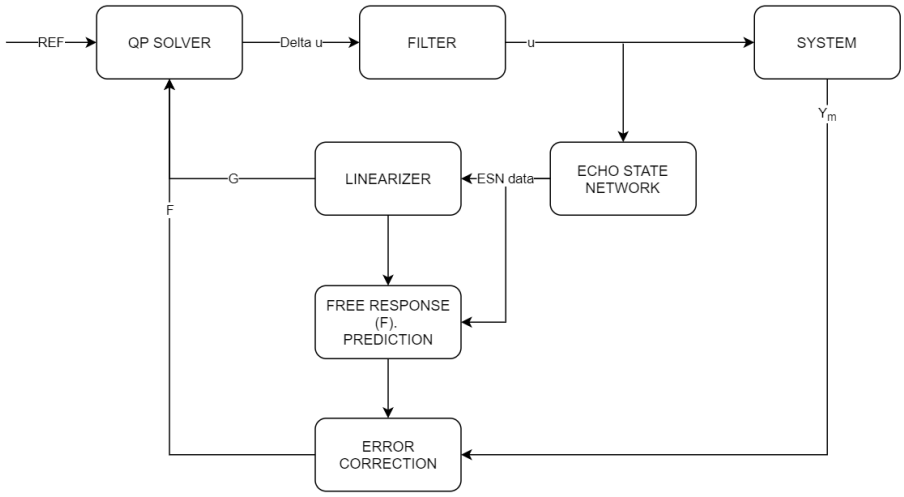


Figure 3.7: Block diagram of how the ESN with the PNMPC works [26]

The block scheme gives a representation of the ESN-PNMPC. In this scheme the ESN block is a trained network used to imitate the dynamic system. Then the ESN is used to linearize the system and to compute the forced response ($\mathbf{G}$), and making the free response prediction block. Then, from the free response block, the error correction block is an integrated filter that computes the correction factor. Then all the models are sent to the quadratic programming block that solves the resulting optimization problem.

Below is an explanation of each block of Figure 3.7. Starting with the linearizer, the error correction and finally the QP problem.

Linearizer

The calculations of the derivatives of the output of the system with respect to the input is done by the chain rule, as follows:

$$\frac{\delta y[k+i]}{\delta \Delta u[k+j]} = \frac{\delta g}{\delta x[k+i]} \frac{\delta x[k+i]}{\delta \Delta u[k+j]} \tag{3.15a}$$

$$\frac{\delta x[k+i]}{\delta \Delta u[k+j]} = \frac{\delta f}{\delta \Delta u[k+j]} + \frac{\delta f}{\delta x[k+i-1]} \frac{\delta x[k+i-1]}{\delta \Delta u[k+j]} \tag{3.15b}$$

The difficulties with this is that the **G** matrix is built recursively by forward propagation. The dynamic matrix is evaluated at $\Delta U = 0$, which makes all the derivatives possible to be evaluated with respect to the control input $u[k-1]$, meaning that:

$$\frac{\delta f(x[k+i])}{\delta \Delta u}\Big]_{\Delta u=0} = \frac{\delta f(x[k+i])}{\delta u}\Big]_{u=u[k-1]} \tag{3.16}$$

When i is bigger than j the control increment ΔU has the same influence on the output as $\Delta U(k+i)$, this is because the input signal was the input in a previous instant. This leads to the notation of $J(i) = \frac{\delta f(x[k+i]}{\delta x[k+i]}$ and to simplify $S(i) = \frac{\delta f(x[k+i])}{\delta x[k+i]}$. With this it is possible to compute the derivatives in a recursive form:

$$G_{ij} = \frac{\delta g}{\delta x[k+i]} \frac{\delta x[k+i]}{\delta \Delta u[k+j]} \tag{3.17}$$

where:

$$\frac{\delta x[k+i]}{\delta \Delta u[k+j]} = \begin{cases} J(i-1) + S(i-1)\dfrac{\delta x[k+i-1]}{\delta \Delta u[k+j]} & i > j \\ J(i-1) & i = j \\ 0 & i < j \end{cases} \tag{3.18}$$

The echo state network works as the prediction model for the PNMPC and is trained offline. The model derivatives are defined from [27, 28] and shown in (3.19a).

$$\frac{\delta g}{\delta x} = W_r^o \tag{3.19a}$$

$$J(i) = \frac{\delta f}{\delta z_i} W_i^r \tag{3.19b}$$

$$S(i) = (1-\alpha)I + \alpha \frac{\delta f}{\delta z_i}(W_r^r + W_o^r + W_r^o) \tag{3.19c}$$

$$z_i = W_r^r \; a[k+i] + w_i^r u[k-1] + W_o^r W_r^o a[k+1] + W_b^r \tag{3.19d}$$

where a is the network state, and x is the model state. A short simplified summary is that the ESN is used to calculate the free response predictions, and the Taylor approximation.

Error correction

To deal with disturbances and errors from the model, a low pass filter is implemented on the error between the output and the prediction [29]. The purpose of the filter is to slow down the error from the controller, which makes the system

more robust but looses response speed [30]. With the introduction of a low pass filter the equation of the free response, as given in (3.20a), becomes:

$$\mathbf{F} = \begin{bmatrix} \mathbf{g}(\mathbf{f}(x[k], u[k-1])) \\ \mathbf{g}(\mathbf{f}(x[k+1], u[k-1])) \\ \vdots \\ \mathbf{g}(\mathbf{f}(x[k+N_y-1], u[k-1])) \end{bmatrix} + n[k] \tag{3.20a}$$

$$\tag{3.20b}$$

$$\Delta n[k] = (1-w)(\widehat{y}[k|k-1] - y_m[k]) + w\Delta n[k-1] \tag{3.20c}$$

$$n[k] = n[k-1] + K\Delta n[k] \tag{3.20d}$$

$$\widehat{y}[k|k-1] = g(f(x[k-1], u[k-1])) + n[k-1] \tag{3.20e}$$

To tune the filter parameters, we use the characteristic polynomial which is of the form:

$$K = \frac{a^2 - 2a + 1}{1 - a^2} \tag{3.21}$$

With this polynomial it is possible to choose the desired pole for the error correction dynamics. As seen in (3.21) the filter vanishes when $a = 0$.

Quadratic Programming (QP)

In this block the optimization of the QP problem will be solved. When the error is used in the cost function, the equation becomes:

$$J = (\mathbf{Y}_{ref} - \widehat{\mathbf{Y}})^T \mathbf{Q} (\mathbf{Y}_{ref} - \widehat{\mathbf{Y}}) + \Delta\mathbf{U}^T \mathbf{R} \Delta\mathbf{U} \tag{3.22}$$

where $\mathbf{Y}_{ref}$ is the reference, $\mathbf{Q}$ and $\mathbf{R}$ are the weighting matrices which are diagonal. These weighting matrices assert the variables importance when solving (3.22). This is a normal cost function for the NMPC. The structure of the free response is on a vectorized form of the prediction, which is related to approaches like DMC and GPC [30]. Making the cost function:

$$J = \Delta\mathbf{U}^T \ \mathbf{H} \ \Delta\mathbf{U} + \mathbf{c}^T \ \Delta\mathbf{U} \tag{3.23a}$$

$$\mathbf{H} = \mathbf{G}^T \ \mathbf{Q} \ \mathbf{G} + \mathbf{R} \tag{3.23b}$$

$$\mathbf{c} = \mathbf{G}^T \ \mathbf{Q}^T \ (\mathbf{Y}_{ref} - \mathbf{F}) \tag{3.23c}$$

with the constraints:

$$\mathbf{U}_{min} \leq \mathbf{U} \leq \mathbf{U}_{max} \tag{3.24a}$$

$$\mathbf{Y}_{min} \leq \mathbf{Y} \leq \mathbf{Y}_{max} \tag{3.24b}$$

$$\Delta\mathbf{U}_{min} \leq \Delta\mathbf{U} \leq \Delta\mathbf{U}_{max} \tag{3.24c}$$

where $\mathbf{U}_{max}$ and $\mathbf{U}_{min}$ are the maximum and minimum input values, $\mathbf{Y}_{max}$ and $\mathbf{Y}_{min}$ are the maximum and minimum output values, and $\Delta\mathbf{U}_{max}$ and $\Delta\mathbf{U}_{min}$ define the maximum and minimum input change.

3.10 Error metrics

To measure how the well the echo state network and the practical nonlinear model predictive control are performing, different validations were performed. The validation was achieved by measuring the error between the true value and the predicted value. This was always performed on the test set. This section will describe different error metrics [26].

1. **Mean square error (MSE)** is the error between the ESN and the true value from the process. It is the sum of the error between each measure point in the test set squared. Where Y_{true} is the correct value from the model and Y_{pred} is the predicted value. Mathematically, MSE is defined as

$$MSE = \sum_{i=0}^{i_{testset}} (Y_{true}[i] - Y_{pred}[i])^2 \tag{3.25}$$

2. **Root mean square error (RMSE)** is the same as MSE only now the total error is rooted. Mathematically, RMSE is defined as

$$RMSE = \sqrt{\sum_{i=0}^{i_{testset}} (Y_{true}[i] - Y_{pred}[i])^2} \tag{3.26}$$

3. **Relative prior error** is the error with the ESN given in percentage. The relative error is given in mathematical form as follows

$$error_{\%} = 100\ (\frac{Y_{true} - Y_{pred}}{Y_{true}}) \tag{3.27}$$

4. **Integral of absolute error (IAE)** for the prediction on the ESN. This metric is given by

$$IAE = \sum_{i=0}^{i_{testset}} (Y_{true}[i] - Y_{pred}[i]) \tag{3.28}$$

5. **IAE** performed on the tracking on the controller, comparing the deviation on the plant and the referrence. Mathematically, IAE is defined as follows

$$IAE = \sum_{i=0}^{i_{testset}} (Y_{true}[i] - Y_{ref}[i]) \tag{3.29}$$

Chapter 4

Implementation

In this chapter, the implementation will be explained along with the parameters used to get the results. The chapter starts by explaining how the ESP model was built, then how the Echo State Network was created, and finally the ESN-PNMPC implementation.

4.1 ESP CasADi

In this project, CasADi (Computer algebra system for Automatic Differentiation) was used to solve DAEs and simulate the system model of the ESP. CasADi is an open-source library used in Matlab, Python, or C++. CasADi is used for nonlinear optimization, algorithmic differentiation, and used to solve differential-algebraic equations employing explicit or implicit Runge-Kutta methods. To solve ordinary differential equations CasADi uses the solver CVODED, and for differential-algebraic equations, CasADi uses the IDAS solver. The motivation for using CasADi in this project is because of its fast computations and its simplicity [31]. In CasADI the class DaeBuilder was used. This is a class that intends to help the modeling of complex dynamical systems and used later for optimal control problems. In this dissertation Python version (3.7) was used along with the CasADi version 3.5.1.

When creating the model with CasADi, there are three parameters to be defined; the initial value for the states in the differential equations, input values to the system, and the time grid, which reflects on how far and how many steps the solver is going to integrate.

The initial value to the differential equations are shown below and are chosen in this manner because it is not far from a steady-state when the system runs on the maximum input values.

- $P_{bh0} = 70$ bar;
- $P_{wh0} = 30$ bar;
- $q_0 = 36$ m^3/h.

In this case, the time grid reflects the number of data points the system will be sampled per minute. The correct sampling rate is key to capture the system

dynamics while having a sampling rate that is plausible for the sensors inside an ESP. Figure 4.1 gives a plot comparing the sampling rate between 12 samples per minute (indicated with the color blue) and 6 samples minute (indicated with the color red). The data sets collected for the training set were 12 samples per minute. Since then the system is significantly nonlinear and the dynamics for the system are better represented.

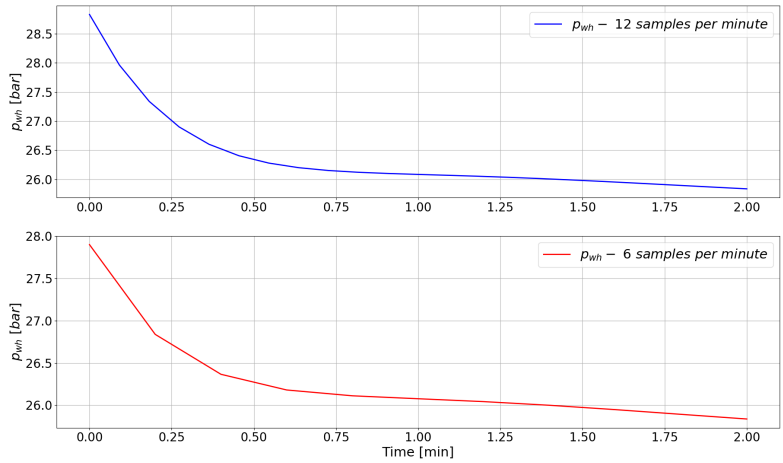


Figure 4.1: Sampling rate 12(blue) vs 6(red)

After some experiments the most optimal sampling rate for this dissertation was 12 samples per minute.

4.2 Echo State Network

To create the Echo State Network the Python programming language was used. One network was created using the Oger toolbox in Python version 2. Oger is a abbreviation for OrGanic Environment for Reservoir computing and is mainly used for reservoir computing networks [32]. The second one was made from scratch using Numpy in python version 3. Numpy is a library for adding and calculating multidimensional arrays and matrices [33].

The reason for using the Oger toolbox is twofold. One reason is to validate the network that was built from scratch, using the same parameters and compare the results. The second reason is that the toolbox has a grid search program, making it less complicated to find the suitable parameters to design an optimal network.

As explained above in Section 3.6.4 the choice of hyper-parameters for the ESN is important to design an optimal network. A good way of finding these parameters is by doing a grid search for the different parameters. The grid search runs the network with different values for the hyper-parameters and compares the results, in order to find the best hyper-parameters that gives the lowest generalization error. Figure 4.2 shows the results of a search performed on the leak rate.

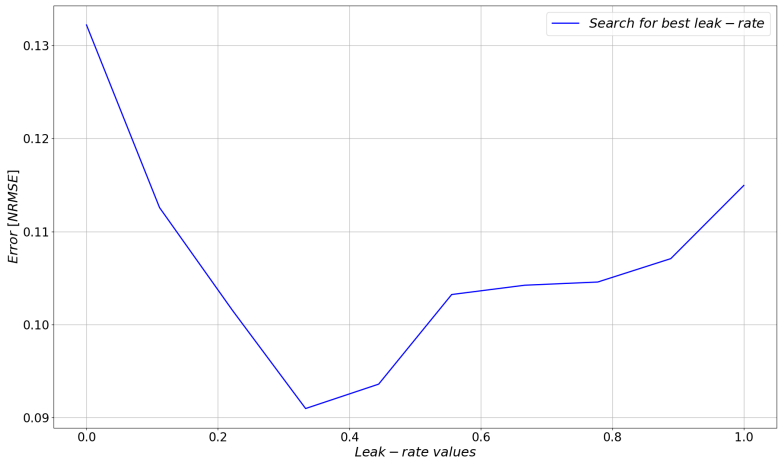


Figure 4.2: Search for the best leak-rate.

As seen in the plot above it is clear that the lowest error is around a leak rate of 0.3. Another search is then performed in more detail around the area with the lowest error. This is shown in Figure 4.3.

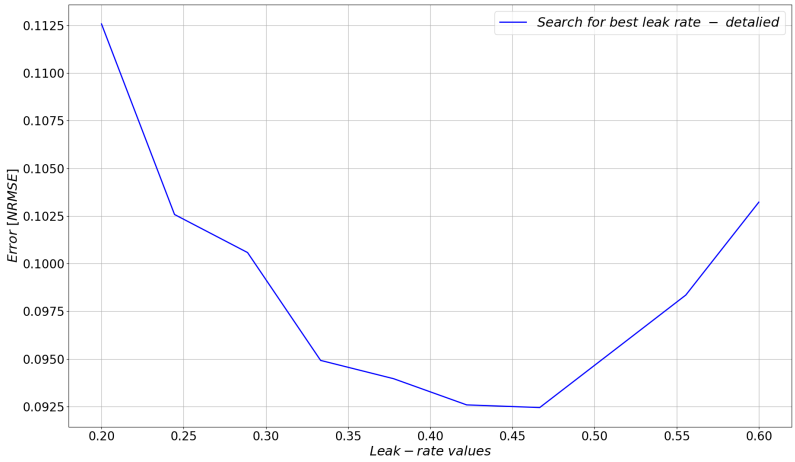


Figure 4.3: Search for the best leak-rate in more detail, within a small region of interest

Different training sets

Data sets were collected from the ESP model to train the Echo State Network. When controlling the system, one of the goals is to reach a reference value and stay on this value. Often a problem that occurs is a deviation from the values predicted by the ESN and the plant itself when reaching a steady-state. This happens when the training set has a too frequent input change. In order for the ESN to learn both fast and slow dynamics, data sets are created considering different dynamics for the input signal. This makes it feasible for the ESN to reach a steady-state with a low error with respect to the plant, as well as it can track rapid changes. For each data set, a larger training set and a smaller test set were created. The data sets that were created to validate and train the ESN are described below:

1. The inputs were constant and the system arrived at a steady-state.
2. Two data sets with random inputs. The Random Frequency Random Amplitude Signal (RFRAS) was used to create the random inputs. This is done with a NumPy random function that gives a uniform distribution. The equation for the random signal is given in (4.1).

$$val = min + (max - min) \cdot random\ value \tag{4.1}$$

(4.1) is put inside a for loop, making it easy to change how often the value

is going to change. The parameters chosen for (4.1) are shown in the table below and are suggested in [10]:

	Valve	**Frequency**
Min	0.1	35
Max	1	65

Where min reflects the minimum value the inputs, and max is the maximum value for the inputs. One data set is created with a slower frequency of input change and another with faster input change. Figures 4.4 and 4.5 show two plots with the two training sets produced with the RFRAS equation.

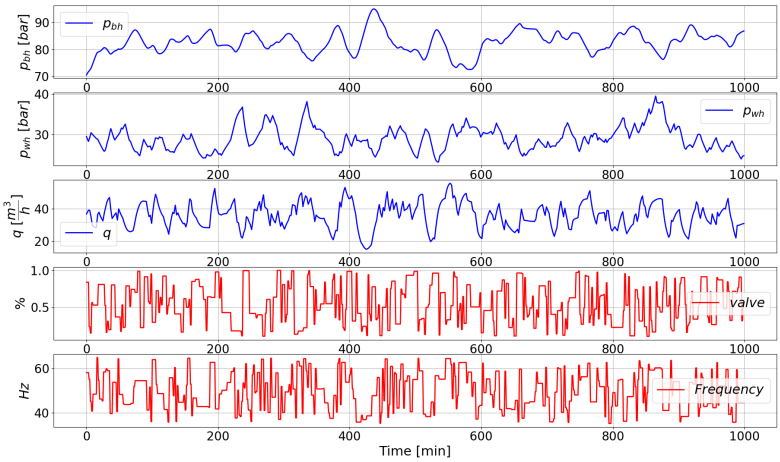


Figure 4.4: RFRAS with fast change

Figure 4.4 is a plot showing 1000 of the data points of the training set. In this plot the input value is changing fast, meaning that the value changes every 10 data samples.

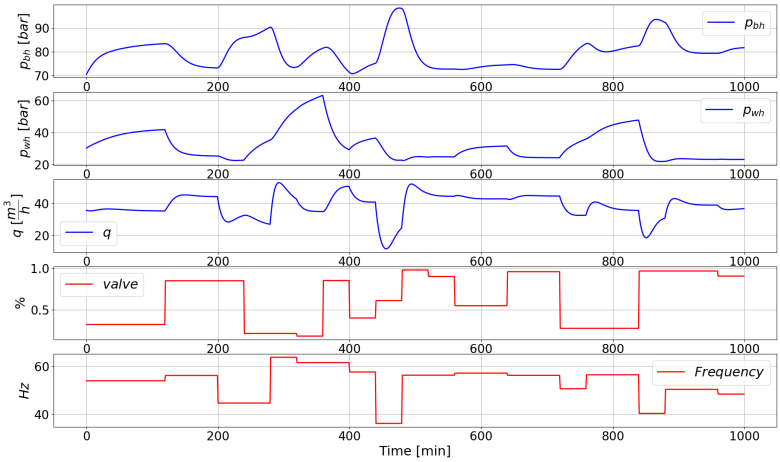


Figure 4.5: RFRAS with slow change

Figure 4.5 is a plot showing 1000 of the data points of the training set. In this plot, the update for the input is slow, and the system reaches a steady-state for each input, the input is the same for 40-time steps before changing.

3. One data set that combines the slow and the fast frequency of input change as shown above. The data set contains 50 % with fast change and the remaining 50 % with slower change, making the system reach a steady-state for each input and also react fast to quickly-changing inputs. A plot of this training set is shown in Figure 4.6.

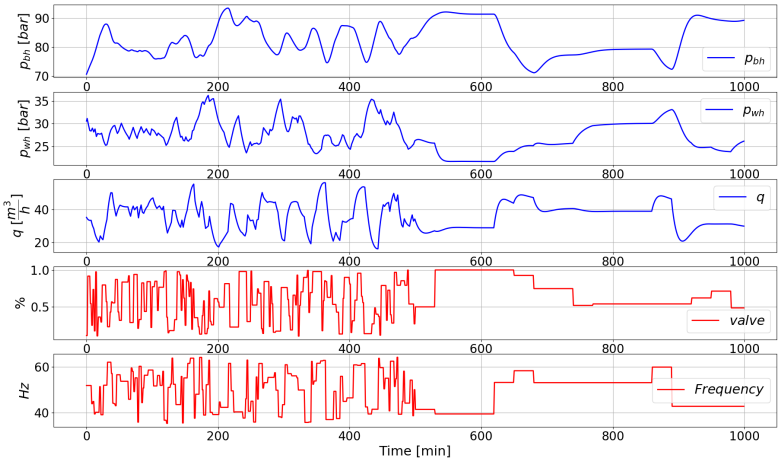


Figure 4.6: RFRAS combining the two sets above

As seen above, Figure 4.6 shows the plot combining fast input change and slow input change. The plot is only showing 1000 data points of the training set.

4. Noise is added to the training part of the set, while the test set is free of noise. The noise is added to the output, since it is more realistic that some noise is on the state sensors. This is done by adding a random signal of ± 10 % of the maximum value on each data sample in the training set. The training set is shown in Figure 4.7.

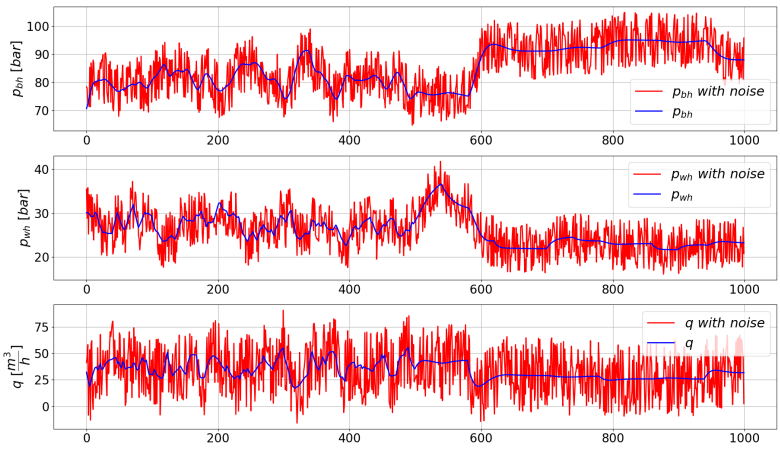


Figure 4.7: Training set with noise

Each set that was created containing 12300 data points, where the training set contains 12000 data points and the test set has 300 points.

Number of neurons

The choice of the number of neurons for the network consists in finding the smallest number, since it influences computational time, while minimizing the errors from the network predictions. Figure 4.8 shows a bar plot with computation time to run the ESN as a function of the number of neurons.

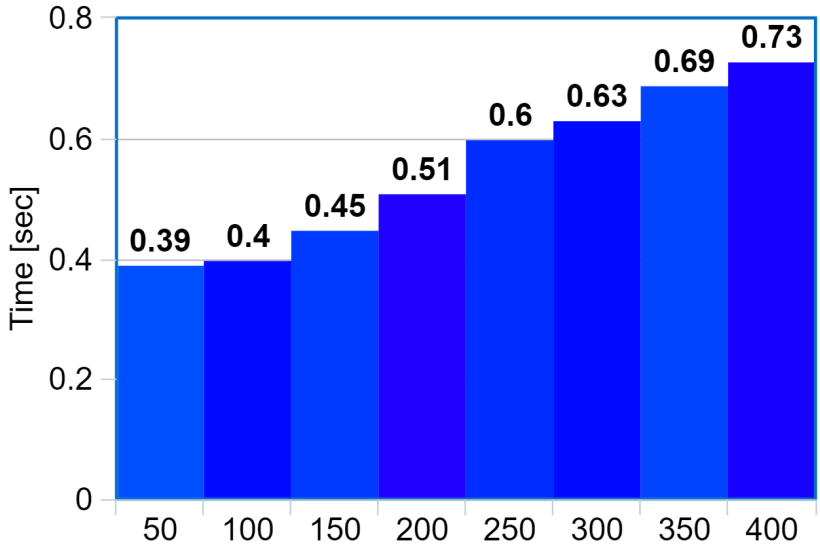


Figure 4.8: Running time with different reservoir sizes from the self made network

Figure 4.8 shows the result of the ESN with different sizes of the reservoir. The values are the mean of five runs of the ESN program for each reservoir size. Table 4.1 presents the average error with the different reservoir sizes.

Neurons	NRMSE
50	0.129
100	0.116
150	0.112
200	0.109
250	0.107
300	0.106
350	0.107
400	0.106

Table 4.1: NRMSE with different reservoir sizes on the test set

After testing the different sizes on the reservoir and comparing their error, the reservoir size was set to 300 neurons.

After running the grid search and experimenting with different hyper-parameters the optimal parameters were chosen. The parameters used in the Echo State Network are reported in Table 4.2.

Parameters used in ESN	
Leak rate	0.14
Sparsity factor	0
Size of the reservoir	300
Spectral radius	0.99
Input scaling	0.1
Bias scale	0.1
Warmup	200

Table 4.2: Hyperparameters used in the ESN

These values are as expected since the dynamic system is nonlinear and has relatively slow dynamics.

4.3 ESN-PNMPC

The controller ESN-PNMPC was crated in Python 3. The controller was adapted from an already existing library develped by Jean Panaioti Jordanou [34]. The same parameters mentioned in Section 4.1 are used for the ESP model, and the parameters for the Echo State Network defined in Table 4.2 were used to create the controller.

First, a low-pass filter for the predictions was implemented to treat disturbances and modeling errors. The pole for the filter given in (3.21) is placed at 0.5 for each variable.

A rule of thumb in choosing the control and prediction horizon is that the prediction horizon should be as long as the system needs to reach a steady-state. A longer prediction horizon will increase the computational time drastically, and unnecessarily. With some experiments, the system needs little more than two minutes to reach a steady-state. As the system has 12 data samples per minute, the prediction horizon is thus set to 30 samples in order for the prediction horizon to capture the steady-state. The control horizon was chosen by some trial and error approach which, after some experiments, was set to 10 samples. The control horizon decides how many decision variables the solver (QP) must optimize, meaning that a smaller control horizon will reduce the computational complexity.

Now that the predictor is created, the next task is the control instance. The main goal is to control the p_{bh} state while maximizing the flow (q). This is done by adding a reference point on the p_{bh} state, which is chosen by an operator, and a reference point on the flow. Since the goal is to maximize the flow, the flow reference is set to $55m^3/h$ which is the maximum value that the flow can achieve with the constraints implemented. The first reference on p_{bh} is 82 bar, therefore the initials values for the inputs are 75 % for the valve opening and 50 Hz for the frequency.

As mentioned in Section 3.9 the are two weighting matrices that help prioritize what the controller is going to do. Since the main goal is to reach the reference on the p_{bh} state, the bottom-hole pressure receives higher values in the weighting matrix than the flow (q). The choice of values in the weighting matrices was found with a trial and error approach. After finding values that make the system reach the reference without any oscillations and with a smooth trajectory, the weighing matrices $\mathbf{Q}$ and $\mathbf{R}$ were set to:

$$\mathbf{Q} = \begin{bmatrix} 0.07 & 0 \\ 0 & 0.001 \end{bmatrix} \tag{4.2}$$

$$\mathbf{R} = \begin{bmatrix} 3 & 0 \\ 0 & 5 \end{bmatrix} \tag{4.3}$$

In Section 2.2.1 it was mentioned that the lifetime of an ESP can be expanded with the implementation of suitable constraints. The constraints are implemented

on the inputs and the outputs. The authors of [10] suggest some values for the constraints, which are defined based on physical considerations and system knowledge. The constraints are shown below in (4.4).

$$0 \leq z\ (valve) \leq 1\ [\%] \tag{4.4a}$$
$$35 \leq f\ (frequency) \leq 65\ [Hz] \tag{4.4b}$$
$$0 \leq q\ (flow)\ [m^3/s] \tag{4.4c}$$
$$\Delta U \leq 0.1 \tag{4.4d}$$
$$1 \leq p_{wh} \leq 60\ [bar] \tag{4.4e}$$
$$1 \leq p_{bh} \leq p_r\ [bar] \tag{4.4f}$$

4.4 Summary

In this chapter the implementation was presented. Along with all the hyper-parameters and libraries used for creating the ESP model, ESN and ESN-PNMPC. A short summary for each hyper-parameter is shown below:

ESP - CasADi:	
p_{bh0}	70 bar
p_{wh0}	30 bar
q_0	$36m^3/h$
Samples per minute	12

ESN:	
Leak rate	0.14
Sparsity factor	0
Size of the reservoir	300
Spectral radius	0.99
Input scaling	0.1
Bias scale	0.1
Warmup	200

ESN-PNMPC:	
Control Horizon	10
Prediction Horizon	30
Filter pole	0.5
Q	$\begin{bmatrix} 0.07 & 0 \\ 0 & 0.001 \end{bmatrix}$
R	$\begin{bmatrix} 3 & 0 \\ 0 & 5 \end{bmatrix}$

Table 4.3: Summary of the hyperparameters used

Chapter 5

Results

In this chapter, the results are presented. As mentioned in Chapter 2, this dissertation addresses three main problems. This chapter is structured in the same way, below is a short summary of the main problems:

1. Synthesis of the ESP;
2. ESN model for the ESP;
3. ESN-PNMPC for the control of the ESP.

5.1 ESP with CasADi

This section will simulate the system obtained from Section 2.3.1 to verify the model. Different input values are tested to see how the system is responding. All simulations use the same initial values:

- $P_{bh0} = 70$ bar.
- $P_{wh0} = 30$ bar.
- $q_0 = 36$ m^3/h

5.1.1 Valve opening and frequency as constants

In the following simulation, the valve opening and the pump frequency are constants, with the values:

- Valve opening at 100% ($z = 1$).
- Frequency at 53 Hz ($f = 53$ Hz).

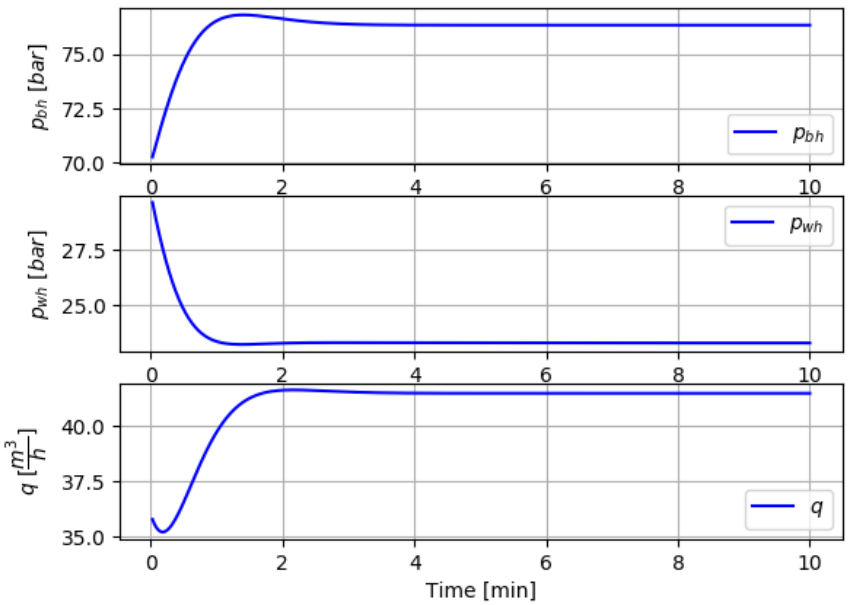


Figure 5.1: System response to a valve opening at 100% and an ESP frequency at 53 Hz.

A desirable behavior for this simulation is that each state reaches its steady-state. This simulation was also used to find suitable initial values. From Figure 5.1 one can see that all states reach their steady-state and are stable. This simulation provided the basis for further simulation.

5.1.2 Step response on valve opening

In Figure 5.2 a step response is used on the valve opening while keeping the frequency constant. The values are given as:

- Valve opening starting at 50% then increasing to 100% after five minutes ($z_0 = 0.5$, $z = 1$).
- Frequency is constant at 53 Hz ($f = 53$ Hz).

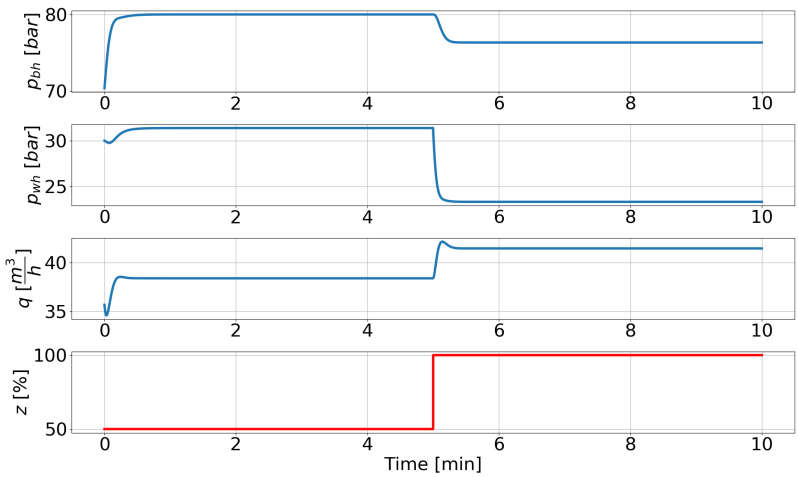


Figure 5.2: System response to a step response on the valve, starting at 50% and increasing to 100%.

The simulation in this section is to look at how the system is responding to a step response in the valve opening after 5 minutes. When the valve opening increases from 50% to 100% it is expected that both the bottomhole pressure p_{bh} and the wellhead pressure p_{wh} will drop, while the liquid flow q will increase. An open choke valve will lead to more flow and less pressure, a behavior that can be seen in Figure 5.2.

5.1.3 Increasing frequency

The last experiment done on the simulation was to increase the frequency while keeping the valve opening as a constant. The values for the inputs are:

- Valve opening constant at 100% ($z = 1$).
- Frequency starting at 40 Hz ($f = 40$ Hz), and increasing with 2 Hz every minute for 10 minutes.

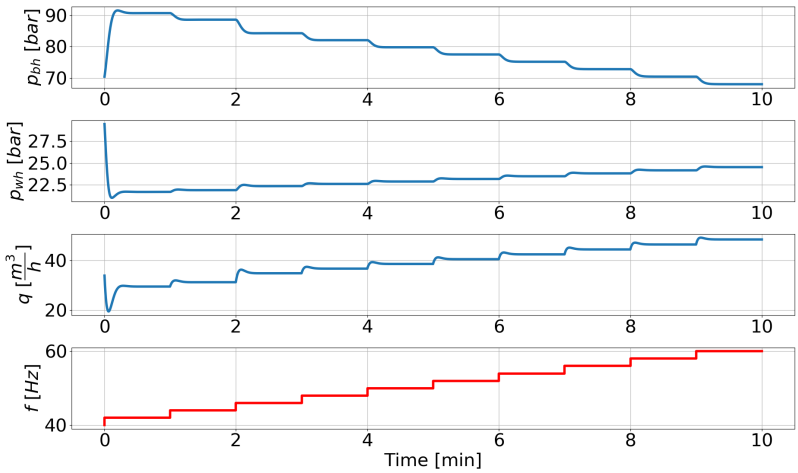


Figure 5.3: System response with an increasing frequency

This simulation is done to check how the system would respond to a change of frequency in the ESP. An expected outcome would be an increase of pressure in the wellhead p_{wh} and a decrease in the bottomhole p_{bh} as well as an increase of the liquid flow. Simulations showed that the system responded well to the change of frequency and performed as expected.

5.1.4 Discussion

The goal of this section was to verify the system model based on simulations. Different scenarios were used to see how the system model responded. It is essential to perform different simulations to a DAE model since small errors could cause huge deviations and the fact that the system is too complex to understand from equations alone. The model responded as expected on all simulations, and will be used for further control and optimization in this dissertation.

5.2 ESN

In this section, the results from the data-driven ESN model will be presented. This section presents plots to analyze the ESN performance in different scenarios and with different training sets. Starting with a validation of the ESN made from scratch, then how different training sets influence the result, and finally how the ESN performs when noise is added to the training set. All the plots are from a test set that the network has not been trained on.

5.2.1 Comparison between Oger and the self-made ESN

In this section, the result from using an ESN made from scratch and an ESN made with the Oger toolbox will be compared. The plots below show the test with 1000 data points, which is equal to 83 minutes of pump operation.

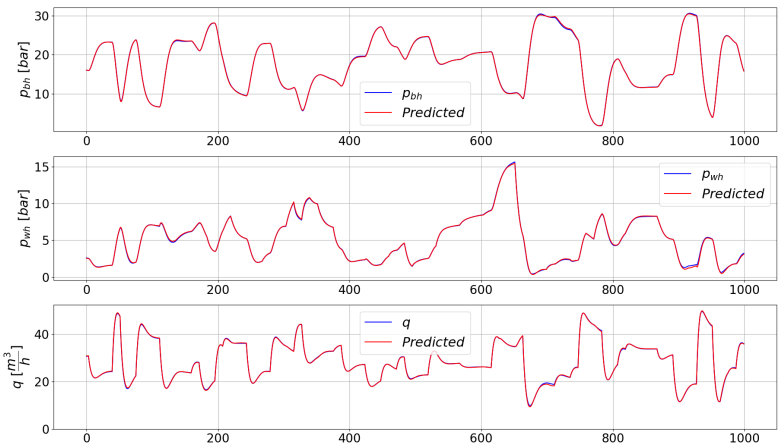


Figure 5.4: Results for ESN made from scratch

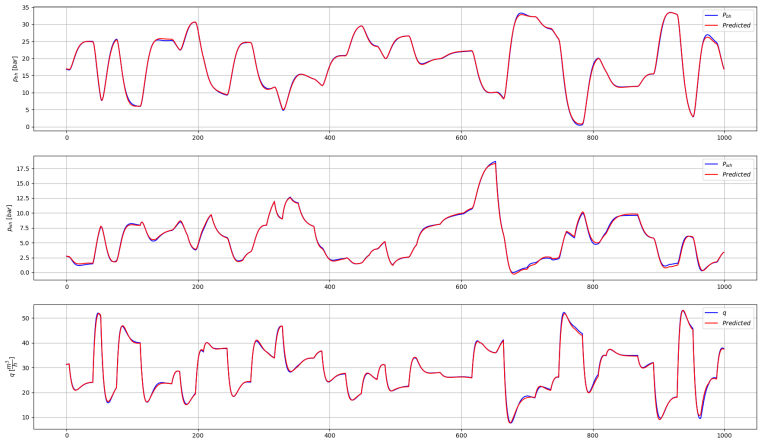


Figure 5.5: Results for ESN made from Oger

When comparing the two plots, it is easy to see that they are identical. Concluding that the ESN made from scratch is correct and can be used further in this dissertation.

5.2.2 Different training sets

This subsection shows the result of the ESN with different training sets. To validate the network, a test set was created with two different frequencies of input change, one containing slow input change and another with fast input change. This is because it is desired to see the network's performance in every scenario. With this test and the steady-state test, the next section will determine which training set will be used later when controlling the ESP.

Frequent change of input

This training set for the ESN only contains input signals that are changing frequently (quickly), as shown in Figure 4.4. Figure 5.6 shows the predictions of the trained ESN on a test set composed of both fast and slow frequency of input change.

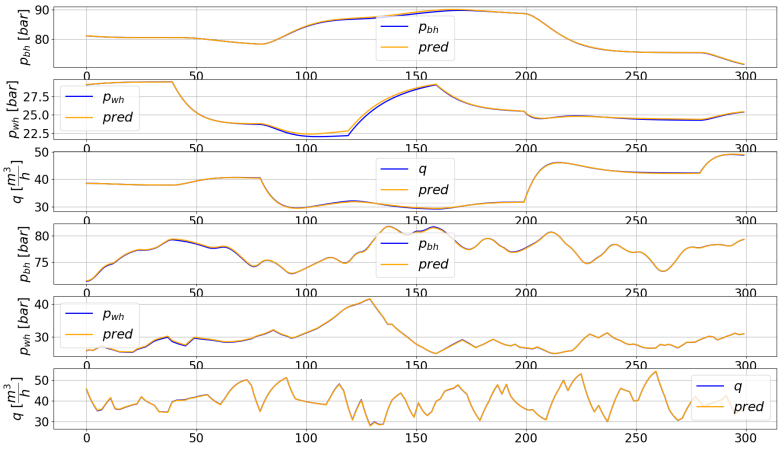


Figure 5.6: Results for ESN trained on a training set with a frequent change of input

Figure 5.6 shows the result of running the network with the test set. The first three subplots are generated with a slow frequency of input change, while the remaining three are produced with a fast frequency of input change. The network performs well with regards to the state prediction on fast and slow input change. Some deviations when predicting the slow input change were noticed, which is expected since the network has only trained on fast frequency.

Slow changing input

The training set used in this network is shown in Figure 4.5, which consists of training data obtained with slow changing input. Accordingly, the output results from the network are shown in Figure 5.7.

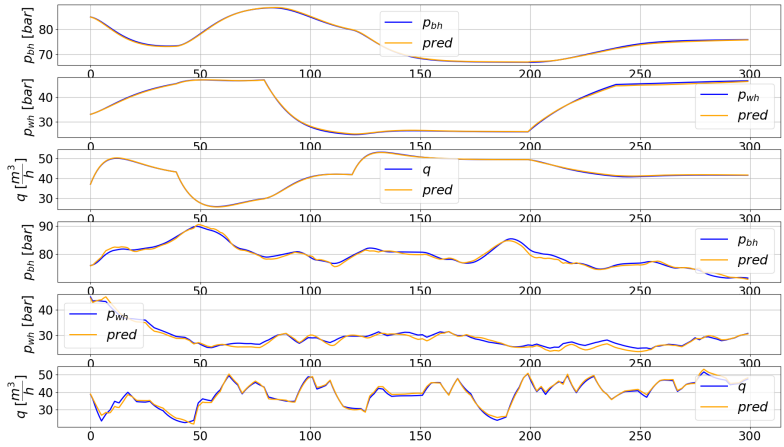


Figure 5.7: Results for ESN trained on a training set with a slow input change

As seen in Figure 5.7 the network performs better on the slow input change and worse on the fast input change. This behavior is expected because the network was trained only with slow changing input data.

Combining fast and slow input change

Here the training set is the one shown in Figure 4.6. The results appears in Figure 5.8 when the network is tested on a data set with slow and fast changing inputs.

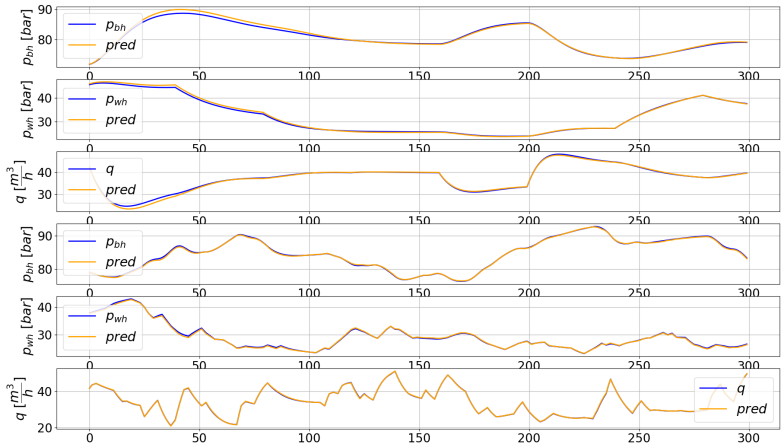


Figure 5.8: Results for ESN trained on a training set with a combination of slow and fast input change

As seen in Figure 5.8 the network performs better on the fast changing input, with some small deviations on the slow changing input, but overall the performance is good.

Conclusion

When comparing the different results in this subsection, the best results were obtained when using the training set with fast changing inputs and the combination of fast and slow changing inputs. In Table 5.1 the error is measured using the error metric MSE shown in (3.25). The next experiment is to test the both networks on a steady-state test. Using a test set where the inputs are constant, in order to determine which training set is best suited for this dissertation.

MSE ERROR	$\mathbf{p_{bh}}$	$\mathbf{p_{wh}}$	$\mathbf{q}$
Slow input change	$1.23363852e^{9}$	$4.74439494e^{9}$	$2.94843411e^{-7}$
Frequent input change	$1.63031166e^{9}$	$2.73441057e^{9}$	$1.98888881e^{-7}$
Combination of both	$1.90994956e^{9}$	$1.76191664e^{9}$	$1.91219397e^{-7}$

Table 5.1: MSE error of the experiments above

5.2.3 Steady-state using previously trained networks

This subsection demonstrates the performance with the trained networks from Section 5.2.2, with a view to reach a steady-state. It is important that the network predicts the correct steady-state because when the controller is implemented, the goal is to reach a reference and settle. If the predicted value has a deviation from the plant itself the controller will perform poorly. The results from this and the last subsection will determine which training set is best suited for training the ESP controller.

Frequent change of input

Figure 5.9 shows the steady-state test performed on the network trained with a frequent changing input signal.

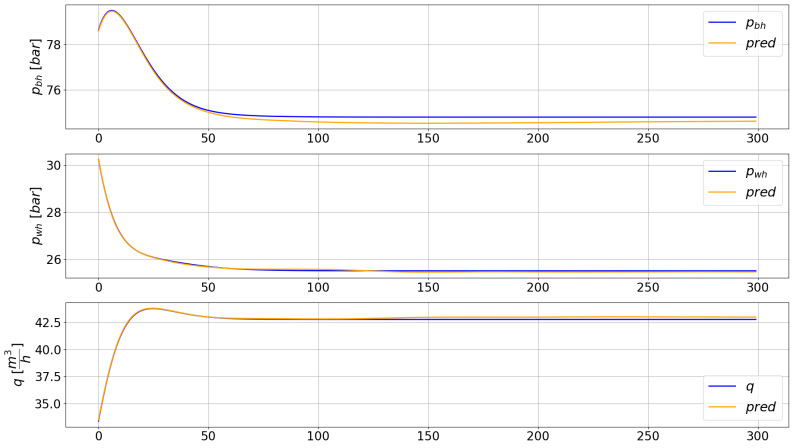


Figure 5.9: Steady state test with a fast changing input as the training set

As seen in Figure 5.9 the network performs well with respect to reaching a steady-state. Only a small deviation on the predicted value from the true value on the states p_{bh} and q is observed.

Slow changing input

This subsection will show the performance of the network trained with a slow changing input.

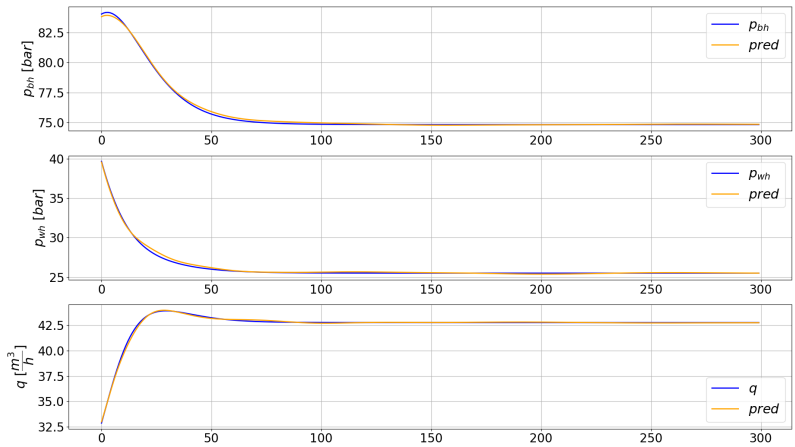


Figure 5.10: Steady-state test with slow input change as the training set

As seen in Figure 5.10 the network performs almost perfectly when testing the steady state.

Combining fast and slow changing inputs

This subsection shows when using the combination of both fast and slow changing inputs on the training set.

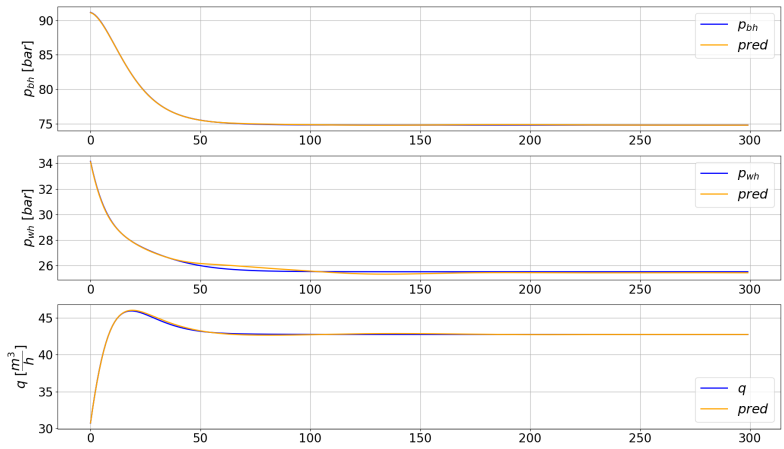


Figure 5.11: Steady state test with slow and fast changing inputs

As seen in Figure 5.11, the network predicts the same steady-state values, which is the goal for this experiment.

Conclusion of which training set to use

As seen in the experiments above, the performance of the networks is quite good when tested with constant input and the system goes to steady-state. There is a little deviation on the p_{bh} state when using the fast input change as a training set. The slow input change network has some deviations when it predicts fast input change. As a conclusion of this section, the training set that performs best in all experiments is when combining both slow and fast input change, and this training set will be used later when controlling the pump. Below in Table 5.2 is the MSE error of the steady state experiments above.

MSE ERROR	$\mathbf{p_{bh}}$	$\mathbf{p_{wh}}$	$\mathbf{q}$
Slow input change	$1.47251573e^{8}$	$8.14165536e^{8}$	$4.44466134e^{-8}$
Frequent input change	$2.22688637e8$	$9.33468400e^{8}$	$5.58092668e^{-8}$
Combination of both	$1.39484844e^{8}$	$8.08089634e^{8}$	$4.37245426e^{-8}$

Table 5.2: MSE error of the steady-state experiments above

5.2.4 Noise added

This section will test how the network performs when noise is added to the training set. The last section concluded that the best training set was when combining fast and slow input change and therefore it is used further. The training set with noise added is shown in Figure 4.7. Figure 5.12 shows the result of the test set which is free of noise, the test set is free of noise because it is desired to see how the network performs when it is trained with a noisy training set and compare the result with a test set without noise.

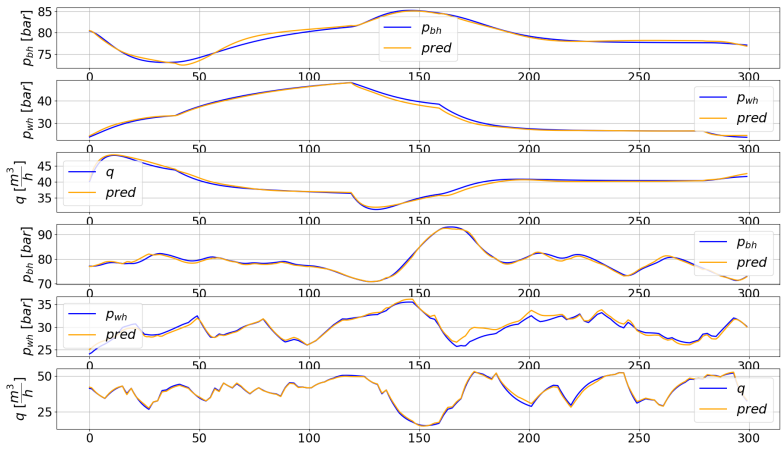


Figure 5.12: Predictions of a network trained with a data set with noise added

As seen in Figure 5.12, even though the network was trained with a noisy training set, the result is relatively good. When comparing it to the result without noise (Figure 5.8) it misses a little bit more in some places which are expected when using a training set with 10 % noise added to each data point.

5.2.5 Summary

The goal of this chapter was to create a well functioning Echo State Network, that imitates the behavior of the pump. Many experiments were performed with different parameters and sizes of the data sets to find the best combinations. All tests had good results and all the parameters had an expected value since the system has slow dynamics and is nonlinear. If the results had big deviations from the actual model, then it would result in poor control performance when the controller uses the ESN as the predictor part of the PNMPC. In conclusion, the parameters and training set used for the ESN are adequate and optimal for

predicting the behavior of the pump. The network achieved good performance and will be used later to control the pump.

5.3 ESN-PNMPC

This section will present the results of using the ESN-PNMPC controller. The goal is to set a reference on the state p_{bh} while maximizing the flow (q). The reference on p_{bh} is usually set by an operator and is commonly not changed frequently. In these experiments, the reference is changed often so the controller can be tested to the fullest. It is desired to have a controller that controls the pump to the reference slowly and without any overshoot and oscillations, because it may be harmful to the pump.

5.3.1 Step response

In this experiment, an easy step response is performed. The reference starts at 82 bar on p_{bh} and then after 15 minutes the reference is reduced to 70 bar. The simulation is done over 30 minutes.

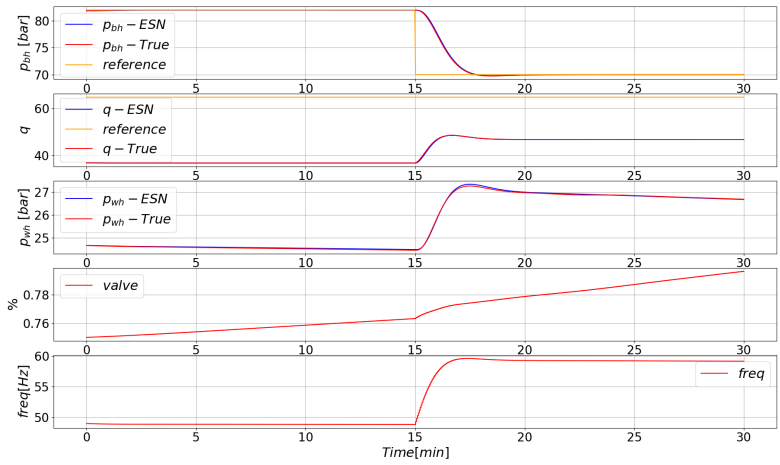


Figure 5.13: ESN-PNMPC performance with respect to a step response

As seen in Figure 5.13, the controller reaches the reference without any overshoot which is desired for the operation of the ESP. Using the error metrics called IAE explained in Section 3.10 shows that the gathered error is 412.45.

5.3.2 Stair response

In this experiment, the reference was arranged in a ladder set up, starting at 82 bar then reduced stepwise down to 72 bar.

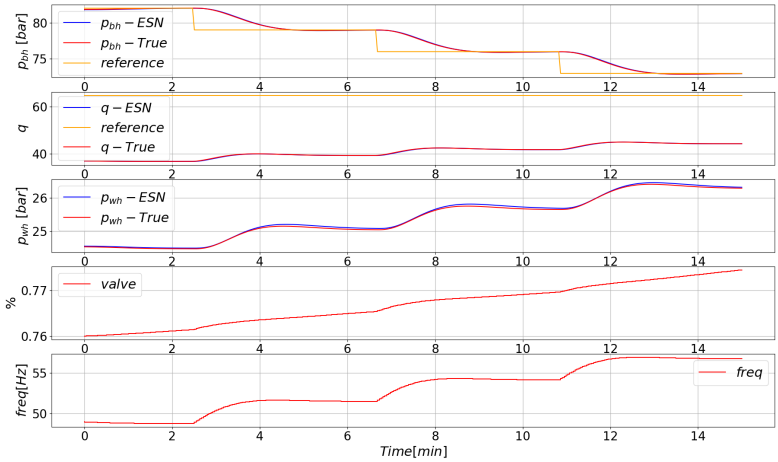


Figure 5.14: ESN-PNMPC performance with references going down

This was a more demanding experiment, designed to assess how the controller performs in a different scenario. As seen in Figure 5.14, the controller works fine when a more difficult experiment is performed. It reaches all the different references with a smooth trajectory. The same error metric is used for evaluating this experiments as for the last. The gathered error is 303.71.

5.3.3 Steps down and up

This is the final experiment tested with the controller. In this experiment, the reference goes stepwise down, then up again back to the start point.

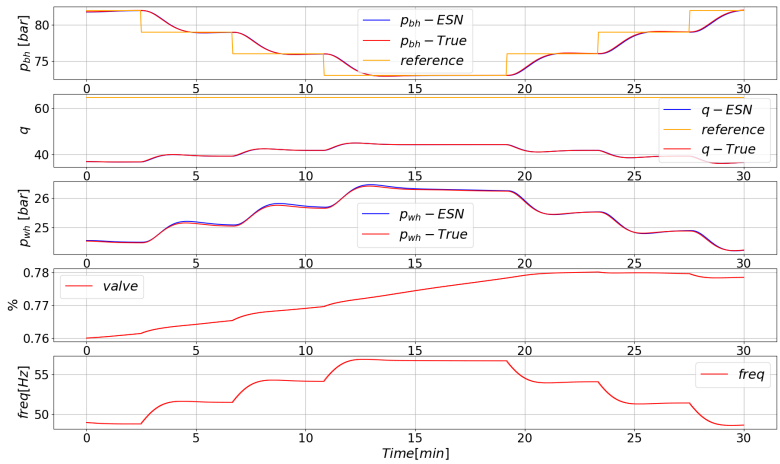


Figure 5.15: ESN-PNMPC performance with references going down and up

This was the most demanding experiment performed and the controller still reaches all the different references, without any overshoot and with a smooth trajectory. The IAE gathered from the experiment is 586.23

5.3.4 Conclusion of ESN-PNMPC

The goal of the controller is to control the state p_{bh} to a reference with a smooth trajectory, without oscillations and overshoot. By selecting suitable hyperparameters, a desired behavior is achieved for the controller. The results from the different experiments show that the controller can operate the pump effectively and the chosen parameters are suitable.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation, an ESN-PNMPC approach is used to control an ESP. The dissertation consists of three main subjects that needed to be fulfilled: the ESP model, ESN of the model, and the ESN-PNMPC. The conclusion of each of these subjects is described below.

The ESP model was created in CasADi, this toolbox offers fast computations for nonlinear optimization and algorithmic differentials. For this problem, CasADi worked great for modeling the ESP. The only downside for using CasADi is the feedback for the error messages, which is confusing. There are two hyperparameters to choose when using this model, and it is the time step and the initial value for the differential equations. Different experiments were performed for both of these parameters. The initial value does not change the performance of the model, but should be chosen somewhere close to the first reference so that the pump does not have to change too much and abruptly before reaching the reference. The other hyperparameter is the time step which influences how many samples per minute the pump is giving. This value should be high enough to catch the dynamics from the pump, and after some testing, the sample rate should be 12 or more samples per minute. To verify the model, the plots from the different scenarios were compared to other ESP articles [7] and [10].

The next subject is the Echo State Network, that brings fast and efficient system identification. When creating the ESN, multiple hyperparameters need to be set correctly. Finding these parameters can be very time consuming, and therefore the toolbox Oger was used. With the gridsearch function from Oger, the best combination for the hyperparameters that gives the lowest error is found. For creating the ESN from scratch, the library NumPy was used, which is easy to use and worked well for this project. Finding the right training set was crucial for making the ESN is optimal. Therefore several different experiments were performed to find the best training set that gave the best results. The experiments were comprehensive and after comparing the results a combination of a slow input change and fast input change were chosen. It has been shown that the ESN successfully learns the dynamics of an ESP and predicts the behavior of the pump with very little deviations.

The last subject is the ESN-PNMPC, with comprehensive experiments performed. The goal was to control the ESP to the desired reference on the state p_{bh} while maximizing the flow. It was very time consuming tuning the controller for the

desired trajectory and many different values for the prediction horizon, control horizon, and the weighting matrices was tested to find the most optimal values. The difficulties were finding a reference that was suitable for testing the controller to the fullest and was realistic for a real-time scenario. The controller showed good performance controlling the ESP to the desired reference without any overshoot, oscillations, and with a smooth trajectory. If the tuning of the weighting matrices was too high often a big overshoot could occur which can be harmful to the ESP.

The conclusion of the problem statement **"How will an ESP perform when an ESN-PNMPC approach drives the pump to its desired references?"** is that the controller is showing good performance with excellent results. It can follow the desired reference without any faults.

All in all, the scenarios that were created were validated with comprehensive experiments and showed good results in all cases.

6.2 Further work

The further work for this dissertation should be geared towards implementing different cost functions. One aspect that should be considered is the economic part of using the pump. The pump uses a lot of power in the form of electricity, which has an economic impact on the usage of the pump. Therefore, a cost function that reaches the desired reference while minimizing the power usage is desirable.

Another aspect of further work is to make a more realistic ESP model. In this dissertation, some assumptions were made on the ESP to make it easier to model. Further, the viscosity and the well productivity index (PI) should be implemented as variables.

Also, some improvements in the implementation with the CasADi model should be pursed to make it more robust. Currently, if a particular combination of inputs capable of pushing the model's states out of its boundaries, the model will fail to produce meaningful results, which can lead to a system crash. Therefore, additional constraints should be implemented in the RFRAS function for creating the random inputs, making it more robust.

References

[1] S. Hou and Z. Wang. "From model-based control to data-driven control: Survey, classification and perspective." In: (2013).

[2] J. P. Jordanou et al. ""Nonlinear model predictive control of an oil well with echo state networks." In: (2018).

[3] A. Plucenio. "Development of non linear control techniques for the lifting of multiphase fluids." In: (2013).

[4] Jean Panaioti Jordanou et al. "Nonlinear model predictive control of an oil well with echo state networks." In: *IFAC-PapersOnLine* 51 (2018), pp. 13–18.

[5] G Takacs. "Electrical Submersible Pumps Manual: Design, Operations, and Maintenanc." In: (2009).

[6] Gabor Takacs. *Electrical Submersible Pumps Manual: Design, Operations, and Maintenance.* San Diego: Elsevier Science, 2017.

[7] *Modelling and model predictive control of oil wells with Electric Submersible Pumps.* IEEE, 2014, pp. 586–592.

[8] *Embedded Model Predictive Control for an Electric Submersible Pump on a Programmable Logic Controller.* IEEE, 2014, pp. 579–585.

[9] *Centrilift Europe and Africa ESP Failures 1999-2008.* Centrilift, 2008.

[10] *Estimation of Flow Rate and Viscosity in a Well with an Electric Submersible Pump using Moving Horizon Estimation.* IFAC-PapersOnLine, 2015, pp. 140–146.

[11] Lev V. Kalmykov and Vyacheslav L. Kalmykov. "A white-box model of S-shaped anddouble S-shaped single-speciespopulation growth." In: (2015).

[12] Ion Matei and Conrad Bock. "Modeling Methodologies and Simulation for Dynamical Systems." In: (2012).

[13] Peter Kunkel and Volker Mehrmann. European Mathematical Society, 2006.

[14] Ding-Zhu DuPanos and M. PardalosWeili Wu. "History of Optimization." In: (2008).

[15] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer Science and Business Media, 2006.

[16] Stephen J. Wright. *Optimization.* URL: https://www.britannica.com/science/optimization (visited on 12/12/2019).

[17] *Ordinary Least Squares Linear Regression: Flaws, Problems and Pitfalls.* URL: https://lcn.people.uic.edu/classes/che205s17/docs/che205s17_reading_06c.pdf (visited on 06/12/2020).

[18] Michael Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[19] URL: https://medium.com/udacity-pytorch-challengers/hyperparameters-for-neural-networks-c50ab565ee3d (visited on 12/12/2019).

[20] Herbert Jaeger et al. "Optimization and applications of echo state networks with leaky- integrator neurons." In: *Neural Networks* 20.3 (2007). Echo State Networks and Liquid State Machines, pp. 335–352. URL: http://www.sciencedirect.com/science/article/pii/S089360800700041X.

[21] Mantas Lukoševičius. "A Practical Guide to Applying Echo State Networks." In: *Neural Networks: Tricks of the Trade: Second Edition.* Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Springer Berlin Heidelberg, 2012. URL: https://doi.org/10.1007/978-3-642-35289-8_36.

[22] Danil Koryakin and Martin V. Butz. *Reservoir Sizes and Feedback Weights Interact Non-linearly in Echo State Networks.* 2012.

[23] Y.J. Reddy B.R. Mehta. "Model Predictive Control." In: (2017).

[24] "Taylor expansions and applications." In: *Mathematical Analysis I.* 2008.

[25] A. Plucenio et al. "A practical approach to predictive control for nonlinear processes." In: 40 (2007).

[26] Jean Panaioti Jordanou, Eric Aislan Antonelo, and Eduardo Camponogara. "Echo State Networks for Practical Nonlinear Model Predictive Control of Unknown Dynamic Systems." In: (2018).

[27] Y. Pan and J. Wang. "Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks." In: (2012).

[28] K. Xiang et al. "Regularized Taylor echo state networks for predictive control of partially observed systems." In: (2016).

[29] Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control.* Springer, 2007.

[30] E. Camacho and C. Bordons. “Model Predictive Control.” In: (1999).

[31] Joel A E Andersson et al. “CasADi – A software framework for nonlinear optimization and optimal control.” In: *Mathematical Programming Computation* (2018).

[32] David Verstraeten et al. “Oger: Modular Learning Architectures For Large-Scale Sequential Processing.” In: *The Journal of Machine Learning Research* 13 (October 2012), pp. 2995–2998.

[33] Stephen J. Wright. *NumPy Introduction*. URL: `https://www.w3schools.com/python/numpy_intro.asp` (visited on 06/18/2020).

[34] Jean Jordanou, Eric Antonelo, and Eduardo Camponogara. “Online learning control with Echo State Networks of an oil production platform.” In: *Engineering Applications of Artificial Intelligence* 85 (October 2019), pp. 214–228.

Appendices

A Model parameters

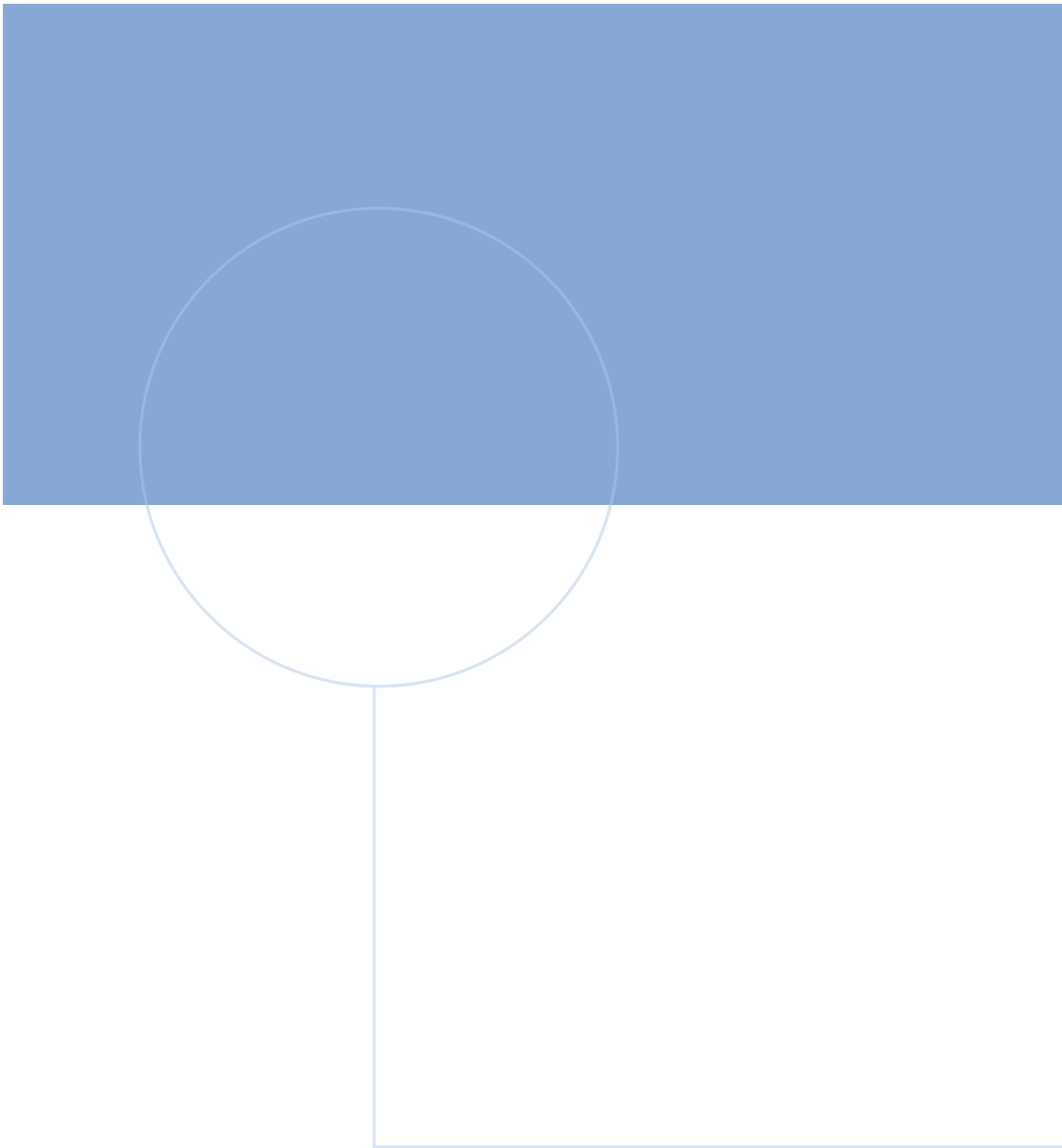
Well dimensions and other known constants			
g	Gravitational acceleration constant	9.81	m/s^2
C_c	Choke valve constant	$2 \cdot 10^{-5}$	*
A_1	Cross-section area of pipe below ESP	0.008107	m^2
A_2	Cross-section area of pipe above ESP	0.008107	m^2
D_1	Pipe diameter below ESP	0.1016	m
D_2	Pipe diameter above ESP	0.1016	m
h_1	Height from reservoir to ESP	200	m
h_w	Total vertical distance in well	1000	m
L_1	Length from reservoir to ESP	500	m
L_2	Length from ESP to choke	1200	m
V_1	Pipe volume below ESP	4.054	m^3
V_2	Pipe volume above ESP	9.729	m^3

ESP data			
f_0	ESP characteristics reference freq.	60	Hz
I_{np}	ESP motor nameplate current	65	A
P_{np}	ESP motor nameplate power	$1.625 \cdot 10^5$	W

Parameters from fluid analysis and well tests			
β_1	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
β_2	Bulk modulus below ESP	$1.5 \cdot 10^9$	Pa
M	Fluid inertia parameter	$1.992 \cdot 10^8$	kg/m^4
ρ	Density of produced fluid	950	kg/m^3
P_r	Reservoir pressure	$1.26 \cdot 10^7$	Pa

Parameters assumed to be constant			
PI	Well productivity index	$2.32 \cdot 10^{-9}$	m^3/s/Pa
μ	Viscosity of produced fluid	0.025	$Pa \cdot s$
P_m	Manifold pressure	20	Pa

Table A.1: Model parameters



NTNU

Norwegian University of
Science and Technology