Recurrent Neural Networks for PID Auto-tuning

Adaptive Control and Identification of Nonlinear Systems

Elias Reichensdörfer, Johannes Günther, Klaus Diepold

Technical Report

Recurrent Neural Networks for PID Auto-tuning

Adaptive Control and Identification of Nonlinear Systems

Elias Reichensdörfer, Johannes Günther, Klaus Diepold

September 9, 2017

Institute for Data Processing
Technische Universität München

Elias Reichensdörfer, Johannes Günther, Klaus Diepold. *Recurrent Neural Networks for PID Auto-tuning. Adaptive Control and Identification of Nonlinear Systems.* Technical Report, Technische Universität München, Munich, Germany, 2017.

Supervised by Prof. Dr.-Ing. K. Diepold ; submitted on September 9, 2017 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

Institute for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de`.

Abstract

In this work it is investigated, how recurrent neural networks with internal, time-dependent dynamics can be used to perform a nonlinear adaptation of parameters of linear PID controllers in closed-loop control systems. For this purpose, recurrent neural networks are embedded into the control loop and adapted by classical machine learning techniques. The outcomes are then compared against both PID controllers with fixed parameters and several sophisticated state-of-the-art control methods. Optionally, a second recurrent neural network is used to identify the control system. Simulation experiments are carried out on four different dynamical systems, namely a nonlinear two-tank system, a linear plant with a large input delay, a nonlinear model of an inverse pendulum on a cart and a chaotic thermal convection loop. It is shown that the proposed method is able to control all four systems in a satisfying way, even outperforming classical control structures most of the time.

Zusammenfassung

Im Rahmen dieser Arbeit werden rekurrente neuronale Netze mit interner, zeitabhängiger Dynamik dafür eingesetzt, die Parameter von linearen PID-Reglern nichtlinear an den zu kontrollierenden Prozess zu adaptieren. Für diesen Zweck werden rekurrente neuronale Netze in den geschlossenen Regelkreis eingebettet und mit Hilfe von klassischen maschinellen Lernverfahren trainiert. Die Ergebnisse werden sowohl mit PID-Reglern mit fest eingestellten Parametern als auch mit verschiedenen aktuellen Methoden der Regelungstechnik verglichen. Optional wird ein zweites neuronales Netz eingesetzt, um die zu regelnde Strecke zu identifizieren. Für vier verschiedene Regelstrecken werden Simulationen durchgeführt, nämlich einem nichtlinearen Zweitank System, einem linearen PT1-System mit hoher Totzeit, einem nichtlinearen Modell eines invertierten Pendels sowie einem experimentellen Fluidsystem mit chaotischem Verhalten. Es konnte gezeigt werden, dass die vorgestellten Verfahren in der Lage sind, alle vier Regelstrecken zu kontrollieren und dabei die klassischen Verfahren in den meisten Fällen übertreffen.

Contents

Glossaries

List of Symbols

A_i	Cross-sectional area of tank i of a two-tank system
A_{oi}	Cross-sectional orifice area of tank i of a two-tank system
α	Amplitude step size of an APRBS
$\boldsymbol{b}$	Bias weight vector of a neural network
β	Substituted Rayleight number of a fluid
$\boldsymbol{e}$	Error signal of a closed-loop control system
ε	Step width used for numerical backwards difference calculations
ε_{min}	Machine epsilon (in this paper of double precision)
η	Learning rate used in Hebbian learning
F_x	Force in x direction of the inverted pendulum on a cart
g	Gravitational constant
γ	Discount factor used in reinforcement learning
$\boldsymbol{h}$	Helper vector to select the correct ε at numerical Jacobian calculations
$\boldsymbol{J}_{\omega}$	Jacobian matrix with respect to a weight vector ω
$\hat{\boldsymbol{J}}_{\omega}$	Numerical approximation of $\boldsymbol{J}_{\omega}$
K	Gain of a LTI system with input delay
k_{opt}	Optimal scaling factor for generating a PRBS
k_P	Pump constant of a two-tank system
K_d, K_i, K_d	Proportional, integral and derivative part of a PID controller
l	Inverted pendulum length to its mass center
M	Mass of the cart of an inverted pendulum on a cart
m	Mass of the pendulum of an inverted pendulum on a cart
n_{int}	Number of intervals of a PRBS
n_{opt}	Optimal grade of a linear feedback shift register
$\boldsymbol{\omega}$	Weight vector of a neural network
p	Prandtl number of a fluid
ϕ	Activation function of a single neuron
π	Policy function of a reinforcement learning agent
Q_t	Q-value at time step t in Q-learning
R_t	Cumulative reward at time step t in reinforcement learning
s	Complex frequency of the Laplace transform

$s_{\max}$	Maximum hold sequence of a PRBS
T, T_D	Time constant T and input delay T_D of a LTI with input delay
θ	Angle of the inverted pendulum on a cart
T_Σ	Total time constant of a general LTI system
$\boldsymbol{u}$	System input signal of a closed-loop control system
$V(s)$	Value function of a reinforcement learning agent
$\boldsymbol{W}^i$	Adjacency matrix of delay layer i of a GDNN
$\boldsymbol{w}$	Setpoint signal of a closed-loop control system
$\boldsymbol{y}$	System output signal of a closed-loop control system
$\boldsymbol{z}$	Disturbance signal of a closed-loop control system

Abbreviations

ANN	Artificial neural network
APRBS	Amplitude modulated pseudo-random binary signal
EEG	Electroencephalography
FFN	Feed-forward network
GA	Genetic algorithm
GDNN	General dynamic neural network
HMM	Hidden Markov model
k-NN	K-nearest neighbor
LQ regulator	Linear quadratic regulator
LTI system	Linear time invariant system
MLP	Multilayer perceptron
NRMSD	Normalized root-mean-square deviation
ODE solver	Ordinary differential equation solver
PID controller	Proportional-Integral-Derivative controller
PRBS	Pseudo-random binary signal
RAII	Resource acquisition is initialization
RAM	Random access memory
RBNN	Radial basis neural network
RMSD	Root-mean-square deviation
SIMO	Single-input and multiple-output
SNR	Signal-to-noise ratio
SOM	Self-organizing map
SRNN	Simple recurrent neural network
SVM	Support vector machine
TD-learning	Temporal difference learning
XOR	Exclusive or

1. Introduction

1.1. Problem Statement and Motivation

Automation and control of industrial processes is a multidisciplinary effort and has become an important and fundamental part of our every-days lives within the last decades. Nearly every part of our environment and the tools we are using is a product of this development: Computers, smart-phones, cars, airplanes and many other electric devices and appliances, but also many non-electric products of our daily lives like food, medicine or clothes are being produced with a high degree of automation. While this increasing level of automation has brought significant benefits, such as higher efficiency at lower costs and less error rates, the problem complexity of the automation of more complicated tasks is continuously increasing. There are several reasons for this phenomenon. One specific is that most (industrial) processes that are about to be automatized exhibit nonlinear behavior [1]. Therefore, nonlinear system control is of high interest in research and a topic of many current publications [2]. However, it is generally difficult to control nonlinear systems [3, 4, 5]. Moreover, linear plants that have a non neglectable time delay are sometimes hard to control [6, 7, 8]. The need for more intelligent solutions to such problems can also be exemplified by what is called “Industry 4.0”, or the fourth industrial revolution. This pushes on a development from embedded systems to so-called Cyber-Physical Systems. Desirable key features of such systems are: “Intelligence”, “Autonomy” and “Self-Learning” [9, pp. 6-12]. Features that partially apply to machine learning techniques and are potentially helpful for nonlinear system control.

There exist numerous different approaches to handle those difficult-to-control plants. In [10] for example, an adaptive sliding mode fuzzy PD algorithm is applied to achieve position control of a robot manipulator. In contrast to that, [11] use a fuzzy PID controller with a feed-forward neural network to control a cascade two-tank system. An adaptive neural network to control a wheeled inverted pendulum is presented in [12]. These are just a handful of examples for nonlinear systems, as there exist many more and for each problem there are various approaches to handle the difficulties of the processes. Nevertheless, the vast majority in industrial practice still uses the standard PID controller [13, 14, 15, 16, 17]. This may relate to the fact that the PID controller is in general one of the best explored controllers and it is fairly easy to implement. Although it is a linear controller, it shows adequate performance even for some nonlinear processes and is quite robust towards measurement noise. Eventually, as the PID controller has only three degrees of freedom, it is also quite easy to parametrize it by trial-and-error and therefore convenient and suitable for practical applications, even without a detailed understanding of the entire control

theory behind it. More sophisticated methods for nonlinear systems like the ones mentioned above often suffer from a higher complexity both in implementation and theoretical understanding. Their methodology might be unfamiliar to many practice-oriented people and discourage them to actually implement these methods. This can be an unwanted barrier for such methods to attract a bigger audience and being applied in more industrial real-world applications.

Therefore, the aim of this work is the combination of the traditional PID control approach with recurrent neural networks. The ultimate goal is then to train a recurrent neural network in such a way, that it adopts the controllers parameters during execution to improve its performance on the control task. This adaptive PID control structure is then benchmarked against the standard PID controller as well as some more advanced state-of-the-art methods. Several similar approaches already exist. Examples are the use of a colonial competitive genetic algorithm to design a PID controller for a distillation column process [18], or a genetic algorithm to tune a PID controller for Glucose concentration control [19]. The authors of [20] apply a particle swarm optimization algorithm to the tuning problem of the PID controller and validate their approach on several physical plants like a bio-reactor, a linear plant with time delay and an isothermal continuous stirred tank. A feed-forward neural network is used to support a PI controller which is controlling a heating coil by [21]. Another application is an online plant identification for PID auto-tuning with feed-forward radial basis neural networks (RBNN) for speed control of a triphasic AC motor [22].

This list is not exhaustive and there are many more publications dealing with the topic of PID auto-tuning, some of them using neural networks as well. However, most of those publications focus on purely feed-forward neural networks with no tapped-delay lines between two neuron connections and therefore without an internal memory. Also most methods confine themselves on either system identification or parameter tuning with neural networks, but not both at the same time. The novelty of this paper is founded in the investigation of those methods for PID auto-tuning and whether the internal, time-dependent dynamics of recurrent neural networks improve the control of dynamical systems.

1.2. Structure of this work

This thesis is divided into seven sections. In the first section the motivation is refined and a short introduction is given. The second section gives a detailed overview about several state-of-the-art methods for PID auto-tuning and system identification. The third section describes the model of the general dynamic neural network (GDNN) and outlines a novel algorithm for generating optimal excitation signals for dynamical systems. In the next section, the developed control structures using recurrent neural networks are presented. Afterward, the fifth section shows simulation results for four different control processes using this structures. Following in the sixth section, a comparison with the standard PID control structure and other modern techniques is carried out. Finally a conclusion is drawn and an outlook for future work is presented in the seventh section.

2. State-of-the-Art

2.1. Control Theory

Control theory deals with the behavioral control of dynamical systems. It is widely applied in the industry since many industrial processes can be modeled as a dynamical system represented by differential equations. The structure is usually divided in two different parts, the controller and the system (also: Plant) that is about to be controlled. The output $y(t)$ of the control plant gives feedback to the controller which can then take appropriate actions $u(t)$ based on the derivation $e(t)$ between the setpoint $w(t)$ and the output of the system, including an external disturbance $z(t)$. This relationship is illustrated in figure 2.1. Since the focus of this paper is PID tuning, it would be to detailed to give a full introduction to this topic. The interested reader can refer to literature about control theory [23, 24, 25].

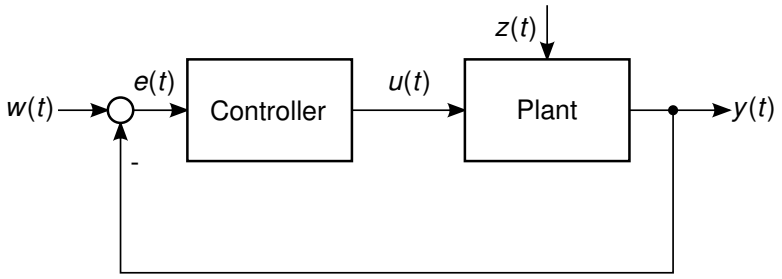


Figure 2.1.: A closed-loop control system

2.1.1. PID Control

In this section a short abstract about the widely used PID control structure is provided. This control structure is a linear dynamical systems and has several problems with delayed or nonlinear plants [26, 27]. PID control stands for “Proportional-Integral-Derivative” control [28]. It receives the difference between the current system output and the desired setpoint as an input and tries to minimize this difference. The PID controller has three different parameters, one that takes into account the current error (P-part), one that represents an accumulation of the past errors (I-part) and one that predicts the next error by calculating its derivative (D-part). Each of these errors is weighted by some scalar value (K_P, K_I, K_D) and summed up. This sum is then used as an input to the control plant. Figure 2.2 shows the structure of a PID controller embedded into a closed-loop control system. Here

the complex valued variable s stands for a differentiation, $\frac{1}{s}$ for an integration over the controller input $e(t)$.

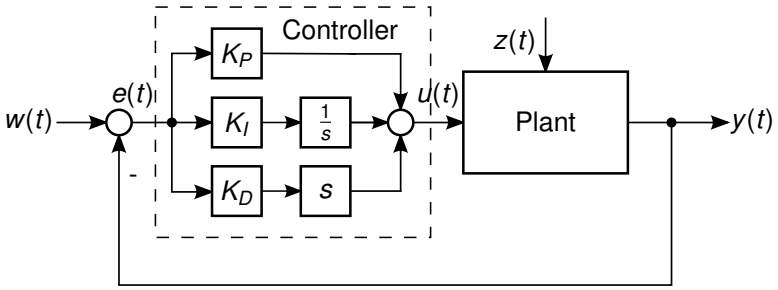


Figure 2.2.: A closed-loop PID controlled system

The control formula of the general PID controller can then be written as following:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t) \,. \tag{2.1}$$

After transforming equation 2.1 into a recurrence relation, it can for example be implemented efficiently on a micro controller. The remaining question is, how to choose the three parameters K_P, K_I and K_D to obtain reasonable controller outputs.

2.1.2. Parametrization of PID Controllers

Methods for estimating reasonable PID parameters are for example the Ziegler-Nichols method [29], the method by Chien, Hrones und Reswick [30] or the T_Σ-Method [31]. These methods require a measured step-response from the control system. The first two provide a look-up table for the PID controller, the T_Σ-Method uses an integration over the measured system response. Other methods are based on root locus analysis [32] or on formulating the tuning of the controller as an optimization problem [33, pp. 234-237]. In this context, genetic algorithms are often used to solve the optimization problem [34, 35]. An overview and comparison of various PID tuning methods is provided by [36].

While most of these methods only aim to identify static PID parameters which don't change during the process, there also exist several adaptive methods for PID parameter refinement [37]. In [38], a static feed-forward network is used, while [39] show a similar structure with neural networks with external dynamics. There also exist approaches using a neural network directly in a PID-like structure [40], for hybrid PID feed-forward control [41] or in conjunction with a neural plant model [42].

However, there only exist very few publications dealing with recurrent neural networks for PID tuning [43, 44], and those only use simple recurrent neural networks like e.g. Jordan networks (described in detail in the next section). This thesis therefore sets its focus especially on general dynamic neural networks (GDNNs) and how they can be used for both nonlinear system identification and online adaptation of PID controllers.

2.2. Artificial Neural Networks

This section will give a short overview about artificial neural networks (ANNs), starting with multilayer perceptrons (MLPs) and general feed-forward networks (FFNs). Then introducing simple recurrent neural networks (SRNNs) and finally general dynamic neural networks (GDNNs). Also, several training techniques are outlined and presented.

2.2.1. Multilayer Perceptrons and Feed-Forward Networks

The concept of the perceptron was introduced by F. Rosenblatt in 1958 [45, 46]. A neuron is modeled as a function $y(\cdot)$ that takes an arbitrary number of input arguments. Those inputs are then weighted, put through an input function (usually a summation) and then passed through a so-called activation function. The equation for a single neuron can then be expressed as follows:

$$y_i = \phi(\sum_{j=1}^{N} w_{i,j} u_j + b_i) \,. \tag{2.2}$$

Here y_i is the output of the i-th neuron and u_j is the j-th input of that neuron. N stands for the total amount of inputs of that neuron. Its output is then computed by calculating the weighted sum of its input, shifting it by some bias weight b_i and then passing this result through an activation function $\phi(\cdot)$. This relationship is illustrated in figure 2.3.

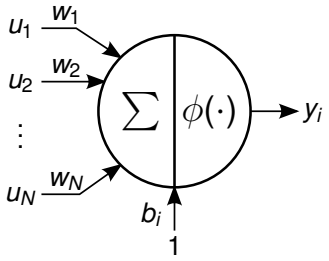


Figure 2.3.: Model of a single neuron

The activation function determines the behavior of the neuron, which has to be chosen by the designer of the artificial neural network. Table 2.1 shows some typical activation functions and their characteristics. By connecting the output of such a neuron to the input of another neuron, it is possible to build up a net of neurons. If this neurons are organized in a layer-wise structure, the net is called a multilayer perceptron (MLP). Figure 2.4 shows an example of such a net. For the sake of clarity, the bias connections are only shown in this net structure and omitted in further net illustrations.

A general MLP can have an arbitrary number of neurons and layers. Each connection between two neurons is weighted with some scalar value. The weights of the neural network can formally be expressed by its adjacency matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ and its bias weights

Table 2.1.: Typical activation functions

Name	**Function $\phi(x)$**	**Differentiable**	**Codomain**
Linear/Identity	x	Yes	$]\infty, \infty[$
Heaviside	$\begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$	No	$\{0, 1\}$
Piecewise Linear/Relu	$\begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$	No	$[0, \infty[$
Unipolar Sigmoid	$\frac{1}{1+e^{-x}}$	Yes	$[0, 1]$
Bipolar Sigmoid	$\frac{1-e^{-x}}{1+e^{-x}}$	Yes	$[-1, 1]$
Tangent Hyperbolic	$\tanh(x)$	Yes	$[-1, 1]$
Gaussian	$e^{-(x-c)^2/2\sigma^2}$	Yes	$[0, \frac{1}{\sigma\sqrt{2\pi e}}]$
Softmax	$\frac{e^{Q(a)/\tau}}{\sum_{b=1}^{n} e^{Q(b)/\tau}}$	Yes	$[0, 1]$

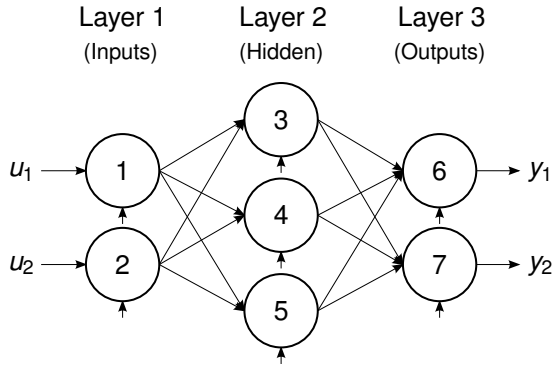


Figure 2.4.: An example MLP with 7 neurons and 3 layers

vector $\boldsymbol{b} \in \mathbb{R}^n$, where n is the number of neurons of the neural network. A non-zero entry at position i, j stands for a connection from neuron j to neuron i. The adjacency matrix and the bias weight vector of the network from figure 2.4 is shown in equation 2.3.

$$\boldsymbol{W}_{\text{MLP}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ w_{3,1} & w_{3,2} & 0 & 0 & 0 & 0 & 0 \\ w_{4,1} & w_{4,2} & 0 & 0 & 0 & 0 & 0 \\ w_{5,1} & w_{5,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{6,3} & w_{6,4} & w_{6,5} & 0 & 0 \\ 0 & 0 & w_{7,3} & w_{7,4} & w_{7,5} & 0 & 0 \end{bmatrix}, \boldsymbol{b}_{\text{MLP}} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}. \tag{2.3}$$

This kind of neural network has been widely applied for various tasks. This might relate to the fact that it has been shown that the output of a MLP is equivalent to the optimal

Bayesian discriminant function [47, 48] and a MLP with at least one hidden layer and sigmoidal activation functions is capable of representing any differentiable function [49, 50]. Example applications of MLPs include continuous speech recognition [51], damage detection in bridge structures [52] or medical decision support systems [53]. Further examples are the usage a MLP for emotion recognition [54], or for the pitch control problem of wind turbines [55]. A cost estimation for sheet metal parts is outlined by [56], while [57] uses a MLP to do bankruptcy forecasting. A summary of various applications, including finance, health and medicine, engineering and manufacturing, marketing and more, was published by [58]. New applications of MLPs include deep learning networks [59] or convolutional networks for several recognition tasks like handwritten text or face recognition [60, 61, 62]. Also, stacked autoencoders are a popular tool for denoising and feature extraction tasks [63, 64, 65].

The concept of a MLP can be generalized to networks with arbitrary forward connections. Such general feed-forward networks introduce shortcut connections, which are connections that skip one or more layers, as well as lateral connections which describe connections between neurons from the same layer. Such a network is still a feed-forward network, as it doesn't contain any cycles; to be precise that means, that there exists no path from an input to an output where a neuron is visited twice. Figure 2.5 shows an example general feed-forward network which has one shortcut and two lateral connections.

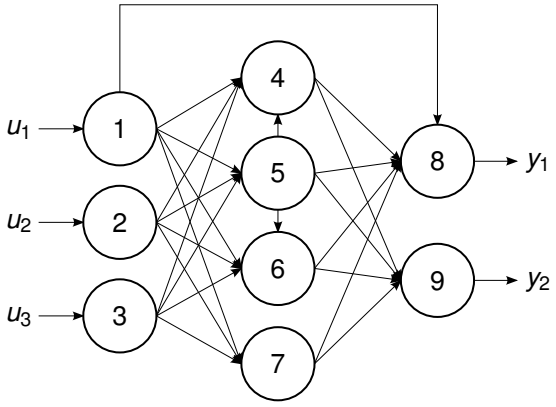


Figure 2.5.: Feed-forward network with shortcut (1 to 8) and lateral connections (5 to 4 and 6)

The effect of lateral connections in feed-forward networks was investigated by [66], while [67] used shortcut connections to improve the performance of a neural network. In [68] a global optimization method is applied on general feed-forward networks containing both lateral and shortcut connections. Although those publications show that feed-forward networks containing those types of connections show better performance than standard MLPs, most researchers exclude lateral and shortcut connections and focus on pure MLP networks. This might relate to the fact that pure MLPs can be computed by a simple concatenation of multiple matrix vector multiplications which simplifies the implementation of

such networks. If lateral and shortcut connections are allowed, a topological sorting has to be performed before it is possible to calculate the output of the network. Also the net structure isn't unique anymore, as neurons between shortcut connections can't be attached to a specific layer anymore. The next section will give a short introduction to simple recurrent neural networks, which not only allow arbitrary forward connections, but also recurrent backward connections.

2.2.2. Jordan and Elman Networks

A recurrent neural network is an artificial neural network with cyclic connections between neurons. An example would be a network with two neurons, both connected to each other. To compute the output of the first neuron, the output of the second neuron is required and vice versa. To break this algebraic loop a delay has to be introduced in one of the two connections. This means that any neural network with recurrent connections must have delays (and therefore an internal memory) to be computable. This dynamic behavior is the most important difference between static networks (like the MLP or feed-forward networks without delays) and dynamic networks (networks with internal delay units). This section will explain two of the very first recurrent network structures.

M. Jordan introduced in 1986 the so-called Jordan network, which is a recurrent neural network that has delayed connections between the output neurons and some "context" neurons, which serve as pseudo input neurons [69]. In figure 2.6 an example of such a network is shown. Here the two output neurons (number 7 and 8) are recurrently connected with two context neurons (number 3 and 4). The output of the context neurons is then again delayed and passed to the context neurons themselves. The z^{-1}-blocks serve as a delay by one time stamp. In [70], a Jordan network is used for software-reliability prediction. Other applications of Jordan networks include road traffic prediction [71] or cement strength prediction [72].

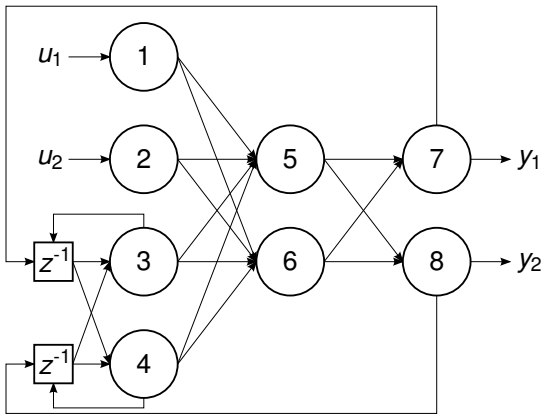


Figure 2.6.: A Jordan network with two context neurons (neurons 3 and 4)

After the Jordan network, in 1990 J. Elman presented the so-called Elman network [73]. It is structured in a similar way as the Jordan network and also uses context neurons. The difference is that for the recurrent connections not the output neurons were taken into account but the first layer of hidden neurons. Figure 2.7 shows an example of such a network: The output of the neurons from the first hidden layer is delayed by one time stamp and passed to the context neurons. Again, the context neurons are recurrently connected to themselves like in the Jordan network structure.

Elman networks were used for semantic word perception [74], predictive control of a six-degree-of-freedom robot [75], fault diagnosis of a hydraulic servo system [76], or epilepsy detection from an Electroencephalography (EEG) [77]. Other applications of Elman networks include the predictive control of reheated steam temperature [78], modeling of thermal deformation of machine tools [79] or prediction of internet traffic [80].

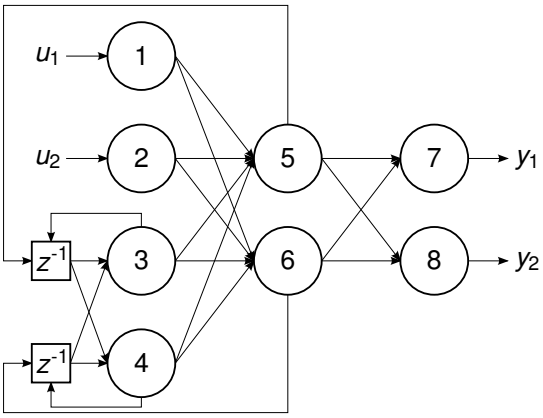


Figure 2.7.: An Elman network with two context neurons (neurons 3 and 4)

There are more special structures for recurrent neural networks, like for example Hopfield networks [81]. However, all those structures can be seen as a special case of the general dynamic neural network (GDNN), which allows arbitrary connections of arbitrary delays between all neurons. This net structure is presented in the next section.

2.2.3. General Dynamic Neural Networks

A general dynamic neural network (GDNN) is the most general form of a neural network and includes all the former described net topologies. The authors of [82, pp. 297-345] give a comprehensive overview about both static and dynamic neural networks, including the GDNN. While the dynamic neural networks discussed before only have recurrent connections between predefined layers or neurons, in a GDNN every neuron can be connected with any other neuron. Also, while Jordan and Elman networks only had a delay of one time stamp in their recurrent connections, a GDNN can have arbitrary delays in any connection. The only constraint is that the net has to be computable, which means that it must not

contain any algebraic loops, as explained in section 2.2.2. Figure 2.8 shows an example of a GDNN. The net is basically the feed-forward network from figure 2.5, but extended by some recurrent/delayed connections. Neuron 4 has a backwards connection to neuron 1, delayed by one time stamp, while neuron 7 is connected to neuron 3, delayed by two time stamps. Neuron 4 finally is also forward connected to neuron 8, one time directly with no delay and another time with a delay of one time stamp. It is important to note that there are two connections between neuron 4 and 8, resulting in two different, independent weights for this path.

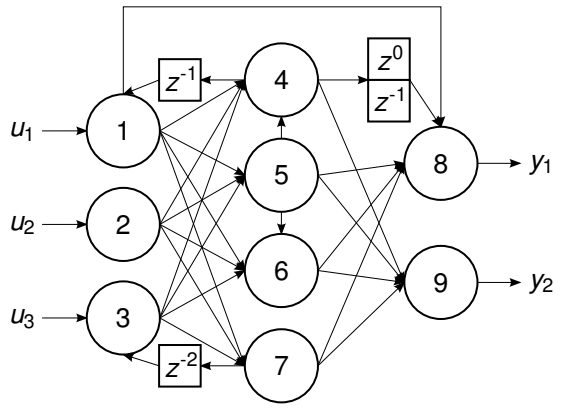


Figure 2.8.: A general dynamic neural network with forward and recurrent connections

There exist several publications dealing with the formal representation of a GDNN [83, 84]. This network structure has been applied for system identification of a two mass system [85]. In [86], some further application of GDNNs for nonlinear system identification is outlined. The authors of [87] present a genetic algorithm that evolves recurrent neural networks, while [88] use a recurrent neural network for wind speed and power forecasting. GDNNs have also been applied for handwriting recognition tasks [89] or for solving nonlinear convex programming problems [90].

Regardless of the topology a designer might choose, a neural network only provides a generic model that might be suitable for a given problem. This model has to be parametrized before it can be used efficiently. In a neural network structure, the parameters are represented by the weights of the inter-neuron connections and each neurons bias weight, which have to be adapted to the actual problem. This process of weight adjustment is called “training” of the network. The next section will introduce some machine learning techniques, including methods to perform this training for both static feed-forward networks and dynamic recurrent neural networks.

2.3. Machine Learning

The term machine learning describes artificial systems, that can learn from experience and generalize from this experience. This means especially that not all possible actions the system can potentially perform have to be implemented explicitly by the designer of the system, but that the system evolves with experience. Machine learning techniques are especially suitable for problems which cannot be solved by an explicit set of rules, like human voice or handwriting recognition. This section will introduce some of the most common machine learning techniques, including unsupervised, supervised and reinforcement learning.

2.3.1. Unsupervised Learning

Unsupervised learning describes the process of finding structure in unlabeled data. Usually no desired output can be defined for unsupervised learning problems, with the exception where the output should be equal the input to the learning system, like it is done with auto-encoders for example. Use cases are for example clustering [91] or dimensionality reduction [92]. Some popular models that are used for unsupervised learning are listed below.

- Hidden Markov models (HMMs) are often used for unsupervised learning tasks. HMMs are statistical models of systems for which it is assumed that the Markov property holds. This means that the state of the system only depends on its current input and not on previous inputs. Dynamical systems usually don't satisfy this condition, however there exist generalizations which introduce HMMs of n-th order, which take the last n time steps into account. An introduction to HMMs is given in [93].
- Another approach are Self-organizing maps (SOMs), or Kohonen maps [94]. A SOM is a type of artificial neural network, that is trained with unsupervised learning methods to produce a low dimensional output on a high dimensional input pattern.

A well known learning rule is Hebb's Rule for artificial neural networks [95]. The rule reads as follows:

$$\Delta w_i = \eta x_i y \,. \tag{2.4}$$

Here, Δw_i is the amount by which the weight of the neural connection i changes based on its input x_i and its response y. The scalar η stands for the learning rate of the method. This rule is often summarized by the statement "Cells that fire together, wire together", Hebbian learning therefore follows the idea that similar input patterns produce similar output patterns, which is biologically inspired. If an agent for example connects the color red with something dangerous, his neurons responsible for feeling danger will also fire when he sees something red that isn't related to a dangerous situation.

2.3.2. Supervised Learning

In contrast to unsupervised learning, the supervised learning approach operates on labeled data. This means that there is a set of desired outputs, which the agent should produce on a set of predefined inputs. The goal is to minimize the error between the desired output and the output the system actually produces for a given set of inputs [96, pp. 3-5]. One popular option to formalize this mapping of inputs to an output as an optimization problem is

$$\min_{\boldsymbol{p}} \left((\boldsymbol{y}(\boldsymbol{u}, \boldsymbol{p}) - \hat{\boldsymbol{y}})^T (\boldsymbol{y}(\boldsymbol{u}, \boldsymbol{p}) - \hat{\boldsymbol{y}}) \right). \tag{2.5}$$

Here, $\hat{\boldsymbol{y}}$ is the desired output the system should produce for a given input vector $\boldsymbol{u}$ and $\boldsymbol{y}$ is the real output of the system. This is achieved by adjusting the parameters of the system, represented by the vector $\boldsymbol{p}$ in such a way that the resulting derivation between the systems output and $\hat{\boldsymbol{y}}$ is minimal. The designer of the supervised learning system can then decide which kind of system he wants to use and which kind of optimization algorithm. Table 2.2 shows some popular methods for supervised learning and their advantages in their corresponding domain of application.

Table 2.2.: Strengths of supervised learning methods [96, pp. 14-15]

Method	**Strengths**
Support vector machines	Multidimensional, continuous features
Artificial neural networks	Multidimensional, continuous features
Decision trees	Discrete/categorical features
K-nearest neighbor	Transparent and intuitive
Gaussian processes	Easy to interpret, result probability
Naive Bayes	Works with small datasets too

Those methods are shortly described in the following enumeration. As artificial neural networks were already discussed in this work, refer to section 2.2 for a detailed overview.

- Support vector machines (SVM) are a popular tool for regression and classification tasks. F. Rosenblatt came up with the idea of fitting a linear hyperplane through nonlinear data by transforming the data to a higher dimensional space [45]. Based on this idea, the concept of support vector machines was introduced [97]. An extensive summary about SVM is provided in [98].
- Decision trees operate on a set of rules to solve decision problems. The goal of supervised learning is here to generate the set of rules in such a way, that the decision tree solves a training problem as desired and can be applied to new problems. A comprehensive overview about decision trees can be found in [99].

- K-nearest-neighbor (k-NN) is an algorithm that classifies objects by their distance to each other [100]. Here the k nearest objects are grouped into one class. As a distance property, usually the Euclidean distance is used, but in principle any measure can be used.
- Gaussian processes are multivariate Gaussian distributions over functions which can be used for regression or classification. Compared to other machine learning techniques, they not only return a function value for a regression or classification problem, but also a variance which indicates the confidence in this value. A detailed overview about Gaussian processes and their applications is provided in [101, pp. 1-30].
- Naive Bayes is a classifier based on the Bayes' theorem. The general assumption here is (what makes the approach "naive"), that every feature of an object only depends on its corresponding class. For an empirical study about Naive Bayes the reader might refer to [102].

The learning itself is then performed on one of these models by minimizing a cost function like the one from equation 2.5. For this purpose gradient descent or more advanced techniques can be used. For neural networks, the difficulty here is to calculate the gradient (respectively the Jacobian matrix) to apply this methods. For this purpose, algorithms like backpropagation [103, pp. 213-220] or backpropagation through time [104] can be used. Those methods are explained in detail in the next chapter.

2.3.3. Reinforcement Learning

Reinforcement learning describes a learning process of an agent within an environment. The agent performs actions in the environment, which lead to a new environment state and receives a reward for this action. The goal of the agent is then to maximize the expected, cumulative, discounted reward by selecting the "best" action for his current situation. This can be formally expressed by equation 2.6

$$\begin{aligned} R_t &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\,, \\ r &: S \times A \to \mathbb{R}\,, \\ \pi &: S \to A\,. \end{aligned} \tag{2.6}$$

Here, R_t is the cumulative reward at time step t and r is a reward function that evaluates an environmental state from the set of possible states S and an action from the set of possible actions A with a scalar real value, the reward. The function π is a policy function which is used by the agent to choose an action given an environmental state. The scalar value $\gamma \in [0, 1]$ finally describes a discount factor. For $\gamma < 1$ this factor weights immediate

rewards higher than the ones that are further in future. By setting the discount factor to 0, the agent only considers the current reward, the closer the discount factor gets to 1, the more "farsighted" it gets. The relationship between the agent and its environment is illustrated in figure 2.9. An introduction to reinforcement learning is provided in [105].

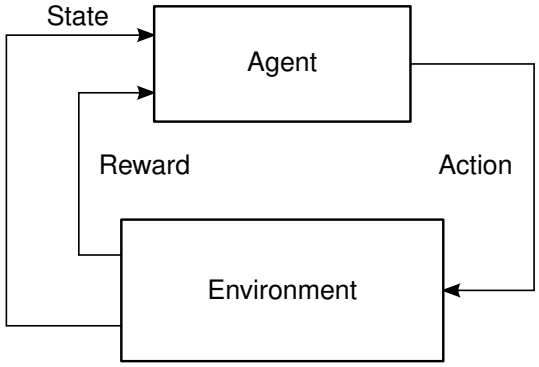


Figure 2.9.: A reinforcement learning agent interacting in its environment

One challenge in reinforcement learning lies in the choice of a suitable reward function. A possible approach to implement a reinforcement learning agent is temporal difference learning (TD-learning) [106]. Here, an estimate for the "value" of the current state $V(s)$ is derived by adding the difference between estimates at two different times, weighted by the step-size parameter α, which influences the rate of learning. This can be expressed by the iterative update formula

$$V(s) \leftarrow V(s) + \alpha \left(V(s') - V(s) \right) . \tag{2.7}$$

This method has been successfully applied for various tasks like playing Backgammon [107, 108], chess [109] or predictive autonomous robot control [110]. A similar approach was presented in [111], the Q-learning algorithm . Its iterative update formula is shown in equation 2.8

$$Q_{t+1}(s) \leftarrow Q_t(s) + \alpha_t \left(R_{t+1} + \gamma \max_a Q_{t+1} - Q_t \right) . \tag{2.8}$$

The Q-learning method has also been widely applied, e.g. for the traveling salesman problem [112], obstacle avoidance of autonomous mobile robots [113] or robotic soccer [114]. Several surveys about reinforcement learning have been published, comparing different methods on various applications to each other [115, 116, 117]. They conclude basically that reinforcement learning can be a powerful tool if applied to the right problem correctly, but that it is difficult to incorporate multiple (cooperative) reinforcement learning agents at the same time. As in this work only a single learning agent is necessary, and a cost function for the PID auto-tuning problem can be defined, both supervised and reinforcement learning seem to be suitable for the experiments of this work.

3. Training of Neural Networks and System Identification

3.1. Efficient Output Computation of a GDNN

One use case in control theory for artificial neural networks is system identification. Before a controller can be designed for a given process, usually a mathematical model of that process is required. If no such model is available (because its too difficult to derive one from the actual process, or the model is computationally too expensive), an artificial neural network can be used to learn the dynamics of the control system. Once the network has learned the dynamics of the process satisfactory to some quality criterion, it can be used in a closed-loop simulation. In this simulation, a PID auto-tuner can be learned by another neural network, which then can be applied to the real process. This chapter will present an algorithm to compute the output of an arbitrary GDNN as well as a novel method for generating optimal excitation signals to train such networks.

3.1.1. Formal Representation and Implementation

For a GDNN, one adjacency matrix is not sufficient, as each delay level can have its own connections. Those weights can be summarized in the 3-dimensional weight matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n \times m}$ as illustrated in figure 3.1. Here n stands for the number of neurons and m is the maximum number of delays z^{-m} in the network.

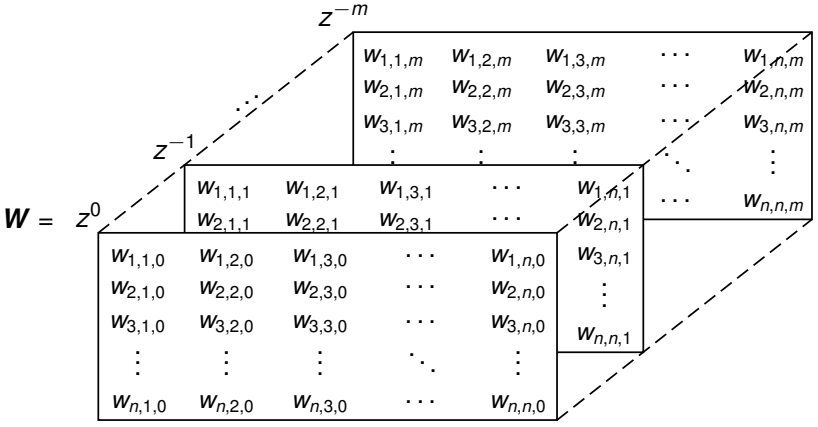


Figure 3.1.: A stack of adjacency matrices of a GDNN with n neurons and m delays

While this representation is convenient for describing connections in the network, from an implementation point of view such a matrix stack would be cost intensive, both in terms of memory usage and runtime. As most practical GDDNs are not fully connected (this means that not every neuron is connected to each other) and especially only a few connections incorporate delays, the adjacency matrices are sparse. Therefore in the implementation of the neural network toolbox, a two dimensional adjacency matrix was used. In the entries of the matrix a dynamic stack for each delay line is stored; so if a path between two neurons contains only one connection (for example one direct connection without any delay), the stack has only one entry. For more connections in one path (multiple time delays) the stack contains the corresponding number of weights.

The GDNN was implemented as a class template in C++11. For the internal memory of the network, every single neuron contains an own queue with the size of the maximum time delay of all outputs of this neuron. After one time step, each queue is shifted by one. Algorithm 1 shows how the output calculation is performed in the GDNN.

Algorithm 1 : Calculate GDNN Output

Input : Data Vector $\boldsymbol{u}$, Adjacency Matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n \times m}$, Bias Weights $\boldsymbol{b} \in \mathbb{R}^{n}$
Output : Data Vector $\boldsymbol{y}$

1 **if** *Sort required* **then**
2 Perform topological sorting using algorithm 2
3 **foreach** *Neuron* N_i *in sorted order* **do**
4 **for** $j = 1$ *to* n **do**
5 **if** N_i *is input* **then**
6 Assign u_j to the output of N_i
7 **if** N_i *is connected to* N_j **then**
8 **if** *connection is instant* **then**
9 $y_i \leftarrow y_i + W_{i,j,1} y_j$
10 **if** *connection has delays* **then**
11 **foreach** *delay d in connection* **do**
12 $y_i \leftarrow y_i + W_{i,j,d} q_{i,d}$ `// q is the memory queue of neuron i`
13 $y_i \leftarrow \phi_i(y_i + b_i)$
14 Shift neuron memory queues by 1 and store $\boldsymbol{y}$
15 **return** $\boldsymbol{y}$

With this algorithm it is possible to compute the output of an arbitrary GDNN. Line 2 is important here, because there are several operations which can be performed on the network that require a reordering of its computation order. Such operations are for example the declaration of new inputs, outputs or inter-neuron connections. The next section will explain this topological sorting in more detail.

3.1.2. Topological Sorting

Before the output of an arbitrary neural network can be computed, it is necessary to perform a topological sorting. Because of shortcut and/or lateral connections as well as recurrent backwards connections, the computation of a neuron can't be performed layer-wise. Looking for example at the feed-forward network in figure 2.5 in chapter 2, it is not possible to start the calculation of the hidden layer with neuron 3, as it requires the output from neuron 4. This problem can be generically solved by iterating recursively over the adjacency matrices of the network. The algorithms 2 and 3 describe the sorting process.

Algorithm 2 : Topological Sort

Input : Adjacency Matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n \times m}$
Output : Sorted Neuron indices $\boldsymbol{I}$

$\boldsymbol{s} \leftarrow \emptyset, \boldsymbol{I} \leftarrow \emptyset$
for *i = 1 to n* **do**
    $\boldsymbol{I} \leftarrow \{\boldsymbol{I}, \mathit{ParseLine}(\boldsymbol{W}, \boldsymbol{s}, \boldsymbol{I}, i)\}$ `// Parse line i of W using algorithm 3`
    if $\boldsymbol{s} = \emptyset$ **then**
        $i \leftarrow \min_x(x \in \mathbb{N}^0 | x \notin \boldsymbol{s})$
    else
        $i \leftarrow s_0$
        $\boldsymbol{s} \leftarrow \emptyset$
return $\boldsymbol{I}$

Algorithm 3 : Parse Adjacency Matrix line

Input : Adjacency Matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n \times m}$, Stack $\boldsymbol{s}$, Index set $\boldsymbol{I}$, Matrix line λ
Output : Next Neuron Index λ_{Next}

foreach *Neuron* N_i **do**
    if N_i *is connected instant to* $N_\lambda \wedge \ (\lambda \notin \boldsymbol{I})$ **then**
        if $i \in \boldsymbol{s}$ **then**
            Error: Network contains Algebraic Loop
        else
            $\boldsymbol{s} \leftarrow \{\boldsymbol{s}, \lambda\}$
            return $\mathit{ParseLine}(\boldsymbol{W}, \boldsymbol{s}, \boldsymbol{I}, i)\}$ `// Recursive call`
return λ

After the sorting is done, a flag can be set such that algorithm 1 doesn't need to perform a sort on every output calculation. This flag can be reset if (and only if) an operation is performed on the network that requires a reordering of the output computation to maximize performance of the GDNN, as the net structure usually remains unchanged during runtime.

3.2. Excitation Signals

For both the identification task of a dynamical system and the training of a neural PID tuner it is important to choose a reasonable excitation signal. As the excitation signal is the only way to interact with the process and collect information about it, the input signal should extract as much information as possible from the control system but at the same time not violate its physical constraints. A Dirac impulse for example might lead directly to the transfer function of a system, but is difficult to generate in real world scenarios. The following section will introduce two suitable excitation signals for system identification. Also a novel algorithm to generate those signals with a predefined hold-time is outlined.

3.2.1. Pseudo-Random Binary Signal

There are several convenient excitation signals available for system identification. O. Nelles describes some popular excitation signals for this task and lists their corresponding advantages and disadvantages [118, pp. 459-465]. Table 3.1 shows this comparison, including the PRBS (pseudo-random binary signal) which is further discussed in this section.

Table 3.1.: Comparison of excitation signals [118, pp. 459-461]

Signal	Suitable	Characteristics
Constant	x	No dynamics are excited
Impulse	x	Gain estimation very inaccurate
Step	✓	Low frequencies emphasized
Rectangular	✓	High frequencies emphasized
PRBS	✓	Excites all frequencies equally well

Although step, rectangular and PRBS are suitable for the identification task, only the latter excites all frequencies equally well. Therefore, the PRBS was chosen as excitation signal. The signal is named “pseudo-random” because it approximates the spectrum and statistical properties of white noise (zero mean and variance of one). The signal generation however is deterministic and can be done efficient with linear feedback shift registers [119, pp. 164-174]. These shift registers consist of several states represented as flip-flops and a feedback through some XOR block to the first flip-flop. Figure 3.2 shows such a linear feedback shift register with four states. A clock shifts these states periodically through the register while the state of the last flip-flop is taken as an output. Once the initial state is reached again, the shift register produces the same output periodically. In principle any state could be used to initialize the shift register, except for the all-zero state, because the shift register would then only be able to produce zeros.

An example output cycle for the four-state shift register from figure 3.2 with initial state 1111 is shown in table 3.2. After the first 15 cycles the internal state of the linear feedback shift register becomes its initial state 1111 again and starts from the beginning.

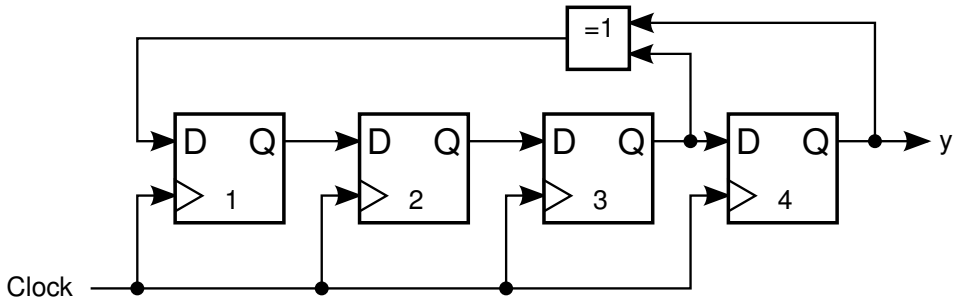


Figure 3.2.: A linear feedback shift register with four states

Table 3.2.: One period in the linear feedback shift register from figure 3.2

Clock	State	Output y	Clock	State	Output y
0	1111	1	8	1100	0
1	0111	1	9	0110	0
2	0011	1	10	1011	1
3	0001	1	11	0101	1
4	1000	0	12	1010	0
5	0100	0	13	1101	1
6	0010	0	14	1110	0
7	1001	1	(15	1111	1)

By defining some bandwidth during which the output doesn't change, it is possible to stretch y to a desired length. The feedback law for a given linear feedback shift register describes which outputs are XOR-ed and feedback to the input of the shift register. This is important, because only with certain feedback laws, a linear feedback shift register can produce a maximum length sequence. These feedback laws are linked to so-called primitive polynomials. A feedback law is described by the non-zero coefficients of such a polynomial. For example the above shift register from figure 3.2 has a feedback law corresponding to the primitive polynomial $x^4 + x^3 + 1$. This means that the output of the flip-flops 3 and 4 is feedback. Some more primitive polynomials for different numbers of internal states and their maximum period are shown in table 3.3. As this list already suggests, the maximum period such a register can generate is

$$p_{\max} = 2^n - 1\,, \tag{3.1}$$

where n is the number of states of the shift register. So by taking the output of the shift register every $k \in \mathbb{N}$ times per clock cycle, it is possible to scale the length of these periods by an integer multiple. The question is now, how to choose the number of states and the bandwidth $\frac{1}{k}$ to generate an optimal PRBS of predefined length for a given control system.

As already discussed, the PRBS excites all frequencies equally well. However, to get a

Table 3.3.: Primitive Polynomials for linear feedback shift registers

States	Polynomial	Period
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511

reliable estimate for the system gain, the maximum hold time of the PRBS should be long enough that the system can reach a stationary state. The maximum hold sequence $s_{\max}$ of a PRBS is defined as the longest output sequence of 1's it can produce and is equal to the number of states n of the shift register. So if for example, a $s_{\max}$ of 100 time steps is required to reach a maximum hold time of 100 seconds (assuming 1 second sample time), one could choose a linear feedback shift register with 100 states and reach a $s_{\max}$ of 100. However, the resulting signal would have the length $2^{100} - 1$ which is to large to be stored in RAM, even on modern computers. Usually a fixed length for the excitation signal is desirable, in this example here 1000. An alternative approach would be to choose a relatively small number of states and increase the scaling k such that the $s_{\max}$ is reached. For example a shift register with four states can generate a $s_{\max}$ of 4, scaled with a $k = 25$ it would produce a $s_{\max}$ of 100 with a total signal length of 375. This is more reasonable than the former signal, but still doesn't have the desired length of 1000. The signal itself can now be replicated and concatenated to reach this length, however the problem is then, that the signal contains many redundancies. The signal would basically be repeated (nearly) three times without adding any new information. The question is, which combination of states n and scaling factor k leads to a signal with minimal repetitions or cutoffs to reach a desired length l and maximum hold sequence $s_{\max}$ for the PRBS. This can be expressed by the following optimization problem

$$\begin{aligned} n_{\text{opt}} &= \arg\min_n \left| (2^n - 1) \left\lfloor \frac{s_{\max}}{n} + \frac{1}{2} \right\rfloor - l \right| \quad \text{with } n, l, s_{\max} \in \mathbb{N}\,, \\ k_{\text{opt}} &= \frac{s_{\max}}{n_{\text{opt}}}\,. \end{aligned} \tag{3.2}$$

Abusing the fact that equation 3.2 only has one global optimum and grows exponentially in n, the optimization can be done efficiently by increasing n iteratively by 1. For the above example of $l = 1000$ and $s_{\max} = 100$, this process is illustrated in table 3.4, the optimal result is marked bold.

Table 3.4.: Example evaluation of the optimization problem for from equation 3.2

n	$2^n - 1$	k = $\lfloor \frac{s_{max}}{n} + \frac{1}{2} \rfloor$	$s_{max} = nk$	$k(2^n - 1)$	**error**
1	1	100	100	100	900
2	3	50	100	150	850
3	7	33	99	231	769
4	15	25	100	375	625
5	31	20	100	620	380
6	**63**	**17**	**102**	**1071**	**71**
7	127	14	98	1778	778

As a result the optimal PRBS for the given configuration of $l = 1000$ and $s_{max} = 100$ can be generated with a linear feedback shift register with $n_{opt} = 6$ internal states and a scaling factor $k_{opt} = 17$. The resulting PRBS has a length of 1071 and a $s_{max} = 102$ which is the optimal compromise for this setup. Taking the first 1000 values of this signal results in a PRBS with the desired length of 1000 with a minimal cutoff of 71 samples and a maximal hold sequence of 102. The generated PRBS with this method is shown in figure 3.3.

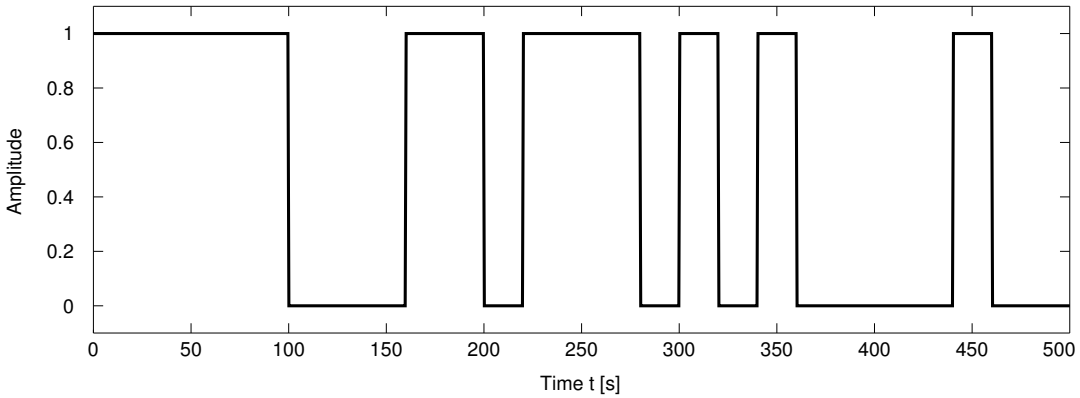


Figure 3.3.: A PRBS with a maximum hold time of 100 seconds

The method for computing the optimal PRBS with respect to the criterion of equation 3.2 is summarized in the algorithms 4 and 5. These algorithms imply that a method for computing the output of arbitrary linear feedback shift registers is available. This can be implemented efficiently by storing the first 64 primitive polynomials. As a polynomial with 64 internal states can generate a vector of length $2^{64} - 1$, which equals the maximum array size in 64 bit computer architectures, storing only those polynomials is sufficient.

Using these algorithms it is possible to generate PRBS with a predefined maximum hold time and length. The next section will introduce amplitude modulated PRBS for nonlinear system identification, as the presented PRBS is only suitable for linear control systems.

Algorithm 4 : Find optimal grade and scale

Input : Desired signal length l, maximum hold sequence $s_{\max}$
Output : Optimal grade n_{opt}, optimal scaling factor k_{opt}

$l_{\max} \leftarrow 1, n \leftarrow 1, n_{\text{opt}} \leftarrow 1, l_{\text{best}} \leftarrow \infty$
while $l_{max} \leq n$ **do**
  $l_{\max} \leftarrow n^2 - 1$
  $l_{\text{current}} \leftarrow l_{\max} \left\lfloor \frac{s_{\max}}{n} + \frac{1}{2} \right\rfloor$ `// The` $\lfloor \cdot \rfloor$ `denotes a floor function`
  if $|l_{current} - l| < l_{best}$ **then**
    $l_{\text{best}} \leftarrow l_{\text{current}}$
    $n_{\text{opt}} \leftarrow n$
  $n \leftarrow n + 1$
$k_{\text{opt}} \leftarrow \frac{s_{\max}}{n_{opt}}$
return n_{opt}, k_{opt}

Algorithm 5 : Generate optimal PRBS

Input : Desired signal length l, maximum hold sequence $s_{\max}$
Output : Optimal PRBS $\boldsymbol{y}$

$\boldsymbol{y} \leftarrow \emptyset$
$n_{\text{opt}}, k_{\text{opt}} \leftarrow$ *FindOptimalGradeAndScale*$(l, s_{\max})$ `// Using algorithm 4`
Load primitive polynomial p for n_{opt} from a look-up table `// Like table 3.3`
Initialize shift register with n_{opt}, k_{opt} and initial state $\boldsymbol{m} = \mathbf{1}$
while $\dim(\boldsymbol{y}) < l$ **do**
  $\boldsymbol{y} \leftarrow \{\boldsymbol{y}, m_{n_{\text{opt}}}\}$
  $x \leftarrow 0$
  foreach *Coefficient c of p* **do**
    $x \leftarrow x \oplus m_c$ `// The` $\oplus$ `denotes an xor operation`
  Rotate register state $\boldsymbol{m}$ by 1
  $m_0 \leftarrow x$
return $\boldsymbol{y}$

3.2.2. Amplitude Modulated PRBS

As already indicated in the previous section, the standard PRBS is not suitable for nonlinear system identification as its amplitude only varies between two different values. However, to encode the system nonlinear dynamics of the system, it must be excited over the whole range of possible input amplitudes. Therefore APRBS (amplitude modulated PRBS) are used for this purpose [119, pp. 172-175]. In this work, the amplitudes were uniformly distributed between the desired minimum and maximum of the APRBS. The step size α be-

tween each amplitude steps is determined by dividing the amplitude range by the amount of intervals of the PRBS. Figure 3.4 shows the transformation of a PRBS to an APRBS.

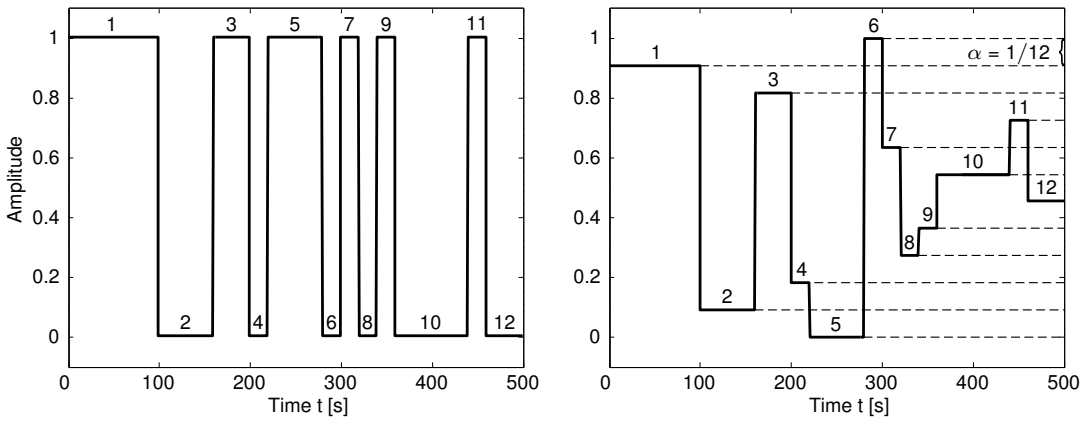


Figure 3.4.: Transformation of a PRBS (left) to an APRBS (right) based on the amount of intervals

This conversion is shown in detail in algorithm 6, which was used in this work to generate amplitude modulated PRBS.

Algorithm 6 : Generate optimal APRBS

Input : Desired signal length l, maximum hold sequence $s_{\max}$, amplitudes $y_{\min}, y_{\max}$
Output : Optimal APRBS $\boldsymbol{y}$

$\boldsymbol{y} \leftarrow$ *GenerateOptimalPRBS*$(l, s_{\max})$ `// Using algorithm 5`
$n_{\text{int}} \leftarrow$ Count intervals of $\boldsymbol{y}$
$\alpha \leftarrow \frac{y_{\max} - y_{\min}}{n_{\text{int}} - 1}$ `// Amplitude step size`
for *Sample* $i \in [0, n_{int} - 1]$ *without replacement* **do**
    $x_i \leftarrow i\alpha$
foreach *Interval* $i \in \boldsymbol{y}$ **do**
    $i \leftarrow x_i$ `// Set whole interval i to` x_i
return $\boldsymbol{y}$

Using these algorithms it is possible to generate efficient training signals for neural networks for both system identification and neural PID tuning.

3.3. Jacobian Calculations for General Feedback Systems

The following section will introduce two methods for calculating the Jacobian matrix of a GDNN, first the standard approach of temporal unfolding described in [104] and second a numerical approximation.

3.3.1. Analytic Computation by Temporal Unfolding

A given GDNN with internal time delays can be unfolded into a feed-forward network. This is achieved by replicating the network k times for $k + 1$ time steps and connecting the delayed outputs to the corresponding preceding network. Figure 3.5 shows an example GDNN with one delayed recurrent connection.

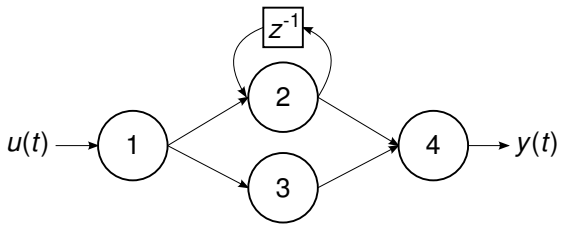


Figure 3.5.: An example GDNN with one recurrent connection

In figure 3.6 the temporal unfolding of the network form figure 3.5 is illustrated for $k + 1$ timesteps. For this network, the Jacobian matrix can be calculated using the standard backpropagation algorithm [103, pp. 213-220].

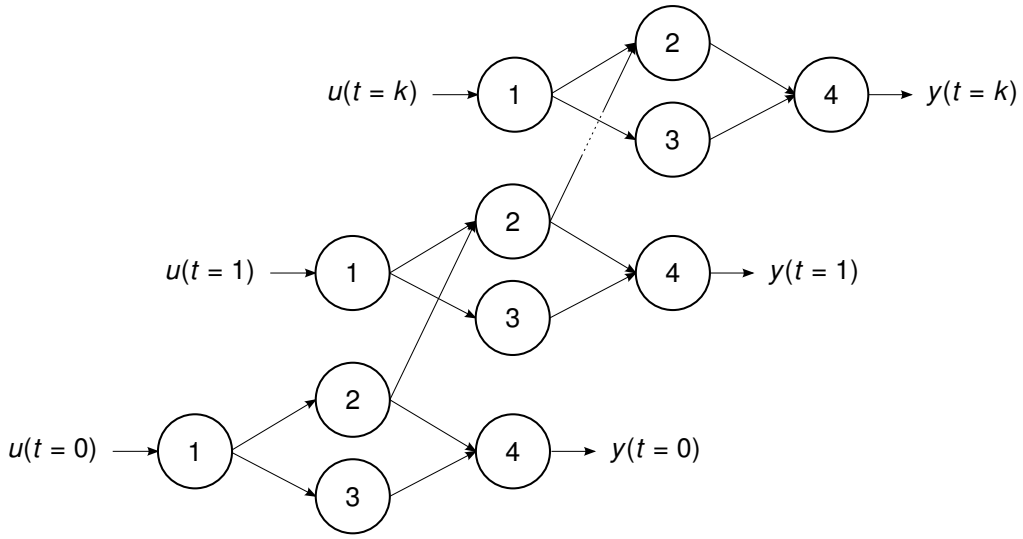


Figure 3.6.: The network of figure 3.5 unfolded for $k + 1$ time steps

Using this relationship, the Jacobian matrix of an arbitrary GDNN with n input neurons, m output neurons, l time steps and k weighted connections can be expressed as

$$\boldsymbol{J}_{\boldsymbol{\omega}}\left(\boldsymbol{y}(\boldsymbol{u},\boldsymbol{\omega})\right) = \begin{bmatrix} \frac{\partial}{\partial\omega_0} y_0(\boldsymbol{u}_0,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_0(\boldsymbol{u}_0,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_0(\boldsymbol{u}_0,\boldsymbol{\omega}) \\ \frac{\partial}{\partial\omega_0} y_0(\boldsymbol{u}_1,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_0(\boldsymbol{u}_1,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_0(\boldsymbol{u}_1,\boldsymbol{\omega}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\omega_0} y_0(\boldsymbol{u}_n,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_0(\boldsymbol{u}_n,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_0(\boldsymbol{u}_n,\boldsymbol{\omega}) \\ \frac{\partial}{\partial\omega_0} y_1(\boldsymbol{u}_0,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_1(\boldsymbol{u}_0,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_1(\boldsymbol{u}_0,\boldsymbol{\omega}) \\ \frac{\partial}{\partial\omega_0} y_1(\boldsymbol{u}_1,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_1(\boldsymbol{u}_1,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_1(\boldsymbol{u}_1,\boldsymbol{\omega}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\omega_0} y_1(\boldsymbol{u}_n,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_1(\boldsymbol{u}_n,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_1(\boldsymbol{u}_n,\boldsymbol{\omega}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\omega_0} y_m(\boldsymbol{u}_0,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_m(\boldsymbol{u}_0,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_m(\boldsymbol{u}_0,\boldsymbol{\omega}) \\ \frac{\partial}{\partial\omega_0} y_m(\boldsymbol{u}_1,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_m(\boldsymbol{u}_1,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_m(\boldsymbol{u}_1,\boldsymbol{\omega}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\omega_0} y_m(\boldsymbol{u}_n,\boldsymbol{\omega}) & \frac{\partial}{\partial\omega_1} y_m(\boldsymbol{u}_n,\boldsymbol{\omega}) & \cdots & \frac{\partial}{\partial\omega_k} y_m(\boldsymbol{u}_n,\boldsymbol{\omega}) \end{bmatrix} \in \mathbb{R}^{lm\times k}. \quad (3.3)$$

Here $\boldsymbol{u}_i \in \mathbb{R}^n$ is the input vector at timestep i, $\boldsymbol{\omega} \in \mathbb{R}^k$ is the vector of weights which describes the network topology and y_i is the i-th output of $\boldsymbol{y}(\boldsymbol{u},\boldsymbol{\omega}) \in \mathbb{R}^m$. This Jacobian matrix can then be used in combination with an optimization algorithm like the Levenberg-Marquardt algorithm to adjust the weights of the network [120]. However, as the Jacobian matrix should be computable for an arbitrary feedback system, a numerical approximation can be used. The next section shows how an estimate for $\boldsymbol{J}_{\boldsymbol{\omega}}$ can be computed for such a system.

3.3.2. Numerical Computation by Forward Propagation

Instead of computing $\boldsymbol{J}_{\boldsymbol{\omega}}$ explicitly using equation 3.3, an estimate $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}} \approx \boldsymbol{J}_{\boldsymbol{\omega}}$ can be calculated numerically. This can be done by caching the internal state of the dynamical system for which the Jacobian matrix should be computed. Then the Jacobian matrix can be estimated with a finite backwards difference

$$\hat{\boldsymbol{J}}_{\boldsymbol{\omega}}\left(\boldsymbol{y}(\boldsymbol{u},\boldsymbol{\omega})\right) = \begin{bmatrix} \boldsymbol{j}_0 \\ \boldsymbol{j}_1 \\ \vdots \\ \boldsymbol{j}_k \end{bmatrix}^T, \ \boldsymbol{j}_i = \frac{\boldsymbol{y}(\boldsymbol{u},\boldsymbol{\omega}) - \boldsymbol{y}(\boldsymbol{u},\boldsymbol{\omega} - \boldsymbol{h}\varepsilon(\omega_i))}{\varepsilon(\omega_i)}, \ h_p = \begin{cases} 1, & p = i \\ 0, & p \neq i \end{cases}. \quad (3.4)$$

Here, $\varepsilon(\omega_i)$ is a step width for calculating the backwards difference, $\boldsymbol{j}_i$ is the i-th column of $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}}$ and $\boldsymbol{h}$ is a vector that selects the correct ε step width in equation 3.4. This step width should be small enough to deliver a reasonable approximation for the first derivative and

large enough to minimize numerical errors. As the size of the weights stored in $\boldsymbol{\omega}$ can vary in a very broad range, a reasonable ε is computed according to [121, pp. 192-196] by

$$\varepsilon(\omega_i) = \max\left(1, |\omega_i|\right)\sqrt{\varepsilon_{\min}}\,. \tag{3.5}$$

Here $\varepsilon_{\min}$ is the machine epsilon of the implementation data type (typically of double precision). The computation of $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}}$ can then be performed using algorithm 7.

Algorithm 7 : Compute Jacobian Numerically

Input : Dynamical System $\boldsymbol{y}(\boldsymbol{u}, \boldsymbol{\omega}) \in \mathbb{R}^m$, inputs $\boldsymbol{u} \in \mathbb{R}^{l \times n}$, weights $\boldsymbol{\omega} \in \mathbb{R}^k$
Output : Estimate of Jacobian matrix $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}} \in \mathbb{R}^{lm \times k}$

foreach *Weight* $\omega_i \in \boldsymbol{w}$ **do**
  $\boldsymbol{y}_{\text{tmp},1} \leftarrow \boldsymbol{y}$, $\boldsymbol{y}_{\text{tmp},2} \leftarrow \boldsymbol{y}$ `// Cache internal state of y`
  $\varepsilon \leftarrow \max\left(1, |\omega_i|\right)\sqrt{\varepsilon_{\min}}$
  for $j = 1$ *to* l **do**
    $\boldsymbol{v} \leftarrow \boldsymbol{y}_{\text{tmp},1}(\boldsymbol{u}_j, \boldsymbol{\omega})$, $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{y}_{\text{tmp},2}(\boldsymbol{u}_j, \boldsymbol{\omega} - \boldsymbol{h}\varepsilon)$
    for $q = 1$ *to* m **do**
      $\hat{\boldsymbol{J}}_{\omega(jm+q,i)} \leftarrow \frac{\boldsymbol{v}_k - \hat{\boldsymbol{v}}_k}{\varepsilon}$ `// Backward difference`
return $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}}$

The training using a modified Levenberg-Marquard method is shown in algorithm 8.

Algorithm 8 : Train Network in Dynamical Feedback System

Input : System $\boldsymbol{y}(\boldsymbol{u}, \boldsymbol{\omega}) \in \mathbb{R}^m$, inputs $\boldsymbol{u} \in \mathbb{R}^{l \times n}$, desired outputs $\hat{\boldsymbol{y}} \in \mathbb{R}^{l \times m}$
Output : Optimal weights $\boldsymbol{\omega}_{\text{opt}}$

while *Not converged and trials left* **do**
  Initialize $\boldsymbol{\omega}$ with random values
  $\boldsymbol{u}_{\text{tmp}} \leftarrow [\boldsymbol{u}_1, ...\boldsymbol{u}_x]$, $\hat{\boldsymbol{y}}_{\text{tmp}} \leftarrow [\hat{\boldsymbol{y}}_1, ...\hat{\boldsymbol{y}}_x]$ `// Take first` x `values of u and y`
  while *Not converged* **do**
    $\hat{\boldsymbol{J}}_{\boldsymbol{\omega}} \leftarrow$ *ComputeJacobian*$(\boldsymbol{y}, \boldsymbol{u}_{\text{tmp}}, \boldsymbol{\omega})$ `// Using algorithm 7`
    Solve $(\hat{\boldsymbol{J}}_{\boldsymbol{\omega}}^T \hat{\boldsymbol{J}}_{\boldsymbol{\omega}} + \lambda \boldsymbol{I})\sigma = \hat{\boldsymbol{J}}_{\boldsymbol{\omega}}^T\left(\boldsymbol{y}(\boldsymbol{u}_{\text{tmp}}, \boldsymbol{\omega}) - \hat{\boldsymbol{y}}_{\text{tmp}}\right)$
    if $\sum\left(\boldsymbol{y}(\boldsymbol{u}_{tmp}, \boldsymbol{\omega} + \sigma) - \hat{\boldsymbol{y}}_{tmp}\right)^2 < \sum\left(\boldsymbol{y}(\boldsymbol{u}_{tmp}, \boldsymbol{\omega}) - \hat{\boldsymbol{y}}_{tmp}\right)^2$ **then**
      $\boldsymbol{\omega}_{\text{opt},i} \leftarrow \boldsymbol{\omega} + \sigma$, $\lambda \leftarrow \lambda/2$
    else
      $\lambda \leftarrow 2\lambda$
  If $\boldsymbol{\omega}_{\text{opt},i}$ is better than $\boldsymbol{\omega}_{\text{opt},i-1}$ increase x until $x = l$
return $\boldsymbol{\omega}_{opt,i}$ with minimal error

4. Auto-tuning of PID Controllers

4.1. General Block Structures

In this chapter, the general control structure of the PID auto-tuner embedded into the control loop will be presented. Figure 4.1 shows this structure including the control plant, the PID controller and the neural network that is used to tune this controller during runtime. The bold arrows indicate vector-valued, the thin arrows scalar-valued signals.

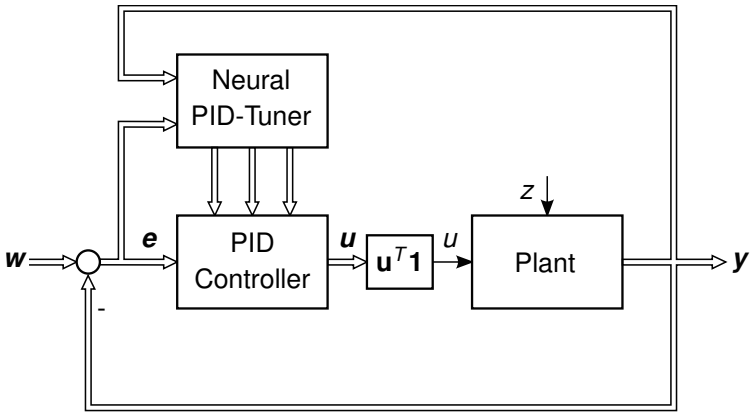


Figure 4.1.: The PID auto-tuning structure using a GDNN in the control loop

This structure can be applied to general SIMO (single-input, multiple-output) control plants with input u and disturbance z. For such a plant with k outputs, k PID controllers and neural PID tuners have to be stacked in parallel. Each PID tuner gets the current system state $\boldsymbol{y} \in \mathbb{R}^k$ and the control error $\boldsymbol{e} \in \mathbb{R}^k$. The PID tuners yield $3k$ parameters to the PID controllers, which are summed up and passed as a scalar value back to the control plant. The optimization task is then to adjust the weights ω of the neural network such that the difference between the setpoint $\boldsymbol{w}$ and the output $\boldsymbol{y}$ is minimal. If this is the case, the control system follows the inputs and behaves like desired. The optimization problem is then defined as in equation 2.5 and solved using algorithm 8. The feedback system is excited using the APRBS generated with algorithm 6 to cover the majority of the relevant state space of the feedback system and therefore provide a rich training set for the PID tuner. The control system itself can be either represented with an explicit mathematical model or with another neural network which represents this system. The next section will introduce those two different approaches as knowledge-free and model based approach.

4.1.1. Knowledge-Free Approach

One method to apply the neural PID auto-tuner is to train it on another neural network which represents the actual control system. This method can be applied if no mathematical model of the control system exists, but it can be excited with input signals to measure its response. The learning then happens in two stages, which are shown in figure 4.2.

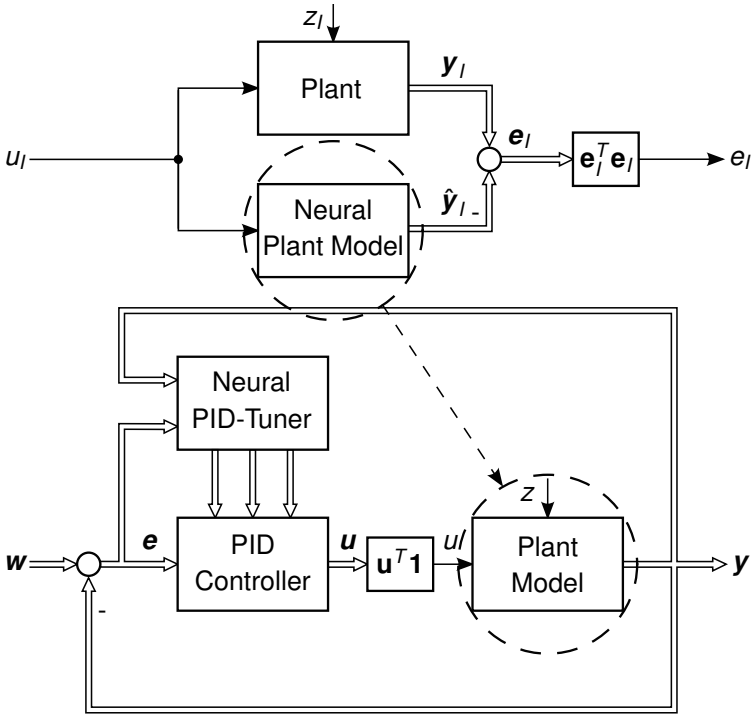


Figure 4.2.: The PID auto-tuning structure combined with system identification

First a model of the actual plant is learned by minimizing the difference between $\boldsymbol{y}_l$ and $\hat{\boldsymbol{y}}_l$. This learning can be either performed "offline" (which means that the learning is done after measuring the whole system response $\boldsymbol{y}_l$) or "online" (during the measurement). For the online learning, only the last k values of the system output are taken into account. Those values are then segmented into a training and a validation set on which the network is trained. After training the model remains valid for a predefined prediction horizon, then the identification is repeated. A more detailed explanation about online and offline identification is provided in [118, pp. 81-83]. Once the model is learned, this neural plant model can be embedded in the feedback structure. Then as a second step, the neural PID tuners can be trained by minimizing the difference between $\boldsymbol{w}$ and $\boldsymbol{y}$. Those steps happen either in serial (offline identification) or in parallel (online identification). Finally, when the stack of neural PID tuner produces reasonable K_P, K_I and K_D for the given plant model, it can be embedded and verified for the real control system. Since the plant is modeled

by another neural network, temporal unfolding as described in section 3.3 can be applied here for Jacobian calculations.

4.1.2. Model Based Approach

The advantage of the described method is, that it is possible to integrate previous knowledge about the control system. If a mathematical model of the control system already exists, the first step of the training process described in section 4.1.1 can be skipped and instead of the neural plant model the model of the system can be used directly. This is especially important for unstable systems, which can be difficult to identify by a neural network without violating the physical constraints of the system. The model based approach starts directly at step two shown in figure 4.2 and trains the neural PID tuner based on the output of the model of the control system.

4.2. Validation and Prefiltering

It is necessary for both the neural plant model and the neural PID tuner to exhibit good generalization capabilities from the learned training set. The neural PID tuner for example is only trained for a specific APRBS input sequence, but should also produce reasonable PID parameters for different, unknown inputs. To achieve this, a validation set was used in this work. This means that the neural network is trained on a specific input signal, but the error is evaluated on both this training signal and another, so-called validation signal which is different to the training signal. Then the parameter set with a minimal error on both sets is used, following the idea of early stopping methods [122].

Additionally, it turned out that the training could be improved significantly if the input signal was low-pass filtered before the training. Therefore a upstream first order Butterworth filter with a gain of 1 was used with all GDNNs in this work. Figure 4.3 shows such a filtered APRBS with a filter time constant of 3 seconds. This time constant is a design parameter and has to be chosen in dependency of the dynamic behavior of the control system. The time constant of the filter should be big enough to to avoid overshots of the GDNN due to the discontinuity jumps in the signal and small enough to reach a satisfying control speed.

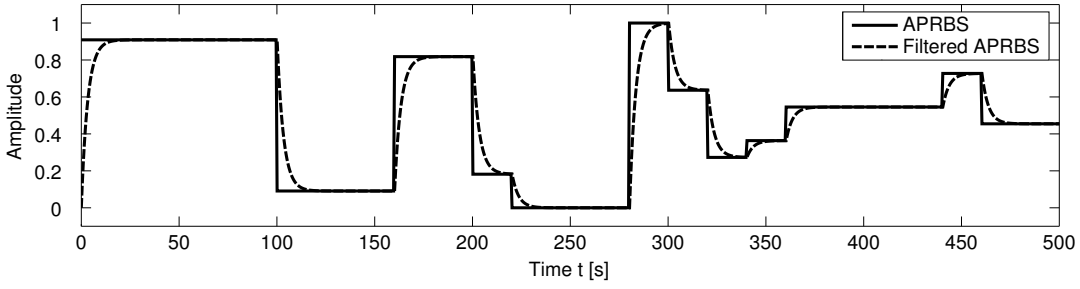


Figure 4.3.: Low-pass filtered APRBS with a time constant of 3 seconds

5. Simulation and Experiments

5.1. Nonlinear Two-Tank System

In this chapter, four different control systems are presented which are then used as a benchmark for the former described neural PID auto-tuner. In this section, the nonlinear two-tank system is presented. The physical structure of this dynamical system is shown in figure 5.1.

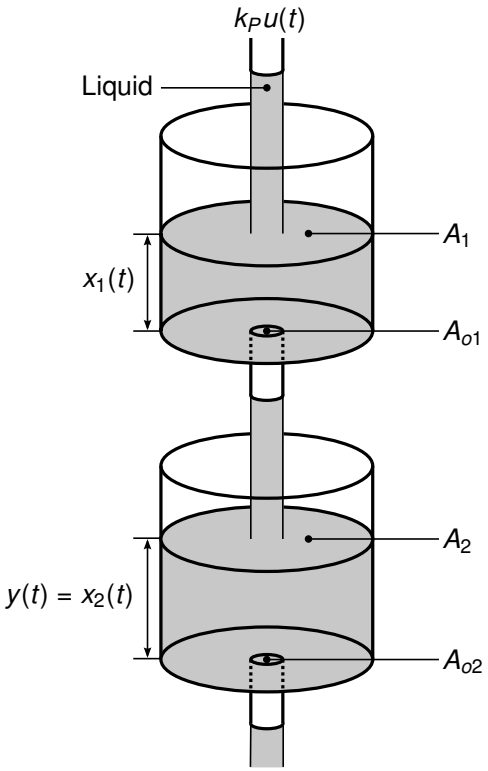


Figure 5.1.: The nonlinear two-tank system

The input voltage $u(t)$ controls the valve which has some pump constant k_P. The cross-sectional areas of the two tanks are denoted with A_1 and A_2, the cross-sectional areas of the output orifices of the tanks with A_{o1} and A_{o2} respectively. The liquid levels of the two tanks are represented by $x_1(t)$ and $x_2(t)$. The goal is to control the liquid level $y(t) = x_2(t)$

of the second tank, while the input $u(t)$ only has a direct influence on the liquid level of the first tank, $x_1(t)$. The differential equations for this dynamical system can then be expressed as follows [123]

$$\begin{aligned} \dot{x_1}(t) &= -\frac{A_{o1}}{A_1}\sqrt{2gx_1(t)} + \frac{k_P}{A_1}u(t)\,, \\ \dot{x_2}(t) &= \frac{A_{o1}}{A_2}\sqrt{2gx_1(t)} - \frac{A_{o2}}{A_2}\sqrt{2gx_2(t)} = y(t)\,, \end{aligned} \tag{5.1}$$

where g denotes the gravitational constant. The numerical values for the parameters of the control system are adapted from [123] and listed in table 5.1.

Table 5.1.: Physical parameters of the two-tank system

Parameter	Symbol	Value	Unit
Tank 1, 2 cross-sectional areas	A_1, A_2	15.38	cm^2
Tank 1, 2 orifice cross-sectional areas	A_{o1}, A_{o2}	0.1781	cm^2
Pump constant	k_P	4.6	cm^3/Vs
Gravitational constant	g	980	cm/s^2

5.2. Linear Plant with Non-Neglectable Input Delay

The next control system which was considered in this work is a first order LTI (linear time invariant) system with a non-neglectable input delay. Its characteristic unit step response is shown in figure 5.2.

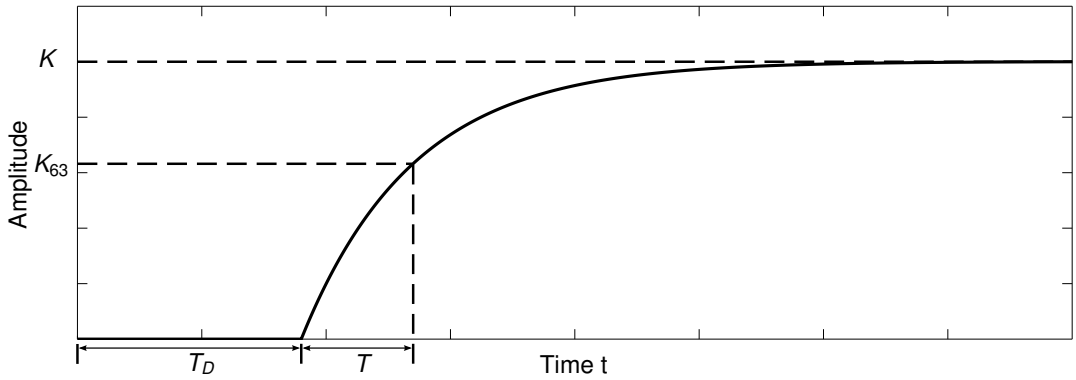


Figure 5.2.: Unit step response of a first order LTI with input delay

Here K is the gain of the of the system and T_D is its input delay. The time constant T is the time the first order LTI needs to reach an amplitude of $K_{63} = K(1 - \frac{1}{e}) \approx 0.63K$ on an

unit step response after the input delay of T_D. Its Laplace transform and its corresponding differential equation are shown in equation 5.2

$$F(s) = \frac{K}{Ts+1} e^{-T_D s} \quad \bullet\!\!-\!\!\circ \quad \dot{x_1}(t) = \frac{1}{T}\Big(Ku(t - T_D) - x_1(t)\Big), \tag{5.2}$$

where $F(s)$ represents the transfer function of the LTI system. The numeric parameters for this system are $K = 0.4$, $T = 0.9$, $T_D = 1.8$ and taken out of [124].

5.3. Nonlinear Inverted Pendulum on a Cart

The third benchmark system is the nonlinear inverted pendulum on a cart. The control task was to balance the pendulum at its unstable equilibrium position while the x position of the cart should be able to follow a predefined trajectory. The inverted pendulum on a cart is shown in figure 5.3.

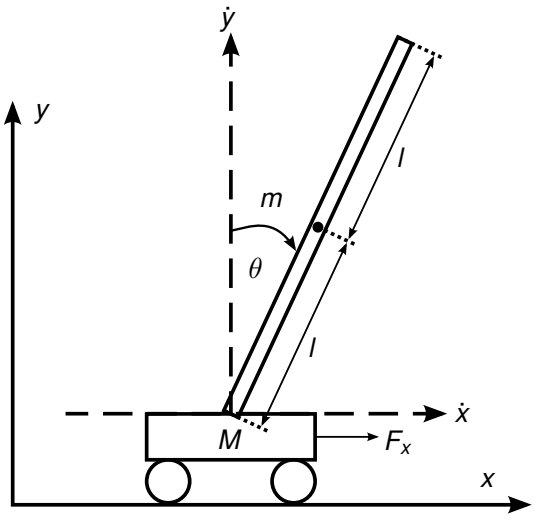


Figure 5.3.: The nonlinear inverted pendulum on a cart

Here M and m denote the mass of the cart respectively the pendulum, θ the angle between the pendulum and the vertical and l the distance from the pivot to the mass center of the pendulum. The position of the pendulum is denoted with x and y, its speed with $\dot{x}$ and $\dot{y}$. The dynamics of the system can be expressed as a system of differential equations [125], where x_1, x_2 denote the position x and the speed $\dot{x}$ of the cart, while x_3, x_4 denote θ and its angular velocity $\dot{\theta}$. The system of differential equations for the inverted pendulum on a cart is shown in equation 5.3. For the sake of clarity, the dependency of the states x_i of t is omitted for the following equations, such that $x_i \coloneqq x_i(t)$.

$$
\begin{aligned}
\dot{x_1}(t) &= x_2\,, \\
\dot{x_2}(t) &= \frac{-mg\cos(x_3)\sin(x_3) + ml\sin(x_3)x_4^2 + F_x}{M + m\sin^2(x_3)} + d_1\,, \\
\dot{x_3}(t) &= x_4\,, \\
\dot{x_4}(t) &= \frac{-ml\cos(x_3)\sin(x_3)x_4^2 - \cos(x_3)F_x + (M+m)g\sin(x_3)}{Ml + ml\sin^2(x_3)} + d_2\,.
\end{aligned}
\tag{5.3}
$$

The variables d_1 and d_2 denote disturbance terms. The numerical values for the parameters of the pendulum are taken from [125] and listed in table 5.2.

Table 5.2.: Physical parameters of the inverted pendulum on a cart

Parameter	Symbol	Value	Unit
Cart mass	M	1	kg
Pendulum mass	m	0.1	kg
Pendulum length to mass center	l	0.3	m
Gravitational constant	g	9.8	m/s^2

5.4. Chaotic Thermal Convection Loop

The last benchmark control system was a chaotic thermal convection loop (also: Lorenz system). It consists of a torus-shaped, water filled pipe. The lower half of the torus can be heated electrically, while the upper half can be cooled down with cooling water. The goal is to control the temperature in such a way, that the liquid in the torus flows with constant velocity. Figure 5.4 shows the thermal convection loop. The temperatures were measured at the points A, B, C and D, the values r_1 and r_2 describe the radius of the convection loop respectivly the torus. According to [126, pp. 222-226], the dynamics of the system can be described as

$$
\begin{aligned}
\dot{x_1}(t) &= p(x_2 - x_1)\,, \\
\dot{x_2}(t) &= x_1 - x_2 - x_3(x_1 + \beta), \\
\dot{x_3}(t) &= x_1x_2 + \beta(x_1 + x_2) - x_3 - u\,.
\end{aligned}
\tag{5.4}
$$

Here, β is a substitution for $\sqrt{R_0 + u}$, with $R = R_0 + u$, where R is the Rayleigh number of the fluid in the convection loop and indicates the type of heat transmission. The value p stands for the so-called Prandtl number and is defined as the ratio of the kinematic viscosity and the thermal diffusivity of a fluid, compare [126, p. 224]. The state x_1 denotes the

average flow velocity, x_2 the temperature difference $T_B - T_A$ between points A and B and x_3 the temperature difference $T_C - T_D$ between points C and D (coordinate transformed by $\sqrt{R_0 - 1}$ in each case).

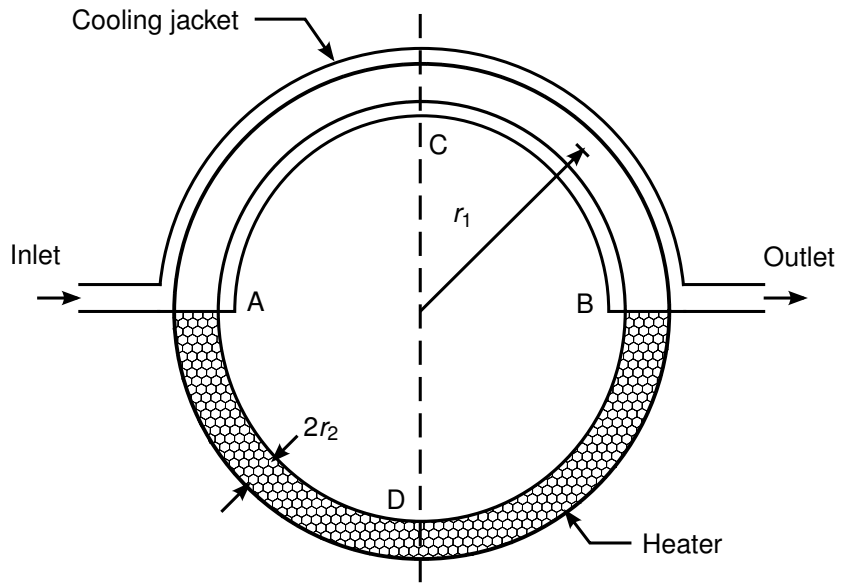


Figure 5.4.: The chaotic thermal convection loop

The numeric values for the physical parameters of the thermal convection loop are taken from [126, p. 226] and listed in table 5.3.

Table 5.3.: Physical parameters of the thermal convection loop

Parameter	**Symbol**	**Value**	**Unit**
Prandtl number	p	10	-
Rayleigh number (substituted)	β	6	-
Convection loop radius	r_1	38	*cm*
Tube radius	r_2	1.5	*cm*

For those values, the system exhibits chaotic behavior. The goal of the controller is here to suppress this chaotic behavior and stabilize the process.

The next chapter will present the results of the neural PID auto-tuner described in the previous chapters on those four benchmark systems, where each has its own difficulties. The results will be compared against the standard PID controller as well as one different state-of-the-art method for each specific system.

6. Comparisons and Results

6.1. Metrics and Methodology

In this section, the methodology and metrics applied to the comparison are presented. The simulation of the dynamical control systems was performed using the Dormand-Prince solver, a numerical, variable step-size, ODE solver [127]. For the implementation, Odeint [128] from the C++ boost library was used [1]. For all simulations, the C++ double precision data type and an absolute and relative tolerance of 10^{-6} was used. The chosen activation function for all neural network was the tanh function, following the rationale of [129].

For system identification, the normalized root-mean-square deviation (NRMSD) was used as an accuracy measurement of the identification result. The formula for the NRMSD between the real system output $\boldsymbol{y}$ and its approximation $\hat{\boldsymbol{y}}$ by the network is given by

$$\begin{aligned} \text{RMSD} &= \sqrt{\frac{(\boldsymbol{y}-\hat{\boldsymbol{y}})^T(\boldsymbol{y}-\hat{\boldsymbol{y}})}{n}}\,, \\ \text{NRMSD} &= \frac{\text{RMSD}}{\max_i y_i - \min_i y_i}\,. \end{aligned} \tag{6.1}$$

For the PID auto-tuner, the standard root-mean-square deviation (RMSD) from equation 6.1 was used. The reason for applying two different metrics for identification and controller tuning was, that in system identification the amplitudes of the inputs and outputs can differ by several orders of magnitude. Therefore, an amplitude normalization was performed which significantly improved identification results. In contrast to this, there was almost no difference between the NRMSD and the RMSD for the PID auto-tuner. Therefore the RMSD measure was applied for the PID auto-tuner.

To simulate real world conditions and test the robustness of the presented methods, a measurement noise $\boldsymbol{z}$ was included in the simulations of both the system identification part and the PID auto-tuner part. For a convenient measure of the noise level, the signal-to-noise ratio (SNR) was used:

$$\text{SNR} = 10\log_{10}\left(\frac{\boldsymbol{y}^T\boldsymbol{y}}{\boldsymbol{z}^T\boldsymbol{z}}\right)\text{dB}\,. \tag{6.2}$$

The noise was generated using a normal distribution with zero mean and a variance so that the desired SNR between the output without noise $\boldsymbol{y}$ and the noise signal $\boldsymbol{z}$ was reached. The noise was applied in addition into the control loop by computing $\boldsymbol{y}_{\text{Noise}} = \boldsymbol{y}+\boldsymbol{z}$.

[1] See `www.boost.org`

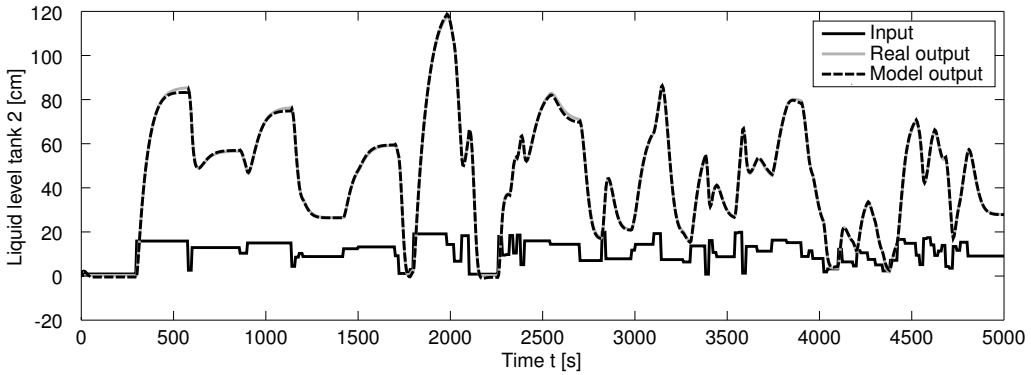
6.2. System Identification Results

In this section, the results of the system identification part are presented, beginning with the nonlinear two-tank system. The initial conditions were set to $x_1 = 5, x_2 = 5$. Table 6.1 shows the configuration parameters of the APRBSs that were used.

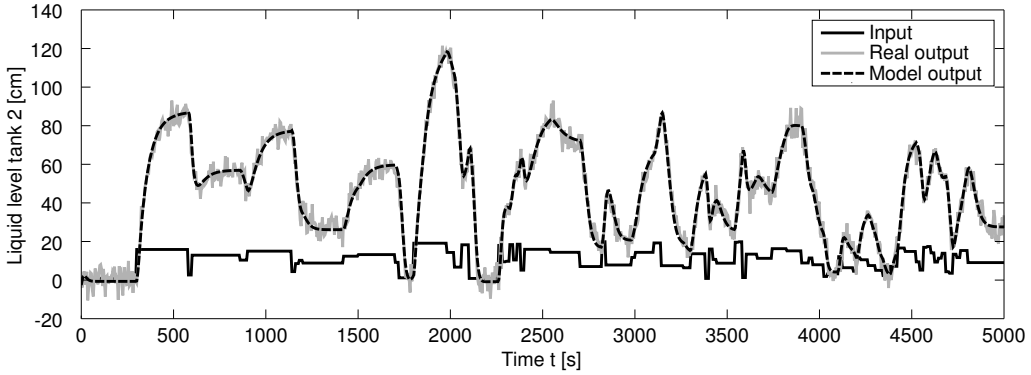
Table 6.1.: Parameters of the APRBS used for identifying the two-tank system

Signal Type	Length	Max. Hold Time	Range
Training signal	5000	200	[0, 20]
Validation signal	40000	200	[0, 20]
Test signal	500000	300	[0, 20]

Figure 6.1 shows the identification results of the chosen recurrent neural network on the test signal, first trained without noise and second with a white noise with a SNR = 20 dB.



(a) Model output (no noise)



(b) Model output (SNR = 20 dB)

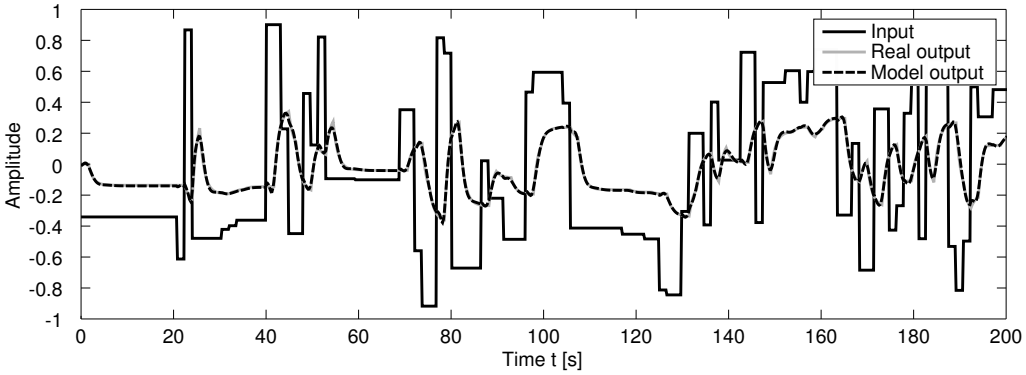
Figure 6.1.: System identification results of the nonlinear two-tank system

The second system is the LTI system with input delay. The APRBS parameters used for its identification are shown in table 6.2.

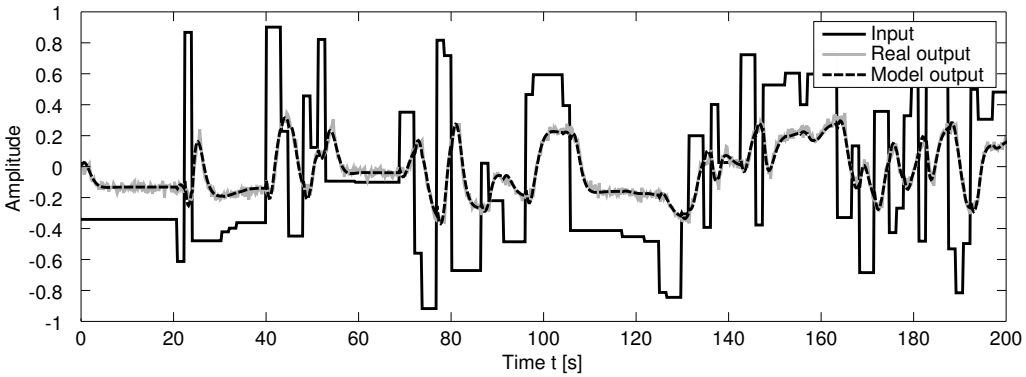
Table 6.2.: Parameters of the APRBS used for identifying the LTI system with input delay

Signal Type	Length	Max. Hold Time	Range
Training signal	500	10	$[-1, 1]$
Validation signal	5000	20	$[-1, 1]$
Test signal	10000	20	$[-1, 1]$

For the identification, the same network as for the two-tank system was used. Figure 6.2 shows the output of the identified model on the test signal, first without noise on the training signal, then with white noise of a SNR = 20 dB. Since the system is linear, it is sufficient to excite it with an amplitude form -1 to 1, as any other amplitude can be reached by a multiplication with the input amplitude. The initial condition of the system was set to $x_1 = 0$.



(a) Model output (no noise)



(b) Model output (SNR = 20 dB)

Figure 6.2.: System identification results of the LTI system with input delay

The remaining results of the system identification (including the inverted pendulum and the thermal convection loop) are listed in table 6.3. Here the mean and the variance of 50 different identification runs based on the NRMSD metric for the corresponding test data are shown.

Table 6.3.: Identification results for the four benchmark systems on test data

Identification Benchmark	SNR	NRMSD (mean)	NRMSD (variance)
Two-tank system	-	$3.4 \cdot 10^{-3}$	$2.0 \cdot 10^{-5}$
Two-tank system	20 dB	$8.8 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$
LTI system with delay	-	$2.4 \cdot 10^{-2}$	$1.9 \cdot 10^{-5}$
LTI system with delay	20 dB	$9.9 \cdot 10^{-2}$	$2.4 \cdot 10^{-3}$
Inverted Pendulum	-	$1.4 \cdot 10^{-1}$	$2.4 \cdot 10^{-5}$
Inverted Pendulum	20 dB	$4.2 \cdot 10^{-1}$	$5.6 \cdot 10^{-4}$
Thermal convection loop	-	$2.4 \cdot 10^{-2}$	$3.4 \cdot 10^{-6}$
Thermal convection loop	20 dB	$4.6 \cdot 10^{-2}$	$8.6 \cdot 10^{-6}$

Since the inverted pendulum is an unstable system, it cannot be excited and learned directly. Therefore a standard LQ regulator [130, pp. 281-306] was designed to stabilize it. The system dynamics were then learned by a neural network by observing the LQ regulator controlling the system while applying an APRBS disturbance signal to the pendulum. This method is called closed-loop identification and summarized for example in [118, pp. 541-546]. Since the LQ regulator is very sensitive towards discontinuities in the disturbance signal, a filtered APRBS was used for the identification. For this purpose, the corresponding APRBS was convoluted using a Gaussian kernel with $\sigma = 0.2$ and $\mu = 0$ and a kernel window from -2 to 2 with a sample size of 400. The remaining plots of the identification and the LQ regulator can be found in appendix A.

While the two-tank and the LTI system with input delay could be identified by the GDNNs offline, this was not possible for the inverted pendulum and the thermal convection loop, since those systems react very sensitive on inputs and have very fast dynamics. For such systems, an offline identification is not suitable [131]. Therefore an online identification with a sliding window was used for those two systems. A brief introduction to online and offline learning and their differences is provided in [118, pp. 81-83]. The identification results for the inverted pendulum and the chaotic thermal convection loop are shown in figure A.1 and figure A.2.

Analyzing the results shown in table 6.3 and the corresponding plots, the recurrent neural networks used for identification are able to follow the dynamics of each benchmark process of this paper. For the two-tank and the LTI system with input delay an offline identification could be used. The inverted pendulum and the thermal convection loop required an online approach. Looking at the variance of the NRMSDs, the results were also relatively similar and regarding their quality not strongly affected by measurement noise.

6.3. Adaptive Controller Results

6.3.1. Comparison with State-of-the-Art Methods

In this section, the performance of the adaptive PID controller on the described benchmark systems is shown and compared to different state-of-the-art methods. As all those methods require a model of the control plant, this section focuses on the model based approached described in chapter 4.

The first system considered, is the inverted pendulum on a cart. In [125], this system is controlled with two stacked PID controllers with fixed parameters. This PID-stack and the LQ regulator from section 6.2 were compared against the adaptive PID controller of this paper. Figure 6.3 shows the results of a simulation of those systems for 35 seconds. After 10 seconds of simulation, a disturbance force of 8.5 N was applied for 0.5 seconds. This

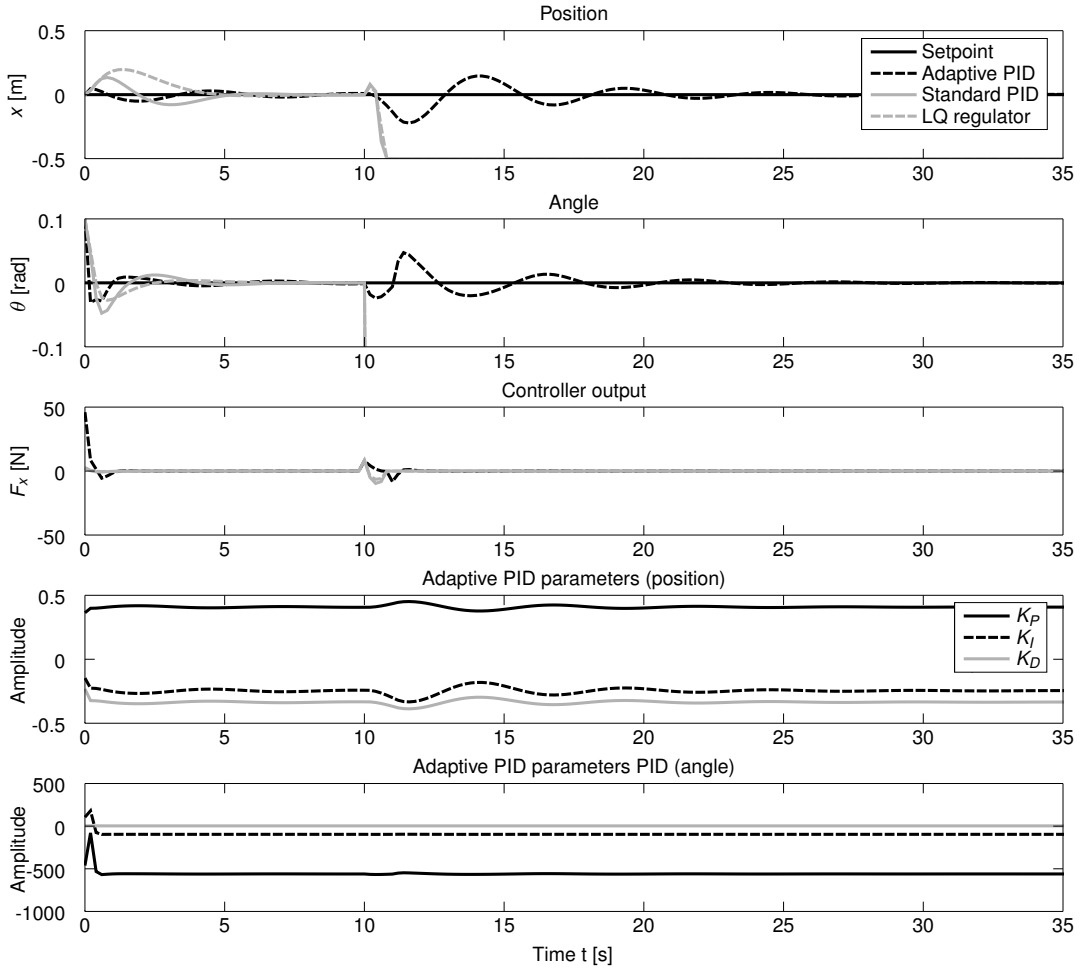


Figure 6.3.: Control of the inverted pendulum with disturbance at t = 10 s

disturbance simulates a hit against the pendulum and was increased from zero until only one controller was still able to control the pendulum (which was the case at 8.5 N). As depicted in figure 6.3, all three controllers are able to stabilize the pendulum from its starting angle of 0.1 rad (the remaining initial conditions were set to zero). However, after applying the disturbance force at second 10, both the non-adaptive PID controller and the LQ regulator fail to compensate this disturbance within the allowed range of $x = \pm 0.5$ m. The adaptive PID controller adopts its parameters online and is able to keep the pendulum stable. The network structure used for the tuning of the PID controller is shown in figure 6.4. Because the inverted pendulum system has two outputs, two of these networks (one for the angle θ and one for the position x) were used in parallel as described in section 4.1.

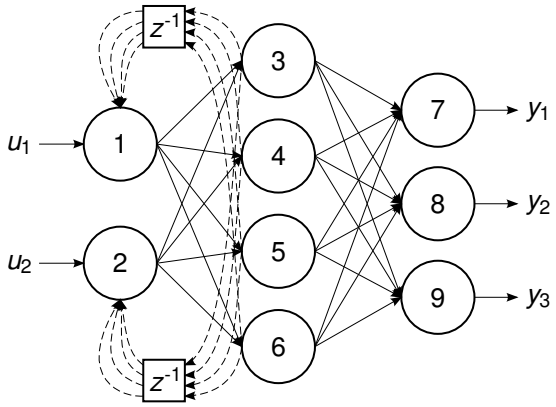


Figure 6.4.: The PID auto-tuner used to control the inverted pendulum

The next system to be considered is the chaotic thermal convection loop. In [126, pp. 222-226], the nonlinear control method backstepping is successfully used to stabilize this system (initial conditions $x_1 = x_2 = x_3 = 5$). The adaptive PID controller is trained to control this system using the neural network from figure 6.4 and compared against the backstepping method. Figure 6.5 shows the results of the two control methods.

As visualized by this figure, both methods are able to stabilize the chaotic process. The adaptive PID controller has an 13.2% lower RMSD than the backstepping method and considerably less over- and undershots. After the setpoint is reached, both the control output and the PID parameters converge to a stationary value, just as the controller output of both methods, which was limited to ± 100 W.

Since both methods are able to control the chaotic system, a disturbance force of -100 W was applied for the duration of 0.5 seconds to the system. This perturbation can be interpreted as a temporary change in the cooling water temperature. Figure 6.6 shows the results of the two controllers for this setup, with the disturbance applied from $t = 5$ to $t = 5.5$ seconds. It is important to note that the adaptive PID controller was not explicitly trained to compensate this disturbance force and its training samples didn't include any disturbances at all. As can be seen in figure 6.6, the adaptive PID controller is able to compensate the

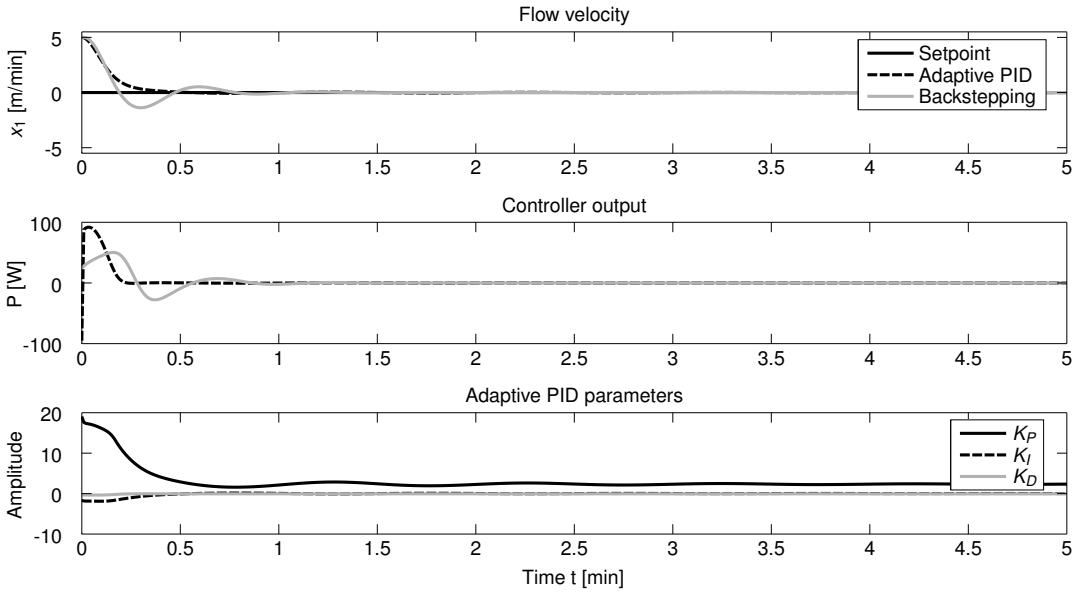


Figure 6.5.: Control of the thermal convection loop

disturbance and after adopting its parameters, it reaches a stationary state again. The backstepping method isn't able to compensate the disturbance and gets unstable.

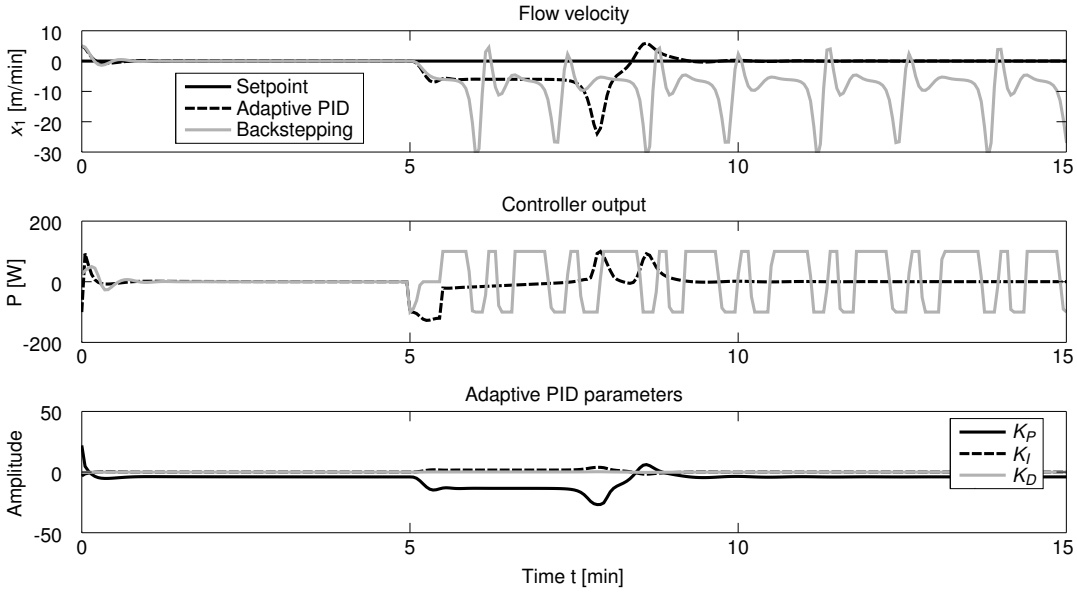


Figure 6.6.: Control of the thermal convection loop with disturbance at t = 5 s

Another comparison can be done between the adaptive PID and a PID with fixed pa-

rameters. Here, the parameters of the non-adaptive PID controller were refined using a genetic algorithm (GA) with a population size and a number of generations of 1000. This GA was executed 100 times, resulting in 10^8 evaluated parameter sets. The parameter set with the least RMSD was compared to the adaptive PID controller and is shown in figure 6.7. While the adaptive PID (as already shown) stabilizes the system at its equilibrium, the PID controller with fixed parameters gets unstable and stuck at a value of $x_1 = -6$.

This behavior can be interpreted when taking a look at the differential equations describing the system, shown in equation 5.4. Especially interesting is here the derivative of the state x_2. If x_1 is exactly $-\beta$ (which is -6 here), the term $x_3(x_1 - \beta)$ from equation 5.4 gets 0. Since this is the only part where the states x_1, x_2 depend on x_3, they get independent of x_3 then. But as our controller can only influence the system at the state x_3, the controller has lost the possibility to interact with the system and gets stuck at $-\beta$. The reason why the GA chooses such a parameter set can be interpreted as follows: As most parameter sets for a fixed PID controller lead to unstable and chaotic behavior, the GA chooses the parameter set with the minimal RMSD and therefore "prefers" a set where it gets stuck. Since no parameter set could be identified that stabilized the system without getting stuck at $-\beta$, it seems reasonable to conclude that a PID controller with fixed parameters is not capable of controlling the thermal convection loop.

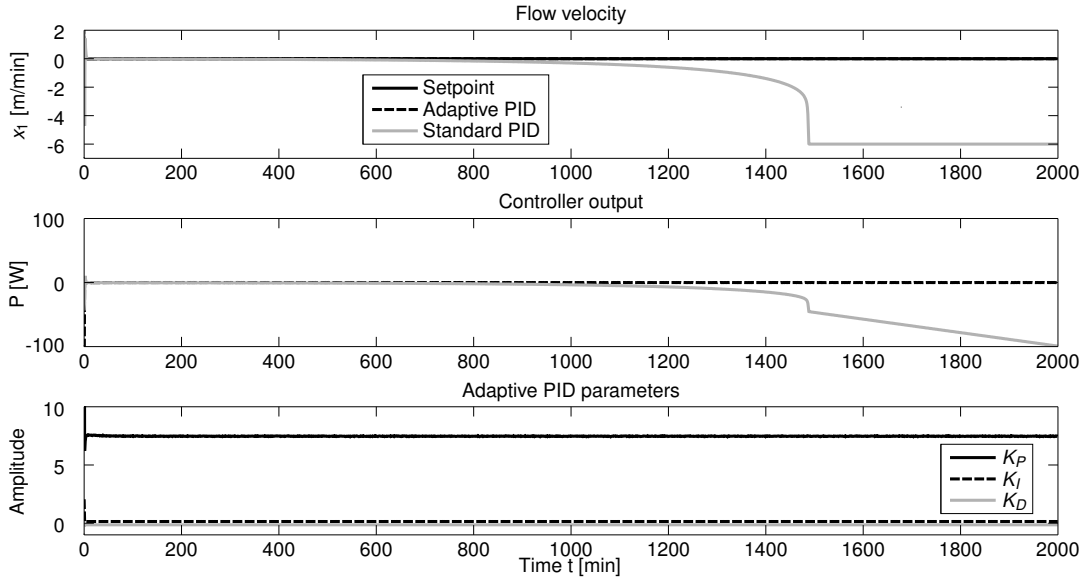


Figure 6.7.: Control of the thermal convection loop, fixed PID performance

The remaining control results including plots of the two-tank system and the LTI with time delay as well as the neural networks used for the PID tuning can be found in appendix B. For the PID tuning 50 independent optimization runs were made for each benchmark configuration. Numerical values for the corresponding RMSDs (means and variances) are listed in table 6.4. The benchmarks cover all possible combinations of disturbance and

noise. In all cases the noise signal was generated in such a way that it had a SNR of 20 dB to the system output signal without noise [2]. The results with the lowest RMSD for the corresponding benchmark are marked **bold**.

Table 6.4.: Control results for the four benchmark systems over 50 independent runs

Control Benchmark		**RMSD on test data over 50 runs**			
Disturbance		-	-	✓	✓
SNR		-	20 dB	-	20 dB
Two-tank system					
Mean	Adaptive PID	$\mathbf{7.4 \cdot 10^{-1}}$	$\mathbf{8.6 \cdot 10^{-1}}$	$\mathbf{8.4 \cdot 10^{-1}}$	$\mathbf{1.0 \cdot 10^{0}}$
	Standard PID	$9.5 \cdot 10^{-1}$	$9.9 \cdot 10^{-1}$	$1.0 \cdot 10^{0}$	$1.1 \cdot 10^{0}$
	Backstepping	$1.1 \cdot 10^{0}$	$1.1 \cdot 10^{0}$	$1.2 \cdot 10^{0}$	$1.1 \cdot 10^{0}$
Variance	Adaptive PID	$3.6 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	$5.6 \cdot 10^{-3}$	$4.6 \cdot 10^{-2}$
	Standard PID	$2.5 \cdot 10^{-3}$	$2.3 \cdot 10^{-3}$	$5.2 \cdot 10^{-3}$	$6.3 \cdot 10^{-3}$
	Backstepping	$1.6 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$	$3.6 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$
LTI system with input delay					
Mean	Adaptive PID	$\mathbf{1.3 \cdot 10^{-1}}$	$\mathbf{1.5 \cdot 10^{-1}}$	$2.6 \cdot 10^{-1}$	$2.8 \cdot 10^{-1}$
	Standard PID	$2.2 \cdot 10^{-1}$	$2.3 \cdot 10^{-1}$	$3.6 \cdot 10^{-1}$	$3.8 \cdot 10^{-1}$
	Smith predictor	$1.8 \cdot 10^{-1}$	$1.9 \cdot 10^{-1}$	$\mathbf{1.9 \cdot 10^{-1}}$	$\mathbf{2.0 \cdot 10^{-1}}$
Variance	Adaptive PID	$6.0 \cdot 10^{-4}$	$5.3 \cdot 10^{-4}$	$6.8 \cdot 10^{-4}$	$3.4 \cdot 10^{-4}$
	Standard PID	$6.7 \cdot 10^{-4}$	$6.1 \cdot 10^{-4}$	$4.0 \cdot 10^{-4}$	$3.2 \cdot 10^{-4}$
	Smith predictor	$5.6 \cdot 10^{-4}$	$4.7 \cdot 10^{-4}$	$4.9 \cdot 10^{-4}$	$3.2 \cdot 10^{-4}$
Inverted pendulum					
Mean	Adaptive PID	$\mathbf{3.4 \cdot 10^{-2}}$	$\mathbf{1.8 \cdot 10^{-2}}$	$\mathbf{9.0 \cdot 10^{-2}}$	$\mathbf{2.7 \cdot 10^{-2}}$
	Standard PID	$3.5 \cdot 10^{-2}$	$3.6 \cdot 10^{-2}$	$1.4 \cdot 10^{2}$	$1.4 \cdot 10^{2}$
	LQ regulator	$5.2 \cdot 10^{-2}$	$5.3 \cdot 10^{-2}$	$1.4 \cdot 10^{2}$	$1.4 \cdot 10^{2}$
Variance	Adaptive PID	$4.0 \cdot 10^{-4}$	$2.6 \cdot 10^{-4}$	$4.9 \cdot 10^{-2}$	$7.3 \cdot 10^{-4}$
	Standard PID	0	$1.1 \cdot 10^{-7}$	0	$2.3 \cdot 10^{-3}$
	LQ regulator	0	$3.3 \cdot 10^{-9}$	0	$2.2 \cdot 10^{-3}$
Thermal convection loop					
Mean	Adaptive PID	$\mathbf{2.3 \cdot 10^{-1}}$	$9.0 \cdot 10^{-1}$	$\mathbf{1.9 \cdot 10^{0}}$	$\mathbf{1.7 \cdot 10^{0}}$
	Standard PID	$6.5 \cdot 10^{-1}$	$1.1 \cdot 10^{0}$	$2.5 \cdot 10^{0}$	$2.4 \cdot 10^{0}$
	Backstepping	$2.6 \cdot 10^{-1}$	$\mathbf{8.9 \cdot 10^{-1}}$	$9.8 \cdot 10^{0}$	$9.8 \cdot 10^{0}$
Variance	Adaptive PID	$1.2 \cdot 10^{-5}$	$3.1 \cdot 10^{-4}$	$1.6 \cdot 10^{-1}$	$2.2 \cdot 10^{-1}$
	Standard PID	$1.4 \cdot 10^{-3}$	$4.3 \cdot 10^{-3}$	$9.6 \cdot 10^{-1}$	$6.8 \cdot 10^{-1}$
	Backstepping	0	$1.3 \cdot 10^{-4}$	0	$1.4 \cdot 10^{-5}$

[2] A variance of (exactly) zero means that the benchmark was purely deterministic (no noise, no APRBS with random amplitude). This is only true for systems that were controlled to stay in their unstable equilibrium.

In the following the above presented results are discussed for each system in detail. Furthermore a short description of the used disturbance signal from the benchmark is provided. The control methods used for comparison were taken out of the cited papers.

- **Two-tank system** For the two-tank system, the adaptive PID controller shows a considerably better performance than both the standard PID controller with fixed parameters (tuned with a GA) and the backstepping approach from [123]. In the simulation between $t = 20$ and $t = 40$ s, the controller output was set to zero to simulate an external disturbance (which can be interpreted as a temporary siphon of fluid). The fixed PID controller has a remarkable overshot after this disturbance. The backstepping algorithm doesn't show this behavior but reacts overall slower than both the adaptive and the fixed PID controller. Looking at table 6.4, the adaptive PID controller has overall 14.9% less RMSD than the fixed PID controller and 23.6% less than the backstepping method.
- **LTI system with input delay** Also on the LTI system with input delay the adaptive PID controller is able to outperform the static PID controller with fixed parameters in all cases (improvement by 31.1%). Here a (dimensionless) disturbance of -5 was added to the system output between seconds 50 and 75 of the simulation. This can be interpreted as a temporary blockage in a fluid transport system. The adaptive PID was also compared to a Smith predictor [132] which was tuned using the Matlab™ PID-tuning tool (based on [133]). Although the adaptive PID controller has no knowledge about the delay, it shows in two of the four benchmarks a 24.3% better control performance than the Smith predictor which requires the exact delay.
- **Inverted pendulum** The inverted pendulum was controlled with the LQ regulator described in A.1 and the PID-stack from [125]. As disturbance, between $t = 10$ and $t = 10.5$ s, a force of 8.5 N was applied. While the LQ regulator and the standard PID controller fail to compensate the disturbance, which results in a high RMSD, the adaptive PID controller is able to keep the pendulum stable and has a overall 99.9% smaller RMSD under disturbance compared to the other methods. This is especially remarkable since the LQ regulator has knowledge about all states of the system, while the adaptive PID controller has only knowledge about the systems output and its deviation from the current setpoint.
- **Thermal convection loop** Analyzing the adaptive PID controller on this benchmark, it shows 35.4% better performance than the standard PID and 1.7% than the backstepping method without disturbance. With a disturbance of -100 W applied between $t = 5$ and $t = 5.5$ s, the backstepping method fails to stabilize the chaotic behavior of the system and yields a 81.6% higher RMSD (standard PID: 26.5%) compared to the adaptive PID controller. It is interesting to note, that the PID controller with fixed parameters is able to stabilize the system within the relatively short range of 15 seconds of simulation. However in the long run, it drifts away as described above (compare figure 6.7) and is therefore not suitable for this control system.

6.3.2. Recurrent against Static Neural Networks for PID Tuning

One important question is whether the recurrent connections which induce a temporal component to the network really lead to a better performance than static feed-forward neural networks like MLPs.

To answer this question, the inverted pendulum was investigated on a separate benchmark (without noise, but with disturbance). This system was chosen because it is the only system considered in this work with multiple outputs and is critical to control because of the unstable equilibrium at θ = 0 rad. Three neural networks were considered for the PID tuning, first the dynamic network used in section 6.2, second a static MLP network with the same amount of weights as the dynamic network, but without dynamics and third the static neural network with external dynamics from [134]. The outcome of this comparison over 50 independent trials (with random weight initialization) is illustrated in figure 6.8.

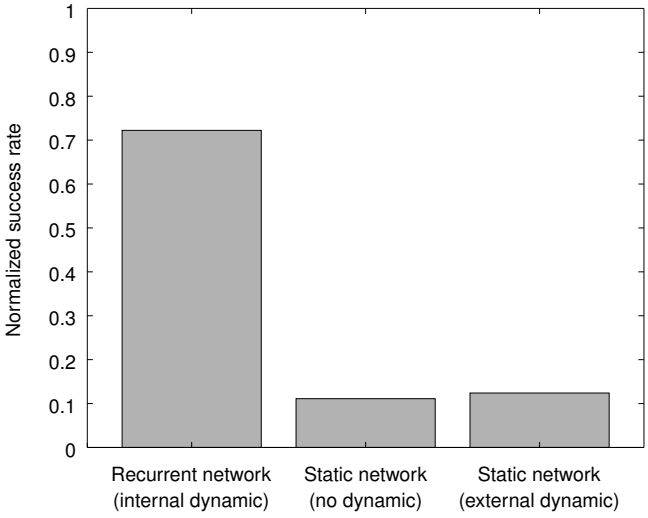


Figure 6.8.: Comparison between static and dynamic neural network performance

Although both networks dispose of the same amount of degrees of freedom, the recurrent network outperforms the static network without dynamics on the chosen control benchmark. It also shows significantly better performance than the static network with external dynamics proposed in [134]. While the nonlinear optimization method from algorithm 8 converged to a stable configuration in 11.1% of the trials for the static network with no dynamics (mean RMSD = $6.8 \cdot 10^{-1}$), and 12.4% for the static network with external dynamics (mean RMSD = $4.0 \cdot 10^{-1}$), for the recurrent network it was able to stabilize the pendulum in 72.2% of the cases (mean RMSD = $2.2 \cdot 10^{-1}$). So it seems the network can take profit of its internal, time dependent memory to achieve a higher performance than both the static network which just takes into account the current control error and state of the system and the static network with external dynamics. Since the optimization has to be repeated with new initial values until a stable configuration is found, a dynamic neural

network approach can decrease computation time compared to a static neural network approach while increasing control performance of the tuned PID controller.

6.3.3. Knowledge-Free against Model Based Approach

This section provides a compact comparison between the knowledge-free and the model based approach described in chapter 4. First a model of the control system is learned by a neural network. Then the PID auto-tuner is trained on that model and finally implemented into the real closed-loop control system. Table 6.5 shows the difference of the RMSDs between the knowledge-free and the model based approach used in section 6.3.1 over 50 independent runs.

Table 6.5.: Knowledge-free compared to model based approach

Control Benchmark	**SNR**	**Model based**		**Knowledge-free**	
		Mean	Variance	Mean	Variance
Two-tank system	-	$7.4 \cdot 10^{-1}$	$3.6 \cdot 10^{-3}$	$1.4 \cdot 10^{0}$	$3.7 \cdot 10^{-2}$
Two-tank system	20 dB	$8.6 \cdot 10^{-1}$	$4.0 \cdot 10^{-3}$	$1.8 \cdot 10^{0}$	$4.4 \cdot 10^{-2}$
LTI system with delay	-	$1.3 \cdot 10^{-1}$	$6.0 \cdot 10^{-4}$	$9.8 \cdot 10^{-1}$	$7.1 \cdot 10^{-3}$
LTI system with delay	20 dB	$1.5 \cdot 10^{-1}$	$5.3 \cdot 10^{-4}$	$1.5 \cdot 10^{0}$	$6.4 \cdot 10^{-2}$
Inverted Pendulum	-	$3.4 \cdot 10^{-2}$	$4.0 \cdot 10^{-4}$	$7.7 \cdot 10^{-2}$	$3.2 \cdot 10^{-3}$
Inverted Pendulum	20 dB	$1.8 \cdot 10^{-2}$	$2.6 \cdot 10^{-4}$	$1.2 \cdot 10^{-1}$	$5.4 \cdot 10^{-1}$
Thermal convection loop	-	$2.3 \cdot 10^{-1}$	$1.2 \cdot 10^{-5}$	$8.2 \cdot 10^{-1}$	$2.2 \cdot 10^{-3}$
Thermal convection loop	20 dB	$9.0 \cdot 10^{-1}$	$3.1 \cdot 10^{-4}$	$1.1 \cdot 10^{0}$	$7.2 \cdot 10^{-3}$

Analyzing these results the model based approach leads to a better control performance in all considered benchmarks. Though the knowledge-free approach is able to achieve a control performance similar to the model based approach. Especially the variance of the different optimization runs is considerably higher for the knowledge-free approach. Therefore it will potentially require more optimization trials until a satisfactory solution is found, with the advantage that no model of the control plant is required to train the PID auto-tuner.

Also a combination of both approaches could be used in industrial practice: First an initial guess for the weights of the PID auto-tuner is derived by knowledge-free training on a neural plant model. Once this PID auto-tuner shows an adequate control performance it can be embedded into the real plant. Training can then be continued using the real plant in the model based learning structure as discussed beforehand.

7. Conclusion

7.1. Summary and Achievements

In this thesis, recurrent neural networks were applied for both system identification and PID tuning tasks. The goal was to investigate whether recurrent neural networks are suitable for those tasks and if they can compete with state-of-the-art control methods. For this purpose several benchmark systems were implemented while each system had its unique difficulties in control. The difficulties includeded several different nonlinearities, instability, input delays or chaotic behavior.

The presented methods turned out to deliver superior results on the tuning of PID parameters on all the considered benchmark systems compared to the conventional PID controller. Moreover several state-of-the-art methods like backstepping or the LQ regulator had a higher number of problems to stabilize systems like the inverted pendulum or the thermal convection loop when high disturbances were applied. The adaptive PID controller showed a more robust behavior in the benchmarks including disturbances and measurement noise. Especially interesting outcomes are that the adaptive PID controller was able to stabilize the inverted pendulum under high disturbances while the LQ regulator failed, although it requires perfect knowledge about all states of the control system. Also the control of the chaotic thermal convection loop lead to a lower RMSD using the adaptive PID controller than the backstepping method, although this nonlinear control method requires the designer to find a suitable Lyapunov function at each design iteration step. Therefore the better control performance should outweigh the implementation efforts of the recurrent neural networks. Also for the two-tank system and the LTI system with input delay, recurrent neural networks proved to be capable of adapting PID parameters in a reasonable way. On the LTI system with input delay the method was able to outperform the Smith predictor in two out of four benchmarks, although the latter requires perfect knowledge about the delay time in the system.

Concluding these observations, it seems that the temporal information, encoded in the recurrent connections, can be abused by the network to increase control performance of the closed-loop system. A comparison of recurrent neural networks with internal dynamic against static neural networks like MLPs showed that the recurrent connections lead to better control performance. The time dependent behavior of dynamical systems is better reproduced by recurrent neural networks which are more suitable for this task than static networks. Also the training algorithm with random weight initialization had a significantly higher success rate when training recurrent neural networks than for their static counterpart. Although static neural networks can be used to model or control dynamical systems,

they often require the system states as input to achieve a good performance, so all states have to be available. This assumption is in general not true for PID controllers, as those controllers require only the output of the system. Therefore recurrent neural networks with internal dynamic seem to be more suitable for neural PID tuning.

A neural network toolbox was implemented as a class template in standard compliant C++11 using modern programming idioms like RAII (resource acquisition is initialization) and supports arbitrary tapped delay lines for inter-neuron connections. Additionally a novel algorithm to parametrize APRBS excitation signals was presented and used in this paper. The algorithm determines the optimal compromise between the desired signal length and maximum hold sequence specified by the user by optimizing a cost function and terminates in logarithmic time.

7.2. Outlook and Future Work

Neural networks in general and their application on system identification and PID controller tuning specifically is still a field that offers plenty unexplored areas for further research. While this paper showed that recurrent neural networks can be successfully applied to both system identification and PID tuning, the open question remains: How to choose the concrete structure of the neural network. This includes especially how many neurons should be used in the network and which connections should be delayed by which amount of time steps. Future work can focus here on structure optimization techniques to identify the optimal network topology for a given control problem.

Furthermore the application of recurrent neural networks for the tuning of other controllers (like LQ regulators or bang-bang controllers) could be investigated and compared against the results presented in this paper. Since the networks used for PID tuning in this work only consist of relatively few neurons, they could be used for real-time computations on a real physical system. So additionally, future work could include the implementation of the presented techniques on a real system and verify the simulation results under real world conditions.

The presented methods demonstrated that recurrent neural networks have the potential to improve PID controlled processes in terms of general control performance and robustness against external perturbations. Companies could therefore use this PID tuning method to achieve a higher control performance on automated processes and lower failure rates due to a more robust controller design.

List of Figures

List of Tables

List of Algorithms

Bibliography

1. S. Bi and M. Deng, “Operator-based robust control design for nonlinear plants with perturbation,” *International Journal of Control*, vol. 84, no. 4, pp. 815–821, 2011.

2. P. Kokotović and M. Arcak, “Constructive nonlinear control: A historical perspective,” *Automatica*, vol. 37, no. 5, pp. 637–662, 2001.

3. M. Wang, X. Liu, and P. Shi, “Adaptive neural control of pure-feedback nonlinear time-delay systems via dynamic surface technique,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 6, pp. 1681–1692, 2011.

4. F.-Y. Wang, N. Jin, D. Liu, and Q. Wei, “Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with-error bound,” *Neural Networks, IEEE Transactions on*, vol. 22, no. 1, pp. 24–36, 2011.

5. S. Heidari, F. Piltan, M. Shamsodini, K. Heidari, and S. Zahmatkesh, “Design new nonlinear controller with parallel fuzzy inference system compensator to control of continuum robot manipulator,” *International Journal of Control and Automation*, vol. 6, no. 4, pp. 115–134, 2013.

6. E. Fridman and U. Shaked, “A descriptor system approach to H-infinity control of linear time-delay systems,” *Automatic Control, IEEE Transactions on*, vol. 47, no. 2, pp. 253–270, 2002.

7. L. Yu and J. Chu, “An LMI approach to guaranteed cost control of linear uncertain time-delay systems,” *Automatica*, vol. 35, no. 6, pp. 1155–1159, 1999.

8. M. Krstic, “Lyapunov stability of linear predictor feedback for time-varying input delay,” *Automatic Control, IEEE Transactions on*, vol. 55, no. 2, pp. 554–559, 2010.

9. H. Hirsch-Kreinsen and J. Weyer, “Wandel von Produktionsarbeit–„Industrie 4.0“,” *Soziologisches Arbeitspapier*, no. 38, 2014.

10. F. Piltan, N. Sulaiman, S. Allahdadi, M. Dialame, and A. Zare, “Position control of robot manipulator: Design a novel SISO adaptive sliding mode fuzzy PD fuzzy sliding mode control,” *International Journal of Artificial intelligence and Expert System*, vol. 2, no. 5, pp. 208–228, 2011.

11. M. Petrov, I. Ganchev, and A. Taneva, “Fuzzy PID control of nonlinear plants,” in *Intelligent Systems, 2002. Proceedings. 2002 First International IEEE Symposium*, vol. 1, pp. 30–35, IEEE, 2002.

12. Z. Li and C. Yang, “Neural-adaptive output feedback control of a class of transportation vehicles based on wheeled inverted pendulum models,” *Control Systems Technology, IEEE Transactions on*, vol. 20, no. 6, pp. 1583–1591, 2012.

13. S. Bennett, “Nicholas Minorsky and the automatic steering of ships,” *Control Systems Magazine, IEEE*, vol. 4, no. 4, pp. 10–15, 1984.

14. K. J. Åström and T. Hägglund, “The future of PID control,” *Control engineering practice*, vol. 9, no. 11, pp. 1163–1175, 2001.

15. K. Åström and T. Hägglund, “Revisiting the Ziegler–Nichols step response method for PID control,” *Journal of process control*, vol. 14, no. 6, pp. 635–650, 2004.

16. Y. Peng, D. Vrancic, and R. Hanus, “Anti-windup, bumpless, and conditioned transfer techniques for PID controllers,” *Control Systems, IEEE*, vol. 16, no. 4, pp. 48–57, 1996.

17. J. Carvajal, G. Chen, and H. Ogmen, “Fuzzy PID controller: Design, performance evaluation, and stability analysis,” *Information Sciences*, vol. 123, no. 3, pp. 249–270, 2000.

18. E. A. Gargari, F. Hashemzadeh, R. Rajabioun, and C. Lucas, “Colonial competitive algorithm: A novel approach for PID controller design in MIMO distillation column process,” *International Journal of Intelligent Computing and Cybernetics*, vol. 1, no. 3, pp. 337–355, 2008.

19. T. Slavov and O. Roeva, “Application of genetic algorithm to tuning a PID controller for glucose concentration control,” *WSEAS Trans Syst*, vol. 11, no. 7, pp. 223–233, 2012.

20. V. Rajinikanth and K. Latha, “Tuning and retuning of PID controller for unstable systems using evolutionary algorithm,” *ISRN Chemical Engineering*, vol. 2012, no. 3, pp. 10–21, 2012.

21. C. W. Anderson, D. C. Hittle, A. D. Katz, and R. Kretchman, “Reinforcement learning, neural networks and PI control applied to a heating coil,” in *Proceedings of the International Conference on EANN*, vol. 96, pp. 135–42, 1996.

22. O. Dominguez-Ramirez, C. Carrillo-Santos, L. Ramos-Velasco, J. Ramos-Fernandez, M. Marquez-Vera, V. Parra-Vega, and A. Sanchez, “Online identification for auto-tuning PID based on wavelet neural networks: An experimental validation on an AC motor,” in *Control Conference (ECC), 2013 European*, pp. 4610–4615, IEEE, 2013.

23. O. Föllinger, *Regelungstechnik: Einführung in die Methoden und ihre Anwendung*. Heidelberg: Hüthig Verlag, 10 ed., 2008.

24. L. Lennart, *System identification: Theory for the user*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1999.

25. S. Engell and F. Allgöwer, *Entwurf nichtlinearer Regelungen*. München: Oldenbourg Verlag, 1995.

26. S. W. Sung and I.-B. Lee, "Limitations and countermeasures of PID controllers," *Industrial & engineering chemistry research*, vol. 35, no. 8, pp. 2596–2610, 1996.

27. J. Han, "From PID to active disturbance rejection control," *Industrial Electronics, IEEE transactions on*, vol. 56, no. 3, pp. 900–906, 2009.

28. S. Bennett, "Development of the PID controller," *Control Systems, IEEE*, vol. 13, no. 6, pp. 58–62, 1993.

29. J. Ziegler and N. Nichols, "Optimum settings for automatic controllers," *Trans. ASME*, vol. 64, no. 11, pp. 759–768, 1942.

30. K. L. Chien, "On the automatic control of generalized passive systems," *Trans. ASME*, vol. 74, pp. 175–185, 1952.

31. U. Kuhn, "Eine praxisnahe Einstellregel für PID-Regler: Die T-Summen-Regel," *Automatisierungstechnische Praxis*, vol. 1995, no. 5, pp. 10–16, 1995.

32. J. Basilio and S. Matos, "Design of PI and PID controllers with transient performance specification," *Education, IEEE Transactions on*, vol. 45, no. 4, pp. 364–370, 2002.

33. G. Schulz, *Regelungstechnik 1: Lineare und Nichtlineare Regelung, Rechnergestützter Reglerentwurf*, vol. 1. München: Oldenbourg Verlag, 3 ed., 2007.

34. N. Thomas and D. P. Poongodi, "Position control of DC motor using genetic algorithm based PID controller," in *Proceedings of the World Congress on Engineering*, vol. 2, pp. 1–3, 2009.

35. A. Bagis, "Determination of the PID controller parameters by modified genetic algorithm for improved performance," *Journal of information science and engineering*, vol. 23, no. 5, pp. 1469–1480, 2007.

36. K. H. Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *Control Systems Technology, IEEE Transactions on*, vol. 13, no. 4, pp. 559–576, 2005.

37. D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *Control Systems, IEEE*, vol. 26, no. 3, pp. 96–114, 2006.

38. J. Henriques, P. Gil, A. Cardoso, and A. Dourado, “Scheduling of PID controllers by means of a neural network with application to a solar power plant,” in *Neural Networks, 2002. IJCNN’02. Proceedings of the 2002 International Joint Conference on*, vol. 1, pp. 311–316, IEEE, 2002.

39. S. Omatu and M. Yoshioka, “Self-tuning Neuro-PID control and applications,” in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, vol. 3, pp. 1985–1989, IEEE, 1997.

40. R. Jafari and R. Dhaouadi, “Adaptive PID control of a nonlinear servomechanism using recurrent neural networks,” in *Advances in Reinforcement Learning*, pp. 275–296, INTECH, 2011.

41. X. D. Ji and B. O. Familoni, “A diagonal recurrent neural network-based hybrid direct adaptive SPSA control system,” *Automatic Control, IEEE Transactions on*, vol. 44, no. 7, pp. 1469–1473, 1999.

42. Z. Xiong and J. Zhang, “A batch-to-batch iterative optimal control strategy based on recurrent neural network models,” *Journal of Process Control*, vol. 15, no. 1, pp. 11–21, 2005.

43. Y. Abe, M. Konishi, J. Imai, R. Hasegawa, M. Watanabe, and H. Kamijo, “Neural network-based PID gain tuning of chemical plant controller,” *Electrical Engineering in Japan*, vol. 171, no. 4, pp. 28–36, 2010.

44. F.-J. Lin and C.-H. Lin, “On-line gain-tuning IP controller using RFNN,” *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 37, no. 2, pp. 655–670, 2001.

45. F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological review*, vol. 65, no. 6, pp. 386–408, 1958.

46. F. Rosenblatt, “Principles of neurodynamics. perceptrons and the theory of brain mechanisms,” tech. rep., DTIC Document, 1961.

47. D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, “The multilayer perceptron as an approximation to a bayes optimal discriminant function,” *Neural Networks, IEEE Transactions on*, vol. 1, no. 4, pp. 296–298, 1990.

48. J. B. Hampshire and B. Pearlmutter, “Equivalence proofs for multi-layer perceptron classifiers and the bayesian discriminant function,” in *Proceedings of the 1990 Connectionist Models Summer School* (T. S. D. Touretzky, J. Elman and G. Hinton, eds.), pp. 1–17, 1990.

49. G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

50. K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

51. N. Morgan and H. Bourlard, “Continuous speech recognition using multilayer perceptrons with hidden markov models,” in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*, pp. 413–416, IEEE, 1990.

52. P. Pandey and S. Barai, “Multilayer perceptron in damage detection of bridge structures,” *Computers & Structures*, vol. 54, no. 4, pp. 597–608, 1995.

53. H. Yan, Y. Jiang, J. Zheng, C. Peng, and Q. Li, “A multilayer perceptron-based medical decision support system for heart disease diagnosis,” *Expert Systems with Applications*, vol. 30, no. 2, pp. 272–281, 2006.

54. K. Khanchandani and M. A. Hussain, “Emotion recognition using multilayer perceptron and generalized feed forward neural network,” *Journal of Scientific and Industrial Research*, vol. 68, no. 5, p. 367, 2009.

55. A. S. Yilmaz and Z. Özer, “Pitch angle control in wind turbines above the rated wind speed by multi-layer perceptron and radial basis function neural networks,” *Expert Systems with Applications*, vol. 36, no. 6, pp. 9767–9775, 2009.

56. B. Verlinden, J. Duflou, P. Collin, and D. Cattrysse, “Cost estimation for sheet metal parts using multiple regression and artificial neural networks: A case study,” *International Journal of Production Economics*, vol. 111, no. 2, pp. 484–492, 2008.

57. M. Perez, “Artificial neural networks and bankruptcy forecasting: A state of the art,” *Neural Computing & Applications*, vol. 15, no. 2, pp. 154–163, 2006.

58. M. Paliwal and U. A. Kumar, “Neural networks and statistical techniques: A review of applications,” *Expert systems with applications*, vol. 36, no. 1, pp. 2–17, 2009.

59. G. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

60. Y. Lecun and Y. Bengio, “Convolutional networks for images, speech, and time series,” *The handbook of brain theory and neural networks*, vol. 3361, pp. 255–258, 1995.

61. V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung, “Supervised learning of image restoration with convolutional networks,” in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007.

62. Y. Lecun, K. Kavukcuoglu, and C. Farabet, “Convolutional networks and applications in vision,” in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 253–256, IEEE, 2010.

63. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, no. 11, pp. 3371–3408, 2010.

64. P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

65. H.-C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach, "Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4D patient data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1930–1943, 2013.

66. R. Kothari and K. Agyepong, "On lateral connections in feed-forward neural networks," in *Neural Networks, 1996., IEEE International Conference on*, vol. 1, pp. 13–18, IEEE, 1996.

67. T. Raiko, H. Valpola, and Y. Lecun, "Deep learning made easier by linear transformations in perceptrons," in *International Conference on Artificial Intelligence and Statistics*, vol. 22, pp. 924–932, 2012.

68. Y. Shang and B. Wah, "Global optimization for neural network training," *Computer*, vol. 29, no. 3, pp. 45–54, 1996.

69. M. I. Jordan, "Serial order: A parallel distributed processing approach," *Advances in psychology*, vol. 121, pp. 471–495, 1997.

70. N. Karunanithi, D. Whitley, and Y. K. Malaiya, "Using neural networks in reliability prediction," *Software, IEEE*, vol. 9, no. 4, pp. 53–59, 1992.

71. R. Yasdi, "Prediction of road traffic using a neural network approach," *Neural computing & applications*, vol. 8, no. 2, pp. 135–142, 1999.

72. A. Baykasoğlu, T. Dereli, and S. Tanış, "Prediction of cement strength using soft computing techniques," *Cement and concrete research*, vol. 34, no. 11, pp. 2083–2090, 2004.

73. J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

74. C.-Y. Liou, J.-C. Huang, and W.-C. Yang, "Modeling word perception using the elman network," *Neurocomputing*, vol. 71, no. 16, pp. 3150–3157, 2008.

75. R. Köker, "Design and performance of an intelligent predictive controller for a six-degree-of-freedom robot using the Elman network," *Information Sciences*, vol. 176, no. 12, pp. 1781–1799, 2006.

76. L. Hongmei, W. Shaoping, and O. Pingchao, “Fault diagnosis based on improved Elman neural network for a hydraulic servo system,” in *Robotics, Automation and Mechatronics, 2006 IEEE Conference on*, pp. 1–6, IEEE, 2006.

77. V. Srinivasan, C. Eswaran, and N. Sriraam, “Artificial neural network based epileptic detection using time-domain and frequency-domain features,” *Journal of Medical Systems*, vol. 29, no. 6, pp. 647–660, 2005.

78. X.-Y. Yang, D.-P. Xu, X.-J. Han, and H.-N. Zhou, “Predictive functional control with modified Elman neural network for reheated steam temperature,” in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 8, pp. 4699–4703, IEEE, 2005.

79. Z. Yang, M. Sun, W. Li, and W. Liang, “Modified Elman network for thermal deformation compensation modeling in machine tools,” *The International Journal of Advanced Manufacturing Technology*, vol. 54, no. 5-8, pp. 669–676, 2011.

80. W. Junsong, W. Jiukun, Z. Maohua, and W. Junjie, “Prediction of internet traffic based on Elman neural network,” in *Control and Decision Conference, 2009. CCDC’09. Chinese*, pp. 1248–1252, IEEE, 2009.

81. J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

82. M. Gupta, L. Jin, and N. Homma, *Static and dynamic neural networks: From fundamentals to advanced theory*. New York: John Wiley & Sons, 2003.

83. C. Endisch, C. Hackl, and D. Schröder, “Optimal brain surgeon for general dynamic neural networks,” in *Progress in Artificial Intelligence*, pp. 15–28, Springer Verlag, 2007.

84. O. De Jesus and M. T. Hagan, “Backpropagation algorithms for a broad class of dynamic networks,” *Neural Networks, IEEE Transactions on*, vol. 18, no. 1, pp. 14–27, 2007.

85. C. Endisch, M. Brache, and R. Kennel, “Parameter identification of a nonlinear two mass system using prior knowledge,” in *Machine Learning and Systems Engineering*, pp. 197–211, Springer Verlag, 2010.

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

87. P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *Neural Networks, IEEE Transactions on*, vol. 5, no. 1, pp. 54–65, 1994.

88. T. G. Barbounis, J. B. Theocharis, M. C. Alexiadis, and P. S. Dokopoulos, "Long-term wind speed and power forecasting using local recurrent neural network models," *Energy Conversion, IEEE Transactions on*, vol. 21, no. 1, pp. 273–284, 2006.

89. A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems*, vol. 21, pp. 545–552, 2009.

90. Y. Xia and J. Wang, "A recurrent neural network for solving nonlinear convex programs subject to linear constraints," *Neural Networks, IEEE Transactions on*, vol. 16, no. 2, pp. 379–386, 2005.

91. M. A. Figueiredo and A. K. Jain, "Unsupervised learning of finite mixture models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 3, pp. 381–396, 2002.

92. S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

93. L. Rabiner and B.-H. Juang, "An introduction to hidden Markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

94. T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

95. D. O. Hebb, *The organization of behavior: A neuropsychological theory*. New York: John Wiley & Sons, 1949.

96. S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," in *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pp. 3–24, IOS Press, 2007.

97. V. N. Vapnik and A. J. Chervonenkis, *Theory of pattern recognition* [in Russian]. Moscow: Nauka, 1974. (German Translation: W. Wapnik and A. Tscherwonenkis, *Theorie der Zeichenerkennung*. Berlin: Akademie-Verlag, 1979).

98. C. J. Burges, "A tutorial on Support Vector Machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

99. S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data mining and knowledge discovery*, vol. 2, no. 4, pp. 345–389, 1998.

100. J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbors," *IEEE Transactions on computers*, vol. 24, no. 10, pp. 1000–1006, 1975.

101. C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. Cambridge: The MIT Press, 2006.

102. I. Rish, "An empirical study of the naive Bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, 2001.

103. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

104. P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

105. R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. Cambridge: MIT Press, 1998.

106. R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

107. G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

108. G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

109. J. Baxter, A. Tridgell, and L. Weaver, "Learning to play chess using temporal differences," *Machine Learning*, vol. 40, no. 3, pp. 243–263, 2000.

110. J. Modayil, A. White, and R. S. Sutton, "Multi-timescale nexting in a reinforcement learning robot," *Adaptive Behavior*, vol. 22, no. 2, pp. 146–160, 2014.

111. C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, University of Cambridge, 1989.

112. L. M. Gambardella, M. Dorigo, *et al.*, "Ant-Q: A reinforcement learning approach to the traveling salesman problem," in *ICML*, pp. 252–260, 1995.

113. B.-Q. Huang, G.-Y. Cao, and M. Guo, "Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 1, pp. 85–89, IEEE, 2005.

114. K.-H. Park, Y.-J. Kim, and J.-H. Kim, "Modular Q-learning based multi-agent cooperation for robot soccer," *Robotics and Autonomous Systems*, vol. 35, no. 2, pp. 109–122, 2001.

115. L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

116. L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 2, pp. 156–172, 2008.

117. Y. Shoham, R. Powers, and T. Grenager, “Multi-agent reinforcement learning: A critical survey,” tech. rep., Stanford University, 2003.

118. O. Nelles, *Nonlinear system identification: From classical approaches to neural networks and fuzzy models*. Berlin, Heidelberg: Springer Verlag, 1 ed., 2010.

119. R. Isermann and M. Munchhof, *Identification of Dynamical Systems*. Berlin, Heidelberg: Springer-Verlag, 2009.

120. D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the Society for Industrial & Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

121. W. H. Press, *Numerical recipes in C++: The art of scientific computing. Example book in C++*. Cambridge: Cambridge University Press, 2002.

122. L. Prechelt, “Automatic early stopping using cross validation: Quantifying the criteria,” *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.

123. H. Pan, H. Wong, V. Kapila, and M. S. de Queiroz, “Experimental validation of a nonlinear backstepping liquid level controller for a state coupled two tank system,” *Control Engineering Practice*, vol. 13, no. 1, pp. 27–40, 2005.

124. S. Tavakoli and M. Tavakoli, “Optimal tuning of PID controllers for first order plus time delay models using dimensional analysis,” in *Control and Automation, 2003. ICCA’03. Proceedings. 4th International Conference on*, pp. 942–946, IEEE, 2003.

125. J.-J. Wang, “Simulation studies of inverted pendulum based on PID controllers,” *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 440–449, 2011.

126. J. Adamy, *Nichtlineare Regelungen*, vol. 1. Berlin: Springer Verlag, 2009.

127. J. R. Dormand and P. J. Prince, “A family of embedded Runge-Kutta formulae,” *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

128. K. Ahnert and M. Mulansky, “Odeint-solving ordinary differential equations in C++,” in *Proceedings AIP Conf. Numerical Analysis Applied Mathematics*, vol. 1389, pp. 1586–1589, 2011.

129. B. L. Kalman and S. C. Kwasny, “Why tanh: Choosing a sigmoidal function,” in *Neural Networks, 1992. IJCNN., International Joint Conference on*, vol. 4, pp. 578–581, IEEE, 1992.

130. J. Lunze, *Regelungstechnik 2: Mehrgrößensysteme, Digitale Regelung*, vol. 1. Berlin, Heidelberg: Springer Verlag, 2010.

131. A. Poznyak, W. Yu, and E. Sanchez, "Identification and control of unknown chaotic systems via dynamic neural networks," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 46, no. 12, pp. 1491–1495, 1999.

132. O. J. M. Smith, "Closed control of loops with dead time," *Chemical Engineering Progress*, vol. 53, no. 5, pp. 217–219, 1957.

133. A. Ingimundarson and T. Hägglund, "Robust tuning procedures of dead-time compensating controllers," *Control Engineering Practice*, vol. 9, no. 11, pp. 1195–1208, 2001.

134. S. Omatu, T. Fujinaka, and M. Yoshioka, "Neuro-PID control for inverted single and double pendulums," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4, pp. 2685–2690, IEEE, 2000.

135. J. Lunze, *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf Einschleifiger Regelungen*. Berlin, Heidelberg: Springer Verlag, 2002.

Appendices

A. Additional System Identification Material

A.1. LQ Regulator Computations for the Inverted Pendulum

To apply a LQ regulator to a system like the inverted pendulum, it has to be linearized beforehand. For linearization, the system has to be differentiated with respect to its states x_1, x_2, x_3, x_4 and its input u. This leads to the standard description of a linear system

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)\,, \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0\,, \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t)\,. \end{aligned} \tag{A.1}$$

A detailed introduction to linearization of nonlinear systems is provided in [135, pp. 106-117]. In the case of the inverted pendulum, these matrices are calculated as

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{g(M+m)}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix}, \boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}^T, \boldsymbol{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{A.2}$$

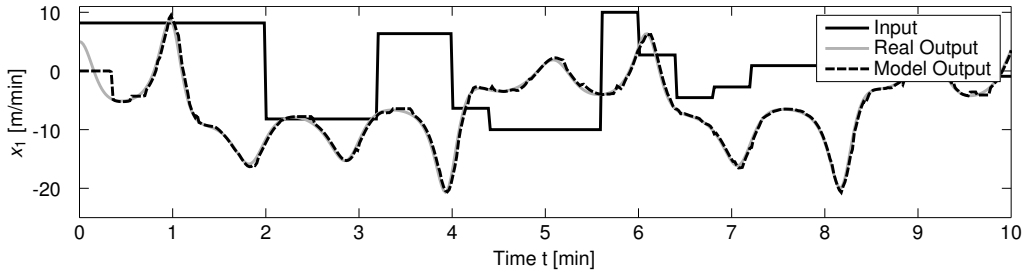
Now applying the method described in [130, pp. 281-306] and inserting the physical parameters of the inverted pendulum from table 5.2, the optimal feedback matrix $\boldsymbol{K}$ and the weight matrix $\boldsymbol{Q}$ for the linearized system can be written as

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{K} = \begin{bmatrix} -27.0283 \\ -4.5509 \\ -1 \\ -1.821 \end{bmatrix}^T. \tag{A.3}$$

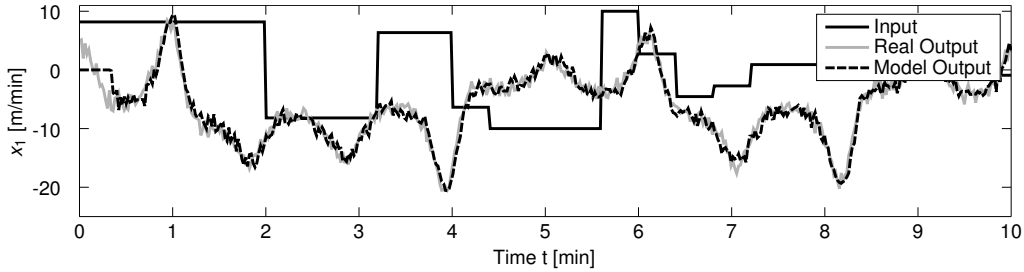
A.2. System Identification Results

Enclosed are the system identification results of the thermal convection loop, shown in figure A.1 and the inverted pendulum on a cart, shown in figure A.2. For the thermal convection loop, a training signal of 10 seconds with 1000 samples, for the inverted pendulum a training signal of 50 seconds with 1000 samples was used. The former was excited using an APRBS with a maximum hold time of 2 seconds and an amplitude range from -10 to 10. The disturbance signal used for closed-loop identification of the inverted pendulum was an APRBS with 5 seconds maximum hold time and an amplitude range from -0.5 to

0.5, convoluted with a Gaussian kernel as described in section 6.2. The sliding window used for online identification had a size of 30 samples, with a validation and prediction horizon of one sample for both systems. As depicted in the system identification plots, the online identification is able to deliver reasonable predictions for both systems after the sliding window has reached its full size (first 0.3 seconds of the thermal convection loop, respectively first 1.67 seconds of the inverted pendulum simulation).



(a) Model output (no noise)



(b) Model output (SNR = 20 dB)

Figure A.1.: System identification results of the chaotic thermal convection loop

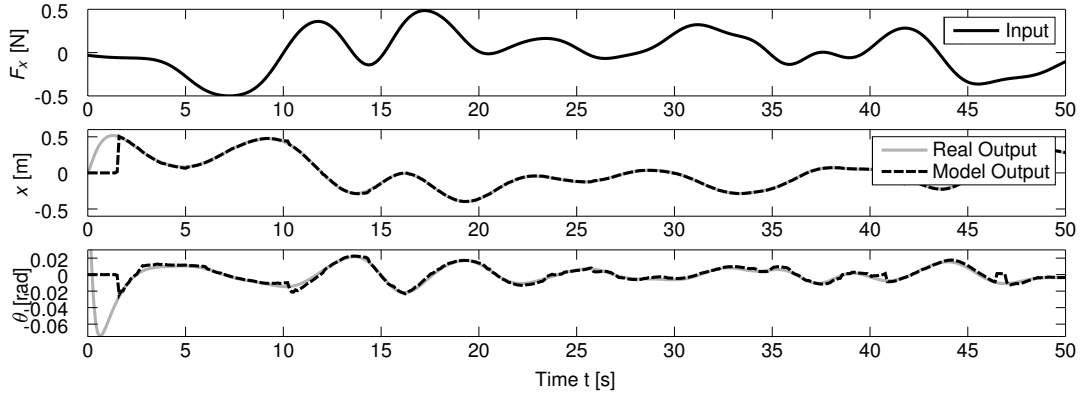


Figure A.2.: System identification results of the inverted pendulum

B. Additional Adaptive Control Material

B.1. Control Results for the Two-Tank System

In this section, the remaining results of the adaptive PID controller are presented. Table B.1 shows the parameters used to generate the APRBS for training the PID auto-tuner to control the two-tank system, figure B.1 illustrates the control results including the results from the fixed PID and the backstepping method. The sample time was 0.1 seconds for all experiments with the two-tank system in this thesis.

Table B.1.: APRBS parameters for control of the two-tank system

Signal Type	Length	Max. Hold Time	Range
Training signal	5000	30	[0.5, 5]
Validation signal	10000	30	[0.5, 5]
Test signal	20000	30	[0.5, 5]

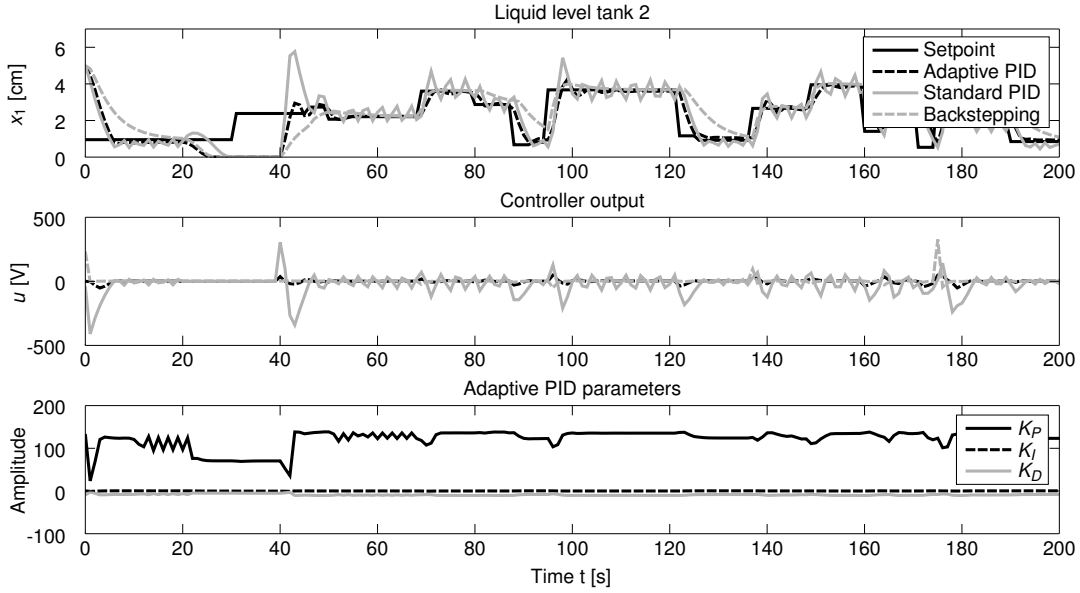


Figure B.1.: Control of the two-tank system with disturbance

B.2. Control Results for the LTI System with Input Delay

Table B.2 shows the APRBS parameters used for the LTI system with input delay, figure B.2 the corresponding control results including the results of the standard PID controller and the Smith predictor as described in section 6.3.1. The sample time was 0.1 seconds for all experiments on the LTI system with input delay in this thesis.

Table B.2.: APRBS parameters for control of the LTI system with input delay

Signal Type	Length	Max. Hold Time	Range
Training signal	2000	40	$[-1, 1]$
Validation signal	3000	40	$[-1, 1]$
Test signal	5000	50	$[-1, 1]$

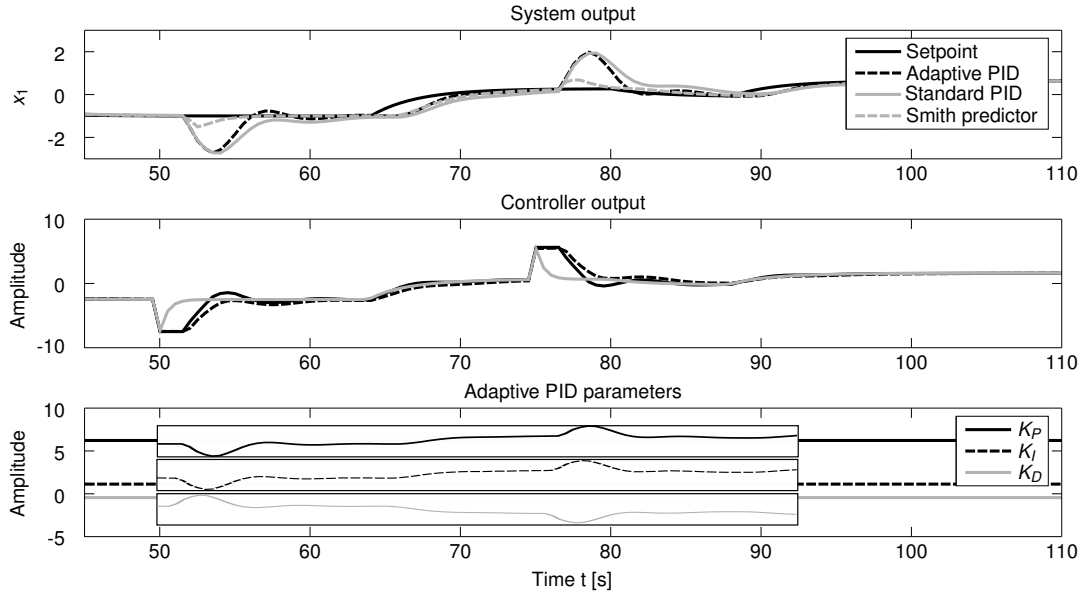


Figure B.2.: Control of the LTI system with input delay with disturbance

It is important to note that the PID parameters for the LTI system with input delay were only adapted in a very small range by the neural network (though making a difference in control performance compared to the fixed PID controller). To illustrate this adaptation, the parameter development shown in figure B.2 was zoomed for each PID parameter. The parameters varied from 6.21918 to 6.2255 for K_P, from 1.1285 to 1.13003 for K_I and from -0.44297 to -0.441411 for K_D.