CRANFIELD UNIVERSITY

FRANCISCO AURA CAMARENA

USING MACHINE LEARNING TO CONTROL CHEMICAL REACTORS

SWEE
Advanced Mechanical Engineering

MSc
Academic Year: 2018 - 2019

Supervisor: Dr Peter Clough
Associate Supervisor: Dr Stuart Wagland
September 2019

CRANFIELD UNIVERSITY

SWEE
Advanced Mechanical Engineering

MSc

Academic Year 2018 - 2019

FRANCISCO AURA CAMARENA

USING MACHINE LEARNING TO CONTROL CHEMICAL REACTORS

Supervisor: Dr Peter Clough
Associate Supervisor: Dr Stuart Wagland
September 2019

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc Advanced Mechanical Engineering
(NB. This section can be removed if the award of the degree is based solely on examination of the thesis)

ABSTRACT

Controlling reactor temperatures is critical to ensure chemical reactions progress and complete to the planned extent. Currently, the industry standard is to utilise PID controllers to control operating conditions in chemical reactors. A newer approach is to utilise machine learning – based algorithms such as neural networks (NN) to perform controlling duties.

In this project, PID control is implemented in a reactor and NN control is aimed to be implemented. A NN model of a reactor is trained with sights to design a NN controller and evaluate its performance on a temperature-only control process by comparting it to PID control.

The degrees of freedom are limited by simplifying the process within the reactor to a constant air flow and an on/off heating system. DAQ, control action and signal transferring take place in a computer (developed LabVIEW^{TM} programme). Control is designed to follow a reference signal. The controlled variable is the reactor's inner temperature and the control variable is the on/off state of the heating system.

PD (Proportional – Derivative) control is enabled within a feedback. Reasonable control action is achieved but performance could be improved with better PID tuning and introduction of integral action.

A NN model is designed with sights to train a second neural network (controller) that can later be utilised to perform control action on the real system. The model is trained through back-propagation as a feed-forward NN with discretized training data and later, closed to create a recurrent NN that models the reactor with only the control signal as input. The model achieves an R^2 fitting of 97 % to the validation data set but the presence of noise waves in the prediction prevents it from being suitable to train the NN controller. Future work would include training the NN controller and compare both methods.

Keywords:

Neural network, control, pid, system modelling, controller, backpropagation

ACKNOWLEDGEMENTS

From a personal point of view, this project has meant the confirmation of all the effort and work I have put in becoming an Industrial Engineer. My engineering studies have taken me from Polytechnic University of Valencia (UPV) to Cranfield University. The Erasmus programme made it possible and the departmental funding programme, a reality. Thank you Cranfield for this opportunity as I feel this year has really made an impact in my academic learning, skills and self-development.

I would like to thank my supervisor, Dr Peter Clough for all his support, patience and dedication to the guidance of this project. I feel I was always in control of my own work while knowing that I could turn to him for help whenever I needed. Thank you for enhancing my self-development and taking the experience in Cranfield University to a new level. In addition, I would like to thank Howard and Robert, both working in the lab, for their assistance in the project, effort and time.

I only have kind words for that special people from all over the world that I have met this course: friends and Cranfield Wolves handball team. You all made me feel like at home during my time here.

Finally, and most importantly, thank you to my family and loved ones. Thanks Ana simply for being there. Thanks GSD for being what you are. Thanks dad for planting the engineering seed and being a mind-set reference. Thanks mom for forging that personality and providing harsh doses of common sense, reality and truth. Thanks both for withstanding me all these years: I hope this is just the beginning of many great achievements.

Human life stands as a continuous exploring experience. Rise and shine. Run or die. Make adventure your grind: go out and pick up your feet.

TABLE OF CONTENTS

ABSTRACT i
ACKNOWLEDGEMENTS ii
LIST OF FIGURES v
LIST OF TABLES vii
LIST OF EQUATIONS viii
LIST OF ABBREVIATIONS ix
1 INTRODUCTION 1
1.1 Aims and Objectives 2
1.2 Thesis Structure 3
2 LITERATURE REVIEW 4
2.1 Control problem 4
2.2 PID Control 4
2.2.1 Control structure 5
2.3 Artificial Neural Networks (ANN) 5
2.3.1 Concept 6
2.3.2 Learning: objective function 7
2.3.3 Size 7
2.3.4 Control structure 8
2.3.5 System modelling with neural networks 10
3 METHODOLOGY 13
3.1 Set-up 13
3.1.1 Components 13
3.1.2 Airflow 15
3.1.3 Heating system 15
3.1.4 Control action: LabVIEW™ programme 15
3.2 PID control 18
3.2.1 Methodology 18
3.2.2 Structure 19
3.2.3 Control signal 19
3.2.4 Control programme 20
3.3 Neural network control 21
3.3.1 Methodology 21
3.3.2 Structure 22
3.3.3 System modelling 22
3.3.4 Neural network controller 27
3.3.5 Control programme 27
3.4 Reference signal 28
3.5 Fitting and performance 28
4 RESULTS & DISCUSSION 29
4.1 Training signal 29

4.2 Validation signal 31
4.3 Training & validation data size 32
4.4 PID Control 33
4.4.1 Control programme 33
4.4.2 PID Parameters 35
4.4.3 PID control results 39
4.5 Neural network control 41
4.5.1 System modelling 41
4.5.2 NN controller and control 53
5 CONCLUSION 55
REFERENCES 60
APPENDICES 63
Appendix A LabVIEW™ programme 63
Appendix B PID Tuning – System Identification 69
Appendix C MATLAB Code 71

LIST OF FIGURES

Figure 2-1. Feedback control loop. 5

Figure 2-2. Neural Network architecture [11]. 6

Figure 2-3. General learning architecture, also known as inverse modelling [14]. 8

Figure 2-4. Specialized learning architecture [14]. 9

Figure 2-5. Forward modelling scheme [9]. 10

Figure 2-6. NN training strategy. 11

Figure 3-1. Diagram of the laboratory set-up. 13

Figure 3-2. Laboratory set-up. 13

Figure 3-3. Reactor: air inlet (right), air outlet (left). 15

Figure 3-4. Programme flow during operation. 16

Figure 3-5. Data acquisition part of the programme. 17

Figure 3-6. Data transmission part of the programme. 18

Figure 3-7. PID Control loop structure. 19

Figure 3-8. (PID programme) Main panel. 20

Figure 3-9. (PID programme) Control action loop. 21

Figure 3-10. NN control structure with a NN controller and a NN model for training the controller. 22

Figure 3-11. Programmed inner temperature thresholds to control input signal (heat on – off) during training signal acquisition. 23

Figure 3-12. Programmed inner temperature thresholds to control input signal (heat on – off) during validation signal acquisition. 24

Figure 3-13. FANN (Open loop) with one hidden layer of one neuron. 25

Figure 3-14. RNN (Open loop) with one hidden layer of one neuron. 26

Figure 3-15. (NN programme) Main panel. 27

Figure 3-16. (NN programme) Control action loop. 28

Figure 4-1. Training signal: temperature inside. 30

Figure 4-2. Training signal section: ON until 75 °C. 31

Figure 4-3. Validation signal: temperature inside. 32

Figure 4-4. Noise present in training data..... 33

Figure 4-5. Models' responses when simulated with the regular training data set. 37

Figure 4-6. Models' responses when simulated with the shifted training data set. 38

Figure 4-7. Results of PID control to follow a reference signal with a set point of 100 °C..... 40

Figure 4-8. NN Model performance in closed loop on the training data set. 50

Figure 4-9. NN Model performance in closed loop on the validation 2 data set and zoom on the noise from the predicted output..... 51

Figure 4-10. Ripples within model response and filtered signal..... 52

Figure 4-11. NN Model behaviour after filtering the predicted output. 52

Figure A-1. (PID programme) Tab 0: File creation. 63

Figure A-2. (PID programme) Tab 1: File structuring. 64

Figure A-3. (PID programme) Temperature selection..... 65

Figure A-4. (PID programme) Data saving in correspondent files. 65

Figure A-5. (NN programme) Tab 0: File creation. 66

Figure A-6. (NN programme) Tab 1: File structuring. 67

Figure A-7. (NN programme) Data saving in correspondent files. 68

Figure B-1. Models' responses when simulated with the regular validation 1 data set..... 69

Figure B-2. Models' responses when simulated with the regular validation 2 data set..... 69

Figure B-3. Models' responses when simulated with the shifted validation 1 data set..... 70

Figure B-4. Models' responses when simulated with the shifted validation 2 data set..... 70

LIST OF TABLES

Table 2-1. Legend. 10

Table 2-2. Model fitting (%) to the real system's output [17]. 12

Table 3-1. USB-1208Fs-Plus relevant characteristics. 14

Table 3-2. D2425 relay relevant characteristics. 14

Table 3-3. Different relevant time periods to the programme. 16

Table 3-4. PID output vs. time ON in a PWM signal of period 2 seconds. 20

Table 4-1. Sizes of data sets for model training and validation (dt = 1s). 32

Table 4-2. Parameters of the models considered. 36

Table 4-3. NRMSE fitting (%) to the Regular data by both models. 37

Table 4-4. NRMSE fitting (%) to the Shifted data by both models. 38

Table 4-5. Predicted response of the system under PID control. 39

Table 4-6. Sizes of data sets for model training and validation (dt = 10s). 42

Table 4-7. R^2 fitting of predicted output to target data by the neural network ten second model under different training conditions. 45

Table 4-8. Training parameters for the four cases during ten second open loop training. 46

Table 4-9. R^2 fitting of predicted output to target data by the neural network one second model under different training conditions. 48

Table 4-10. Training parameters for the four cases during one second open loop training. 48

LIST OF EQUATIONS

(2-1)..... 11
(2-2)..... 11
(3-1)..... 25
(3-2)..... 28
(3-3)..... 28
(4-1)..... 36
(4-2)..... 50

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
FANN	Feedforward Artificial Neural Network
MSE	Mean Squared Error
NN	Neural Network
NRMSE	Normalized Root Mean-Square Error
RDNN	Recurrent Dynamic Neural Network
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
R^2	R - Squared

1 INTRODUCTION

Chemical reactions take place in chemical reactors as part of industrial processes. Within an industrial environment, the aim is to complete these processes efficiently and within cost. Controlling reactor temperatures, pressures and flow rates is utterly critical to ensure chemical reactions progress at the planned kinetic rate and complete to the planned extent.

Control systems engineering is linked to not only the control action but also to the knowledge of the controlled processes themselves. An extended knowledge of a system is usually required to control it appropriately and therefore system modelling becomes relevant to the subject. System models are mathematical expressions or representations of the real systems.

In the field of chemical engineering several control options are nowadays available to control chemical reactors. Among these, PID controllers seem to arise as the most used option due to their simplicity, ability to maintain steady state operations [1] and their proved functionality. They operate within a feedback loop where the controlled variable is fed back to the controller enabling a continuous reading of the state of the system. PID controllers rely on computing the proportional, integral and derivative from the error signal between the current state of the controlled variable and the reference signal [2].

The emerging field of machine learning is based on applying artificial intelligence to the automation of data analysis. Artificial neural networks are machine learning – based algorithms that aim to resemble the structure and operation of biological neural networks. Neural networks are known to be able learn to map functions based on input and output data including non-linear behaviours [3]. They have also been considered to be fault-tolerant when learning from data and insensitivity to noise [4]. These abilities is what positions them to be excellent system modellers given the right inputs and outputs as training data.

They have experienced an important research evolution in the past two decades and have been heavily linked to control systems due to the mention modelling capabilities, high parallel processing [3], self-learning abilities and ability to work

within a control structure. The development of computers, in terms of computing power and cost, and the extended knowledge available on artificial neural networks based on previous research invite to implement them in industrial applications in control systems.

In this project, a reactor has been simplified to a temperature-only control process limiting its degrees of freedom. Temperature control was enabled by operating on a heating system. The variables involved were:

- Manipulated variable: heating system on/off.
- Controlled variable: temperature inside the reactor.
- Reference: desired temperature inside the reactor.

Based on the previous, the aim of the control problem was to control the temperature inside the reactor (driving it to match the reference signal) by manipulating the heating system.

Two control structures were aimed to be implemented and tested for comparison. On the one hand, a PID controller operating within a feed-back temperature control loop. On the other, an artificial neural network addressing controlling duties. These control actions were aimed to be tested while performing temperature control in the reactor following a reference signal.

Finally, neural network results were aimed to be compared with traditional PID loops results. In the end, PID control was enabled but neural network control was not due to lack of time and unfeasible neural network modelling. This is all discussed in results & discussion section.

1.1 Aims and Objectives

The main aim of this project was to implement the temperature control of a simplified chemical reactor utilising machine learning techniques (artificial neural networks) to enable a self-learning control method and evaluate its performance by comparing it with traditional PID control.

This was to be fulfilled by aiming to achieve the following objectives:

- To implement a temperature-only control process in a simplified reactor with a feed-back control loop and a PID controller.
- To tune the PID parameters and optimise the control action according to the standard control in the university's laboratories.
- To implement a temperature-only control process in a simplified reactor utilising artificial neural networks.
- To enable control by tracking a reference signal (set points of temperature) in a range from 30 to 200 degrees Celsius.
- To generate datasets of temperature points during the control actions of both methods.
- To compare both control methods and offer an insight into the performance of the machine learning technique utilised.

1.2 Thesis Structure

This thesis is divided into 5 chapters. Chapter 1 contains the introduction, aims and objectives and description of the project. Chapter 2 contains the literature review on the subjects debated in this thesis. Chapter 3 includes the methodology related to the PID control and the neural network control. Chapter 4 presents together the results and discussion. Finally, chapter 5 contains the conclusion.

2 LITERATURE REVIEW

2.1 Control problem

As expressed by Narendra in [4], “the objective of control is to influence the behaviour of dynamical systems” by “maintaining the outputs of systems at constant values (regulation) or forcing them to follow prescribed time functions (tracking)”. For this purpose, PID control has been around since early stages and new approaches are arising based on machine learning. Machine learning techniques have been applied to control engineering and along this literature review several applications will be referenced. Their irruption in control systems covers from defining the control structure based completely on machine learning to modelling the system with artificial neural networks.

Furthermore, researchers have compared the performance of both methods. Khalid and Omatu [3] compared a neural network controller with a PI controller (no derivative action) and showed the potential of neural networks for a faster control action and broader range of operation in the system.

2.2 PID Control

PID control is “the most used industrial control method owing to its simplicity and ease of use” [2]. Being utilised even in simple applications [5], they have been well studied and information on PID control is broadly available in textbooks like [1] and [6].

The challenge when it comes to PID control is tuning the PID parameters. The performance of the control action will depend on these parameters as they are themselves dependent on the controlled system. The perfect PID tuning method is yet to be defined and the field has developed into a “fit for purpose” approach where PID parameters are defined based on the fact that they enable a good-enough control action. Several methods have been proposed and information can be found in [2] and [7]. Machine learning techniques have also been applied to tuning PID parameters. Zulu [2] proposes an approach in which a machine learning algorithm is used to compute the most optimal PID parameters by learning from a regression model of the plant.

The fact that PID controllers are widely used motivated the selection of this control method for the comparison of control performance in this project. A challenge was the nature of the manipulated variable itself. While the manipulated variable ranged on an ON/OFF state, PIDs output a continuous control signal and the PID output needed to be adapted.

2.2.1 Control structure

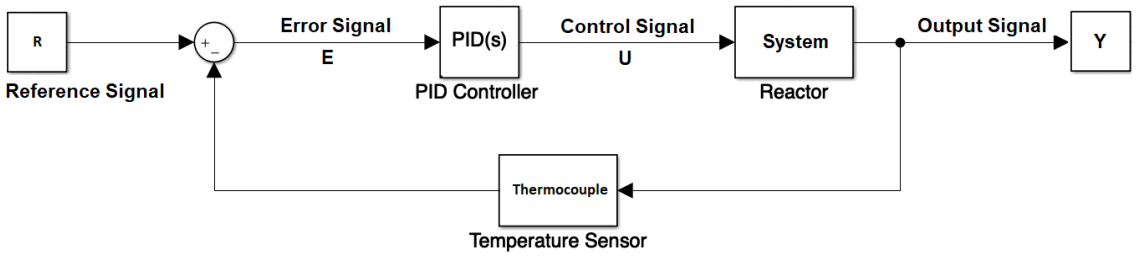


Figure 2-1. Feedback control loop.

PID controllers operate by computing the error between the reference signal and the current state of the controlled variable [2]. To enable this, a reading of the controlled variable is fed back to the beginning of the control loop and ultimately, to the PID controller. From here, machine learning techniques such as neural networks can be applied directly to this control scheme and substitute the PID controller.

2.3 Artificial Neural Networks (ANN)

Artificial neural networks, generally known as “neural networks” (NN), have been studied extensively during the las two decades and there is a broad range of literature available to understand their behaviour. For a general understanding of the subject the lector is referred to the textbooks [8] and [9] and for a review on the las two decades on research, information can be found in [4] , [10], [11], [12] and [13].

Only relevant information to this project is displayed in this literature review: a brief introduction to neural networks (concept and learning procedure), their embedment in control structures and use in system modelling. Also, neural

network control applications are referred across the section when explaining the different points.

2.3.1 Concept

A neural network, is essentially a neural structure composed of a series of computing units, known as neurons, arranged in layers. Their structure can be summarised in an input layer (neurons matching the number of inputs), an output layer (neurons matching the number of outputs) and one-to-several middle layers (hidden layers). A multi-layered NN is that one which has at least three layers: input, hidden and output layer. Neurons are linked between themselves through weighted connections that relate them to each other and allow to map the inputs to the outputs through the mesh.

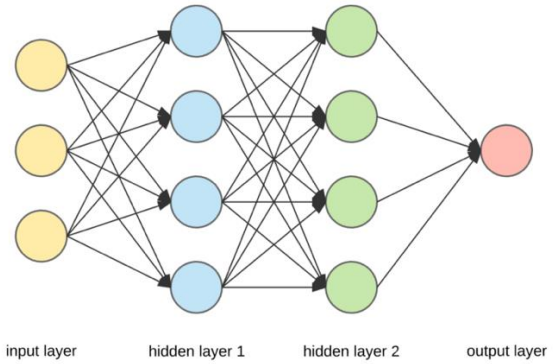


Figure 2-2. Neural Network architecture [14]**.**

Two types of NN are considered in this literature review: feedforward artificial neural networks (FANN) and recurrent neural networks (RNN). FANN only have forward connections and there is no feedback (*Figure 2-2*) while RNN have feedback connections within the network and neurons' outputs from later layers can be inputs of neurons located at early layers.

A NN can be understood as a black box that when presented with an input provides an output. The general idea is to let the NN learn the appropriate relations between inputs and outputs to adjust its behaviour by providing training examples.

2.3.2 Learning: objective function

During a learning phase the NN is presented with a set of training data containing related input and output values. During this phase, the weights of the connections within the network are updated iteratively through a learning algorithm. The training is completed when the NN is capable of mapping the training data accordingly.

Learning algorithms aim to minimize an objective function during the training phase. That function is usually an error function such as MSE (between predicted output of the NN and desired output) or some sort of “cost function” that ensures better performance of the network. In [15], Nguyen and Bernard train a neural network controller that drives a truck both by minimizing the RMSE error function of the final position and by minimizing an objective function that includes the path taken to reach that final position.

Neural networks are generally trained by the backpropagation algorithm [16]. The algorithm works updating the weights of the neural network connections by propagating the error backwards all the way to the input. This is done so by computing the partial derivative of the error function with respect to each weighted connection. Also, modifications of the backpropagation algorithms are used in literature to adapt it to different situations. For example, by computing the error at the output of a model and not the NN itself [17].

2.3.3 Size

From previous work, it is widely accepted that multi-layered NN with at least two hidden layers can map any nonlinear function given enough neurons in the first layer. Nonetheless, higher number of layers are being used because it is believed to improve NN’s performance by improving robustness, generalisation ability and convergence properties [15]. Khalid and Omatu [3] simply try several sizes and pick the NN they consider has the best behaviour. This constitutes a feasible solution in a time where computers can deal with NN training in reasonable amounts of time and was applied in this project.

2.3.4 Control structure

Psaltis et al. [17] propose the implementation of a NN as a feedforward controller as opposed to a feedback controller (PID). Two feasible control architectures are proposed: general and specialized learning.

2.3.4.1 General learning architecture

It was also implemented by Khalid and Omatu [3]. A random input signal “u” (manipulated variable) is applied to the system (plant) to produce variations in the output “y”. The NN controller is trained by reverse-learning the behaviour of the system. The output of the system corresponds to the input of the NN and the output of the NN is checked against the input of the system for training purposes. This is also known as inverse modelling [10]. When the NN is trained, it is used as a feedforward controller placed in front of the plant receiving as an input a reference signal and outputting a control signal that will drive the system to the desired output.

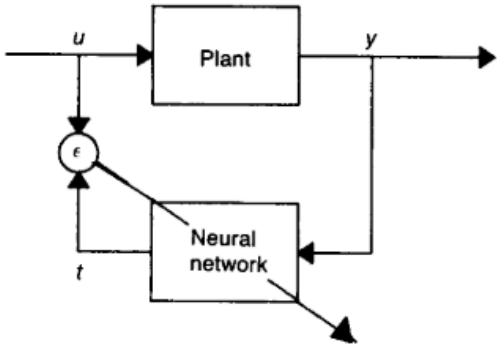


Figure 2-3. General learning architecture, also known as inverse modelling [17]**.**

This method is not goal-oriented as the training signal “u” is a random input and is not focused on the regions of interest of the controlled variable “y”. To obtain good performance on regions of interest, the whole range of possible outputs should be covered during training data acquisition by ensuring the control signal “u” covers the whole operational range of the system. It has as a main advantage that no previous knowledge regarding the system’s behaviour and dynamics is required.

2.3.4.2 Specialized learning architecture

It was also implemented by Narendra and Parthasarathy in [18].

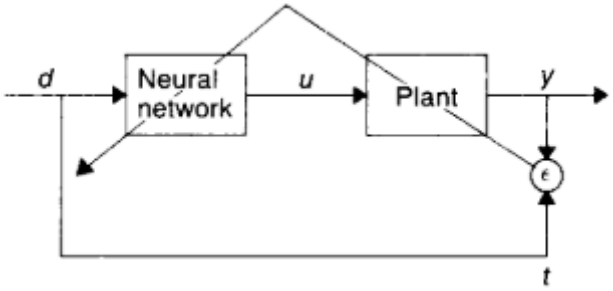


Figure 2-4. Specialized learning architecture [17]**.**

This method specializes training by aiming to train the NN controller in regions of interest providing directly the reference signal “d”. During NN training, knowledge of the system’s (plant) model is required. In order to back-propagate the error from the output of the system to the input, the system itself needs to be mathematically derivable in order to compute the derivatives during back-iteration. A real system would not allow a training algorithm to mathematically compute error through it and therefore this method requires knowledge of the plant to create a mathematical model.

In order to deal with this situation, Nguyen and Widrow [15] approached the problem by modelling the system with a second NN. They propose a feed-forward neural network controller capable of controlling a truck during backing manoeuvres. In this case, two FNN are utilised in the control structure. First, a FNN (known as the emulator) is used to model the system by learning its behaviour across a different range of manoeuvres. This is known as forward modelling [18]. Then, a second FNN (the controller) is trained by using the model (first FNN – emulator) as a predictor of the truck’s behaviour. Finally, once the controller is trained, it is directly utilised to control the real truck.

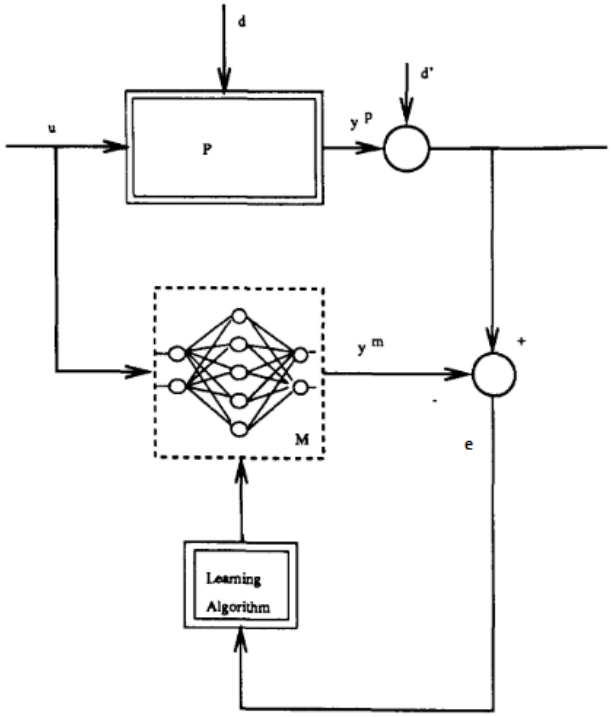


Table 2-1. Legend.

P	Plant / system
U	Input
Y^P	Output plant
Y^m	Output model
d	Disturbances
e	Error

Figure 2-5. Forward modelling scheme [10]**.**

No model of the system is known before hand and everything is learned by the NNs. The controller is trainable due to the fact that the model is in itself a NN. This enables the back propagation of the error from the output of the model (system's predicted behaviour) to the input of the model which in turn is the output of the NN controller.

2.3.4.3 Combination

Psaltis et al. [17] also propose combining both methods during training to improve overall performance. The controller can be trained offline through general learning across the whole range of operation of the system and later, specialize-trained in regions of interest.

2.3.5 System modelling with neural networks

System modelling involves creating a model that can simulate the behaviour of the real system it represents. Neural networks have been directly utilised in the field of chemical engineering to model chemical processes [19]. They are fit for this purpose as they can map nonlinear functions accordingly [4], [8], [10].

One of the main concerns when it comes to capturing the behaviour of a system is its dynamics. The future state of a system may not only depend on its input but also on its current state. For instance, the future temperature of a tank receiving heat depends on the heat input and on the current temperature when the heat is applied. When it comes to processing temporal signals, the most used method is to treat the time domain like a discrete dataset [10].

2.3.5.1 Feedforward artificial neural networks (FANN)

In order to capture dynamic behaviour with a feedforward neural network Khalid and Omatu [3] propose feeding the NN with past states of the system, and even, past inputs. After training several NN with different combinations of previous states, they choose the one that matches the system's real behaviour better.

$$\begin{aligned} \text{Inputs} \quad & \mathrm{U}_{\mathrm{train}} = [u(k), y(k)] \\ \text{Outputs} \quad & \mathrm{Y}_{\mathrm{train}} = [y(k+1)] \end{aligned} \tag{2-1}$$

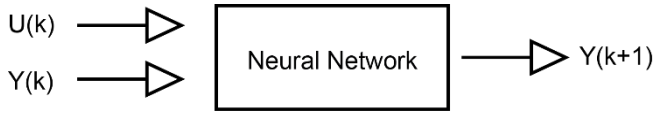


Figure 2-6. NN training strategy.

This approach is also used for modelling purposes and comparisons in other research [20]. In this case, the NN is fed with 4 inputs and a linked output as training data and validated via a recursive method during prediction.

$$\begin{aligned} \text{Inputs} \quad & \mathrm{U}_{\mathrm{train}} = [u(k-2), u(k-1), y(k-2), y(k-1)] \\ \text{Outputs} \quad & \mathrm{Y}_{\mathrm{train}} = [y(k)] \end{aligned} \tag{2-2}$$

The recursive verification method involves self-feeding predicted outputs of the NN as inputs in following iterations. These values need to be stored from iteration to iteration outside of the NN.

2.3.5.2 Recurrent dynamic neural networks (RDNN)

Shaw et al. [20] propose recurrent "dynamic" neural networks in system modelling. This approach improves mapping nonlinearities and dynamic behaviour. They compare the performance of RDNN to FANN when it comes to modelling the same dynamic system and prove that RDNN provide a better performance than FANN. Anyways, the performance of the FANN is not that far

from the real system's output under a random input signal and depending on the application could provide an accurate-enough model of the system (*Table 2-2*). It was not able to capture a significant non linearity under a certain input.

Table 2-2. Model fitting (%) to the real system's output [20]**.**

Case study	RDNN (%)	FANN (%)
1	98.91	88.47
2	92.65	70

Knowing the dynamic order of the process is relevant as the number of neurons required to map the process with a RDNN is linked to this parameter [20]. Hence, knowledge or analysis of the process is needed. Even though, given that the computational training time is reasonable, a simpler approach would be to train several RDNN (of different sizes) and use the one that models the system better. No knowledge of the system would then be needed.

3 METHODOLOGY

3.1 Set-up

The laboratory set-up is depicted in Figure 3-1. Data acquisition, control action and control signal transmission were carried out by the computer. The control signal was received by a solid-state relay that switched on/off a transformer. In turn, the transformer powered the heating system in the reactor.

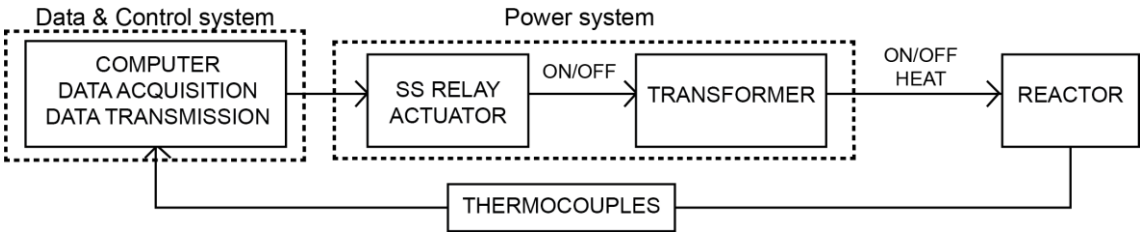


Figure 3-1. Diagram of the laboratory set-up.

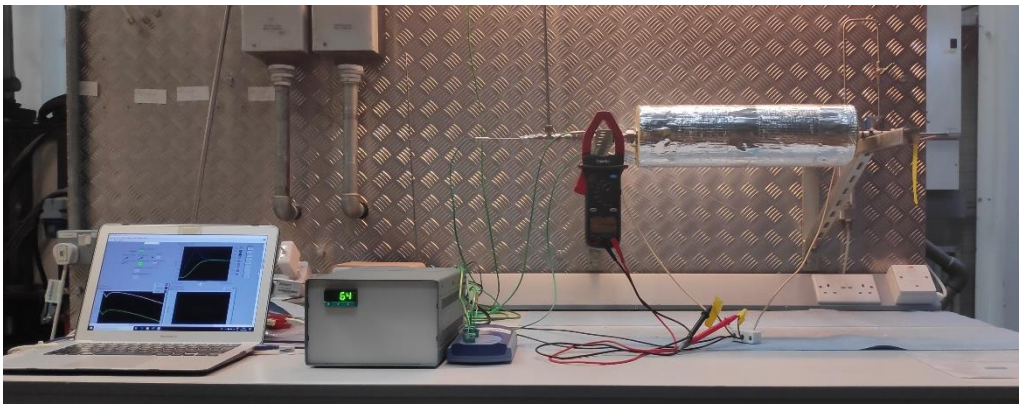


Figure 3-2. Laboratory set-up.

3.1.1 Components

3.1.1.1 DAQ – Data acquisition

The thermocouples' signal was acquired with the thermocouple data logger TC-08 from PICO Technology [21]. Drivers also available on [21].

It was connected to the computer. The communication PC – TC-08 was through a USB cable. The thermocouples were plugged to the TC-08 and the readings were taken through a LabVIEW™ programme in the computer.

3.1.1.2 Control signal transmission

The control signal was transmitted through an I/O device: the USB-1208Fs-Plus from MC Measurement Computing [22], [23].

It was connected to the computer. The communication PC – I/O device was through a USB cable. The signal transmission was done through an analog output between 0 and 4V from the I/O device. It was programmed in LabVIEW™.

Table 3-1. USB-1208Fs-Plus relevant characteristics.

Analog output range	0-5 VDC
Number of analog outputs	2
Communication PC - Device	Drivers [24] ULx for NI LabVIEW™ [25]

3.1.1.3 Power system: SS relay & transformer.

The solid state relay used was the D2425 from Crydom [26], [27].

The SS relay received the control signal from the I/O device and switched on/off the power arriving to the transformer. The communication I/O device – SS relay was enabled through copper cables. The relay could be activated with 4 VDC (ON value) from the control signal and it could handle the voltage from the mains at its output (*Table 3-2*).

Table 3-2. D2425 relay relevant characteristics.

Control voltage range	3-32 VDC
Operating voltage range	24-280 VAC

The transformer was a 230 VAC - 55 VAC.

3.1.1.4 Reactor

Figure 3-3. Reactor: air inlet (right), air outlet (left).

The reactor was simplified to an airflow with a heating system. It was covered by insulating super-wool and heating tape. Two thermocouples (red circle in Figure 3-3) were placed to take measures: one inside the reactor and one on the wall.

3.1.2 Airflow

The airflow was continuous during the operation of the reactor. It was quantified with a rotameter at the air inlet.

$$\text{Airflow:} \qquad 600 \quad \frac{cm^3}{min}$$

3.1.3 Heating system

The heat was provided by heating tape wrapped around the reactor acting as a resistor that would heat up when current was running trough. It was powered by AC current at 55V and the two only possible states were on and off.

3.1.4 Control action: LabVIEW™ programme

The control action took place in the computer. A LabVIEW programme was coded to acquire the signals from the thermocouples, compute the control action and pass on the control signal to the power system through the components detailed above. Two versions of the programme were created: one for PID control and another for machine learning-based control.

Only relevant parts of the programme are shown in this document. Snapshots of the rest can be seen in Appendix A LabVIEWTM programme.

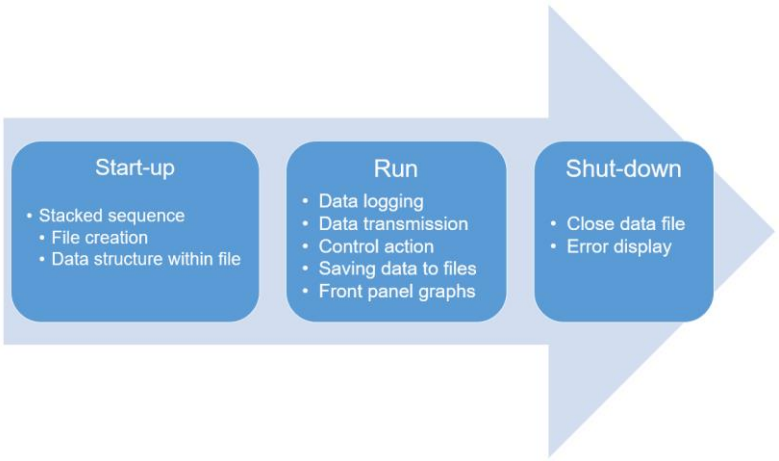


Figure 3-4. Programme flow during operation.

3.1.4.1 Start-up

Depending on the programme, up to three files were created: a file to save the control signal (“control file”), a file to save the temperature signals (“temperature file”) and a file to save the PID parameters (“pid file”). Files were formatted at this stage to receive the data later.

The PID control programme had the three files and NN control programme had only the control and temperature files.

3.1.4.2 Run

In this stage the programme ran on a continuous base until it was stopped. Data was acquired and saved, the control action took place and the control signal was transmitted periodically.

Table 3-3. Different relevant time periods to the programme.

Action	Period (ms)
Acquiring data	1000
Saving data	1000
Running control action	2000 (PID) 1000 (NN)
Transmitting signal	20

3.1.4.2.1 Data acquisition

Data acquisition was common to both the PID control and NN control programmes. It acquired the temperature value from all the channels of the TC-08 and later only the relevant information was kept.

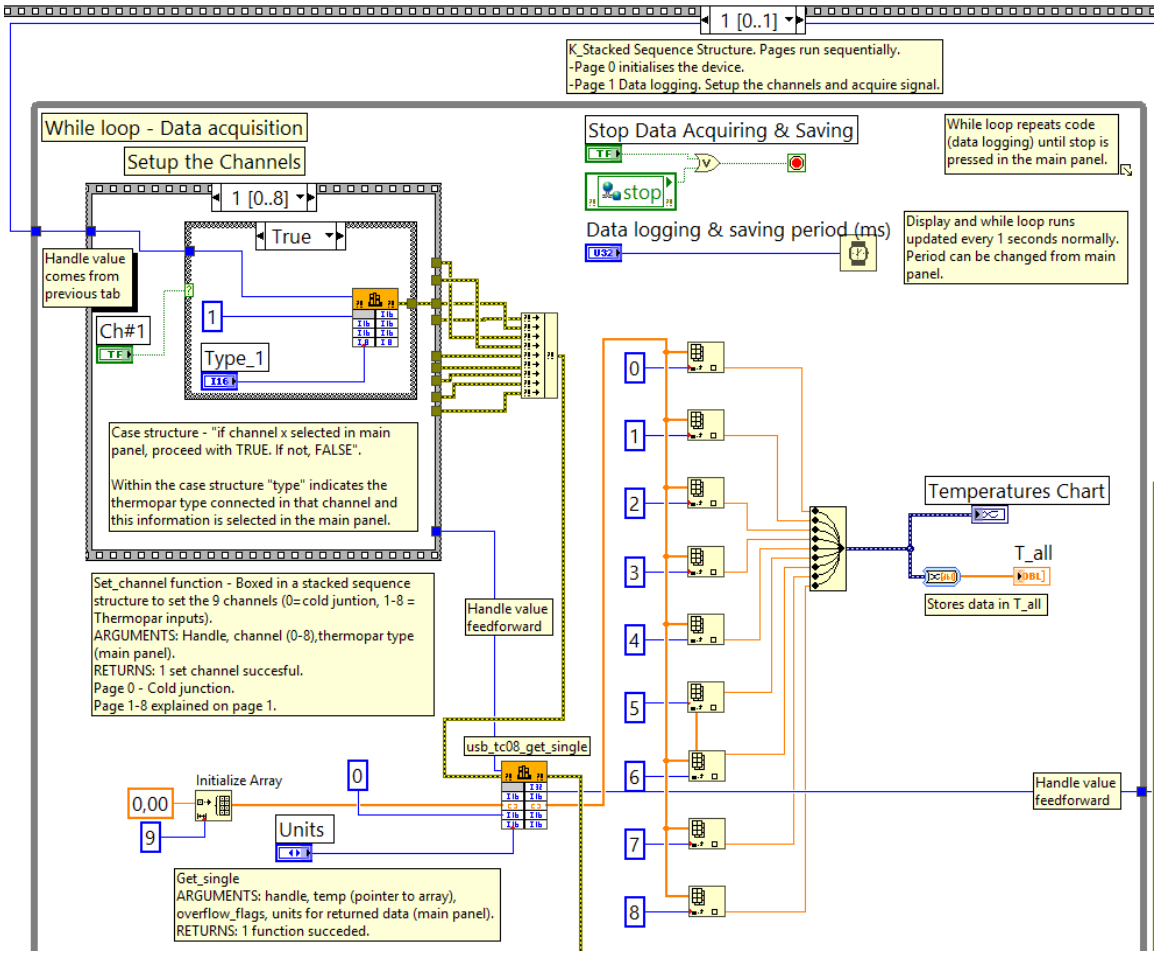


Figure 3-5. Data acquisition part of the programme.

3.1.4.2.2 Data transmission

Data transmission was common to both programmes. The PWM signal created during control (in another while loop) was sampled at 20 ms intervals (in the data transmission while loop) and transmitted to the SS Relay. This fast interval was selected to capture the PWM signal correctly.

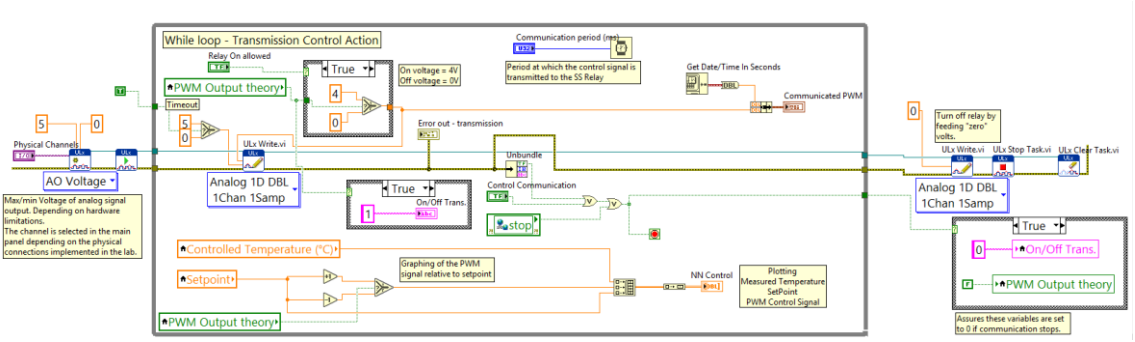


Figure 3-6. Data transmission part of the programme.

3.1.4.2.3 Control action

It is described in the present document in the correspondent section. For the PID control programme, it can be found in 3.2.4 Control program and for the NN control programme, in 3.3.5 Control programme.

3.1.4.3 Shut-down

During this stage the programme closed the files and showed errors in screen.

3.2 PID control

3.2.1 Methodology

The PID control was implemented as follows:

1. A LabVIEW™ programme was created to perform the controlling duties. It acquired the temperatures, processed the PID control action and transmitted the control signal to the heating system (*3.2.4 Control programme*).
2. The control system (computer with programme) was set into the laboratory set-up according to 3.1 Set-up.
3. PID was tuned. PID parameters were selected using the PID Tuner App in MATLAB by modelling the system with a linear model (*4.4.2 PID Parameters*).
4. PID control was enabled to follow a reference signal (*4.4.3 PID control results*).

3.2.2 Structure

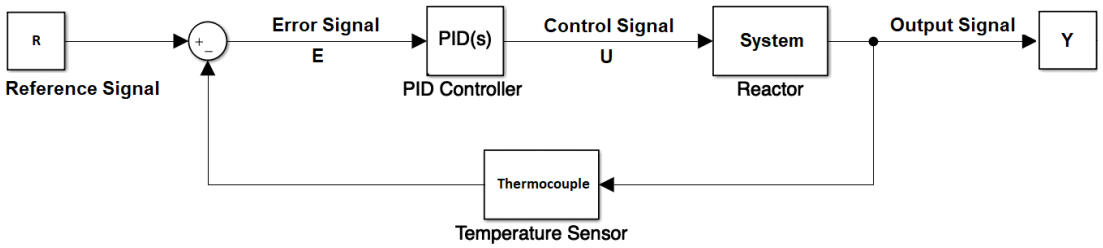


Figure 3-7. PID Control loop structure.

The PID control was implemented as depicted above, having control processing taking place in the computer. The reference signal was introduced by the user (or programmed) in the computer. As explained, the computer processed the signal from the thermocouples, computed the PID control action and transmitted the control signal to the power system.

3.2.3 Control signal

PIDs base control action in computing error along time and outputting a continuous control signal. The heating system only worked on an on/off basis. To deal with this situation, the PID was configured to output a continuous value between 0 and 100. This value corresponded to a percentage of activated time on a pulse-with-modulated (PWM) signal with a period of two seconds, which corresponded to the computing period of control action (*Table 3-3*). The PID output acted as the duty cycle parameter of the PWM signal.

One percent of 2 seconds is 20 milliseconds. This was too fast for the heating system to respond to the signal. Therefore, the output signal from the PID was discretized to 6 values (0, 20, 40, 60, 80 and 100% of duty cycle) by transforming the original signal (0 to 100) to the nearest value from the discretized scale. This way, the fastest interval “on-off” seen by the ss relay was 400ms (*Table 3-4*) which corresponded to a PID output of 20.

Table 3-4. PID output vs. time ON in a PWM signal of period 2 seconds.

PID Output (% Duty Cycle)	Time ON (ms)
0	0
20	400
40	800
60	1200
80	1600
100	2000

3.2.4 Control programme

This section includes a description of the most relevant aspects of the PID control programme. The rest is available in Appendix A LabVIEWTM programme. Computing control action took place every two seconds to update the duty cycle of the PWM signal that ran on the same period.

3.2.4.1 Main panel

Main panel allowed for controlling the set point, PID parameters and selecting the different channels from the thermocouple data logger that were relevant. For security reasons, an extra button was added to allow the relay to turn “on” the heating system.

Figure 3-8. (PID programme) Main panel.

3.2.4.2 Control action

Control action ran in a while loop every two seconds (*Table 3-3*) outputting a PWM signal out of the loop of period equal to those two seconds. The PWM signal was generated within the control loop in another while loop storing the on/off value every 10 milliseconds in a variable “PWM Output theory”. This secondary while loop read the outputted value of the PID controller (duty cycle) every time it completed a whole cycle (200 times 10 milliseconds to a total of 2 seconds).

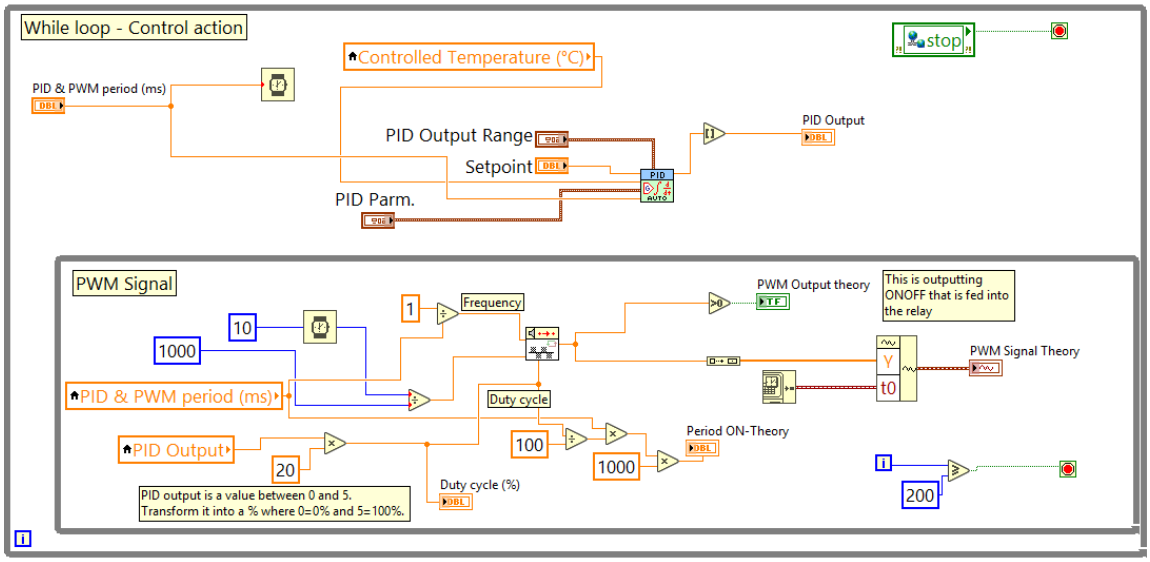


Figure 3-9. (PID programme) Control action loop.

3.3 Neural network control

3.3.1 Methodology

The neural network control was implemented as follows:

1. A LabVIEW™ programme was created to perform the controlling duties. It acquired the temperatures, aimed to process the NN control action and transmitted the control signal to the heating system (*3.3.5 Control programme*).
2. The system was modelled with a neural network: “NN Model” (*3.3.3 System modelling*).
    a. A training signal was created by operating the system.
    b. A validation signal was created by operating the system.

c. The system was modelled.
    i. NN model was trained with training data.
    ii. NN model was validated by evaluating performance with validation data.

3. A neural network controller was aimed to be designed: NN controller (*3.3.4 Neural network controller*).
4. The control system (computer with programme) was set into the laboratory set-up according to 3.1 Set-up.
5. NN control was aimed to be enabled to follow a reference signal.

3.3.2 Structure

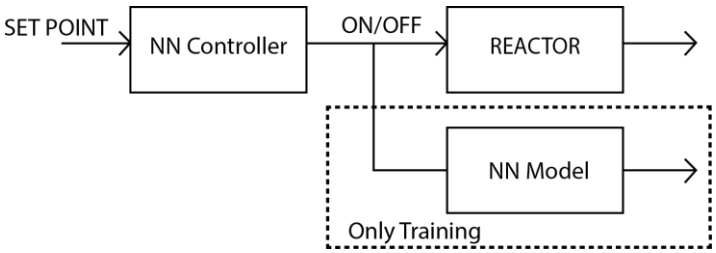


Figure 3-10. NN control structure with a NN controller and a NN model for training the controller.

The intent was to implement NN control as depicted above, having control processing taking place in the computer. The NN controller was to be trained utilising a NN model of the system in order to allow backpropagation of the error (temperature error – output of both reactor and NN model) through the NN model to the output of the NN controller.

3.3.3 System modelling

A neural network was trained to act as a model of the system. This neural network is known as “NN model” in this document. As a summary, a feed-forward neural network was trained in open loop. By closing the loop, the network became a recurrent neural network that was in itself the NN Model.

3.3.3.1 Training signal

Training data was required to train the NN model. The training data was obtained by applying a designed input signal (on-off action) to the system and recording the output (temperature variations) experienced. The methodology used was:

1. Design input signal to the reactor.
2. Create a LabVIEW™ programme to apply the input signal automatically.
3. Set-up the system.
4. Apply the input signal and record the temperature variations in the reactor.

In order to cover the whole operation range of temperatures of the reactor, the input signal was such that the inner temperature would experience steps across a range from 40 °C to around 200 °C. A programme was created to switch on and off the input signal when certain temperature thresholds were surpassed (*Figure 3-11*). The time was unbounded and the programme was a pure sequence changing to the next step only when the conditions were reached.

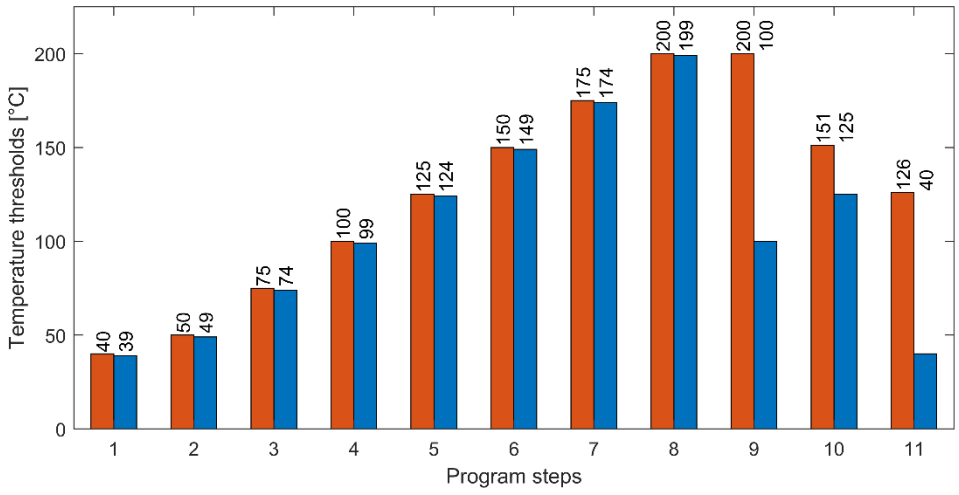


Figure 3-11. Programmed inner temperature thresholds to control input signal (heat on – off) during training signal acquisition.

Red bars correspond to upper thresholds and blue bars to lower ones. For instance, in step one, the input signal was “ON” (heating on) until the temperature reached 40 °C. When 40 °C were surpassed the input signal switched to “OFF” (heating off) until the temperature fell below 39°C. The programme then kept moving to following steps.

3.3.3.2 Validation signal

An independent validation data set was required to evaluate the performance of the NN model trained with training data. A validation signal was obtained applying the same methodology as for the training signal. In this case, less temperature steps were considered (*Figure 3-12*).

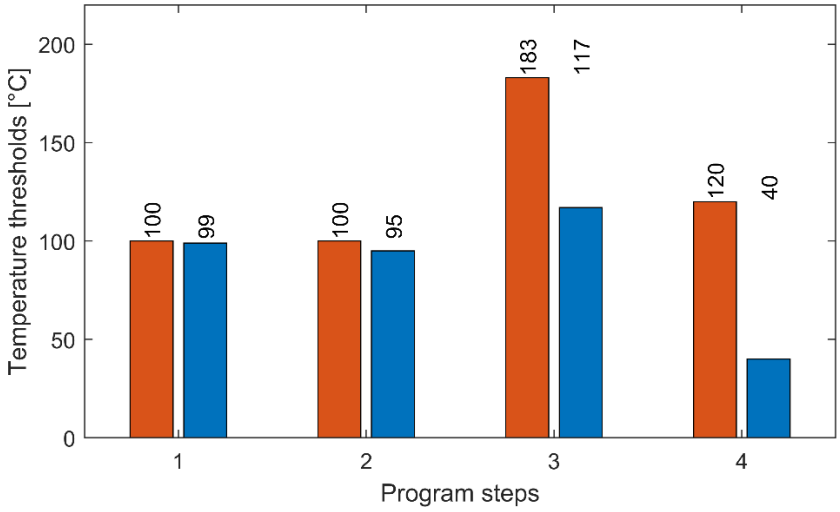


Figure 3-12. Programmed inner temperature thresholds to control input signal (heat on – off) during validation signal acquisition.

3.3.3.3 NN Model training: FANN - open loop

The NN model was trained and run in MATLAB. A set of training data was created from the training signal and two sets of validation data. The methodology used was:

1. Create training data set.
2. Create validation data sets.
3. Training the NN model with the training data set as a feed-forward neural network in open loop.
    a. Evaluate the training by checking the MSE error.
    b. Evaluate the training by checking the fitting of the model's output.

The model was trained by a trial and error approach varying different parameters: output delays, training algorithms, number of neurons and training adjustment. Training was stopped by selecting the maximum number of epochs.

3.3.3.3.1 Time lapse discretization

The time-based data was discretized at time lapses of one second or ten seconds because not good performance was being obtained from one second time lapse data. This is explained in detail in the results and discussion section.

3.3.3.3.2 Training algorithms

Two algorithms were considered: Levenberg-Marquardt backpropagation [28] and Bayesian Regularization backpropagation [29], both available in MATLAB.

3.3.3.3.3 Output delays

Up to thirty (1 second time lapse discretization) or three (10 second time lapse discretization) output delays were considered from the output signal. None were considered from the input signal.

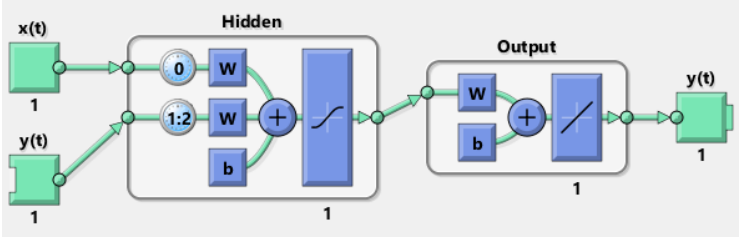


Figure 3-13. FANN (Open loop) with one hidden layer of one neuron.

$$\begin{array}{rc} \text{Inputs} & \mathrm{U}_{\mathrm{train}} = [u(k), y(k-2), y(k-1)] \\ \text{Targets} & \mathrm{Y}_{\mathrm{train}} = [y(k)] \end{array} \qquad (3\text{-}1)$$

For example, Figure 3-13 and (3-1) show a feedforward artificial neural network with two output delays $[y(k-2), y(k-1)]$. Therefore, the network was being trained with three inputs corresponding to the control signal "*u*" in the present time and the two past values of the output signal "*y*" $[u(t), y(k-2), y(k-1)]$ and one target output $[y(k)]$. This kept being changed in search of the delays that accounted as the best input data for training.

3.3.3.3.4 Neural network structure

Neural networks of one hidden layer were considered only varying the number of neurons within that hidden layer (*Figure 3-13*). The hidden layer contained up to

ten neurons. A for loop was run to cover the training of ten different networks of one hidden layer (one for each possible number of neurons in the hidden layer).

3.3.3.3.5 Training adjustment

Epochs are the number of times that the training data set is run across the network during training. A base of 2000 was used while testing training for all the previous parameters.

The aiming MSE during training was very low (0.001) to ensure that training reached the maximum number of epochs and did not stop before (by reaching the aimed MSE). Longer trainings, with very low open-loop MSE error, were in place because, when finished and turned into closed loop, the performance would decrease.

3.3.3.4 NN Model evaluation: RNN - closed loop.

Following the open loop training, the methodology used was:

1. Close the NN model (FANN) from open loop to closed loop. This originated a recurrent neural network (RNN).
2. Run the NN model with input data only and obtain predicted outputs both on training and validation data sets.
3. Plot predicted output values vs. real output values.
4. Evaluate NN model performance in closed loop operation by checking the fitting of the model's predicted output to the real system's behaviour.

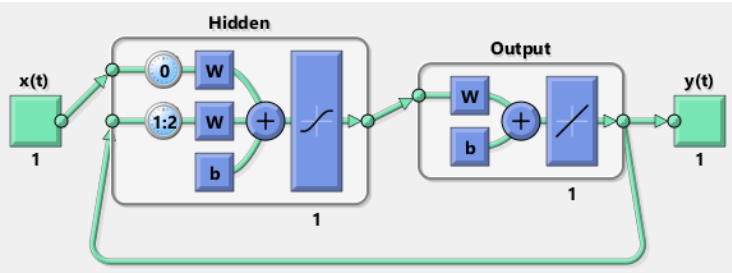


Figure 3-14. RNN (Open loop) with one hidden layer of one neuron.

When closed, the FANN became a RNN (). With this structure the model simulated the reactor having as an input only the control signal (on/off).

3.3.4 Neural network controller

Following the NN model, a second neural network is aimed to be trained to act as the controller of the system. This second neural network is known as “NN controller” in this document. The NN controller was not trained during the project and stands as possible future work.

3.3.5 Control programme

This section includes a description of the most relevant aspects of the NN control programme. It is similar to the PID control programme. The rest is available in Appendix A LabVIEWTM programme. Computing control action was designed to take place every second.

3.3.5.1 Main panel

Main panel behaved as in the PID control programme. In this case, the PID related controls were not available.

Figure 3-15. (NN programme) Main panel.

3.3.5.2 Control action

Control action ran within the main while loop (data acquisition and transmission) every one second (*Table 3-3*) outputting an ON/OFF signal every second.

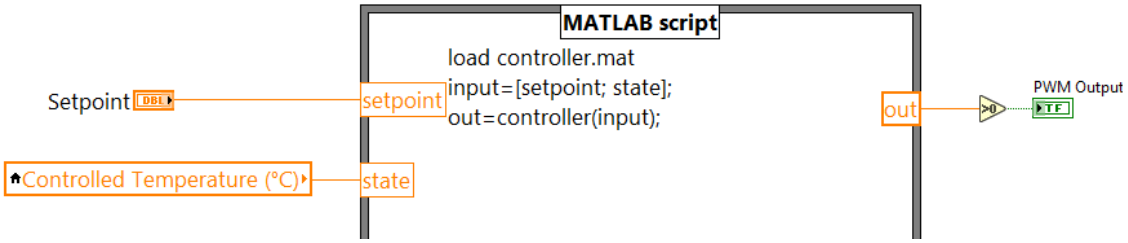


Figure 3-16. (NN programme) Control action loop.

3.4 Reference signal

The reactor always started at room temperature. The reference signal was a step signal to reach 100 °C and constantly stay there. It was limited to that value due to time restrictions.

3.5 Fitting and performance

Along this work, signals were compared with either the NRMSE (Normalized Root Square Mean-Square Error) or the R^2 (R-squared) fitting relationship, known as coefficient of determination. Each section specifies which one was in place.

NRMSE fitting $$\text{Fitting } = 100 * (1 - \frac{\|y - \bar{y}\|}{\|y - mean(y)\|}) \quad (3\text{-}2)$$

Target data y

Predicted data by the model $\bar{y}$

R^2 fitting $$R^2 = 100 * (1 - \frac{MSE(y - \bar{y})}{Var(y)}) \quad (3\text{-}3)$$

Target data y

Predicted data by the model $\bar{y}$

4 RESULTS & DISCUSSION

Results on acquiring the training and validation data (*4.1 Training signal and 4.2 Validation signal*) are presented first because they were also used afterwards during PID tuning (and not only for neural network modelling). Following the results on PID control (*4.4 PID Control*) are presented and finally the ones on NN control (*4.5 Neural network control*).

4.1 Training signal

The training signal obtained was the one showed in Figure 4-1. The black line represents the training signal (temperature inside the reactor) and the dotted line represents the input signal (heating on/off) which is meant to be the control signal of the system. In addition, the temperature of the wall was recorded and is represented by the dashed line.

Three flaws were experienced while running the reactor. They are marked in Figure 4-1 for a better representation:

1. A surge in airflow produced a suddenly-higher cooling action. It can be seen how the temperature drops at a higher rate during a brief period of time.
2. An error in the thermocouple's signal originated a poor registry of the temperatures. This was probably due to a sudden movement of the thermocouple data logger.
3. The programme in the computer failed and stopped recording temperatures. It was restarted afterwards to continue with the experiment recording a second signal. The two signals were concatenated together to show the whole range of training data in Figure 4-1. This would affect later work as the training data had a sudden drop half-way through the array. To avoid this, the training data was split in two: the first part was used as training data and the second (from the third error onwards), as another set of validation data.

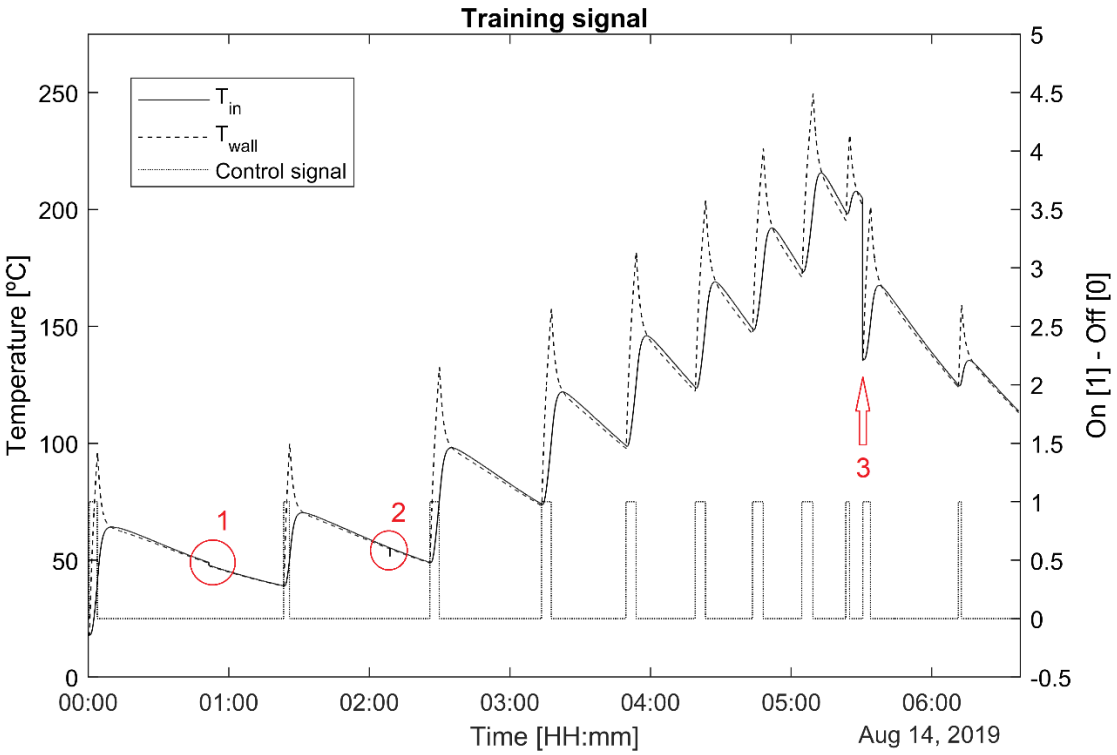


Figure 4-1. Training signal: temperature inside.

It is interesting to mention the dynamic behaviour of the system as it was appreciated during this experience (*Figure 4-2*). When the input signal switched from “ON” to “OFF” (heating on to off) the temperature inside the reactor kept still raising. This was due to the accumulated heat in the reactor’s mass (wall). Heat was still being transferred to the air even when the heating action was off. In this case (*Figure 4-2*), the input signal switched off when the inner temperature was above 75°C. Then the inner temperature kept raining until 98.13 °C which accounts for 23.13 degrees of inertia.

A representation of the reactor’s heat state is the temperature of the wall. The wall temperature started dropping immediately after the heating action was turned off. It had reached 132.5°C when the inner temperature was at 75°C. That temperature difference kept reducing as both temperatures approached each other and met at the highest point of the inner temperature (98.13 °C).

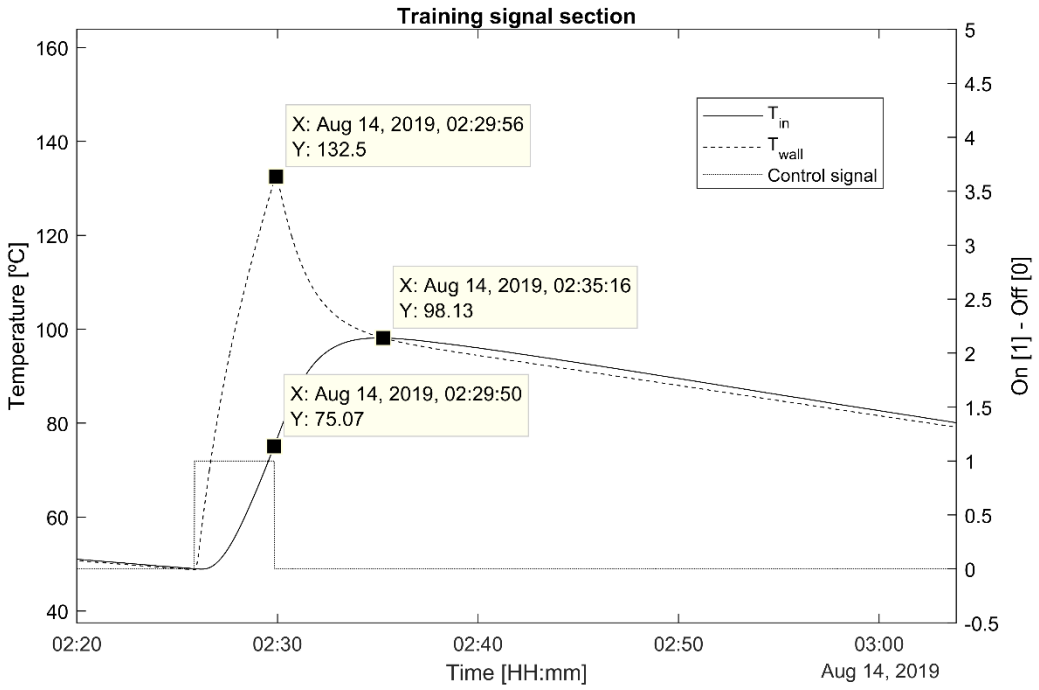


Figure 4-2. Training signal section: ON until 75 °C.

In summary, when the input signal is switched off (heating off), heat transfer to the air keeps occurring for a period of time. The wall temperature immediately starts decreasing but the air temperature keeps raising until they meet. When the air temperature reaches the wall temperature, both start decreasing at a similar rate. This is due to the fact that the reactor's body accumulates heat that is transferred to the air even when the heating action is off.

Finally, it was noted that the output signal (inner temperature) was unbounded. When the input signal is "ON" (heating on) the output signal keeps raising indefinitely until the input signal is switched to "OFF". This would affect the control action as the controller would not reach a steady-state operation point and would need to keep continuously switching the input signal to achieve control.

4.2 Validation signal

The validation signal obtained was the one showed in Figure 4-3. Again, the black line represents the validation signal, the dotted line represents the input signal and the dashed line, the wall temperature. No flaws were experienced during this experience.

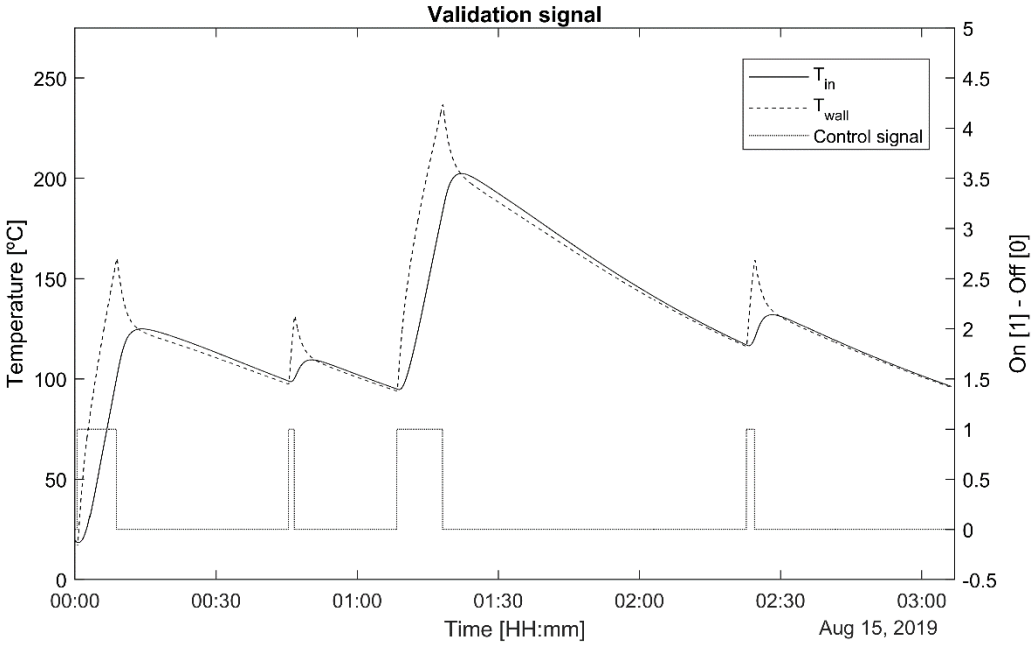


Figure 4-3. Validation signal: temperature inside.

4.3 Training & validation data size

Taking into account the accounted, the three sets of data available were the ones available in Table 4-1.

Table 4-1. Sizes of data sets for model training and validation (dt = 1s).

Data Set	Samples	Time (H:min)
Training	19848	5:30
Validation 1	3995	1:06
Validation 2	11178	3:06

The training data covers the whole range of operation of the reactor during a period of five and a half hours. It is stepped covering temperature raises and drops along this time. For this, it was considered to represent properly the reactor. On the other hand, the validation data sets were designed with the purpose of creating different steps to the ones contained in the training data set to introduce variability and force the model to predict unseen temperature changes.

Data was recorded every second with the intention of then using it to train a model of the system that would represent the system with an accuracy of up to one second. Furthermore, the idea of having a controller operating on the reactor

added to the idea of achieving to design a model that could reach that accuracy. Having such a model would enable to perform simulations in which the controller could refresh the control action at that rate (once per second).

Recording data at one second rate introduced errors in the signal as the ones explained above (*4.1 Training signal*) and also introduced noise (*Figure 4-4*). This noise would later become a problem when modelling the system because it could introduce false trend in close-range consecutive data points. Noise was present in all acquired signals and was introduced into the data due to the slow dynamics of the reactor, especially when cooling down. Below it can be seen how the temperature only drops by 0.14 °C during about 35 seconds. This accounts for a rate of 0.004 °C per second.

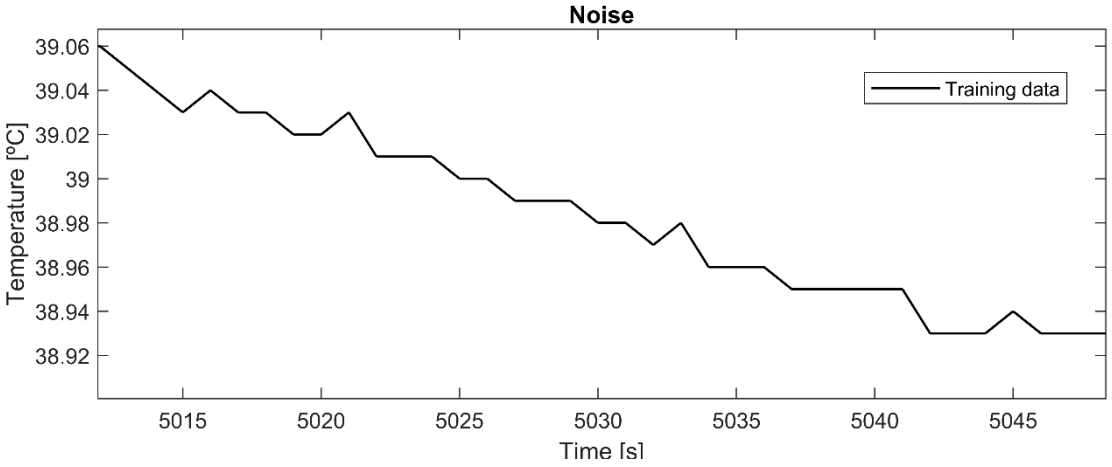


Figure 4-4. Noise present in training data.

Acquiring the temperature at a slower rate would reduce the influence of noise (every 10 seconds for instance) but would then also compromise the accuracy of the model trained with that data, especially when using the model to tune / train controllers that need to operate at a faster rate for better control performance.

4.4 PID Control

4.4.1 Control programme

The control programme has been explained in the methodology (*3.2.4 Control programme*) and in an appendix (*Appendix A LabVIEWTM programme*).

A good amount of time was invested in developing the control programme in LabVIEW™ and ensuring its proper operation. The programme was built from scratch and the work involved a learning process of LabVIEW™, software environment, proceedings and graphical coding methods.

The programme enables communication with two different pieces of hardware (*3.1.1 Components*) from different manufacturers: thermocouple data logger and I/O device. This involved using two extra libraries with specific functions to enable communication between hardware and LabVIEW™, reading hardware manuals to code the programme and subsequent learning process.

Difficulties were also faced to create the on/off signal based on the PID computations. Originally, it was based on greater-than/lower-than PID output value basis and updated every second. If the threshold was at 1 (for instance), anything above meant “on” and everything below meant “off”. A second option considered was to base the on/off in the immediately-previous PID output to capture if the PID tried to increase the control action or decrease it. Finally a PWM signal was created to keep a proportional logic in the control action (*3.2.3 Control signal*).

The first option was basing the on/off on a two-step scale for a period of one second while the implemented PWM signal had a six-step scale over a two second period. This captured better the proportionality of the PID output transferring the “slow-down control action” idea (when reaching the reference signal) or “accelerate control action” (when leaving the reference signal) while not being condemned to do it by a pure on or off value. The second option (immediately-previous basis) was discarded due to the unrealistic behaviour it bonded to the PID output near low/high values. For instance, if the PID outputted the sequential values { 0.5, 0.6, 0.5 } it translated into { 0, 1, 0 } which meant a sudden “on” value for one second when the PID was actually “meaning” a near-zero control action value.

Regarding the general programme operation and design, parallel loops and handling of variables were the main concerns. Parallel while loops are present in the current design and introduced errors in the precision of data acquisition. Data

was acquired every one second (*Table 3-3*) in a loop but, for instance, the control action was running in a parallel loop with a different period (two seconds) and therefore the data was generated at another rate. This meant that a value was generated in time at one point and at acquired at another introducing small time errors of less than one second. In other cases, the errors derived from running code in parallel loops was overcame by using fast sampling period in the "reading" loop. For instance, the PID output value was created every two seconds in a loop and the PWM signal was transferred to the I/O device in a parallel loop every 20 milliseconds. Handling of while loops could be improved by eliminating parallel ones uniting them into a bigger outer one with inner sub-loops or by synchronizing the existent ones.

Handling of variables were generally simply stored in memory until another part of the programme needed to read them. Depending on the design, data could be transferred directly between blocks within the programme.

The programme itself can be improved, especially if reviewed by an experienced LabVIEW™ user. The errors introduced were not considered significant as they were always below one second and could not be captured anyways by data acquisition (as it occurred every one second). As a summary, the programme works, is operational and serves its purpose. While it could be improved, the flaws in design are due to the fact that newcomer to LabVIEW™ developed it.

4.4.2 PID Parameters

PID parameters were tuned using the PID tuner app in MATLAB. For doing so, the app utilises a model of the system to calculate responses with different PID parameters. The user, can then select the ones that provide desired control.

A linear model of the reactor was calculated exclusively for the purpose of PID tuning and was independent from the model designed with neural networks. The linear model was designed and validated with the training and validation data recorded at one second rate. Linked to the discussed previously (*4.3 Training & validation data size*), if the training data did not have a one second resolution, the model could not have a one second resolution. Then, the PID parameters could

not be selected based on that one-second approach. In fact, an alike-problem is what determined the unsuccessful tune of the PID in this project.

4.4.2.1 System Identification for PID tuning

The system was identified with a linear model with two real poles and one delay. This structure was selected through a trial and error process because it provided the best fit to the training data among the structures available in MATLAB's PID Tuner App. The following transfer function describes the model:

$$\text{Transfer function} \qquad \mathrm{G}(s) = \frac{K_P}{(1 + T_{P1} * s)\,(1 + T_{P2} * s)} * e^{-T_d * s}) \qquad (4\text{-}1)$$

The training data had been split in two due to the error that originated a sudden temperature drop mid-way along the set (*4.1 Training signal*). The first part was used as training data and the second as validation. Therefore, three sets of data were available and were used for the analysis of the model:

- "Training data": first part of the training signal.
- "Validation 1 data": second part of the training signal, used as validation.
- "Validation 2 data": original validation signal (*4.2 Validation signal*).

Two models based on the same transfer function ((4-1) were designed and validated by comparing to each other to choose the one that performed better.

- "Regular Model" (Model R): Trained with the regular data.
- "Shifted Model" (Model S): Trained with data shifted to start at 0 °C.

Table 4-2. Parameters of the models considered.

	Regular Model	Shifted Model
K_P	2036.6	1338.9
T_{P1}	55.982	5839.9
T_{P2}	11481	116.1
T_d	63.281	20.947

The reactor always started operating at room temperature (20-25 °C) and the data was shifted to eliminate that offset. Both models were then validated with

both sets of data and compared. Table 4-3 shows the performances of both models when exposed to the regular data. Also available in Figure 4-5.

Table 4-3. NRMSE fitting (%) to the Regular data by both models.

Regular Data	Training (%)	Validation 1 (%)	Validation 2 (%)
Model – Regular	91.06	48.41	58.78
Model – Shifted	75.56	82.44	80.81

Even when working with regular data, the Model S performed better than the Model R in both validation sets. The Model R responded better than the Model S only in the training set. This was due to the fact that the Model R had been actually designed according to that set of training data while the Model S had been designed according to the shifted data. The graphic comparison for both sets of validation data can be found in Appendix B PID Tuning – System Identification.

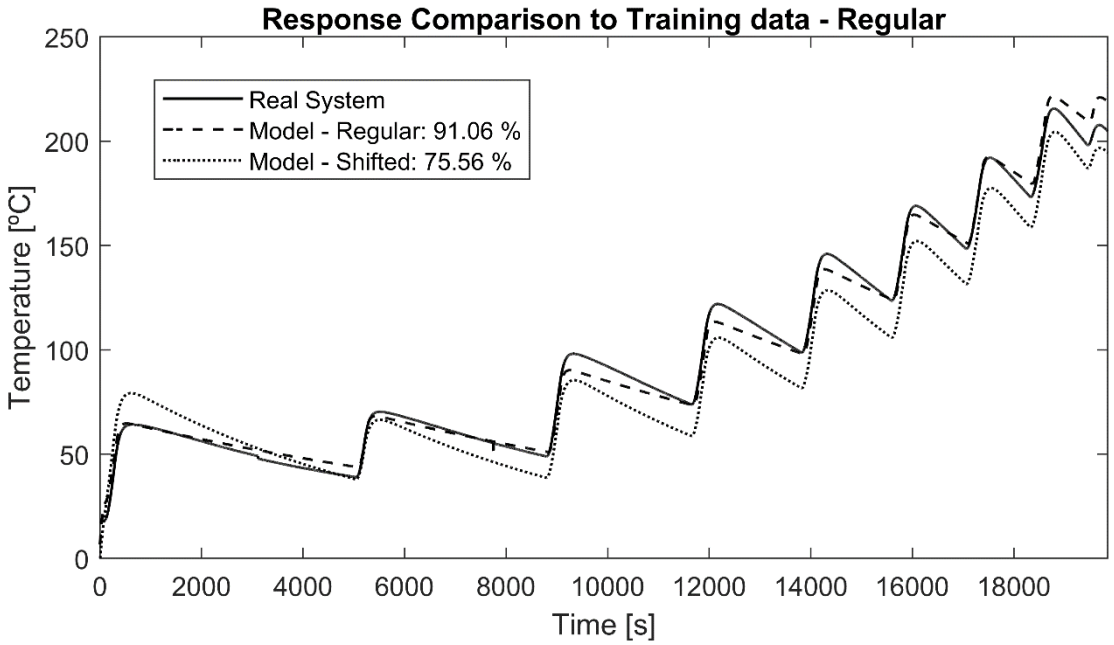


Figure 4-5. Models' responses when simulated with the regular training data set.

On the other hand, Table 4-4 shows the performances of both models when exposed to the shifted data. In this case, the Model S (which had been designed according to the shifted data) outperforms the Model R in every set of data. The

graphic comparison for both sets of validation data can be found in Appendix B PID Tuning – System Identification.

Table 4-4. NRMSE fitting (%) to the Shifted data by both models.

Shifted Data	Training (%)	Validation 1 (%)	Validation 2 (%)
Model – Regular	84.02	37.25	46.74
Model – Shifted	95.58	96.33	93.81

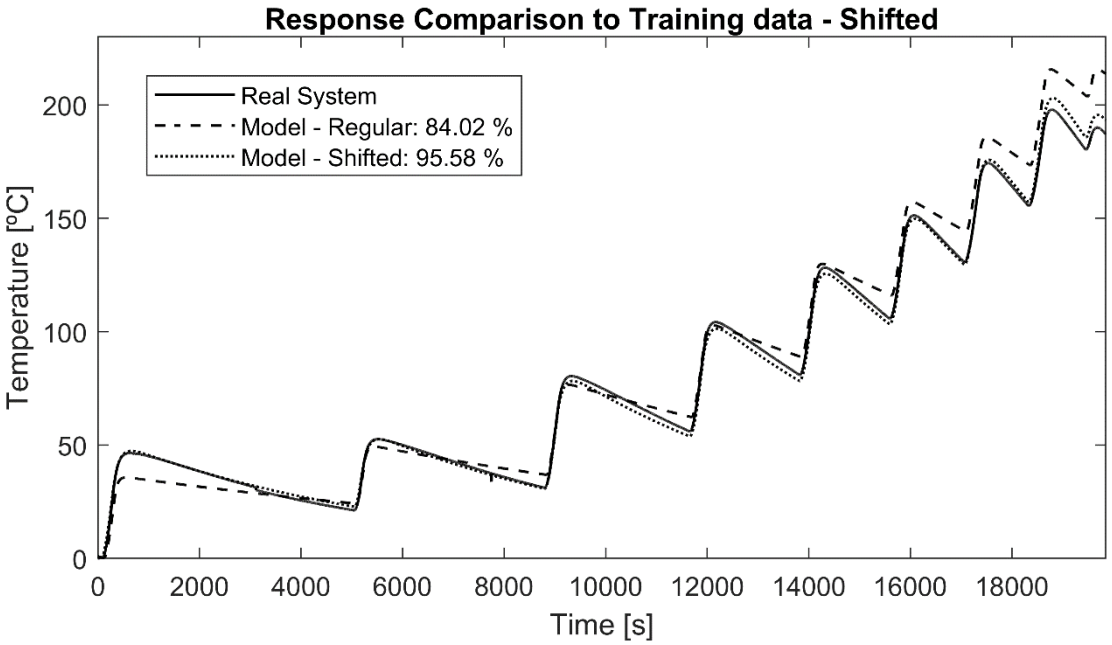


Figure 4-6. Models' responses when simulated with the shifted training data set.

Therefore, the Model S was considered to be the one that modelled better the system. It outperformed the Model R in every comparison except in the one that the Model R had been designed on. This means that shifting the data to zero allowed to identify the system with a better model. Model S was then the one used to select the PID parameters in the PID tuner app in MATLAB.

4.4.2.2 Parameters

By a trial and error approach, the PID parameters selected were:

$$K_P = 0.03, \qquad T_i = 0\ s, \qquad T_d = 40\ s$$

These were responsible for the prediction that can be seen in Table 4-5. As discussed previously, when the input signal switches from “on” to “off” the wall temperature in the reactor is much higher than the inner temperature. In the previous analysis from the training data (*4.1 Training signal*) this difference was of 57 °C which then led the inner temperature to still rise 23 °C after the heating system had been turned off. This response was selected aiming to have a low overshoot and minimize this effect by anticipating the inertia of the inner temperature.

Table 4-5. Predicted response of the system under PID control.

Rise time	3:47	min
Settling time	12:05	Min
Overshoot	7.86	%

4.4.3 PID control results

As explained in the methodology, the PID control action had been designed to output a value to control the duty cycle of a PWM signal (*3.2.3 Control signal*) every two seconds. Although the PID programme had been designed that way, later difficulties and time restrictions led to not being able to tune the PID according to that behaviour and another solution was implemented. The PWM signal operated with a minimum period “on/off” of 400 milliseconds and data with that resolution was not available to create the model in which compute the responses and select the PID parameters.

The PID was tuned on an “on/off” basis (every second) (*4.4.2 PID Parameters*) as can be read in the previous section due to the availability of training data with that time lapse to model the system. In addition the PID programme was modified for this experience by assigning an “on” value to anything above 50% of duty cycle and “off” to anything below.

PID control was set-up in the lab and the results from the control to follow the reference signal can be seen below in Figure 4-7. The PID parameters were the ones available in 4.4.2.2 Parameters.

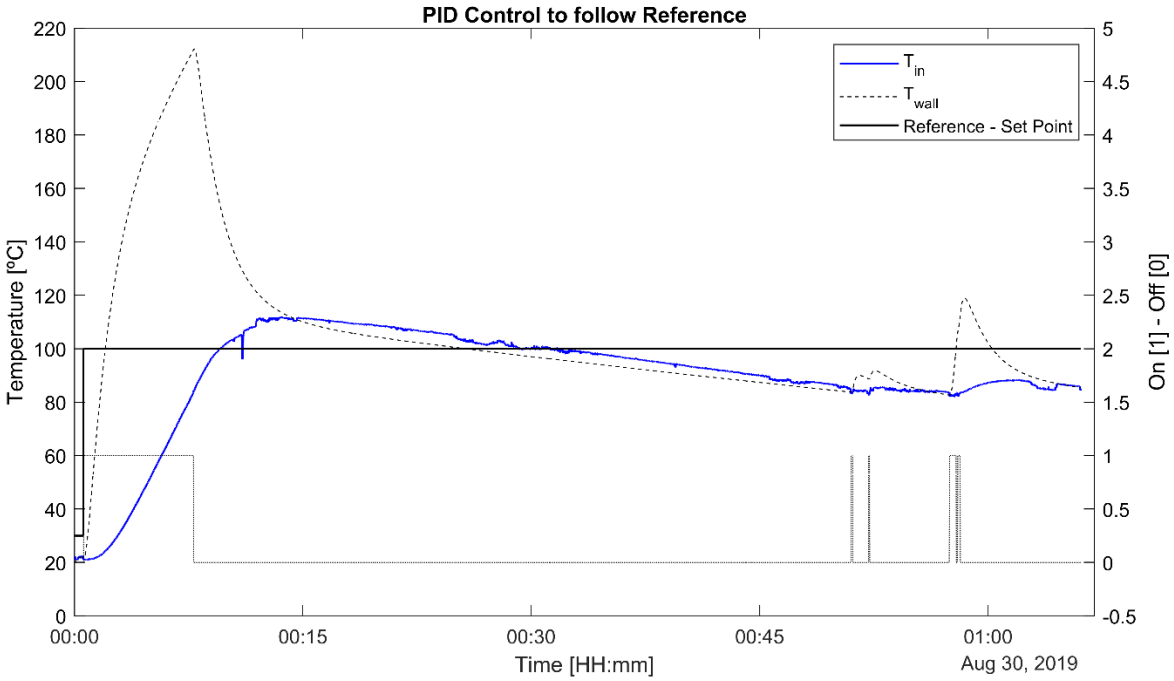


Figure 4-7. Results of PID control to follow a reference signal with a set point of 100 °C.

The response overshoots above the 100 °C reference by almost 18°C. It is more than the expected 7.86 % (7.86 °C). As can be appreciated, the control signal turns to “off” before the inner temperature reaches 100 °C which accounts for good control and avoids an even greater overshoot. Then the PID lets the temperature drop below the reference signal for 20 °C and slightly tries to raise it in two occasions but the control action is not enough.

This probably occurred due to poor tuning and to the fact that the controller was a PD (proportional-derivative) and had no integral action. Integral error accounts for the added previous error and its presence could force the controller to raise the temperature closer to 100 °C when fallen below it for a while. Though, the absence of integral action allowed for a low overshoot (as compared to what it could have been).

Although performance was not satisfactory, this result was accepted as proof that the programme worked and PID control was achieved. Control performance can definitely be improved by improving the parameters selection and also introducing integral action. In addition, implementing the original PID control

(PWM signal) and tuning it accordingly would allow for a finer approach to the reference signal.

4.5 Neural network control

4.5.1 System modelling

System modelling was a challenge during this project. Modelling the reactor with discretized data having time lapses of one second proved to be challenging and the desired behaviour was not achieved. Initially, the model was not even achieving good performances. In order to understand better the process, minimize the influence of noise and to try to achieve a good performance the amount of data was reduced by discretizing it to having time lapses of ten seconds. This enabled the possibility of modelling the system properly and good results were achieved.

The system model trained with ten second data could not be used to train the NN controller because the controller aimed to operate every second and not every ten seconds. What this model did is to prove that the system could be modelled achieving good performances and motivated the continuation of the modelling with one second time lapses. Finally, good performances would be achieved with the one second data model but the behaviour was not favourable. This is discussed later in detail.

Therefore, the approach was to model the system with a reduced amount of data (discretized at ten second time lapses) according to 3.3.3 System modelling and once the reasonable parameters were established, attempt modelling with data discretized at one second time lapses.

4.5.1.1 Resizing data

Again, three sets of data were available for modelling.

- “Training data”: first part of the training signal.
- “Validation 1 data”: second part of the training signal, used as validation.
- “Validation 2 data”: original validation signal (*4.2 Validation signal*).

Table 4-1 shows the original data sets with time lapses of one second. By discretizing the signals with time lapses of 10 seconds, the result was:

Table 4-6. Sizes of data sets for model training and validation (dt = 10s).

Data Set	Samples	Time (H:min)
Training	1983	5:30
Validation 1	398	1:06
Validation 2	1116	3:06

Reducing the amount of data by an order of 10 allowed for faster computational training times and searches of optimal parameters. In addition, the noise influence was reduced as the slow temperature trends in the behaviour of the system were captured better with fewer data points.

4.5.1.2 Model training

A general discussion is provided on model training and later, the specific results for models with ten-second and one-second data are discussed. Modelling was approached by a trial and error and many combinations of the following variables were tried. Only relevant results, including the best models, are showed.

The different parameters in place were: output delays, training algorithms, number of neurons and training adjustment.

4.5.1.2.1 Training goal and methodology: open loop and closed loop

The models were trained in open loop as a feed-forward NN (*Figure 3-13*) and then closed to create the recurrent NN that would model the reactor. This methodology was utilised because training the model in closed loop never provided better results. It was tried two different ways: training the model from scratch in closed loop and training the model first in open loop and then re-training it in closed loop having as a starting point the weights from the open loop training. None of those two methods provided better results that just training the NN in open loop.

Therefore, the NN was trained in open loop and then closed because it proved to be the better approach. The key resided in the fact that performance was being

evaluated in closed loop, meaning that the open loop training had to be good enough to provide good performances in closed loop operation. For this reason, training (open loop) was run for longer periods aiming for a very low MSE error in hopes that the network would perform well in closed loop. As discussed later, it did.

Due to the very low MSE error goal, training was stopped by selecting the maximum number of epochs rather than selecting a minimum MSE. As explained in the methodology, epochs are the number of times that the training data set is run across the network during training. They were the last parameter to be changed while in search of higher performances and a base of 2000 was used to ensure training ran for sufficient time while trying combinations of other parameters.

The aiming MSE during training was set to 0.001 (very low) to ensure that training reached the maximum number of epochs and did not stop before (by reaching the aimed MSE). As a consequence, the open loop performances were outstanding and the models predicted the reactor's behaviour (in open loop) with almost 100% fitting. This, in fact, meant that the models were overfitting the training data, and even, capturing the noise within it and capturing the errors present in the training data. Performances while predicting both validation data sets were also around 100% and therefore that overfitting was not a problem. It was not a problem as a consequence of having a good-enough training data set that covered the whole range of operation.

As a summary, a very ambitious MSE goal was in place to achieve great open-loop prediction performance that would allow for a good one while running in closed loop.

4.5.1.2.2 Training algorithms

Two algorithms were considered: Levenberg-Marquardt backpropagation [28] and Bayesian Regularization backpropagation [29], both available in MATLAB.

4.5.1.2.3 Input delays

No input delays were considered because the input signal to the reactor was a binary variable: on or off that accounted for zero or one. Having an input delay could only be interesting if the previous values were relevant to the behaviour of the system but, in this case, the input signal changed very few times during operation compared to the temperature and kept constant. The most usual behaviour was to switch from one state to another and stay that way rather than fluctuate. No input delays were used for modelling for these reasons and for the fact that including input delays would increase the difficulties during the modelling process itself (more input data, more variability, possibly bigger NN and only being a binary variable).

4.5.1.2.4 Output delays

As mentioned earlier, initially the one second model was not achieving good performance and the ten second model was implemented as a mean of understanding better how to handle the problem. Output delays of the temperature signal became the main parameter to model the reactor, resulting to be key when training the model and obtaining a feasible performance in closed loop.

During open loop prediction, performance was always high due to the very low MSE goal aim during training. When the NN was closed though, having selected key output delays is what dictated if the NN model was going to behave accordingly. Up to thirty (one second time lapse discretization) or three (ten second time lapse discretization) output delays were considered from the output signal. They are discussed later in the corresponded section.

4.5.1.2.5 Neural network structure

As mentioned in the methodology, only neural networks of one hidden layer were considered, containing the hidden layer up to ten neurons. As Khalid and Omatu [3] did, several sizes were tried during training and the best one was utilised. A for loop was run to cover the training of all the plausible networks of one hidden layer (one for each possible number of neurons in the hidden layer). The outcome, as expected, was the finding of NNs that had good performance. This

approach was reasonable taking into account that the longest training period was 12 seconds (*Table 4-10*).

4.5.1.3 Time lapse: ten seconds

The ten second model was trained in search of understanding and good initial parameters for the one second model. As discussed earlier, it was trained several times varying different parameters: training algorithm, output delays, number of neurons and training adjustments like epochs.

In this case, the output delays considered were of up to three. Three output delays in a ten second discretized data accounted for a 30 second period, meaning that the NN was receiving information about the state (temperature) of the reactor of up to 30 seconds ago. If presented with three temperature inputs (output delays of the reactor) $[y(k-3), y(k-2), y(k-1)]$, the NN was being presented with the value of the temperature 10, 20 and 30 seconds ago. This was considered sufficient because this period was significant enough to provide an insight to the NN of the trend of temperature. Since trial and error was in place the model was tried with one, two or three output delays. Table 4-7 and Table 4-8 show the most promising models achieved.

Table 4-7. R^2 fitting of predicted output to target data by the neural network ten second model under different training conditions.

Output delays	1,2				1,2,3			
Data Sets	Lm – 9 N		Br – 4 N		Lm – 1 N		Br – 8 N	
	OL	CL	OL	CL	OL	CL	OL	CL
T	100	93.16	1	98.46	99.99	99.48	100	97.67
V1	100	88.1	1	90.58	99.99	90.98	100	96.04
V2	99.99	96	1	89.29	100	98.66	100	98.37

T = Training, V1 = Validation 1, V2 = Validation 2, N = Neuron
OL = Open loop, CL = Closed loop
Lm = Levenberg-Marquardt backpropagation algorithm
Br = Bayesian Regularization backpropagation algorithm

Table 4-8. Training parameters for the four cases during ten second open loop training.

Output delays	1,2		1,2,3	
	Lm – 9 N	Br – 4 N	Lm – 1 N	Br – 8 N
Epoch	2000	2000	2000	2000
MSE (Open loop)	0.00832	0.00816	0.0232	0.00771
Training Time (s)	4	3	2	4

MSE = Mean Squared Error in Open Loop Training
Lm = Levenberg-Marquardt backpropagation algorithm
Br = Bayesian Regularization backpropagation algorithm

This modelling allowed for the following ideas to be settled:

First, as expected and detailed before, open loop performance was outstanding: reaching 100% continuously. This allowed for good closed loop performances in the range of 88 % to 99 %.

Second, Bayesian Regularization algorithm provided better results if taking into account the combined performances of the three data sets. Anyways performances are so close that, in this model, both of them achieve significant results.

Third, a very simple NN of only one neuron in the hidden layer achieves one of the highest overall performances having 99.48 % in the training data set, 90.98 % for validation 1 data set and 98.66 % for validation 2 data set.

Fourth, the output delays play a significant role and zero delays and one delay are discarded. At least two are needed to achieve good performance and it is proved that three achieve the best results. This is so due to the slow nature of the dynamics of the reactor. Three delays (30 seconds) provide a trend with enough accuracy to predict the next step 10 seconds (one step for 10 second discretized data) ahead.

As a summary, a simple NN of one neuron can achieve a performance of 98.66 % predicting a validation data set in closed loop. This is significant when taking into account that this NN model was able to predict by itself more than 3 hours of

future behaviour of the reactor when only provided with the input signal (that the reactor would see in those 3 hours) and the training time was only of two seconds. This was possible probably due to the good selection of the output delays and was identified as a key parameter to select when training the one second model. The relevance in this experiment resided in the fact that a model was feasible and from here the aim shifted to model the reactor with a one second model.

4.5.1.4 Time lapse: one second

The one second model was trained aiming to have a feasible model that would allow to train the NN controller afterwards with it. It was also trained several times varying different parameters: training algorithm, output delays, number of neurons and training adjustments like epochs.

In this case, the output delays considered were of up to thirty. In reality, only up to four were utilised but they were up to thirty steps back in time. Following the good results achieved with the ten second model, the output delays used during that training were translated and implemented in the one second model. For instance the $y(k-1)$ output delay from the ten second model became the $y(k-10)$ in the one second model. This way, in the one second model, the output delays considered were: $[y(k-30), y(k-20), y(k-10), y(k-1)]$ based in the fact that the first three had worked very well before and adding the $y(k-1)$ to check if the immediately-earlier state could provide any benefit. Again the model was receiving information of the state of the reactor of up to 30 seconds ago. Table 4-9 and Table 4-10 show the most promising models achieved.

Table 4-9. R^2 fitting of predicted output to target data by the neural network one second model under different training conditions.

Output delays	10, 20				10, 20, 30			
Data Sets	Lm – 2 N		Br – 2 N		Lm – 1 N		Br – 1 N	
	OL	CL	OL	CL	OL	CL	OL	CL
T	100	98.22	100	97.97	100	99.08	100	99.24
V1	100	94.85	100	95.14	100	92.18	100	93.6
V2	100	80.43	100	82.35	100	96.77	100	96.99

T = Training, V1 = Validation 1, V2 = Validation 2, N = Neuron
OL = Open loop, CL = Closed loop
Lm = Levenberg-Marquardt backpropagation algorithm
Br = Bayesian Regularization backpropagation algorithm

Table 4-10. Training parameters for the four cases during one second open loop training.

Output delays	1,2		1,2,3	
	Lm – 2 N	Br – 2 N	Lm – 1 N	Br – 1 N
Epoch	2000	1660	1340	1300
MSE	0.0175	0.0175	0.0118	0.0118
Training Time (s)	12	10	6	6

MSE = Mean Squared Error in Open Loop Training
Lm = Levenberg-Marquardt backpropagation algorithm
Br = Bayesian Regularization backpropagation algorithm

Training the one second model provide the following insights:

Again, open loop performance was outstanding reaching 100% continuously and allowing for good closed loop performances in the range of 80 % to 99 %.

Bayesian Regularization algorithm provided again slightly better results if taking into account the combined performances of the three data sets. Performances were very close anyway.

All of the most successful NN that are brought up as the best resulting ones are very simple. Only one or two neurons are in place in the hidden layer. The best

performing model actually, again, achieves the highest performance having 99.24 % in the training data set, 93.6 % for validation 1 data set and 96.99 % for validation 2 data set.

The output delays considered putting into play the previous experience with the ten second model play a significant role in the performance. Other delays also considered, and not showed in the results, did not achieve such good performances. The $y(k-1)$ delay was not significant and therefore not included when looking for the best performance. Regarding the ones used and similarly to the analysis of the ten second model, at least two output delays are needed to achieve good performance and it is proved that three achieve the best results.

Training time increase with respect to the ten second model due to the amount of data now present, which is ten times greater. In addition, better performances were reached by reducing the number of epochs during training: “less training” was better. This was probably due to the presence of noise in the training data and is discussed later.

As a summary, again, the best performance is from a simple NN of one neuron that can achieve a performance of 96.99 % predicting a validation data set in closed loop. This was possible due to the selection of the output delays that had been selected during the ten second model training.

4.5.1.5 Model selection

Since the aim was to train a NN controller on a one second basis, a one second model was to be selected. The one selected was the one that provided the best overall performance on the three data sets taking into account that it had performed sufficiently well in the validation data sets.

From all the previous models, the recurrent neural network that achieved better fitting overall in closed loop was the one with one hidden layer of one neuron trained through Bayesian Regularization backpropagation and was selected as the NN Model.

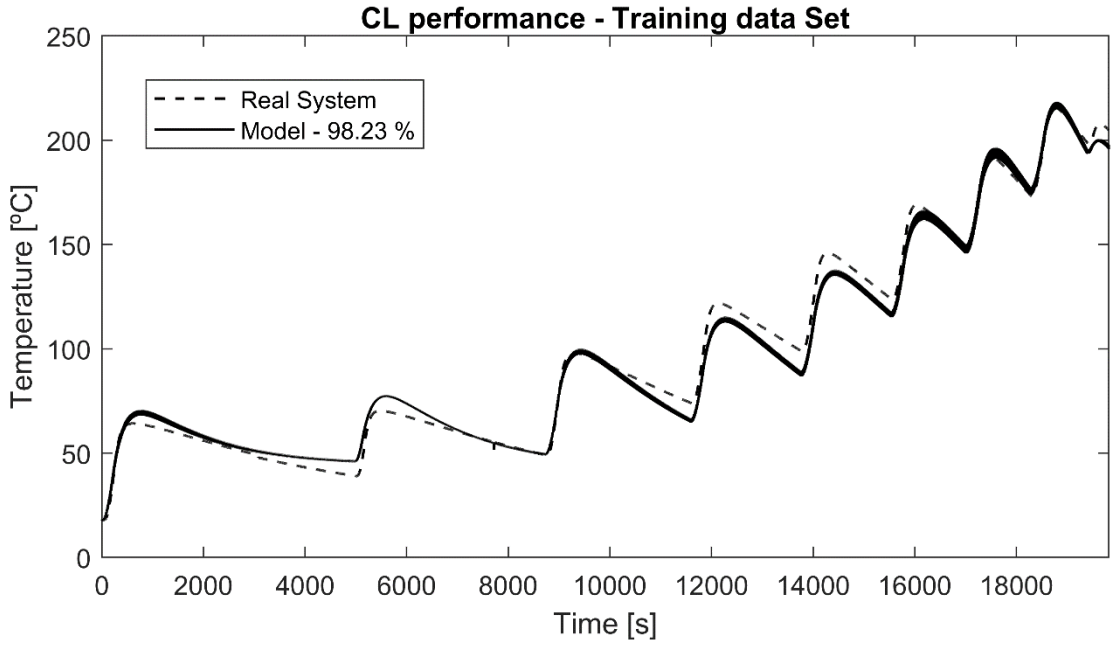
The NN Model: one hidden layer with one neuron, trained with Bayesian Regularization algorithm (*Table 4-9 and Table 4-10*) during 1300 epochs. The input and output delays are showed below.

$$\begin{array}{rc} \text{Inputs} & \mathrm{U_{train}} = [u(k), y(k-30), y(k-20), y(k-10)] \\ \text{Targets} & \mathrm{Y_{train}} = [y(k)] \end{array} \quad (4\text{-}2)$$

Up to here, everything looked right: the model was working and performances were high. The problem was identified when plotting the prediction of the one second model. Even though performance was high, noise appeared in the prediction and is discussed in the following section.

4.5.1.6 Model behaviour

In this section only the behaviour of the selected NN Model is reflected in the figures. Take into account that all the figures represent only this model (*4.5.1.5 Model selection*). This behaviour is discussed and in general terms, compared to other behaviours that were experienced when searching for the best model.



.

Figure 4-8. NN Model performance in closed loop on the training data set.

Figure 4-8 shows the fitting to the training data set of the NN Model. As can be appreciated, the prediction of the model is a thick line. This is because there are

vibrations in the signal at high frequency and the scale of the figure depict it as thickness.

Figure 4-9 shows the fitting to the validation data two of the NN Model. A zoom view of the noisy prediction has been provided in this figure. That noise is produced by the neural network model and makes impossible its use to train the NN controller.

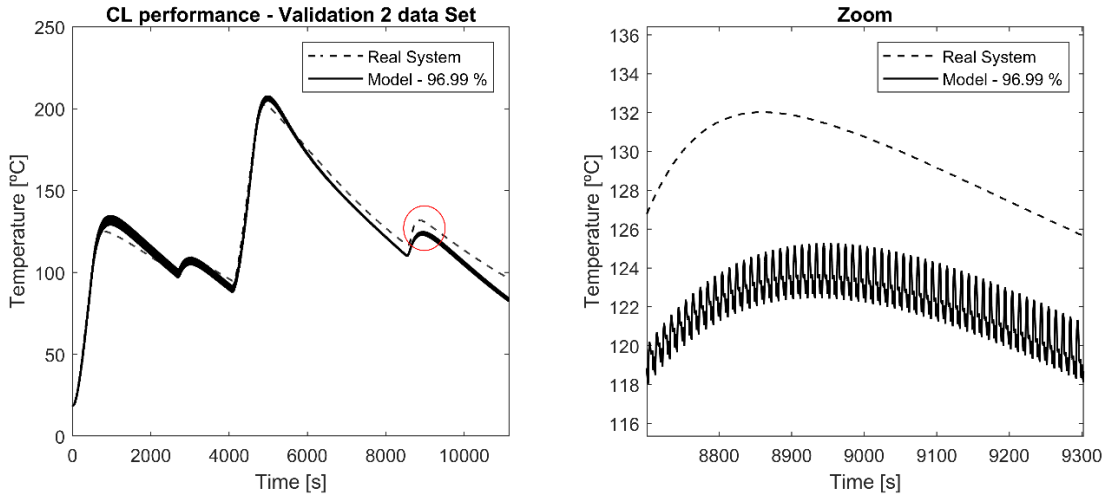


Figure 4-9. NN Model performance in closed loop on the validation 2 data set and zoom on the noise from the predicted output.

The predicted signal containing the high frequency noise produced within the NN Model was filtered with a zero-phase low pass filter coded in MATLAB (*Appendix C MATLAB Code*). The ripples of the noisy prediction can be appreciated in Figure 4-10. Filtering enabled to obtain a good signal of the prediction of the reactor's behaviour during operation as can be seen in Figure 4-11.

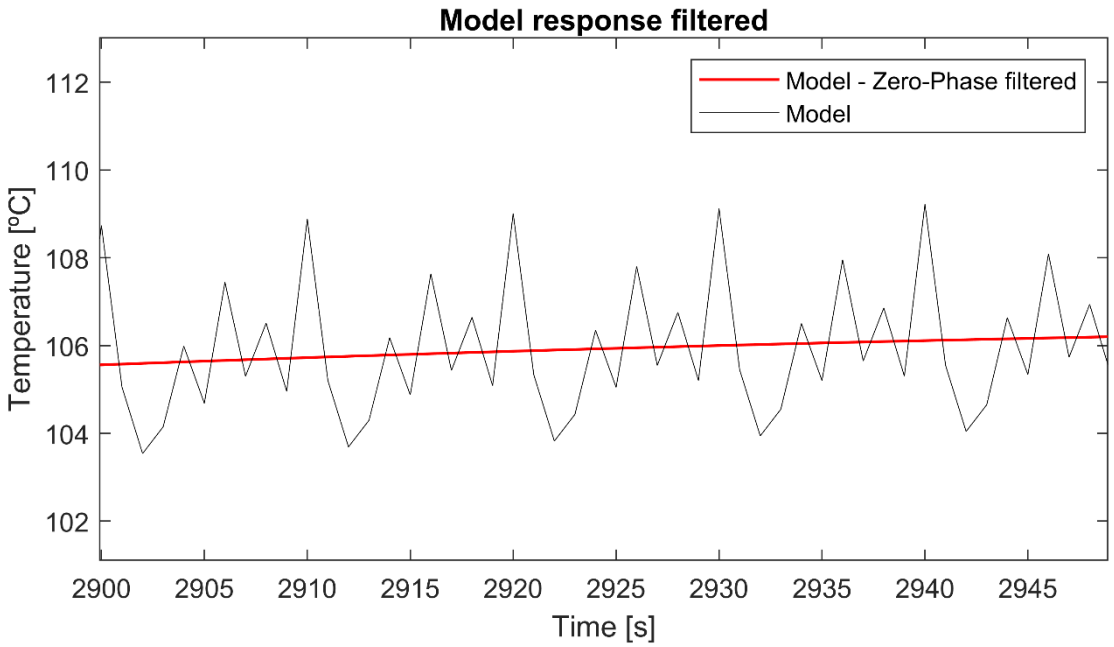


Figure 4-10. Ripples within model response and filtered signal.

Therefore, the NN Model outputted a noisy signal that after filtering provided a very good prediction (98.23 % fitting) of the reactor's behaviour. Hence, the system was modelled and the model could be operative if used with the filter afterwards.

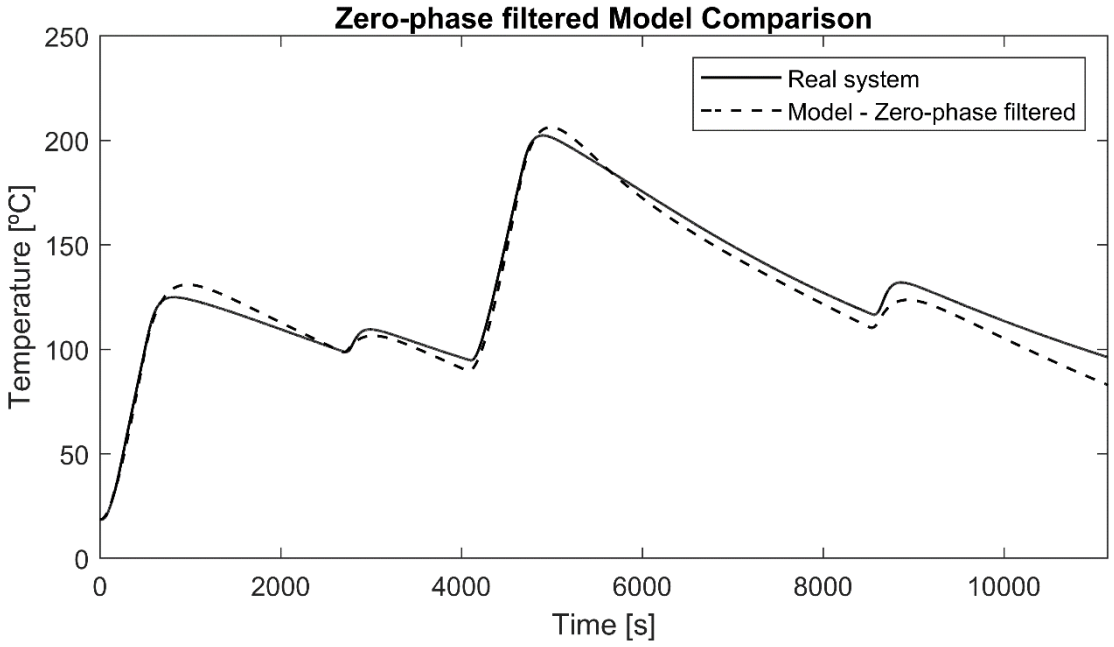


Figure 4-11. NN Model behaviour after filtering the predicted output.

This was a general issue for the one second models and appeared in every single one of them. It was only avoided by using very low epochs that then would not achieve desired performance. On the other hand the ten second model did not have this problem and could provide a prediction without noise and similar in terms of smoothness to the signal depicted in Figure 4-11.

This behaviour was suspected to be originated in the training data itself. The output delays (previous temperature value at 10, 20 and 30 seconds) were fed as inputs to the model at time “k” and then “slid” along the array of training data. Imagine this values were stored in the following positions in the array: position 1 for “k-30”, position 11 for “k-20” and position 21 for “k-10”, having the present time “k” at position 31. At time “k+1” (the next time step) the new output delays were still at minus 10, 20 and 30 but, in the array, they had “slid” to positions 12, 22 and 32 respectively.

This meant that if noise was present in the training data, it was being fed to the NN Model anyway (even if the output delays were separated 10 time steps in the array). They were then being fed but “shifted” in time.

To aim to solve this, it was tried to filter the training data before feeding it to the NN for training. It did not provide better results and the NN Model would predict with a noisy wave afterwards anyway. Therefore, this noisy behaviour became the problem that prevented further development, together with time restrictions in this project.

4.5.2 NN controller and control

The NN Model could not be used to train the NN controller because of the nature of the operation at hand. The NN controller was to operate at one second rate to provide fine control being able to refresh the “on/off” command every second. Therefore the model needed to train that controller needed to be able to simulate the behaviour of the system at a one second rate. The one second model was the one to be used and the ten second model (which did not introduce noise in the prediction) could not be used for this purpose.

Furthermore, training the NN controller would occur with a discretized data set. The procedure would be something similar to: feed a set point to the NN controller, predict a control signal value (0 or 1), feed that value to the NN Model and compare the predicted output of the model with the real temperature of the reactor. Then, see if the control input had been good evaluating that comparison and back-propagate the error backwards through the model and train the controller itself. This would be an iterative process that would occur for every time step. Computing one value at a time on a discretized data set, a filter could not get rid of noise and therefore the prediction of the model (disturbed by noise) would be inadmissible to compare it to the real temperature of the reactor and hence, inadmissible to back-propagate the error training the NN controller.

Due to time restrictions and the fact that a NN Model (on a one-second time basis) with a reasonable behaviour could not be trained, the NN controller was not designed. The design of the NN controller and implementation of NN control was left as reasonable start for future work.

5 CONCLUSION

Controlling operating conditions in chemical reactions is a critical matter in chemical engineering to achieve and even improve desired results. PID control is considered as a standard within industry due to its proven functionality but presents challenges in tuning and the fact that knowledge of the system is required for this purpose. Machine learning – based algorithms such as artificial neural networks (ANN) are in place because they can perform controlling duties, utilise a learning-from-the-system approach rather than requiring knowledge beforehand and are potentially simple to implement with nowadays technologies.

In this project, the main aim was to implement temperature control in a simplified chemical reactor utilising ANN to enable control and evaluate its performance by comparing it with traditional PID control.

For this purpose, the process within the reactor was limited to a constant air flow and an on/off heating system. DAQ, control action and signal transferring take place in a computer through a LabVIEW™ programme developed by the author. Control was designed to follow a reference signal. The controlled variable was the reactor's inner temperature and the control variable was the on/off state of the heating system.

The reference signal was limited to a constant value of 100 °C due to time restrictions during the last days of the project. Therefore the objective of tracking a signal along the whole range of operation in steps was reduced to a single constant step that raised from room temperature to the mentioned one. Datasets of temperature points were generated during control to register the activity.

PD (Proportional – Derivative) control was enabled within a feedback control loop through a LabVIEW™ programme. Temperature control aimed to follow a constant referent signal at 100 °C starting from room temperature (25 °C). PID parameters were selected according to a MATLAB simulation with the PID tuner app. The system was identified with a linear model with two real poles and one delay and then, PID parameters were selected by simulating a response in the model that would be satisfactory. Reasonable control action was achieved taking

into account the slow response of the reactor in time. Nonetheless, performance was not satisfactory as overshoot was 18 °C when it had been predicted to be around 8°C for a set point of 100 °C (accounting for an 18% vs 8% overshoot). It could be improved by improving PID tuning and introducing integral action in the controller as it was implemented only with proportional derivative action. As a conclusion on PID control, PID control was achieved and PID tuning was performed (although it could be improved).

A NN model was designed with sights to train a second NN as controller that could later be utilised to perform control action on the real system. One-second discretized training and validation data was acquired from the reactor by operating it along its whole range of operation. A set of training data spanning more than 5 hours was acquired and two sets of validation data that span a total of 4 hours. Acquiring the signals every second introduced noise in the data due to the slow dynamics of the reactor and would prove to be influential afterwards. It was done this way because it was needed such a high resolution in the data to afterwards train models with one second resolution.

The model was trained through back-propagation in open loop as a feed-forward NN and later, closed to create a recurrent NN that modelled the reactor. The model was trained only in open loop (and not in closed) because it was experienced that closed loop training provided worse results. In order to achieve feasible performances in closed loop when the model was not being trained in closed loop, open loop training aimed to achieve very small MSE errors (0.001) during training. As a consequence the models over fitted the data in open loop but behaved accordingly in closed loop reaching R^2 fittings of 98 % to the validation data.

Initially the NN proved to be challenging to model on one-second basis. The data was re-discretized at ten second time lapses and a ten second model was trained with sights to achieve modelling the reactor with a feasible performance and understand which parameters were more influential during training. The ten second model allowed to reach a performance R^2 fitting of 98.37 % to the validation data, defined the significant delay outputs to be at (k-3), (k-2) and (k-

1) and proved that a simple NN of one neuron in one hidden layer could reach that 98.37 % performance mentioned above.

Once this information was available, a one second NN model was trained. The output delays were translated to a one second time base and proved to work using this time (k-30), (k-20), (k-10). A 96.99% fitting to the validation data was achieved with a NN Model of only one neuron in one hidden layer. Nonetheless, the NN Model introduced noise in the prediction signal. This noise was filtered with a zero-phase low pass filter and the resulting signal proved to be a very good predictor of the behaviour of the reactor maintaining the 96.99 % fitting.

The presence of noise in the prediction of the NN Model prevented it from being suitable to train the NN controller. The NN controller would need to be trained over discretized data (on a one second base to achieve control accuracy) in an iterative process. The noise present in the NN Model prediction would made the temperature to shift in each iteration uncontrollably (as no filter could be applied) and therefore could not be used to train the controller.

As a summary of NN control, it was possible to train a ten second model that predicts the behaviour of the reactor accordingly (98.37 %) and it was possible to train also a one second model that along a filter provides a good prediction (96.99 %). The one second model could not be used to train a NN controller as noise is present during iterative operation and therefore, and due to time restrictions the NN controller stands as something to do.

As a conclusion, neural network control was not achieved and its implementation in the set-up stands as plausible future work following the one developed in this project.

As a conclusion of the project, the main objective of implementing neural network control and compare it to PID control was not accomplished. PID control was achieved and registered and good progress was made towards modelling the reactor with neural networks in order to train the NN controller. Future work would include focusing on eliminating noisy predictions from the NN models, train the

NN controller and implementing it in the laboratory to finally compare both methods.

REFERENCES

[1] K. J. Åström and T. Hägglund, *Advanced PID Control*, 1st ed. ISA - Instrumentation, Systems, and Automation Society, 2006.

[2] A. Zulu, “Towards explicit PID control tuning using machine learning,” in *2017 IEEE AFRICON*, 2017, pp. 430–433.

[3] M. Khalid and S. Omatu, “A neural network controller for a temperature sontrol system,” *IEEE Control Systems Magazine*, vol. 12, no. 3, pp. 58–64, 1992.

[4] K. S. Narendra, “Neural networks for control: Theory and practice,” *Proc. IEEE*, vol. 84, no. 10, pp. 1385–1406, 1996.

[5] M. B. N. Shah *et al.*, “PID-based temperature control device for electric kettle,” *Int. J. Electr. Comput. Eng.*, vol. 9, no. 3, pp. 1683–1693, 2019.

[6] J. Schwarzenbach and K. F. Gill, *System Modelling and Control*, Third. Edward Arnold, 1992.

[7] K. H. Ang, G. Chong, S. Member, and Y. Li, “PID control system analysis and design,” *IEEE Control Syst.*, vol. 26, no. 1, pp. 32–41, 2006.

[8] T. M. Mitchell, *Machine Learning*, Internatio. Singapore: McGraw Hill Book Company, 1997.

[9] G. W. Irwin, K. Warwick, and K. J. Hunt, *Neural Network Applications in Control*. Institution of Electrical Engineers, 1995.

[10] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, “Neural networks for control systems-A survey,” *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[11] Z. Ahmad, R. ’Adawiah Mat Noor, and J. Zhang, “Multiple neural networks modeling techniques in process control: a review,” *Asia-Pacific J. Chem. Eng.*, vol. 4, no. 4, pp. 403–419, 2009.

[12] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp.

436–444, 2015.

[13] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.

[14] A. Dertat, “Applied Deep Learning - Part 1: Artificial Neural Networks,” *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6. [Accessed: 12-Aug-2019].

[15] D. H. Nguyen and B. Widrow, “Neural networks for self-learning control systems,” *International Journal of Control*, vol. 54, no. 6. pp. 1439–1451, 1991.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.

[17] D. Psaltis, A. Sideris, and A. A. Yamamura, “A multilayered neural network controller,” *IEEE Control Syst. Mag.*, vol. 8, no. 2, pp. 17–21, 1988.

[18] K. S. Narendra and K. Parthasarathy, “Identification and control of dynamical systems using neural networks,” *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[19] N. V. Bhat, P. A. Minderman, T. McAvoy, and N. S. Wang, “Modeling Chemical Process Systems via Neural Computation,” *IEEE Control Syst. Mag.*, vol. 10, no. 3, pp. 24–30, 1990.

[20] A. M. Shaw, F. J. Doyle, and J. S. Schwaber, “A dynamic neural network approach to nonlinear process modeling,” *Comput. Chem. Eng.*, vol. 21, no. 4, pp. 371–385, 1997.

[21] P. Technology, “Thermocouple Data Logger,” 2019. [Online]. Available: https://www.picotech.com/data-logger/tc-08/thermocouple-data-logger.

[22] M. Measurement Computing, “USB-1208FS-Plus,” 2019. [Online]. Available: https://www.mccdaq.com/usb-data-acquisition/USB-1208FS-

Plus.aspx. [Accessed: 20-Aug-2019].

[23] M. Measurement Computing, “USB-1208FS-Plus User’s Guide,” 2019. [Online]. Available: https://www.mccdaq.com/PDFs/manuals/USB-1208FS-Plus.pdf. [Accessed: 20-Aug-2019].

[24] M. Measurement Computing, “MCC Data Acquisition Software,” 2019. [Online]. Available: https://www.mccdaq.com/MCC-Software. [Accessed: 20-Aug-2019].

[25] M. Measurement Computing, “ULx for NI LabVIEW™,” 2019. [Online]. Available: https://www.mccdaq.com/daq-software/universal-library-extensions-lv.aspx. [Accessed: 20-Aug-2019].

[26] Crydom, “Series 1 240 VAC,” 2018. [Online]. Available: http://www.crydom.com/en/products/catalog/series-1-240-ac-panel-mount.pdf. [Accessed: 20-Aug-2019].

[27] Crydom, “D2425 Panel Mount Perfect Fit AC Output Series 1,” 2019. [Online]. Available: http://www.crydom.com/en/products/panel-mount/perfect-fit/ac-output/series-1/d2425/. [Accessed: 20-Aug-2019].

[28] Mathworks, “trainlm,” *Documentation*, 2019. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/trainlm.html?searchHighlight=trainlm&s_tid=doc_srchtitle. [Accessed: 30-Aug-2019].

[29] Mathworks, “trainbr,” *Documentation*, 2019. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ref/trainbr.html?searchHighlight=trainbr&s_tid=doc_srchtitle. [Accessed: 30-Aug-2019].

APPENDICES

Appendix A LabVIEW™ programme

A general view of the programme operation is given in this appendix. While a detailed explanation is available, notes can be seen in the snapshots.

A.1 PID Control programme

A.1.1 Start-up

During start-up, a stacked sequence structure creates the files in tab 0 and structures them in tab 1 (Figure A-1 and Figure A-2).

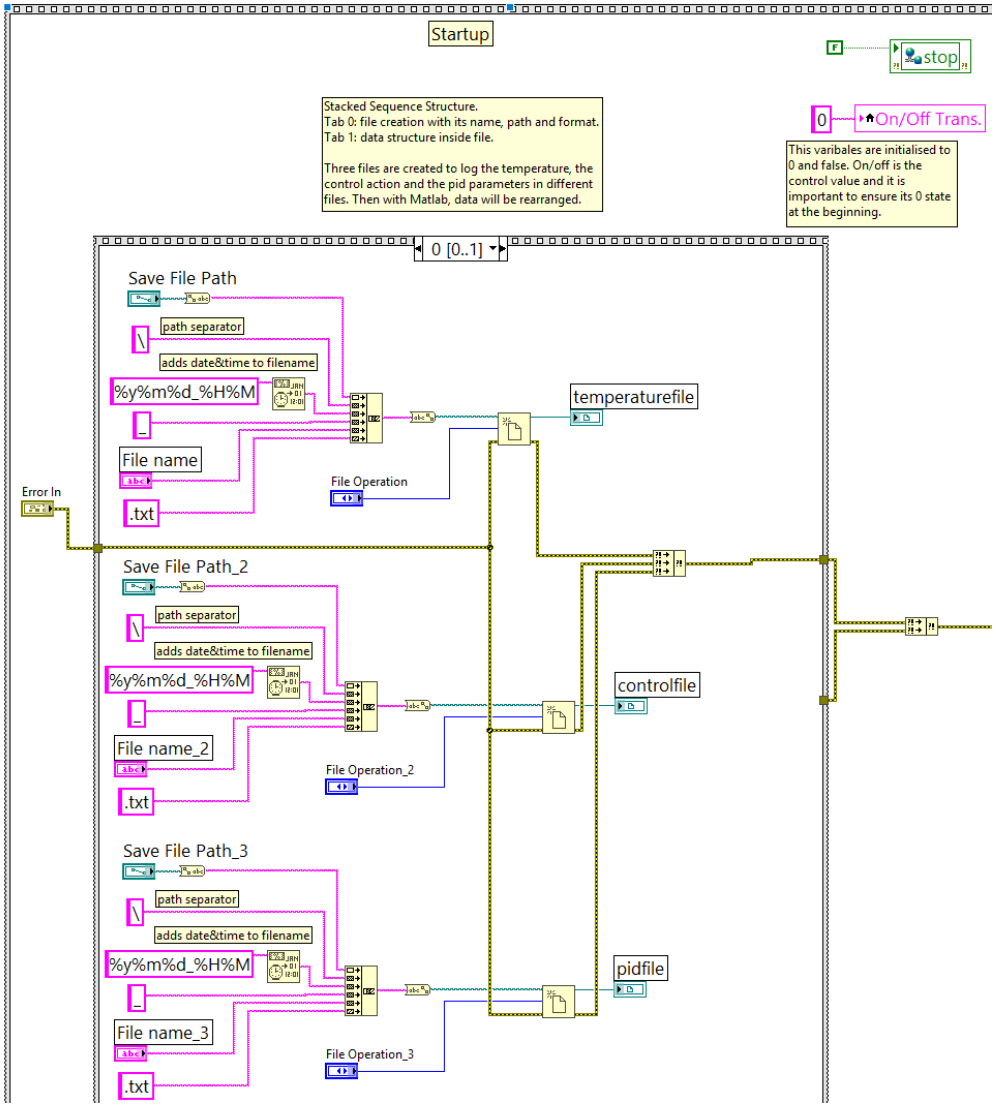


Figure A-1. (PID programme) Tab 0: File creation.

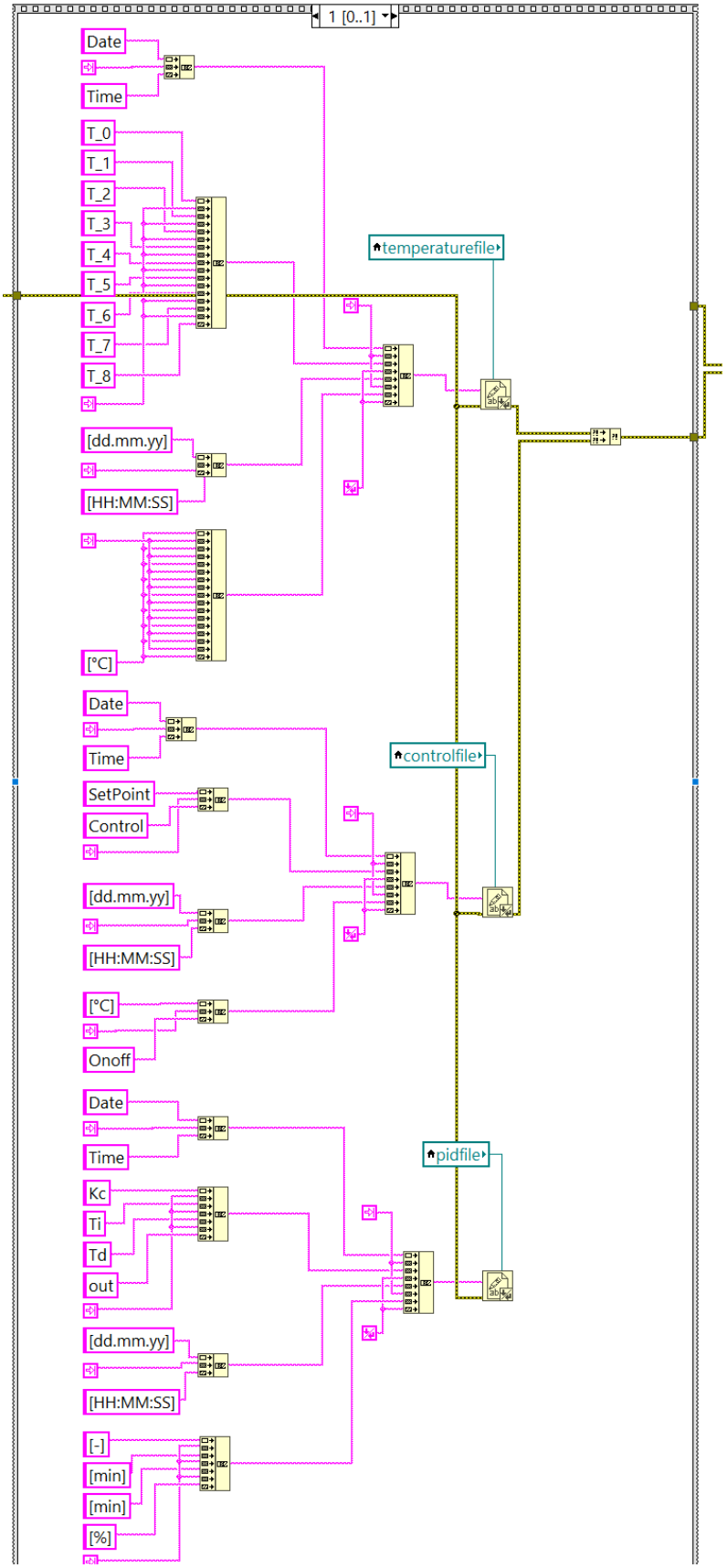


Figure A-2. (PID programme) Tab 1: File structuring.

A.1.2 Run

Data acquisition is described in the thesis report (3.1.4.2.1 Data acquisition). As mentioned there, only relevant temperatures are kept by selecting them in the main panel (Figure A-3). Data is saved in the pertinent files according to Figure A-4. Acquisition and saving both run within the same while loop.

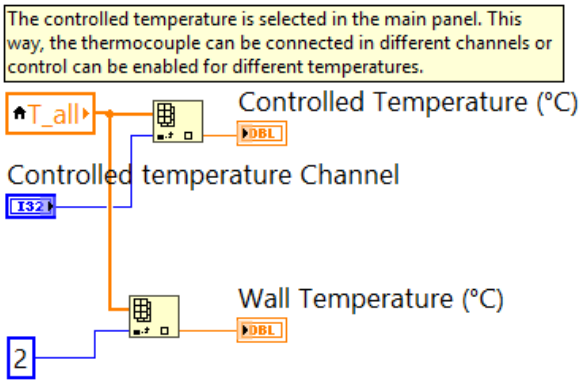


Figure A-3. (PID programme) Temperature selection.

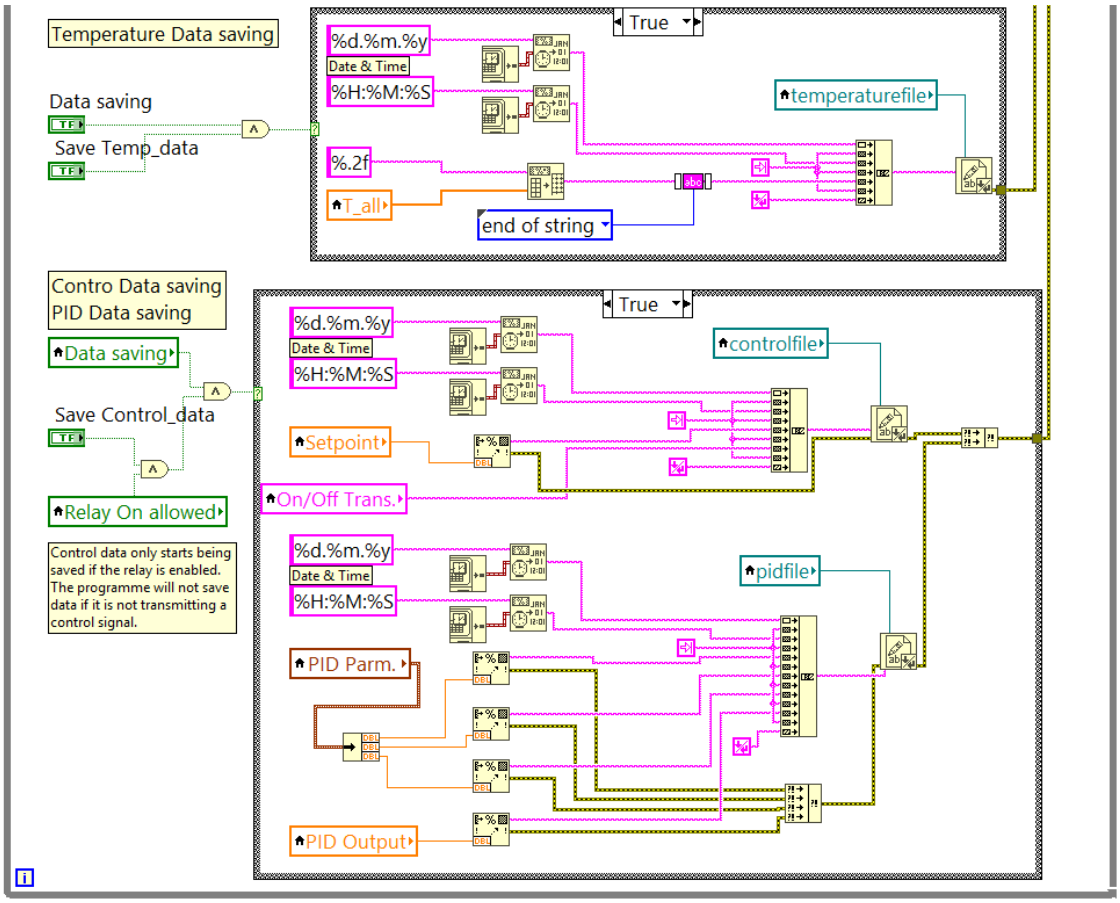


Figure A-4. (PID programme) Data saving in correspondent files.

The control action from the PID programme is described in the thesis report in 3.2.4.2 Control action. And the main panel is available in 3.2.4.1 Main panel.

A.1.3 Shut-down

The reader is referred to 3.1.4.3 Shut-down.

A.2 NN Control programme

A.2.1 Start-up

During start-up, a stacked sequence structure creates the files in tab 0 and structures them in tab 1 (Figure A-5 and Figure A-6).

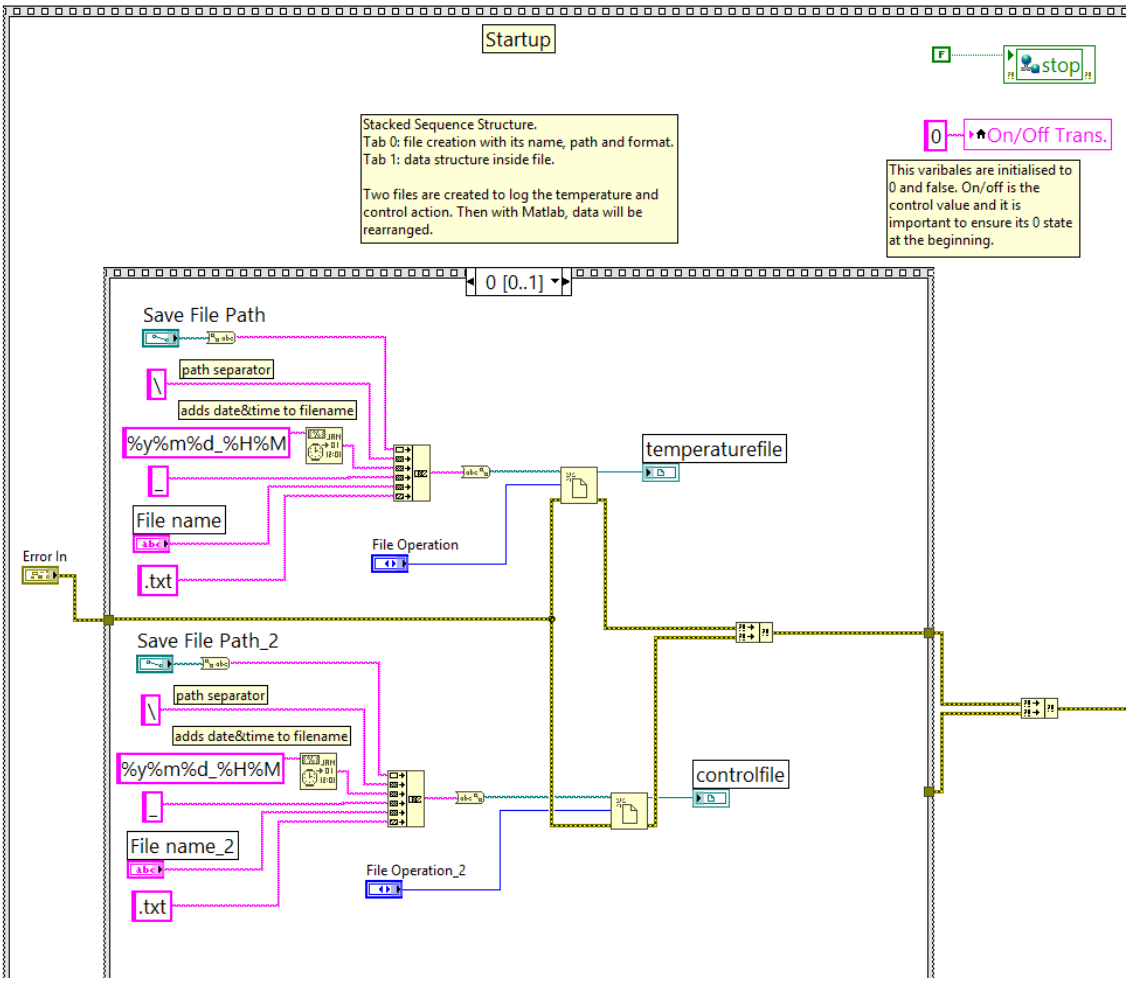


Figure A-5. (NN programme) Tab 0: File creation.

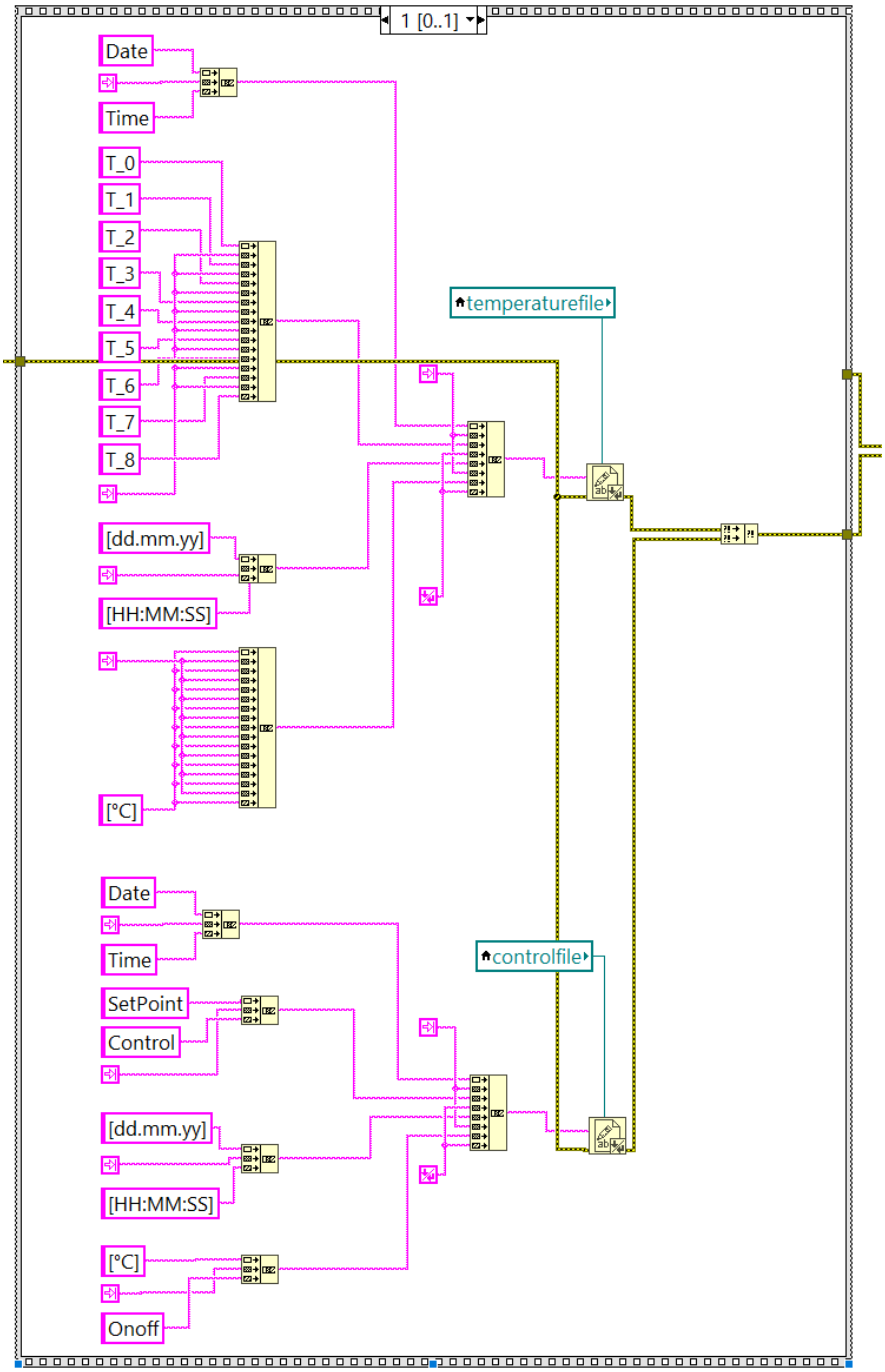


Figure A-6. (NN programme) Tab 1: File structuring.

A.2.2 Run

Data acquisition can be seen in the thesis report (3.1.4.2.1 Data acquisition). As mentioned there, only relevant temperatures are kept by selecting them in the main panel (Figure A-3). Data is then saved in the pertinent files according to Figure A-7.

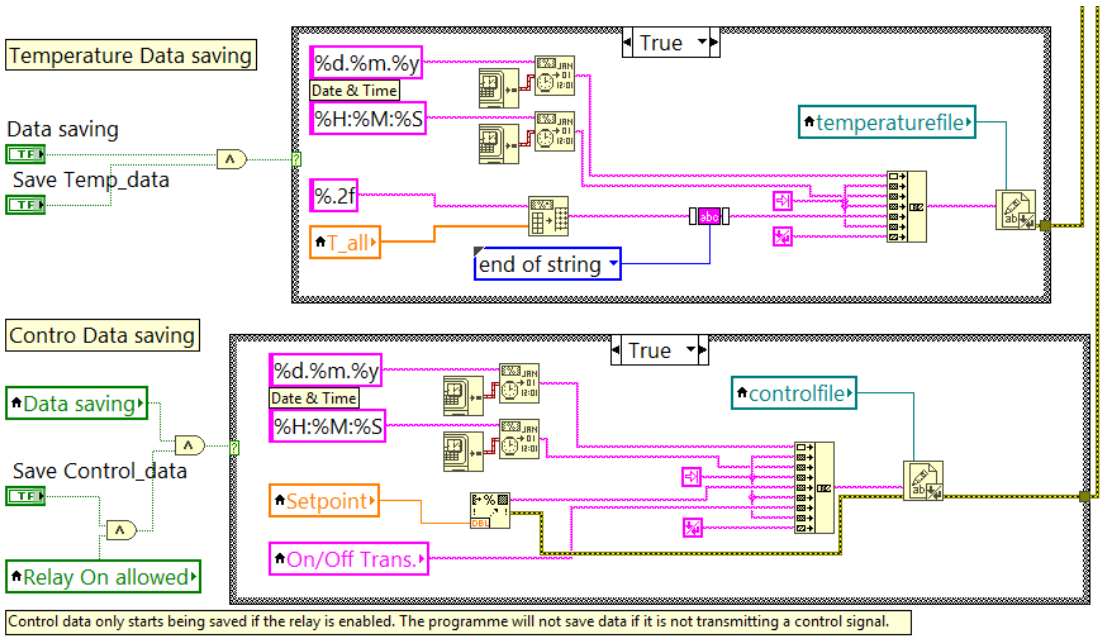


Figure A-7. (NN programme) Data saving in correspondent files.

The control action from the NN programme is described in the thesis report in 3.3.5.2 Control action. And the main panel is available in 3.3.5.1 Main panel.

A.2.3 Shut-down

The reader is referred to 3.1.4.3 Shut-down.

Appendix B PID Tuning – System Identification

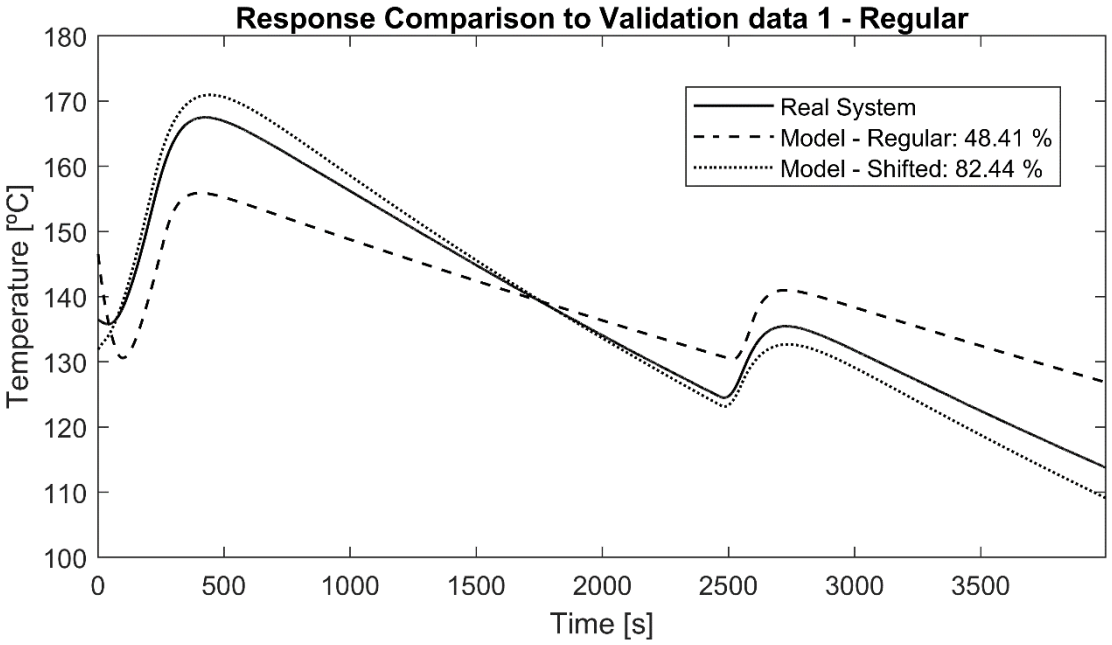


Figure B-1. Models' responses when simulated with the regular validation 1 data set.

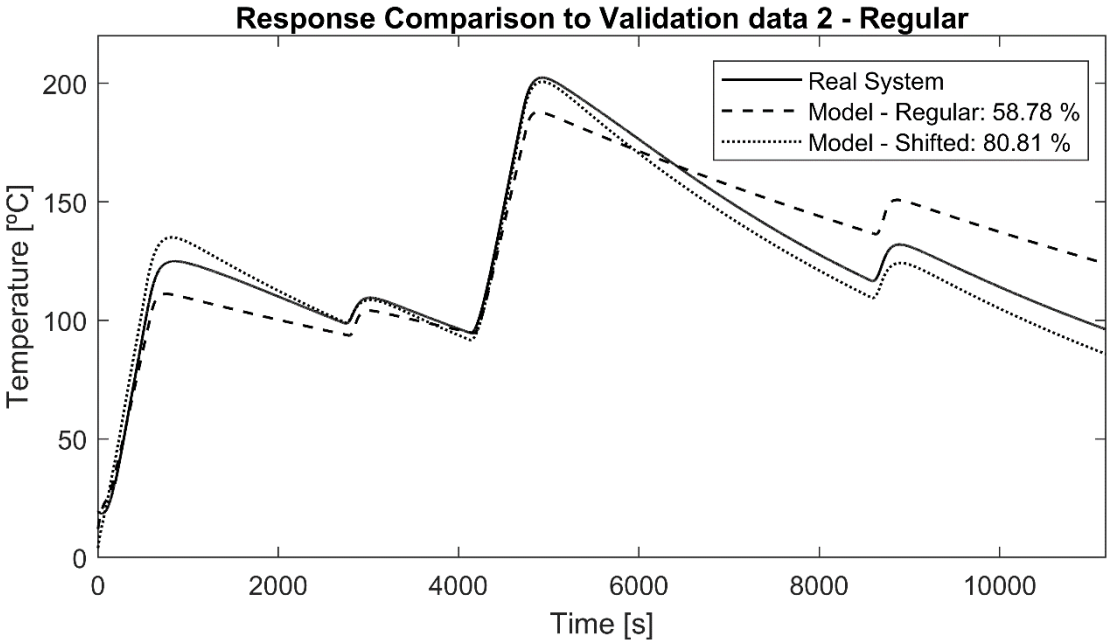


Figure B-2. Models' responses when simulated with the regular validation 2 data set.

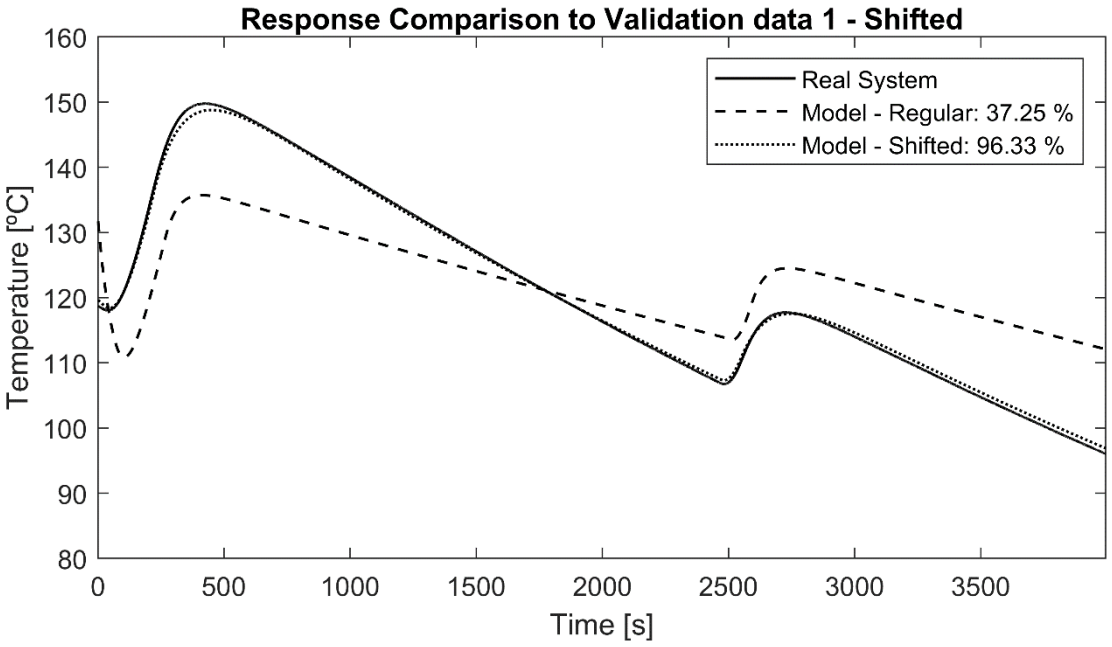


Figure B-3. Models' responses when simulated with the shifted validation 1 data set.

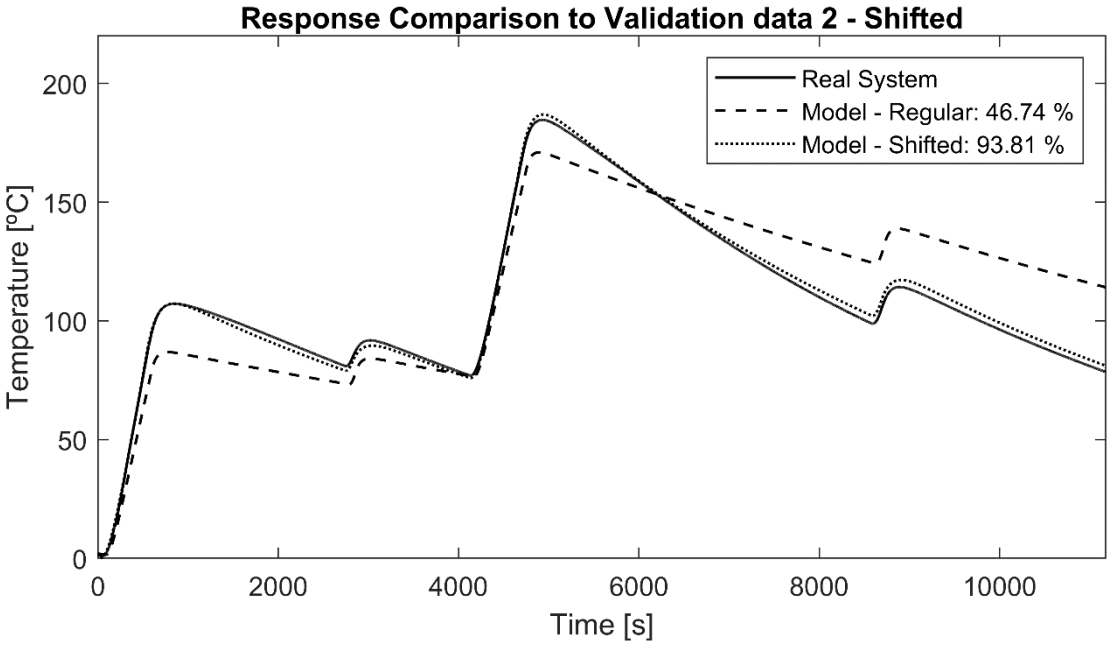


Figure B-4. Models' responses when simulated with the shifted validation 2 data set.

Appendix C MATLAB Code

C.1 NARX

```
%% Training data: Input vector (x), Target vector (t).
x_t=NN_training1(1,:);
t_t=NN_training1(4,:);
x_t=con2seq(x_t); %Arrays to cells
t_t=con2seq(t_t);

%% NARXNET %%
%% Delays
d1=[0]; %inputDelays - none
d2=[10,20,30]; %feedbackDelays
neuron=1; %Number of neurons
net=narxnet(d1,d2,neuron); % [neuron neuron] for two layers

%% Training Parameters
net.trainFcn='trainbr'; %Training algorithm. Pick one
% net.trainFcn='trainlm'; %Training algorithm
% net.trainParam.max_fail=100; %Use only with trainlm
net.divideFcn=''; %No division, all training data
% net.divideFcn='divideblock'; %Division between training and
validation data
net.trainParam.min_grad=1e-8;
net.trainParam.goal=0.001; %Error small
net.trainParam.epochs=1300;

%% Network preparation & Training
%Preparation of input and target time series data for network
simulation or training
[p,Pi,Ai,t]=preparets(net,x_t,{},t_t);
%x_t are "non-feedback inputs from training data"
%t_t are "feedback targets from training data"
%p=shifted inputs
%t=shifted targets
%Pi= Initial input delay states
%Ai= Initial layer delay states

rng('default'); % Normalization Reference. Added for
reproducibility. Everytime training starts with the same random
weights in the net.

[net, tr, yo_t, e]=train(net,p,t,Pi); tr
%y_t Predicted output by the net.
%e Error between target and network prediction.
```

```
%% CLOSED-LOOP %%
net_closed=closeloop(net);
net_closed.name=[net_closed.name ' - Closed Loop'];

%% Prediction training data
[pc,Pic,Aic,tc_t]=preparets(net_closed,x_t,{},t_t);
yc_t=net_closed(pc,Pic,Aic);
ec=cell2mat(yc_t)-cell2mat(tc_t); %Error between target and
network prediction.
```

C.2 Low pass filter

```
%%Low pass Filter %%
d = designfilt('lowpassfir', 'PassbandFrequency', 0.08,
'StopbandFrequency', 0.09,'PassbandRipple', 1,
'StopbandAttenuation', 60, 'DesignMethod', 'kaiserwin');

predicted=y; %Signal to filter

yzero = filtfilt(d,predicted); %Zero-phase digital filtering

figure,hold on
plot(predicted)
plot(yzero);
title('Filtered Waveforms')
legend('Unfiltered','Zero-phase Filtering')
%% END %%
```