NTNU
Department of Engineering
Cybernetics

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

TTK4551: SPECIALIZATION PROJECT (7,5 CREDITS)

# Model Predictive Control using data-driven models obtained from Artificial Neural Networks

*Iver Osnes*

supervised by

Prof. Lars Imsland, NTNU

December 21, 2019

# Problem description

## MPC using data-driven models

In many engineering systems, it is challenging to build mechanistic models due to e.g. lack of understanding of physical mechanisms, and/or difficulty in obtaining parameters in the models from data. Sometimes one has to resort to data-driven models.
The topic of this project is to implement (nonlinear) model predictive control on a case study of a gravity separator. The model should be a data-driven model, obtained using ANN.

### Work description

1. Give a brief overview over Artificial Neural Networks (ANNs)
2. Give a brief overview of Model Predictive Control (MPC) for nonlinear (data-driven) models, and numerical methods for optimization in MPC.
3. Implement a chosen nonlinear MPC approach using CasADi and IPOPT, for the data-driven model obtained from ANN.

# Abstract

Real-world problems are frequently hard to model considering its physics. It is normal to find these kinds of problems in oil and gas production, where both modeling and measuring can be difficult. A possibility can be to resort to data-driven models. If one can manage to obtain optimal control for a data-driven model, it is possible to transfer its solution into a real-world problem if the data-driven model is accurate. Earlier studies have shown that one can use neural networks to generate accurate data-driven models. Such models would be preferable since it gives us the possibility of decision taking on behalf of hard data rather than decisions based on observation alone. This project aims to create a data-driven model on a case study of a gravity separator using Artificial Neural Networks (ANNs). Further, Model Predictive Control (MPC) will be implemented to yield optimal control of the obtained model. The MPC gives the ability to predict the future based on previous states while satisfying a set of constraints. Results from this case study has shown that model predictive control is promising to yield optimal control for a three-phase gravity separator. In addition, it showed that neural networks are well suited for model recognition based on large data sets.

# Preface

This specialization project is submitted as a part of the requirements for the master's degree at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). It is submitted for the specialization project TTK4551. The work presented in this report has been carried out under the supervision of Prof. Lars Imsland at the Department of Engineering Cybernetics, NTNU.

I would like to thank Prof. Lars Imsland for guidance and tips throughout the development of this project.

*Iver Osnes*

*Trondheim, December 2019*

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Model-based control (MBC) can be a powerful tool in multiple industries. Real-world systems are often hard to both model and control using on-line data, and it is therefore suitable to utilize MBC for controlling the plant. However, using MBC does not come without drawbacks. The controller may not work well if the plants' behavior deviates from the assumed model. For this reason, it may not be a good idea to create an inaccurate model since it could lead to either bad performance or an unstable closed-loop system [6].

With the development in information science and technology, it has shown that practical processes have undergone significant changes. Specific industries have custom-made technologies and equipment, and the processes are more complex. Modeling based on first principles and identification has become more demanding, and traditional MBC theory has become impractical for these industries [6].

Data-driven control (DDC) is an appropriate alternative for controlling modern processes. Modern processes generate and store huge amounts of data during its processing time. These data can be used to design controllers when an accurate process model is missing [6]. There are multiple definitions of data-driven control. They are, however, based on the same fundament; a controller designed based on the input/output data, and no physical information is used [8].

Data from several studies suggest that artificial intelligence and machine learning is a good approach when it comes to providing data-driven models [6]. Neural networks can be used without prior knowledge and can provide accurate models with a black-box modeling approach. That is, searching for a model purely from input-output data.

Data-driven models are a major area of interest within the field of petroleum. There are multiple complex processes, and each process is critical for maximizing profit. Sufficient models are often hard to both derive and control. The three-phase gravity separator is one of the processes that we define as a black box. A multiphase wellstream is injected into a separation tank, where the states are hard to obtain. A data-driven model of the gravity separator are hard to both obtain and control without priori knowledge.

Model Predictive Control (MPC) has shown to be useful in systems where parts of it are data-driven, e.g. autonomous systems with forecasts as constraints [14]. It has also shown its strength in delivering optimal control policy for a real system with a wrong model [5]. MPC seems to be a good choice when it comes to controlling data-driven models.

## 1.2 Objective

This project explores how MPC can be used to control nonlinear data-driven models. The project consists of two main parts. The first part involves creating a data-driven model of a nonlinear tank using an Artificial Neural Network (ANN). The second part is to control the obtained model using MPC.

## 1.3 Contribution

This project has resulted in the following contributions:

- A nonlinear tank simulator in Matlab/Simulink, suitable for testing and comparing different types of models.
- The development of a nonlinear data-driven model of a tank system. The data-driven model is obtained with the use of ANN.
- The development of an MPC capable to yield optimal control for Optimal Control Problems (OCPs). The MPC program is capable of converting OCPs into Nonlinear Programming (NLP) problems. It is made easy to adjust the MPC to fit larger systems.

## 1.4 Structure

This report consists of six chapters.

- Chapter 2 aims to give an introduction to the theory related to this project.
- Chapter 3 will introduce the tank model, which is the model that will be used in this case study. This chapter will also describe how ANN is used to create a

data-driven model and how we can use MPC to control the obtained model.

- Chapter 4 provides the simulation and results of this case study.
- Chapter 5 will discuss the results and how this work can be scaled into larger systems.
- Chapter 6 presents the conclusion of this work and present proposals to future work.

# Chapter 2

# Theory

This chapter aims to give an introduction to theory related to the implementation in Chapter 3. This includes theory related to the structure of Artificial Neural Networks and Model Predictive Control.

## 2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) can, in its simplest form, be viewed as an imitation of the human brain [15]. The network consists of processing nodes, called neurons. The neurons have an input and an output, as well as a function that determines the activation of the neuron [15]. The connection between the neurons leads to the global behavior of the network and is said to be emergent. This means that the global abilities of the network supersede the abilities of its elements, making the network a powerful tool [6]. A common architecture of an ANN can be seen in Figure 2.1. It is worth noticing that a neuron in a layer is connected with every neuron in the next layer.

The first layer, the input layer, receives the input values. The layer in the middle, the hidden layer, is a set of neurons between the input layer and the output layer. There can be multiple layers inside the hidden layer. The last layer, the output layer, produces the output.

The connection between the neurons is referred to as weights. The input of a neuron will depend on the weight value of this connection. Weightings can be positive, negative or zero. If there is no connection between two neurons, the weight will be zero. The weights are the tuning parameter that makes the network obtain a required output. Algorithms are used to tune the weights of an ANN. The process of tuning wights is called learning or training. [4]

The most frequently used training method for neural networks is the backpropagation algorithm [15]. This method compares the obtained output with the targeted output, and propagate the error back through the layers. The weights between the neurons change as the flow move back. The cycle of going from input to output, and then from

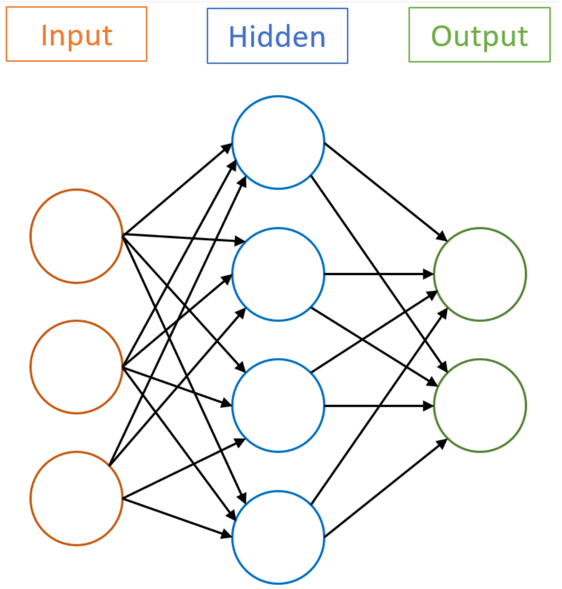


Figure 2.1: A common architecture of an ANN with three inputs and two outputs.

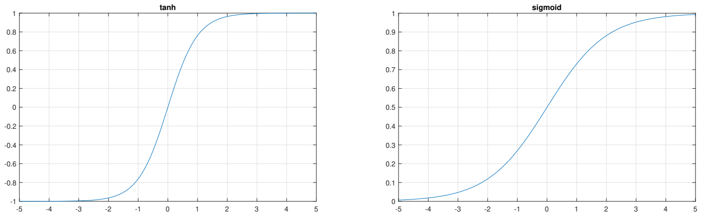
output to input is called an epoch. The system starts with a training set, where the input is known and is asked to obtain a known output. The network performs epochs until the error is within a certain tolerance. When a chosen tolerance is reached, we define our network as trained. The weights obtained from the training are later used for calculating responses from the network when the data is unknown. [15]

### 2.1.1 Activation function

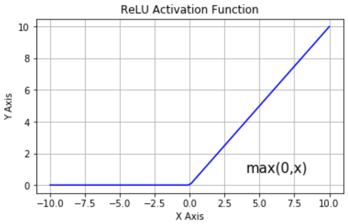
Activation functions give the network the ability to learn and make sense of nonlinear complex mappings between input and output. Without an activation function, the output would be a simple linear function. A linear function is easy to solve, but it dismisses the possibilities of learning complex mappings from data. We can look at a network without an activation function as a linear regression model, with limited power [16]. Three common activation functions can be seen in Figure 2.2.

To understand the architecture, we can look at a simple example of a singe-input neuron in Figure 2.3. $p$ is a scalar input and $a$ is a scalar output. The scalar input $p$ is multiplied with a weighting $w$ before it enters a summation where a bias $b$ is added. The net input $n$ will then proceed into an activation function. We can describe the system as a function:

$$a = f(n) = f(wp + b) \tag{2.1}$$



(a) Tanh function. (b) Sigmoid function.



(c) ReLU function [10].

Figure 2.2: Different types of activation functions.

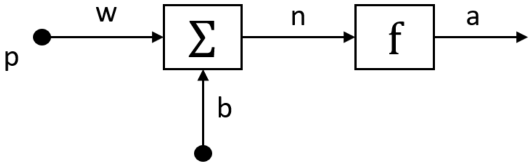


Figure 2.3: Network Architecture and Notation, recreated from [9].

## 2.2 Model Predictive Control

### 2.2.1 Discrete time systems

A general way to describe a system is as a relation between an input and an output, where the input will generate a unique output. If a system is dependent on previous inputs as well as the current input, we call it a dynamical system[2]. Since it is excessively hard to compute a dynamic system that is dependent on all inputs from time $-\infty$ to $t$, it is sufficient to transform the system into a state-space representation or a set of high-order differential equations. When it comes to selecting either state-space representation or high-order differential equations, it can be compared based on simplicity [2]:

1. A system with one or two states will be equally hard to describe with the two methods. However, if a system consists of three or more variables, it would be simpler to develop a state-space equation. Furthermore, state-space equations are more compact.
2. A state-space equation describes not only the relationship between the input and output but also the internal variables, while a high-order differential equation is an external description.
3. High-order differential equations are not suitable for computer computation, because it is hard to discretize. State-space equations, on the other hand, only consists of the first derivative in the discretization, and are more suitable for computer computation.
4. State-space equations are easier to extend than high-order differential equations to describe nonlinear and time-varying systems.

From this comparison, there seems to be no reason to go any further with high-order differential equations. State-space representation of a dynamical system can be expressed as:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(x(t), u(t)) \tag{2.2a}$$

$$\mathbf{y}(t) = \mathbf{h}(x(t), u(t)) \tag{2.2b}$$

where (2.2a) is the state equation and (2.2b) is the output equation. When we sample a continuous system into discrete points, we call it a discrete system. A discrete system can be obtained using the forward Euler method. The discrete-time system can be written as

$$x_{k+1} = x_k + hf(x_k, u_k) \tag{2.3}$$

where $h$ is the step size.

### 2.2.2 Optimization

An optimization problem is a problem of minimizing or maximizing an objective function on a finite set of feasible solutions. A general notation can be expressed as [13]:

- $x$ is the vector of variables, the unknown parameter,
- $f$ is the objective function, the function that we want to maximize or minimize,
- $c_i$ are the constraint functions, which are functions of $x$ that define equalities and inequalities that the unknown vector $x$ must satisfy.

By using this notation, we can rewrite the optimization problem as follow:

$$\min f(x) \quad \text{subject to} \begin{cases} c_i x = 0, i \in \mathcal{E} \\ c_i x \geq 0, i \in \mathcal{I} \end{cases} \tag{2.4}$$

where $\mathcal{E}$ is the set of equality constraints and $\mathcal{I}$ is the set of inequality constraints.

How an optimization problem is solved depends on its objective function and constraints. The three most common problems with their respective solving algorithm are:

- Linear programming (LP): All functions are linear in this problem. The simplex algorithm is a suitable algorithm for LP problems. LP problems are convex.
- Quadratic programming (QP): Quadratic objective function and linear constraint functions. An active set algorithm is suitable to solve QP problems. This problem is convex.
- Nonlinear programming (NLP): The problem has nonlinear equality constraints. Sequential quadratic programming (SQP) is a suitable algorithm for NLP problems. This problem is not convex.

### 2.2.3 Building an MPC

Model Predictive Control (MPC) refers to a class of control algorithms that can be used to predict and optimize future process behavior by utilizing an explicitly formulated process model [12]. MPC controllers can take constraints in account both in manipulated variables (input) and states/controlled variables. A general term for the MPC can be described as a controller which [7]:

- predicts future behavior with the use of a multivariable process model,
- optimizing predicted future performance by using a class of control algorithms,
- can handle constraints on both inputs and states/controlled variables.

When it comes to optimization, minimizing the sum of the running cost is a normal approach. The running cost is defined as the difference between the predicted state and the reference state, and the difference between the control action and the control reference. We define the cost function as the sum of the running costs along the whole prediction horizon. The running cost can be written as:

$$\ell(\mathbf{x}, \mathbf{u}) = \left\|\mathbf{x}_u - \mathbf{x}^{ref}\right\|_Q^2 + \left\|\mathbf{u} - \mathbf{u}^{ref}\right\|_R^2 \tag{2.5}$$

and the cost function can then be formulated as:

$$\mathrm{J_N}(\mathbf{x}, \mathbf{u}) = \sum_{\mathrm{k=0}}^{\mathrm{N-1}} \ell(\mathbf{x}_\mathrm{u}(\mathrm{k}), \mathbf{u}(\mathrm{k})) \tag{2.6}$$

we can now formulate the optimal control problem (OCP) as:

$$\min_{u} \quad \mathrm{J_N}(\mathbf{x}_0, \mathbf{u}) = \sum_{\mathrm{k=0}}^{\mathrm{N-1}} \ell(\mathbf{x}_\mathrm{u}(\mathrm{k}), \mathbf{u}(\mathrm{k})) \quad \textit{subject to:} \begin{cases} \mathbf{x}_u(k+1) = \mathbf{f}(\mathbf{x}_u(k), \mathbf{u}(k)) \\ \mathbf{x}_u(0) = \mathbf{x}_0 \\ \mathbf{x}(k) \in \mathrm{U}, \forall \mathrm{k} \in [0, \mathrm{N}-1] \\ \mathbf{x}(k) \in \mathrm{X}, \forall \mathrm{k} \in [0, \mathrm{N}] \end{cases} \tag{2.7}$$

Future outputs for a horizon $N$, named the prediction horizon, are predicted at each time instant $t$ using the process model. The future control signals $u$ are obtained by solving an optimization problem $l$ at each time step $t$. The control signal $u(t|t)$ is sent to the process, while the other control signals in $N$ are rejected, since the sampling instant $y(t+1)$ is now know, and would be used in the next time instant where all the new values will be brought up to date [1]. An illustration on the MPC principle can be seen in Figure 2.4. MPC is very attractive to staff that only has a limited understanding of

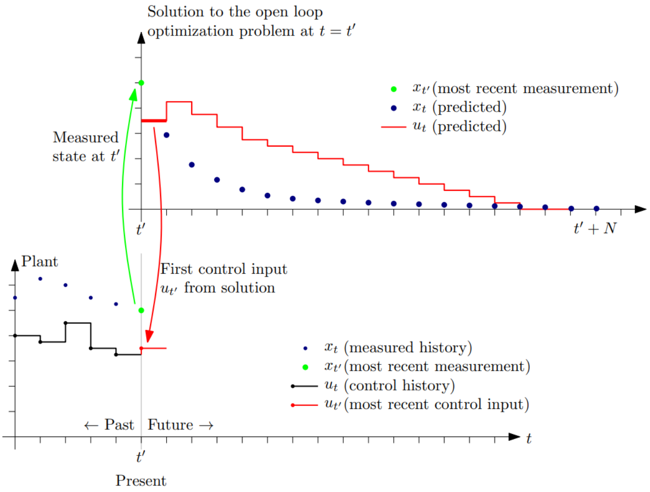


Figure 2.4: Illustration of the MPC principle. [3]

control since the concepts are very intuitive and it is quite easy to tune. It is useful when the future references are known, and the resulting controller is an easy-to-implement control law [1].

### 2.2.4 Solving MPC optimization problems

There exist multiple approaches for solving nonlinear MPC optimization problems. Most of them are, however, based on Sequential Quadratic Programming (SQP) methods. SQP methods are iterative which means that it makes a quadratic approximation to the objective function and a linear approximation to the constraints, and solves a Quadratic Program (QP) at each iteration to find the search direction. A line search is then used to find the next iterate. [7]

There are different approaches to how the QP is specified. The three main approaches are [7]:

- *Single shooting*: Also called *sequential approach*. The optimization variables consists of discretized controls, while the states are removed. The states are eliminated by a forward simulation. A reduced QP problem is then solved.
- *Simultaneous approach*: The model is implemented as explicit equality constraints. Both controls and states are used as optimization variables. Will result in a large number of optimization variables. This approach demand high computational cost.
- *Multiple shooting*: A hybrid of the two other approaches. The control horizon is divided into intervals. The model is, as in simultaneous approach, simulated on each interval. The state at the end of each interval should match the first state at the next interval. This is added as an equality constraint, similar to the simultaneous approach. Multiple shooting will have fewer optimization variables than the simultaneous approach, but will at the same time have more control than the sequential approach over the simulation.

# Chapter 3

# Implementation

This chapter aims to give an overview of the implementation of the case study. The chapter is divided into three parts. It will start with a brief introduction to a nonlinear tank model that will be used in the case study. The next part describes how ANN could obtain a data-driven model. At last, an MPC is implemented to control the data-driven model.

## 3.1 Tank model

The model used in this case study is a nonlinear tank with known dynamics. Figure 3.1 illustrates the tank setup. The inflow rate $Q_{in}$ is set to a constant value for this case study, but this is likely varying in a true system since it is dependent on forces outside the system. The outflow rate $Q_{out}$ is dependent on the water height $h$ and the control variable $u$ and can be formulated as $Cu\sqrt{\rho g h}$. The parameters $A$, $C$, $\rho$ and $g$ are all known constant values. The tank dynamics can be derived from Bernoulli's equation and results in a nonlinear system described by:

$$\dot{h} = \frac{1}{A}(Q_{in} - Cu\sqrt{\rho g h}) \tag{3.1}$$

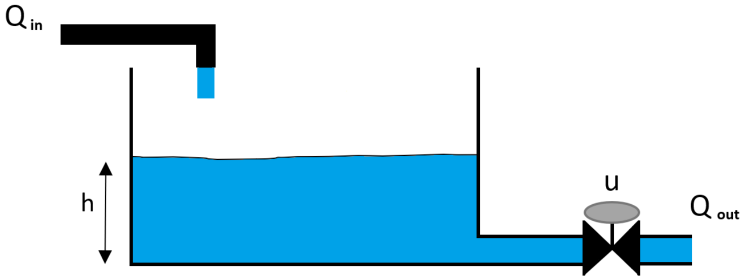


Figure 3.1: Illustration of tank.

Table 3.1: Parameters and variables.

| Parameter | Value | Description |
|---|---|---|
| $h$ | var | Height of the water level [m] |
| $\dot{h}$ | var | Velocity of the water level [m/s] |
| $u$ | var | Control variable |
| $Q_{in}$ | 5 | Water inflow [$\mathrm{m}^3$/s] |
| $\rho$ | 1000 | Fluid density [kg/$\mathrm{m}^3$] |
| $g$ | 9.81 | Acceleration of gravity [m/$\mathrm{s}^2$] |
| $A$ | 20 | Area of tank [$\mathrm{m}^2$] |
| $C$ | 0.15 | Valve constant |

Matlab and Simulink were used to build the system. A PID-controller was implemented on the valve to control the system during the development. The parameters $A$ and $C$ were adjusted until the system reached a favorable steady-state value.

### 3.1.1 Data set

It requires a large amount of data to build a data-driven model. A simulation of the system with a random control variable $u$ was used to create the data set. The control variable $u$ was changed to a random value between 0 and 1 every $20s$. A new sample was made every $0.2s$ and the simulation lasted for $10\,000s$, which generated a total of 50 000 data points. Each data point consisted of the control variable $u$, the water level $h$ and the velocity of the water level $\dot{h}$. Since $\dot{h}$ is dependent of $u$ and $h$, we label $u$ and $h$ as input, and $\dot{h}$ as output.

## 3.2 Creating a data-driven model

The Artificial Neural Network was created in Spyder. Spyder is an open source integrated development environment (IDE). The network was designed using the TensorFlow library and Keras as a model-level library. TensorFlow is an open source platform for machine learning. It was implemented using high-level APIs with Python as the programming language. The data set created in 3.1.1 were split into two separate sets. The first data set, the training data, was a set of 40 000 data points. The second data set, the test data, was a set of 10 000 data points. The network was built with a ”try and error” approach, and the flowchart for the process can be viewed in Figure 3.2.

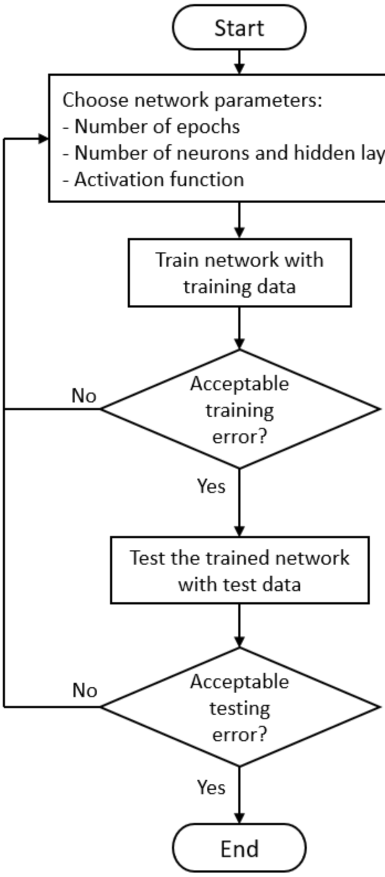

Start
Choose network parameters:
- Number of epochs
- Number of neurons and hidden layers
- Activation function
Train network with training data
No
Acceptable training error?
Yes
Test the trained network with test data
No
Acceptable testing error?
Yes
End

Figure 3.2: Flowchart of the network building process.

It was desirable to keep the network as simple as possible since it was tended to recreate a simple model. The chosen network structure can be seen in Figure 3.3. The input is $u$ and $h$, and the output is $\dot{h}$. Two hidden layers were added. 50 epochs were used in the training process.

Since the system is nonlinear, a nonlinear activation function was selected. *Tanh* was selected for both the hidden and output layer. This makes it easy for the model to adapt with a variety of data. *Tanh* have an output range from -1 to 1, and are well suited for feed-forward networks.

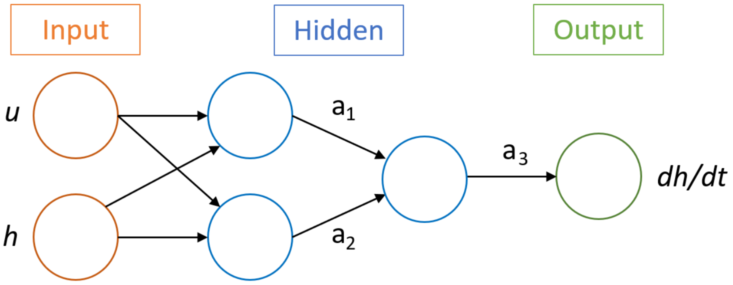


Figure 3.3: The neural network obtained from testing.

"Adam" was used as optimizing algorithm with mean squared error (MSE) as loss function. The testing plot from the trained network can be seen in Figure 3.4. The transition from Spyder to Matlab did not go as intended. Matlab had issues with the file format of the network, and it had to be calculated manually. Weights $W$ and biases $b$ from the trained network were as follows:

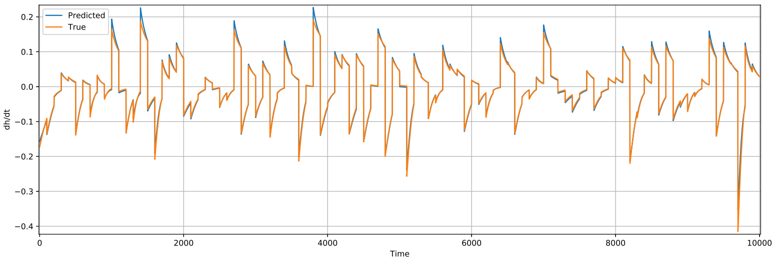


Figure 3.4: Trained network on testing data in Spyder.

$$W_1 = \begin{bmatrix} 1.3845 & -0.1701 \\ -0.5985 & -0.1124 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 1.0905 \\ 1.0039 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} -0.8537 & 1.4550 \end{bmatrix}, \quad b_2 = -0.1414$$

The data-driven model was obtained using eq. 3.2 recursively.

$$a = f(n) = tanh(wp + b) \tag{3.2}$$

## 3.3 Building the MPC

The last step of the implementation part was to implement an MPC to perform optimal control on the data-driven model. This was done in Matlab with CasADi as a nonlinear optimization tool. The MPC aims to control the liquid height $h$ inside the tank.

### 3.3.1 Defining the OCP problem

The first step in the process was to define an Optimal Control Problem (OCP). In control theory, this is equal to finding a control law for a dynamical system for optimal control. A common approach is to look at the deviation in the state and control. The OCP was therefore defined as:

$$\min \sum_{k=0}^{n} (h_k - h_r)^2 + (u_k - u_{ss})^2 \quad \text{subject to} \quad \begin{cases} h_{k+1} = f(h_k, u_k) \\ 0 \leq u_k \leq 1 \\ \underline{h} \leq h \leq \overline{h} \end{cases} \tag{3.3}$$

where $h_k$ is the height at time step $k$ and $h_r$ is the reference height. Further, $u_k$ is the control input at time step $k$, while $u_{ss}$ is the control input when the system is in steady state. In other words, we are penalizing the system for height offset and high control usage. Weighting matrices $Q$ and $R$ are set to identity.

In order to solve an OCP numerically, we need to transform it into a nonlinear programming (NLP) problem [11]. The method of choice is *single shooting* since it is the most intuitive method. NLP problems are solved using the CasADi optimizing tool. It is important to remember that *single shooting* is unfavorable if the prediction horizon $N$ is large, since the integrator function tends to become highly nonlinear. [11]

### 3.3.2 Implementation in Matlab

The NLP problem was defined in Matlab with CasADi. CasADi is an open-source software tool for numerical optimization and optimal control. Interior Point Optimizer (IPOPT) was used as a solver for the NLP problems. IPOPT is a software package for large-scale nonlinear optimization. The simulation time was set to $20s$ with a sampling time of $0.2s$, which gives a total of 100 iterations. Control variable $u$ was constrained between 0 and 1. The MPC was made inside a for loop where IPOPT was used to solve a new NLP problem for each iteration. A block diagram of the MPC implementation can be seen in Figure 3.5. The fundamentals of the program were based on a CasADi workshop [11].

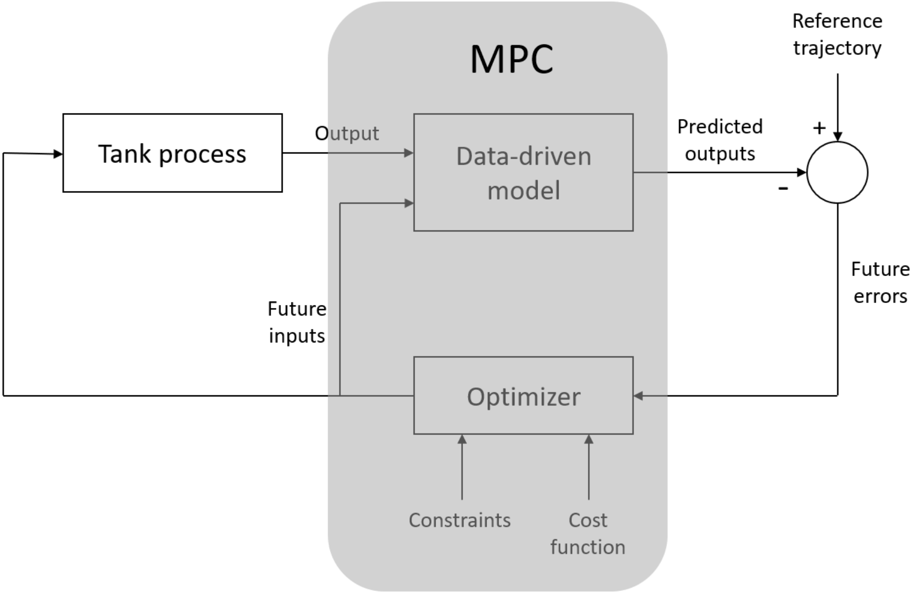


Figure 3.5: Block diagram for MPC.

The control variable $u$ controls the drain of the tank since the valve is connected on the output of the tank. If the tank are in a filling process, it will give a control variable $u = 0$. This means that the optimal control is obtained when the water level is decreasing.

# Chapter 4

# Simulation and results

This chapter explains how the simulations were conducted and show the results. The chapter is, as the other chapters, divided into two parts; one for the data-driven model and one for the MPC.

## 4.1 Data-driven model

The first part of the simulation was to approve the data-driven model created in Section 3.2. It was expected that some information was lost in the conversion between Spyder and Matlab. A Simulink program was made to compare the data-driven model with the physics-based model. They received the same random control variable $u$ and a start height $h_0$. The rate of the height $\dot{h}$ were plotted against each other. We can see from Figure 4.1 that there is way more deviation than the trained network in Section 3.2. The model is still accurate enough to be used as process model for the MPC.

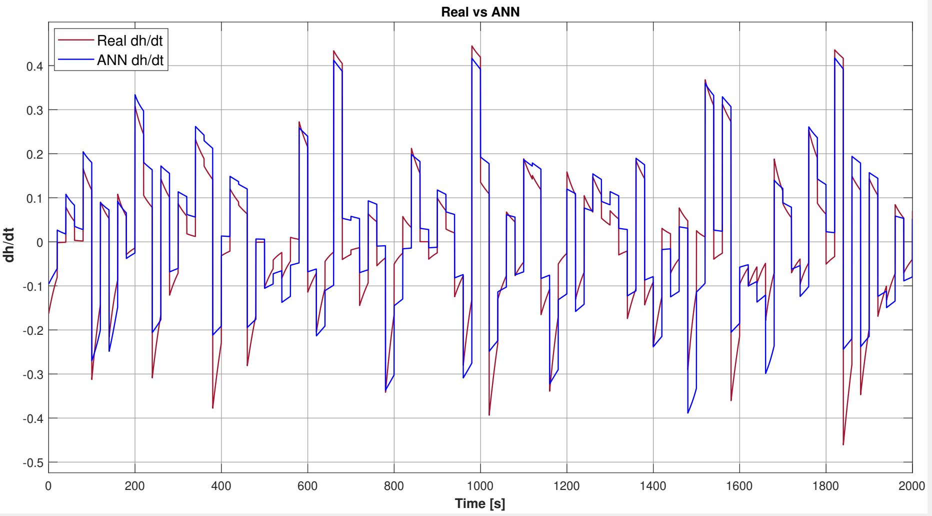


Figure 4.1: Data-driven model compared to real model.

## 4.2 MPC

A draining process was simulated on the tank from Section 3.1. The data-driven model was not as precise as expected, but simulation showed that it worked well as a process model for the MPC. A simulation with initial condition $h_0 = 10$ and reference value $h = 5$ was conducted. Different prediction horizons were used to see how the MPC responded. An MPC with the physics-based system as reference was also plotted in comparison. The draining process was simulated for $10s$, with three different prediction horizons, $N = 2$, $N = 5$ and $N = 20$.

In Figure 4.2 a prediction horizon $N = 2$ was used. We can see that the data-driven model yields less aggressive control, with smaller steps $\Delta u$ on the control variable $u$. The two models reache steady-state at the same time, after $7s$.

In Figure 4.3 a prediction horizon $N = 5$ was utilized in the MPC. The control responses were similar to each other, and the systems reached steady-state after about $3.5s$. If we look closely, we can see that there is a small steady-state error in the height $h$ for the data-driven model.

At last, a simulation with a prediction horizon of $N = 20$ was utilized, which can be seen in Figure 4.4. Here, the deviation between the heights is larger than the previous simulations. The two control variables $u$ are much alike.

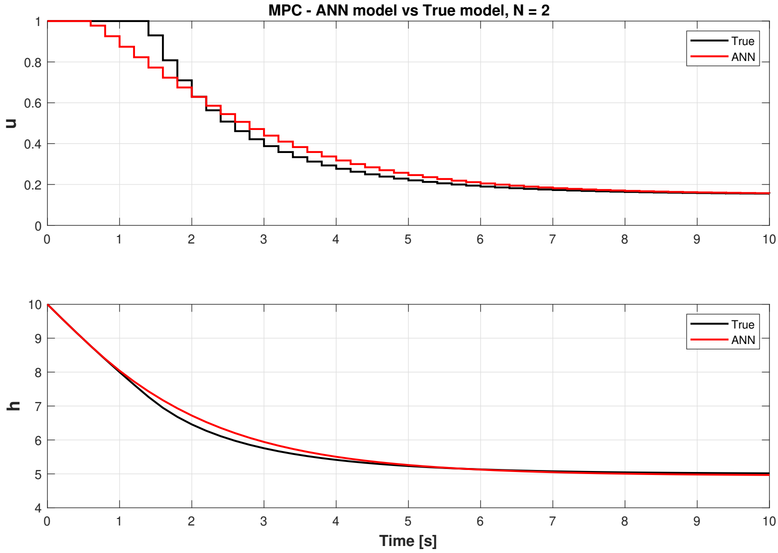


Figure 4.2: Simulation of tank draining with $N = 2$.

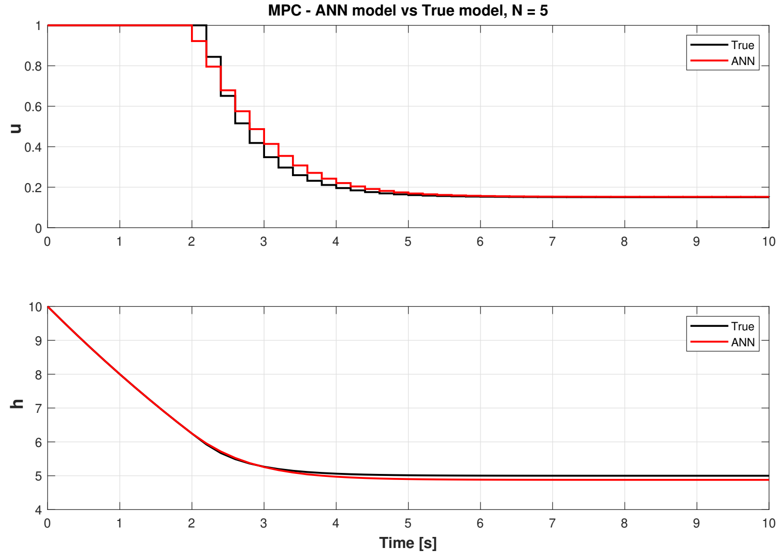


Figure 4.3: Simulation of tank draining with $N = 5$.

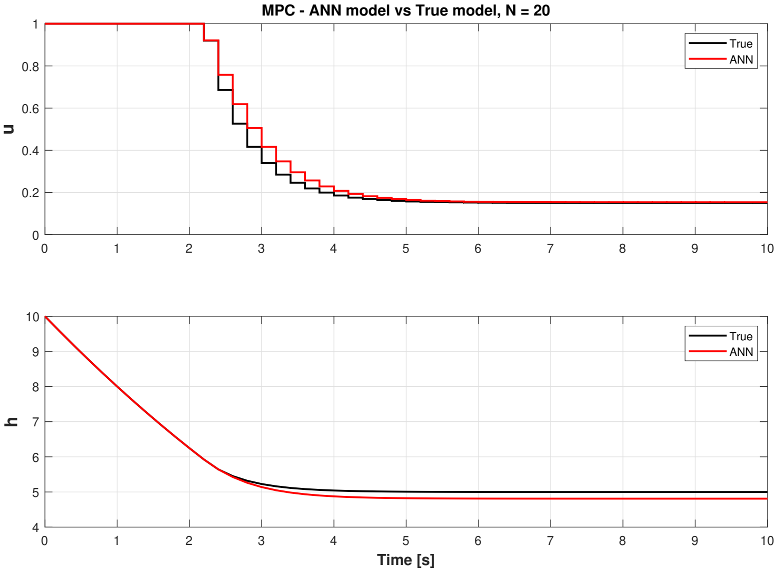


Figure 4.4: Simulation of tank draining with $N = 20$.

# Chapter 5

# Discussion

In this work, a nonlinear tank model was designed in Matlab. The nonlinear tank model was a case study of a gravity separator. A simulator developed in Matlab made it possible to test and compare different models. The simulator was modular and easy to modify.

AANs were used to create a nonlinear data-driven model for the tank model. Simulations in Section 4.1 showed that there was a deviation from the physics-based model. However, it was still interesting to see how the imperfect model performed as a process model for the MPC. The deviation between data-driven and physics-based models can occur in multiple forms. The transition from Spyder to Matlab could have led to information loss. Using the same framework for the ANN and the MPC would have been advantageous. Also, overfitting could have played a role for the deviation. Overfitting happens when the neural network is closely fitted to the training set. It makes it difficult to generalize and make predictions for new data.

The development of the model predictive control was done in Matlab with CasADi as a tool for nonlinear optimization. It worked sufficiently with both the physics-based and data-driven models. The single shooting method was used to transform the OCP into NLP problems. Single shooting was intuitive for this study, but the method is not recommended for systems with multiple control variables and large prediction horizons because of its high computational cost.

A steady-state error on the height $h$ occurred in the simulation when the prediction horizon increased. This is most likely a result of the deviation in the data-driven model. Since the error is growing as the prediction horizon is increasing, it can indicate that there is a deviation in one of the variables in the cost function.

The nonlinear tank model in this case study was a little too simplistic for validation of the quality of the obtained model. Adding more tanks dependent of each other could have increased the complexity of the system leading to a more qualified result. This study showed however that MPC is suitable for data-driven models and that neural networks are capable of obtaining data-driven models without priori knowledge.

# Chapter 6

# Conclusion

This project investigated how model predictive control could be used to yield optimal control on data-driven models. The motivation was to control real-world systems like the gravity separator, which are hard to both model and control by using data-driven models instead of on-line data.

The tank simulator in Matlab worked sufficiently for this project, and are scalable for future work. The artificial neural network managed to find a data-driven model, but combining two different frameworks weakened the model. Matlab combined with CasADi was suitable for nonlinear optimization. Deviation on the data-driven model can lead to a steady-state error because the minimization of the cost function uses deviated values. The results from this study has shown that MPC is promising to yield optimal control for a data-driven model of a gravity separator.

## 6.1 Future work

This project has been based on assumptions and simplifications to narrow the scope of the task. This section will present proposals on improvements and recommended future work to increase the quality and achieve more realistic results.

### 6.1.1 Gravity separator

The systems' complexity will increase when the three-phase gravity separator is implemented. This will lead to more qualified results. It will also require more computational cost. Using *multiple shooting* in the transformation between OCP to NLPs would be convenient. When noise is introduced, it is important to take the noise-sensitive variable $\dot{h}$ in account. Discretizing a noise-sensitive variable without noise-filtering will give an inoperable discrete-time system.

### 6.1.2 Neural networks

ANN fits the simplistic system well, but it will be necessary to focus on computational cost when complexity is increasing and several state variables are introduced. Other neural network structures with less computational cost should be considered.

# References

[1] E. F. Camacho and C. Bordons. *Model Predictive Control.* Springer, 2007.

[2] C. Chen. *Linear System Theory and Design.* Oxford University Press, 1999.

[3] B. Foss and T. A. N. Heirung. Merging optimization and control. 2016.

[4] C. Gershenson. Artificial neural networks for beginners. 09 2003.

[5] S. Gros and M. Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 2019.

[6] Z.-S. Hou and Z. Wang. From model-based control to data-driven control: Survey, classification and perspective. *Information Sciences*, 235:3–35, 06 2013.

[7] L. Imsland. Introduction to model predictive control, 2007.

[8] J. P. Joranou. Echo state networks for online learning control and mpc of unknown dynamic systems: Applications in the control of oil wells. 2019.

[9] S. Katz. Introduction to neural networks for senior design. University Lecture, 2004.

[10] K. Maladkar. Types of activation functions in neural networks and rationale behind it. Analyticsindiamag, 2018.

[11] M. W. Mehrez. Optimization based solutions for control and state estimation in dynamical systems, a workshop. 2019.

[12] K. R. Muske and J. B. Rawlings. Model predicitve control with linear models. 1993.

[13] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 1999.

[14] U. Rosolia, X. Zhang, and F. Borrelli. Data-driven predictive control for autonomous systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):259–286, 2018.

[15] K. Shiruru. An introduction to artificial neural network. *International Journal of Advance Research and Innovative Ideas in Education*, 1:27–30, 09 2016.

[16] A. S. Walia. Activation functions and it's types-which is better?, 2017.