

# The generalization of feedforward control for a periodic motion system

A comparison of data-driven methods

Stijn Bosma

Master of Science Thesis



# **The generalization of feedforward control for a periodic motion system**

## **A comparison of data-driven methods**

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Systems and Control at Delft  
University of Technology

Stijn Bosma

March 15, 2019

# ASML

The work in this thesis was supported by ASML. Their cooperation is hereby gratefully acknowledged.

 **TU Delft** Delft  
University of  
Technology

Copyright © Delft Center for Systems and Control (DCSC)  
All rights reserved.

# DCSC

DELFT UNIVERSITY OF TECHNOLOGY  
DEPARTMENT OF  
DELFT CENTER FOR SYSTEMS AND CONTROL (DCSC)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

THE GENERALIZATION OF FEEDFORWARD CONTROL FOR A PERIODIC MOTION  
SYSTEM

by

STIJN BOSMA

in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE SYSTEMS AND CONTROL

Dated: March 15, 2019

Supervisor(s):

\_\_\_\_\_  
Ir. C. Verdier

\_\_\_\_\_  
Dr. ir. M. Mazo Jr.

Reader(s):

\_\_\_\_\_  
Prof. dr. ir. M. Wisse

\_\_\_\_\_  
Dr. ir. M. Mazo Jr.

\_\_\_\_\_  
Ir. C. Verdier



---

# Abstract

A repetitive motion system supporting nano meter precision is positioned at high accelerations, which produces a force that disturbs the demanded accuracy requirements. Iterative learning control is used to learn optimal feedforward control signals for the attenuation this disturbance force. The iterative method comes with a limitation, as it has to learn the compensation signals one by one. In this study, a data-driven approach is taken to design a feedforward controller that generalizes the control action, by finding a model that fits the learned optimal compensation signals. The multi-objective evolutionary algorithm genetic programming is used for its unique ability to search directly in the program space, therefore returning compact analytic equations as feedforward controller. Feedforward artificial neural networks with the ReLU and tanh activation function are used, due to their excellent approximation qualities. SHERLOCK is a tool that efficiently computes the maximum and minimum output off a ReLU network for a compact input set. The tool is used to find a guaranteed output range for the feedforward controller constructed from ReLU networks, which is desirable from a safety perspective. Finally, the designed controller are compared based on their attenuation of the disturbance force and their model size. The designed feedforward controllers are capable of reducing the disturbance force by at least 40.01%. The analytic controllers found with genetic programming provide the user with insights in the control problem, cost less memory storage and come with a faster computation time.





---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1-1	Motivation . . . . .	1
1-2	Problem statement . . . . .	3
1-2-1	Approach . . . . .	4
1-2-2	State of the art . . . . .	6
1-3	Contribution of the thesis . . . . .	7
1-4	Thesis outline . . . . .	8
<b>2</b>	<b>Preliminaries</b>	<b>11</b>
2-1	The data sets available for the feedforward controller design . . . . .	11
2-2	Measures used for the processing of te results . . . . .	13
2-3	Genetic Programming . . . . .	15
2-4	Artificial Neural Networks . . . . .	17
2-4-1	Framework . . . . .	17
2-4-2	Training of a neural network . . . . .	20
2-5	SHERLOCK . . . . .	20
<b>3</b>	<b>Feedforward controller design by means of genetic programming</b>	<b>23</b>
3-1	Experimental Set-Up . . . . .	24
3-1-1	Research questions . . . . .	24
3-1-2	Varying parameters in the experimental set-up . . . . .	24
3-1-3	Other parameters in the experimental set-up . . . . .	26
3-1-4	Processing the results of the genetic programming experiment . . . . .	28
3-2	Results . . . . .	29
3-2-1	The analysis of compact models found by genetic programming . . . . .	29
3-2-2	The influence of of the input variables on the feedforward controller performance . . . . .	33
3-2-3	Feedforward controller performance . . . . .	37
3-2-4	Concluding remarks . . . . .	38
3-3	Discussion . . . . .	39
3-4	Conclusion . . . . .	40

<b>4</b>	<b>Feedforward controller design by means of feedforward artificial neural networks</b>	<b>41</b>
4-1	Experimental Set-Up . . . . .	42
4-1-1	Research questions . . . . .	42
4-1-2	Varying parameters in the experimental set-up . . . . .	42
4-1-3	Other parameters in the experimental set-up . . . . .	45
4-1-4	Processing the results of the neural networks experiment . . . . .	46
4-2	Results . . . . .	47
4-2-1	Accuracy Analysis . . . . .	47
4-2-2	Feedforward controller performance . . . . .	51
4-2-3	Concluding remarks . . . . .	53
4-3	Discussion . . . . .	54
4-4	Conclusion . . . . .	54
<b>5</b>	<b>Feedforward controller comparison</b>	<b>57</b>
5-1	Comparing the feedforward controllers from a performance perspective . . . . .	58
5-2	Comparing the feedforward controllers from a complexity perspective . . . . .	61
5-2-1	Gaining insight in the behavior of the compensation signals in the different planes . . . . .	61
5-2-2	The memory consumption of the feedforward controllers . . . . .	63
5-2-3	The computation time of the feedforward controllers . . . . .	63
5-3	Comparing the feedforward controllers controllers on other aspects . . . . .	64
5-3-1	Output bound verification of the feedforward controllers . . . . .	64
5-3-2	The influence of the number of training data points on the performance of the feedforward controllers . . . . .	64
5-4	Concluding Remarks . . . . .	66
<b>6</b>	<b>Discussion</b>	<b>69</b>
6-1	On the performance of the feedforward controllers . . . . .	69
6-1-1	The limitation in accuracy for the models found with genetic programming . . . . .	69
6-1-2	The difference in accuracy of the neural network models . . . . .	71
6-1-3	The extrapolating qualities of the network controllers . . . . .	71
6-2	On the complexity of the feedforward controllers . . . . .	72
6-3	On the output range analysis of the feedforward controllers . . . . .	72
6-4	On the training data needed for the design methods . . . . .	73
6-5	On another approach to design the feedforward controller . . . . .	73
<b>7</b>	<b>Conclusion</b>	<b>75</b>
<b>A</b>	<b>Appendix Genetic Programming</b>	<b>77</b>

---

<b>B</b>	<b>Neural Networks</b>	<b>87</b>
B-1	The influence of the hidden layers on the accuracy of the tanh networks . . .	88
B-2	The influence of the training data set on the accuracy of the tanh networks .	92
B-3	The influence of the hidden layers on the accuracy of the ReLU networks . .	95
B-4	The influence of the training data set on the accuracy of the ReLU networks	99
B-5	The influence of the hidden layers on the performance of the tanh controllers on the training set $Q$ . . . . .	103
B-6	The influence of the training data sets on the performance of the tanh controllers	104
B-7	The influence of the hidden layers on the performance of the ReLU controllers	106
B-8	The influence of the training data sets on the performance of the ReLU controllers	107



---

# List of Figures

1-1	The control loop of the system and an example of a reduced compensation signal . . . . .	2
1-2	Control loop including the feedforward controller . . . . .	2
1-3	Schematic overview of the outline of the thesis . . . . .	9
2-1	An example of a setpoint signal and its velocity and acceleration . . . . .	11
2-2	The parameterized setpoint signals in the space $\mathcal{P}$ . . . . .	12
2-3	The repetitive nature of the setpoint signal and its corresponding compensation signals . . . . .	13
2-4	An example of a syntax expression . . . . .	16
2-5	Schematic representation of a feedforward neural networks . . . . .	18
2-6	An overview of all activation functions considered in this study . . . . .	19
3-1	The signals used in $Q_{GP}$ sampled in the parameterized training dataset $Q$ . . . . .	27
3-2	The normalized frequency envelope for all signals $\mathbf{u}_1^*$ in training data set $Q$ . . . . .	27
3-3	The location of the setpoint signals in space $\mathcal{P}$ . . . . .	30
3-4	The output of the trivial model and the optimal compensation signal for both setpoint signals . . . . .	31
3-5	The MSE for all signals in dataset $Q$ for the trivial model $\mathcal{U}_{1,triv}$ . . . . .	31
3-6	MSE for all signals in training set $Q$ for the models $\mathcal{U}_{2,triv}$ and $\mathcal{U}_{3,triv}$ . . . . .	32
3-7	The influence of the input variables on the accuracy of the models . . . . .	33
3-8	An overview of models which can be chosen from to create the controller $\mathcal{U}_{IS1}$ dependent on $\mathcal{R}_{r1}$ . . . . .	34
3-9	The accuracy of the models $\mathcal{U}_1$ chosen for the four feedforward controllers . . . . .	36
3-10	The output of $\mathcal{U}_{1,triv}, \mathcal{U}_{1,IS1}$ and signal $\mathbf{u}_1^*$ for setpoint signal 2 . . . . .	36
3-11	The accuracy of the models considered in the four feedforward controllers . . . . .	37
3-12	The controller performance of the 4 feedforward controllers . . . . .	38
4-1	The signals per training dataset sampled in the parameterized training set $Q$ in the neural networks experiment . . . . .	44

4-2	The influence of the hidden layers on the performance of the networks representing signal $u_2$ . . . . .	48
4-3	Time domain plots of the output of a $\tanh$ and ReLU network with 3 hidden layers. . . . .	49
4-4	The influence of the hidden layers in time domain for the $\tanh$ networks . . .	49
4-5	The influence of the hidden layers in time domain for the ReLU networks . .	50
4-6	An overview of the influence of the training data sets on the accuracy of the neural networks . . . . .	51
4-7	All results summarized for the networks $\mathcal{U}_2$ . . . . .	51
4-8	The influence of the hidden layers on the controllers performing on all signals in training set $Q$ . . . . .	52
4-9	The influence of the hidden layers on the controllers performing on all signals in set $R_3$ . . . . .	52
4-10	The influence of the hidden layers on the controllers performing for all signals in set $R$ . . . . .	53
5-1	a Pareto front of the designed feedforward controllers . . . . .	57
5-2	The set of performance measures for the controller on the setpoint signals in the training set $R$ . . . . .	58
5-3	The set of reduction measures for the feedforward controllers on all setpoint signals in validation set $R$ . . . . .	59
5-4	The performance of the $\tanh$ controller expressed in percentages of reduction visualized throughout parameterized space $\mathcal{P}$ . . . . .	60
5-5	The performance of the ReLU controller expressed in percentages of reduction visualized throughout parameterized space $\mathcal{P}$ . . . . .	60
5-6	The performance of the GP controller expressed in percentages of reduction visualized throughout parameterized space $\mathcal{P}$ . . . . .	61
5-7	The two setpoint signals used to gain insight in the behavior of the $\tanh$ network output . . . . .	62
5-8	Analyzing the output of the $\tanh$ neural network $\mathcal{U}_2$ with 3 hidden layers for signals outside the training set . . . . .	62
5-9	The calculation time the three feedforward controllers . . . . .	63
5-10	The sets $\mathcal{Z}_R$ for the controllers trained on dataset $Q_{GP}$ and $Q_{q1}$ . . . . .	65
5-11	The influence of the number of training data points on the performance of the controller . . . . .	66
6-1	Visualizing the component dependent on $p_2$ and the way in which more complex GP controllers approximate it . . . . .	70
6-2	Zoomed in plot of networks $\mathcal{U}_\epsilon$ with 3 hidden layers . . . . .	71
A-1	Normalized frequency domain envelope for the compensation signals $U$ in training set $Q$ . . . . .	77
A-2	Normalized frequency domain envelope for the input signals in training set $Q$	78
A-3	Normalized frequency domain envelope for the input signals in validation set $R$	79
A-4	The influence of the input sets on the models $\mathcal{U}_1$ . . . . .	80
A-5	The influence of the input sets on the models $\mathcal{U}_2$ . . . . .	80
A-6	The influence of the input sets on the models $\mathcal{U}_3$ . . . . .	81

A-7	All models found with genetic programming dependent on $\mathcal{R}_{r1}$ . . . . .	81
A-8	All models found with genetic programming dependent on $\mathcal{R}_{r2}$ . . . . .	82
A-9	All models found with genetic programming dependent on $\mathcal{R}_{r3}$ . . . . .	82
A-10	Comparing the output of the trivial model with the target data $u_2^*$ . . . . .	83
A-11	Comparing the output of the trivial model with the target data $u_3^*$ . . . . .	83
A-12	The output of the trivial models $\mathcal{U}_{i,triv}$ and $\mathcal{U}_{i,IS3}$ . . . . .	84
A-13	Comparing the compensation signals for $u_2$ for setpoint signal 1 & 2 . . . . .	86
A-14	Comparing the compensation signals for $u_2$ for setpoint signal 1 & 2 . . . . .	86
B-1	The influence of the hidden layers on the accuracy of the $\tanh$ neural networks trained on data set $Q_1$ . . . . .	88
B-2	The influence of the hidden layers on the accuracy of the $\tanh$ neural networks trained on data set $Q_2$ . . . . .	89
B-3	The influence of the hidden layers on the accuracy of the $\tanh$ neural networks trained on data set $Q_3$ . . . . .	90
B-4	The influence of the hidden layers on the accuracy of the $\tanh$ neural networks trained on data set $Q_4$ . . . . .	91
B-5	The influence of the training data sets on the accuracy of the $\tanh$ networks $\mathcal{U}_1$	92
B-6	The influence of the training data sets on the accuracy of the $\tanh$ networks $\mathcal{U}_2$	93
B-7	The influence of the training data sets on the accuracy of the $\tanh$ networks $\mathcal{U}_3$	94
B-8	The influence of the hidden layers on the accuracy of the $ReLU$ neural networks trained on data set $Q_1$ . . . . .	95
B-9	The influence of the hidden layers on the accuracy of the $ReLU$ neural networks trained on data set $Q_2$ . . . . .	96
B-10	The influence of the hidden layers on the accuracy of the $ReLU$ neural networks trained on data set $Q_3$ . . . . .	97
B-11	The influence of the hidden layers on the accuracy of the $ReLU$ neural networks trained on data set $Q_4$ . . . . .	98
B-12	The influence of the training data sets on the accuracy of the ReLU networks $\mathcal{U}_1$	99
B-13	The influence of the training data sets on the accuracy of the ReLU networks $\mathcal{U}_2$	100
B-14	The influence of the training data sets on the accuracy of the ReLU networks $\mathcal{U}_3$	101
B-15	An overview of the influence of the training data on the performance of the neural networks . . . . .	102
B-16	The influence of the hidden layers on the performance of $\tanh$ controllers for signals in the training data set $Q$ . . . . .	103
B-17	The influence of the hidden layers on the performance of $\tanh$ controllers for signals in set $R_3$ . . . . .	103
B-18	The influence of the hidden layers on the performance of $\tanh$ controllers for signals in validation data set $R$ . . . . .	104
B-19	The influence of the training data on the performance of the $\tanh$ controllers on signals from the training data set $Q$ . . . . .	104
B-20	The influence of the training data on the performance of the $\tanh$ controllers on signals from set $R_3$ . . . . .	105
B-21	The influence of the training data on the performance of the $\tanh$ controllers on signals from validation set $R$ . . . . .	105

B-22	The influence of the hidden layers on the performance of the ReLU controllers on signals from training set $Q$ . . . . .	106
B-23	The influence of the hidden layers on the performance of the ReLU controllers on signals from set $R_3$ . . . . .	106
B-24	The influence of the hidden layers on the performance of the ReLU controllers on signals from validation set $R$ . . . . .	107
B-25	The influence of the training data on the performance of the ReLU controllers on signals from training set $Q$ . . . . .	107
B-26	The influence of the training data on the performance of the ReLU controllers on signals from set $R_3$ . . . . .	108
B-27	The influence of the training data on the performance of the ReLU controllers on signals from validation set $R$ . . . . .	108
B-28	Analyzing the output of the tanh neural network $\mathcal{U}_1$ with 3 hidden layers for signals outside the training set . . . . .	109
B-29	Analyzing the output of the tanh neural network $\mathcal{U}_2$ with 3 hidden layers for signals outside the training set . . . . .	109
B-30	Analyzing the output of the tanh neural network $\mathcal{U}_3$ with 3 hidden layers for signals outside the training set . . . . .	110



---

## List of Tables

3-1	An overview of the collection of input vectors used for the genetic programming experiment . . . . .	25
3-2	An overview of the sets of mathematical operators used in the genetic programming experiments . . . . .	26
3-3	An overview of the varying parameters in the genetic programming experiment	26
3-4	An overview of the training data provided to genetic programming . . . . .	27
3-5	An overview of main settings in the DataModeler tool . . . . .	28
3-6	An overview of all measures used to present the results of the genetic programming experiments . . . . .	29
3-7	The set of trivial models dependent on $\mathcal{R}_{r1}$ . . . . .	30
3-8	Similar expressions found for experiments with target data $u_2^*$ . . . . .	32
3-9	The genetic programming controller $\mathcal{U}_{IS1}$ . . . . .	35
3-10	The genetic programming controller $\mathcal{U}_{IS2}$ . . . . .	35
3-11	The genetic programming controller $\mathcal{U}_{IS3}$ . . . . .	35
3-12	Information about the chosen models for the controllers $\mathcal{U}_{IS1}, \mathcal{U}_{IS2}$ and $\mathcal{U}_{IS3}$	35
3-13	The minimum and maximum reduction for the controllers found with genetic programming . . . . .	38
3-14	An overview of the relevant input variables per target dataset . . . . .	40
4-1	Overview of the used network sizes in the neural networks experiment . . . . .	43
4-2	An overview of the amount of data per training dataset used in the neural network experiment . . . . .	45
4-3	An overview of varying parameters in the neural networks experiment . . . . .	45
4-4	An overview of input variables per target dataset used in the neural networks experiment . . . . .	45
4-5	Parameters used for the training process in the neural networks experiment .	47
4-6	An overview of all measures used to present the results of the neural networks experiments . . . . .	47
5-1	An overview of all measures used to present the comparison between the controllers . . . . .	58

---

5-2	An overview of the maximum, mean and minimum percentage of reduction in $\mathcal{Z}_R$ for each controller . . . . .	59
5-3	The memory consumption of the three feedforward controllers . . . . .	63
5-4	The guaranteed output bounds found with SHERLOCK for the ReLU controller with respect to its measured output bounds . . . . .	64
7-1	An overview of the comparison for the three controller types on the four qualities	76
A-1	The Pareto front of the models $\mathcal{U}_1$ depending on $\mathcal{R}_{r1}$ . . . . .	85

---

# Preface and Acknowledgements

I never was the stereo-type mechanical engineering kid that knew all car brands and diesel motor details. However, as a teenager I was good in maths and science, which led to the choice of mechanical engineering as bachelor. During my bachelor I did my minor in biomedical engineering at the UPC in Barcelona, which led to the decision to do my bachelor end project in biomedical engineering as well. Nevertheless, I never continued in this field. I discovered that my drive for engineering comes from the tackling challenges. It is not in what I build, but rather in what puzzles to solve to get there. This motivated my choice for a master in Systems and Control, which is to me a more abstract field closer to mathematics. Besides that, the challenge the master offers appealed to me. After a quick introduction to control by my colleague Yvo I was convinced. DCSC was the place I was going to learn the coming years!

For this final thesis I had the opportunity to help with the design process for the new machine at ASML and to contribute to the CADUSY project, for which I am grateful. Cees, thank you for your guidance during my thesis; your office door was always open for me! Manual, thanks for the supervision and the help with the realization of this work. Koos and Joost, thanks for your assistance from ASML.

I want to thank both mom and dad, for their never-ending support throughout my life and especially during the final stages of my study. Thanks Tess and Femke for putting me with both feet on the ground whenever I was talking about those weird algorithms at the dinner table. Thank you Nynke, for your love and undeniable help throughout this thesis and the most part of my study. Coen and Wiegert, thanks for the open discussions and laughs during coffee breaks.



---

# Chapter 1

---

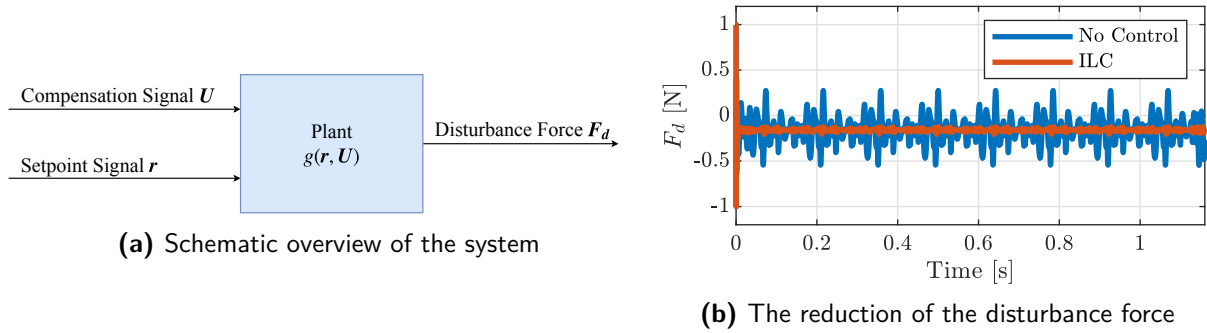
## Introduction

### 1-1 Motivation

In 1965, a paper was published observing the rate of development for processors. Gordon Moore observed that the number of components per integrated circuit has been doubling every year and he predicted it will continue to do so for the next decade [1]. A decade later he revised his statement and concluded that the numbers would double roughly every two years [2]. Up onto today, this observation, often referred to as Moore's law, has been used as roadmap for progress in the chip-making industry [3]. In this day and age minimal sized, powerful processors are critical in many of our day to day activities. The interest in processor power and high-speed memory for an affordable price has not been tempered and pushes the industry to develop more advanced computer chips [3].

In this thesis, a challenge is tackled that arises at the design of a new lithography machine which supports nano meter accuracy and reaches high productivity by handling fast accelerations. Within the chip-making machine, a periodic motion system is designed to position an essential component. The periodic motion system is actuated by an electronically commuted motor. The electromagnetism, commutation, the dynamics of the amplifier and other parasitic effects are non linear of nature. Furthermore, the model is higher dimensional resulting in a non-linear MIMO system. The system is controlled by active servo control to obtain sufficient reference tracking performance. However, the controller generates a force which excites system dynamics and disturbs the high precision. Therefore, there is need to attenuate this disturbance force, which is done by feedforward control.

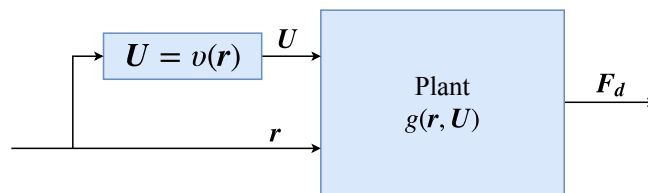
The model of the system is a blackbox closed-loop model including the active servo feedback control. Figure 1-1a visualizes the model considered in this thesis, where the input  $\mathbf{U}$  represents the feedforward control signal and input  $\mathbf{r}$  the setpoint signal, which is tracked by the active servo control. Diverging from traditional control problems is the output of the model, which is the resulting disturbance force acting on the system, which will be referred to as  $\mathbf{F}_d$ .



**Figure 1-1:** The control loop of the system and an example of a reduced compensation signal

Iterative Learning Control (ILC) [4] is a method that learns control signals from previous executions and obtains optimal performance for systems that are of repetitive nature. Feedback control in combination with ILC based feedforward control has shown to obtain excellent tracking performances [5–8]. By combining feedback servo control and feedforward signals learned with ILC, the disturbance is reduced and the performance requirements are met. Figure 1-1b illustrates the reduction of the disturbance force, where the signal with and without ILC is given.

For each setpoint signal a compensation feedforward signal has to be learned on forehand via ILC. Computing the ILC signals for all setpoint signals executable by the machine is not viable. The signals have to be learned on the machine, resulting in downtime, which makes learning an expensive process. However, if a mapping  $v(\cdot)$  is known that describes the relation between the compensation signal  $\mathbf{U}$  and the setpoint signal  $\mathbf{r}$ , there is no need to learn the compensation signals a priori. In other words, the compensation signal is now given by the output of the feedforward controller,  $\mathbf{U} = v(\mathbf{r})$ . The control scheme is given in Figure 1-2.



**Figure 1-2:** Control loop including the feedforward controller

The goal of this study is to design a feedforward controller  $v(\cdot)$  that reduces the disturbance for all relevant and executable setpoint signals, therefore generalizing the feedforward compensation signals. For the design, a data set of setpoint signals with their corresponding optimal compensation signals is provided. A larger set consisting of relevant setpoint signals is also given, which is used to validate the feedforward controller performance. Furthermore, a nonlinear MIMO closed loop model of the system that takes as input a compensation signal and a setpoint signal and returns as output the resulting disturbance force is provided. The problem statement and the approach of the problem are formally defined in the following section.

## 1-2 Problem statement

Let the machine be represented by the nonlinear function  $g(\mathbf{r}, \mathbf{U})$  with  $g : \mathbb{R}^N \times \mathbb{R}^{N \times 3} \rightarrow \mathbb{R}^N$ . Here,  $\mathbf{r}$  represents a setpoint signal  $\mathbf{r} \in \mathbb{R}^N$  and  $\mathbf{U}$  represents a compensation signal,

$$\mathbf{U} \in \mathbb{R}^{N \times 3}. \text{ Each signal has } N \text{ samples, i.e. } \mathbf{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_N \end{bmatrix} \text{ and } \mathbf{U} = \begin{bmatrix} u_{1,1} & u_{2,1} & u_{3,1} \\ \vdots & \vdots & \vdots \\ u_{2,N} & u_{2,N} & u_{3,N} \end{bmatrix}.$$

Let  $P$  be the set of all relevant executable setpoint signals by the machine, which is defined in more detail in section 2-1. Furthermore, let us denote the function space as  $\mathcal{S}$ , the set existing of of all possible functions created from all binary operations (e.g.  $+$ ,  $-$ ,  $\times$ ) and mathematical functions (e.g.  $\sin$ ,  $\cos$ ,  $e^{(\cdot)}$ ).

From the physical insights of the machine, we can expect that the compensation signals are dependent not only on the current sample but also previous samples in the setpoint signal. Therefore, the matrix  $\mathcal{R}_r \in \mathbb{R}^{N \times n}$  with  $n \in \mathbb{N}$  is introduced, containing all input variables available to the feedforward controller directly computed from information in  $\mathbf{r}$ . These input variables are calculated as  $\mathcal{R}_r = \omega(\mathbf{r})$  with  $\omega : \mathbb{R}^N \rightarrow \mathbb{R}^{N \times n}$ . Examples of signals in  $\mathcal{R}_r$  are numerical approximations  $\dot{\mathbf{r}}$ ,  $\ddot{\mathbf{r}}$  or shifted input vectors  $\mathbf{r}_{k-1}$ ,  $\mathbf{r}_{k+3}$ .

Furthermore, let  $\mathbf{R}_{rj} \in \mathbb{R}^n$  be a row vector representing the  $j^{\text{th}}$  sample of all input signals in  $\mathcal{R}_r$  for any  $j \in [1, N]$ , given as:

$$\mathcal{R}_r = \begin{bmatrix} \mathbf{R}_{r1} \\ \vdots \\ \mathbf{R}_{rN} \end{bmatrix} \quad (1-1)$$

Now let  $v : \mathbb{R}^n \rightarrow \mathbb{R}^3$  be the feedforward controller taking one sample of each input signal in  $\mathcal{R}_r$ . We define function  $\mathcal{U} : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  as a function taking all  $N$  samples of each input signal in  $\mathcal{R}_r$ , returning a full signal, as:

$$\mathbf{U} = \mathcal{U}(\mathcal{R}_r) = \begin{bmatrix} v(\mathbf{R}_{r1}) \\ \vdots \\ v(\mathbf{R}_{rN}) \end{bmatrix} \quad (1-2)$$

The compensation signal consist of three separate signals, i.e.  $\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \mathbf{u}_3]$ . Therefore, the design of the feedforward controller is divided into finding three separate mappings, mapping a combination of input signals from  $\mathcal{R}_r$  to a signal in  $\mathbf{U}$ . The feedforward controller is than given by  $v = [v_1 \ v_2 \ v_3]$  with  $v_i : \mathbb{R}^{m_i} \rightarrow \mathbb{R}$  for  $i = 1, 2, 3$ . Now let  $\boldsymbol{\rho}_{ri} \in \mathbb{R}^{N \times m_i}$  be the input signals for the function  $v_i$ . Furthermore, let  $\mathbf{p}_{rij} \in \mathbb{R}^{m_i}$  with  $j \in [1, N]$  be a row vector containing the  $j^{\text{th}}$  sample of each input signal in  $\boldsymbol{\rho}_{ri}$  given as:

$$\boldsymbol{\rho}_{ri} = \begin{bmatrix} \mathbf{p}_{ri1} \\ \vdots \\ \mathbf{p}_{riN} \end{bmatrix} \quad (1-3)$$

Analogous to Equation 1-2, we can define three functions  $\mathcal{U}_i : \mathbb{R}^{N \times m_i} \rightarrow \mathbb{R}^N$  that take each sample from the input signals  $\boldsymbol{\rho}_{ri}$  and return a vector  $\mathbf{u}_i$  using  $v_i$  as:

$$\mathbf{u}_i = \mathcal{U}_i(\boldsymbol{\rho}_{ri}) = \begin{bmatrix} v_i(\boldsymbol{p}_{ri1}) \\ \vdots \\ v_i(\boldsymbol{p}_{riN}) \end{bmatrix} \text{ for } i = 1, 2, 3 \quad (1-4)$$

The feedforward controller is given by:

$$v(\boldsymbol{p}_{rij}) = [v_1(\boldsymbol{p}_{r1j}) \quad v_2(\boldsymbol{p}_{r2j}) \quad v_3(\boldsymbol{p}_{r3j})] \quad (1-5)$$

for any  $j \in [1, N]$ . This function can be used to create a signal with  $N$  samples, referred to as:

$$\mathcal{U}(\mathcal{R}_r) = [\mathcal{U}_1(\boldsymbol{\rho}_{r1}) \quad \mathcal{U}_2(\boldsymbol{\rho}_{r2}) \quad \mathcal{U}_3(\boldsymbol{\rho}_{r3})] \quad (1-6)$$

We want the output of the feedforward controller to reduce the disturbance force, which is given by function  $g$ . We are looking for a mapping  $\mathcal{U}$  which output minimizes the 2-norm of the output of function  $g$  for all relevant and executable setpoint signals  $\mathbf{r}$  in  $P$ . The problem statement is given in Problem 1-2.1.

**Problem 1-2.1.** *Design feedforward controller  $\mathcal{U}^* : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  such that:*

$$\forall \mathbf{r} \in P, \mathcal{U}^* = \arg \min_{\mathcal{U} \in \mathcal{S}} \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_r))\|_2. \quad (1-7)$$

As it is unlikely that the global optimum in the program space is found [9], the problem statement is relaxed and provided in Problem 1-2.2.

**Problem 1-2.2.** *Design feedforward controller  $\mathcal{U} : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  by satisfying the following equation:*

$$\forall \mathbf{r} \in P, \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_r))\|_2 \leq \epsilon_P \quad (1-8)$$

### 1-2-1 Approach

Problem 1-2.1 is a nonlinear optimization problem where we search directly in the program space. As long as we do not have any knowledge on which signals will reduce the disturbance force, it is devious to find any set of functions  $\mathcal{U}$  that will reduce the 2-norm of output  $g$ .

As was mentioned earlier, ILC is a method that given any  $\mathbf{r} \in P$  is capable of finding an optimal compensation signal  $\mathbf{U}^* \in \mathbb{R}^{N \times 3}$ , hence solving the minimization problem:

$$\mathbf{U}^* = \arg \min_{\mathbf{U}} \|g(\mathbf{r}, \mathbf{U})\|_2 \quad (1-9)$$

We assume that the function  $g$  is continuous as it represents continuous processes in the physical domain. Therefore, we assume that  $\mathbf{U}^*$  is also a set of continuous functions and



that it can be used to return the optimal compensation signal  $\mathbf{U}^*$  for any setpoint signal  $\mathbf{r} \in P$ :

$$\forall \mathbf{r} \in P, \mathbf{U}^* = \mathcal{U}^*(\mathcal{R}_r) \quad (1-10)$$

Based on these observations a data-driven approach is chosen, where the optimal compensation signals is found with ILC and these signals are used to find  $\mathcal{U}^*$  by fitting the output to the optimal compensation signals. By finding mapping  $\mathcal{U}^*$  we generalize the feedforward signal for any setpoint signal in  $P$ .

Given is a dataset consisting of 2-tuples,  $Q = \{(\mathbf{r}_i, \mathbf{U}_i^*)\}_{i=1 \dots M_1}$  with  $M_1 \in \mathbb{N}$  and where  $\forall i \in \{1 \dots M_1\}, \mathbf{r}_i \in P$ . Hence, the problem is approached by finding a mapping  $v^*$  that represents the data in  $Q$  as accurate as possible. Problem 1-2.3 provides the new problem statement.

**Problem 1-2.3.** Design feedforward controller  $\mathcal{U}^* : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  such that:

$$\forall (\mathbf{r}, \mathbf{U}^*) \in Q, \mathbf{U}^* = \arg \min_{\mathcal{U} \in \mathcal{S}} \frac{1}{N} \|\mathbf{U}^* - \mathcal{U}(\mathcal{R}_r)\|_2^2 \quad (1-11)$$

Furthermore, as the feedforward controller is given by three separate functions as  $\mathcal{U} = [\mathcal{U}_1 \ \mathcal{U}_2 \ \mathcal{U}_3]$  where  $\mathcal{U}_i : \mathbb{R}^{n_i} \rightarrow \mathbb{R}$  for  $i = 1, 2, 3$  with  $n_i \leq n \in \mathbb{N}$ , the complete problem statement is given by 3 optimization problems given in Problem 1-2.4.

**Problem 1-2.4.** Design a feedforward controller  $\mathcal{U} : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  defined as:

$$\mathcal{U}^*(\mathcal{R}_r) = [\mathcal{U}_1^*(\boldsymbol{\rho}_{r1}) \ \mathcal{U}_2^*(\boldsymbol{\rho}_{r2}) \ \mathcal{U}_3^*(\boldsymbol{\rho}_{r3})] \quad (1-12)$$

by solving the following three optimization problems:

$$\forall (\mathbf{r}, \mathbf{u}_i^*) \in Q, \mathbf{u}_i^* = \arg \min_{\mathbf{u}_i \in \mathcal{S}} \frac{1}{N} \|\mathbf{u}_i^* - \mathcal{U}_i(\boldsymbol{\rho}_{ri})\|_2^2 \text{ for } i = 1, 2, 3 \quad (1-13)$$

Furthermore, validation set  $R = \{\mathbf{r}_i\}_{i=1 \dots M_2}$  with  $R \subseteq P$ ,  $M_2 \in \mathbb{N}$  and  $M_2 \gg M_1$  is used to validate the performance of the feedforward controller. If the controller works for the setpoint signals in validation set  $R$ , it is assumed to perform for the set of all relevant executable setpoint signals  $P$ . Therefore, we can rewrite the earlier presented Problem 1-2.2 as:

**Problem 1-2.5.** The controller performs sufficiently if the following equation holds:

$$\forall \mathbf{r} \in R, \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_r))\|_2 \leq \epsilon_P \quad (1-14)$$

From a design perspective, there is no value known for  $\epsilon_P$  after which the force is attenuated sufficiently. Setting an upper bound on the 2-norm of the disturbance is conservative as the reduction of the force is dependent on its initial proportion. Therefore, a less conservative bound is introduced, where we demand that the 2-norm of the force after control to be lower than before control. This provides a hard constraint on the design of the feedforward controller, meaning that the feedforward controller performs sufficient if Condition 1-2.1 is satisfied.

**Condition 1-2.1.** *A feedforward controller  $\mathcal{U}$  performs sufficiently if for all signals in validation set  $R$  if the following equation holds:*

$$\forall \mathbf{r} \in R, \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_{\mathbf{r}}))\|_2 < \|g(\mathbf{r}, \mathbf{0})\|_2 \quad (1-15)$$

Here,  $\mathbf{0} \in 0^N$  is a vector of  $N$  samples with only zeros, representing the situation in which no control is applied.

## 1-2-2 State of the art

Given the complexity of the system  $g(\cdot)$  it is assumed that the mapping  $v(\cdot)$  is nonlinear and the problem is classified as nonlinear data-driven modeling.

Traditionally, methods in nonlinear blackbox modeling fix a structure for the model and optimize the parameters within using information available in the data [10] [11]. A general framework for the structure of the function, which summarizes several methods used for discrete blackbox nonlinear modeling is presented by Sjöberg et al. [11]. The structure of most nonlinear blackbox models can be fixed in a general form as:

$$f(\boldsymbol{\phi}_{\mathbf{k}}, \boldsymbol{\theta}) = \sum_{i=1}^m \theta_i g_i(\boldsymbol{\phi}_{\mathbf{k}}) \quad (1-16)$$

Where  $\boldsymbol{\theta}$  refers to all the parameters in the model,  $\boldsymbol{\phi}_{\mathbf{k}}$  refers to the regressor; the vector containing all variables of the model,  $g_i(\cdot)$  a basis function and  $m$  the amount of basis functions. The basis function generally can be expressed as:

$$g_i(\boldsymbol{\phi}_{\mathbf{k}}) = \kappa_i(\boldsymbol{\phi}_{\mathbf{k}}, \beta_i, \gamma_i) \quad (1-17)$$

In this expression,  $\beta_i$  and  $\gamma_i$  are parameters related to different aspects of the basis function. Now how these parameters are related depends on which nonlinear blackbox modelling method is preferred. For feedforward artificial neural networks, these parameters are constructed in a ridge fashion:

$$g_i(\boldsymbol{\phi}_{\mathbf{k}}) = \kappa_i(\boldsymbol{\beta}_i^T \boldsymbol{\phi}_{\mathbf{k}} + \gamma_i) \quad (1-18)$$

Clearly, there are design choices to make. The user has to decide on the choice of regressor and the type of basis function to choose and in what way the parameter  $\beta_i$  and  $\gamma_i$  are related. Also, the number of basis functions has to be determined. Once the model structure is fixed, the parameters  $\boldsymbol{\theta}$  in Equation 1-16 can be optimized. However, there are no strict guidelines for these design choices. The size and type of function structure to use are generally chosen based on the experience of the user or in empirical fashion.

Combining Equation 1-16 Equation 1-18 results in a feedforward network with one hidden layer. Picking a sigmoid function for  $\kappa$  results in sigmoid neural networks. These are known to be universal function approximators [12] with many successive examples in nonlinear modeling and control [13]. The downside of the sigmoid feedforward artificial

neural networks is their blackbox character, resulting in a model that works arbitrarily well, but is not interpretive to the user.

Recently, the Linear Rectifier Unit as activation function is growing in popularity due to their advantages in deep neural networks [14]. ReLU networks can easily become sparse, are cheaper in computation and they do not encounter the vanishing gradient problem [15]. Using ReLU networks as controller comes with the perk of the use of SHERLOCK [16]. This tool allows for a guaranteed range on the output for the network given a set of inputs represented as convex polyhedron by casting the ReLU network as a MILP feasibility problem. Output bounds assure that the controller will never generate a control input that is undesirable. ReLU networks result in a model that is still blackbox of nature, but SHERLOCK provides a bound on the output.

Contradictory, symbolic regression is the type of regression where both model structure and parameters are optimized and the algorithms strive to explain the data with compact, interpretive equations often referred to as a whitebox modeling [17]. It searches in the function space, which is constructed from operations, mathematical functions and parameters. Usually, the user defines from which operations and mathematical functions the space is constructed. Also, the user lays limitations on the parameters, most commonly demanding them to be real valued and within a specific range. The algorithms that can perform symbolic regression are limited. Genetic Programming [18] is an evolutionary inspired non deterministic algorithm, which is capable of searching the function space for functions that explain the in and output data. Generally, a multi-objective variant of the algorithm is used, where the functions are optimized for both fitness and complexity. There are multiple works where Genetic Programming is used to successfully identify nonlinear models that are fitting their target data accurately [19] [20].

### 1-3 Contribution of the thesis

In this thesis a feedforward controller is designed using a data-driven approach, where we explore two options. The first option is the multi-objective evolutionary algorithm genetic programming, which endeavors to return compact equations that approximate the target data by searching directly in the program space. Analytic models are interpretive to the user and allow for traditional verification methods to ensure their correctness [21]. Furthermore, genetic programming will automatically favor input variables from  $\mathcal{R}_r$  and reveals which of these are useful to describe the data. Neural networks are blackbox models that are known to be excellent for data fitting and extrapolate meticulously if sufficient training data is available. The main drawback of a neural network is its blackbox character, which cannot be obviated. The complex nature of neural network models withholds the user to fully understand its behavior, which precludes their general acceptance in the industry [22]. However, it is possible to compute a guaranteed output bound for neural networks with the ReLU activation function by using the SHERLOCK tool. Therefore, two types of neural networks are considered in this thesis.

Three types of feedforward controllers are designed by solving Problem 1-2.4 using two data-driven methods, where the controller performs sufficiently if Condition 1-2.1 holds. These types of models are found with genetic programming, neural networks with the tanh activation function and neural networks with the ReLU activation function.

We will explore the advantages and disadvantages of both methods, which differ in the model they use to fit the data. Genetic programming will return compact and transparent

models whereas the neural network models are complex of nature. Therefore, comparing the methods will present a trade-off between the complexity of a model and its accuracy. In the context of disturbance force reduction, a model with a higher accuracy comes with the advantage of a better controller performance. A lower complexity model comes with numerous advantages, such as transparency, the use of traditional verification methods, their memory storage and computation time. The controller found by the three methods will be compared on four aspects presented below:

- *Disturbance reduction*

The controllers are compared base on their reduction of the disturbance force. The controller design succeeds if Condition Equation 1-14 holds. We compare the controllers on how much they attenuate the force, which gives an indication on which controller performs the best.

- *Model complexity*

A model given by a compact analytic equation gives the user a feel for how the variables influence its output. Using such model as control law is therefore transparent and understandable. We will exploit the advantages of the controller designed with genetic programming, which is given as a whitebox model. In this thesis we can use the whitebox models to acquire an intuition on how the compensation signals behave. This perk is not available for the network controllers, which are given as a complex function. Furthermore, we will compare the memory consumption and the computation time of the controller, which are implementation matters influences by the complexity of a model.

- *Output bound verification*

From a safety perspective, it is favorable to have a controller for which we can guarantee the control output bound for a given compact set of bounded inputs. We will compare the output bounds provided by SHERLOCK with the measured output bounds and discuss the possibilities for output bound verification for the other types of controllers.

- *Training data set*

Gathering training data with ILC is a time-consuming process. Since the signals have to be learned on the machine, the process results in a lot of downtime, which makes the learning expensive. Therefore, a method that uses less data is preferable. We will compare the methods based on the number of data points they demand to find a controller.

## 1-4 Thesis outline

Chapter 2 will present the preliminaries, which is split up in three parts. The first part focuses on the data sets that are used in this work. We will introduce a parameterization of the setpoint signals, which is used throughout the thesis. Furthermore, we will discuss the periodic character of the signals. The second part will introduce the measure that used in this thesis to describe the performance of the cotntrollers. The final part briefly presents the main methods used.

In this work two main experiments are presented from which the results are used to facilitate the comparison between methods. The first experiment focuses on the use of

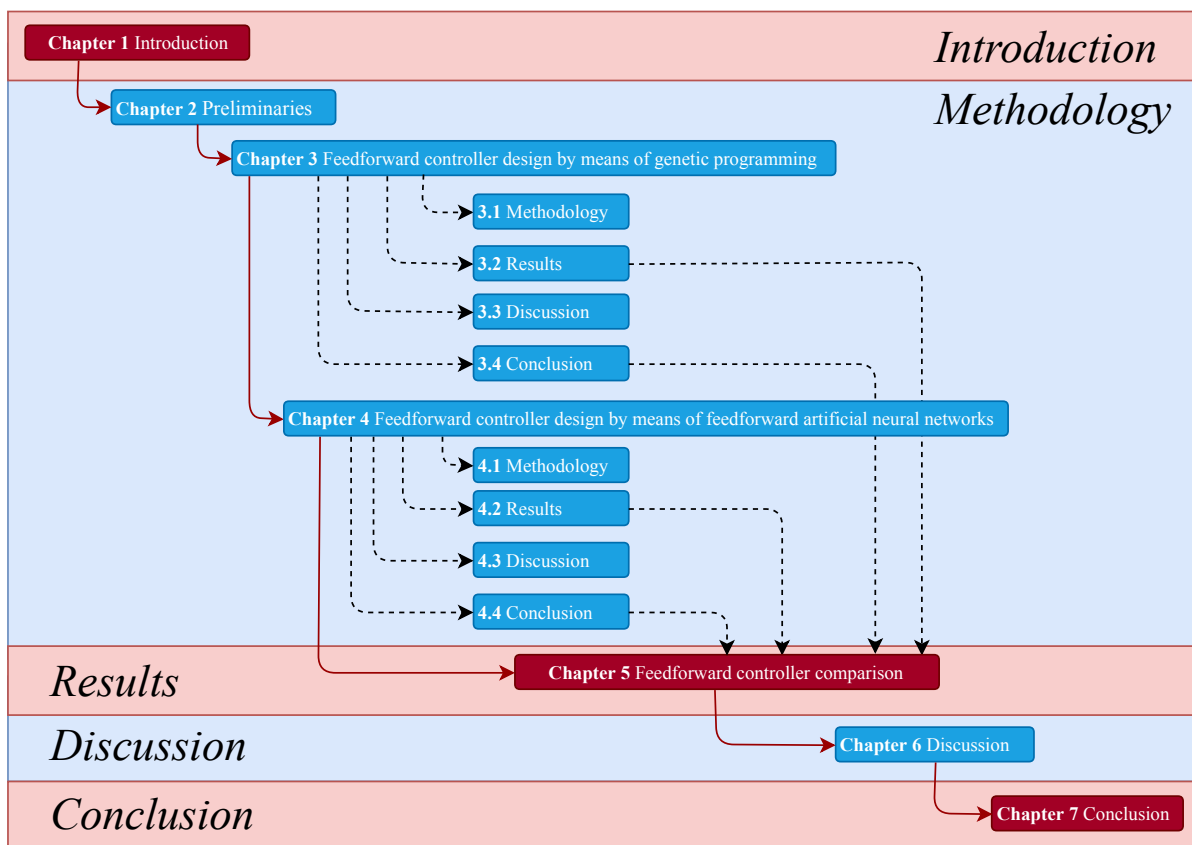
genetic programming for the design of the feedforward controller, where we investigate the input sets that return accurate models. Furthermore, we are looking for a set of input variables that can be used as input for the neural networks. The genetic programming experiment is presented in Chapter 3.

In Chapter 4 we focus on the design of the feedforward controller by means of neural networks, where an experiment is performed to find a sufficient network size and looks at how the training data influences the network performance.

In Chapter 6 the comparison between the genetic programming controller and the network controllers is made, where the controllers follow from the results of the experiments presented in Chapter 3 and 4.

Chapter 6 presents the discussion of the results and the thesis is finalized by the conclusion in Chapter 7.

Figure 1-3 presents a schematic overview of the chapters, where the red arrows present the order of the chapters and the dotted black arrows present the interconnection between the experiments and the comparison.



**Figure 1-3:** Schematic overview of the outline of the thesis

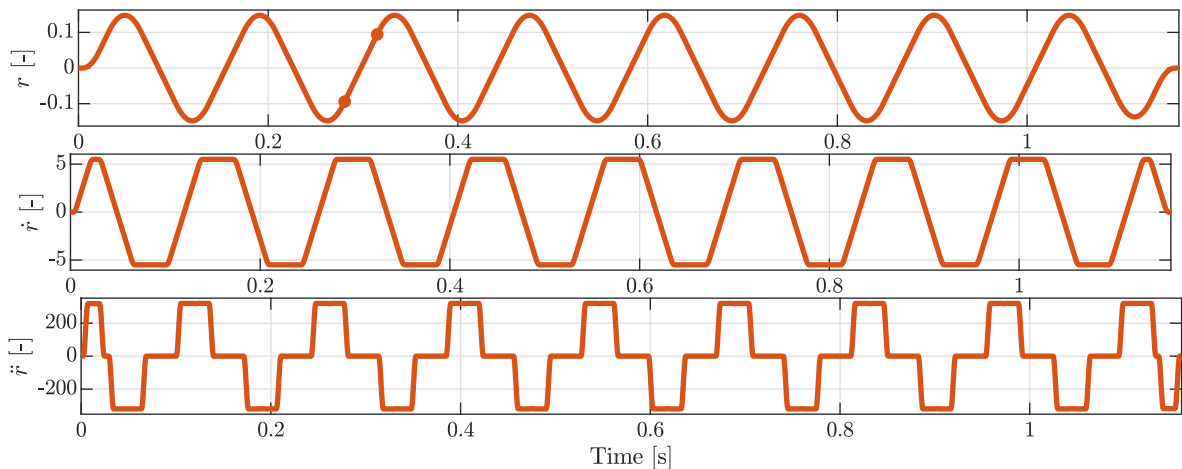


## Preliminaries

In this chapter we will present the background information which is used within the work. We will start by elaborating on the used data sets and introduce a parameterization of the setpoint signals which we can use to formally define the set of all relevant and executable setpoint signals by the machine  $P$ . Moreover, their repetitive character is shown. After that, we formally introduce the set of measures used for the design of the feedforward controller. Finally, we will briefly discuss the algorithms used in this work.

### 2-1 The data sets available for the feedforward controller design

A setpoint signal  $\mathbf{r}$  depends on three core properties, which are given by  $p_1, p_2$  and  $p_3$ . Here,  $p_1$  refers to the length in a period of  $\mathbf{r}$  where the gradient is constant i.e.  $\dot{\mathbf{r}} = c \in \mathbb{R}$ ,  $p_2$  refers to the maximum velocity of the setpoint signal  $p_2 = \max(\dot{\mathbf{r}})$  and  $p_3$  refers to the maximum acceleration of the setpoint signal  $p_3 = \max(\ddot{\mathbf{r}})$ . The parameters are illustrated by Figure 2-1, where  $p_1$  is defined in the upper plot as the distance between the two point.



**Figure 2-1:** An example of a setpoint signal and its velocity and acceleration

Hence, all relevant setpoint signals in this study are defined as a function of these three parameters,  $\mathbf{r} := \pi(p_1, p_2, p_3)$  with  $\pi : \mathbb{R}^3 \rightarrow \mathbb{R}^N$  and  $p_1, p_2, p_3 \in \mathbb{R}$ . The number of

samples in the setpoint signal is dependent on the parameters  $p_1, p_2, p_3$  and given as  $N = \mathcal{N}(p_1, p_2, p_3)$  where  $\mathcal{N} : \mathbb{R}^3 \rightarrow \mathbb{N}$ .

The parameterized space  $\mathcal{P} \subseteq \mathbb{R}^3$  is defined as:

$$\mathcal{P} := \prod_i^3 [\mathcal{L}_i, \mathcal{U}_i] \quad (2-1)$$

Here,  $\mathcal{L}_i, \mathcal{U}_i \in \mathbb{R}$  with  $i = 1, 2, 3$  are bounds on the parameterized setpoint signals provided by the limitation of the machine. Now we can define the set of relevant setpoint signals executable by the system  $P$  in more detail as:

$$P := \{\pi(p_1, p_2, p_3) | (p_1, p_2, p_3) \in \mathcal{P}\} \quad (2-2)$$

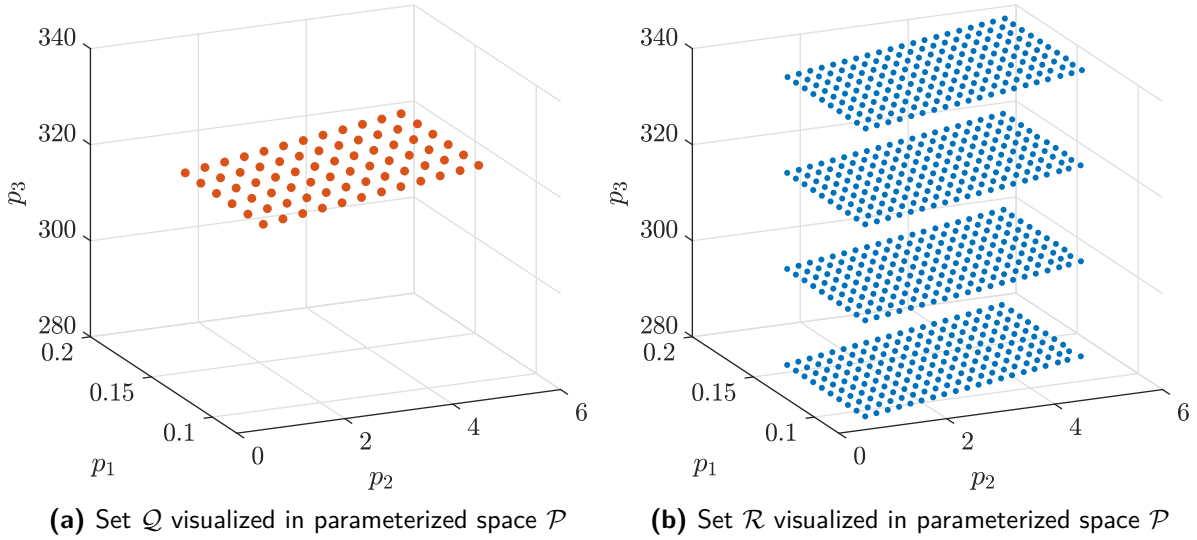
Furthermore, let  $\mathcal{R}$  be the set of all parameters needed to describe the setpoint signals in validation set  $R$ . Similarly, let  $\mathcal{Q}$  be the set of all parameters needed to describe the setpoint signals in training set  $Q$ .

$$\mathcal{R} := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in R\} \quad (2-3)$$

$$\mathcal{Q} := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in Q\} \quad (2-4)$$

$$(2-5)$$

Figure 2-2a visualized set  $\mathcal{Q} \subseteq \mathcal{P}$ , the parameterized setpoint signals in the training set  $Q$ . Figure 2-2b visualizes set  $\mathcal{R} \subseteq \mathcal{P}$ , the parameterized setpoint signals in validation set  $R$ .



**Figure 2-2:** The parameterized setpoint signals in the space  $\mathcal{P}$

The 4 planes visible in the set  $\mathcal{R}$  are defined as separate sets of setpoint signals as:

$$R_1 := \{\pi(p_1, p_2, 280) | (p_1, p_2, 280) \in \mathcal{R}\} \quad (2-6)$$

$$R_2 := \{\pi(p_1, p_2, 300) | (p_1, p_2, 300) \in \mathcal{R}\} \quad (2-7)$$

$$R_3 := \{\pi(p_1, p_2, 320) | (p_1, p_2, 320) \in \mathcal{R}\} \quad (2-8)$$

$$R_4 := \{\pi(p_1, p_2, 340) | (p_1, p_2, 340) \in \mathcal{R}\} \quad (2-9)$$



Similarly, let us define the sets of parameters corresponding with these planes as:

$$\mathcal{R}_1 := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in R_1\} \quad (2-10)$$

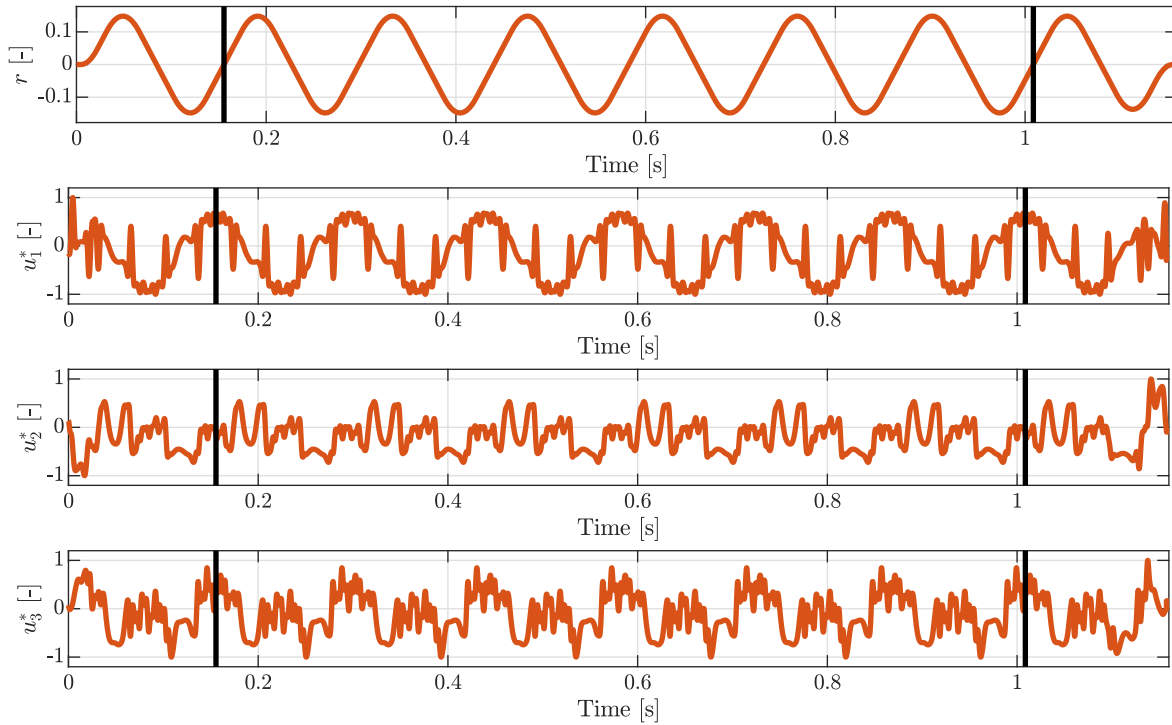
$$\mathcal{R}_2 := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in R_2\} \quad (2-11)$$

$$\mathcal{R}_3 := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in R_3\} \quad (2-12)$$

$$\mathcal{R}_4 := \{(p_1, p_2, p_3) | \pi(p_1, p_2, p_3) \in R_4\} \quad (2-13)$$

Notice that set  $\mathcal{Q}$  lays in plane  $\mathcal{R}_3$  as  $\forall (p_1, p_2, p_3) \in \mathcal{Q}, (p_1, p_2, 320)$ .

Furthermore, the setpoint signals and their corresponding compensation signal are repetitive of nature. To limit the presence of repeated information in the training data, periods of the signals are discarded. Figure 2-3 shows a setpoint signal and its corresponding normalized compensation signal. The signals behave aberrant for the first and last period, visualized with a black horizontal line. Therefore, these periods are never discarded, to capture the complete nature of the signals.



**Figure 2-3:** The repetitive nature of the setpoint signal and its corresponding compensation signals

## 2-2 Measures used for the processing of te results

Several measures are used to get an indication on the performance of the feedforward controller. The investigation of the performance of the feedforward controller is a two-step procedure. First, we analyze how well a mapping fits its target data, which we refer to as the accuracy analysis. Secondly, we combine a set of models to create a feedforward controller and investigate how well it attenuates the force, which is referred to as the feedforward controller analysis. In this section we will first introduce the measures used for the accuracy analysis and after that we will introduce the measures used for the performance analysis of the controller. Finally, we will explain in detail how the quantities of the boxplots used in this work are defined.

### Accuracy analysis

For the analysis of the accuracy of a model, VAF, is used [23, p. 383, eq. 10.22]. This error function is defined as:

$$\text{VAF}(\mathbf{u}_i^*, \mathbf{u}_i) = \max \left( 0, \left( 1 - \frac{\|\mathbf{u}_i^* - \mathbf{u}_i\|_2^2}{\|\mathbf{u}_i^*\|_2^2} \right) \cdot 100 \right) \quad (2-14)$$

where  $\text{VAF} : \mathbb{R}^N \times \mathbb{R}^N \rightarrow [0, 100]$ .  $\mathbf{u}_i \in \mathbb{R}^N$  is the output of the designed model  $\mathcal{U}_i$  and  $\mathbf{u}_i^* \in \mathbb{R}^N$  is the optimal compensation signal found with ILC. A higher VAF corresponds with a better fit. The VAF is a scaled error measure, which facilitates a direct comparison between models representing different target signals.

To analyze the accuracy of a model, the set of VAF values for all signals in the training set  $Q$  is considered, formally defined as:

$$\mathcal{V}_{Q,i} := \left\{ \text{VAF}(\mathbf{u}_i^*, \mathcal{U}_i(\boldsymbol{\rho}_{r_i})) \mid \forall (\mathbf{r}, \mathbf{u}_i^*) \in Q \right\} \text{ for } i = 1, 2, 3 \quad (2-15)$$

Here,  $\mathcal{V}_{Q,i}$  is the set of VAF values for any model  $\mathcal{U}_i$  with its set of input variables  $\boldsymbol{\rho}_{r_i}$  which is explicitly dependent on  $\mathbf{r}$  and  $\mathbf{u}_i^*$  the target data of model  $\mathcal{U}_i$ .

### Feedforward controller analysis

A feedforward controller  $\mathcal{U}$  is build by combining three models  $\mathcal{U}_1, \mathcal{U}_2$  and  $\mathcal{U}_3$ . Running both signals  $\mathbf{r} \in \mathbb{R}^N$  and  $\mathbf{U} \in \mathbb{R}^N$  through the simulation results in the disturbance force  $\mathbf{F}_d \in \mathbb{R}^N$ . The beginning of the disturbance force signal exhibits stabilization behavior. To compare the attenuating, the stabilization behavior is left out of consideration. Therefore, the performance measure is defined as the 2-norm of the disturbance force from the 400<sup>th</sup> sample. Another disturbance measure is introduced which refers to the the amplitude of the disturbance force if no control is applied, defined by running a compensation signal  $\mathbf{0} \in \mathbb{R}^N$  through the simulation and taking the 2-norm of the disturbance force from the 400<sup>th</sup> sample. The performance measures  $\epsilon$ ,  $\epsilon_0$  and  $\epsilon_{ILC}$  are formally defined as:

$$\epsilon(\mathbf{r}) = \|g(\mathbf{r}, \mathcal{U}(\omega(\mathbf{r})))\|_2 \quad (2-16)$$

$$\epsilon_0(\mathbf{r}) = \|g(\mathbf{r}, \mathbf{0})\|_2 \quad (2-17)$$

$$\epsilon_{ILC}(\mathbf{r}, \mathbf{U}^*) = \|g(\mathbf{r}, \mathbf{U}^*)\|_2 \quad (2-18)$$

$$(2-19)$$

As we will test the feedforward controller for all setpoint signals in the training set, the sets of all performance measures for all these signals if control, no control or optimal control is applied are formally defined as:

$$\mathcal{E}_Q = \{\epsilon(\mathbf{r}) \mid \forall \mathbf{r} \in Q\} \quad (2-20)$$

$$\mathcal{E}_0 = \{\epsilon_0(\mathbf{r}) \mid \forall \mathbf{r} \in Q\} \quad (2-21)$$

$$\mathcal{E}_{ILC} = \{\epsilon_{ILC}(\mathbf{r}, \mathbf{U}^*) \mid \forall (\mathbf{r}, \mathbf{U}^*) \in Q\} \quad (2-22)$$

For the performance on the training set, an optimal reduction can be computed with the iterative learning control compensation signals. This is not possible for setpoint signals in the validation set. To give more context to the performance measures found for signals in the validation set, their reduction is expressed in percentages. The new reduction measure  $\zeta$  is defined in Equation 2-23.

$$\zeta(\mathbf{r}) = \left(1 - \frac{\epsilon(\mathbf{r})}{\epsilon_0(\mathbf{r})}\right) 100 \quad (2-23)$$

$$\zeta_{ILC}(\mathbf{r}, \mathbf{U}) = \left(1 - \frac{\epsilon_{ILC}(\mathbf{r}, \mathbf{U})}{\epsilon_0(\mathbf{r})}\right) 100 \quad (2-24)$$

Please notice that a negative  $\zeta$  corresponds with a control action that is not capable of reduction the performance measure. If this happens for any setpoint signal in  $R$ , the controller does not satisfy Condition 1-2.1

Finally, analogous to the sets of performance measures, we can introduce the sets of reduction measures  $\zeta$  as:

$$\mathcal{Z}_{ILC} = \{\zeta(\mathbf{r}, \mathbf{U}) | \forall (\mathbf{r}, \mathbf{U}) \in Q\} \quad (2-25)$$

$$\mathcal{Z}_Q = \{\zeta(\mathbf{r}) | \forall \mathbf{r} \in Q\} \quad (2-26)$$

$$\mathcal{Z}_R = \{\zeta(\mathbf{r}) | \forall \mathbf{r} \in R\} \quad (2-27)$$

$$\mathcal{Z}_{R_3} = \{\zeta(\mathbf{r}) | \forall \mathbf{r} \in R_3\} \quad (2-28)$$

## Boxplots

Furthermore, the results are presented in a boxplot. Here, the median of the data set is presented as vertical red line. The bottom and top edges of the box indicate the 25th and 75th percentiles referred to as  $q_1$  and  $q_3$ , respectively. The whiskers extend to the extreme datapoints that are not considered as outliers. An outlier is plotted as a red plus. A datapoint is considered an outlier if it is greater than  $q_3 + w(q_3 - q_1)$  or lower than  $q_1 - w(q_3 - q_1)$ .  $w$  is the maximum whisker length set at approximately  $\pm 2.7\sigma$  and 99.3 % coverage for a normally distributed dataset.

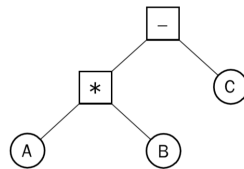
For the 3D boxplot, the median is plotted in black. The upper part of the body of the boxplot refers to 25% data position, whereas the lower part of the body is selected at 75 % of the data. The tails are referring to the extreme of the datasets. Data point are not marked as outliers in this boxplot.

## 2-3 Genetic Programming

Genetic Programming, firstly introduced by Koza [18] is an evolutionary inspired algorithm that is capable of automatically generating computer programs. Given any user-specified building blocks, the algorithm is capable of searching the space of programs composed of these building blocks. It is inspired by Darwin's theory of survival of the fittest. First a generation of solutions is created, the solutions breed and create a new generation, such that it is more likely that more fit solutions will survive in the next

generation and the process is repeated. The algorithm is briefly discussed below. Later on in this section, multiple extensions and different approaches are presented.

Classical GP was originally written in LIPS, one of the oldest high-level programming languages. LIPS works with syntax expressions, which can be visualized as a tree structure. An example of such S expression would be  $-(A * B)C$ , which is visualized in Figure 2-4 and represents the executable  $A * B - C$ . A candidate as tree expression is referred to as the genotype or genome. A tree exists of function block and terminal blocks. As the name suggest, the terminal block, terminates the tree and can only exist at the end of the final node. A function block has either one or two connection nodes and connects with either another function or a terminal. Within the presented example the terminal set is given by  $T = \{A, B, C\}$  and the function set by  $F = \{-, *\}$ . The user has to specify the function set and terminal set and GP can create any combination from the sets. Please note that the terminal set may contain constants and variables, whereas the function set only contains functions.



**Figure 2-4:** An example of a syntax expression

Furthermore, breeding a new generation of candidates is based on three operators which are shortly discussed. *The cross-over operator* selects two candidates in current generation  $G_i$ , randomly selects a subtree from each and swaps them, placing two new offspring into  $G_{i+1}$ .

*The Mutation operator* selects one candidate and randomly mutates a node. If the node is a function, the function is replaced by another randomly chosen function. If the node is a terminal, it is replaced by another random terminal. Constants are also exposed to mutation, which randomly mutates them by selecting a new constant from a predefined constant range.

*The cloning operation* selects one candidate in  $G_i$  and places it into  $G_{i+1}$ .

Each operator has a chance to be picked. These chances are algorithm parameters and are chosen such that they add up to one and only one operator is chosen at a time.

The chance for a candidate to be selected by an operator is based on its fitness, a higher fitness values means a higher chance to be picked.

The algorithm works by initializing  $G_0$  with population size  $M$  candidate trees with Tree Depth  $N_t$ . All candidates are executed and their fitness value is calculated. The candidate with the best fitness is copied to  $G_1$ . Next, the offspring is calculated using the three genetic operators until size  $G_1$  has  $M$  candidates. The candidate trees are executed and their fitness is calculated and the process repeats. The user has to specify a termination criteria; either after an amount of generations that is evolved or a fitness criteria is met.

As candidates with a higher fitness are more likely to survive to the next generation, the average fitness of a generation is more likely to increase. Genetic Programming needs to balance its search between exploitation vs exploration. By implementing random nodes by the mutation operator, the algorithm explores other unseen parts of the search space, which can help to escape local optima. Since the algorithm was created, many

improvements are made, for an extensive overview of the algorithm, the reader is referred to [17]

In this thesis we use the tool DataModeler from Evolved Analytics, which is a state-of-the-art genetic programming tool implemented in Mathematica. We use the ClassicGP and leave most settings at its default value. This implements a multi-objective optimization algorithm, which returns a subset of models that are rated both on their modeling objectives; complexity and fitness. Throughout the experiments, two extra secondary modeling objectives are considered in the multi-objective optimization, which are used within the run, but are not provided in the final results. Model age is a quantity of the model that keeps track on how long the model has been in the selection [24]. By including ModelAge as secondary objective, we deviate the balance between exploitation-exploration more towards exploration, which is favorable for long runs to promote continuing new innovations. New models will have their model age set to zero. Mutating a model or creating a new model with crossover will increase its ModelAge. The second secondary modeling objective is the dimensionality of the model, which gives the number of unique variables in the model. This promotes models that have minimal number of different variables.

Furthermore, the implementation of genetic programming in the DataModeler automatically takes the  $H$  best functions closest to the four-dimensional Pareto front and copy these to the next generation. This ensures the persistence of fit models across multiple generations. The other  $M - H$  models in the generation are created with three evolutionary operators, which operates on models that are selected with tournament selection.  $G$  models are randomly sampled from the generation, from which the Pareto-optimal model is used to create the new generation. The algorithm uses cross-over and implements two types of subtree mutation operators. MutateSubtreeOperator tends to increase the complexity of the model after mutation, while the DepthPreservingSubtreeMutation strives to produce a less complex genome.

## 2-4 Artificial Neural Networks

To applications of artificial neural networks are classification or nonlinear regression [25]. Classification is the problem of assigning a new observation in the category of others. An example of this application is image recognition. Nonlinear regression is the problem type where a model tries to fit its observational data as accurate as possible, where the output is a nonlinear combinations of its input variables.

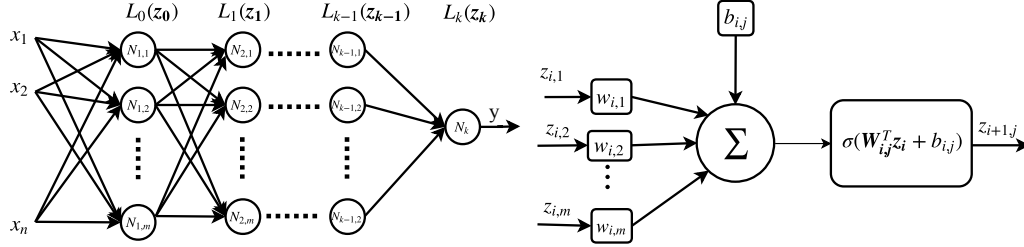
Over the years several types of neural networks have arose. Feedforward artificial neural networks is the most basic neural network type, where the output of the model is directly given by a nonlinear transformation of its inputs. Another example of a neural network type is recurrent artificial neural networks, which take their previous output as their input, therefore modeling dynamics [25].

In this thesis we will focus on multi-layered feedforward artificial neural networks for nonlinear regression. We will briefly highlight the structure of a neural networks, after which we will explain the training algorithm used to optimize the model.

### 2-4-1 Framework

The framework of an feedforward artificial neural network is well known [16] [25] and is presented by considering the first function  $v_1(\mathbf{p}_{r,1})$  in the feedforward controller. Let

$v_1$  be a neural network with  $n$  inputs  $\mathbf{p}_{r1} \in \mathbb{R}^n$  and one output  $u_1 \in \mathbb{R}$ . The networks considered are build from  $k$  hidden layers and  $m$  neurons per layer. A schematic overview of a feedforward artificial neural network is given in Figure 2-5.



**Figure 2-5:** Schematic representation of a feedforward neural networks

Each neuron  $N_{i,j}$  represents a function which output is a scaled summation of its inputs shifted by a bias passed through a nonlinear activation function  $\sigma(\cdot)$ . Now let us the following vectors, which represent the input signals for the  $i^{th}$  layer in the network, where  $i \in [1, k]$ .

$$\mathbf{z}_0 = \begin{bmatrix} p_{r1,1} \\ \vdots \\ p_{r1,n} \end{bmatrix}, \mathbf{z}_i = \begin{bmatrix} z_{i,1} \\ \vdots \\ z_{i,m} \end{bmatrix} \quad (2-29)$$

Now we can formally define the neurons. The output of neuron that is connected by is given by Equation 2-30 and the output by the final neuron  $N_k$  is given as Equation 2-31.

$$z_{i+1,j} = \sigma(\mathbf{W}_{i,j}^T \mathbf{z}_i + b) \quad (2-30)$$

$$z_{k+1} = y = \sigma(\mathbf{W}_k^T \mathbf{z}_k + b_k) \quad (2-31)$$

The weight matrices for neurons in the first, last or any layer in between are given by:

$$\mathbf{W}_0 = \begin{bmatrix} w_{0,1} \\ \vdots \\ w_{0,n} \end{bmatrix}, \mathbf{W}_{i,j} = \begin{bmatrix} w_{i,1} \\ \vdots \\ w_{i,m} \end{bmatrix}, \mathbf{W}_k = \begin{bmatrix} w_{k,1} \\ \vdots \\ w_{k,m} \end{bmatrix} \quad (2-32)$$

With  $i \in [1, k-1]$  and  $j \in [1, m]$ . A neuron from any layer  $L_1, \dots, L_k$  is visualized in Figure 2-5. Visualizing a neuron for the first layer  $L_0$  differs only in the number of inputs and corresponding inputs, which changes to  $n$  instead of  $m$ .

We can now define the output of a layer by combing Equation 2-30 and Equation 2-29 as:

$$L_0(\mathbf{z}_0) = \begin{bmatrix} \sigma(\mathbf{W}_{0,1}^T \mathbf{z}_0 + b_{0,1}) \\ \vdots \\ \sigma(\mathbf{W}_{0,n}^T \mathbf{z}_0 + b_{0,m}) \end{bmatrix}, L_i(\mathbf{z}_i) = \begin{bmatrix} \sigma(\mathbf{W}_{i,1}^T \mathbf{z}_i + b_{i,1}) \\ \vdots \\ \sigma(\mathbf{W}_{i,m}^T \mathbf{z}_i + b_{i,m}) \end{bmatrix}, L_k(\mathbf{z}_k) = \sigma(\mathbf{W}_k^T \mathbf{z}_k + b_k) \quad (2-33)$$

With again  $i \in [1, k - 1]$ . Here,  $L_0$  refers to the first layer,  $L_i$  to any layer in between the first and last layer and  $L_k$  refers to the last layer. Now a neural network can be written as a nested functions of layers as:

$$v_1(\mathbf{z}_0) = L_0(\mathbf{z}_0) \circ \dots \circ L_k(\mathbf{z}_k) \quad (2-34)$$

Furthermore, we introduce the following notation for the weights and biases of the neural network as:

$$\mathbf{W}_i = \begin{bmatrix} \mathbf{W}_{i,1}^T \\ \vdots \\ \mathbf{W}_{i,m}^T \end{bmatrix}, \mathbf{B}_i = \begin{bmatrix} b_{i,1} \\ \vdots \\ b_{i,m} \end{bmatrix} \quad (2-35)$$

With  $i \in [0, k - 1]$

Hence, a neural network can be described by a set of weight matrices and biases:

$$(\mathbf{W}_0, \mathbf{B}_0), \dots, (\mathbf{W}_{k-1}, \mathbf{B}_{k-1}), (\mathbf{W}_k, b_k) \quad (2-36)$$

here,  $(\mathbf{W}_0, \mathbf{B}_0)$  are matrices  $N \times n$  and  $N \times 1$  connecting the input with the first layer,  $(\mathbf{W}_i, \mathbf{B}_i)$  with  $i \in [1, k - 1]$  connecting layer  $L_i$  with  $L_{i+1}$  are matrices  $N \times N$  and  $N \times 1$ . Finally,  $(\mathbf{W}_k, b_k)$  connects the last layer with the final output with size  $N \times 1$  and  $b_k = \mathbb{R}$

### Activation function

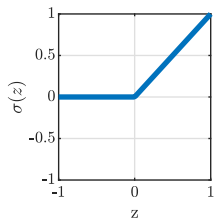
In this thesis, three types of activation functions are used. We will use the hyperbolic tangent-sigmoid or tanh activation function given by Equation 2-38 and visualized in Figure 2-6b. The network build form tanh activation functions takes a linear activation function as final activation function given in Equation 2-39 and visualized in Figure 2-6c.

The second feedforward artificial neural network uses the rectifier linear unit (ReLU) activation function given in Equation 2-37 and visualized in Figure 2-6a. The network uses a ReLU activation function as final activation function.

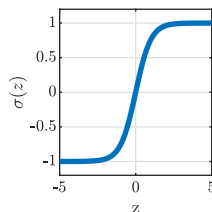
$$\sigma(z) = \max(0, z) \quad (2-37)$$

$$\sigma(z) = \frac{2}{1 + e^{-2z}} - 1 = \tanh(z) \quad (2-38)$$

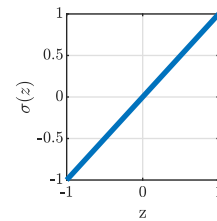
$$\sigma(z) = z \quad (2-39)$$



(a) The ReLU activation function



(b) The tanh activation function



(c) The linear activation function

**Figure 2-6:** An overview of all activation functions considered in this study

### 2-4-2 Training of a neural network

Now that we have discussed the framework, we will continue by how the training of a neural network works. The weights and biases given in Equation 2-36 are adjusted such that the error between the target data and the output is minimized.

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^p} \|\mathbf{u}_1^* - \mathcal{U}_1(\boldsymbol{\rho}_{r1}, \theta)\|_2^2 \quad (2-40)$$

Equation 2-40 is the non-linear optimization problem we solve when training the neural network. In this thesis, we train the neural networks using the Levenberg-Marquardt [26] training algorithm, where we back propagate the error through the network by adapting the errors.

## 2-5 SHERLOCK

SHERLOCK [16] is a tool that will provide output bounds of a feedforward artificial neural network with ReLU activation functions for a given polyhedron set of input values. Given a bounded set of input constrains  $P : \mathbf{A}\boldsymbol{\rho}_{r1} \leq \mathbf{b}$ , Sherlock will provide the following guarantee:

an output  $[l, u]$  such that  $\forall \boldsymbol{\rho}_{r1} \in P : \mathcal{U}_1(\boldsymbol{\rho}_{r1}) \in [l, u]$ .

$$\left( \max_{\boldsymbol{\rho}_{r1} \in P} \mathcal{U}_1(\boldsymbol{\rho}_{r1}) \geq u - \delta \right), \left( \min_{\boldsymbol{\rho}_{r1} \in P} \mathcal{U}_1(\boldsymbol{\rho}_{r1}) \leq l + \delta \right) \quad (2-41)$$

This bound is provided by solving the optimization problem with a combination of local search and a global search, by solving a MILP feasibility problem. The way in which a ReLU network is cast as a MILP feasibility is roughly presented below. For an extended presentation of the tool, the reader is referred to the paper [16]. The MILP feasibility problem is given as:

$$\begin{aligned} \min_x \quad & 0 \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{w} \leq \mathbf{c} \\ & \mathbf{x} \in \mathbb{R}^n, \mathbf{w} \in \mathbb{Z}^m \end{aligned}$$

The piecewise character of the ReLU network is encoded as a set of constraints containing binary variables with  $i \in [0, k - 1]$  as:

$$C_0 : \mathbf{A}\boldsymbol{\rho}_r \leq \mathbf{b} \quad (2-42)$$

$$C_{i+1} : \begin{cases} \mathbf{z}_{i+1} \geq \mathbf{W}_i \mathbf{z}_i + \mathbf{B}_i \\ \mathbf{z}_{i+1} \leq \mathbf{W}_i \mathbf{z}_i + \mathbf{B}_i + M \mathbf{t}_{i+1} \\ \mathbf{z}_{i+1} \geq 0 \\ \mathbf{z}_{i+1} \leq M(1 - \mathbf{t}_{i+1}) \end{cases} \quad (2-43)$$

$$C_{k+1} : \mathbf{y} = \mathbf{W}_k \mathbf{z}_k + \mathbf{b}_k \quad (2-44)$$

The problem is now cast as a MILP feasibility problem as:



$$\begin{aligned} \min_x \quad & 0 \\ \text{s.t.} \quad & C_0, \dots, C_{k+1} \\ & \mathbf{x}, \mathbf{z}_1, \dots, \mathbf{z}_k, y \in \mathbb{R}^{kN+n+1} \\ & \mathbf{t}_1, \dots, \mathbf{t}_{k-1} \in \mathbb{Z}^{(k-1)N} \end{aligned} \tag{2-45}$$

Define 2 norm



# Feedforward controller design by means of genetic programming

In this chapter we will elaborate on the design of the feedforward control using genetic programming. We approach the design by finding three models representing all three signals in the optimal compensation signal learned with ILC. In this chapter we will construct the feedforward controller by solving the following three minimization problems:

**Problem 3-0.1.** *Design a feedforward controller  $\mathcal{U} : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  defined as:*

$$\mathcal{U}^*(\mathcal{R}_{rp}) = [\mathcal{U}_1^*(\rho_{r1p}) \quad \mathcal{U}_2^*(\rho_{r2p}) \quad \mathcal{U}_3^*(\rho_{r3p})] \quad (3-1)$$

by solving the following three optimization problems:

$$\forall(\mathbf{r}, \mathbf{u}_i^*) \in Q_{GP} : \mathcal{U}_i^* = \arg \min_{\mathcal{U}_i \in S_p} \frac{1}{N} \|\mathbf{u}_i^* - \mathcal{U}_i(\rho_{rip})\|_2^2 \text{ for } i = 1, 2, 3 \quad (3-2)$$

Here,  $N$  is the number of samples in a signal,  $\mathbf{u}_1^*$ ,  $\mathbf{u}_2^*$  and  $\mathbf{u}_3^*$  represent the optimal compensation signals,  $\mathcal{U}_1^*$ ,  $\mathcal{U}_2^*$  and  $\mathcal{U}_3^*$  the models we are searching for with genetic programming,  $\rho_{r1p}$ ,  $\rho_{r2p}$  and  $\rho_{r3p}$  the input variables per model for which we know  $\rho_{rip} \in \mathcal{R}_{rp}$ ,  $Q_{GP}$  the training set we use for genetic programming, which is a subset of all training data,  $Q_{GP} \subseteq Q$  and  $S_q \subseteq \mathcal{S}$ , the programming space we let genetic programming search in, which is always a subset of the program space as defined in the introduction. This search space is constructed from the input variables  $\mathcal{R}_{rp}$  and the set of mathematical operators  $\mathcal{M}_p$  provided to the algorithm. Furthermore,  $q$  and  $p$  are used to indicate the varying parameters provided to the algorithm.

This chapter will explain in detail how the feedforward controller is designed. The first section will focus on the construction and motivation of a set of experiments, where all options and varying parameters are provided. Furthermore, the results of the experiments will be presented, after which a subset of found models is combined as feedforward controllers to see how they perform on both the data in the training set and the data in the validation set. After that, a discussion of the results is presented and the chapter is finalized by concluding on the found observations.

### 3-1 Experimental Set-Up

In this section, we will present the experimental methodology. First, the research questions are motivated and presented. After that, the varying parameters and their motivation are presented in detail followed by the options and algorithm parameters which are kept constant throughout the experiments are provided. Lastly, we will present the framework which we use to examine the results.

#### 3-1-1 Research questions

The feedforward controller is designed by searching for a mapping between the setpoint signal  $\mathbf{r}$  and the compensation signal  $\mathbf{U}^*$ . Since  $\mathbf{r}$  is a signal with multiple samples fully known a priori, any information that is available in the setpoint signal can be used for the mapping, that is  $\mathbf{U} = \mathcal{U}(\mathcal{R}_r)$ . One of the main advantages of genetic programming is its ability to directly search in the program space. The algorithm returns equations that explain the target data using inputs from the set of input variables provided by the user. This means that genetic programming searches in  $\mathcal{R}_{rp} \in \mathcal{R}_r$  with  $\mathcal{R}_{rp} \in \mathbb{R}^{N \times n_p}$ ,  $\mathcal{R}_r \in \mathbb{R}^{N \times n}$  and  $n_p \leq n$ . Therefore, genetic programming can be used as tool to provide an indication for which input variables are useful in a set of input variables. In other words, genetic programming can be used to find a combination of interesting input variables,  $\rho_{rip} \in \mathcal{R}_{rp} \in \mathcal{R}_r$ . The design of the feedforward controller by means of neural networks demands us to fix the structure of the network a priori. Therefore, we have to provide the networks with a set of input variables. The genetic programming experiments give an indication which combination of input variables  $\rho_{rip}$  is favorable.

Genetic programming searches in a space which is constrained by the mathematical operators specified by the user. Accordingly, if the algorithm searches for a model build from mathematical operators that are not available in the search space, it will not succeed. However, providing all mathematical operators comes with two downsides. First of all, it enlarges the search space enormously, which results in a lower possibility that an accurate model is found. Secondly, it comes with the possibility that the algorithm finds a model with a high fit, but is mostly constructed of nonlinear mathematical operators. The user might favor controllers that link better to the understanding of the underlying physical system over controllers with higher fit. In this experiment, we are not directly interested in a best performing set of mathematical operators, but we want to use genetic programming to find a set of models that provides the user with the possibility of picking a result that is understandable and performs sufficiently.

Therefore, a set of experiments is done to answer the following two questions:

1. *Which combination of variables  $\rho_{r1}$ ,  $\rho_{r2}$  and  $\rho_{r3}$  are beneficial for the performance of the feedforward controller?*
2. *Can we use genetic programming to find a set of models from which the user can select a model that is interpretive and has a high fit?*

#### 3-1-2 Varying parameters in the experimental set-up

In this subsection, we will present all parameters we vary throughout the experiment. We will let genetic programming search in a set of program spaces by varying both the input variables  $\mathcal{R}_{rp}$  and the sets of mathematical operators  $\mathcal{M}_p$  for  $p = 1, 2, 3$ .

### Combinations of input variables

The setpoint signal specifies the path, which the periodic motion system will follow. It is therefore known a priori and we are not limited to causal mappings. Three different combinations of input variables are created, referred to as  $\mathcal{R}_{r1}$ ,  $\mathcal{R}_{r2}$  and  $\mathcal{R}_{r3}$ . From the physical meaning of components in the system, we expect that the compensation signals are dependent on derivatives of the setpoint signals. The mass, damping and amplifier dynamics in the system motivates the construction of  $\mathcal{R}_{r1}$ , which consist of the setpoint signal and its derivatives up to the third order. Here, the derivatives are numerical approximations using a central order difference scheme.  $\mathcal{R}_{r2}$  focuses on discrete time, where the setpoint signals are shifted forward and backwards a priori. As the setpoint signal starts and ends at 0 and has no direct physical meaning, the initial conditions which are needed to shift the signals are defined as 0.  $\mathcal{R}_{r3}$  combines set 1 and 2, by computing the numerical approximation of the derivatives and shift these vectors forward and backwards by three steps. An overview for the used combinations of input variables are provided in Table 3-1.

Input combinations	Input variables
$\mathcal{R}_{r1}$	$[r, \dot{r}, \ddot{r}, \dddot{r}]$
$\mathcal{R}_{r2}$	$[r_{k-3}, r_{k-2}, r_{k-1}, r_k, r_{k+1}, r_{k+2}, r_{k+3}]$
$\mathcal{R}_{r3}$	$[r_{k-3}, r_{k-2}, r_{k-1}, r_k, r_{k+1}, r_{k+2}, r_{k+3}, \dot{r}_{k-3}, \dot{r}_{k-2}, \dot{r}_{k-1}, \dot{r}_k, \dot{r}_{k+1}, \dot{r}_{k+2}, \dot{r}_{k+3}, \ddot{r}_{k-3}, \ddot{r}_{k-2}, \ddot{r}_{k-1}, \ddot{r}_k, \ddot{r}_{k+1}, \ddot{r}_{k+2}, \ddot{r}_{k+3}, \ddot{r}_{k-3}, \ddot{r}_{k-2}, \ddot{r}_{k-1}, \ddot{r}_k, \ddot{r}_{k+1}, \ddot{r}_{k+2}, \ddot{r}_{k+3}]$

**Table 3-1:** An overview of the collection of input vectors used for the genetic programming experiment

### Mathematical operators

Three different sets of mathematical operators are used in the experiments, denoted with  $\mathcal{M}_1$ ,  $\mathcal{M}_2$  and  $\mathcal{M}_3$ . The first set  $\mathcal{M}_1$  limits the algorithm to build polynomial functions. The Stone-Weierstrass approximation theorem [27] states that all continuous functions defined on a closed interval can be approximated by a polynomial function, which motivates the use of  $\mathcal{M}_1$ . The compensation signals are coming from processes in the physical setup. Therefore, set  $\mathcal{M}_2$  consists of mathematical operators commonly used to describe classical mechanical systems. Set  $\mathcal{M}_2$  consists of sinusoidal functions, the squared and cube operator, addition, subtraction, division and multiplication. The motivation behind set  $\mathcal{M}_3$  is to extend the search space of the algorithm to see if the target data is explained by complex nonlinear functions. Math set  $\mathcal{M}_3$  extends set  $\mathcal{M}_2$  with 3 nonlinear functions. Also, the squared and cube operator is replaced with the to power operator,  $(\cdot)^N$ , where  $N$  can either be a real valued number or any other function created from mathematical operator in the set. The sigmoid function used is defined as  $\sigma(x) = \frac{x}{1 + |x|}$ . An overview of the sets of mathematical operators used is presented in Table 3-2.

### Target Data

The compensation signal exist of three target data signals,  $\mathbf{U}^* = [\mathbf{u}_1^* \ \mathbf{u}_2^* \ \mathbf{u}_3^*]$  and the feedforward controller is created from three independent models. Here, each signal

Set of mathematical operators	Elements
$\mathcal{M}_1$	$\{+, \times\}$
$\mathcal{M}_2$	$\{+, -, \times, \div, \sin, \cos, (\cdot)^2, (\cdot)^3\}$
$\mathcal{M}_3$	$\{+, -, \times, \div, \sin, \cos, (\cdot)^N, \sigma(\cdot),  \cdot , \tanh(\cdot), e^{(\cdot)}\}$

**Table 3-2:** An overview of the sets of mathematical operators used in the genetic programming experiments

represents a different compensation signal in  $\mathbf{U}$ . Therefore, each experiments is repeated for every signal in the compensation signal.

We perform an experiment in a search space constructed of each combination of input variables and mathematical operators, yielding 9 program spaces  $S_q$  for  $q = 1, \dots, 9$ . As we are looking for models representing each compensation signal in  $\mathbf{U}$ , a total of 27 experiments are considered. An overview of the varying parameters of the experiments are provided in Table 3-3.

Sets of mathematical operators	$\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$
Input variables	$\mathcal{R}_{r1}, \mathcal{R}_{r2}, \mathcal{R}_{r3}$
Target data	$u_1^*, u_2^*, u_3^*$

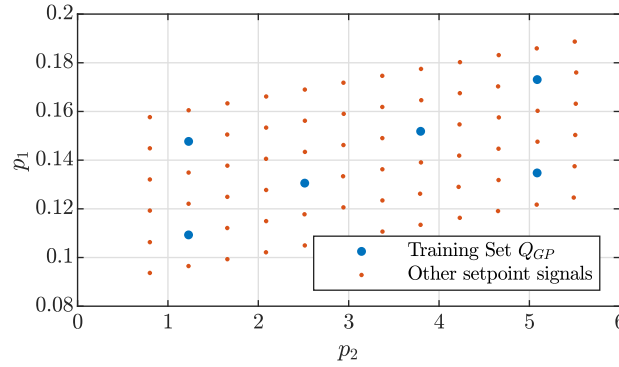
**Table 3-3:** An overview of the varying parameters in the genetic programming experiment

### 3-1-3 Other parameters in the experimental set-up

There are several options and algorithm parameters chosen to perform the experiments. These will be presented in this subsection, where we first explain how we built the data set  $Q_{GP}$  and after that the options used for the genetic programming runs are presented.

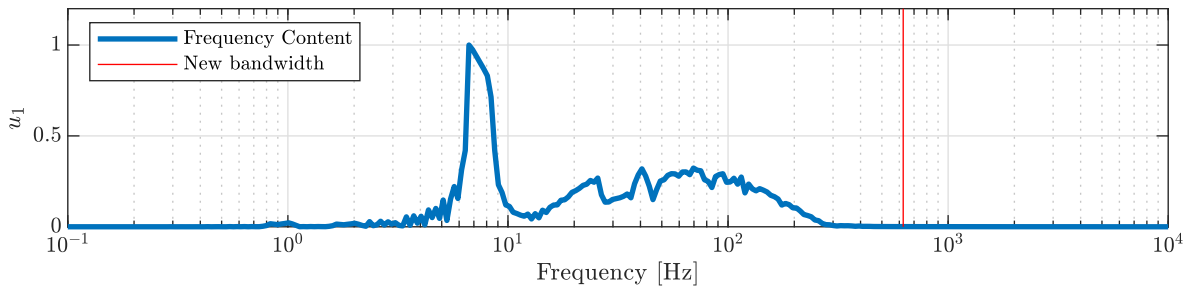
#### Training Data

DataModeler is a commercialized package focusing especially on the implementation of genetic programming. The exact settings and the motivation for this implementation is provided in Table 3-1-3. Unfortunately, DataModeler is only capable of handling limited data within reasonable time. Therefore,  $Q_{GP}$  is designed with the purpose to have most information of  $\mathcal{Q}$  in the training data, while keeping the number of data points minimal. The training set used for genetic programming,  $Q_{GP}$  consist of 6 signals from the training data set  $\mathcal{Q}$ . Signals on the vertices of the data set  $\mathcal{Q}$  are left out of consideration as they contain the most samples. Instead, the corner signals near the vertices are considered. To get more information throughout the space, signals near the center of the set are also added to training data set  $Q_{GP}$ . The position of these 6 signals in space  $\mathcal{P}$  are given in Figure 3-1. Furthermore, the setpoint signals and their corresponding compensation signal are of repetitive nature, as was highlighted in section 2-1. To limit the presence of repetitive information in the training data, only the first and the last period of a setpoint signal and their corresponding compensation signal is used in training set  $Q_{GP}$ .



**Figure 3-1:** The signals used in  $Q_{GP}$  sampled in the parameterized training dataset  $Q$

Furthermore, the training data can be decreased by down sampling the data. To get an insight in the frequency content of the set of signals, the frequency domain envelope is used. The measure can be interpreted as the maximum amplitude per frequency bin for a set of signals and gives an indication on where the maximum frequency content for a set of signals lays. It reveals nothing about how this frequency content is spread throughout the set. The normalized frequency domain envelope of signal  $\mathbf{u}_1^*$  is provided in Figure 3-2. This figure illustrates that the current sampling rate of  $h = 1e^{-4}$  s is superfluous. A slower sampling rate of  $8e^{-4}$  s is chosen, which restricts the bandwidth at 625 Hz. The new bandwidth is illustrated by a red horizontal line in Figure 3-2. The effect of the new sampling rate on the bandwidth of the other sets of signals is presented in Figure A-1 A-2, A-3 in Appendix A .



**Figure 3-2:** The normalized frequency envelope for all signals  $\mathbf{u}_1^*$  in training data set  $Q$

Finally, the dataset is even further reduced by taking 70% of the data by sampling randomly throughout the training dataset  $Q_{GP}$  to facilitate a fair comparison between genetic programming and neural networks, which is done later on in Chapter 5. An overview on the training set is given in Table 3-4.

Data set	Number of signals	Total number of data points
$Q_{GP}$	6	1811

**Table 3-4:** An overview of the training data provided to genetic programming

### Other parameters with genetic programming

In this thesis we use the tool DataModeler [28] from Evolved Analytics, which is a state-of-the-art genetic programming tool implemented in Mathematica. The tool provides the user not only with the use of the algorithm, but comes with a set of possibilities to

guide the user in the data-driven modeling process, varying from data analysis and model selection guidance to the analysis of sensitive variables. The tool comes with a range of possibilities for the algorithm, varying from search strategies to advanced settings which give the user many degrees of freedoms. In this thesis we use the ClassicGP search strategy which implements a multi-objective optimization algorithm and leave most settings at its default value. The main algorithm settings are discussed. The experiments are performed with Mathematica version 11.2. The experiments are conducted on a Intel Xeon CPU E5-1660 v3 3.00GHz using 8 CPU cores. An independent search was done in parallel on each physical core, resulting in 8 parallel runs. The experiments are designed to cover a large part of the search space. Therefore, each experiment uses a population size of 300 models, combined with a total of 1000 generations per evolutionary run. To guide the evolution of the models, the parameters within the in-linear model are optimized using Least Squares. Each model is rated on 4 objectives; Mean Squared Error, Model Complexity, ModelAge, ModelDimensionality, which are explained in section 2-3. Here, the ModelAge and ModelDimensionality are secondary modeling objectives only used in the run and MSE and model complexity the objectives for which the final models are provided to the user. MSE is defined as:

$$\text{MSE}(\mathbf{u}_i^*, \mathbf{u}_i) = \|\mathbf{u}_i^* - \mathbf{u}_i\|_2^2 \quad (3-3)$$

The parameter that expresses the model complexity is defined as the total sum of nodes in all the sub-trees available in the genome of a model. A maximum complexity of the model is set at 1000, which does not constrain the search to find large models. Furthermore, the ratio crossover/mutation was set at 0.9/0.1. Least squares [23] is used as parameter optimization.

An overview of the main choices used in DataModeler are presented in Table 3-5.

Settings	Value
Parameter optimization	Least Squares
Generations per run	1000
Independent evolutions	8
Modeling objectives	MSE, ModelComplexity
Secondary modeling objectives	ModelAge, ModelDimensionality
Maximum allowable complexity	1000
Population size	300
Chance crossover	90 %
Chance mutation	5 %
Chance depth-preserving subtree mutation	5 %

**Table 3-5:** An overview of main settings in the DataModeler tool

### 3-1-4 Processing the results of the genetic programming experiment

DataModeler returns a set of models rated both on their complexity and fitness. From this set of models, 5 % best performing models that are closest to the Pareto front are used for further analysis. The fit of the model is acquired by calculating the MSE between the output of the model and the target data for  $Q_{GP}$ . However, we are interested in how the models perform for all signals in training set  $Q$  and eventually for all the signals in validation set  $R$ . For further analysis, a new fitness value is seized for the subset of



models by computing the MSE for all data in training set  $Q$ . Furthermore, models with parameters lower than  $1e^{-8}$  or higher than  $1e^8$  are left out of consideration, as these are sensitive for marginal changes in the input variables. These large parameters can also lead to precision errors if they are not carefully implemented. A subset of models that are not sensitive nor over-fit for data in data set  $Q$  are considered for further analysis.

Furthermore, different measures are used for the accuracy analysis of the models and the controller performance. An overview of all used measures in section 3-2 are given in Table 3-6 and are defined in section 2-2.

Measure	Interpretation	Formula
VAF	The percentage of fit of a model with its target data	Equation 2-14
$\mathcal{V}_{Q,i}$	The VAF values for all signals in $Q$	Equation 2-15
$\epsilon$	The performance measure given as the 2-norm of the disturbance force	Equation 2-16
$\mathcal{E}_Q$	The performance measures for all signals in $Q$	Equation 2-20
$\mathcal{E}_0$	The performance measures for all signals in $Q$ if no control is applied	Equation 2-21
$\mathcal{E}_{ILC}$	The performance measures for all signals in $Q$ if ILC is applied	Equation 2-22
$\zeta$	The reduction of the performance measure in percentage	Equation 2-23
$\mathcal{Z}_R$	The reduction measures for all signals in $R$	Equation 2-27

**Table 3-6:** An overview of all measures used to present the results of the genetic programming experiments

## 3-2 Results

In this section, the results of the experiment are presented, where we begin by focusing on the compact models found with genetic programming. Here, we analyze how the compact models fit the target data. After that, we focus on the influence of the input variables on the performance of the feedforward controller. Furthermore, the results are presented as a set of models which present the trade-off in transparency and accuracy. From this set of models we select three feedforward controllers dependent on  $\mathcal{R}_{r1}$ ,  $\mathcal{R}_{r2}$  and  $\mathcal{R}_{r3}$  which are considered for further analysis, together with a controller build from a set of models considered in the first section.

### 3-2-1 The analysis of compact models found by genetic programming

Two interesting results emerge by analyzing the more compact equations returned by genetic programming. First of all, for each compensation signal, the algorithm found expressions that have a low complexity and a reasonable fit, while searching in different program spaces. Secondly, the algorithm emphasizes the importance of dependence on the derivatives of the setpoint, as it was capable of finding numerical approximations of the derivatives by searching in a search space constructed of only shifted values  $\mathcal{R}_{r2}$ .

#### Insights in the behavior of the compensation signal by means of trivial models

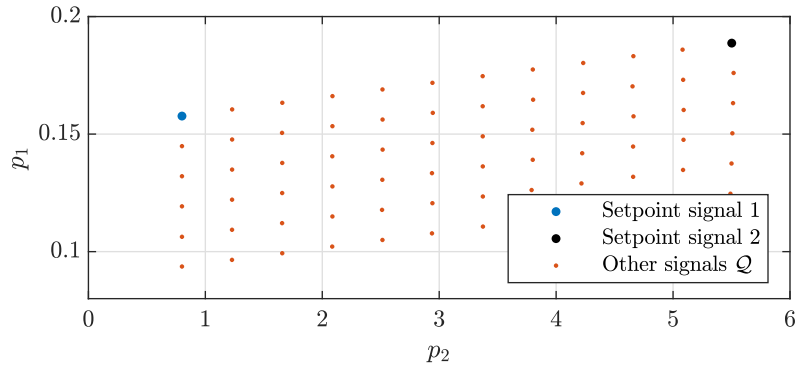
We will start by investigating identical compact expressions found in different experiments. These compact equations give an understanding in the relation between the input variables and the target data. For each input set the algorithm found identical models, but for the sake of brevity we will only present the models that depend on continuous time input variables  $\mathcal{R}_{r1}$ . The models will be referred to as trivial models, due to their

non-complex nature. The trivial models  $\mathcal{U}_{1,triv}$ ,  $\mathcal{U}_{2,triv}$  and  $\mathcal{U}_{3,triv}$  are presented in Table 3-7. All three models have complexity less than 25. Model  $\mathcal{U}_{1,triv}$  and  $\mathcal{U}_{2,triv}$  are first-order polynomial, whereas model  $\mathcal{U}_{3,triv}$  is a second-order polynomial function.

Model	Complexity	Fitness	Variables	Expression
$\mathcal{U}_{1,triv}$	19	5310.920	$\dot{r}, \ddot{r}$	$u_1 = 2.69 - 3.43e^{-3}\ddot{r} + 33.53\dot{r}$
$\mathcal{U}_{2,triv}$	19	61.922	$r, \ddot{r}$	$u_2 = 0.44 - 0.16\ddot{r} - 155.70r$
$\mathcal{U}_{3,triv}$	22	1.065	$r, \dot{r}, \ddot{r}$	$u_3 = -3.29e^{-2} + 0.19\dot{r} + 6.54e^{-2}\ddot{r}r$

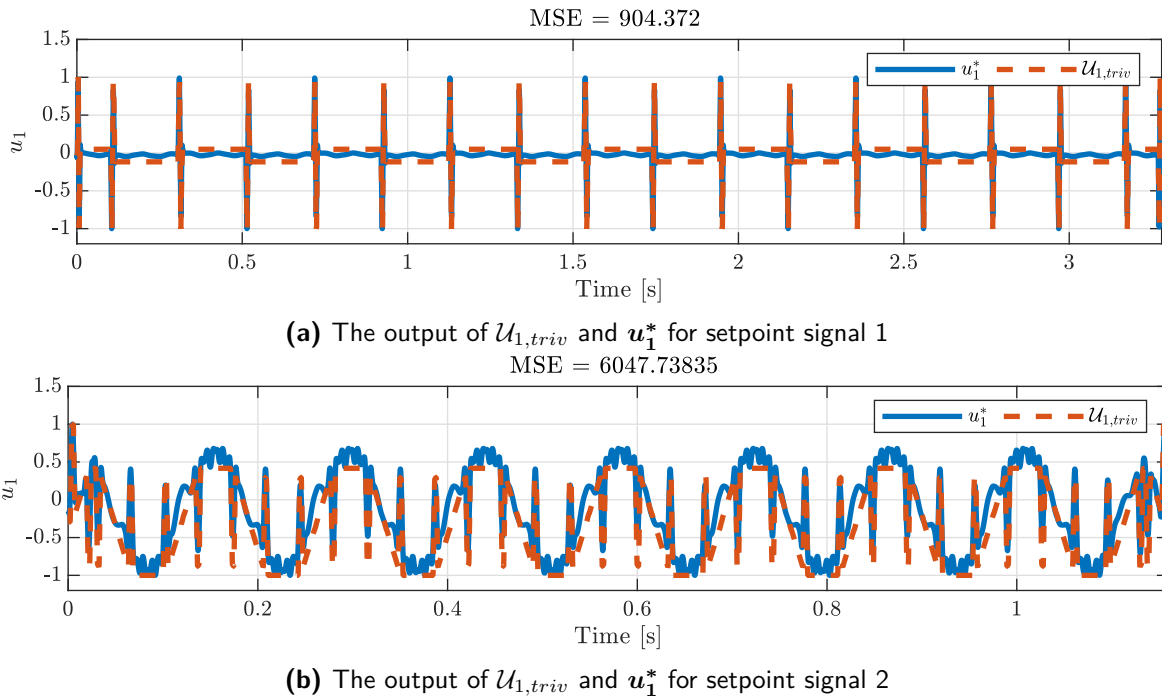
**Table 3-7:** The set of trivial models dependent on  $\mathcal{R}_{r_1}$

The trivial models can be used to get an intuition for how the compensation signals evolve as the setpoint signals change in the parameterized space  $\mathcal{P}$ . This is explained by means of model  $\mathcal{U}_{1,triv}$ . First, two setpoint signals are chosen for illustrative purposes by taking two corner signals in set  $\mathcal{Q}$  as is illustrated in Figure 3-3. Notice as  $p_2$  refers to the maximum velocity of a setpoint signal, we are considering two signals mainly varying in their maximum speed.



**Figure 3-3:** The location of the setpoint signals in space  $\mathcal{P}$

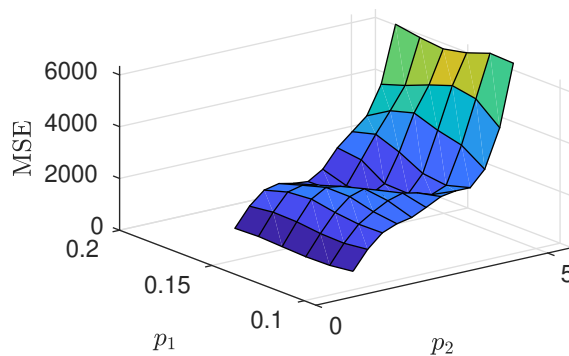
The output of the trivial model is calculated for setpoint signals 1 and 2. The model output and the target data signal  $u_1^*$  for both setpoint signals are provided in Figure 3-4. Please notice the figure presents normalized signals. The accuracy is expressed as MSE and is provided in the title of the plots. MSE is given with respect to the unscaled signals.



**Figure 3-4:** The output of the trivial model and the optimal compensation signal for both setpoint signals

Figure 3-4a shows the output of  $\mathcal{U}_{1,triv}$  and  $u_1^*$  corresponding with setpoint signal 1. First thing to notice is that signal  $u_1^*$  mainly consists of large peak, which are explained by the model. The accuracy of the model for setpoint signal 1 is  $MSE = 904.37$ . Figure 3-4b illustrates the output of  $\mathcal{U}_{1,triv}$  and the signal  $u_1^*$  corresponding with setpoint signal 2. Here, we can see that the signal behaves differently and a component that occurs between the peaks of the signal is not explained by the model. The fit on this signal is poorer, resulting in a higher value of  $MSE = 6047.74$ .

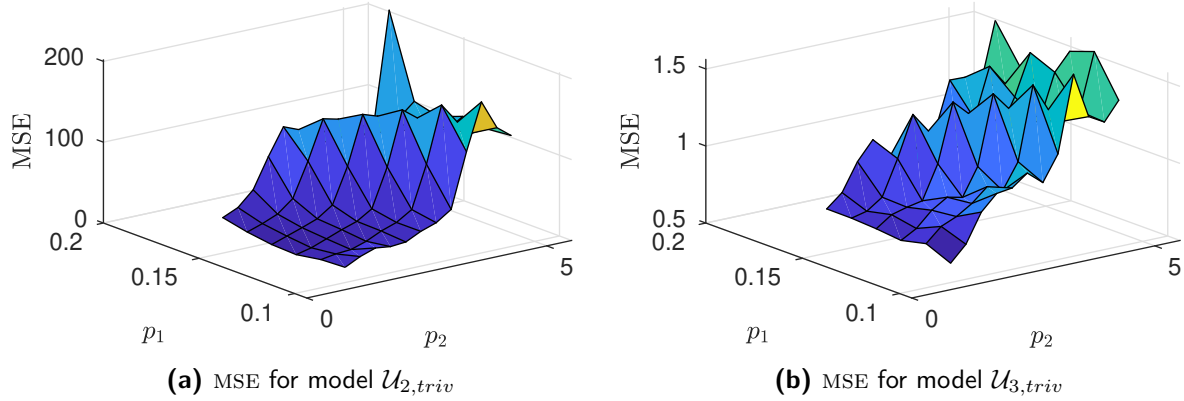
Since the parameterized values in  $\mathcal{Q}$  do not vary in their  $p_3$  value, they can be visualized in a 2D plane, as is done in for example Figure 3-3. Now if we compute the MSE for the trivial model  $\mathcal{U}_{1,triv}$  for all setpoint signals in  $\mathcal{Q}$ , we can visualize its behavior in a 3D plot, which is done in Figure 3-5. This illustrates that as the setpoint signals increase in  $p_2$ , the trivial model results in a poorer fit, as the MSE value increases. The component of the signal noticed in Figure 3-4b seems to appear for higher  $p_2$  values.



**Figure 3-5:** The MSE for all signals in dataset  $\mathcal{Q}$  for the trivial model  $\mathcal{U}_{1,triv}$

Identically, the behavior of the MSE for the trivial models  $\mathcal{U}_{2,triv}$  and  $\mathcal{U}_{3,triv}$  are visualized

in Figure 3-6. Likewise, we can see that signals with lower  $p_2$  values result in a lower MSE, implicating that there is a component in the target signals which is not explained by the trivial models that has more influence as the setpoint signals increase in their maximum velocity. Figure A-10 and A-11 in Appendix A illustrate the fit for  $\mathcal{U}_{2,triv}$  and  $\mathcal{U}_{3,triv}$  on setpoint signal 1 and 2.



**Figure 3-6:** MSE for all signals in training set  $Q$  for the models  $\mathcal{U}_{2,triv}$  and  $\mathcal{U}_{3,triv}$

### Approximating the second order derivative by means of compact models

For the models representing target data  $\mathbf{u}_2^*$ , interesting simple equations are found for all three combinations of variables  $\mathcal{R}_{r1}$ ,  $\mathcal{R}_{r2}$  and  $\mathcal{R}_{r3}$ . The found expressions are provided in Table 3-8. First of all, genetic programming was capable of finding almost an identical expressions while searching in different program spaces. For the experiments searching in a space constructed of  $\mathcal{R}_{r1}$  and  $\mathcal{R}_{r3}$ , two similar models are found, where the latter differs only in one input variable, which is the setpoint signal shifted forward by one step. This result in different parameters and a slightly lower MSE. Secondly, the model dependent on  $\mathcal{R}_{r2}$  looks similar to the numerical approximation of the second order derivative used to obtain  $\ddot{\mathbf{r}}$ .

Input	Complexity	Fitness	Variables	Expression
$\mathcal{R}_{r1}$	19	61.922	$\dot{\mathbf{r}}, \ddot{\mathbf{r}}$	$\mathbf{u}_2 = 0.44 - 0.16 \ddot{\mathbf{r}} - 155.70 \mathbf{r}$
$\mathcal{R}_{r2}$	27	62.124	$\mathbf{r}_{k-1}, \mathbf{r}_k, \mathbf{r}_{k+1}$	$\mathbf{u}_2 = 0.44 + 4.87e5 \mathbf{r}_k - 2.44e5 \mathbf{r}_{k-1} - 2.44e5 \mathbf{r}_{k+1}$
$\mathcal{R}_{r3}$	19	61.767	$\mathbf{r}_{k+1}, \ddot{\mathbf{r}}_k$	$\mathbf{u}_2 = 0.44 - 0.16 \ddot{\mathbf{r}}_k - 155.66 \mathbf{r}_{k+1}$

**Table 3-8:** Similar expressions found for experiments with target data  $\mathbf{u}_2^*$

A central difference scheme is used to calculate the derivatives, given by:

$$\ddot{\mathbf{r}} = -\frac{2}{4h^2} \mathbf{r}_k + \frac{1}{4h^2} \mathbf{r}_{k-2} + \frac{1}{4h^2} \mathbf{r}_{k+2} \quad (3-4)$$

We are comparing the following two expressions given in more significant digits as:

$$\mathbf{u}_2 = 0.436462 - 0.155851 \ddot{\mathbf{r}} + -155.702 \mathbf{r} \quad (3-5)$$

$$\mathbf{u}_2 = 0.444085 + 4.87368e^5 \mathbf{r}_k - 2.43689e^5 \mathbf{r}_{k-1} - 2.43836e^5 \mathbf{r}_{k+1} \quad (3-6)$$

Although Equation 3-6 has only values shifted by one, please remember that these are downsampled values. These shifted values are given as  $\mathbf{r}_{k \pm p * 1}$ , where  $p$  is the downsampling rate, in our case 8.

Now if we take a downsampling rate of 4, and we express the shifted values as  $r_{k\pm 8} = r_{k\pm 2*4}$ , we can compare Equation 3-6 with Equation 3-5 as:

$$-0.155851\ddot{r} = \frac{0.311702}{4h^2}r_k - \frac{0.155851}{4h^2}r_{k-2} - \frac{0.155851}{4h^2}r_{k+2} \quad (3-7)$$

Now taking a sampling time of  $h = 0.0004$  we end up with:

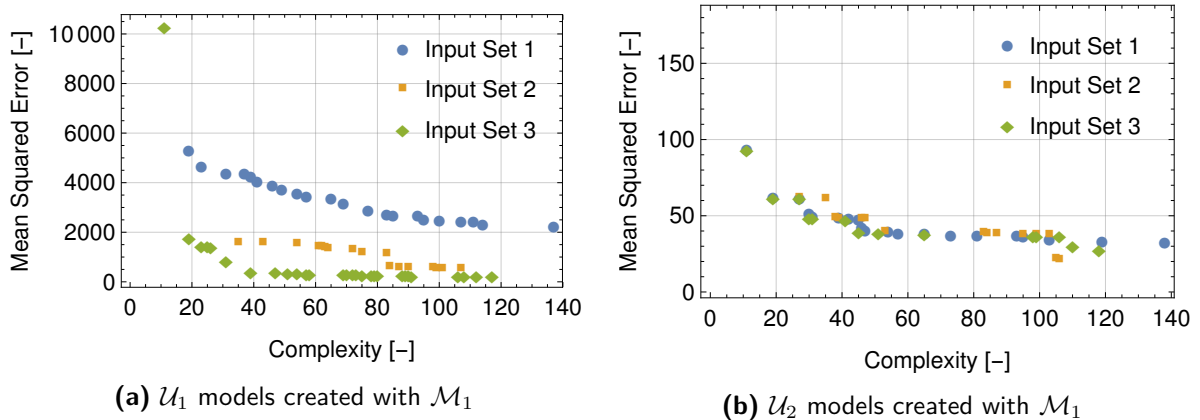
$-0.155851\ddot{r} = 4.87034e^5r_k - 2.43517e^5r_{k-2} - 2.43517e^5r_{k+2}$  and substituting this in Equation 3-5 results in an expression very similar to Equation 3-6.

### 3-2-2 The influence of of the input variables on the feedforward controller performance

In this section we will investigate the influence of the input variables by first directly comparing the models found with genetic programming with respect to their input variables. After that, we will structure the results in such a way that we have a subset of models from which the user can pick an expression to create a feedforward controller. Three types of feedforward controllers are created dependent on the varying input variables and together with the trivial models are considered for further analysis.

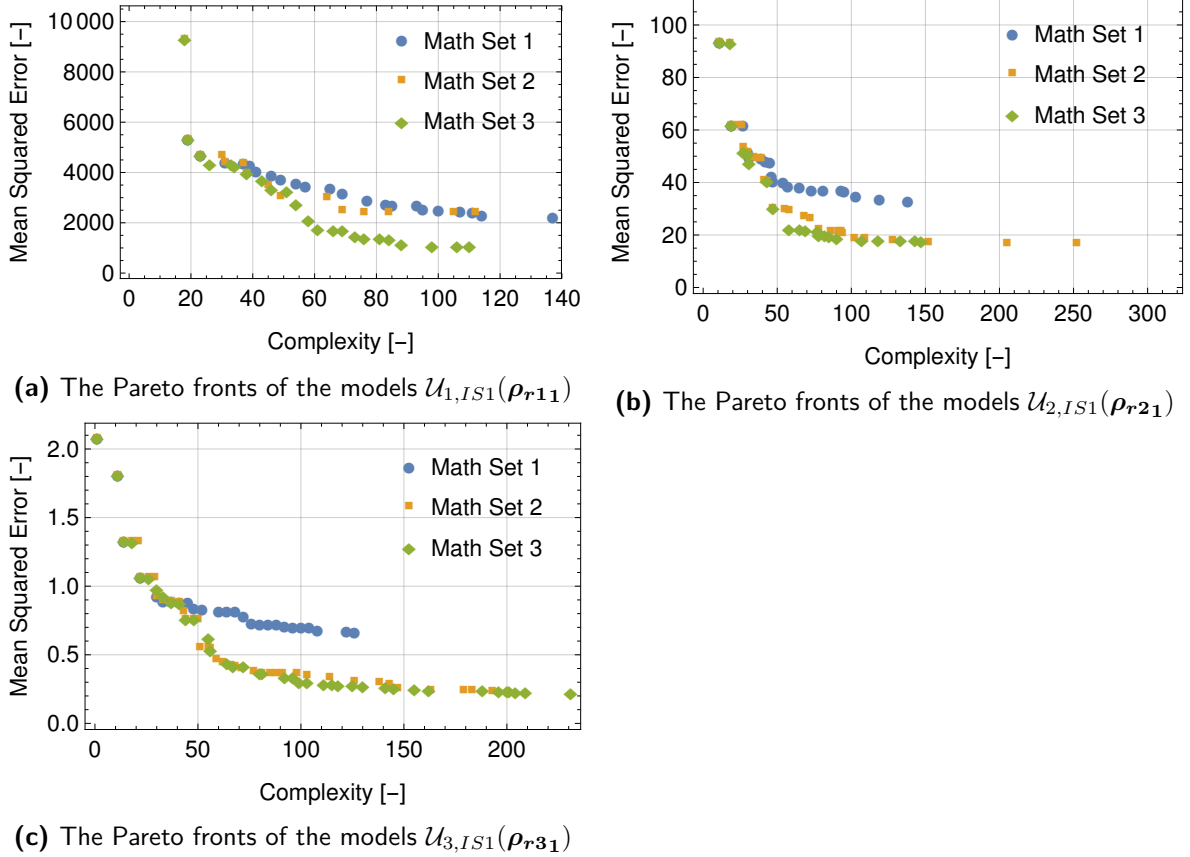
#### The influence of the input set on the accuracy of the models

In this sections we will investigate the influence of  $\mathcal{R}_{rp}$  on the accuracy of the models. We refer to  $\mathcal{R}_{r_1}$  as input set 1,  $\mathcal{R}_{r_2}$  as input set 2 and  $\mathcal{R}_{r_3}$  as input set 3. A subset of found models found with mathematical operators  $\mathcal{M}_1$  for target data  $\mathbf{u}_1^*$  are visualized in a plot in Figure 3-7a. Here, every point corresponds with a model found with genetic programming which is rated on its complexity and accuracy. We observe that the models dependent on input set 2 are more accurate than input set 1. Furthermore, models depending on input set 3 result in even more accurate models than the other input sets, as these settings result in a lower Pareto front. This trend is noticed in a weaker fashion for models found with  $\mathcal{M}_2$  and  $\mathcal{M}_3$ , which can be seen in Figure A-4 in Appendix A. For the models  $\mathcal{U}_2$ , there is not a general trend visible which returns for all math set, as is illustrated by Figure A-5 in Appendix A. The models  $\mathcal{U}_3$  which depend on input set 2 are outperformed by models found with input set 1 and 3. However, the difference in accuracy is minimal. This trend is found for all three sets of mathematical operators. Their Pareto fronts are visualized in A-6 in Appendix A.



**Figure 3-7:** The influence of the input variables on the accuracy of the models

After combining the results found with genetic programming, we can present a subset of models representing the three different target signals. As was mentioned earlier, we are not explicitly interested in which set of mathematical operators returns the best models. Rather, we are looking for a subset of models from which we can select three models that are both interpretable and have a high fit. Figure 3-8 visualizes all found models for input variables  $\mathcal{R}_{r1}$ . From these models the user can select one per compensation signal to create a feedforward controller. Table A-1 in Appendix A provides all models on the Pareto front of Figure 3-8a. Similarly, Figure A-8 and A-9 in Appendix A present the results for the models depending on  $\mathcal{R}_{r2}$  and  $\mathcal{R}_{r3}$ .



**Figure 3-8:** An overview of models which can be chosen from to create the controller  $\mathcal{U}_{IS1}$  dependent on  $\mathcal{R}_{r1}$

We will continue by creating a feedforward controller from the trivial models and another three dependent on  $\mathcal{R}_{r1}$ ,  $\mathcal{R}_{r2}$  and  $\mathcal{R}_{r3}$  as:

$$\mathcal{U}_{triv}(\mathcal{R}_{r1}) = [\mathcal{U}_{1,triv}(\rho_{r11}) \quad \mathcal{U}_{2,triv}(\rho_{r21}) \quad \mathcal{U}_{3,triv}(\rho_{r31})] \quad (3-8)$$

$$\mathcal{U}_{IS1}(\mathcal{R}_{r1}) = [\mathcal{U}_{1,IS1}(\rho_{r11}) \quad \mathcal{U}_{2,IS1}(\rho_{r21}) \quad \mathcal{U}_{3,IS1}(\rho_{r31})] \quad (3-9)$$

$$\mathcal{U}_{IS2}(\mathcal{R}_{r2}) = [\mathcal{U}_{1,IS2}(\rho_{r12}) \quad \mathcal{U}_{2,IS2}(\rho_{r22}) \quad \mathcal{U}_{3,IS2}(\rho_{r32})] \quad (3-10)$$

$$\mathcal{U}_{IS3}(\mathcal{R}_{r3}) = [\mathcal{U}_{1,IS3}(\rho_{r13}) \quad \mathcal{U}_{2,IS3}(\rho_{r23}) \quad \mathcal{U}_{3,IS3}(\rho_{r33})] \quad (3-11)$$

These models are chosen such that they are interpretable and nonlinear operators are avoided. Furthermore, they are sampled at the middle of the Pareto front, considering the

trade-off in the complexity and the MSE of the model. The three controllers created from the three models are presented in Table 3-9 3-10 and 3-11, respectively. The information of the chosen models is given in Table 3-12, where their complexity, MSE and their input variables are presented.

Input	Expression
$\mathcal{R}_{r1}$	$u_1 = 4.63 + 152.72 \sin(1.93e^{-2}\ddot{r}) + 3.87e^{-3}\ddot{r} + 1.93\dot{r}^3$
$\mathcal{R}_{r1}$	$u_2 = 2.22 - 0.17\ddot{r} + 87.86 \cos(3124.49r^3)r - 197.43r + 6797.27 \cos(22.65r)^2r^2 - 3.84e^5r^4$
$\mathcal{R}_{r1}$	$u_3 = 0.71 - 1.87 \cos(16.09 + 41.47r)^4 - 4.31e^{-3}\ddot{r} - 9.65e^{-3} \sin(41.47 + 11.87r)\ddot{r}$

**Table 3-9:** The genetic programming controller  $\mathcal{U}_{IS1}$

Input	Expression
$\mathcal{R}_{r2}$	$u_1 = 4.04 + 7.32e^6r_{k-1} - 4.58e^6r_{k-2} + 4.17e^5r_{k-3} - 4.47e^6r_{k+1} + 1.42e^7r_{k+1}^3 - 5.81e^7r_{k-3}r_{k+1}r_{k+2} + 1.31e^6r_{k+3} + 4.40r_{k-1}r_{k-3}r_{k+3}$
$\mathcal{R}_{r2}$	$u_2 = -4.33 + 8.83 \cos(38.16 - 445.49r_k^2) + 94.77r_k \cos(0.75 - 445.49r_k^2)r_k + 2.66e^5r_k - 1.77e^5r_{k-1} - 8.89e^4r_{k+2}$
$\mathcal{R}_{r2}$	$u_3 = -1.29e^5 + 0.97 \cos(84.97(7.22 + r_k)) + 1.29e^5 \cos(r_{k+1}) + 245.80r_k - 240.95r_{k-1} + 1.20e^4r_{k-3}r_{k+1} + 5.15e^4r_{k+1}r_{k+2}$

**Table 3-10:** The genetic programming controller  $\mathcal{U}_{IS2}$

Input	Expression
$\mathcal{R}_{r3}$	$u_1 = 4.84 + 5.20\ddot{r}_{k+1} + 3129.67\ddot{r}_{k-1} - 10.72\dot{r}_{k-2}r_{k-3}^2 + 12.18\ddot{r}_{k-3}^3 - 3117.17\dot{r}_{k+1} + 0.19\dot{r}$
$\mathcal{R}_{r3}$	$u_2 = -12.06 + 14.67 \cos(366.18r_k^2) - 0.17\ddot{r}_k + 91.02 \cos(366.18r_k^2)r_k - 175.98r_k + 3041.98r_k^2 - 1.74e^5r_kr_{k-2}^3$
$\mathcal{R}_{r3}$	$u_3 = -1.88e^{-2} - 0.27 * \cos(80.93 + r_k)^2 + \cos((\cos(r_k) + 40.46 + r_k)^2) - 0.28\ddot{r}_{k-3}r_k^2 + 0.20\dot{r}_{k+3} + 7.22e^{-2}\ddot{r}_kr_{k+3}$

**Table 3-11:** The genetic programming controller  $\mathcal{U}_{IS3}$

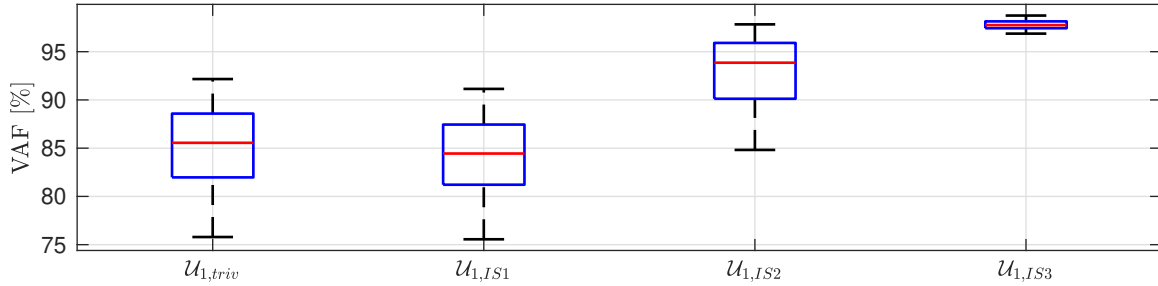
Output	Input	Complexity	MSE	Variables
$u_1$	$\mathcal{R}_{r1}$	38	3975.150	$\dot{r}, \ddot{r}, \ddot{r}$
$u_2$	$\mathcal{R}_{r1}$	102	18.843	$r, \ddot{r}$
$u_3$	$\mathcal{R}_{r1}$	103	0.349	$r, \dot{r}, \ddot{r}$
$u_1$	$\mathcal{R}_{r2}$	83	1160.450	$r_{k-3}r_{k-2}r_{k-1}r_{k+1}r_{k+2}r_{k+3}$
$u_2$	$\mathcal{R}_{r2}$	112	18.473	$r_{k-1}, r_k, r_{k+2}$
$u_3$	$\mathcal{R}_{r2}$	87	0.428	$r_{k-3}, r_{k-1}, r_k, r_{k+1}, r_{k+2}$
$u_1$	$\mathcal{R}_{r3}$	54	350.938	$\ddot{r}_{k-3}, \dot{r}_{k-1}, \ddot{r}_{k+3}, r_{k-3}, r_{k-2}, r_{k+1}$
$u_2$	$\mathcal{R}_{r3}$	105	19.036	$r_{k-2}, r_k, \ddot{r}_k$
$u_3$	$\mathcal{R}_{r3}$	108	0.340	$r_k, r_{k+3}, \dot{r}_{k+3}, \ddot{r}_k$

**Table 3-12:** Information about the chosen models for the controllers  $\mathcal{U}_{IS1}, \mathcal{U}_{IS2}$  and  $\mathcal{U}_{IS3}$

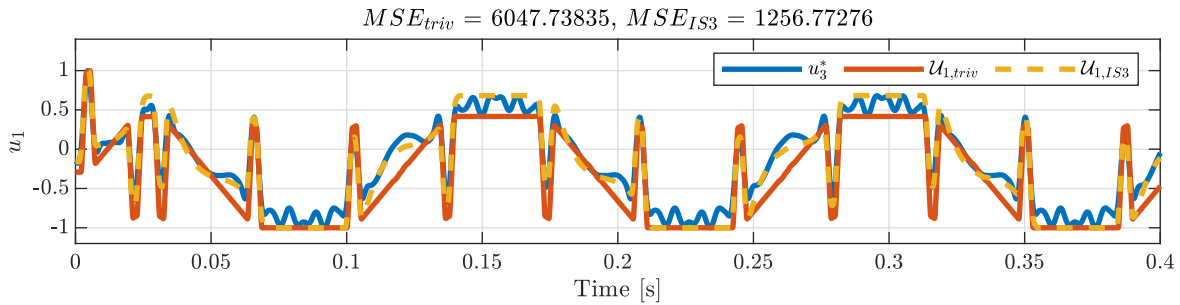
### The accuracy analysis for the models chosen as feedforward controller

The set  $\mathcal{V}_{Q,1}$  for the models  $\mathcal{U}_{1,triv}, \mathcal{U}_{1,IS1}, \mathcal{U}_{1,IS2}$  and  $\mathcal{U}_{1,IS3}$  are given in Figure 3-9. We can see that model  $\mathcal{U}_{1,IS3}$  is performing the best, as the VAF values are the highest. This is noticed earlier in Figure 3-7b, where the lowest Pareto front is obtained for input set 3. Furthermore, we can see that the difference in accuracy between  $\mathcal{U}_{1,triv}$  and  $\mathcal{U}_{1,IS1}$  is

minimal. A time domain plot with the target data, the output of  $\mathcal{U}_{1,triv}$  and the output of the best performing model  $\mathcal{U}_{1,IS3}$  for setpoint signal 2 is provided in Figure 3-10.



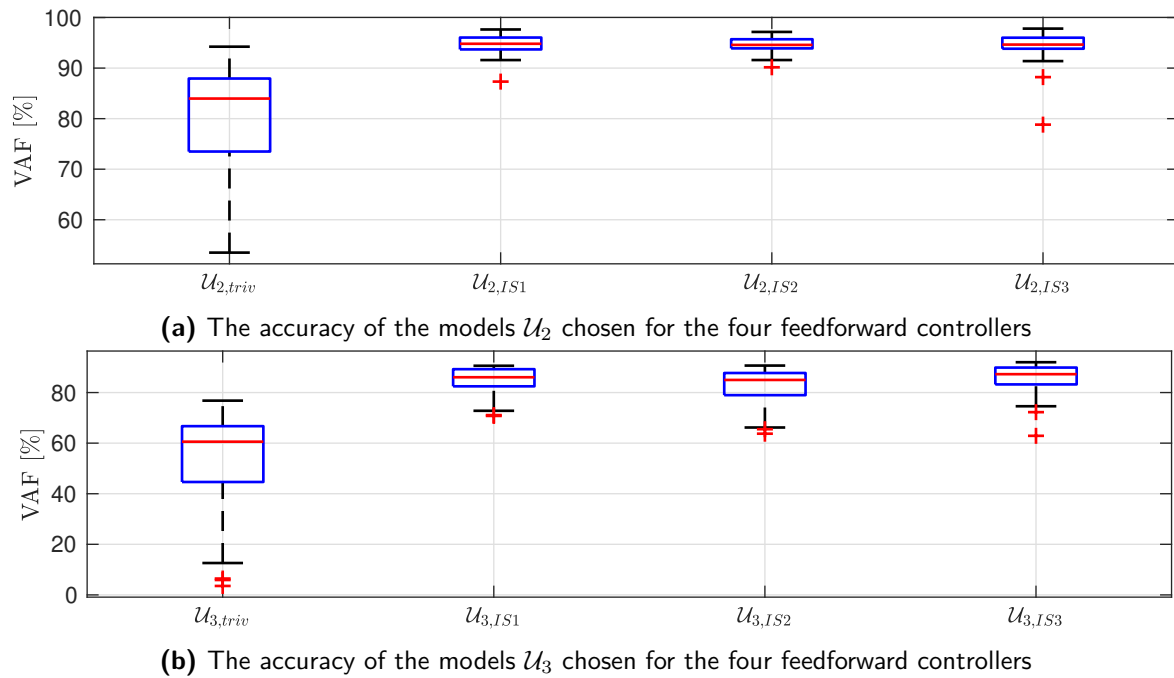
**Figure 3-9:** The accuracy of the models  $\mathcal{U}_1$  chosen for the four feedforward controllers



**Figure 3-10:** The output of  $\mathcal{U}_{1,triv}, \mathcal{U}_{1,IS1}$  and signal  $u_1^*$  for setpoint signal 2

The sets  $\mathcal{V}_{Q,2}$  and  $\mathcal{V}_{Q,3}$  for the corresponding models in the four considered feedforward controllers are given in Figure 3-11a and Figure 3-11b. Here, we can see in both cases that the trivial model is the least accurate. Furthermore, there is little difference in accuracy for the models dependent on input set 1, 2 and 3. For a time domain plot of the output of models  $\mathcal{U}_{2,triv}$  and  $\mathcal{U}_{2,IS3}$  and the output of the models  $\mathcal{U}_{3,triv}$  and  $\mathcal{U}_{3,IS3}$  corresponding with setpoint signal 2, please see Figure A-12 in Appendix A.

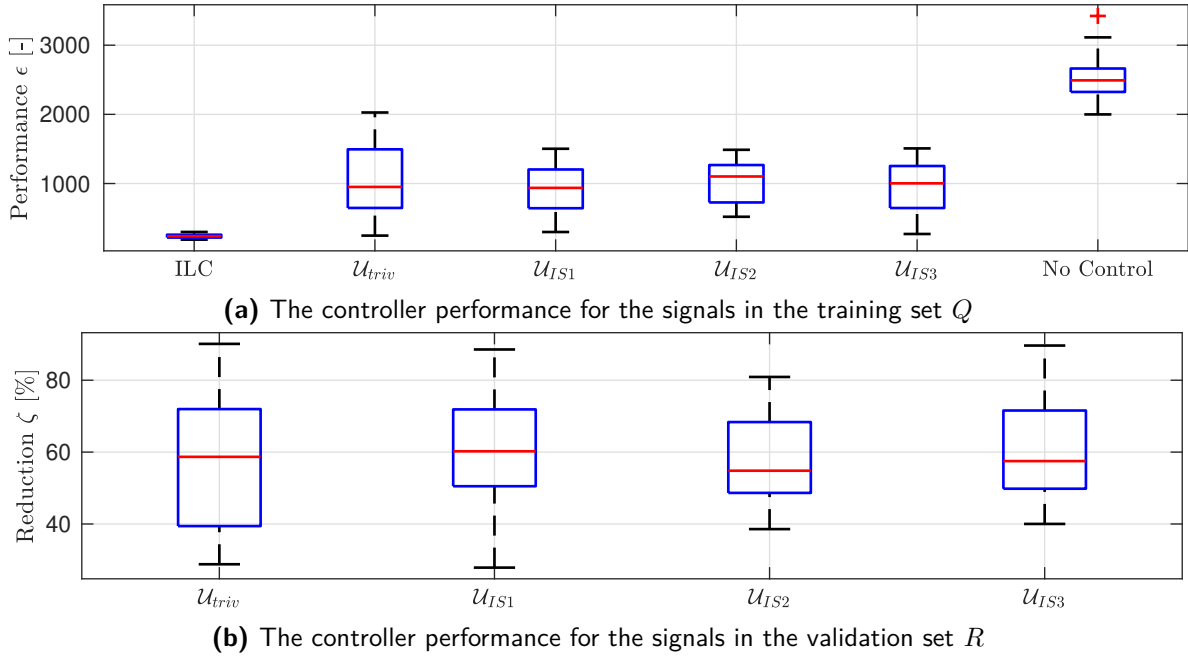




**Figure 3-11:** The accuracy of the models considered in the four feedforward controllers

### 3-2-3 Feedforward controller performance

The sets of performance measures  $\mathcal{E}_Q$  for the feedforward controllers  $\mathcal{U}_{IS1}, \mathcal{U}_{IS2}, \mathcal{U}_{IS3}$  and  $\mathcal{U}_{triv}$  are given in Figure 3-12a, where the performance sets for optimal control and no control,  $\mathcal{E}_{ILC}$  and  $\mathcal{E}_0$  are also presented. We can see that the ILC signals reduce the disturbance the most. The other four controllers all reduce the median of the set of performance measures. Furthermore, the trivial controller reduces the disturbance less than the other controllers as the set of disturbance measures is more spread out. This is in line with the previous observations, where we saw that the trivial models had the lowest set of VAF values. There is little performance difference between the other three controllers. Figure 3-9 showed that the model  $\mathcal{U}_{1,IS3}$  is the most accurate. However, this is not directly reflected on the performance of the controllers.



**Figure 3-12:** The controller performance of the 4 feedforward controllers

To investigate if the controllers are satisfying condition 1-2.1, their performance is expressed in a percentage of reduction  $\zeta$ . Figure 3-12b shows the sets of reduction measures  $\mathcal{Z}_R$  for all 4 controllers. From this figure we can conclude that the performance measures are reduced for all signals in the validation set and that the controllers satisfy Condition 1-2.1. The minimum and maximum percentage of reduction obtained by the controllers are given in Table 3-13. Furthermore, based on these values, controller  $\mathcal{U}_{IS3}$  returns the best performance, as it has the highest minimum reduction.

Controller	Minimum reduction	Maximum reduction
$\mathcal{U}_{triv}$	28.79 %	90.11%
$\mathcal{U}_{IS1}$	27.86 %	88.56 %
$\mathcal{U}_{IS2}$	38.57 %	80.91 %
$\mathcal{U}_{IS3}$	40.01 %	89.65 %

**Table 3-13:** The minimum and maximum reduction for the controllers found with genetic programming

### 3-2-4 Concluding remarks

- We observed that the general trend of the compensation signals is explained by a set of trivial models. Furthermore, we noticed that the trivial models have a better fit for setpoint signals with lower  $p_2$  values and their fit decreases for signals with higher  $p_2$  values. Therefore, we speak of a component that is not captured by the trivial models, which is dependent on  $p_2 = \max(\dot{\mathbf{r}})$ . We also noticed that genetic programming managed to find the approximation of the second order derivative.
- The influence of the different input sets on the accuracy of the models was strongly noticed for the  $\mathcal{U}_1$  models. For the  $\mathcal{U}_2$  and  $\mathcal{U}_3$  models, the influence was minimal.
- The results of the genetic programming experiments are merged to 9 sets of models. Feedforward controllers are created by taking models that depend on the same input

sets. We created four feedforward controllers. The trivial controller was created by sampling models with low complexity. The other three controller were created by sampling models that are average in accuracy and complexity. All four controllers reduced the disturbance measure with at least 28.79 %, therefore satisfying Condition 1-2.1. The difference in performance between the trivial controller and the other controllers was noticeable but not significant.

### 3-3 Discussion

We noticed that the models  $\mathcal{U}_1$  are strongly influenced by the type of input set that was chosen. From a physical interpretation of the signals, this might be explained by the fact that both signal  $\mathbf{u}_1$  and setpoint signal  $\mathbf{r}$  act in the same dimension. For the other models, there was no strong increase in accuracy for different input sets. Furthermore, we noticed that the increase in accuracy was not directly reflected on the performance of the controller. This indicates that the disturbance force is less dependent on signal  $u_1$  than the other two or that the increase of accuracy in  $u_1$  was to minimal to influence the controller performance. From a physical interpretation of the signals, the first suggestion is more likely, as the disturbance force and signal  $u_1$  operate in different dimensions. There is an increase in the minimal reduction percentage  $\min(\zeta)$  detected for controllers  $\mathcal{U}_{IS1}, \mathcal{U}_{IS2}$  and  $\mathcal{U}_{IS3}$ . Moreover, we noticed that all four genetic programming controllers are capable of reducing the performance measure for the set of setpoint signals in the validation set with at least 27.86 %.

The discovery of the numerical approximation of the second order derivative suggest the mapping depends on setpoint signal derivatives. By analyzing the trivial models we could get an insight in the behavior of the compensation signals throughout the parameterized space  $\mathcal{P}$ . We discovered a component in the signals which is dependent on  $p_2 = \max(\dot{\mathbf{r}})$ . Now combining the observation that the trivial models fail to describe a component that is influenced by  $p_2$  and by looking at the dependent variables of the trivial models, we can select the combination of relevant input variables. For the mapping  $\mathcal{U}_2$  and  $\mathcal{U}_3$ , the experiments suggest that the input variables are given as  $\boldsymbol{\rho}_{r1} = \boldsymbol{\rho}_{r2} = [\mathbf{r} \ \dot{\mathbf{r}} \ \ddot{\mathbf{r}}]$ . For model  $\mathcal{U}_1$ , the trivial model suggest that the input set consist of  $\boldsymbol{\rho}_{r1} = [\dot{\mathbf{r}} \ \ddot{\mathbf{r}}]$ . From the physical system we can also suspect that  $\boldsymbol{\rho}_{r1}$  depends on  $\mathbf{r}$  and  $\ddot{\mathbf{r}}$  as there is a mass present in the system, which have second-order dynamics. This is confirmed when analyzing the Pareto front of models  $\mathcal{U}_1$  dependent on  $\mathcal{R}_{r1}$  in Table A-1 in the Appendix A.

Since no significant improvement in controller performance was observed and the algorithm suggested the dependence on the numerical approximation, the input sets  $\boldsymbol{\rho}_{r1}, \boldsymbol{\rho}_{r2}$  and  $\boldsymbol{\rho}_{r3}$  are chosen from the first combination of input variables  $\mathcal{R}_{r1}$  consisting of the setpoint signals and their numerical derivatives.

The main advantage of genetic programming is its ability to find analytic equations representing the target data. The set of models that are found with genetic programming vary in complexity and are composed from mathematical operators from all three sets of mathematical operators. Therefore, it reflects the trade-off between the transparency of the model and its accuracy. We sampled a set of models low in complexity and used it as controller. We also sampled a set of models which where average in complexity and average in accuracy. The increase in complexity came with an increase in controller performance. Sampling the models with the highest fit and largest complexity came with

a controller that did not perform for all signals in validation set  $R$ , thus not satisfying Condition 1-2.1.

### 3-4 Conclusion

Now to conclude the chapter, we answer the following research questions:

1. *Which combination of variables  $\rho_{r1}$ ,  $\rho_{r2}$  and  $\rho_{r3}$  are beneficial for the performance of the feedforward controller?*

There was minimal difference in controller performance between the models  $\mathcal{U}_{IS1}$ ,  $\mathcal{U}_{IS2}$  and  $\mathcal{U}_{IS3}$ . Furthermore, the discovery of the approximation of the second order derivative emphasizes that the data is explained by continuous time derivatives. Therefore, the set of input variables that are used for the neural network models is presented in Table 3-14.

2. *Can we use genetic programming to find a set of models from which the user can select a model that is interpretive and has a high fit?*

We can see that the found models are represented rated on their complexity and fitness. From this set of results, it is up to the user to select three models to combine as feedforward controller. Preference might be for models that are created without any nonlinear functions and are giving more insight in the relation between the input variables and the output of the model. Perhaps the controller performance is preferred over the transparency of a model and the user selects a set of models with high complexity and high fit.

Target data	Input set
signal $\mathbf{u}_1^*$	$\rho_{r1} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}, \dot{\ddot{\mathbf{r}}}]$
signal $\mathbf{u}_2^*$	$\rho_{r2} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}]$
signal $\mathbf{u}_3^*$	$\rho_{r3} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}]$

**Table 3-14:** An overview of the relevant input variables per target dataset

# Feedforward controller design by means of feedforward artificial neural networks

In this chapter we will explain the design of the feedforward controller using feedforward artificial neural networks. Within the blackbox nonlinear modeling method, the structure of the model is fixed a priori and the parameters in the structure are optimized by minimizing the error between the output of the model and its target data. Two networks are considered, which mainly differ in their activation function. The first network uses the hyperbolic tangent sigmoid transfer function referred to as the tanh network, whereas the second network uses the Rectifier Linear Unit activation function, referred to as the ReLU network.

The neural networks are used to solve Problem 1-2.4, which is formulated as an optimization problem directly searching in the program space. Since the neural networks are not capable of doing that, the problem is written as an optimization problem where we are looking for the optimal parameter vector  $\theta^*$ . The feedforward controller is designed by solving Problem 4-0.1.

**Problem 4-0.1.** *Design a feedforward controller  $\mathcal{U} : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  defined as:*

$$\mathcal{U}^*(\mathcal{R}_r) = [\mathcal{U}_1^*(\rho_{r1}) \quad \mathcal{U}_2^*(\rho_{r2}) \quad \mathcal{U}_3^*(\rho_{r3})] \quad (4-1)$$

*by solving the following three optimization problems:*

$$\forall (r, \mathbf{u}_i^*) \in Q_j, \theta_i^* = \arg \min_{\theta_i \in \mathbb{R}^M} \frac{1}{N} \|\mathbf{u}_i^* - \mathcal{U}_i(\rho_{ri}, \theta_i)\|_2^2 \text{ for } i = 1, 2, 3 \quad (4-2)$$

Here  $\theta_1, \theta_2$  and  $\theta_3$  are vectors containing all the  $M$  weight and biases in the neural network,  $\mathbf{u}_1^*, \mathbf{u}_2^*$  and  $\mathbf{u}_3^*$  are the three optimal compensation signals obtained with ILC,  $\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3$  are the different networks representing different optimal compensation signals,  $\rho_{r1}, \rho_{r2}$  and  $\rho_{r3}$  are their corresponding input sets and  $N$  the amount of samples in the signals that are considered. Furthermore,  $Q_j$  is the data set on which the neural networks are trained and this is always a subset of the training set  $Q$ ,  $Q_j \subseteq Q$ .

The chapter is split up in two main sections. First, the experimental set-up is given, where we will explain what we are investigating and how we achieve this. After that, the results of the experiments are presented, followed by the discussion of the results. The chapter closes with a conclusion based on the experiments.

## 4-1 Experimental Set-Up

The experimental set-up focuses on the influence of the size of a neural network and the influence of the training data on the performance of the neural network feedforward controller. The first subsection will present the research questions we answer in this chapter. The second subsection will motivate the choices of varying parameters and explain in detail how these parameters are varied. The third subsection will discuss the other settings chosen during the experiments. The last subsection provides the framework used to process the results.

### 4-1-1 Research questions

Generally, the accuracy of a neural network is influenced by its size [12]. However, using a neural networks with extensive hidden layers and neurons comes with the risk of overfitting, which means that the network performs well on the training data, but fails to predict unseen data. Contradictory, too little hidden layers and neurons might result in a neural network that does not succeed in predicting the data accurately. It is therefore of importance to pick a neural network with sufficient size. Since there is no formal method to pick a sufficient network size, one of the goals of the experiments is to find a neural network size which is satisfactory. The research question 1 is given below.

Each combination of setpoint signals and optimal compensation signals in training set  $Q$  contains information on what the model should predict. This means that training a network on all signals, results in a model that tries to capture all information. However, since gathering training data is time-consuming and in our case also cost-consuming, it is beneficial to train on as little data possible. Therefore, the set of experiments investigate the influence of the training data on the performance of the neural networks, motivating research question 2.

1. *What size of network is sufficient for its performance as feedforward controller?*
2. *When does the amount of data points influences the performance of the feedforward controller significantly?*

### 4-1-2 Varying parameters in the experimental set-up

The set of designed experiments are created by varying parameters in the training of the neural networks. In this subsection, we will present all the parameters that are varied throughout the experimental set-up. We will present the chosen settings and their motivation and wrap up the subsection with an overview of all neural networks considered in this experiment.

## Hidden Layers

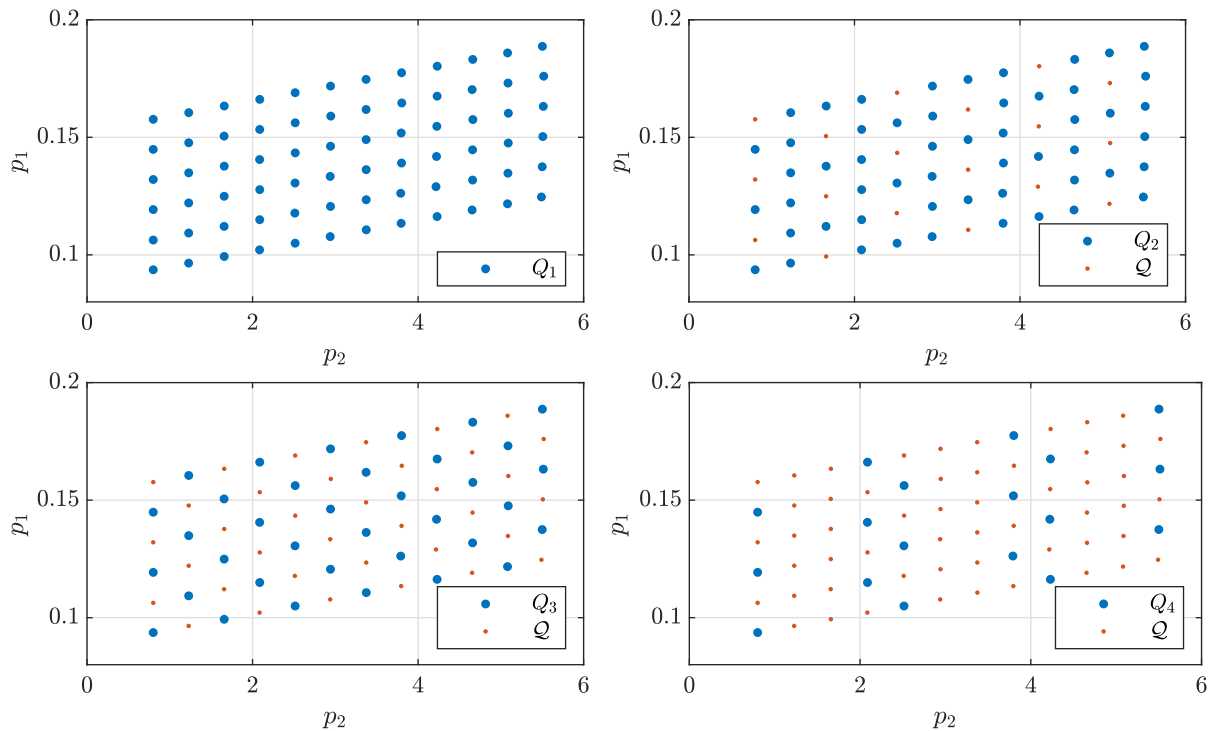
Three different hidden layer sizes are considered, which are chosen in empirical fashion. Networks with 2, 3 and 4 hidden layers are used in the experiment. For the networks with the tanh activation function, 10 neurons per layers are used. For the networks using the ReLU activation function, 20 neurons per layers are used. The number of neurons are increased until the performance of both networks are similar. Adding more neurons per layer to the ReLU network, resulted in a training process that converged poorly, as it started to overfit immediately. An overview of the complexity of these networks is presented in Table 4-1.

Activation Function	Hidden Layers	Neurons per Layer	Inputs	Weights and biases
tanh	2	10	3/4	161/171
tanh	3	10	3/4	271/371
tanh	4	10	3/4	381/291
ReLU	2	20	3/4	521/541
ReLU	3	20	3/4	941/961
ReLU	4	20	3/4	1361/1381

**Table 4-1:** Overview of the used network sizes in the neural networks experiment

## Training data sets

Four training data sets are considered, descending in the number of data points. The training data sets  $Q_1$ ,  $Q_2$ ,  $Q_3$  and  $Q_4$  contain 72, 54, 36 and 18 signals, respectively. These signals are uniformly sampled throughout the parameterized training setpoint signals  $\mathcal{Q}$  as is illustrated in Figure 4-1. Since all signals have the same  $p_3$  value, the training data sets are provided in a 2D plot. Furthermore, training data set  $Q_4$  is converging from the uniform sampling. The signals with the maximum and minimum  $p_2$  in  $\mathcal{Q}$  are included in the data sets, with the idea that if once a neural network learns the signals on the vertices, it is capable of interpolating in between.



**Figure 4-1:** The signals per training dataset sampled in the parameterized training set  $\mathcal{Q}$  in the neural networks experiment

Each point expressed in Figure 4-1 corresponds with a combination of  $\mathbf{r}$  and  $\mathbf{U}^*$ . As was presented in section 2-1, the signals consist of repetitive periods containing identical information. The training sets are reduced further by removing redundant periods. For each signal, the first, the last and two normal periods are considered.

The training data sets are randomly split up in a training, test and validation set. The training set uses 70 % of the data, whereas both the test and validation set uses 15 % of the data. The networks are trained on the training set. To see if the neural network is over-fitting, its error on the validation set is checked. By checking if the error increases on the validation set, while it does not on the training set, we can get an indication that the network is over-fitting. The test set is used to give an unbiased indication of the performance of the neural network.

The Levenberg Marquardt (LM) training algorithm [26] is used to optimize the parameters within the network. This training method uses the gradient of the error. By scaling both the input and target data we ensure that the data is in the same output range as the activation function, resulting in a better gradient function. A lower gradient leads to smaller performance increments per iteration, resulting in a slower training process. For the tanh network, both in- and output data is scaled between  $[-1, 1]$ . This differs from the scaling used for the ReLU network. Here, the input data is also scaled between  $[-1, 1]$ , whereas the target data is scaled between  $[1, 2]$ . Please note that the final activation of the ReLU network is a ReLU activation function, which is needed following the convention of SHERLOCK. Therefore, any negative output is saturated at 0, so the target data cannot be scaled between  $[-1, 1]$ . Scaling the data close to zero is preferable to keep a larger gradient. However, scaling between  $[0, 2]$  results in a fragile lower bound, because a negative output is saturated by the final activation function.



Dataset	Number of signals	Number of datapoints
Dataset $Q_1$	72	481, 289
Dataset $Q_2$	54	351, 860
Dataset $Q_3$	36	239, 585
Dataset $Q_4$	18	123, 791

**Table 4-2:** An overview of the amount of data per training dataset used in the neural network experiment

### Repetition, Target Data and Activation Function

The neural network are initialized randomly with the Nguyen-Widrow layer (NW1) initialization [29], which chooses weight and biases per layer in such manner that the active regions of the activation functions are approximately evenly distributed over the input space of that layer. The initialization is random, meaning that the training process of the networks is nondeterministic. Every experiment has to be repeated several times, to filter out the influence of the initialization on the performance.

Each experiment is repeated 3 times with the exact same settings. Every signal  $\mathbf{u}_1^*$ ,  $\mathbf{u}_2^*$  and  $\mathbf{u}_3^*$  is represented by a different neural network, which results in 3 varying options. Finally, as two types of activation functions are considered, 2 extra varying parameters are added. As is presented, the training data, amount of hidden layers, target data and activation function are varied. An overview of the varying parameters is given in Table 4-3. In total  $4 \times 3 \times 3 \times 2 = 72$  different neural networks are considered.

<b>Training data</b>	$(Q_1, Q_2, Q_3, Q_4)$
<b>Hidden layers</b>	$(2, 3, 4)$
<b>Target data</b>	$\{\mathbf{u}_1^*, \mathbf{u}_2^*, \mathbf{u}_3^*\}$
<b>Activation function</b>	$(\tanh, ReLU)$

**Table 4-3:** An overview of varying parameters in the neural networks experiment

### 4-1-3 Other parameters in the experimental set-up

The choices for the settings of the other parameters needed to design the feedforward controller by means of the neural networks and their motivation are given in this subsection.

#### Input variables per network

The local optimum models found in the genetic programming experiments suggested which derivatives are interesting per target data signal. An overview of the input sets used for the neural networks is presented in Table 4-4.

Target data	Input set
$\mathbf{u}_1^*$	$\rho_{r1} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}, \ddot{\dot{\mathbf{r}}}]$
$\mathbf{u}_2^*$	$\rho_{r2} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}]$
$\mathbf{u}_3^*$	$\rho_{r3} = [\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}]$

**Table 4-4:** An overview of input variables per target dataset used in the neural networks experiment

### Training process of the neural networks

The weights and biases in the neural network are initialized using the NW1 initialization. The networks are trained using the LM optimization algorithm [26], which is a nonlinear optimization algorithm capable of handling numerically ill-conditioned problems. Batch training is used, where in one iteration, all training samples in the training set are used together in the optimization, followed by an update in the weights and biases. An iteration is referred to as an epoch. The training process is done using the deep learning toolbox in MATLAB R2017b [30], which supports parallel computation. The experiments are performed on an Intel(R) Core (TM) i5-6500 CPU @ 3.20 GHz.

The training of neural networks can be terminated for multiple reasons. First of all, the training is terminated if 1000 epochs are reached. Secondly, during the training process, the error on the validation set is monitored. If the error increases for 6 consecutive epochs, the training process is terminated to prevent over-fitting.

Furthermore, the LM algorithm is given as:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - (\mu_k I + H(\boldsymbol{\theta}_k))^{-1} \Delta \mathcal{U}(\boldsymbol{\theta}_k) \quad (4-3)$$

Where  $H$  is the hessian and  $\Delta \mathcal{U}(\boldsymbol{\theta}_k)$  the gradient of function  $\mathcal{U}(\boldsymbol{\theta}_k)$ . If  $\|H(\boldsymbol{\theta}_k)\| \gg \mu$ , the algorithm behaves as Newton's algorithm, which generally performs more accurate and faster nearby an error minimum. If  $\|H(\boldsymbol{\theta}_k)\| \ll \mu$ , the algorithm behaves as the gradient descent algorithm with small step size. Therefore, we start the algorithm with an initial  $\mu_0 = 0.001$ . The  $\mu$  value is decreased with a factor of  $\mu_{k+1} = \mu_{decr} \mu_k$  and is only increased if the following steps lead to a decrease in performance. If this happens, the  $\mu$  is increased as  $\mu_{k+1} = \mu_{incr} \mu_k$ . In this way, the algorithm acts initially as newton's algorithm by lowering the  $\mu$  value. Once the convergence approaches an optimum, the final steps are done by an approximation of the steepest decent method with a small step size.

If the  $\mu$ -value in the LM training algorithm reaches a maximum of  $\mu_{max} = 10^{10}$ , the training process is terminated. Continuing training with high  $\mu$  values does not improve learning, since we are taking minimal steps with the steepest decent method. Furthermore, the training process is terminated if a minimum gradient of  $10^{-7}$  is reached, indicating we have approximated an optimum. An overview of all training thresholds and parameters is presented in Table 4-5. Furthermore, the performance of the neural network is measured using the MSE defined as:

$$e_k = \frac{1}{N} \|\mathbf{u}^* - \mathcal{U}(\boldsymbol{\theta}_k)\|_2^2 \quad (4-4)$$

#### 4-1-4 Processing the results of the neural networks experiment

As each experiment was repeated 3 times, the best neural network is chosen based on the lowest MSE for all target data in training set Q. The best neural network will be considered for further analysis. The results will be processed equivalent to how the problem is approached. First, we will analyze how well the neural networks fit their target data, which is referred to as the accuracy analysis. Secondly, we will combine the neural networks into a feedforward controller and evaluate how the disturbance force is attenuated, which we refer to as the feedforward controller analysis.

Algorithm Setting	Value
<i>Maximum epochs</i>	1000
<i>Minimum gradient</i>	$1e^{-7}$
<i>Validation Checks</i>	6
$\mu_{\max}$	$1e^{10}$
$\mu_0$	0.001
<i>Increase factor <math>\mu_{incr}</math></i>	10
<i>Decrease factor <math>\mu_{decr}</math></i>	0.1
<i>Maximum Time</i>	$\infty$
<i>Performance goal</i>	0

**Table 4-5:** Parameters used for the training process in the neural networks experiment

Measure	Interpretation	Formula
VAF	The percentage of fit of a model with its target data	Equation 2-14
$\mathcal{V}_{Q,i}$	The VAF values for all signals in $Q$	Equation 2-15
$\zeta$	The reduction of the performance measure in percentage	Equation 2-23
$\mathcal{Z}_Q$	The reduction measures for all signals in $Q$	Equation 2-26
$\mathcal{Z}_R$	The reduction measures for all signals in $R$	Equation 2-27
$\mathcal{Z}_{R_3}$	The reduction measures for all signals in $R_3$	Equation 2-28

**Table 4-6:** An overview of all measures used to present the results of the neural networks experiments

## 4-2 Results

In this section we will present the results by first looking at how the neural networks fit the available optimal compensation signals, which we refer to as the accuracy analysis. We will investigate when the hidden layers do no longer influence the accuracy of the networks. After that, we look at what point the number of training data has impact on the accuracy. From there on we will look at the performance of the feedforward controller formed by the found networks. Similar to the accuracy analysis, we will first look at when the increasing of the hidden layers no longer influence of the performance. After that, we will look at the impact of the reducing of the training data on the performance of the neural networks.

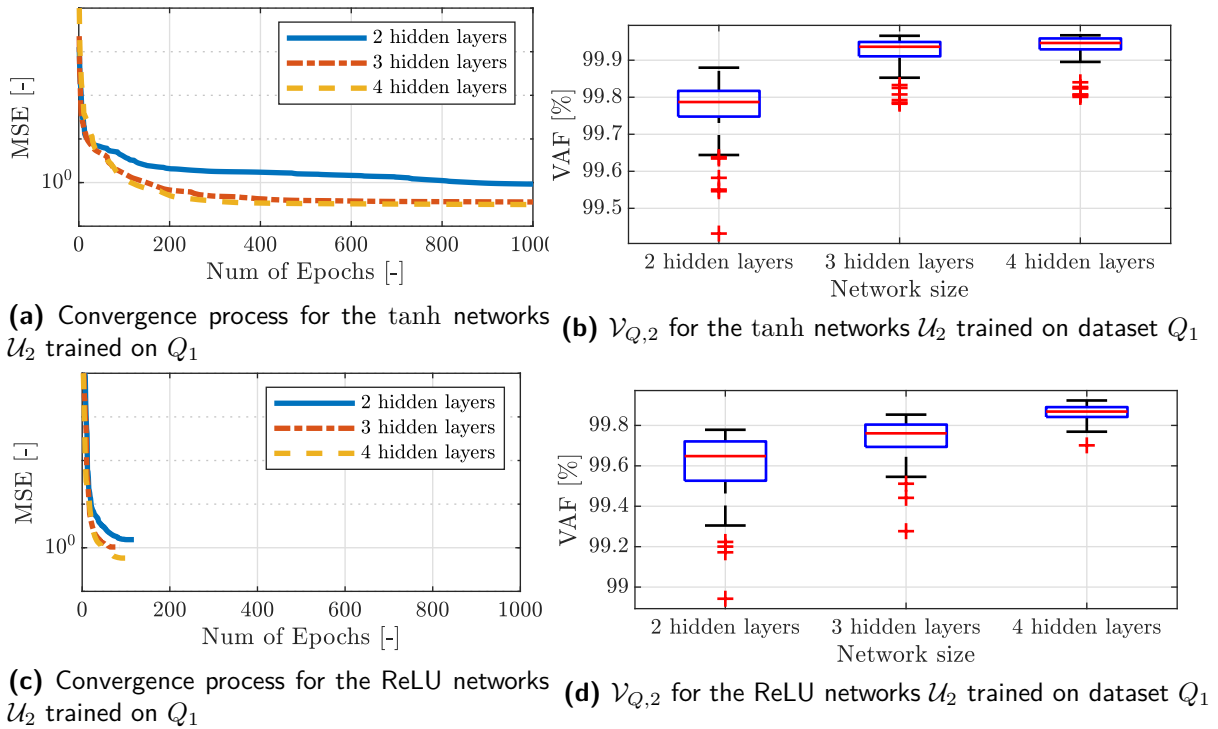
### 4-2-1 Accuracy Analysis

We will present the results by means of networks that are trained for signal  $u_2$ . However, similar observations are found for the networks representing the other signals. An overview of all results is presented in Appendix B. To show the results of the networks with varying hidden layers, we will only show the networks that have been trained on training data set  $Q_1$ . To show the results for varying training sets on the accuracy of the neural networks, we only show the networks with 3 hidden layers.

#### The influence of the hidden layers on the accuracy of the neural networks

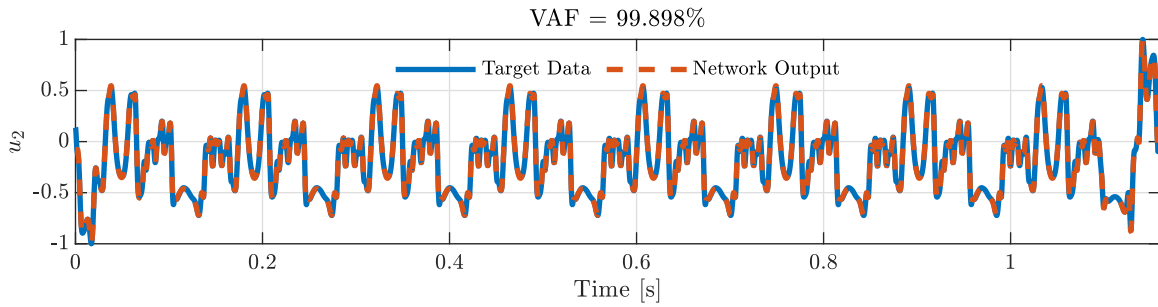
In Figure 4-2a and 4-2c the convergence processes of the tanh and ReLU networks with all sizes on dataset  $Q_1$  are displayed, where the error on the test set is given. We see that for both cases the neural network with 4 hidden layers converges to the lowest MSE.

The figures show that an increase in hidden layers results in better convergence. In both figures the difference in convergence between 3 and 4 hidden layers is less than between 2 and 3. Furthermore, we can see that the ReLU network is reaching other stopping criteria than the epoch threshold for all three sizes. For the 2 hidden layers, the training process was reached for to prevent further over-fitting. For the networks with 3 and 4 hidden layers, the maximum  $\mu$  value was reached, indicating that it is ineffective to continue training. The convergence processes of the ReLU networks are terminated at a higher MSE than the tanh networks. Figure 4-2d points at an increase in accuracy after adding a 4<sup>th</sup> hidden layer.

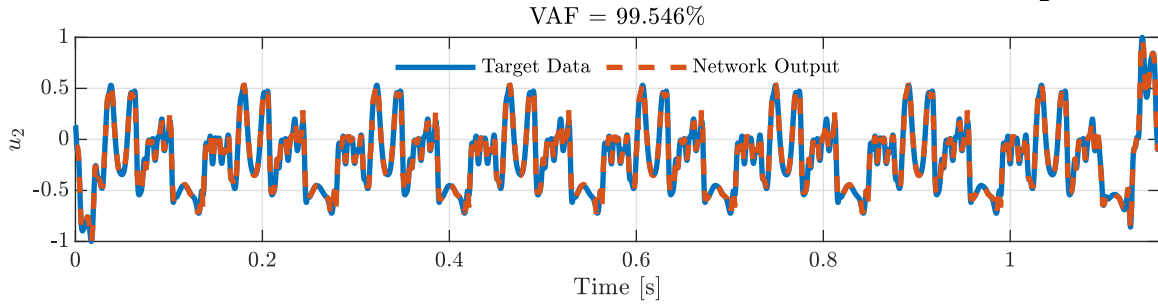


**Figure 4-2:** The influence of the hidden layers on the performance of the networks representing signal  $u_2$

The set of VAF values  $\mathcal{V}_{Q,i}$  are calculated for the three tanh and ReLU networks and are visualized in Figure 4-2b and 4-2d. The figures show a similar trend as Figure 4-2a and 4-2c; more hidden layers increases the accuracy of a neural network, as the median of the set of VAF values is higher. Furthermore, we see that the tanh network reaches higher VAF-values than the ReLU network. From Figure 4-2b it follows that the difference in performance for tanh networks with 3 hidden layers and 4 hidden layers is minimal. Figure 4-2d illustrates a difference in performance between 3 and 4 layered neural networks. However, this trend is only visible for  $\mathcal{U}_2$  models. In Figure B-8b and Figure B-8f in Appendix B this tendency is not occurring and the networks  $\mathcal{U}_1$  and  $\mathcal{U}_2$  with 3 and 4 hidden layers have similar accuracy. Figure 4-3 shows the output of both the tanh and the ReLU network with 3 hidden layers in the time domain. Both networks produce an output that fits accurately with a VAF of 99.898 % for the tanh network and a VAF of 99.546 % for the ReLU network.



(a) The output of tanh network  $u_2$  with 3 hidden layers and its target data  $u_2^*$



(b) The output of ReLU network  $u_2$  with 3 hidden layers and its target data  $u_2^*$

Figure 4-3: Time domain plots of the output of a tanh and ReLU network with 3 hidden layers.

To illustrate the performance difference, we consider a zoomed-in section of Figure 4-3. In Figure 4-4 the output of three tanh networking ascending in size are plotted together with their target data. The output is smooth and close to the target data. The difference in VAF between 3 and 4 hidden layers is minimal and hardly visible in the plots.

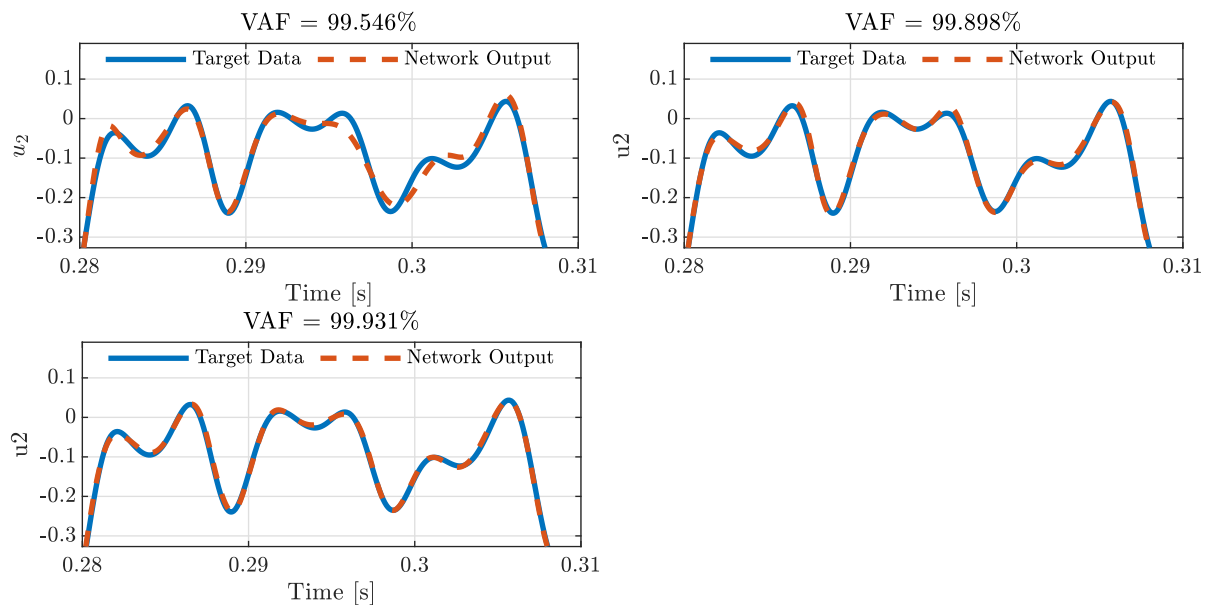
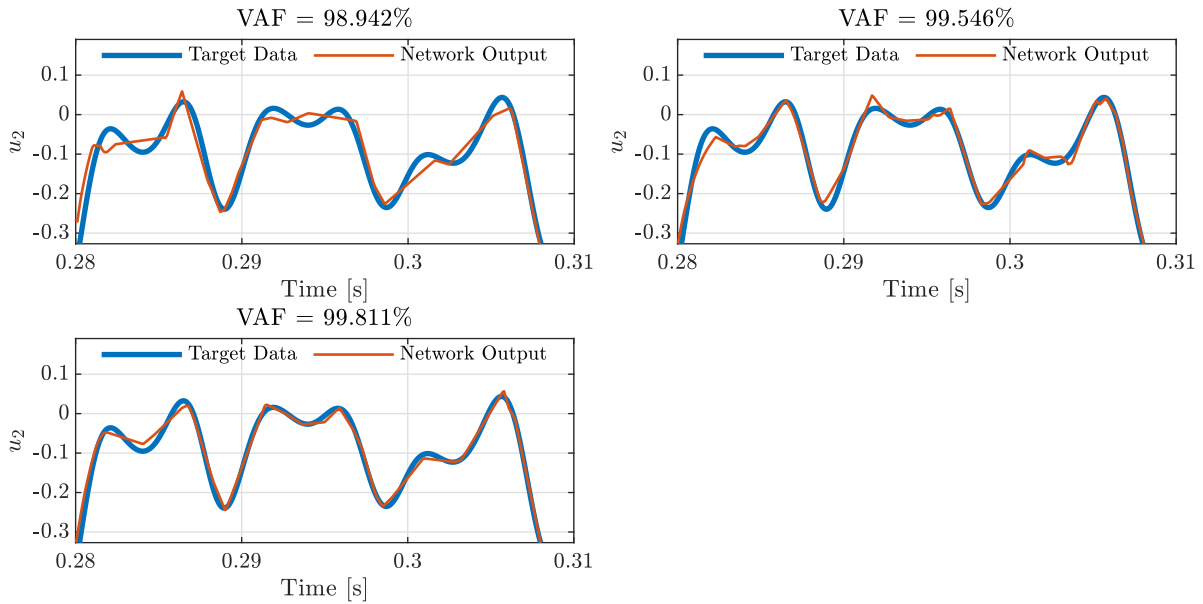


Figure 4-4: The influence of the hidden layers in time domain for the tanh networks

In Figure 4-5, the same is done for the output of the ReLU networks. First thing we notice is that the signal is not smooth. Furthermore, the size of the network has more impact on the VAF values, which is also visible in the plots.



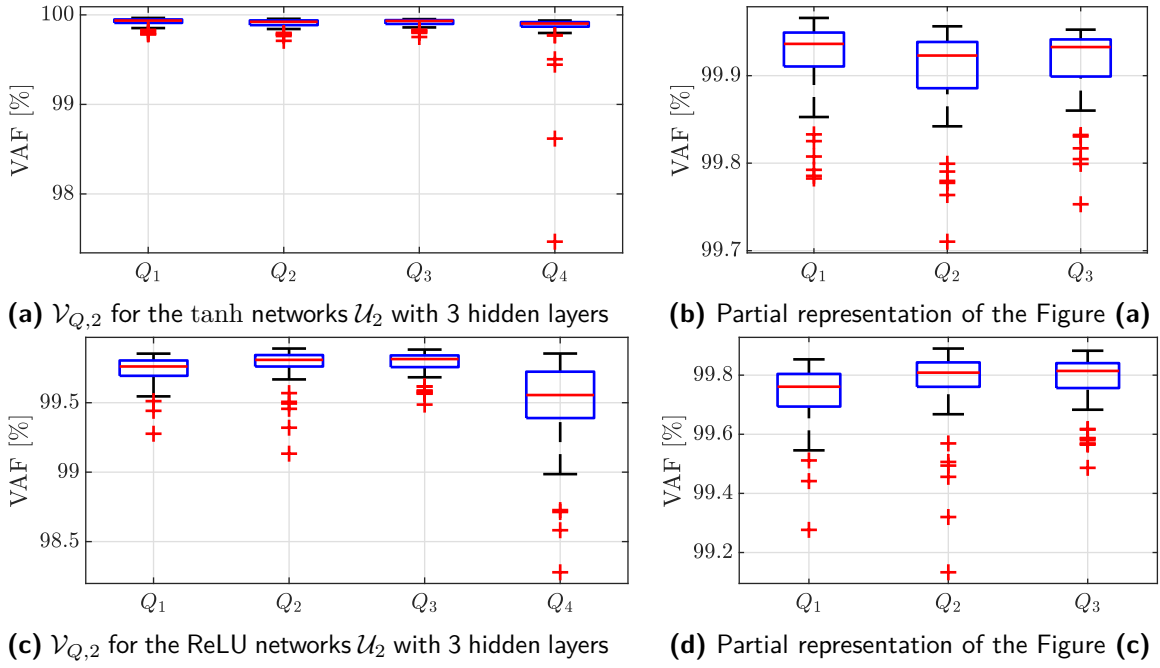
**Figure 4-5:** The influence of the hidden layers in time domain for the ReLU networks

### The influence of the training data sets on the accuracy of the neural networks

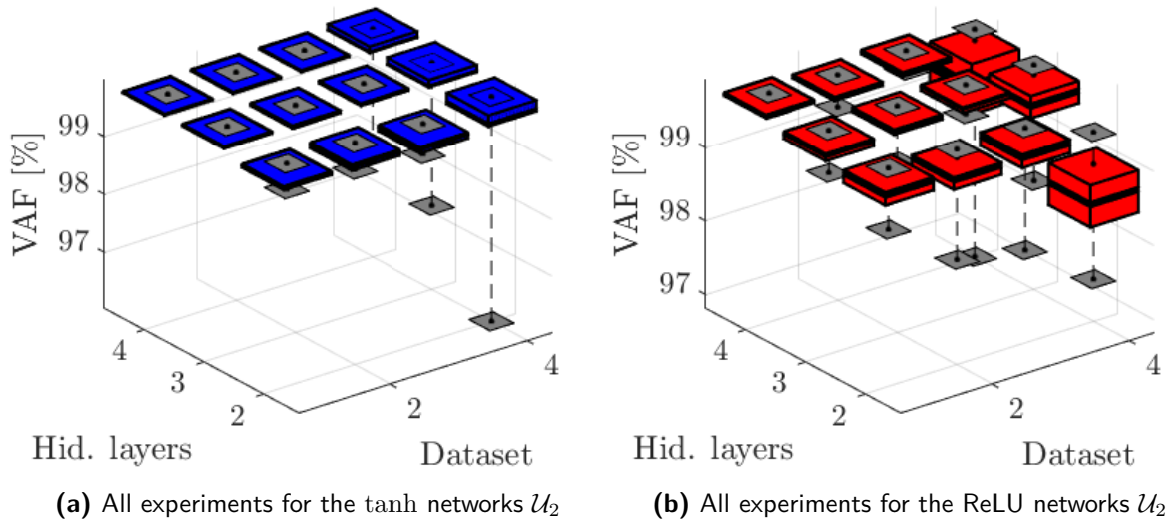
Figure 4-6 illustrates the influence of the training data on the accuracy of the tanh and ReLU network with 4 hidden layers. Please remember that the data sets are numbered in descending fashion, i.e.  $Q_1$  is the largest training data set, while  $Q_4$  is the smallest. The boxplots show that the networks trained on training data set  $Q_4$  have the set with the lowest VAF values. To get a better view on the influence of dataset  $Q_1$ ,  $Q_2$  and  $Q_3$  on the accuracy of the networks, their VAF values are plotted in Figure 4-6b and 4-6d. These figures show that there is little accuracy lost if the networks are trained on less data. Similar behavior is found for the other signals, where the networks start to predict poorly when trained on dataset  $Q_4$ . This is presented in the Figure B-5, B-6, B-7, B-12, B-13, B-14 in Appendix B.

### An overview of all the performed neural networks experiments

An overview of the experiments done for the networks  $\mathcal{U}_2$  is given in Figure 4-7. Here, the hidden layer axis refers to amount of hidden layers that is considered in the experiment, reaching from 2 to 4. The data set axis refers to the training set which was used to train on in the experiment, reaching from 1 to 4. These numbers refers to the previously presented datasets  $Q_1$ ,  $Q_2$ ,  $Q_3$  and  $Q_4$ . Three observations can be made, which are all mentioned earlier. First of all, the tanh networks in blue, are performing better, as the VAF values are higher. If we look at the results of the experiments for the ReLU networks in red, we can see that the VAF values are lower and more spread out. Secondly, we see in both cases that an increase of hidden layers comes with an increase in accuracy and that little is gained between the networks with 3 and 4 hidden layers. Finally, we can see that accuracy decreases significantly if the networks are trained on training data set  $Q_4$ .



**Figure 4-6:** An overview of the influence of the training data sets on the accuracy of the neural networks



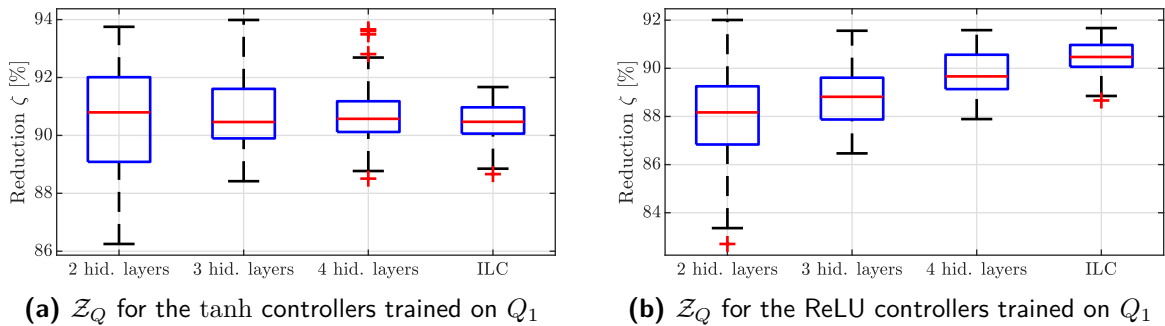
**Figure 4-7:** All results summarized for the networks  $\mathcal{U}_2$

### 4-2-2 Feedforward controller performance

The analysis of the network controllers is done by looking at the reduction of the performance measure in percentages  $\zeta$ . First, we will discuss the influence of the controllers with varying layers on the performance of the controllers. After that, we will discuss the influence of the varying training data sets on the controller performance. The analysis is split up in two. We will analyze the controllers for the training set  $Q$  and the set of signals that lay in the same plane,  $R_3$ . After that, the performance of the controller for all the setpoint signals in the validation set are considered.

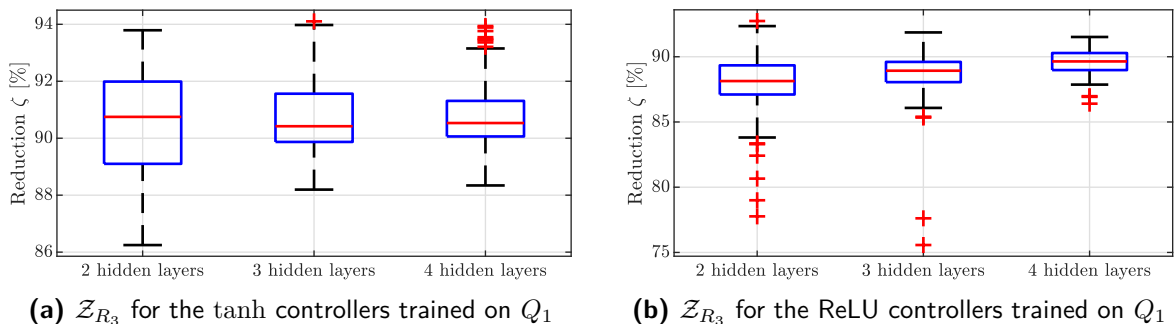
### The influence of the hidden layers on the performance of the feedforward controller

Figure 4-8 illustrates the set of performance reduction values  $\mathcal{Z}_Q$  for the feedforward controller together with  $\mathcal{Z}_{ILC}$ . Figure 4-8a illustrates that there is little performance difference for the tanh controllers with 3 or 4 hidden layers, as was already noticed in the accuracy analysis. For the ReLU controllers, there is minimal performance difference visual, as is shown in Figure 4-8b. For the tanh controllers trained on various training sets a similar trend occurs, as is presented in Figure B-16 in Appendix B, where we see that the performance of the controllers with 3 and 4 hidden layer is comparable. For the ReLU controller, the performance difference between controllers with 3 and 4 layers is varying minimally (see Figure B-16 in the Appendix). For both controller types, we can observe that the controllers with 3 and 4 layers are outperforming the controllers with 2 layers. Noteworthy is the observation that the tanh controller outperforms ILC, which should not be possible, as it is the optimal solution. For the ReLU networks, the performance is comparable with ILC. Furthermore, we can see that the ReLU controller is outperformed by the tanh controller.



**Figure 4-8:** The influence of the hidden layers on the controllers performing on all signals in training set  $Q$

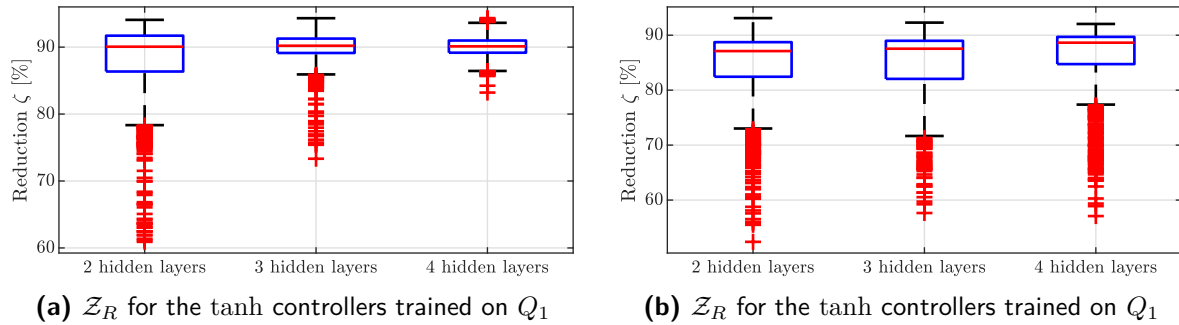
To indicate how the network behave for signals that are in the same plane as the training data, the set  $\mathcal{Z}_{R_3}$  for various controllers is presented Figure 4-9. Here, we can see a similar trend. Figure 4-9a illustrates little performance difference between the controllers with 3 and 4 hidden layers. For the ReLU controller, we can see that the largest controller results in a set of  $\zeta$ -values with less outliers compared to the set of  $\zeta$ -values for 3 hidden layers for signals in set  $R_3$ , indicating a minimal performance difference. For tanh controllers trained on other training data sets, we see that the 3 and 4 hidden layers perform similar, as Figure B-17 in the Appendix illustrates. Figure B-23 in Appendix B illustrate that the ReLU controllers with 3 and 4 hidden layers have little difference in performance for signals in set  $R_3$ .



**Figure 4-9:** The influence of the hidden layers on the controllers performing on all signals in set  $R_3$



Figure 4-10 illustrates the performance of the controller for all signals in validation set  $R$  by visualizing various sets  $\mathcal{Z}_R$  in a boxplot. Figure 4-10a shows a notable increase in performance between 3 and 4 hidden layers for the tanh controllers trained on dataset  $Q_1$ . For the tanh networks trained on the other data sets, there is no general tendency noticed as is illustrated in Figure B-18 in Appendix B. For the ReLU controller there is little performance difference between the network with 2 and 3 hidden, the controller with 4 hidden layers is performing a little better. There is once more no general trend noticeable in controller performance for the controllers trained on the other training data sets (see Figure B-24 in Appendix B).



**Figure 4-10:** The influence of the hidden layers on the controllers performing for all signals in set  $R$

### The influence of the training data sets on the performance of the feedforward controller

For the influence of the training data on the performance of the feedforward controller we can observe a similar trend. Figure B-19 and Figure B-25 in Appendix B show the different sets  $\mathcal{Z}_Q$  for various controllers. We observe a similar tendency as is found in the accuracy analysis. The controllers trained on data set  $Q_4$  fail to attenuate the force as accurate as the controllers trained on the other datasets,

Similarly, Figure B-20 and Figure B-26 in Appendix B illustrate the performance of the controllers for the signals in set  $R_3$ . We observe a comparable trend, where the controller trained on training set  $Q_1, Q_2$  and  $Q_3$  have comparable performance and the controllers trained on  $Q_4$  perform the poorest.

Finally, Figure B-21 and Figure B-27 show the sets of reduction measures  $\mathcal{Z}_R$  for various tanh and ReLU controllers. Here, the observations we made for the performance on the setpoint signals in  $Q$  and  $R_3$  are not reflected in the performance on the validation set.

### 4-2-3 Concluding remarks

- In the accuracy analysis, we noticed that adding a fourth layer to the networks improves the accuracy minimally and that a network with 3 hidden layers is accurate. This is the case for the network with the ReLU and the tanh activation function. There is a loss of accuracy noticed for the networks trained on the smallest training data set  $Q_4$ , while the networks trained on data set  $Q_1, Q_2$  and  $Q_3$  have a similar accuracy. This is observed for both type of networks.
- Similar trends are observed for the controller performance on signals in training set  $Q$  and set  $R_3$ . Adding a 4<sup>th</sup> layer increase the controller performance minimally and

a drop in performance was noticed for network controllers trained on data set  $Q_4$ .

- The tendency we noticed for the controller performance on signals from set  $Q$  and  $R_3$  are not observed for the performance on all signals in validation set  $R$ .

### 4-3 Discussion

The loss in accuracy after training on the smallest data set  $Q_4$  is noticeable as several outliers occur which are not visible for networks trained on the other 3 training data sets. One of the reasons why the accuracy drops might not necessarily be the decreasing of the number of data points but by the way they are sampled throughout space  $\mathcal{Q}$ . As was presented in the beginning, a non-uniform sampling was chosen. A network might perform better if it is trained on an uniform sampled set as it needs to interpolate less.

For the controller performance on validation set  $R$ , the trends that we observed for the performance on set  $Q$  and  $R_3$  were not found. The signals in  $Q$  and  $R_3$  vary in their  $p_3$  value, which explains the difference in performance behavior. To obtain a better performing feedforward controller, we should create a new training set uniformly sampled throughout the parameterized space. Since we can obtain the optimal compensation signals for any setpoint signal in  $P$  and as we saw that the networks are capable of approximating the target signals excellently, an uniform sampling should result in a better performing feedforward controller.

Furthermore, we noticed that the tanh controller outperforms ILC, which should not be possible. ILC is an iterative approach converging to an optimal solution. The data set we are working with might be terminated prematurely resulting in a set of sub-optimal signals. Another explanation might be the objective function. Here it is assumed to be the 2-norm of the disturbance force, but ILC might be optimized over another performance measure, explaining the sub-optimal performance.

In the next chapter, we will elaborate on how the performance of the controller cultivates throughout space  $\mathcal{P}$ . In chapter 6 we will discuss the performance difference between the ReLU and tanh controller and we will explain the excellent controller performance on signals with varying  $p_3$  values.

### 4-4 Conclusion

To finalize the experiment, we answer the research questions presented in the beginning of this chapter. A difference in controller performance on setpoint signals in set  $Q$  and  $R_3$  and the controller performance on the signals in the validation set  $R$  was noticed. Therefore, we will base the conclusion on controller performance of setpoint signals in the training set  $Q$  and  $R_3$ .

1. *What size of network is sufficient for its performance as feedforward controller?*  
We noticed that adding a 4<sup>th</sup> hidden layer to the network does not influence its performance significantly and we can conclude that the networks with 3 hidden layers suffices as feedforward controller.
2. *When does the amount of data points influences the performance of the feedforward controller significantly?*

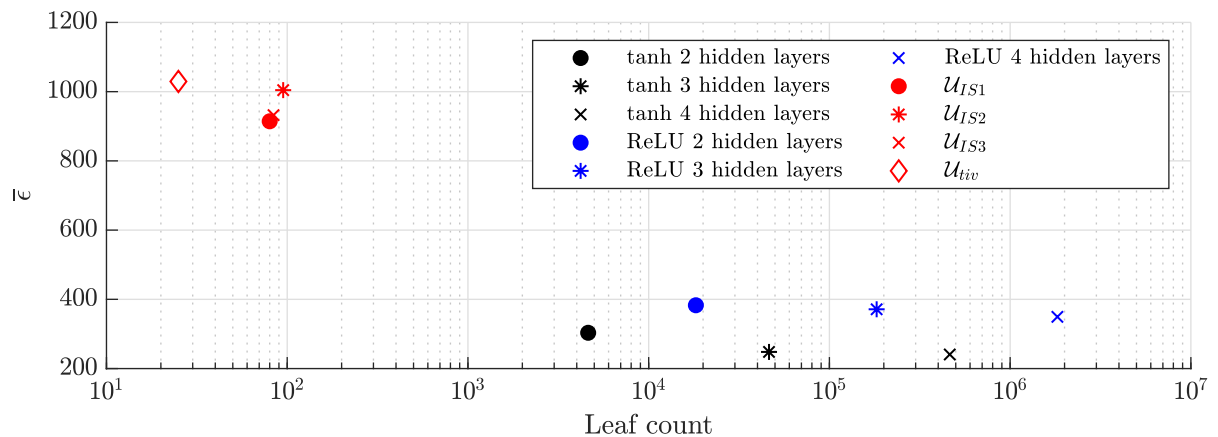
There was little performance difference for feedforward controllers constructed from neural networks trained on dataset  $Q_1, Q_2$  or  $Q_3$ . Training the neural networks on 18 signals resulted in a drop in accuracy and at last in feedforward controller performance.



## Feedforward controller comparison

In this chapter, we will explore the trade-off that arises when choosing a nonlinear data-driven modeling approach by rating the previously designed controllers on their performance and their size. After that, we will compare the controllers based on the four qualities presented in Chapter 1. We extend on the controller performance by investigating the reduction throughout the setpoint signals  $P$ . We discuss the advantages of the whitebox models returned by genetic programming. Furthermore, we will present the found output bound for the ReLU controller and finally, we compare the impact of the amount of training data on the data-driven modeling methods.

We will start by visualizing the trade-off in the designed controllers by rating them on complexity and performance. The performance of a controller is expressed as the mean of the set  $\mathcal{E}_R$  defined in Equation 2-20 obtained by the controller. If the controller obtains a lower  $\bar{\epsilon}$ , this refers to an on average better performing controller, as the mean of set of performance values is lower. To rate the complexity of the controller, the leaf count is used. This measure counts the leaves if a function is written as an expression tree. It can be referred to as the total number of indivisible sub-expressions in an expression. To illustrate how the previous designed controllers are rated on the performance and complexity, they are provided in Figure 5-1.



**Figure 5-1:** a Pareto front of the designed feedforward controllers

If we focus on the genetic programming controllers, visualized in red, we can observe

that these have a lower complexity but have a higher  $\bar{\epsilon}$  and therefore have a poorer performance. We can see that the network controllers visualized in blue and black are performing better, as they result in lower mean values of the set of performance measures. Nevertheless, the superior performance comes with the increase in their complexity, as they have a higher leaf count. Furthermore, the figure confirms that the tanh networks outperform the ReLU networks based on this multi-objective rating, as they have a lower complexity and a better performance, which we noticed earlier in Chapter 4.

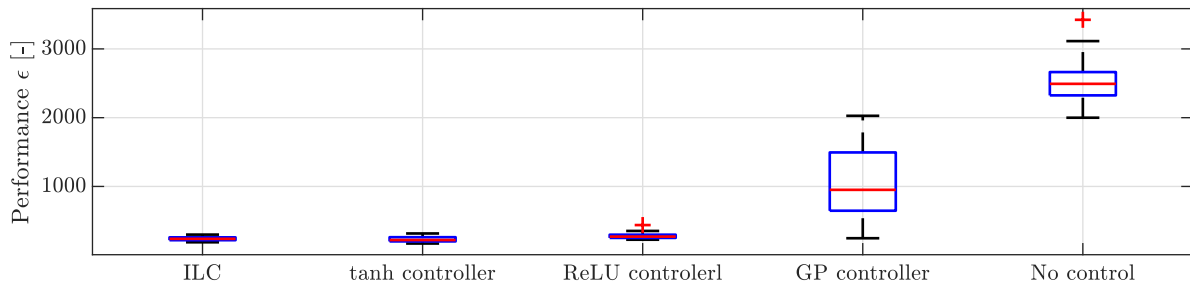
## 5-1 Comparing the feedforward controllers from a performance perspective

In this section, the designed controllers are compared on their ability to attenuate the disturbance force. For this performance comparison, genetic programming controller  $\mathcal{U}_{IS3}$  is considered as we observed in Chapter 3 that it reached the highest minimum reduction. Moreover, network controllers with 3 hidden layers and trained on dataset  $Q_1$  are considered as we concluded in Chapter 4 that this size is sufficient. The measures that are used in this section together with their equation in which they are defined are given in Table 5-1.

Measure	Interpretation	Formula
$\epsilon$	The performance measure given as the 2-norm of the disturbance force	Equation 2-16
$\mathcal{E}_Q$	The performance measures for all signals in $Q$	Equation 2-20
$\mathcal{E}_0$	The performance measures for all signals in $Q$ if no control is applied	Equation 2-21
$\mathcal{E}_{ILC}$	The performance measures for all signals in $Q$ if ILC is applied	Equation 2-22
$\zeta$	The reduction of the performance measure in percentage	Equation 2-23
$\mathcal{Z}_Q$	The reduction measures for all signals in $Q$	Equation 2-26
$\mathcal{Z}_R$	The reduction measures for all signals in $R$	Equation 2-27
$\mathcal{Z}_{R_3}$	The reduction measures for all signals in $R_3$	Equation 2-28

**Table 5-1:** An overview of all measures used to present the comparison between the controllers

The different sets of performance measures  $\mathcal{E}_Q$  obtained for the controllers are given in Figure 5-2. The performance of the optimal compensation signals found with ILC  $\mathcal{E}_{ILC}$  is also provided. Furthermore, the set of performance measures after applying no control  $\mathcal{E}_0$  is given, to envision the proportion of the disturbance forces.

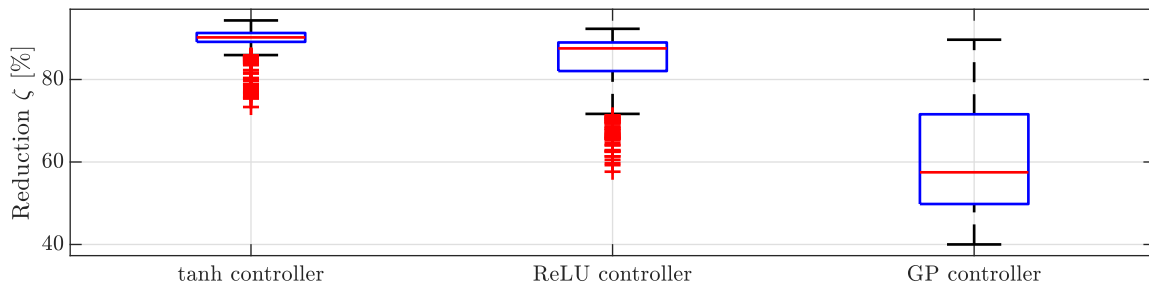


**Figure 5-2:** The set of performance measures for the controller on the setpoint signals in the training set  $R$

When comparing the various sets of performance measures, we observe that the disturbance measures are the highest if no control is applied, indicating that the controllers are reducing the disturbance force. Moreover, we see that the tanh controller is performing

similar to ILC. Furthermore, we see that the  $\epsilon$  values for the ReLU network controller are higher than the tanh controller and one outlier is occurring. We see that the GP controller reduces the median of the set of disturbance measures, but is not reaching the same performance of the other three controllers.

Figure 5-3 visualizes the set of  $\zeta$  values  $\mathcal{Z}_R$  for all three controllers. We can observe that each controller is capable of reducing the performance measures for all signals in the validation set as the worst performing controller is reducing the performance measure with at least 40.01 %, hence satisfying Condition 1-2.1 presented in Chapter 1. We can see that the tanh controller has the highest performance, followed by the ReLU controller. The GP controller reduces the disturbances the least. Table 5-2 shows the extreme values for the reduction per controller.



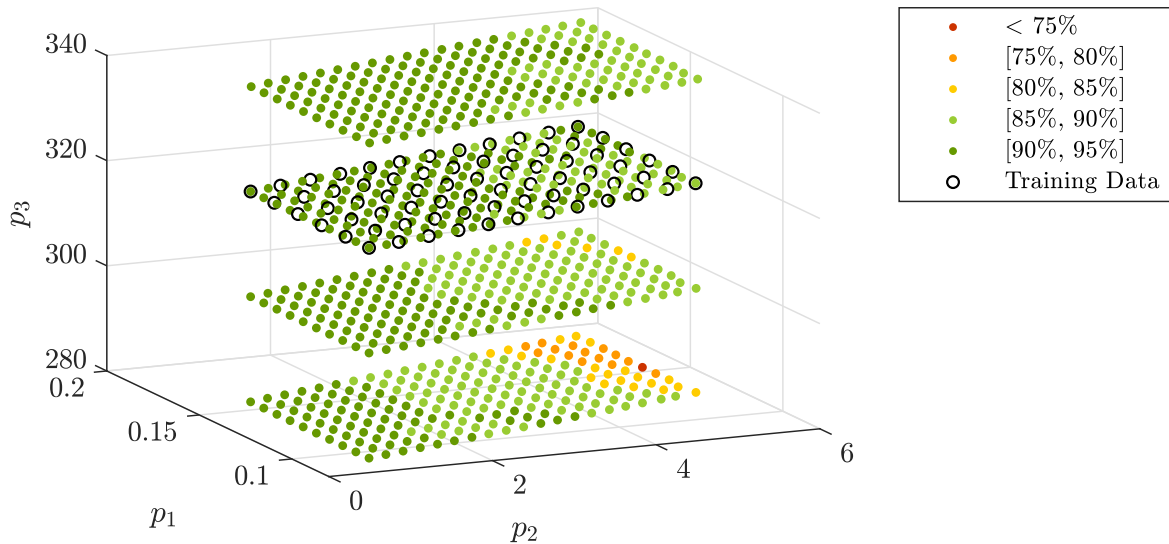
**Figure 5-3:** The set of reduction measures for the feedforward controllers on all setpoint signals in validation set  $R$

Reduction	tanh controller	ReLU controller	GP controller
$\min(\mathcal{Z}_R)$	73.33 %	57.64 %	40.01%
$\bar{\mathcal{Z}}_R$	89.76 %	84.69 %	61.23 %
$\max(\mathcal{Z}_R)$	94.32 %	92.29%	89.65 %

**Table 5-2:** An overview of the maximum, mean and minimum percentage of reduction in  $\mathcal{Z}_R$  for each controller

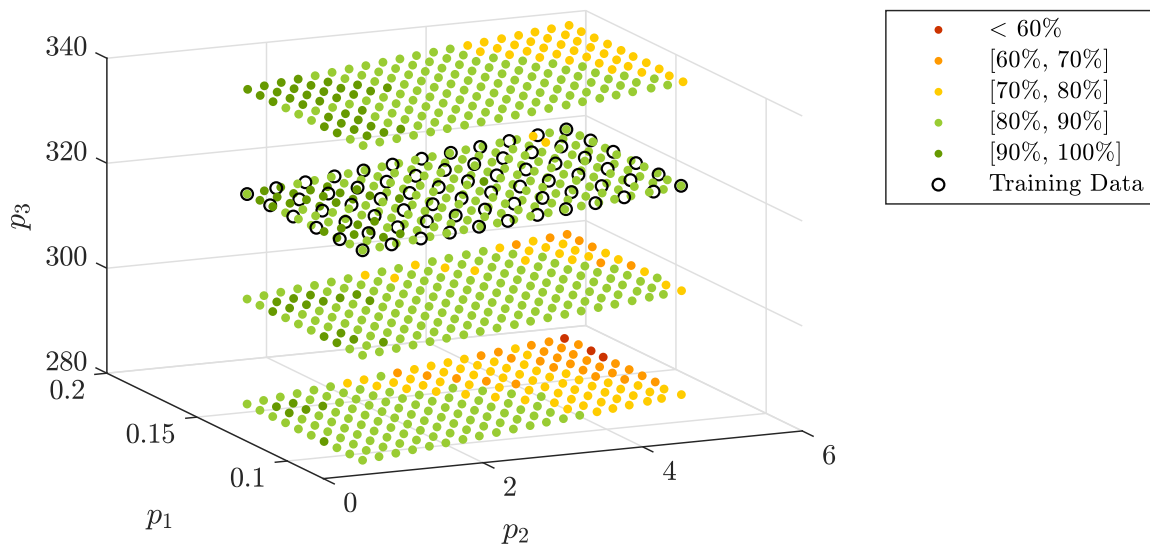
In Chapter 4 we observed a difference in how the controller performs on the setpoint signals that lay in the same plane as the training data and how it performs for the other signals in the validation set  $R$ . We will investigate the performance for all three controllers throughout space  $\mathcal{P}$ .

Figure 5-4 illustrates the reduction percentage per signal plotted in the parameterized space  $\mathcal{P}$  for the tanh controller. The  $\zeta$  value corresponding with each point in  $\mathcal{R}$  is visualized with a color, where each color represent a range of percentage reduction values i.e. the dark green dots visualize the setpoint signals for which the controller reduces the corresponding disturbance force between 90 and 95 %. In black the training data set is plotted. We can see that signals with lower  $p_3$  values are reduced the poorest. Furthermore, the controller performs the best for signals with lower  $p_2$  values as these signals are reduced with the highest percentages.



**Figure 5-4:** The performance of the tanh controller expressed in percentages of reduction visualized throughout parameterized space  $\mathcal{P}$

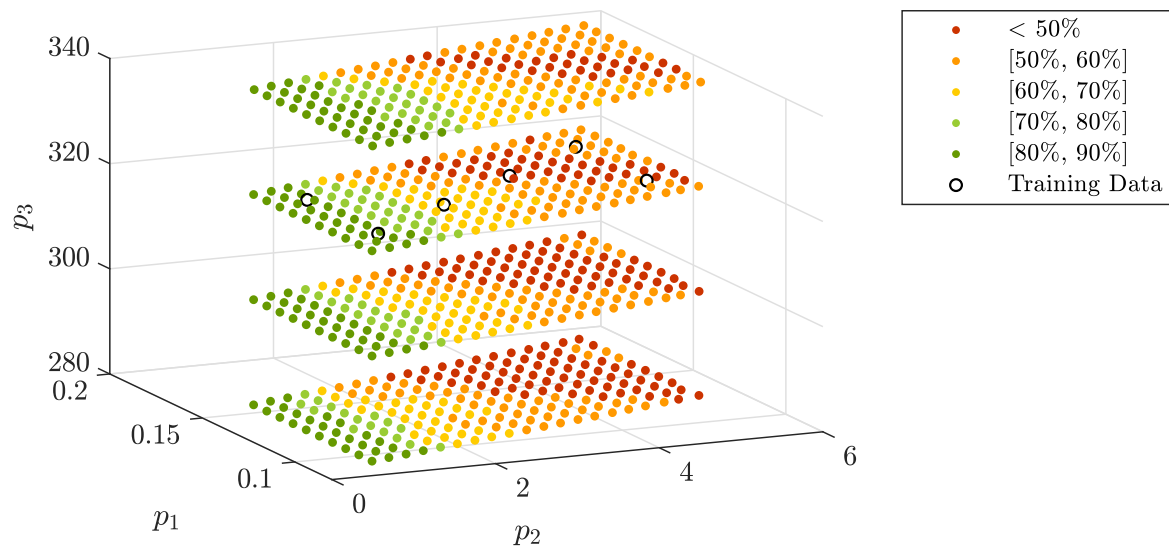
In Figure 5-5 the performance of the ReLU controller is plotted throughout the parameterized space  $\mathcal{P}$ . Please note that the colors correspond with different percentage ranges than in Figure 5-4. In line with Figure 5-3 we can see that the percentages are lower than those found for the tanh controller. We can again see that the poorer outliers occur further away from the training data. The signals for which the ReLU controller reaches  $\zeta$  values between 90% and 100% are found once more for the signals with lower  $p_2$  values.



**Figure 5-5:** The performance of the ReLU controller expressed in percentages of reduction visualized throughout parameterized space  $\mathcal{P}$

Figure 5-6 gives the percentage of reduction visualized in the parameterized space  $\mathcal{P}$  for the GP controller. We observe that the signals with the lower  $p_2$  values are reduced the most. Secondly, we can see that the signals for a reduction of 50 % occur more often for signals that have a lower  $p_3$  value than 320. Generally, the further away we are from plane  $R_3$ , the poorer the performance measure is reduced. Lastly, in each plane  $R_1, R_2, R_3$  and  $R_4$  red points occur in a similar diagonal trend.





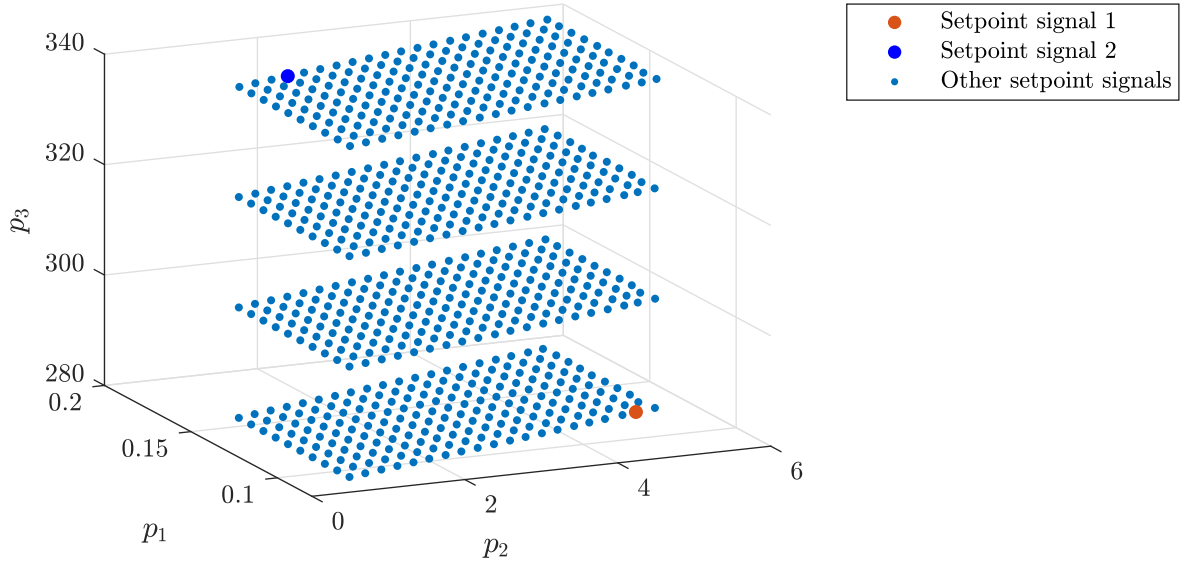
**Figure 5-6:** The performance of the GP controller expressed in percentages of reduction visualized throughout parameterized space  $\mathcal{P}$

## 5-2 Comparing the feedforward controllers from a complexity perspective

As we noticed in Figure Figure 5-1, the genetic programming models are less complex than the network controllers. In this section we will use the trivial models found with genetic programming to extend on the insight in the problem. After that, we will compare the memory usage and investigate the computation time of the designed controllers .

### 5-2-1 Gaining insight in the behavior of the compensation signals in the different planes

The analytic expressions found with genetic programming can be used to understand how the compensation signal develops throughout parameterized space  $\mathcal{P}$ . subsection 3-2-1 accumulated an example, in which we used a set of trivial models to investigate the known compensation signals in  $\mathcal{Q}$ , corresponding with plane  $\mathcal{Q}$  in the parameterized space  $\mathcal{P}$ . We concluded that a set of first and second-order polynomials explained the peaks in the compensation signal and discovered a nonlinear component dependent on  $p_2 = \max(\dot{r})$  which was not captured by these trivial models. In this subsection we will use these same trivial models to get an intuition in the behavior of the compensation signals in the other planes  $\mathcal{R}_1, \mathcal{R}_2$  and  $\mathcal{R}_3$ . The output of the tanh networks are investigated, as we do not know any optimal compensation signals for the setpoint signals in these planes. The tanh controller is the best available approximation of the optimal compensation signal. We will investigate the behavior of the tanh network using two setpoint signals. The first setpoint is chosen in plane  $\mathcal{R}_4$  with a higher  $p_2$  value for which the performance measure is reduced by 85.69 % percent, whereas the other is chosen in plane  $\mathcal{R}_1$  with a lower  $p_2$  value where the performance measure corresponding with setpoint signal 2 is reduced by 93.63 % percent. Figure 5-7 illustrates the two setpoint signals chosen for further analysis.



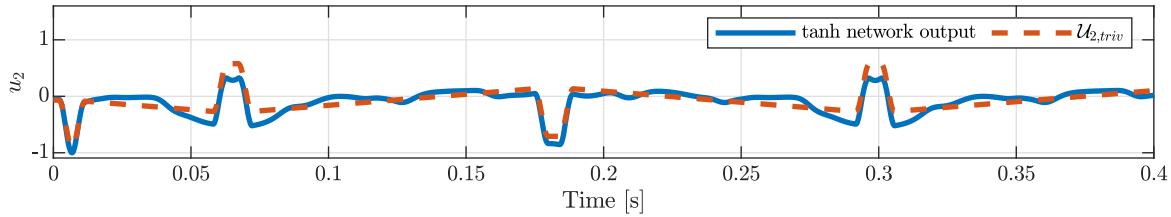
**Figure 5-7:** The two setpoint signals used to gain insight in the behavior of the tanh network output

The setpoint signals are used to compare the output of network model  $\mathcal{U}_2$  with the output of the trivial model found by genetic programming. The trivial model is presented in Equation 5-2, which is a first-order polynomial of  $\ddot{\mathbf{r}}$  and  $\mathbf{r}$ . Figure 5-8 illustrates the output of the tanh network and the trivial model for both setpoint signals.

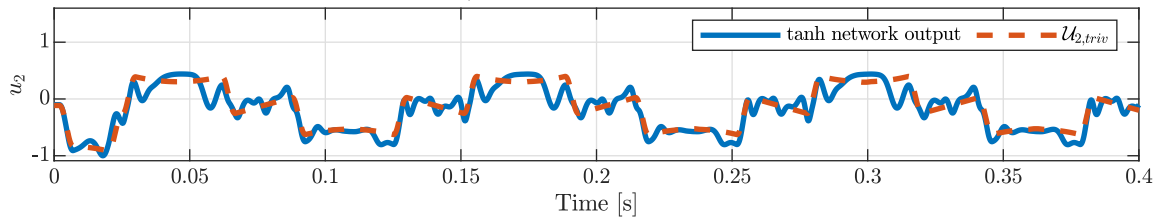
$$\mathbf{u}_1 = \mathcal{U}_{1,triv}(\mathbf{r}, \ddot{\mathbf{r}}) = 2.69 - 3.43e^{-3}\ddot{\mathbf{r}} - 33.53\mathbf{r} \quad (5-1)$$

$$\mathbf{u}_2 = \mathcal{U}_{2,triv}(\mathbf{r}, \ddot{\mathbf{r}}) = 0.44 - 0.16\ddot{\mathbf{r}} - 155.70\mathbf{r} \quad (5-2)$$

$$\mathbf{u}_3 = \mathcal{U}_{2,triv}(\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}) = -3.29e^{-2} - 0.19\dot{\mathbf{r}} + 6.54e^{-2}\ddot{\mathbf{r}} \quad (5-3)$$



**(a)** The output of  $\mathcal{U}_{2,triv}$  and the tanh network for setpoint signal 1



**(b)** The output of  $\mathcal{U}_{2,triv}$  and the tanh network for setpoint signal 2

**Figure 5-8:** Analyzing the output of the tanh neural network  $\mathcal{U}_2$  with 3 hidden layers for signals outside the training set

Based on Figure 5-8 we can make two observations. First of all, we can see that the trivial model follows the general trend of the output of the tanh network. Secondly,

there is a difference in the appearance of the tanh network output for the two setpoint signals, which vary strongly in their in  $p_2 = \max(\dot{\mathbf{r}})$ . This difference in appearance is not explained by the trivial model  $\mathcal{U}_{1,triv}$ , which suggest that the compensation signals depend nonlinear on  $\dot{\mathbf{r}}$ , but the general trend of the compensation scales linearly with  $\mathbf{r}$  and  $\ddot{\mathbf{r}}$ .

Similarly, the output of the tanh networks  $\mathcal{U}_1$  and  $\mathcal{U}_3$  for the presented setpoint signals are compared with the output of the found trivial models  $\mathcal{U}_{1,triv}$  and  $\mathcal{U}_{3,triv}$ . The plots which visualize this comparison can be found in Figure B-28 and Figure B-30 in Appendix Appendix B, where we notice that the main trends of the output of the tanh network  $\mathcal{U}_1$  is roughly followed by the trivial model. For the network  $\mathcal{U}_3$  the trivial model follows the signal in a weaker fashion.

Throughout this study, the compact analytic models found by genetic programming are used in two occasions to get an intuition to see how the compensation signals are behaving, which is not possible for the network models. This can be seen as an advantage of models with lower complexity.

### 5-2-2 The memory consumption of the feedforward controllers

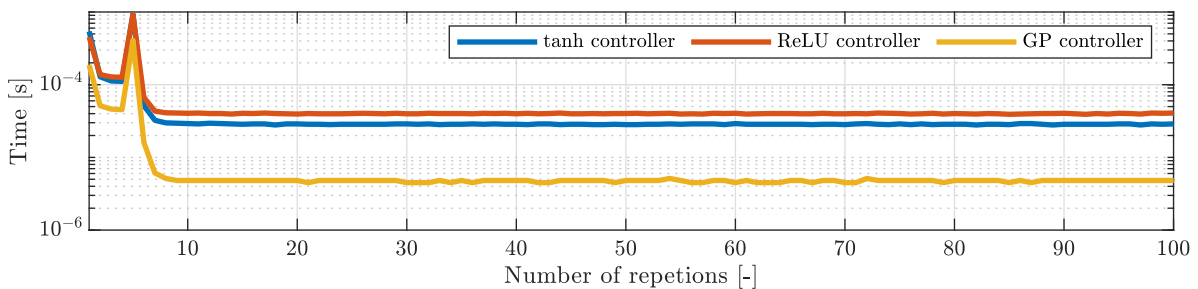
Table 5-3 visualizes the memory consumption of all three controllers. We can see that the GP controllers needs the least kilobytes to be stored, which can be critical depending on the platform the controller is implemented in. The leaf count of the controller is directly reflecting in the memory consumption.

	tanh controller	ReLU controller	GP controller
Memory Consumption [kB]	24.15	71.17	1.28

**Table 5-3:** The memory consumption of the three feedforward controllers

### 5-2-3 The computation time of the feedforward controllers

The complexity of a controller influences the time it takes to compute the next control input. If the computation time is higher than the sampling time of the machine, the controller cannot be implemented in real-time. Therefore, the computation times of the three controllers are investigated. The controllers are compiled as .mex files, which are executable C programs that can be called from the matlab interface. By turning the controllers into executable C programs, the computation time is sped up and a more realistic real-time scenario is created. Each controller is called a 100 times. The experiment is conducted on an Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz. Figure 5-9 illustrates the computation time for the controllers for each iteration.



**Figure 5-9:** The calculation time the three feedforward controllers

We can see that the computation time is above the sampling time of  $h = 1e^{-4}s$  for all three controllers in the first 8 iterations. After the first iterations we can see that the computation time stabilizes. If we look at the computation time after the 10<sup>th</sup> iteration, we do see the difference in complexity return in the computation time. The simplest GP controller has the fastest computation time, where as the most complex ReLU controller has the longest computation time. Furthermore, we can observe that after stabilization, all 3 controller are implementable in real-time.

### 5-3 Comparing the feedforward controllers controllers on other aspects

The controller design methods are compared on two more aspects. First of all, we will use SHERLOCK to find the output range of the ReLU controller. After that, we will discuss the influence of the number of training data on the performance of the controllers.

#### 5-3-1 Output bound verification of the feedforward controllers

When comparing the ReLU network with the tanh network based on complexity and performance, we can conclude that the first mentioned is larger in size and performs poorer. However, the ReLU network facilitates the use of SHERLOCK, which provides a guaranteed upper and lower bound on the output given a compact input set. First, the bounds on the input range are computed by finding the maximum and minimum of  $\mathbf{r} \dots \ddot{\mathbf{r}}$  for all signals in data sets  $Q$  and  $R$ . After that, the measured output bounds are computed by running all setpoint signals in the training and validation set through the network and storing the maximum and minimum output of the network. The output range found by SHERLOCK is expressed as fraction of the measured output bound and is provided for all three networks in Table 5-4.

	Signal $u_1$	Signal $u_2$	Signal $u_3$
SHERLOCK output bounds factor	[1.7627, 2.3809]	[2.8229, 1.0587]	[2.5696, 4.8320]

**Table 5-4:** The guaranteed output bounds found with SHERLOCK for the ReLU controller with respect to its measured output bounds

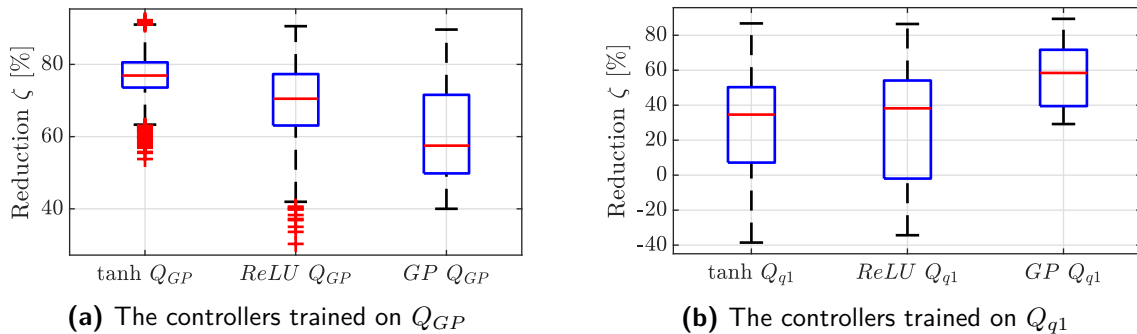
The worst output bound is found at 4.8320 higher than the measured upper bound for network  $vu_3$ . SHERLOCK cannot reconstruct which signal will trigger these output bounds. For the GP and tanh controller we are also capable of finding an output bound. These methods will be discussed in Chapter 6.

#### 5-3-2 The influence of the number of training data points on the performance of the feedforward controllers

Collecting training data is an expensive procedure, due to the down-time of the machine which is caused by the learning of compensation signals. Therefore, controllers that are constructed with less training data are favorable. The implementation of genetic programming limits the training data usage. Accordingly, the GP controllers are not designed on the same data set as is used to design the network controllers, which makes their performance comparison unfair. In this subsection we will investigate the impact

of the training data on the performance of the feedforward controller by training a set of tanh neural network and ReLU neural network on the training set  $Q_{GP}$  used for the search of genetic programming. To extend the training data analysis, we train a tanh and ReLU controller on a training data set consisting of one signal  $Q_{q1}$ , which is down-sampled and for which the redundant periods are removed as is described in Chapter 3. The networks are trained and analyzed in the same way as is described in Chapter 4, where we consider tanh networks with 3 hidden layers with 10 neurons each and ReLU networks with 3 hidden layers of 20 neurons each. Furthermore, an extra genetic programming experiment is performed. Since Genetic Programming finds models by analyzing the fit of given equations with its provided data, we expect it to find the structure of the trivial models on data set  $Q_{q1}$  as well. Since the trivial models are low in complexity and high in accuracy, it is expected to be found fast. Accordingly, we let the algorithm run for 100 generations of 300 models per generations. We provide the algorithm with  $\mathcal{R}_{r1}$  and  $\mathcal{M}_1$ .

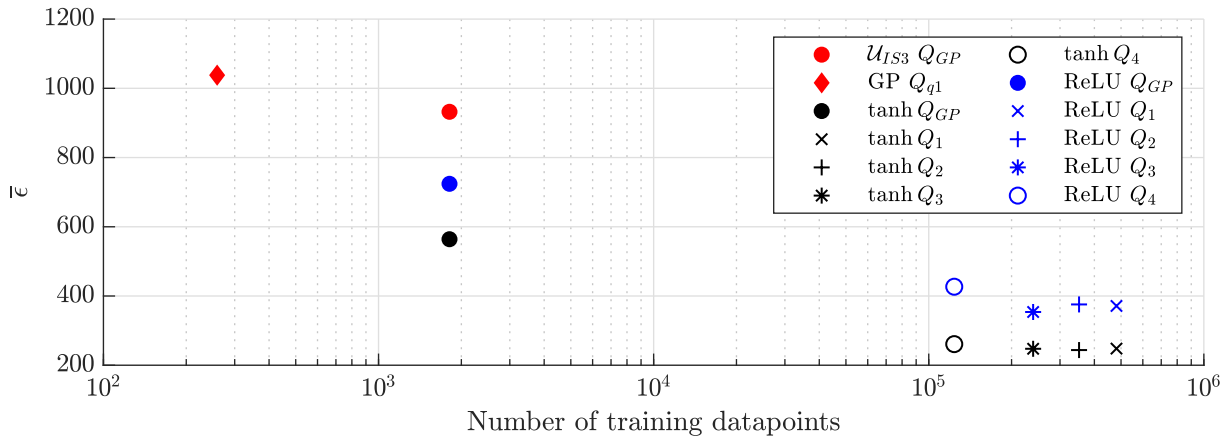
The set of reduction measures  $\mathcal{Z}_R$  for the network controllers and GP controllers found with  $Q_{GP}$  and  $Q_{q1}$  are given in Figure 5-10. Figure 5-10a illustrated that the tanh controller trained on  $Q_{GP}$  still outperforms the GP controller. The ReLU controller reduces most signals better than the GP controller. Now if we compare the  $\zeta$ -values for the network controllers trained on  $Q_{GP}$  with the previous shown values in Figure 5-3, we can see that the training on less data comes with a significant decrease in performance. Figure 5-10b illustrates that the network controllers trained on  $Q_{q1}$  fail to reduce all the signals in validation set  $R$ , as it returns negative  $\zeta$  values. This means these controllers do not satisfy Condition 1-2.1 and do not meet the design criteria.



**Figure 5-10:** The sets  $\mathcal{Z}_R$  for the controllers trained on dataset  $Q_{GP}$  and  $Q_{q1}$

For two out of the three signals, genetic programming managed to find the same model structure as the trivial models presented in Chapter 3. The parameters within the structure vary minimally from these, as they are found with least squares on  $Q_{GP}$  instead of  $Q_{q1}$ . The controller found on  $Q_{q1}$  performed similar to  $\mathcal{U}_{triv}$ .

Figure 5-11 illustrates the influence of the number of training data points on the performance of the controller, which are rated on their performance and the number of training data. The networks trained on  $Q_{q1}$  did not meet the design criterion and are therefore not considered in the figure.



**Figure 5-11:** The influence of the number of training data points on the performance of the controller

We observe a significant correlation between the number of datapoints and performance for the network controllers, whereas for genetic programming the degradation in performance is less if the number of data points is reduced.

## 5-4 Concluding Remarks

Now to conclude the chapter we will present the main observations done in this chapter:

- Controller performance*

All 3 feedforward controller reduce the performance measure for all signals in validation set  $R$  and  $Q$ . The tanh controller reduces the disturbance the most, followed by the ReLU controller. The genetic programming reduces the performance measure the least. Furthermore, we observed that the network controller decrease in performance for signals that are further away from the training data.
- Model complexity*

The compact models found by genetic programming is used to investigate how the network controller output evaluates throughout the parameterized space. We observed that the general trend of the output of the network is explained by a set of trivial models. Furthermore, we investigated the effect off the model complexity on the implementation off the controller. The simplest models resulted in less memory consumption. Furthermore, we investigated the computation time of the controllers, where we observed that the complexity of the controller reflects in its computation time.
- Output bound verification*

SHERLOCK was used to find a guaranteed output bound for the ReLU network, which lead to a maximum output bound of 4.8320 times higher then measured. There are methods to determine the output bounds for both the GP and tanh controller which will be discussed in the next chapter.
- Training data set*

The data provided to the genetic programming is limited in its size by the implementation. We saw that a tanh controller trained on  $Q_{GP}$  resulted in a controller

that is still outperforming the GP controller. However, a significant drop in performance is noticed. The network controllers trained on  $Q_{q1}$  failed to return a controller that satisfy the design criterion, while the controller found on  $Q_{q1}$  with genetic programming succeeded.





---

## Chapter 6

---

# Discussion

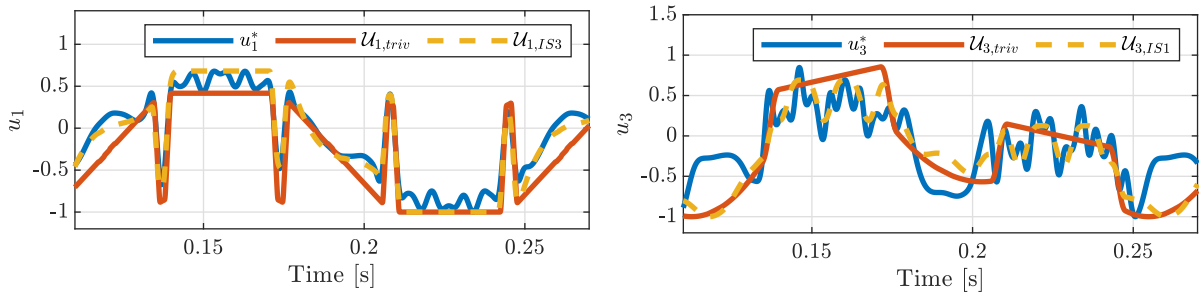
In the thesis we designed three types of feedforward controllers to generalize the feedforward control signals that are found with iterative learning control. We will discuss the difference in performance between the various controllers. We emphasize the complexity of the controller, by mentioning extra advantages of low complexity controllers. Furthermore, we will expand on the output bound verification for the designed controllers and we will discuss the impact of the training data on the performance of the controllers.

### **6-1 On the performance of the feedforward controllers**

In this section the limitation of the accuracy in the models found with genetic programming is discussed. Furthermore, we will discuss the difference in performance between the two types of neural networks and an explanation is given for the performance of the network controllers on signals outside the training set.

#### **6-1-1 The limitation in accuracy for the models found with genetic programming**

In section 2-3 it was discovered that the compensation signals are roughly explained by a set of first and second order polynomials build from the setpoint signal and its derivatives. We also detected a component in the compensation signal which becomes more dominant as the setpoint signals increase in their maximum velocity, which was not captured by first and second order polynomial models. The increase in fitness between the trivial models and models with higher accuracy found by genetic programming is obtained by either by tracking the mean of the component, as is shown in Figure 6-1a or by roughly tracking the component, as is shown in Figure 6-1a.



(a) The output of genetic programming models  $\mathcal{U}_1$  with varying complexity (b) The output of genetic programming models  $\mathcal{U}_3$  with varying complexity

**Figure 6-1:** Visualizing the component dependent on  $p_2$  and the way in which more complex GP controllers approximate it

With the current algorithm settings, genetic programming was unable to find models that fit the region dependent on  $\max(\dot{r})$ . The reason for this might be because it is dependent on a variable that is not explicitly available in the search space or the component is described by a model that has a high complexity, which the algorithm failed to find. The neural network models are capable of fitting the compensation signals, which means that an accurate model is in fact in the search space of  $\mathcal{M}_3$  and  $\mathcal{R}_{r1}$  for the genetic programming experiments.

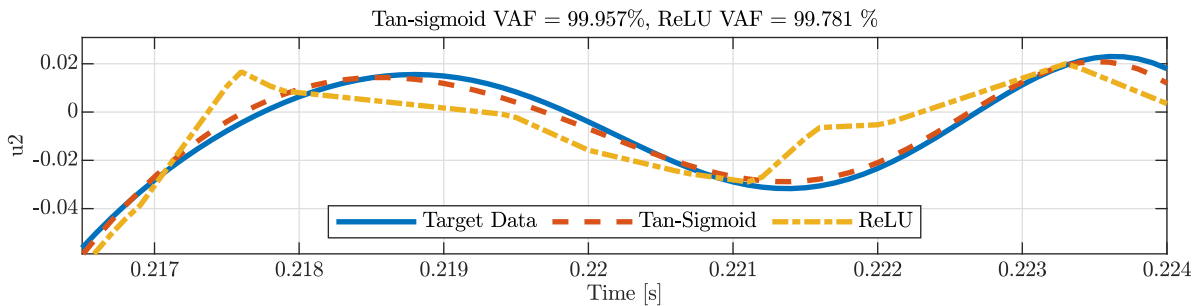
Theoretically, genetic programming is capable of finding a neural network as a model. Nevertheless, it is highly unlikely that the implementation of genetic programming used in this study is capable of recreating a neural network for two reasons. First of all, it is improbable that the neural network structure is found with genetic programming. The algorithm is free to create any model structure possible from its mathematical operators and input variables. Since a network without trained parameters is high in complexity and low in fitness, it is unlikely that it is discovered in the multi-objective optimization. Secondly, the parametric optimization used in the GP experiments is least squares, which is not adequate for training a nested expression as a neural network. Therefore, the parameters in a network are drawn randomly from a range of values. The chance that genetic programming finds the structure of a network in which all parameters are drawn correctly at random is nil.

The experiments in Chapter 3 are designed to search a large piece of the search space, by repeating the algorithm for 1000 generations and by selecting a population size of 300. Nevertheless, the algorithm was only able to find models with limited accuracy. The evolutionary runs might get stuck at a local optima, however, the use of model age as a secondary objective should help prevent this. Nevertheless, we can choose to tune the parameters within the algorithm to balance the exploration vs exploitation trade-off with the idea to find more accurate models. Another approach to find more accurate models is the implementation of grammar-based genetic programming [31], which provides the user with the possibility to guide the search. However, this only helps if more information about the compensation signals is known, such that the algorithm can search with a grammar that is beneficial. Secondly, we could consider to search for recurrent models, where we explicitly expect the mapping to depend on its previous output,  $u_1(k+1) = \mathcal{U}_1(\rho_{r1}(k), u_1(k))$ . To analyze the fitness of each expression in such an approach, we simulate the output for the differential equation and compare it with the target data. The networks approximate the target data excellently and are therefore able to capture the component which was not explained by the models found with genetic programming. This shows that recurrent models are not necessary. However, the network models are

large in complexity and therefore differ from the models found with genetic programming. By searching for recurrent models, genetic programming might be able to find more compact models that have a high fit.

### 6-1-2 The difference in accuracy of the neural network models

In Chapter 4 we observed that the tanh controllers are performing better than the ReLU controllers. This is inconsistent with their size, as the ReLU controllers have more neurons per layer and are therefore more complex. Figure 6-2 visualizes the output of a ReLU and tanh network with 3 hidden layers representing signal  $u_2$ . We observe that the output of the ReLU network is not smooth, which was observed earlier in Figure 4-5. In the end, a ReLU network is a piecewise linear function, since the activation function is either activated and is linear in the input or is not activated and returns a zero. Therefore, the network output will be a non-smooth signal. Training a ReLU network translates to steering the different pieces of the network nearby the target data. Unlike the ReLU network, the tanh network returns a smooth signal as output, which explains the difference in accuracy for the network types. The non-smooth output signals might result in issues once the controller is implemented in the physical setup, where it might not be possible to change direction so abrupt.



**Figure 6-2:** Zoomed in plot of networks  $\mathcal{U}_\epsilon$  with 3 hidden layers

The increase in size of the ReLU networks to obtain comparable accuracy has its impact on the training time. For example, training a tanh network  $\mathcal{U}_2$  with 3 hidden layers representing takes 635 s, while training a ReLU network  $\mathcal{U}_2$  with 20 neurons each takes 1584 s. Furthermore, The training of the ReLU networks tend to terminate prematurely because the threshold value on the upper bound of  $\mu$  is reach, which indicates that converging the optimization longer will not result in any significant improvements. For the tanh networks trained in the experiments, this did not happen, which indicates that for this problem, the tanh activation function result in a network with higher accuracy.

### 6-1-3 The extrapolating qualities of the network controllers

Both network controllers reduce the disturbance for all setpoint signals in the validation set  $R$ , with a percentage of reduction of at least 57.65 %. Both controller managed to perform for signals that vary in their  $p_3 = \max(\ddot{r})$  value, which is unexpected, as the controllers are trained on a set of signals with with the same  $p_3$  values. All networks have as input  $\ddot{r}$ , which means that the neural networks are trained on a set of signals that have the same maximum  $\ddot{r}$  but the neural network learns for various values of  $\ddot{r}$  varying between  $p_3$  and  $-p_3$ . The fact that the network are not trained on one values

for the setpoint acceleration but rather for a range that is similar be an explanation for the controller performance for setpoint signals with different  $p_3$  value. Furthermore, the validation set  $R$  can be split up in 4 planes as is presented in section 2-1.  $R_1, R_2$  and  $R_4$  are all signals with different  $p_3$  values than the signals on which the networks are trained. The planes are relatively close to the training set, which might explain the performance on setpoint signals outside the training set. Taking signals with smaller or larger  $p_3$  values might decrease the performance of the network controllers. Finally, we observed that the general trend of the output of the tanh network  $\mathcal{U}_2$  for signals outside plane  $R_3$  was explained by the trivial model  $\mathcal{U}_2$  found with genetic programming. The trivial model scales linearly in  $\ddot{r}$ , which indicates that the compensation signals scale similar. This might give an explanation on why the controller performs so well for signals with varying  $\max(\ddot{r})$ .

## 6-2 On the complexity of the feedforward controllers

We observed that the models found with genetic programming can be used to explain the behavior of its target data. Furthermore, the memory consumption of the genetic programming was lower than that of the networks. All three controllers are stored in less than 1 Mb, which is probably a negligible bottleneck for implementation. Moreover, we saw that the model complexity reflected on the computation time, where larger models took longer to compute. We also observed an effect in the computation time that stabilized after a couple of iteration. The difference in computation time for the first iterations is caused by a variety of compiling tricks which speeds up the executing after repeating a similar computation multiple times. However, the computation times in the first iterations are slower than the sampling time and to get a conclusive result on the real-time implementation of the controllers, extra experiments on a device that is comparable with the machine on which the controller is implemented are needed.

## 6-3 On the output range analysis of the feedforward controllers

Previously, we discussed the benefits of SHERLOCK tool and applied it for the ReLU controller to obtain guaranteed output range. There are a two aspect of this tool to discuss. First of all, the guaranteed output range is based on a provided input bound as compact polyhedron. We used the maximum and minimum bounds of all input variables. However, the input variables are derivatives of each other and therefore, they are linked. Hence, choosing the input bounds as the maximum and minimum in the set is conservative, since not every combination within these bounds is possible.

Secondly, the convention demanded by the SHERLOCK tool of a network is the use of a ReLU activation function. This adds another empirical choice to the design of the network, as we have to pick a range in which we scale the target data. Scaling the range to far from zero comes with a lower gradient en slower training, while scaling the range close to zero comes with the risk of saturation by the final activation function. In Chapter 4 the target data is scaled between  $[1, 2]$ . Now if the network creates an output that is less than 0 for any input in the bounded input set provided to SHERLOCK, it is saturated by the final activation function. Although the bound on the output for the network is known and is given as 0, the accuracy of the network decreases, since information gets discarded by the saturation.

Recently, Verisig [32] was published, a tool which can be used to verify deep neural networks with tanh activation functions. By translating the entire deep neural network into a hybrid system, an output bound can be determined using existing hybrid system reachability tools such as *Flow\** [33]. The use of Verisig for the output range verification of the designed tanh feedforward controller is left as future work.

For the found genetic programming controllers, we can determine the output range by using interval arithmetic. Interval arithmetic focuses on finding bounds on rounding or measurement errors and is traditionally used to grasp the reliability of numerical methods. One way of finding the bounded output range of the feedforward controller is by means of INTLAB [34] and is considered as future work. Due to the analytic character of the genetic programming controller, we can extend on its output range analysis. For these controller it is possible to calculate their derivatives analytically. Now by calculating the output bounds for the analytic derivatives of the controller, the user can get an insight in both the bounds of the signal and its the dynamic behavior of the signal. In this way, controllers that produces signals that vary abruptly over time can be refused for implementation.

## 6-4 On the training data needed for the design methods

We observed a decrease in the accuracy of the networks trained on  $Q_4$ , which can be due to the non-uniform sampling through space  $\mathcal{P}$ . Training network controllers on data set  $Q_{GP}$  resulted in a decrease in controller performance. The network controllers trained on  $Q_{q1}$  resulted in network controllers that did not reach the design criteria. In this specific user case, genetic programming found a sufficient controller on  $Q_{q1}$ . If there is a constraint on downtime which allows for learning one signal, genetic programming can be used to design a feedforward controller.

Generally, we noticed that the tanh networks are capable of approximating the target data accurate and that the network controllers decreased in performance for setpoint signals the farthest away from the training data. Since we can create the training data by using ILC, it would be more efficient to sample the training signals uniformly throughout parameterized space  $\mathcal{P}$ .

## 6-5 On another approach to design the feedforward controller

In this thesis we took a data driven approach as we have the possibility of learning optimal compensation signals by means of ILC. Another approach that might be interesting is directly learning the feedforward controller that produces the compensation signals. Genetic programming has been used to directly generate control laws [35] and in this case would directly solve the following Problem:

Design feedforward controller  $\mathcal{U}^* : \mathbb{R}^{N \times n} \rightarrow \mathbb{R}^{N \times 3}$  such that:

$$\forall \mathbf{r} \in R, \mathcal{U}^* = \arg \min_{\mathcal{U} \in \mathcal{S}} \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_r))\|_2. \quad (6-1)$$

By using  $\epsilon(\mathbf{r}) = \|g(\mathbf{r}, \mathcal{U}(\mathcal{R}_r))\|_2$  as fitness function for the models in genetic programming, the algorithm can be adapted to directly search for controller actions. In this case, we have to tackle the dimensionality of  $\mathcal{U}$ . In the data-driven approach we could decouple each signal  $\mathbf{u}_i \in \mathcal{U}$ . This is not longer possible if no data is known a priori, as all three signals have impact on  $\epsilon$ .



## Conclusion

We will finalize this chapter by summarizing the comparison that is made in this thesis. The comparison of the feedforward controller is summarized Table 7-1, where we rate the controllers on a scale from  $[- - -, \dots, + + +]$ , where  $- - -$  is the absolute worst score and  $+ + +$  is the absolute best score.

- *Disturbance reduction*

In this thesis we successfully designed 3 feedforward controller that satisfied the performance criterion provided in Problem 1-2.1. The tanh controller performs the best, followed by the *ReLU* controller. The genetic programming controller performs the worst of the three, but still manages to reduce the disturbance measure with at least 40.01% for all setpoint signals in the validation set  $R$ .

- *Complexity*

The designed controllers vary in their complexity. The *ReLU* controller is the largest, followed by the tanh controller. The genetic programming controller was composed of the least complex models. This resulted in insights in the design problem, less memory consumption by the controller and faster average computation times.

- *Output bound verification*

SHERLOCK was used to find a guaranteed output bound for the ReLU network, which lead to a maximum output bound of 4.8320 times higher then measured. There are methods that compute a guaranteed output bound for the tanh controller and the genetic programming controller. The analytic nature of the genetic programming controller allows for guaranteed output bounds on the derivative of its output signal.

- *Training data set*

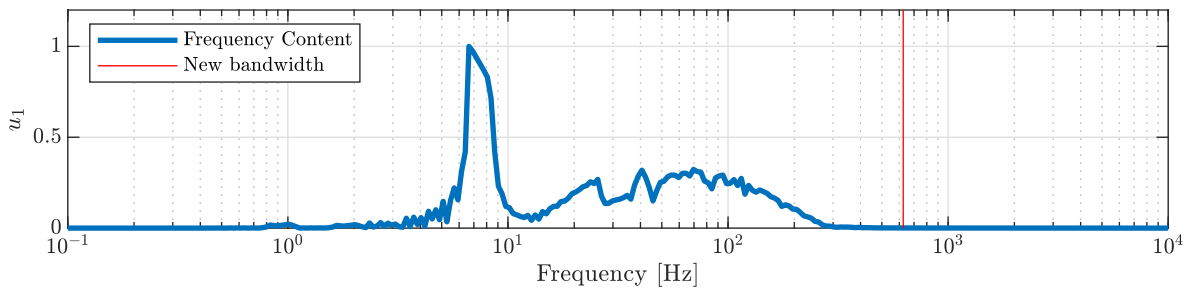
The performance of the tanh and ReLU controller is affected by a decrease in training data. We observed that the performance of the controller found with genetic programming are barely affected by the size of the dataset. Genetic programming might be the solution in a scenario where the downtime of the machine is more critical than the attenuation of the disturbance force.

<b>Quality</b>	<b>Genetic Programming</b>	<b>tanh networks</b>	<b>ReLU networks</b>
<i>Controller Performance</i>	+	+++	++
<i>Complexity</i>	+++	--	---
<i>Output bound verification</i>	++	+	+
<i>Size of training data set</i>	+	-	-

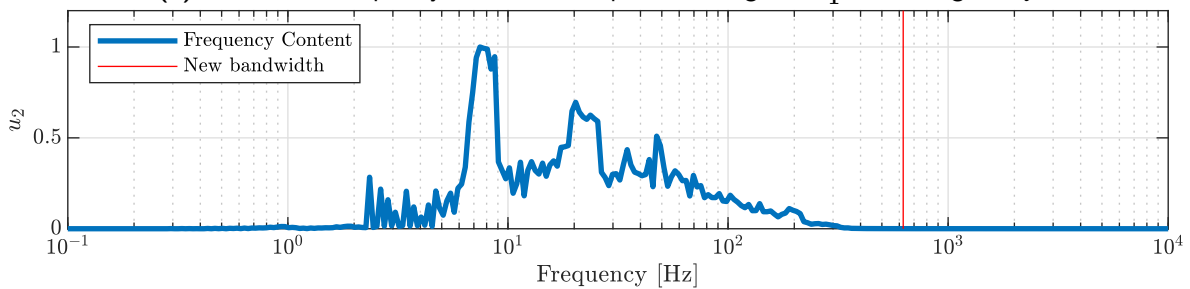
**Table 7-1:** An overview of the comparison for the three controller types on the four qualities



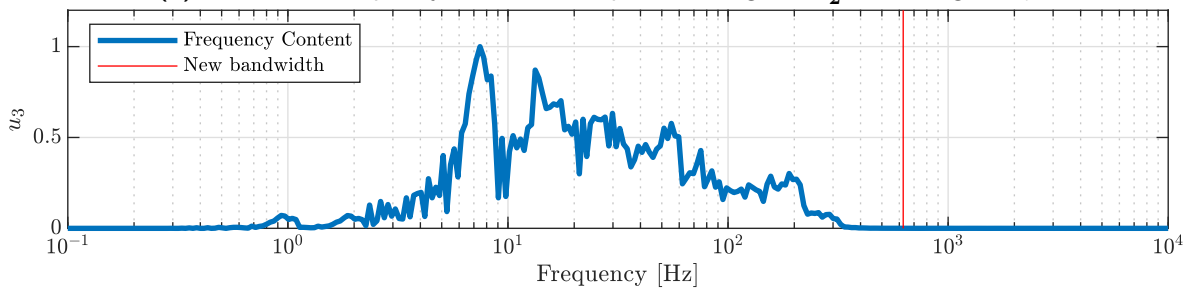
## Appendix Genetic Programming



(a) Normalized frequency domain envelope for the signals  $u_1^*$  in training set  $Q$

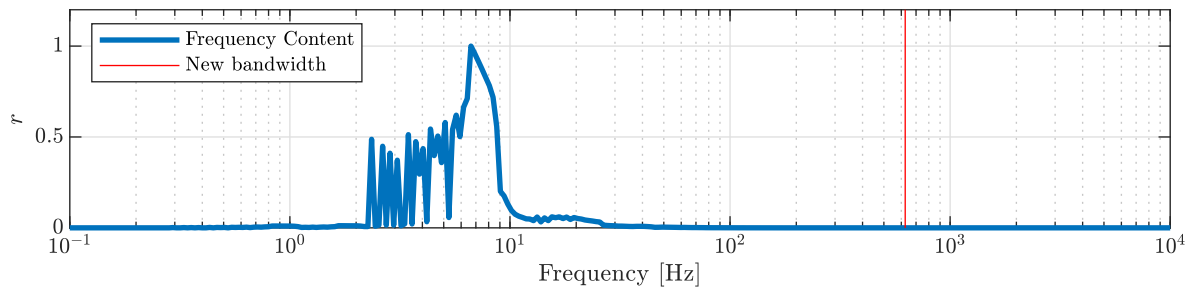
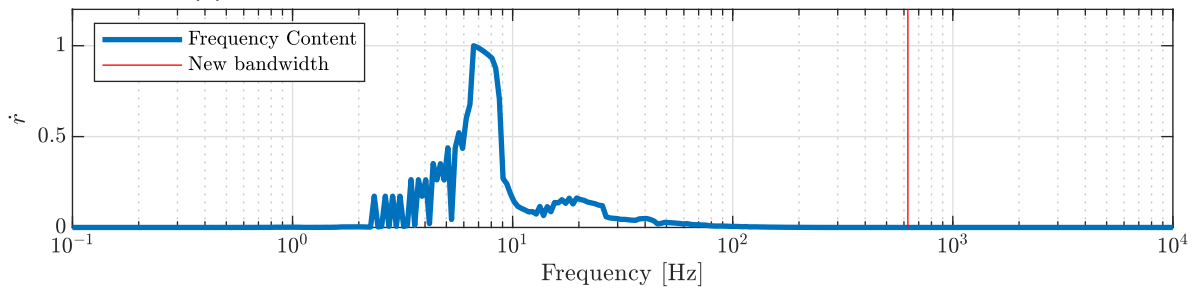
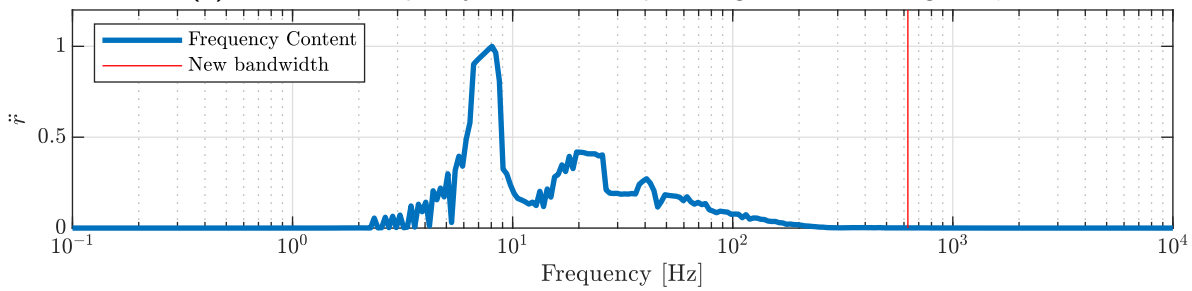
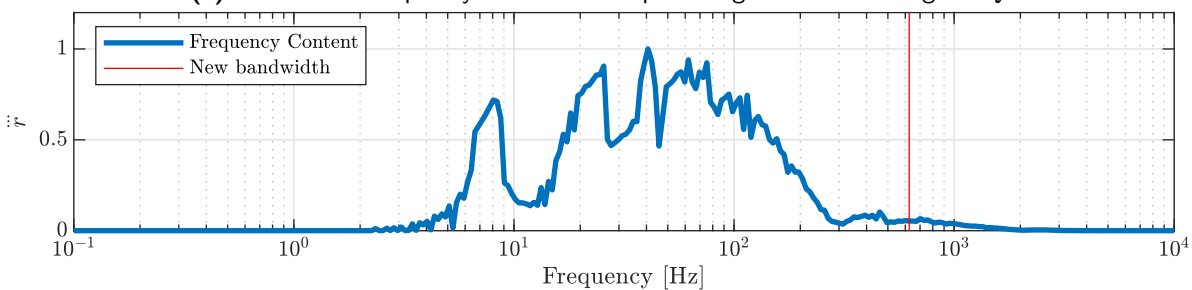


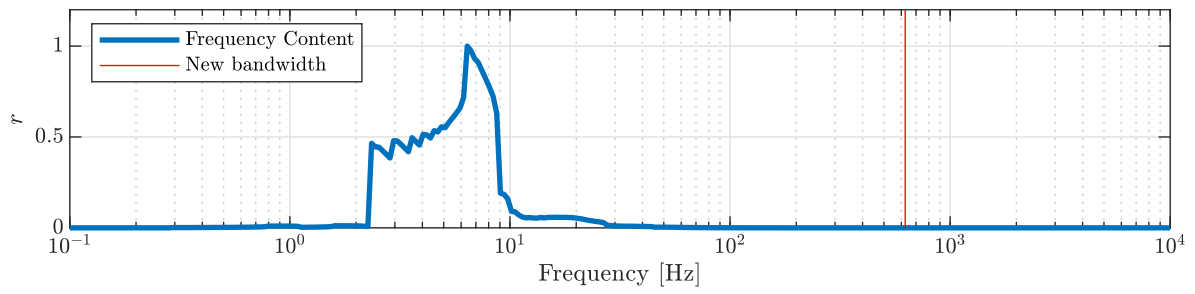
(b) Normalized frequency domain envelope for the signals  $u_2^*$  in training set  $Q$



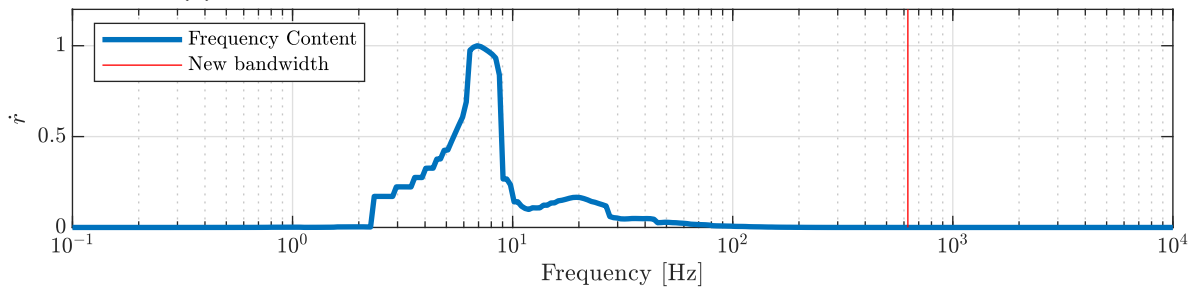
(c) Normalized frequency domain envelope for the signals  $u_3^*$  in training set  $Q$

**Figure A-1:** Normalized frequency domain envelope for the compensation signals  $U$  in training set  $Q$

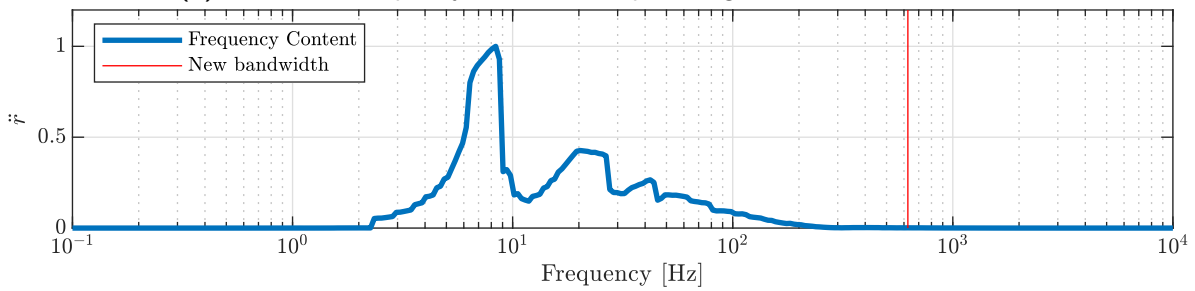
(a) Normalized frequency domain envelope for signals  $r$  in training set  $Q$ (b) Normalized frequency domain envelope for signals  $\dot{r}$  in training set  $Q$ (c) Normalized frequency domain envelope for signals  $\ddot{r}$  in training set  $Q$ (d) Normalized frequency domain envelope for signals  $\dddot{r}$  in training set  $Q$ **Figure A-2:** Normalized frequency domain envelope for the input signals in training set  $Q$



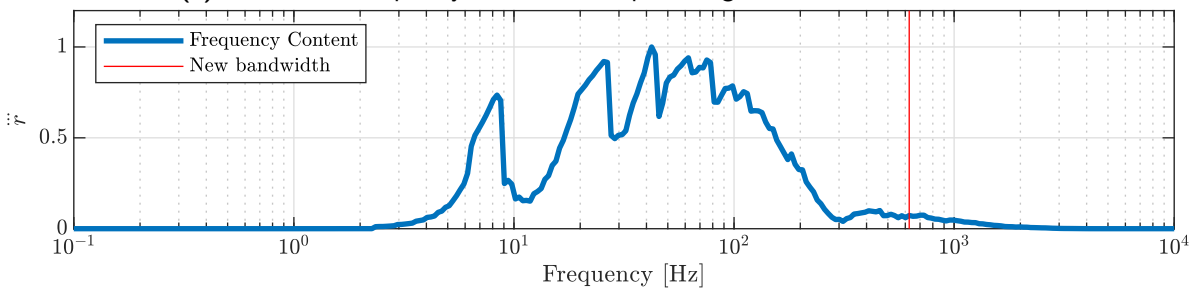
(a) Normalized frequency domain envelope for signals  $r$  in validation set  $R$



(b) Normalized frequency domain envelope for signals  $\dot{r}$  in validation set  $R$

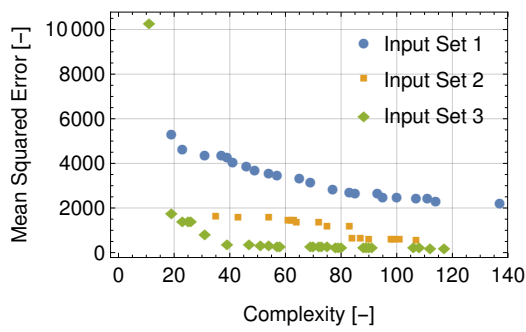
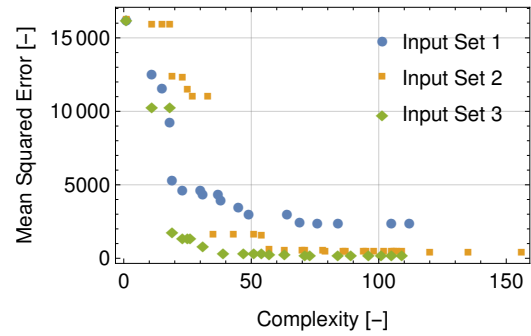
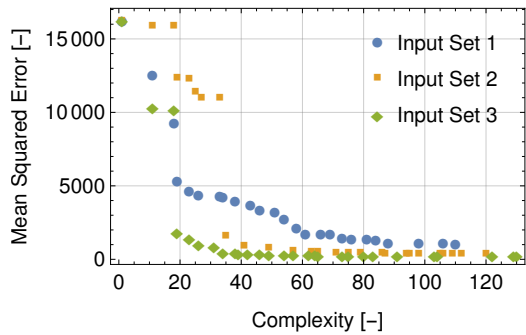
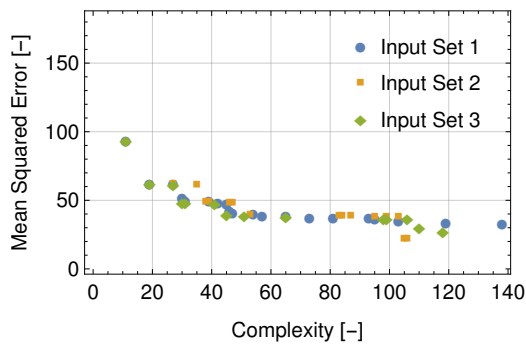
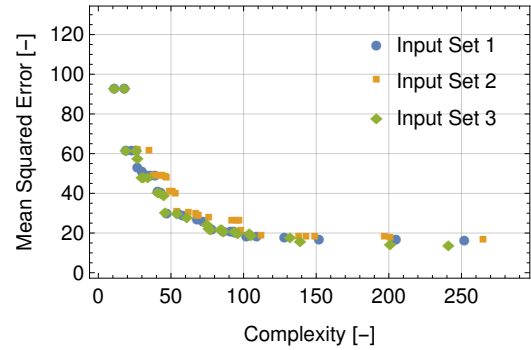
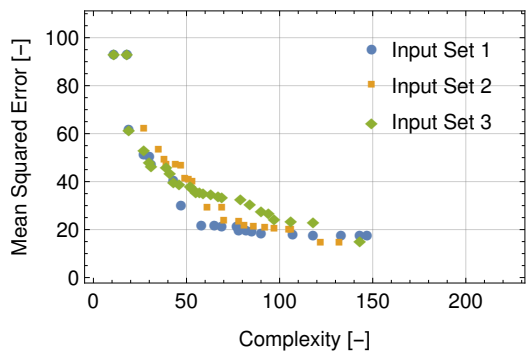


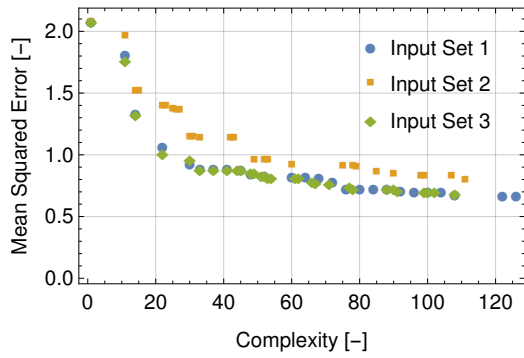
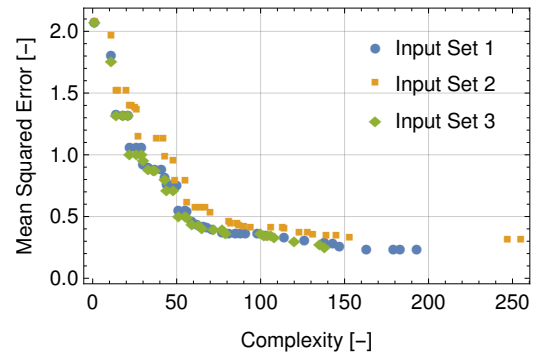
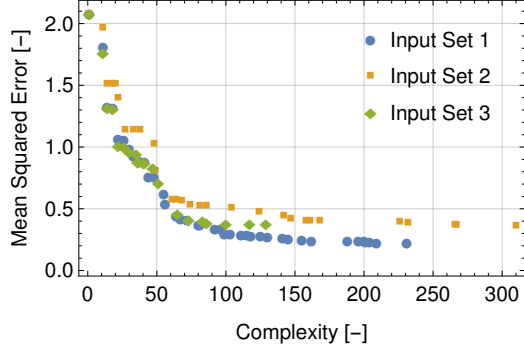
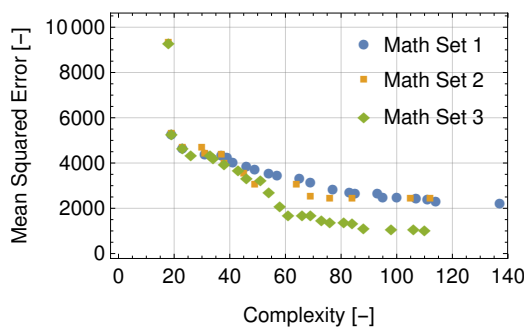
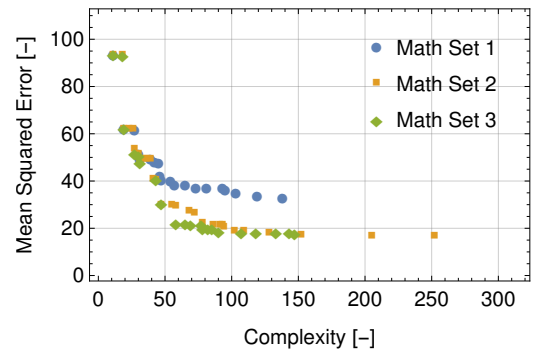
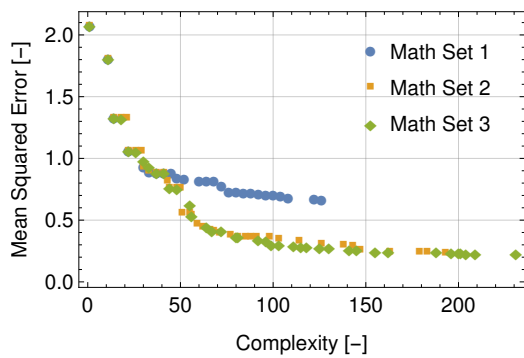
(c) Normalized frequency domain envelope for signals  $\ddot{r}$  in validation set  $R$

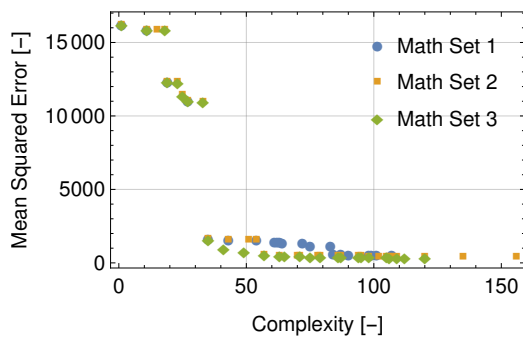
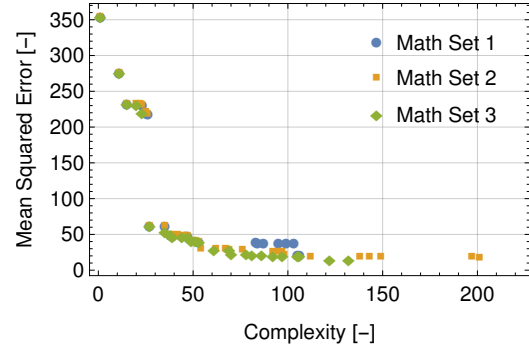
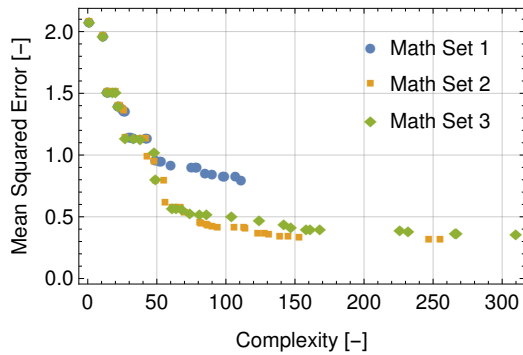
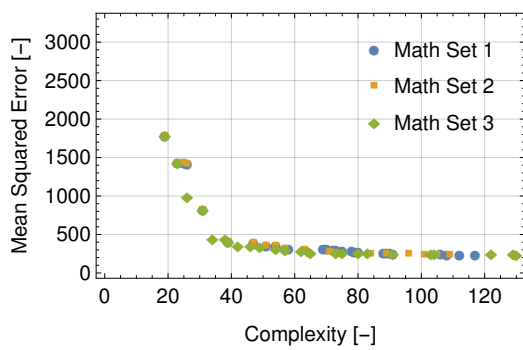
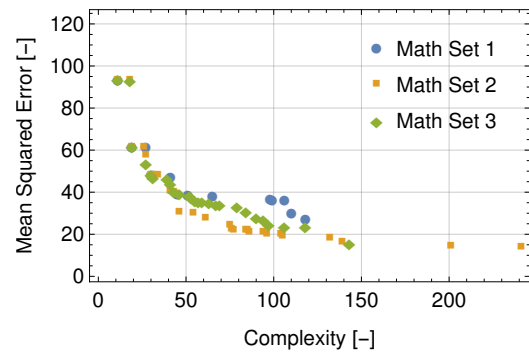
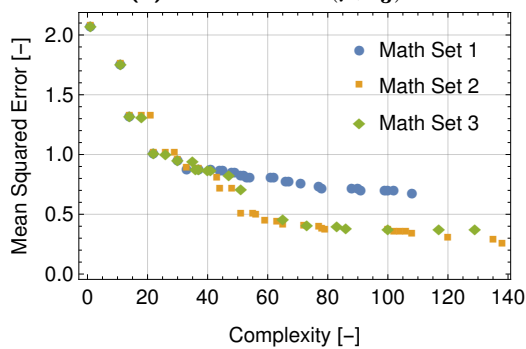


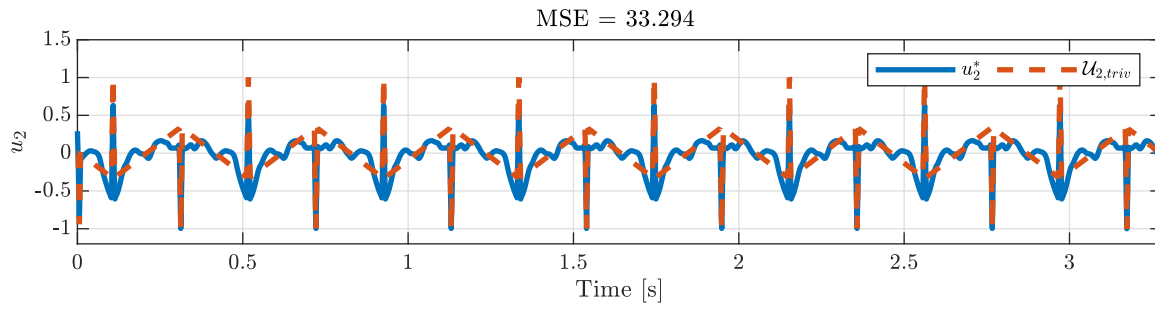
(d) Normalized frequency domain envelope for signals  $\dddot{r}$  in validation set  $R$

**Figure A-3:** Normalized frequency domain envelope for the input signals in validation set  $R$

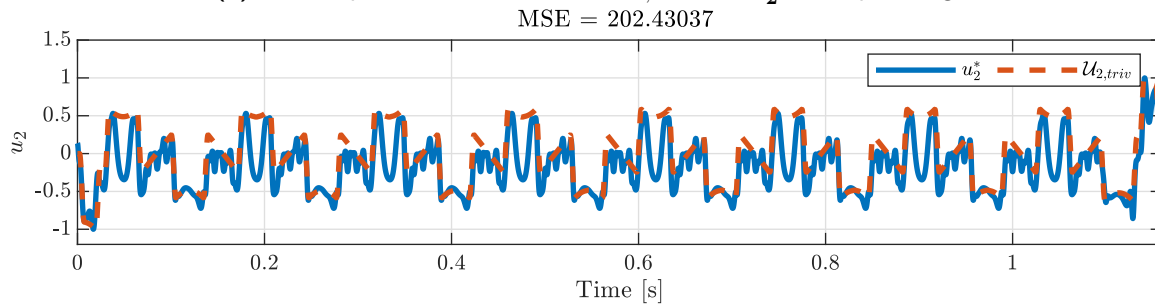
(a) Pareto fronts for models  $\mathcal{U}_1$  build with  $\mathcal{M}_1$ (b) Pareto fronts for models  $\mathcal{U}_1$  build with  $\mathcal{M}_2$ (c) Pareto fronts for models  $\mathcal{U}_1$  build with  $\mathcal{M}_3$ **Figure A-4:** The influence of the input sets on the models  $\mathcal{U}_1$ (a) Pareto fronts for models  $\mathcal{U}_2$  build with  $\mathcal{M}_1$ (b) Pareto fronts for models  $\mathcal{U}_2$  build with  $\mathcal{M}_2$ (c) Pareto fronts for models  $\mathcal{U}_2$  build with  $\mathcal{M}_3$ **Figure A-5:** The influence of the input sets on the models  $\mathcal{U}_2$

(a) Pareto fronts for models  $\mathcal{U}_3$  build with  $\mathcal{M}_1$ (b) Pareto fronts for models  $\mathcal{U}_3$  build with  $\mathcal{M}_2$ (c) Pareto fronts for models  $\mathcal{U}_3$  build with  $\mathcal{M}_3$ **Figure A-6:** The influence of the input sets on the models  $\mathcal{U}_3$ (a) All models  $\mathcal{U}_1(\rho_{r11})$ (b) All models  $\mathcal{U}_2(\rho_{r21})$ (c) All models  $\mathcal{U}_3(\rho_{r31})$ **Figure A-7:** All models found with genetic programming dependent on  $\mathcal{R}_{r1}$

(a) All models  $\mathcal{U}_1(\rho_{r12})$ (b) All models  $\mathcal{U}_2(\rho_{r22})$ (c) All models  $\mathcal{U}_3(\rho_{r32})$ **Figure A-8:** All models found with genetic programming dependent on  $\mathcal{R}_{r2}$ (a) All models  $\mathcal{U}_1(\rho_{r13})$ (b) All models  $\mathcal{U}_2(\rho_{r23})$ (c) All models  $\mathcal{U}_3(\rho_{r33})$ **Figure A-9:** All models found with genetic programming dependent on  $\mathcal{R}_{r3}$

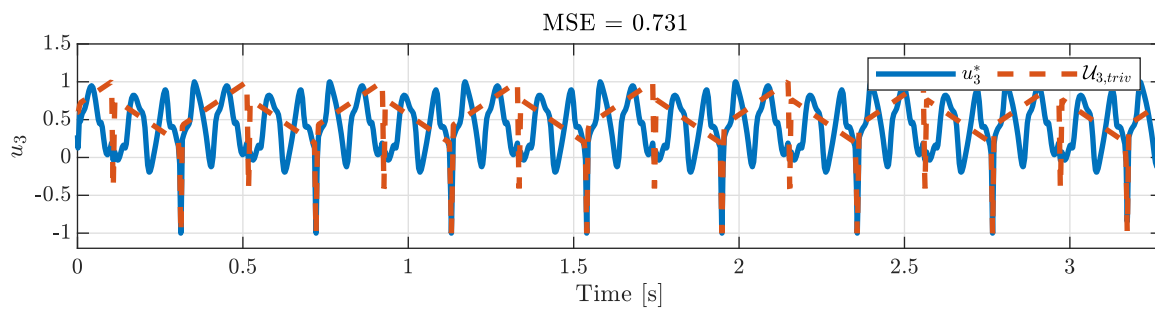


(a) The output of the trivial model  $U_{2,triv}$  and  $u_2^*$  for setpoint signal 1

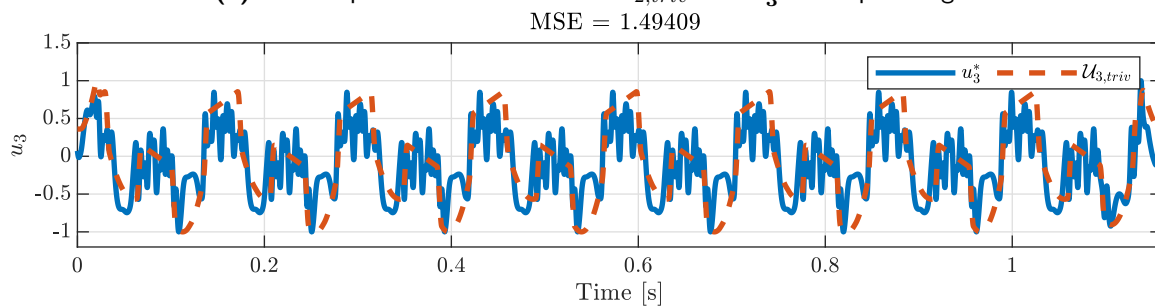


(b) The output of the trivial models  $U_{2,triv}$  and  $u_2^*$  for setpoint signal 2

**Figure A-10:** Comparing the output of the trivial model with the target data  $u_2^*$



(a) The output of the trivial model  $U_{3,triv}$  and  $u_3^*$  for setpoint signal 1



(b) The output of the trivial model  $U_{3,triv}$  and  $u_3^*$  for setpoint signal 2

**Figure A-11:** Comparing the output of the trivial model with the target data  $u_3^*$

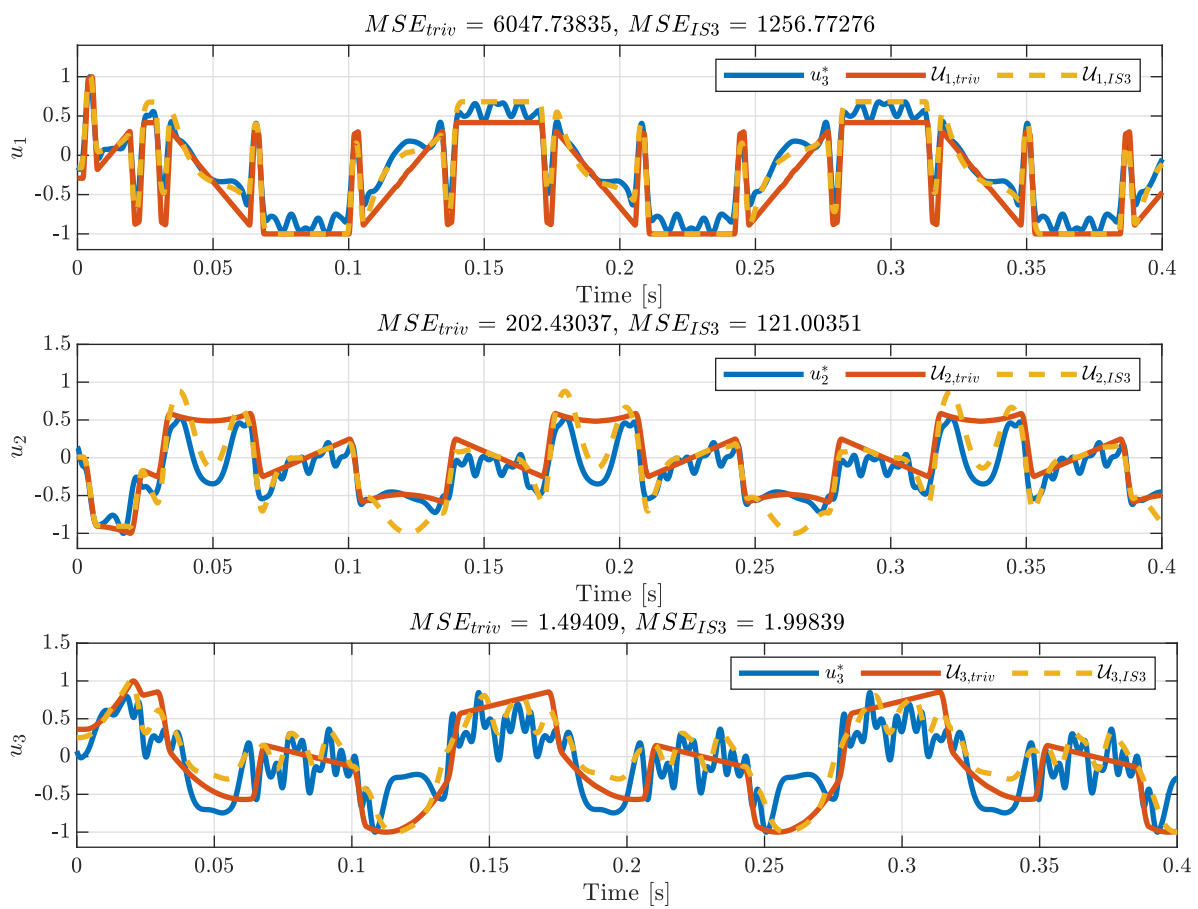
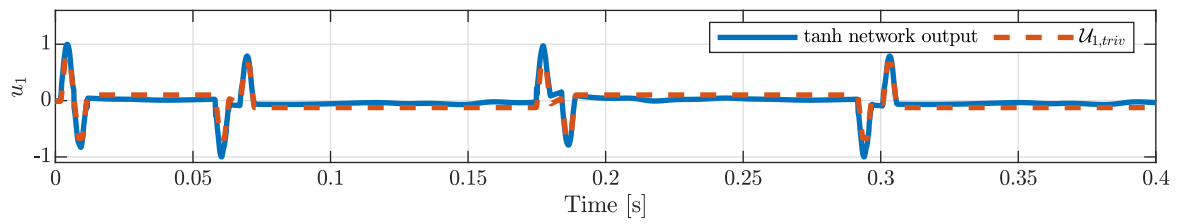


Figure A-12: The output of the trivial models  $U_{i,triv}$  and  $U_{i,IS3}$

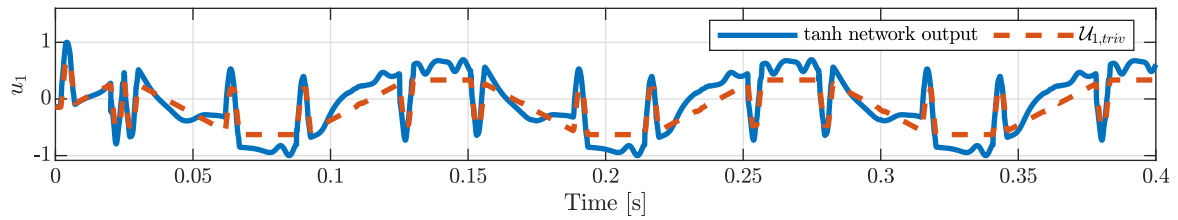


Complexity	MSE	Vars	Function
1	16209.4		4.83
11	12544.2	$\ddot{r}$	$0.0025\ddot{r} + 4.22$
15	11587.9	$\dot{r}$	$1.23\dot{r}^3 + 4.89$
18	9304.5	$\dot{r}, \ddot{r}$	$33.89 \cos(\ddot{r})\dot{r} + 7.55$
19	5310.92	$\dot{r}, \ddot{r}$	$0.0034\ddot{r} + 33.53\dot{r} + 2.69$
23	4674.61	$\dot{r}, \ddot{r}$	$1.86\dot{r}^3 + 0.0034\ddot{r} + 4.07$
26	4355.49	$\dot{r}, \ddot{r}$	$0.0034\ddot{r} + 8.78 \dot{r}  + 3.48$
33	4352.84	$r, \dot{r}, \ddot{r}$	$0.0034\ddot{r} + 8.79 \dot{r}  \cos(r)\dot{r} + 3.47$
34	4239.07	$r, \dot{r}, \ddot{r}$	$0.0034\ddot{r} + 8.77 \dot{r}  - 187.24r + 3.86$
38	3975.15	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$1.31\dot{r}^3 + 15.12 \cos(\ddot{\ddot{r}})\dot{r} + 0.0031\ddot{\ddot{r}} + 5.34$
43	3719.61	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$1.91\dot{r}^3 + 0.0037\ddot{\ddot{r}} - 0.55\ddot{\ddot{r}} + 152.32 \tanh(\ddot{\ddot{r}}) + 4.30$
45	3503.01	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$1.\dot{r}^3 + 152.72 \sin(0.019\ddot{\ddot{r}}) + 0.0039\ddot{\ddot{r}} + 4.63$
46	3334.55	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$0.0038\ddot{\ddot{r}} - 0.57\ddot{\ddot{r}} + 9.06 \dot{r}  + 158.65 \tanh(\ddot{\ddot{r}}) + 3.68$
54	2728.04	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$295.00\sigma(\ddot{\ddot{r}}) + 0.0047\ddot{\ddot{r}} - 0.89\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.08e^{-5}\ddot{\ddot{r}} + 35.97\dot{r} + 2.79$
58	2128.93	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$1.96\dot{r}^3 + 283.06\sigma(\ddot{\ddot{r}}) + 0.0046\ddot{\ddot{r}} - 0.85\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.08e^{-5}\ddot{\ddot{r}} + 4.26$
61	1736.41	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$290.29\sigma(\ddot{\ddot{r}}) + 0.0047\ddot{\ddot{r}} - 0.88\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.08e^{-5}\ddot{\ddot{r}} + 9.29 \dot{r}  + 3.63$
66	1724.06	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$558.94\sigma(\sigma(\ddot{\ddot{r}})) + 0.0047\ddot{\ddot{r}} - 0.85\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.06e^{-5}\ddot{\ddot{r}} + 9.30 \dot{r}  + 3.67$
69	1702.53	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$279.77\sigma(\ddot{\ddot{r}}) + 0.0047\ddot{\ddot{r}} - 0.90\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.05e^{-5}\ddot{\ddot{r}} + 9.32 \dot{r}  + 4.09$
73	1485.01	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$280.29\sigma(\ddot{\ddot{r}}) + 0.0048 \sigma(\ddot{\ddot{r}}) \ddot{\ddot{r}} - 0.85\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.09e^{-5}\ddot{\ddot{r}} + 9.17 \dot{r}  + 3.\dot{r}$
76	1398.86	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.32 \dot{r}  + 2\dot{r}.55\sigma(\ddot{\ddot{r}}) + 0.0047\ddot{\ddot{r}} - 0.80\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.00e^{-5}\ddot{\ddot{r}} + 3.85$
81	1388.65	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.33 \dot{r}  + 564.89\sigma(\sigma(\ddot{\ddot{r}})) + 0.0047\ddot{\ddot{r}} - 0.77\ddot{\ddot{r}} -  \ddot{\ddot{r}}  9.87e^{-6}\ddot{\ddot{r}} + 3.89$
84	1355.68	$r, \dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.35 \dot{r}  + 283.55\sigma(\ddot{\ddot{r}}) + 0.0047\ddot{\ddot{r}} - 0.83\ddot{\ddot{r}} -  \ddot{\ddot{r}}  9.76e^{-6}\ddot{\ddot{r}} - 215.16r + 4.28$
88	1155.29	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.20 \dot{r}  + 283.46\sigma(\ddot{\ddot{r}}) + 0.0048 \sigma(\ddot{\ddot{r}}) \ddot{\ddot{r}} - 0.77\ddot{\ddot{r}} -  \ddot{\ddot{r}}  1.02e^{-5}\ddot{\ddot{r}} + 4.14$
98	1089.01	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.17 \dot{r}  + 245.58\sigma(\ddot{\ddot{r}}) + 0.0049 \sigma(\ddot{\ddot{r}}) \ddot{\ddot{r}} - 0.64\ddot{\ddot{r}} -  \ddot{\ddot{r}}  4.01e^{-8}\ddot{\ddot{r}} + 4.06$
106	1089.01	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.20 \dot{r}  + 0.12\dot{r} + 245.55\sigma(\ddot{\ddot{r}}) + 0.0049 \sigma(\ddot{\ddot{r}}) \ddot{\ddot{r}} - 0.64\ddot{\ddot{r}} -  \ddot{\ddot{r}}  4.01e^{-8}\ddot{\ddot{r}} + 4.06$
110	1056.49	$\dot{r}, \ddot{r}, \ddot{\ddot{r}}$	$-0.018\ddot{\ddot{r}}^2 + 9.29 \dot{r}  + 29.90\sigma(\ddot{\ddot{r}}) + 238.50\sigma(\ddot{\ddot{r}}) + 0.0045 \sigma(\ddot{\ddot{r}}) \ddot{\ddot{r}} - 0.62\ddot{\ddot{r}} -  \ddot{\ddot{r}}  3.94e^{-8}\ddot{\ddot{r}} + 4.14$

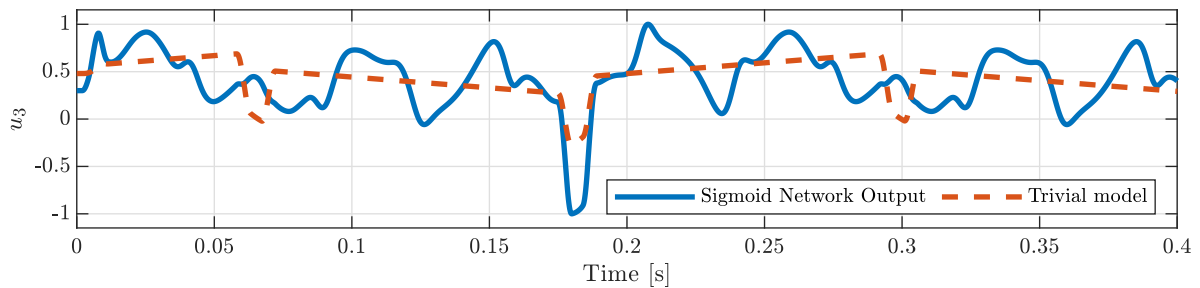
**Table A-1:** The Pareto front of the models  $\mathcal{U}_1$  depending on  $\mathcal{R}_{r1}$



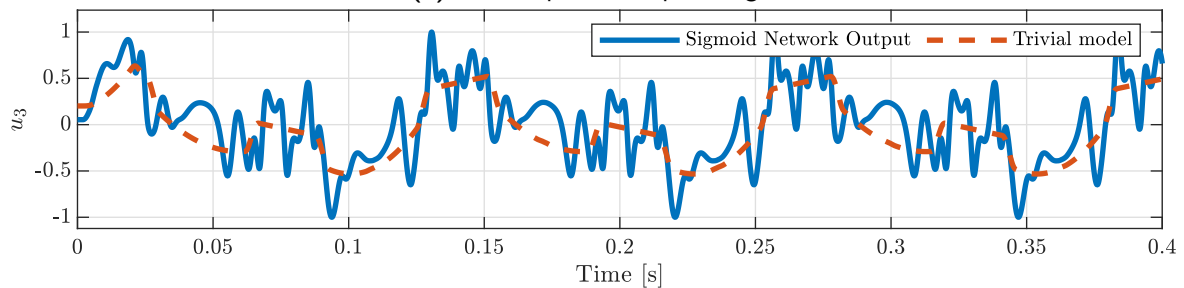
(a) The output for setpoint signal 1



(b) The output for setpoint signal 2

**Figure A-13:** Comparing the compensation signals for  $u_2$  for setpoint signal 1 & 2

(a) The output for setpoint signal 1



(b) The output for setpoint signal 2

**Figure A-14:** Comparing the compensation signals for  $u_2$  for setpoint signal 1 & 2

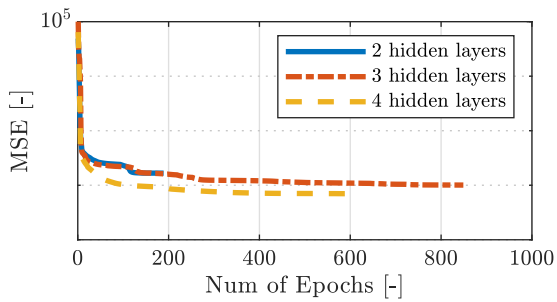
---

## Appendix B

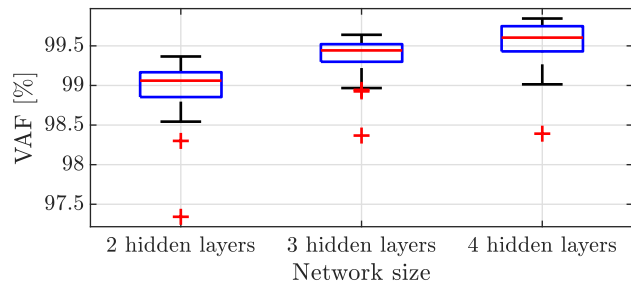
---

# Neural Networks

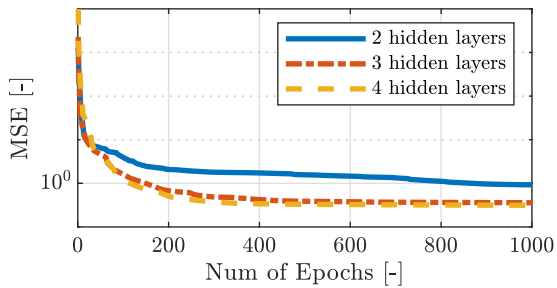
## B-1 The influence of the hidden layers on the accuracy of the tanh networks



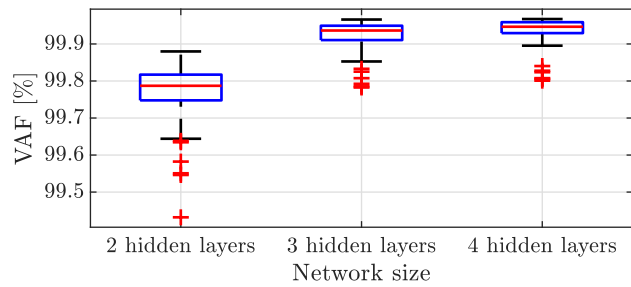
(a) Convergence process for the tanh networks  $\mathcal{U}_1$  trained on  $Q_1$



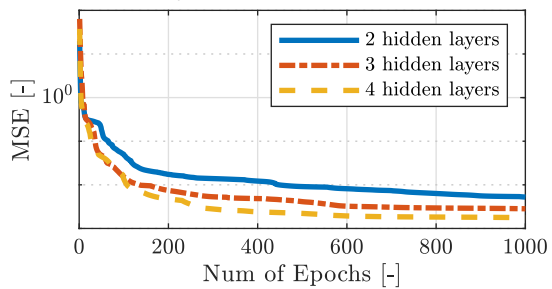
(b)  $\mathcal{V}_{Q,1}$  for the tanh networks  $\mathcal{U}_1$  trained on  $Q_1$



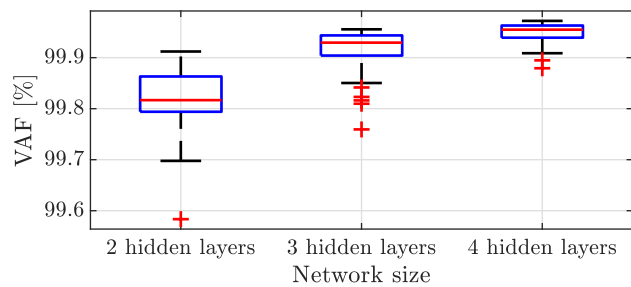
(c) Convergence process for the tanh networks  $\mathcal{U}_2$  trained on  $Q_1$



(d)  $\mathcal{V}_{Q,2}$  for the tanh networks  $\mathcal{U}_2$  trained on  $Q_1$

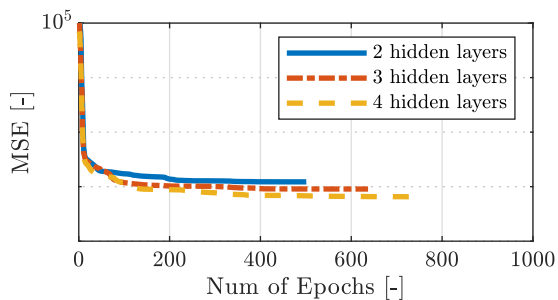


(e) Convergence process for the tanh networks  $\mathcal{U}_3$  trained on  $Q_1$

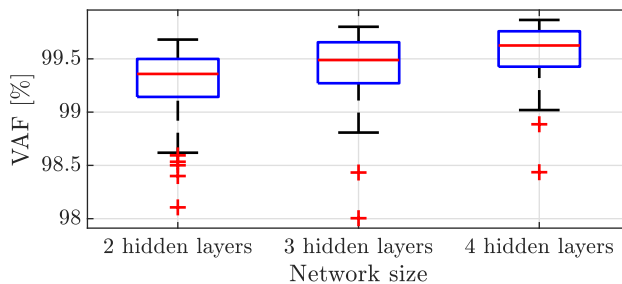


(f)  $\mathcal{V}_{Q,3}$  for the tanh networks  $\mathcal{U}_3$  trained on  $Q_1$

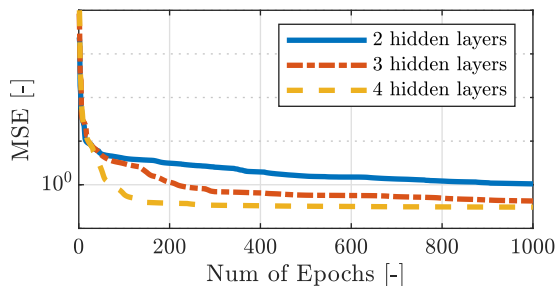
**Figure B-1:** The influence of the hidden layers on the accuracy of the tanh neural networks trained on data set  $Q_1$



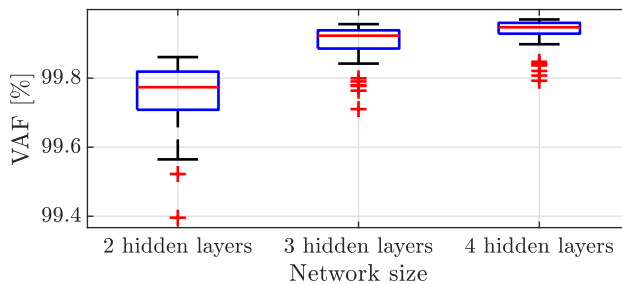
(a) Convergence process for the tanh networks  $\mathcal{U}_1$  trained on  $Q_2$



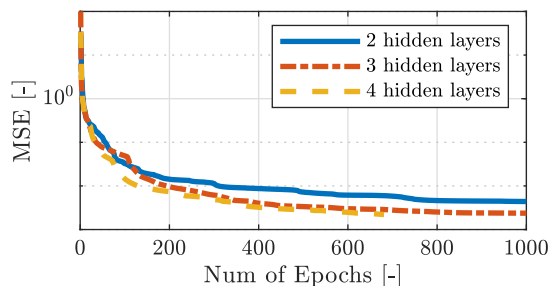
(b)  $\mathcal{V}_{Q,1}$  for the tanh networks  $\mathcal{U}_1$  trained on  $Q_2$



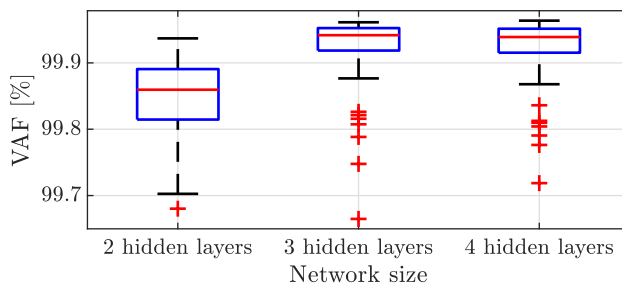
(c) Convergence process for the tanh networks  $\mathcal{U}_2$  trained on  $Q_2$



(d)  $\mathcal{V}_{Q,2}$  for the tanh networks  $\mathcal{U}_2$  trained on  $Q_2$

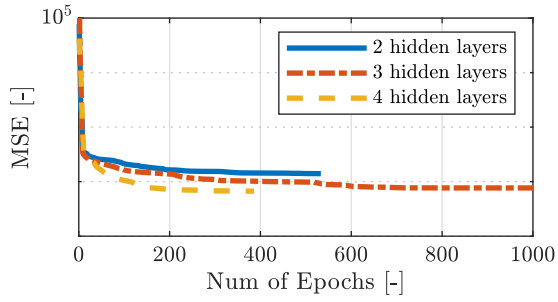


(e) Convergence process for the tanh networks  $\mathcal{U}_3$  trained on  $Q_2$

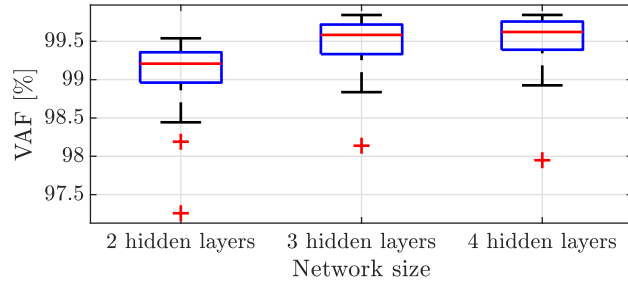


(f)  $\mathcal{V}_{Q,3}$  for the tanh networks  $\mathcal{U}_3$  trained on  $Q_2$

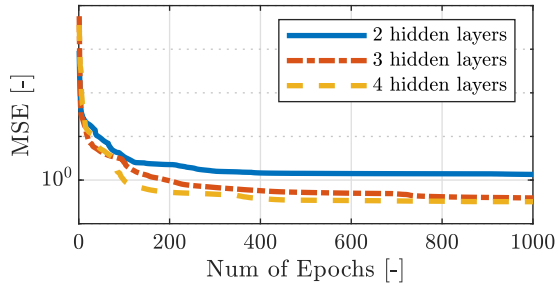
**Figure B-2:** The influence of the hidden layers on the accuracy of the tanh neural networks trained on data set  $Q_2$



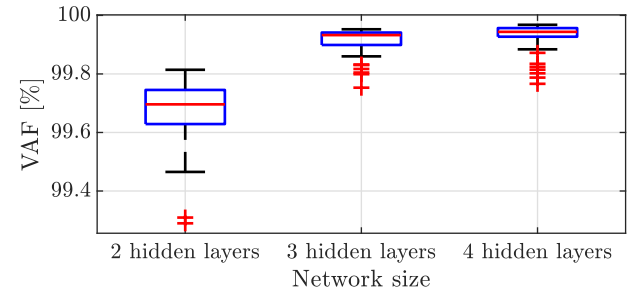
(a) Convergence process for the tanh networks  $\mathcal{U}_1$  trained on  $Q_3$



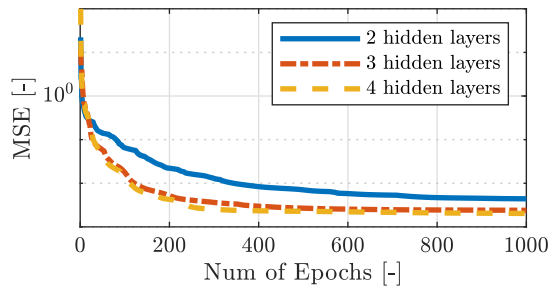
(b)  $\mathcal{V}_{Q,1}$  for the tanh networks  $\mathcal{U}_1$  trained on  $Q_3$



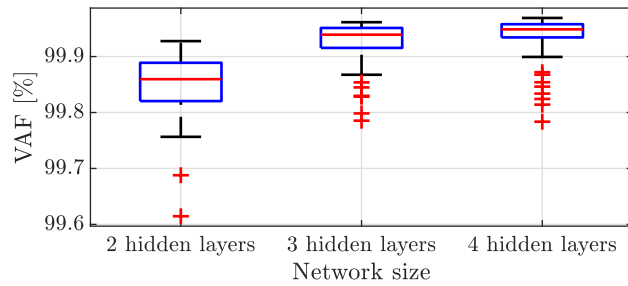
(c) Convergence process for the tanh networks  $\mathcal{U}_2$  trained on  $Q_3$



(d)  $\mathcal{V}_{Q,2}$  for the tanh networks  $\mathcal{U}_2$  trained on  $Q_3$

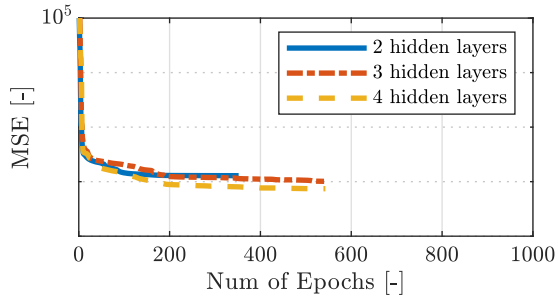


(e) Convergence process for the tanh networks  $\mathcal{U}_3$  trained on  $Q_3$

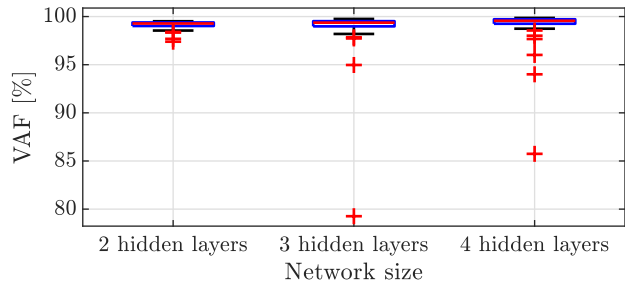


(f)  $\mathcal{V}_{Q,3}$  for the tanh networks  $\mathcal{U}_3$  trained on  $Q_3$

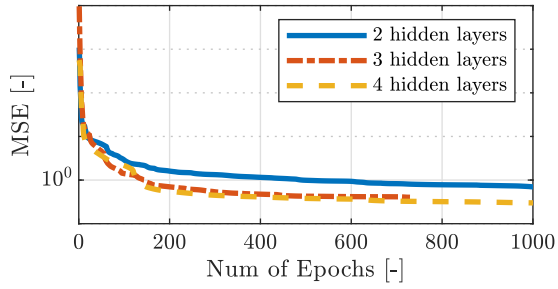
**Figure B-3:** The influence of the hidden layers on the accuracy of the tanh neural networks trained on data set  $Q_3$



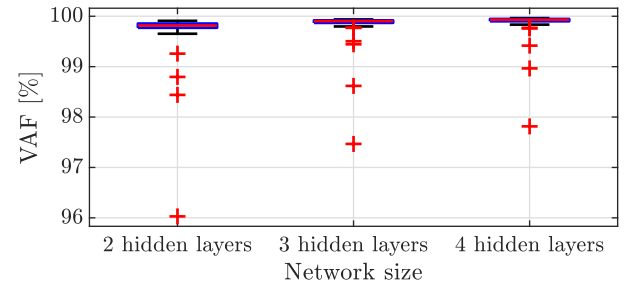
(a) Convergence process for the tanh networks  $\mathcal{U}_1$  trained on  $Q_4$



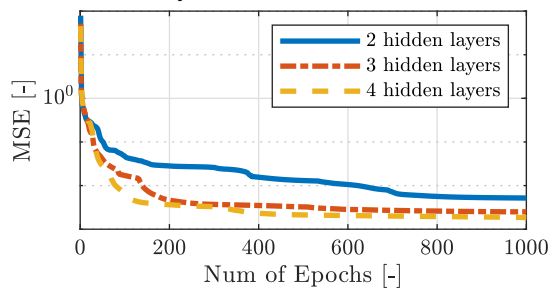
(b)  $\mathcal{V}_{Q,1}$  for the tanh networks  $\mathcal{U}_1$  trained on  $Q_4$



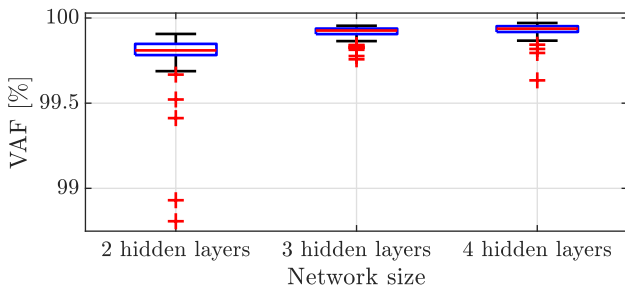
(c) Convergence process for the tanh networks  $\mathcal{U}_2$  trained on  $Q_4$



(d)  $\mathcal{V}_{Q,2}$  for the tanh networks  $\mathcal{U}_2$  trained on  $Q_4$



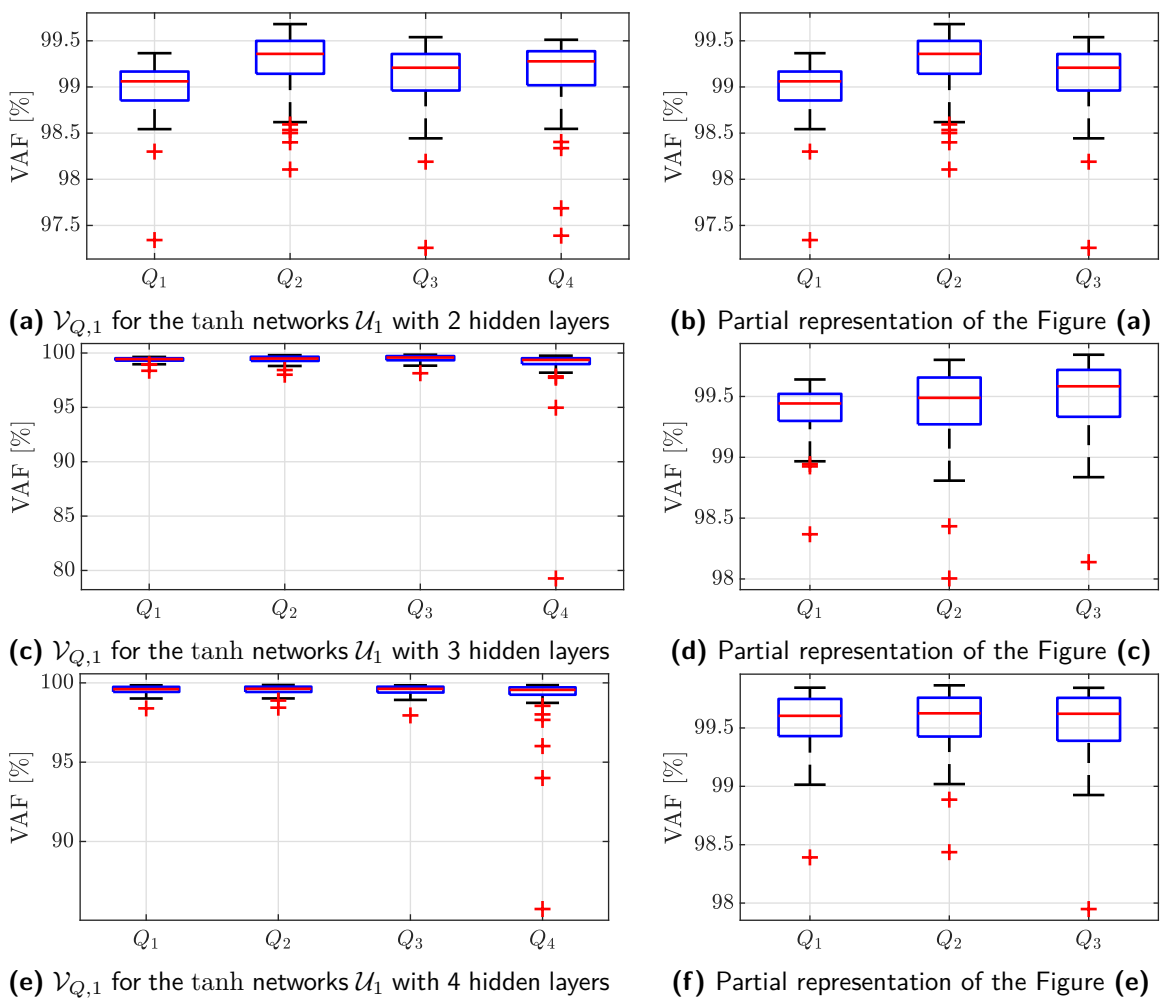
(e) Convergence process for the tanh networks  $\mathcal{U}_3$  trained on  $Q_4$



(f)  $\mathcal{V}_{Q,3}$  for the tanh networks  $\mathcal{U}_3$  trained on  $Q_4$

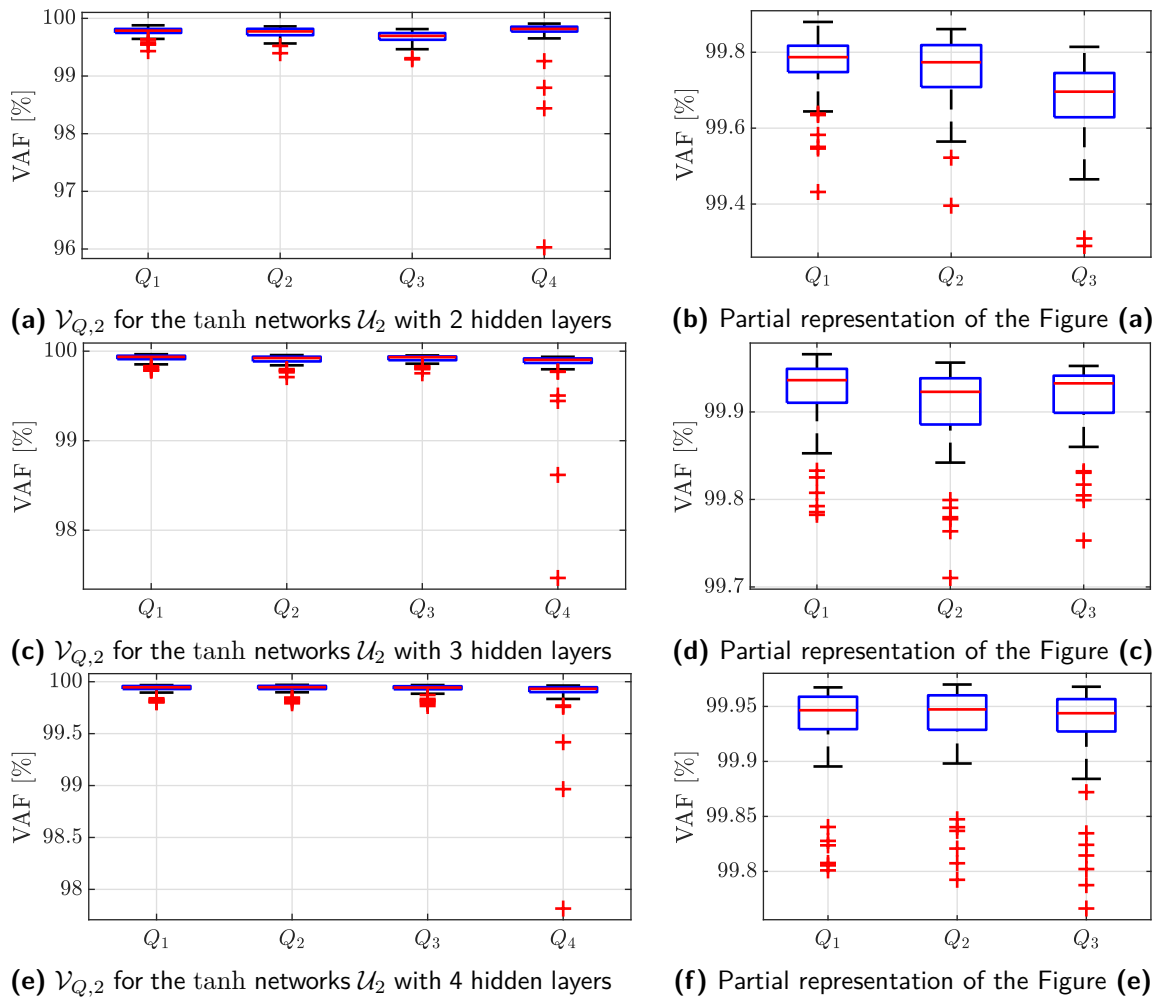
**Figure B-4:** The influence of the hidden layers on the accuracy of the tanh neural networks trained on data set  $Q_4$

## B-2 The influence of the training data set on the accuracy of the tanh networks

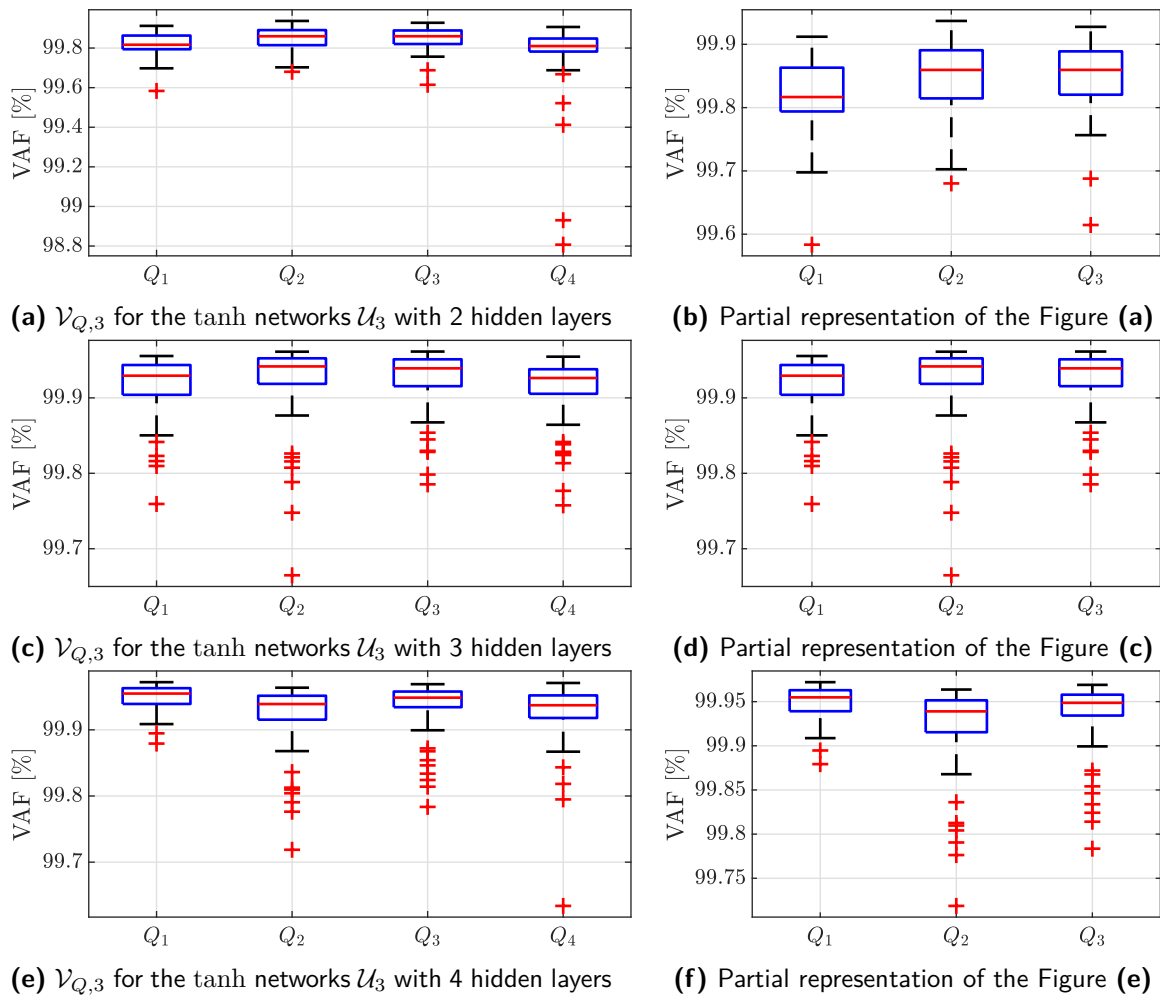


**Figure B-5:** The influence of the training data sets on the accuracy of the tanh networks  $\mathcal{U}_1$



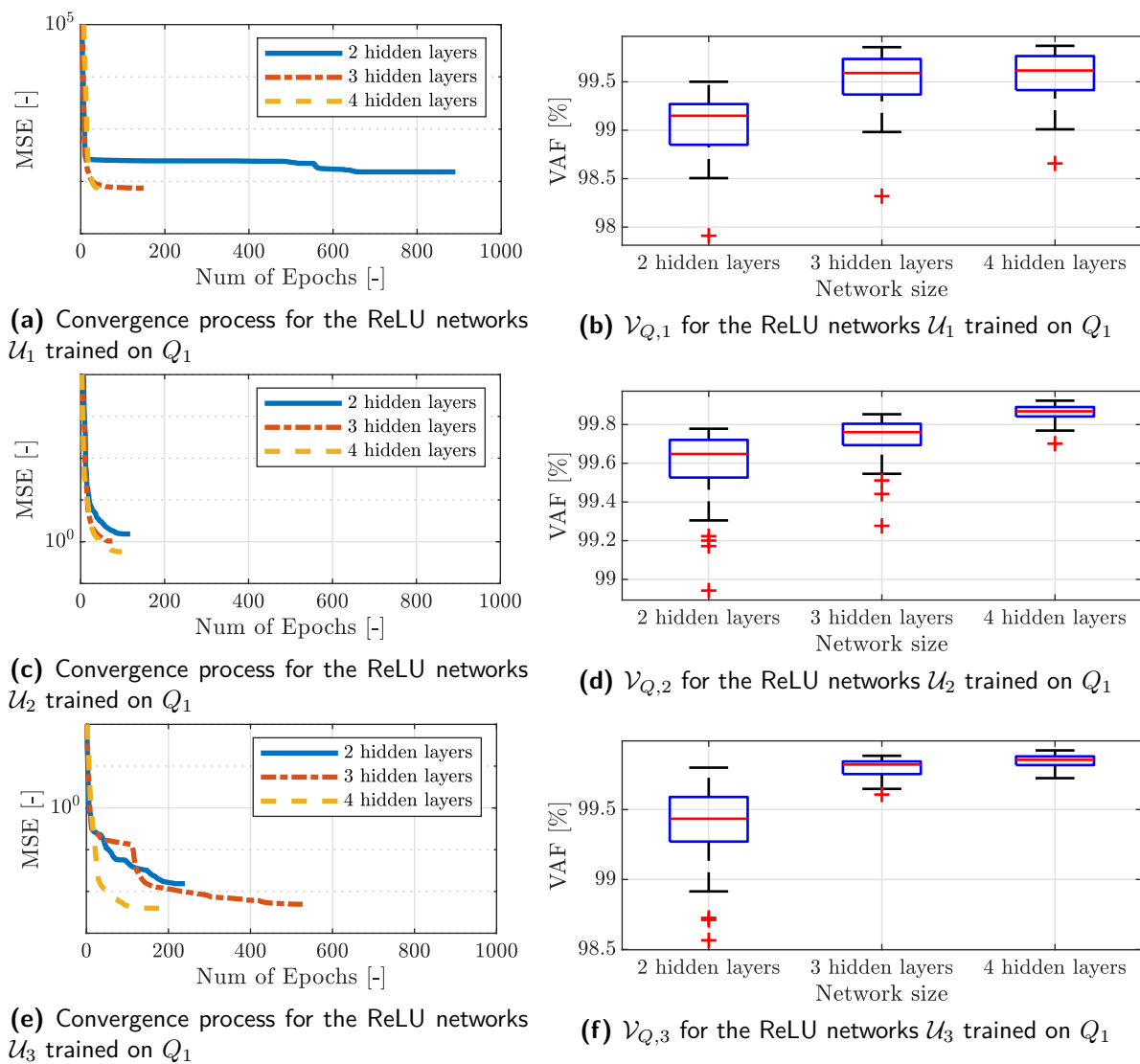


**Figure B-6:** The influence of the training data sets on the accuracy of the tanh networks  $\mathcal{U}_2$

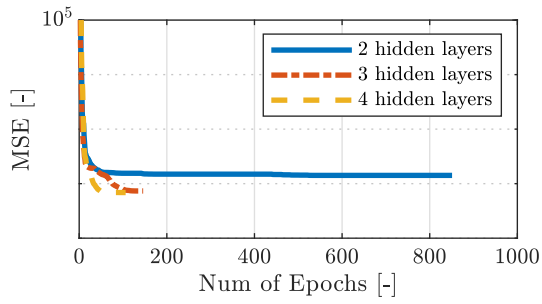


**Figure B-7:** The influence of the training data sets on the accuracy of the tanh networks  $\mathcal{U}_3$

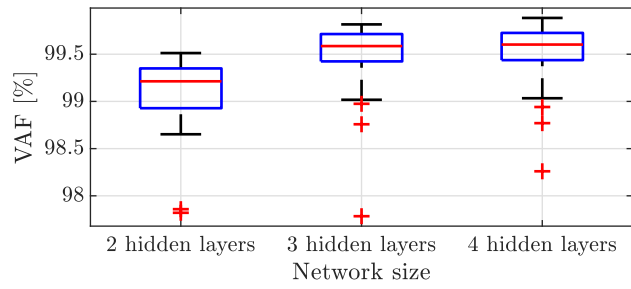
### B-3 The influence of the hidden layers on the accuracy of the ReLU networks



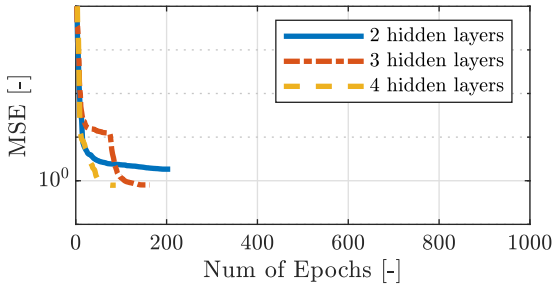
**Figure B-8:** The influence of the hidden layers on the accuracy of the *ReLU* neural networks trained on data set  $Q_1$



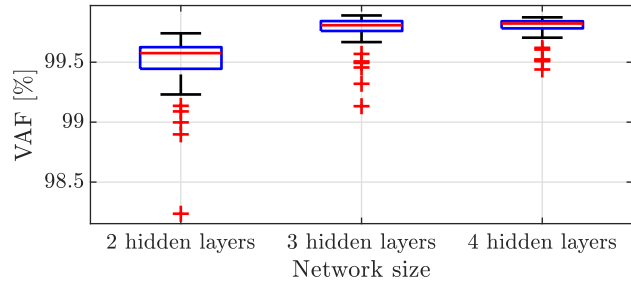
(a) Convergence process for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_2$



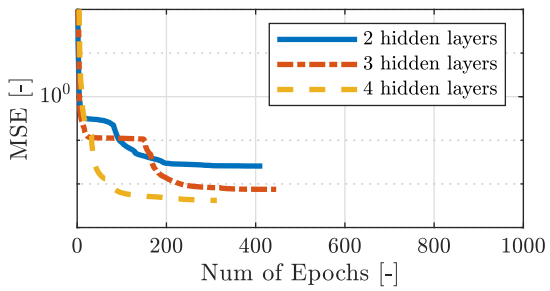
(b)  $\mathcal{V}_{Q,1}$  for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_2$



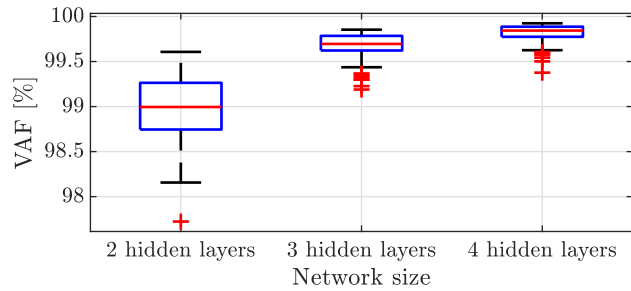
(c) Convergence process for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_2$



(d)  $\mathcal{V}_{Q,2}$  for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_2$

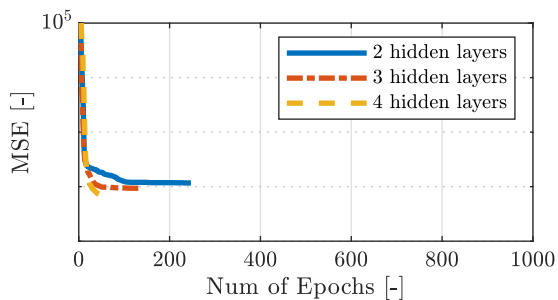


(e) Convergence process for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_2$

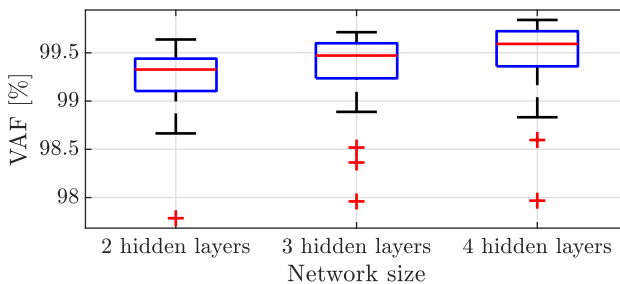


(f)  $\mathcal{V}_{Q,3}$  for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_2$

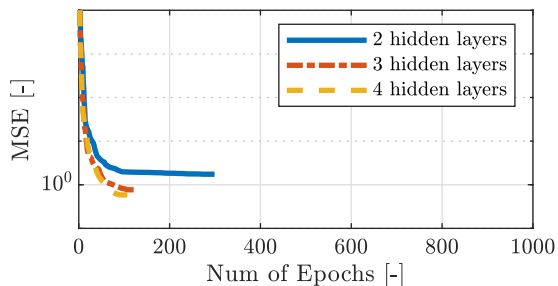
**Figure B-9:** The influence of the hidden layers on the accuracy of the *ReLU* neural networks trained on data set  $Q_2$



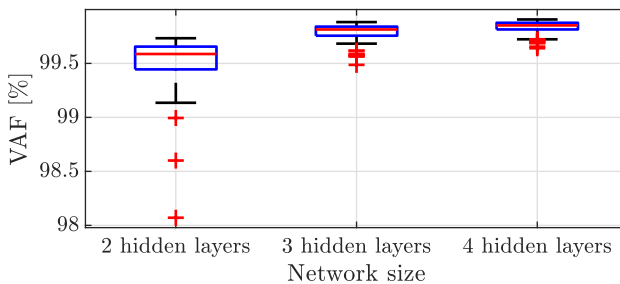
(a) Convergence process for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_3$



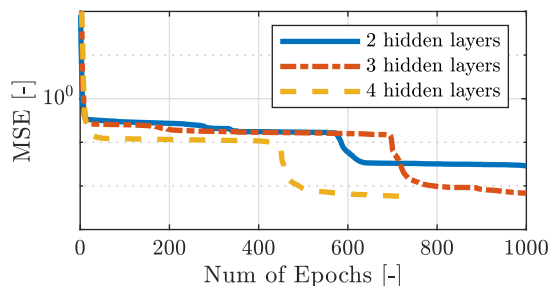
(b)  $\mathcal{V}_{Q,1}$  for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_3$



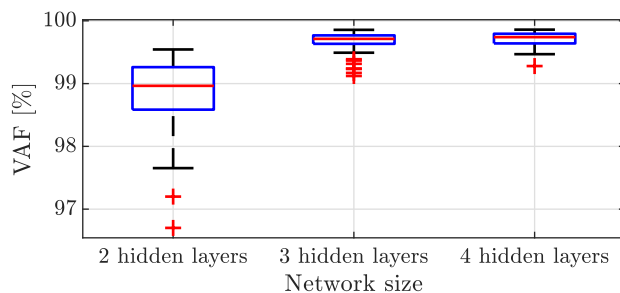
(c) Convergence process for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_3$



(d)  $\mathcal{V}_{Q,2}$  for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_3$

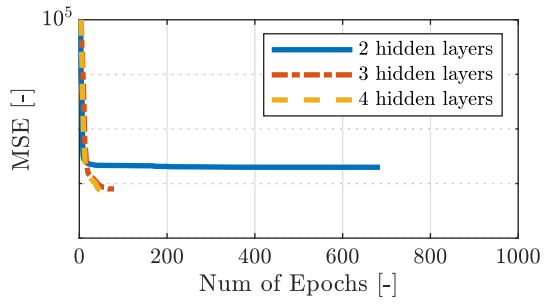


(e) Convergence process for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_3$

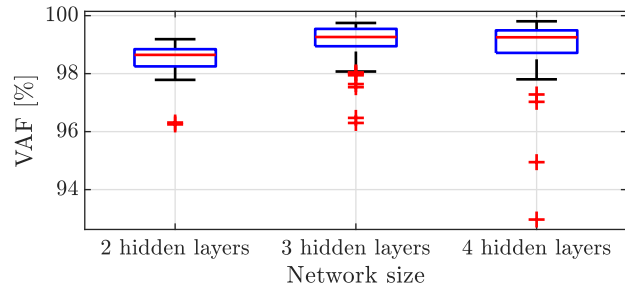


(f)  $\mathcal{V}_{Q,3}$  for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_3$

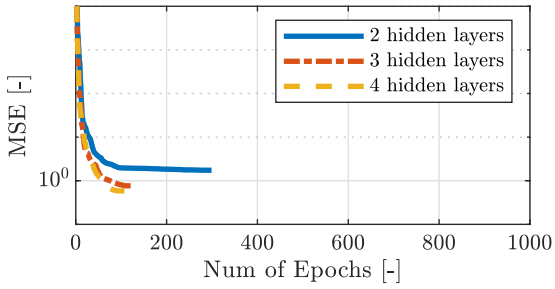
**Figure B-10:** The influence of the hidden layers on the accuracy of the *ReLU* neural networks trained on data set  $Q_3$



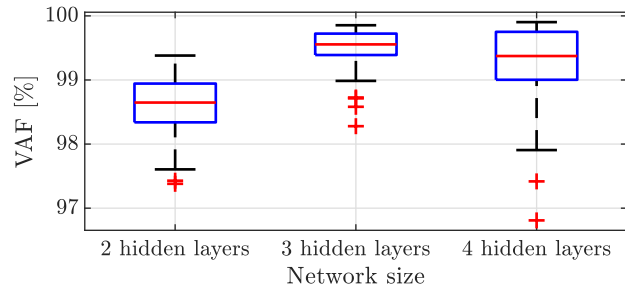
(a) Convergence process for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_4$



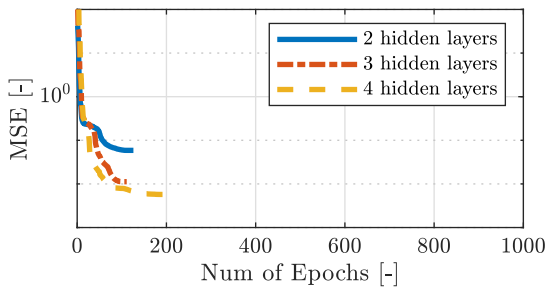
(b)  $\mathcal{V}_{Q,1}$  for the ReLU networks  $\mathcal{U}_1$  trained on  $Q_4$



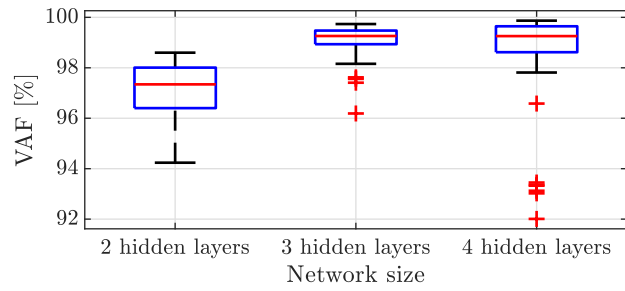
(c) Convergence process for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_4$



(d)  $\mathcal{V}_{Q,2}$  for the ReLU networks  $\mathcal{U}_2$  trained on  $Q_4$



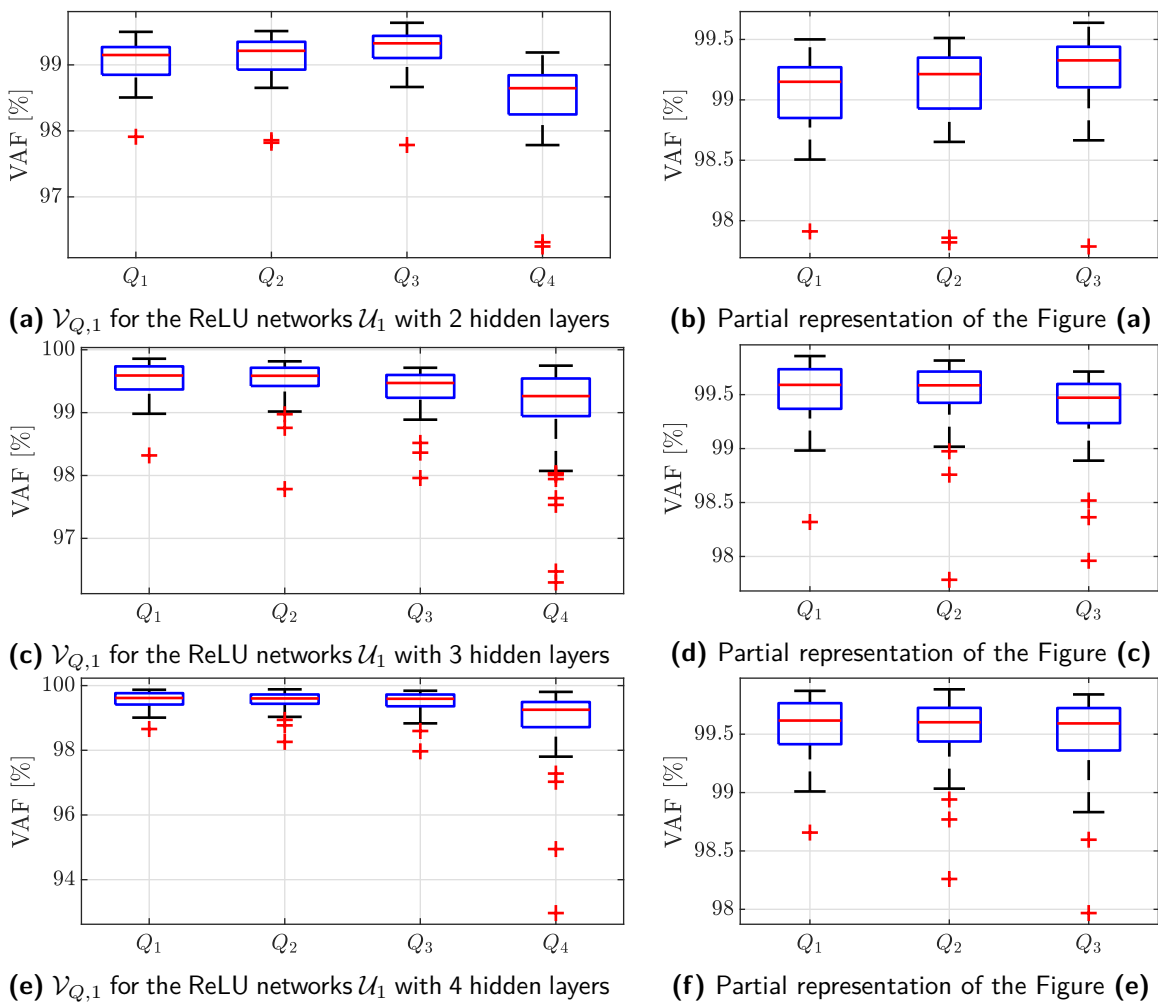
(e) Convergence process for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_4$



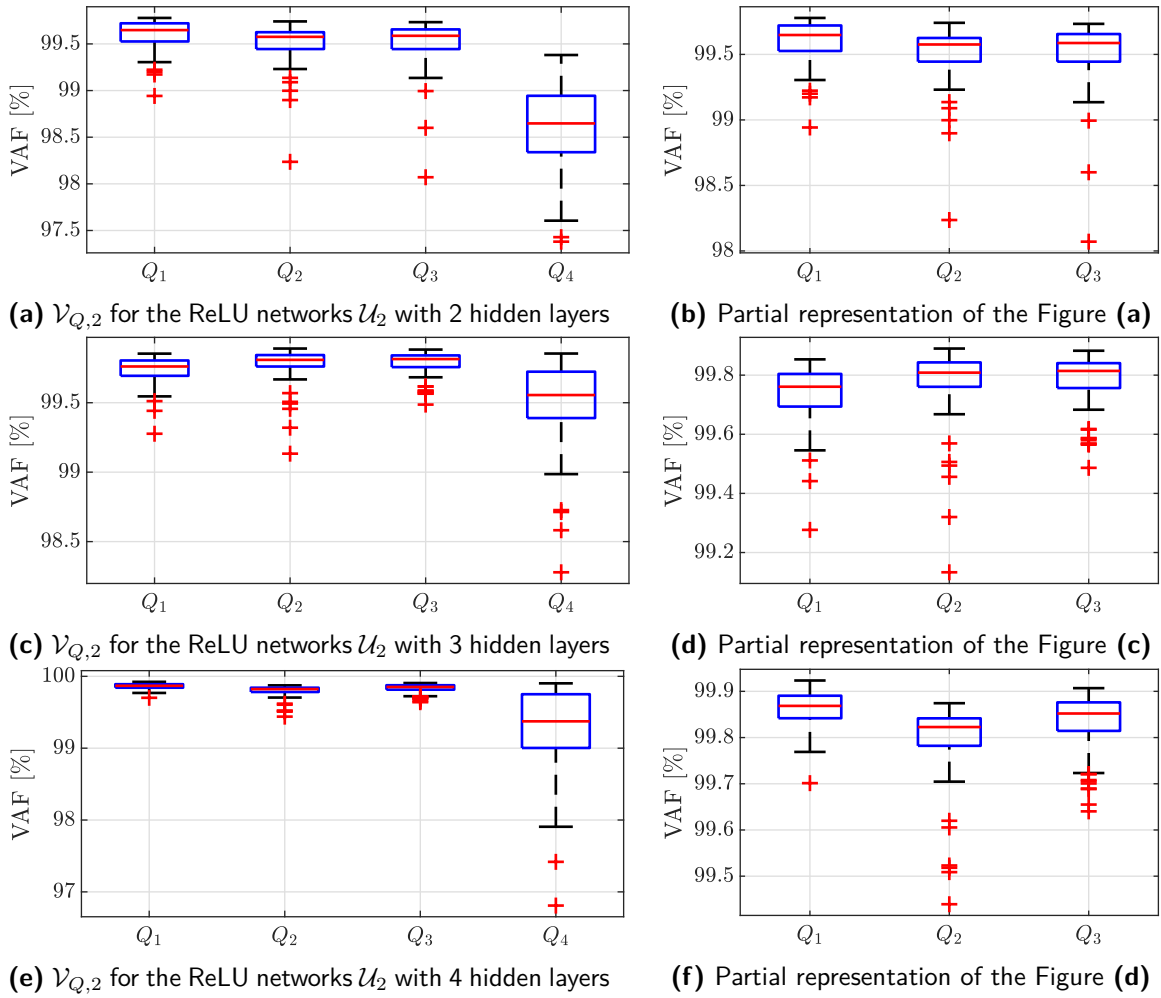
(f)  $\mathcal{V}_{Q,3}$  for the ReLU networks  $\mathcal{U}_3$  trained on  $Q_4$

**Figure B-11:** The influence of the hidden layers on the accuracy of the *ReLU* neural networks trained on data set  $Q_4$

### B-4 The influence of the training data set on the accuracy of the ReLU networks

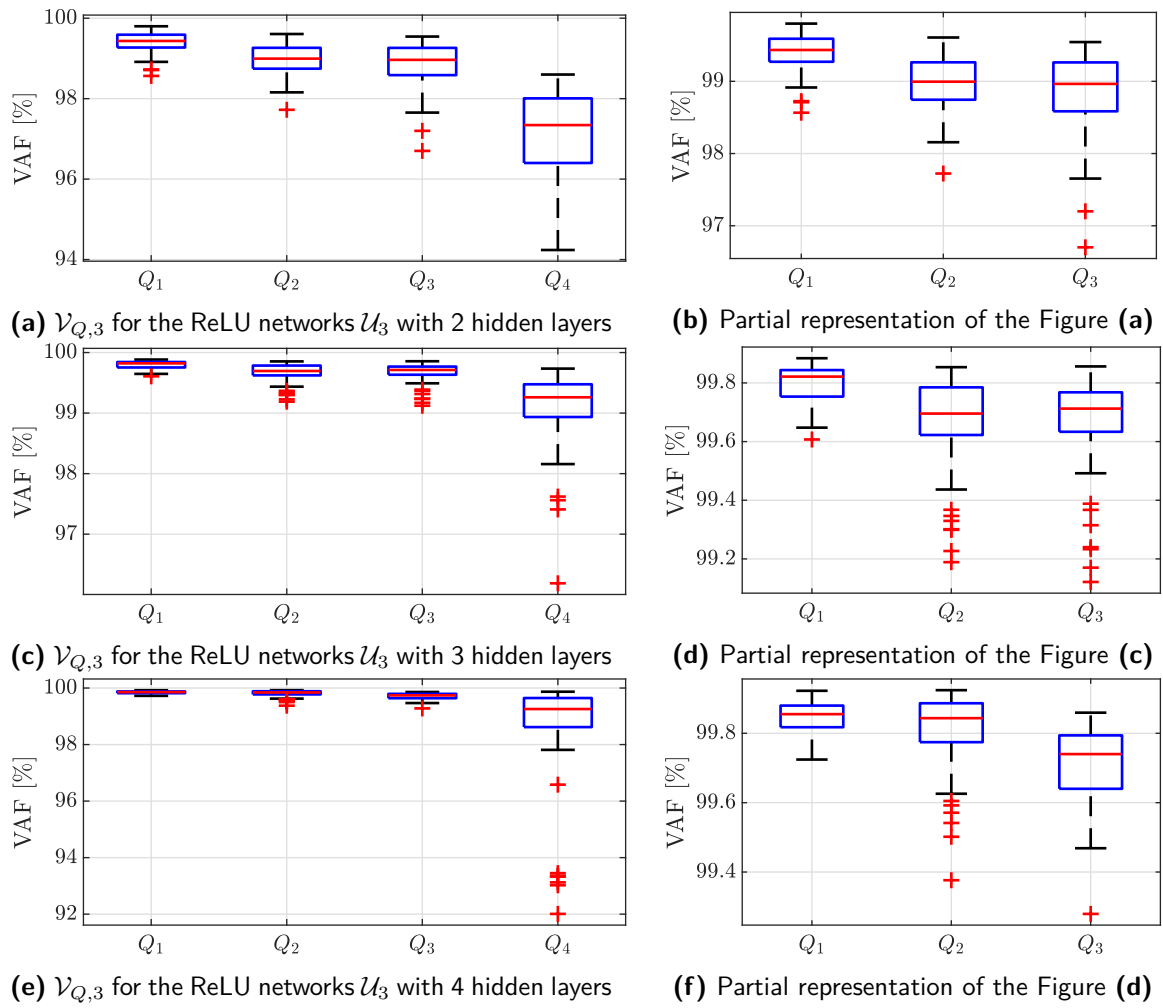


**Figure B-12:** The influence of the training data sets on the accuracy of the ReLU networks  $\mathcal{U}_1$

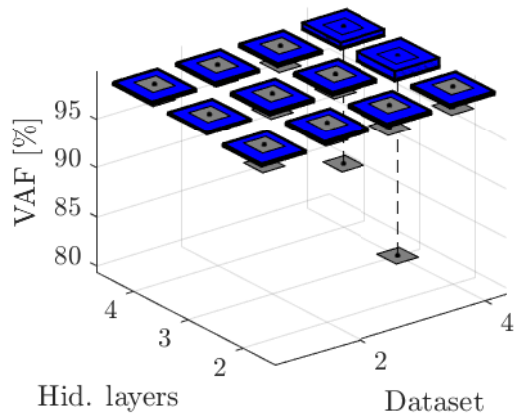
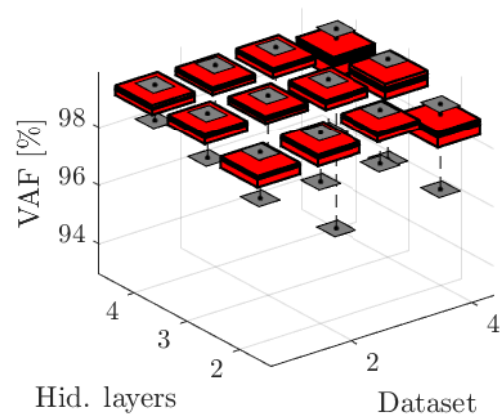
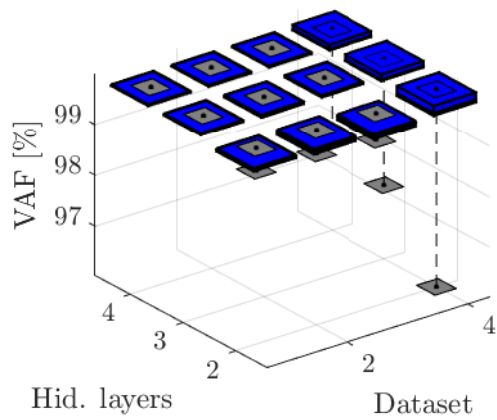
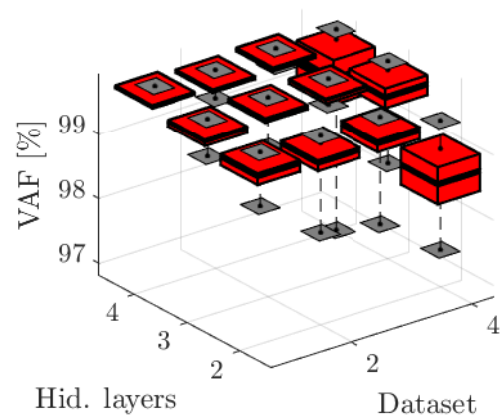
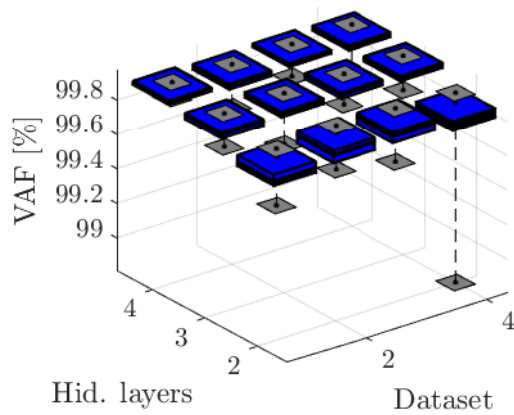
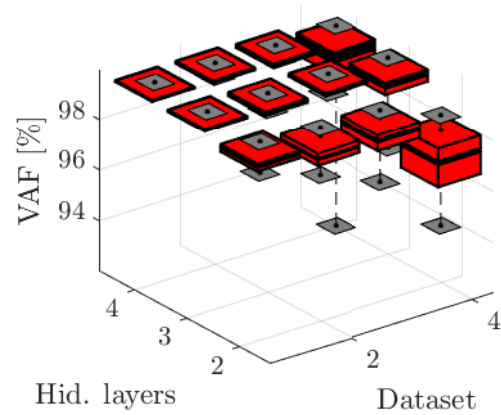


**Figure B-13:** The influence of the training data sets on the accuracy of the ReLU networks  $\mathcal{U}_2$



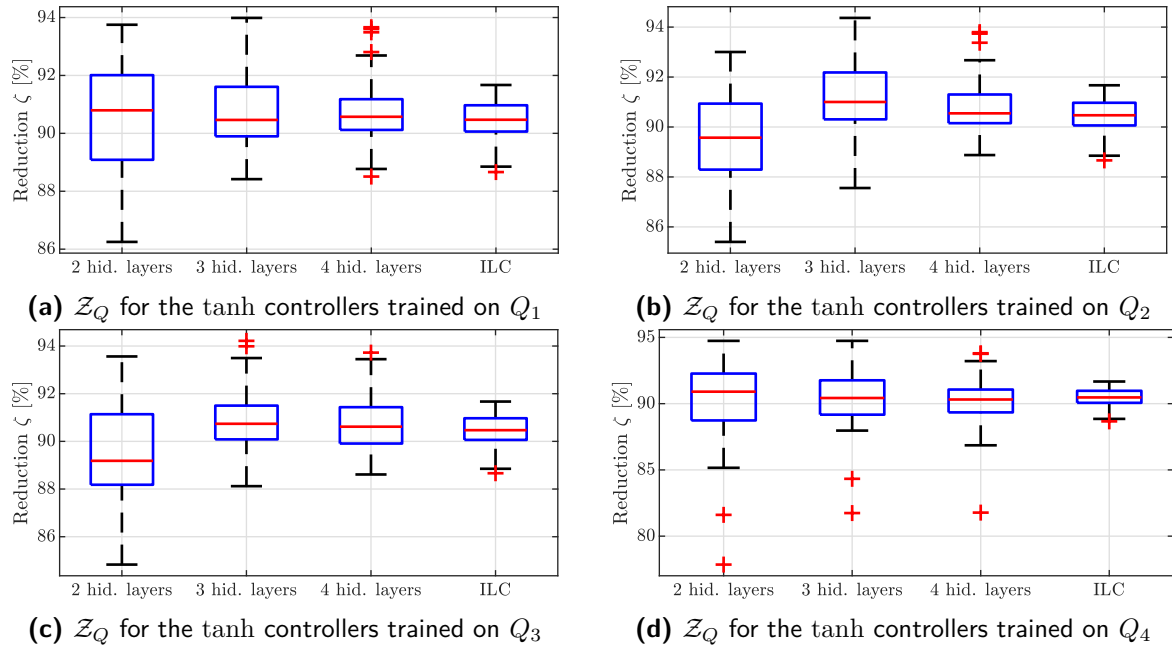


**Figure B-14:** The influence of the training data sets on the accuracy of the ReLU networks  $\mathcal{U}_3$

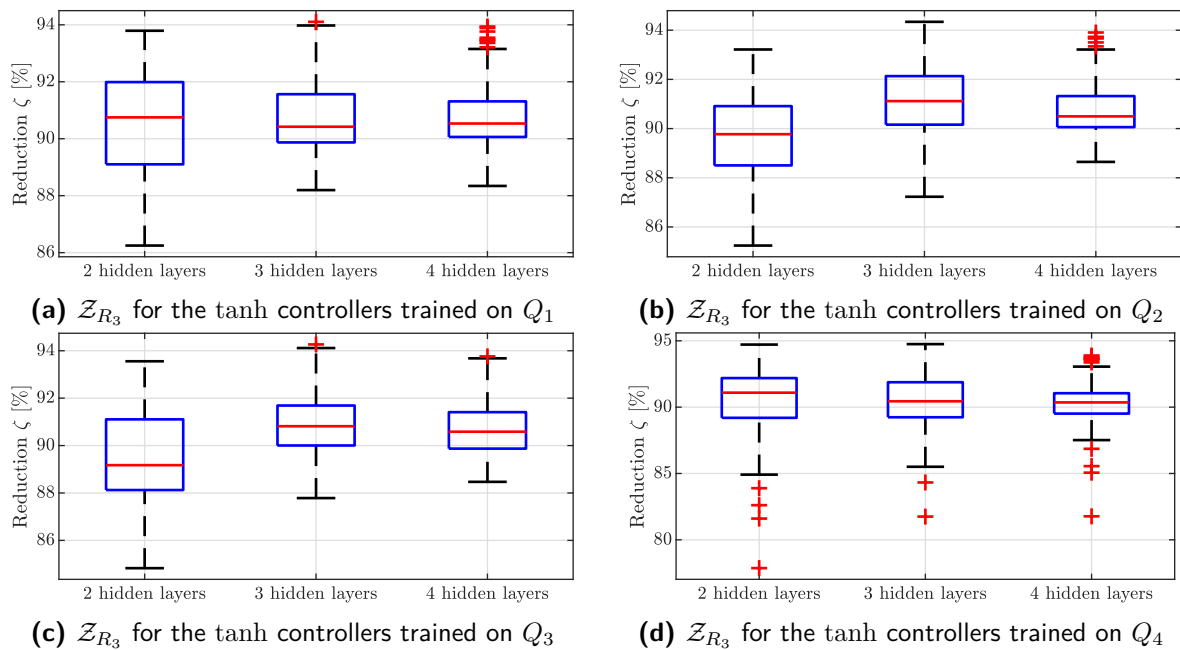
(a) All experiments for the tanh networks  $\mathcal{U}_1$ (b) All experiments for the ReLU networks  $\mathcal{U}_1$ (c) All experiments for the tanh networks  $\mathcal{U}_2$ (d) All experiments for the ReLU networks  $\mathcal{U}_2$ (e) All experiments for the tanh networks  $\mathcal{U}_3$ (f) All experiments for the ReLU networks  $\mathcal{U}_3$ 

**Figure B-15:** An overview of the influence of the training data on the performance of the neural networks

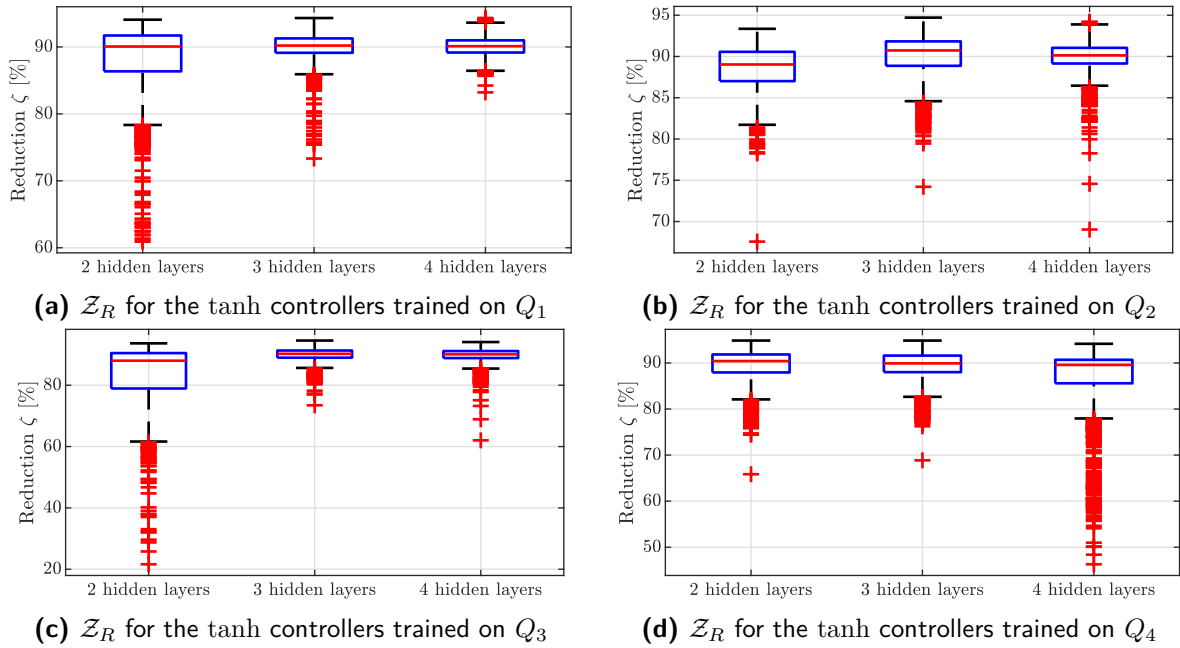
## B-5 The influence of the hidden layers on the performance of the tanh controllers on the training set Q



**Figure B-16:** The influence of the hidden layers on the performance of tanh controllers for signals in the training data set  $Q$

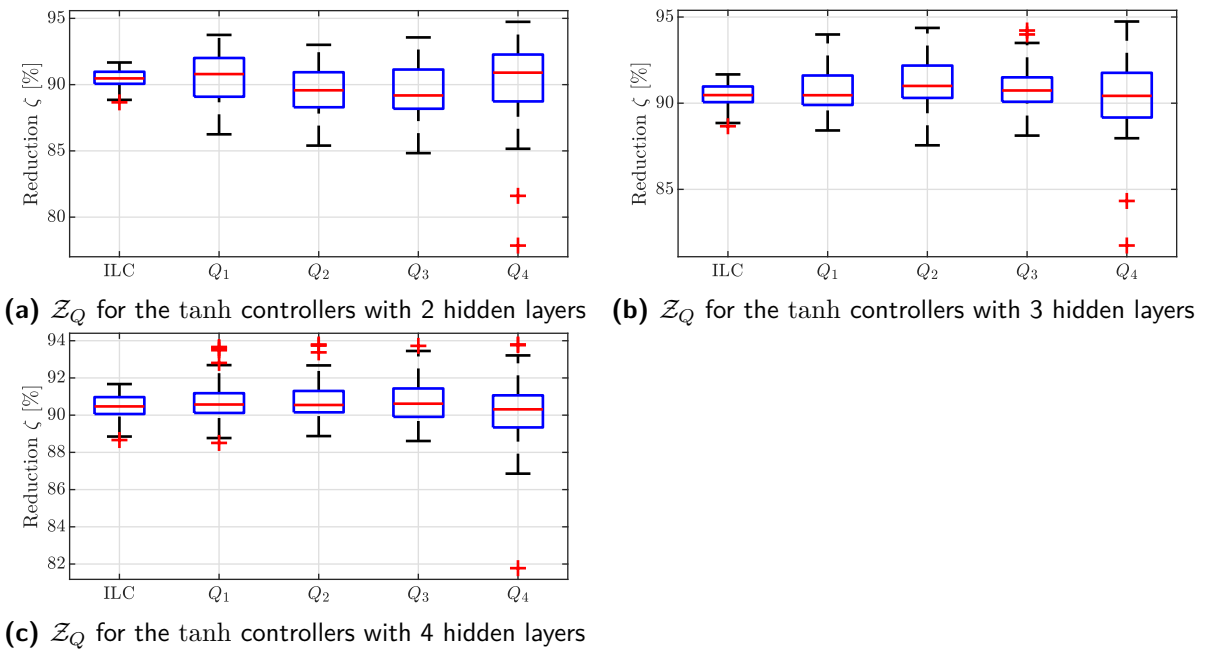


**Figure B-17:** The influence of the hidden layers on the performance of tanh controllers for signals in set  $R_3$

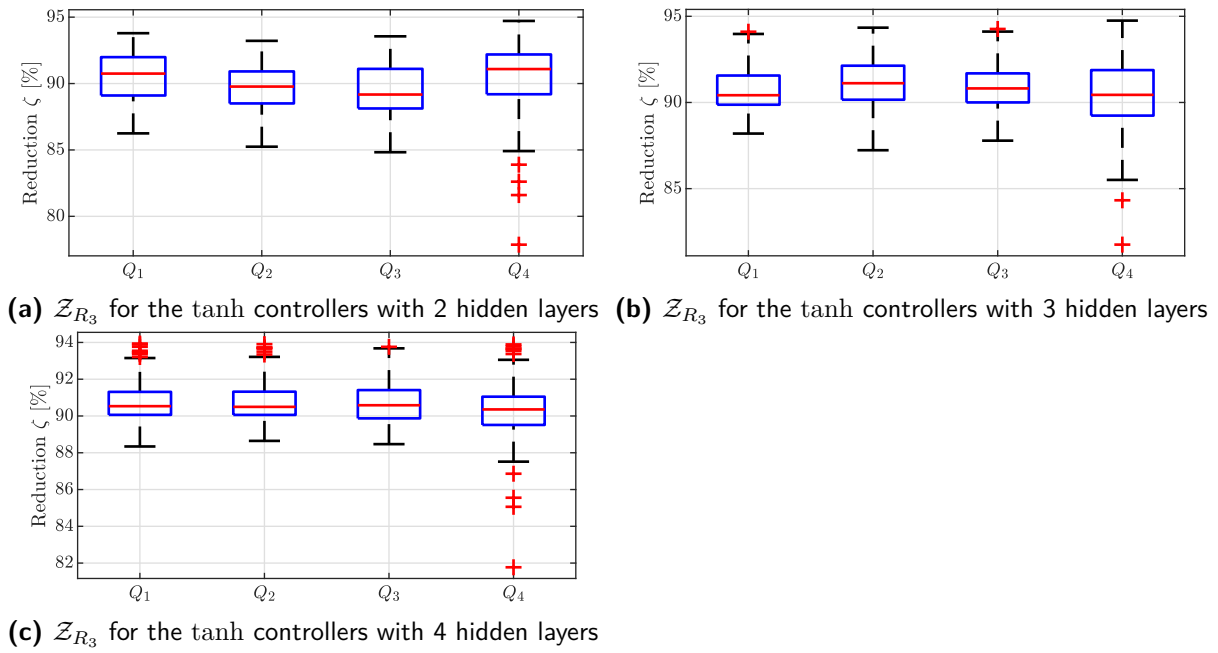


**Figure B-18:** The influence of the hidden layers on the performance of tanh controllers for signals in validation data set  $R$

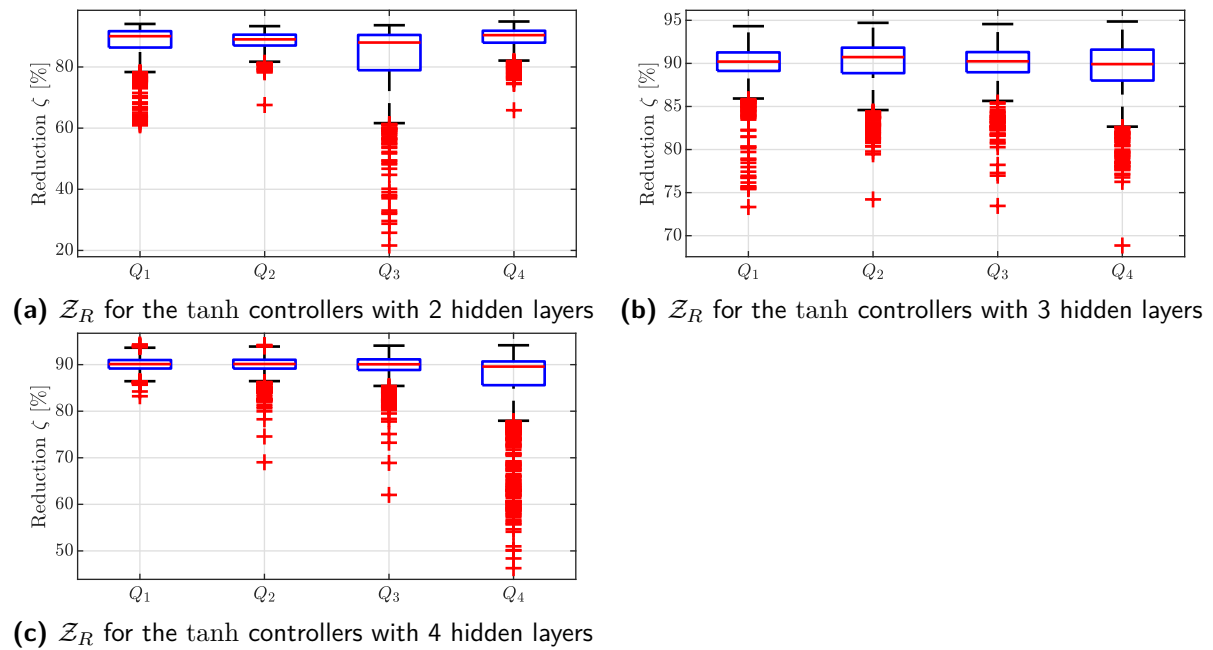
### B-6 The influence of the training data sets on the performance of the tanh controllers



**Figure B-19:** The influence of the training data on the performance of the tanh controllers on signals from the training data set  $Q$

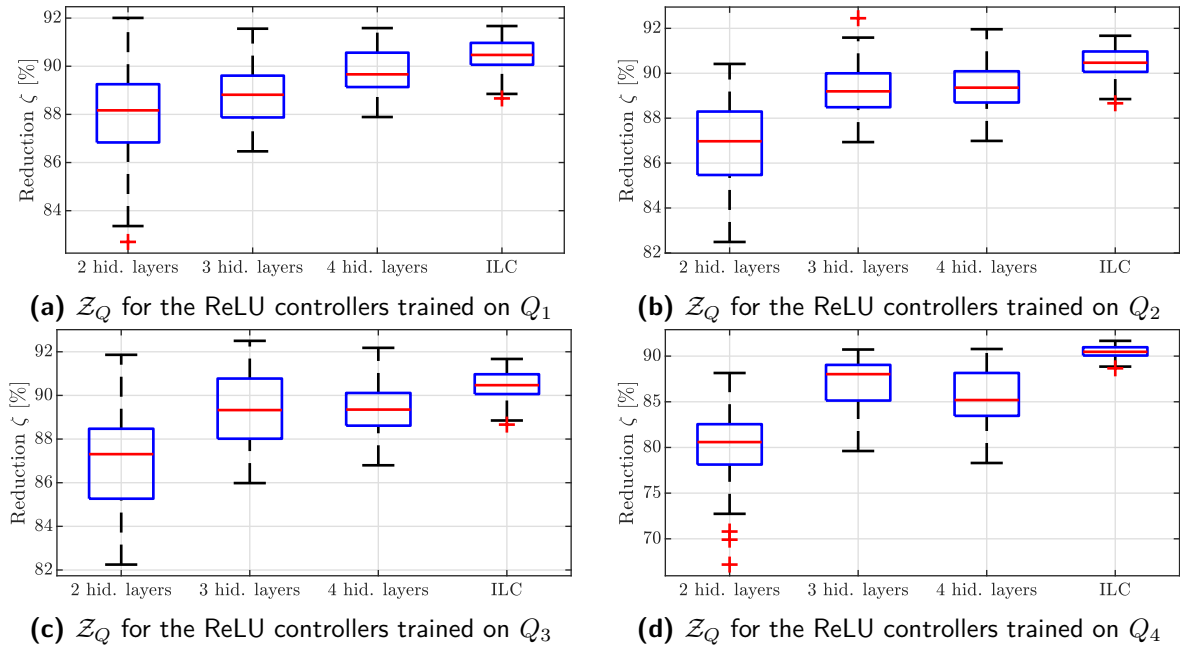


**Figure B-20:** The influence of the training data on the performance of the tanh controllers on signals from set  $R_3$

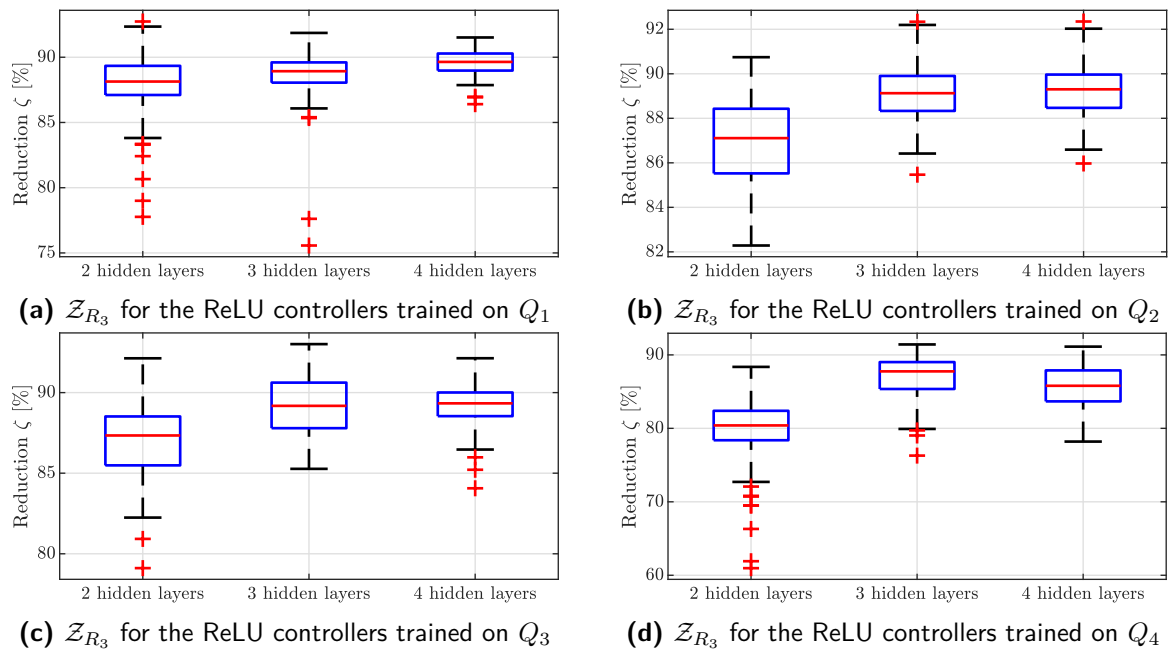


**Figure B-21:** The influence of the training data on the performance of the tanh controllers on signals from validation set  $R$

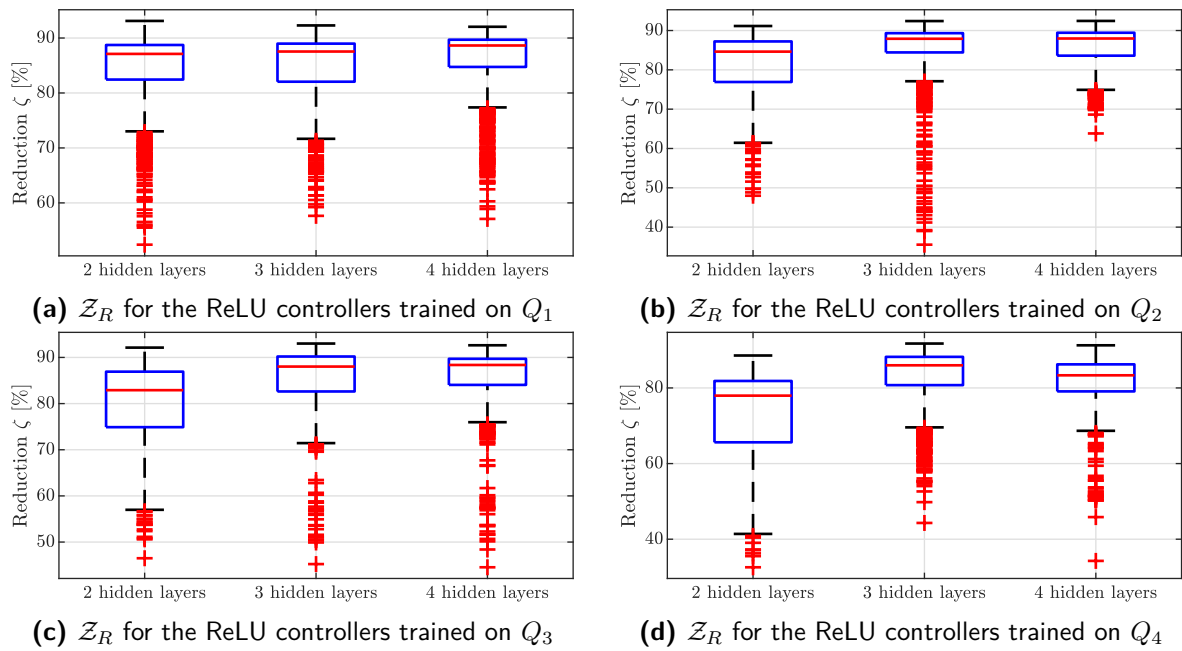
## B-7 The influence of the hidden layers on the performance of the ReLU controllers



**Figure B-22:** The influence of the hidden layers on the performance of the ReLU controllers on signals from training set  $Q$

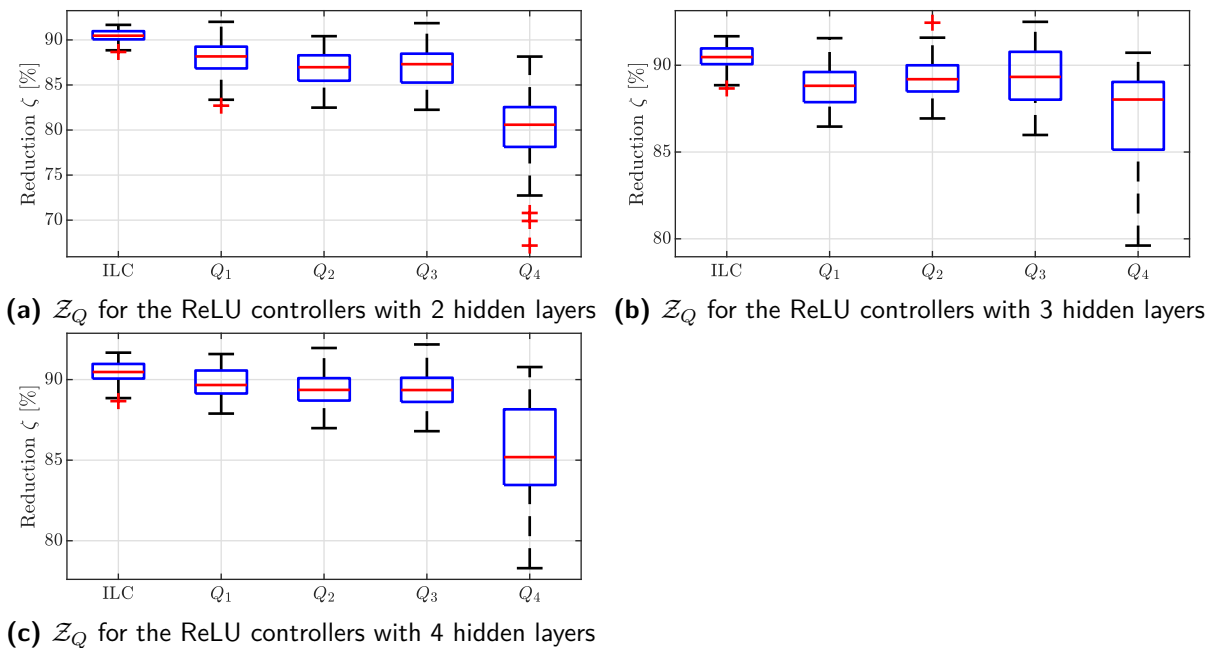


**Figure B-23:** The influence of the hidden layers on the performance of the ReLU controllers on signals from set  $R_3$

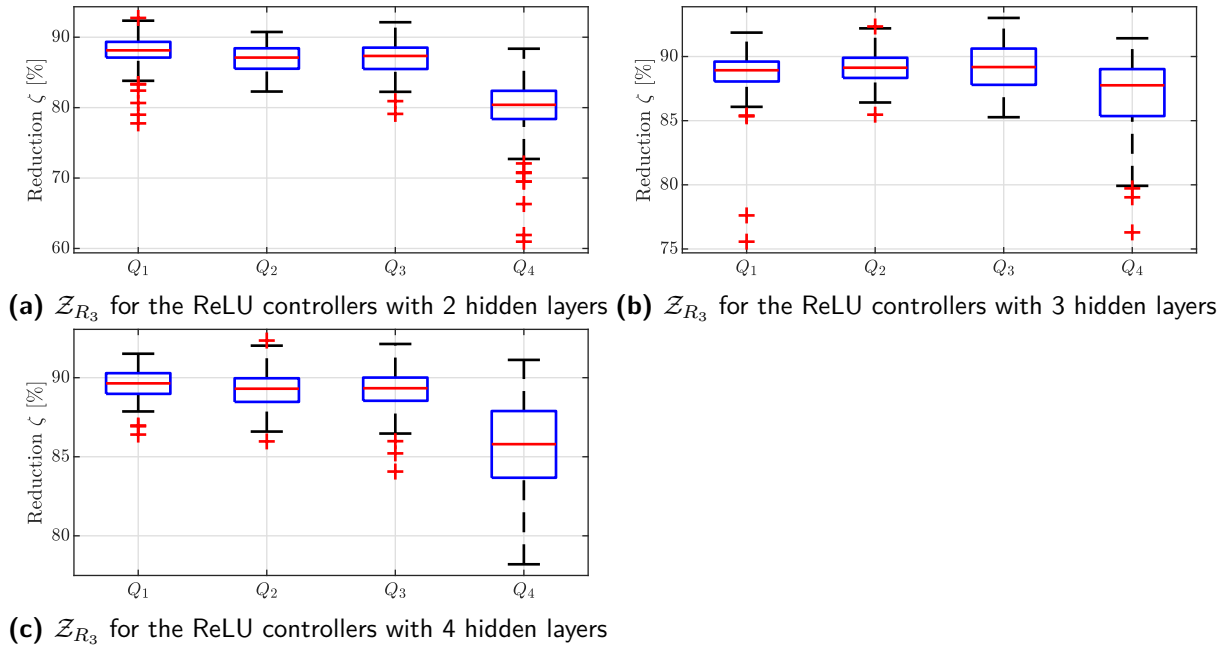


**Figure B-24:** The influence of the hidden layers on the performance of the ReLU controllers on signals from validation set  $R$

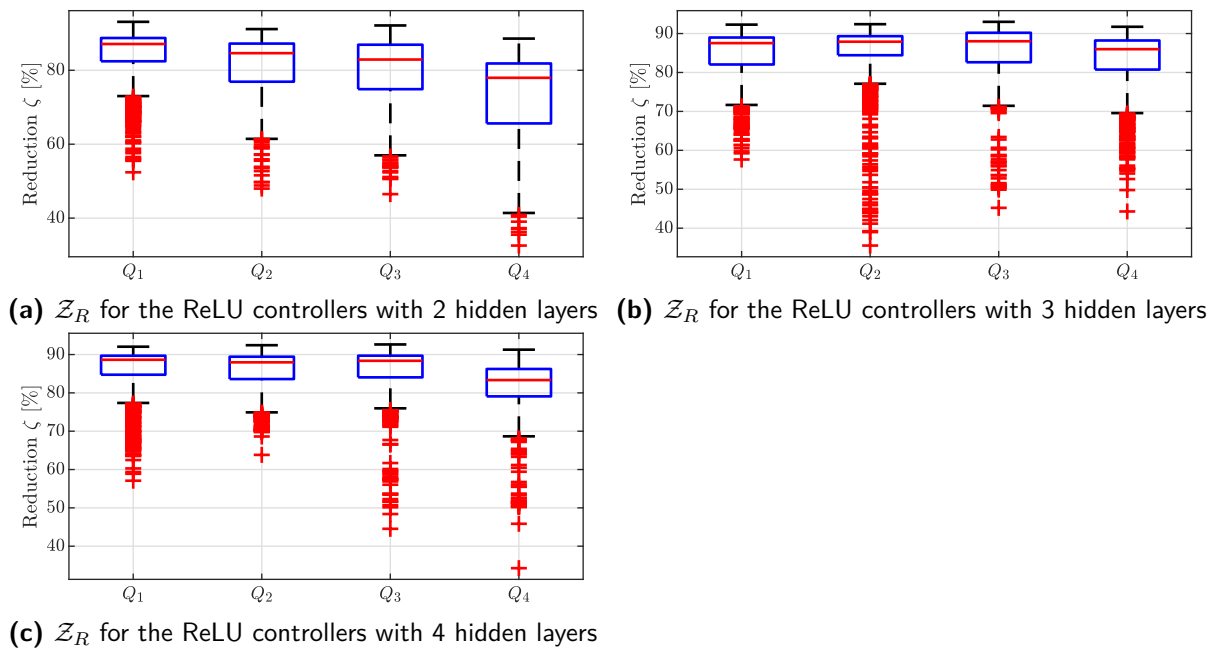
### B-8 The influence of the training data sets on the performance of the ReLU controllers



**Figure B-25:** The influence of the training data on the performance of the ReLU controllers on signals from training set  $Q$

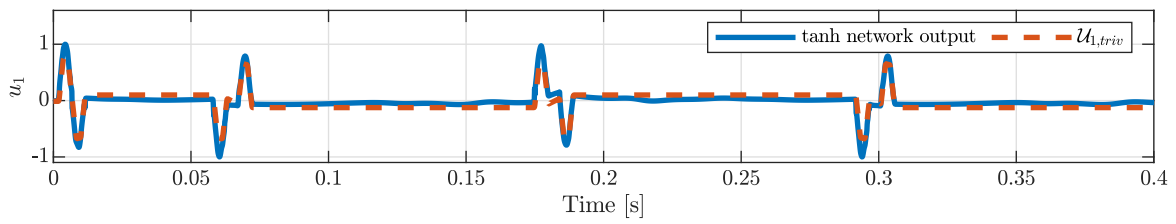
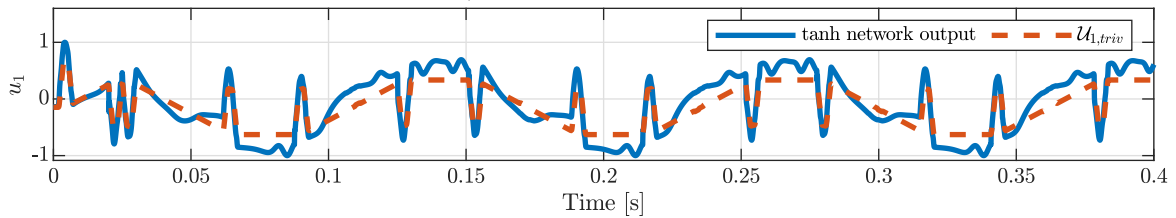
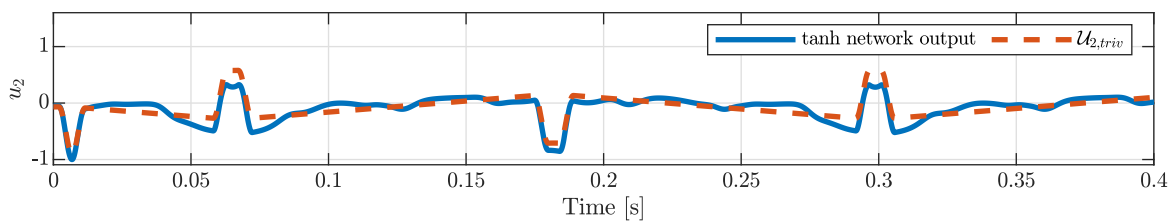
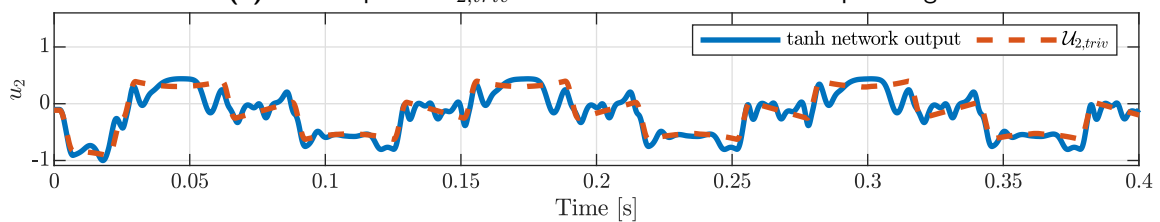


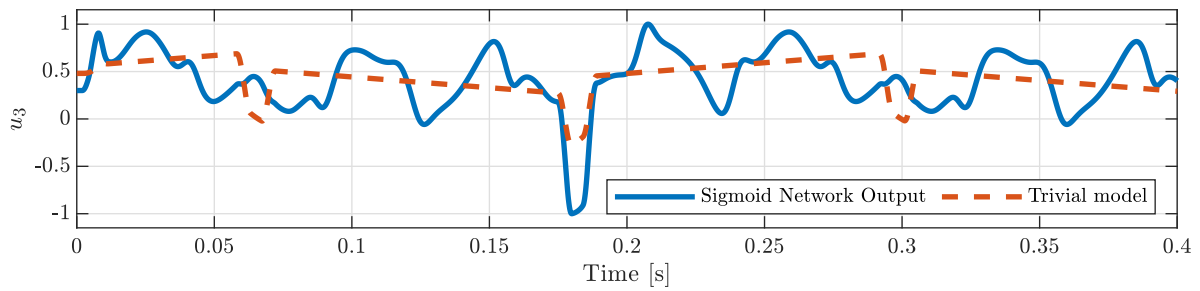
**Figure B-26:** The influence of the training data on the performance of the ReLU controllers on signals from set  $R_3$



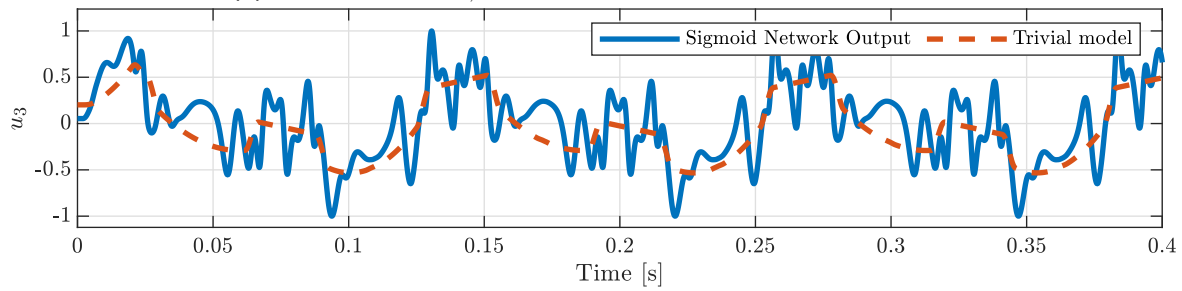
**Figure B-27:** The influence of the training data on the performance of the ReLU controllers on signals from validation set  $R$



(a) The output of  $\mathcal{U}_{1,triv}$  and the tanh network for setpoint signal 1(b) The output of  $\mathcal{U}_{1,triv}$  and the tanh network for setpoint signal 2**Figure B-28:** Analyzing the output of the tanh neural network  $\mathcal{U}_1$  with 3 hidden layers for signals outside the training set(a) The output of  $\mathcal{U}_{2,triv}$  and the tanh network for setpoint signal 1(b) The output of  $\mathcal{U}_{2,triv}$  and the tanh network for setpoint signal 2**Figure B-29:** Analyzing the output of the tanh neural network  $\mathcal{U}_2$  with 3 hidden layers for signals outside the training set



(a) The output of  $U_{3,triv}$  and the tanh network for setpoint signal 1



(b) The output of  $U_{3,triv}$  and the tanh network for setpoint signal 2


**Figure B-30:** Analyzing the output of the tanh neural network  $U_3$  with 3 hidden layers for signals outside the training set

---

# Bibliography

- [1] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. pp. 114 – 117, April 19 1965.
- [2] G. E. Moore *et al.*, “Progress in digital integrated electronics,” in *Electron Devices Meeting*, vol. 21, pp. 11–13, 1975.
- [3] M. M. Waldrop, “The chips are down for moore’s law,” *Nature News*, vol. 530, no. 7589, p. 144, 2016.
- [4] D. D. Bristow, M. Tharayil, and A. a.G. Alleyne, “A survey of iterative learning control,” *IEEE Control Systems Magazine*, vol. 26, no. June, pp. 96–114, 2006.
- [5] K. K. Tan, H. Dou, Y. Chen, and T. H. Lee, “High precision linear motor control via relay-tuning and iterative learning based on zero-phase filtering,” *IEEE Transactions on Control Systems Technology*, vol. 9, pp. 244–253, Mar 2001.
- [6] B. E. Helfrich, C. Lee, D. A. Bristow, X. H. Xiao, J. Dong, A. G. Alleyne, S. M. Salapaka, and P. M. Ferreira, “Combined  $h_\infty$ -feedback control and iterative learning control design with application to nanopositioning systems,” *IEEE Transactions on Control Systems Technology*, vol. 18, pp. 336–351, March 2010.
- [7] J. van Zundert, J. Bolder, S. Koekebakker, and T. Oomen, “Resource-efficient ilc for lti/ltv systems through lq tracking and stable inversion: Enabling large feedforward tasks on a position-dependent printer,” *Mechatronics*, vol. 38, pp. 76 – 90, 2016.
- [8] B. O. van der Meulen SH, Tousain RL, “Fixed structure feedforward controller design exploiting iterative trials: Application to a wafer stage and a desktop printer,” *J. Dyn. Sys. Meas. Control.*, 2008.
- [9] R. Poli, L. Vanneschi, W. B. Langdon, and N. F. McPhee, *Theoretical results in genetic programming: The next ten years?*, vol. 11. 2010.
- [10] S. A. Billings, *Nonlinear System identification.*: John Wiley & Sons, 2013.
- [11] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, “Nonlinear black-box modeling in system identification: a unified overview,” *Automatica*, vol. 31, no. 12, pp. 1691 – 1724, 1995. Trends in System Identification.

- [12] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [13] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, “Neural networks for control systems - a survey,” *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.
- [14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.
- [15] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.
- [16] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Output range analysis for deep feedforward neural networks,” in *NASA Formal Methods* (A. Dutle, C. Muñoz, and A. Narkawicz, eds.), (Cham), pp. 121–138, Springer International Publishing, 2018.
- [17] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming*. Lulu. com, 2008.
- [18] J. R. Koza, “Genetic programming as a means for programming computers by natural selection,” *Statistics and computing*, vol. 4, no. 2, pp. 87–112, 1994.
- [19] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” *Science*, vol. 324, no. 5923, pp. 81–85, 2009.
- [20] W. La Cava, K. Danai, L. Spector, P. Fleming, A. Wright, and M. Lackner, “Automatic identification of wind turbine models using evolutionary multiobjective optimization,” *Renewable Energy*, vol. 87, pp. 892–902, 2016.
- [21] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.
- [22] Z. Kurd and T. Kelly, “Establishing safety criteria for artificial neural networks,” in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 163–169, Springer, 2003.
- [23] M. Verhaegen and V. Verdult, *Filtering and system identification: a least squares approach*. Cambridge university press, 2007.
- [24] M. Schmidt and H. Lipson, “Age-fitness pareto optimization,” in *Genetic Programming Theory and Practice VIII*, pp. 129–146, Springer, 2011.
- [25] R. Babuška, *Knowledge-Based Control Systems*. Delft Center for Systems and Control, 2010.
- [26] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.
- [27] M. H. Stone, “The generalized weierstrass approximation theorem,” *Mathematics Magazine*, vol. 21, no. 5, pp. 237–254, 1948.
- [28] Evolved Analytics LLC ®, “Data Modeler 2018.12.03,”

- 
- [29] D. Nguyen and B. Widrow, “Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights,” in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 21–26, IEEE, 1990.
- [30] Mathworks , “Deep learning toolbox R2018B ,”
- [31] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O’neill, “Grammar-based genetic programming: a survey,” *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 365–396, 2010.
- [32] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verisig: verifying safety properties of hybrid systems with neural network controllers,” *arXiv preprint arXiv:1811.01828*, 2018.
- [33] X. Chen, E. Ábrahám, and S. Sankaranarayanan, “Flow\*: An analyzer for non-linear hybrid systems,” in *International Conference on Computer Aided Verification*, pp. 258–263, Springer, 2013.
- [34] S. Rump, “INTLAB - INTerval LABoratory,” in *Developments in Reliable Computing* (T. Csendes, ed.), pp. 77–104, Dordrecht: Kluwer Academic Publishers, 1999. <http://www.ti3.tuhh.de/rump/>.
- [35] R. Li, B. R. Noack, L. Cordier, J. Borée, and F. Harambat, “Drag reduction of a car model by linear genetic programming control,” *Experiments in Fluids*, vol. 58, no. 8, p. 103, 2017.

