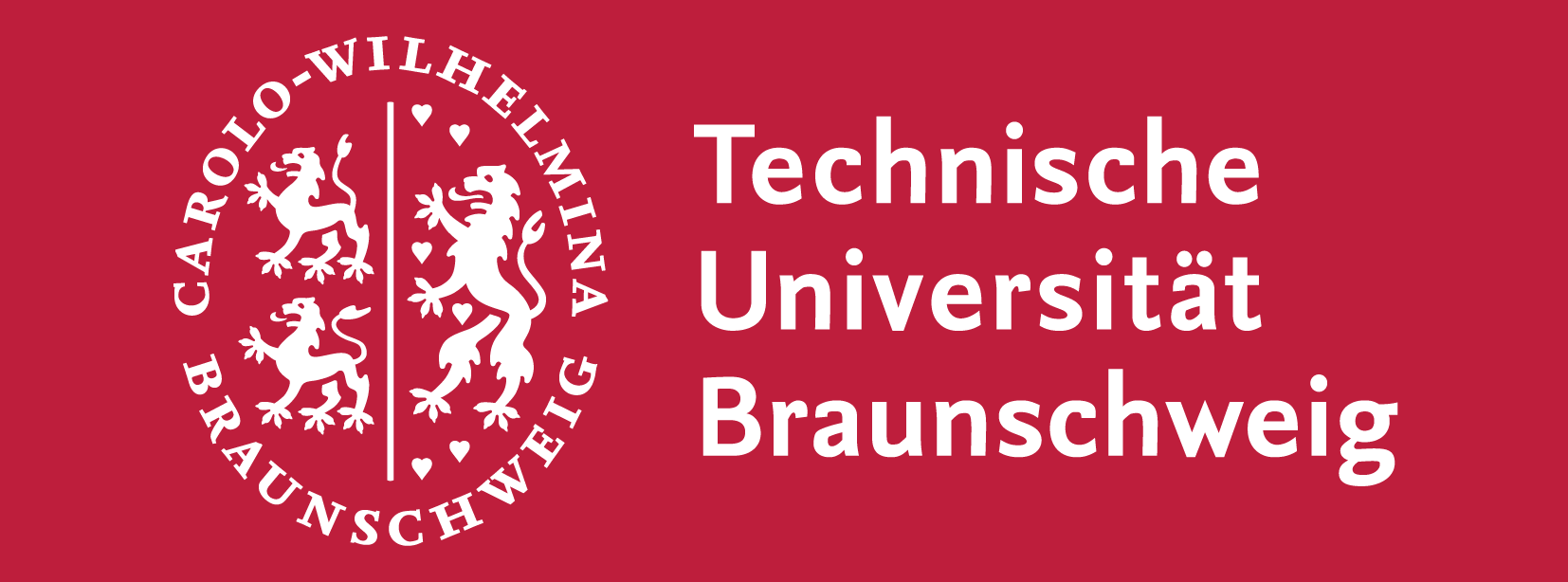
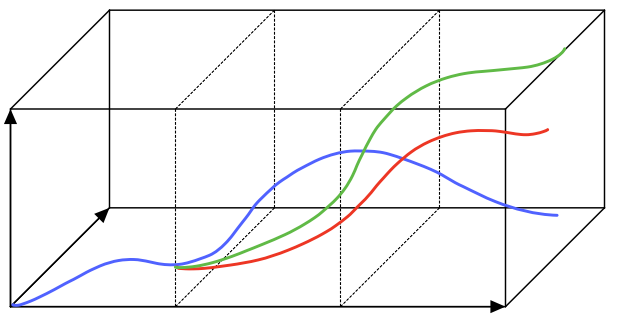
 [Translated from German to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)



ICOMPLEMENTATION OF AADAPTIVES

AUTONOMISTSPPROCESS CONTROL BY MEANS

PRADICTIVEAI-MMODELING



Control

Target

Time

BACHELOR THESIS

NUMERICAL

MADE BY:

JONAS BRINKMANN, 5005717

Braunschweig, NOVEMBER 2022

Supervisor: m. sc. Christoph Thon

First examiner: PROF. DR.-ING. Carsten Schilde

Second examiner: Prof. Dr.-ING. Arno Kwade

**statutory declaration**

I, Jonas Brinkmann, born on March 10, 2000, hereby declare that I wrote this bachelor's thesis independently and only used the tools and sources specified. All literal or analogous quotations taken from published or unpublished writings are marked as such.

Braunschweig,

Jonas Brinkmann

# Table of contents

[Table of contents IV](#_Toc126151981)

[List of abbreviations VII](#_Toc126151982)

[Symbol directory VIII](#_Toc126151983)

[Greek symbols VIII](#_Toc126151984)

[Latin symbols VIII](#_Toc126151985)

[1 Introduction and motivation 11](#_Toc126151986)

[2 Theoretical foundations 12](#_Toc126151987)

[2.1 Process control 12](#_Toc126151988)

[2.1.1 process 12](#_Toc126151989)

[2.1.2 control loop 13](#_Toc126151990)

[2.1.3 Transmission behavior 14](#_Toc126151991)

[2.1.4 transmission links 16](#_Toc126151992)

[2.2 Artificial intelligence 20](#_Toc126151993)

[2.2.1 Machine learning 21](#_Toc126151994)

[2.2.2 Learning methods 21](#_Toc126151995)

[2.2.3 Artificial neural networks 24](#_Toc126151996)

[2.3 Error calculation in prediction 29](#_Toc126151997)

[2.3.1 Calculation during classification 30](#_Toc126151998)

[2.3.2 Calculation in regression 31](#_Toc126151999)

[2.4 Predictive modeling over a sliding time window 33](#_Toc126152000)

[2.4.1 Methods for forecasting over multiple time steps 35](#_Toc126152001)

[2.5 Model Predictive Control (MPC) 36](#_Toc126152002)

[2.5.1 Cost function and additional conditions 37](#_Toc126152003)

[2.5.2 Optimal control strategy 39](#_Toc126152004)

[3 implementation 41](#_Toc126152005)

[3.1 Transforming a time series into supervised learning 43](#_Toc126152006)

[3.2 Rules AI 46](#_Toc126152007)

[3.3 Calculation of the optimal control strategy 49](#_Toc126152008)

[3.4 Virtual system 50](#_Toc126152009)

[3.5 Data evaluation and visualization 53](#_Toc126152010)

[4 Results and discussion 55](#_Toc126152011)

[4.1 Problem with the ReLU activation function 55](#_Toc126152012)

[4.2 Control of a static system 57](#_Toc126152013)

[4.2.1 Influence of the window size on the control success 57](#_Toc126152014)

[4.2.2 Influence of the process parameters on the control success 66](#_Toc126152015)

[4.2.3 Influence of disruptions on the control success 67](#_Toc126152016)

[4.2.4 Effect of the secondary condition Δumax on the control success 73](#_Toc126152017)

[4.2.5 Classification of the influencing factors on the success of the regulation 74](#_Toc126152018)

[4.3 Control of a dynamic system 76](#_Toc126152019)

[4.3.1 Change in the reference variable 76](#_Toc126152020)

[4.3.2 Dynamic state variables k1 and k2 77](#_Toc126152021)

[4.3.3 Change in the reference variable and dynamic state variables 81](#_Toc126152022)

[4.4 Transfer to a real process 85](#_Toc126152023)

[5 Summary and Outlook 87](#_Toc126152024)

[6 bibliography 89](#_Toc126152025)

[7 List of illustrations 93](#_Toc126152026)

[8th Table directory 96](#_Toc126152027)

[9 Attachment 97](#_Toc126152028)

[9.1 Python script from “main.py” 97](#_Toc126152029)

[9.2 Python script of the rule AI from “control\_ai.py” 99](#_Toc126152030)

[9.3 Python script from “utils.py” 103](#_Toc126152031)

[9.4 Python script of the simulation from “simulation.py” 105](#_Toc126152032)

[9.5 Python script for visualizing the results from “export.py” 108](#_Toc126152033)

# List of abbreviations

|  |  |
| --- | --- |
| abbreviation | Explanation |
| AI | Artificial Intelligence |
| ML | Machine learning / machine learning |
| DL | Deep learning |
| ANN | Artificial Neural Network |
| CNN | Convolutional Neural Network |
| FNN | Feedforward Neural Network |
| HMM | Hidden Markov Model |
| HHMM | Hierarchical Hidden Markov Model |
| LSTM | Long Short Term Memory |
| AI | Artificial intelligence |
| KNN | Artificial neural network |
| SVM | Support Vector Machines |
| ReLU | Rectified Linear Unit |
| RP | Really positive |
| FP | False positive |
| FN | False negative |
| RN | Really negative |
| MSE | Mean Square Error |
| RMSE | Root Mean Square Error |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| MdAPE | Median Absolute Percentage Error |
| MPC | Model Predictive Control |
| HyREN | Hybrid Regression Evolutionary Network |

# Symbol directory

## Greek symbols

|  |  |  |
| --- | --- | --- |
| abbreviation | Unit | Explanation |
|  | - | Jump function |
|  | - | Jump height |
|  | - | Dirac impulse |
|  | - | Weight for predicted error |
|  | - | Basis of the exponential weight function |
|  | - | Weight for control increments |
|  | % | Probability of a random action |
|  | - | Multiplier to reduce |
|  | - | Standard deviation of random noise |

## Latin symbols

|  |  |  |
| --- | --- | --- |
| abbreviation | Unit | Explanation |
|  | - | controlled variable |
|  | - | Deviation from the rule |
|  | - | Disturbance variable |
|  | - | manipulated variable |
|  | - | Leadership size |
|  | - | Static reinforcement of a system |
|  | s | Time constant |
|  | - | Damping factor |
|  | - | Input values ​​of a neuron |
|  | - | Output value of a neuron |
|  | - | Bias of a neuron |
|  | - | Weight of an input value of the neuron |
|  | - | Layer of an ANN |
|  | - | Activation function of the layerof a KNN |
|  | - | Absolute deviation of a prediction |
|  | - | Actual value at time |
|  | - | Predicted value for the point in time |
|  | - | Forecast horizon |
|  | % | Percentage error of a prediction at time |
|  | - | Number of past times used for the prediction |
|  | - | Rule horizon |
|  | - | Predicted control deviation for point in time |
|  | - | Total costs of a control strategy |
|  | - | Smallest possible value of the manipulated variable |
|  | - | Largest possible value of the manipulated variable |
|  | - | Minimum rate of change of the manipulated variable |
|  | - | Maximum rate of change of the manipulated variable |
|  | - | Number of time steps necessary for conversion into time windows |
|  | ms | control interval |
|  | ms | Training interval |
|  | ms | Size of the time window |
|  | - | Number of new data per training iteration |
|  | - | Set of all possible discrete values ​​for the manipulated variable |
|  | - | Matrix that contains all possible control strategies |
|  | - | Value at which the gradient of is minimum for the virtual system |
|  | - | Internal state variable of the virtual system |
|  | - | Internal state variable of the virtual system |
|  | - | Random system noise |
|  | - | Size of a discrete disturbance pulse |
|  | - | Dynamic disturbance behavior of the system |
|  |  |  |

# Introduction and motivation

In today's industrial production, it is crucial to control processes efficiently and reliably to ensure high productivity and quality. Process control plays an important role here as it helps to ensure the stability of the process and minimize the influence of disruptions[1]. In process engineering in particular, there are many complex processes that are difficult to regulate. Manually implemented, static control concepts are often costly and inflexible when systematic changes occur. An ideal control system should be able to adapt independently to the requirements of the process, to regulate the process as optimally as possible without prior knowledge of the system properties and to be transferable to different systems without much effort. The availability of higher computing power and advances in machine learning have led to a sharp increase in interest in automating controller design and adaptation based on historical process data[2].

In this work, a control AI is developed that uses the Model Predictive Control (MPC) control method to predict in real time the effects of different control strategies on a process over a certain time horizon and to identify the optimal strategy[3]. An artificial neural network is used as a model, which, through continuous training with a sliding time window, is able to improve the prediction accuracy and thus the control success over time and to adapt to dynamic changes in the system properties. Finally, the implemented control AI is tested on a virtual proxy system in various scenarios and it is discussed whether the application of the control AI to a real process is possible.

# **Theoretical foundations**

## Process control

The task of process control is to ensure the stability of a dynamic process and to minimize the influence of disturbances[1]. This is achieved by influencing the process from outside so that it runs according to a desired specification[4].

### process

A process in the process engineering context is a dynamic system that changes due to certain input and disturbance variables and is described by internal state variables such as temperature, pressure or concentration. The internal state variables show the changes in the system over time. The system's reaction to the input and disturbance variables is determined by the output variables. The effect of an input variable on a specific output variable is usually delayed and can be nonlinear. This must be taken into account when designing a controller. A dynamic system is also known as a controlled system. The representation of a process in the block diagram is inIllustration1shown.[1, 4]

**process**

Input variables

Disturbances

Output variables

Illustration1: Block diagram of a process

A process can be, for example, a water heater in which the internal state variables are the temperature and the flow of the water. The input variables are the position of the temperature controller and the ambient temperature, which influence the flow rate and thus the temperature of the heater. The temperature change when adjusting the temperature controller is delayed. A disturbance could be the fluctuation of the room temperature, for example by opening a window. In this case, an output variable could be the measured temperature of the heater.[4]

### control loop

The goal of a control loop is to approximate a controlled variable of a process or a controlled system as closely as possible to a specified reference variable. To do this, the controlled variable must either be measured directly or calculated from other measured variables.

manipulated variable

**Regulator**

**Controlled system**

Leadership size

Deviation from the rule

Disturbance variable

controlled variable

Illustration2: Structure of a control loop. Created based on[4]

Illustration2shows the structure of a control loop. In its simplified form, it consists of a controller and the route to be controlled, which is influenced by the manipulated variable and the disturbance variable. The controlled variable is fed back negatively and compared with the reference variable. The difference between the controlled variable and the reference variable is the control deviation, which should be minimized by a suitable controller.[4]

|  |  |
| --- | --- |
|  | (1) |

Depending on the control deviation, the controller then determines the input variable or the manipulated variable, which in turn changes the process state. The reaction of a controlled variable to the change in the manipulated variable is described by the transmission behavior of a system.[4]

|  |  |
| --- | --- |
|  | (2) |

Finding a suitable control law for the given process is one of the main tasks of a control engineer[4].

An example of a control loop is driving a car. In this case, the person is the controller, the car is the system to be regulated and the speed is the controlled variable that is measured and displayed on the speedometer. The manipulated variable here is the angle of the accelerator pedal, which influences the speed of the car. The driver's goal is to reach or maintain a certain speed. If the car is driving too slowly, the driver must press the accelerator pedal further to increase the speed. If the vehicle is traveling too fast, the driver must ease off the accelerator pedal to reduce the speed. In this way the desired speed is achieved and maintained.

### Transmission behavior

The transfer behavior of a system describes the reaction of the output variable to a change in the input variable and can be examined, for example, using the step and impulse response. Step and impulse response analysis can help predict the dynamic behavior of a system, assess the stability of the system, and understand the system's response to disturbances. The transmission behavior can be modeled using so-called transmission elements.[4]

**Step answer:**The step response describes the time response of the system to a sudden change in the manipulated variable. It can be used to develop and optimize control strategies and thus to improve system behavior. The manipulated variable is represented by the jump function with the jump height.[4]

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |

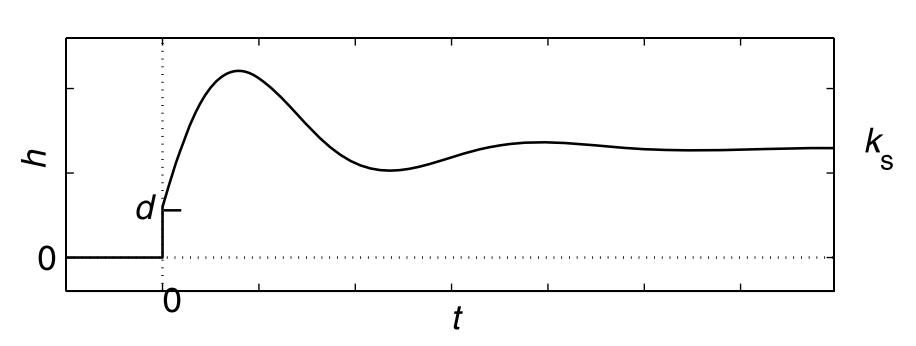


Illustration3: Step response of a second-order system[4]

The step response of a step function with the step height is called the transition function.Illustration3shows an example of a step response or transition function of a second-order system. It can be seen that the transition function for converges to a certain limit, which represents the static gain of the system. The static gain indicates how strongly the system reacts to a change in the input signal. The value of indicates the extent to which the output variable follows the input variable immediately and without delay. Systems for which is are called jumpable systems.[4]

**Impulse response:**The impulse response describes the time response of the system to an impulse-like change in the input variable. It is often used to understand how the system responds to disturbances. The system is excited by a short pulse, which can be represented by the square wave pulse.[4]

|  |  |
| --- | --- |
|  | (5) |

For each, the area of ​​the rectangular momentum is equal to one. A system is excited with a Dirac pulse, which results from the square-wave pulse for.[4]

|  |  |
| --- | --- |
|  | (6) |



Illustration4: Representation of the square wave pulse and the Dirac pulse[4]

The Dirac momentum is the derivative of the step function and is infinitely large and infinitely short. It is represented graphically by an arrow of length one. In reality, the Dirac pulse cannot be implemented, but similar system behavior can be achieved with the square-wave pulse for a small one.[4]

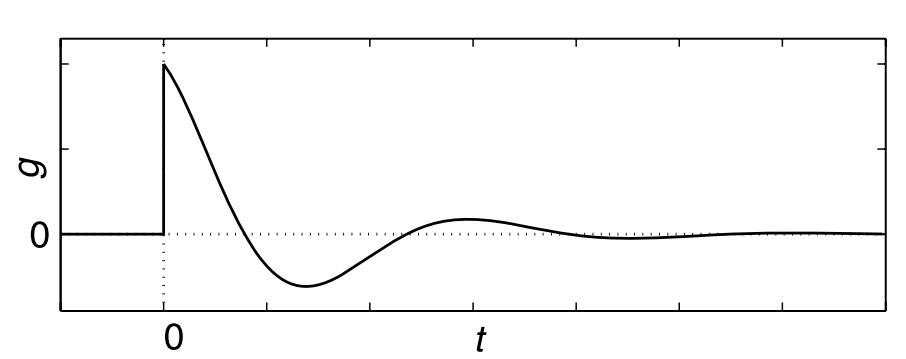


Illustration5: Impulse response of a second-order system[4]

The impulse response of a system excited by the Dirac pulse is also called the weight function.Illustration5shows an example of an impulse response or weight function. A stable system returns to its initial state with a time delay after excitation with a pulse.

### transmission links

Transfer elements are used to model the reaction of output variables of a dynamic system to changes in input variables. They are shown as blocks in structural images and can be divided into proportional, integrating, differentiating and dead time elements according to the qualitative course of their step response. The type of transmission element determines the behavior of the modeled system when the input variable changes.[4, 5]

**Proportional elements (P elements):**Proportional elements are dynamic transmission elements which, with a constant input variable, have an output variable proportional to the value of the input variable in the stationary state.[4]

|  |  |
| --- | --- |
|  | (7) |

Proportional terms can be divided into delayed and instantaneous terms. A P-element without delay is a static transmission element in which the output signal at time t is times as large as the input signal. A P element with a delay is called a PTn element, where n indicates the order of the system. Thus, the PT1 element is a first-order delay element and the PT2 element is a second-order delay element.[4]

In contrast to the instantaneous P element, with PTn elements the output variable is only -times as large as the input variable. The output variable does not reach the final value immediately, but only after a delay. PTn elements can also be implemented by connecting n PT1 elements in series. The more elements are connected in series, the longer the input signal is delayed and the smaller the amplitude of the impulse response.[4]

Table1: Functional relationships and block symbols of proportional elements[4, 5]

|  |  |  |
| --- | --- | --- |
| **transmission link** | **Functional relationship** | **Block symbol** |
| P-member |  |  |
| PT1 link |  |  |
| PT2 link |  |  |

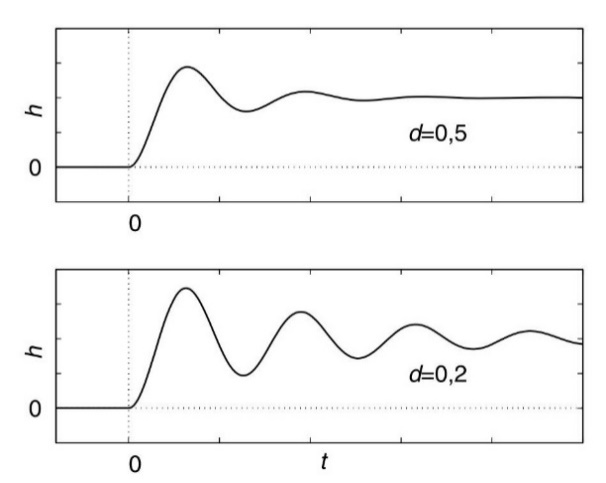


Illustration6: Step response of oscillatory PT2 elements with small attenuation[4]

In delay elements there is a time constant that influences the delay time and is a damping factor. The variation of and can lead to very different step responses. The influence of the damping factor on the transition function of a PT2 element is inIllustration6shown. If the damping factor is in the range , an overshoot of the step response occurs. The smaller the damping factor, the greater the overshoot.[4]

**Integrating terms (I-terms):**With integrating elements, the output variable is determined by integrating the input variable. If the input variable is constant, the output variable takes the form of a ramp function. The output variable only assumes a constant value when the input variable is equal to zero. With stationary system behavior, the output variable is proportional to the integrated input variable.[4]

|  |  |
| --- | --- |
|  | (8) |

Table2: Functional relationship and block symbol of the I-term[4, 5]

|  |  |  |
| --- | --- | --- |
| **transmission link** | **Functional relationship** | **Block symbol** |
| I-member |  |  |

**Differentiating elements (D elements):**With differentiating elements, the output variable is determined by the change in the input variable. With a constant input variable, the output variable therefore converges to zero. With stationary system behavior, the output variable is proportional to the derived input variable.[4]

|  |  |
| --- | --- |
|  | (9) |

The step response of the instantaneous D element contains a Dirac pulse and is therefore only theoretically possible. In real systems, D behavior only occurs with a delay. The delayed D element is referred to as the DTn element, where n describes the number of delay elements. A DT1 element is additionally damped by a PT1 element.[4]

Table3: Functional relationships and block symbols of differentiating elements[4, 5]

|  |  |  |
| --- | --- | --- |
| **transmission link** | **Functional relationship** | **Block symbol** |
| D-link |  |  |
| DT1 link |  |  |

**Dead time element (Tt element):**A dead time element shifts the input signal on the time axis by . They are usually combined with other transmission links.[4]

Table4: Functional relationship and block symbol of the dead time element[4, 5]

|  |  |  |
| --- | --- | --- |
| **transmission link** | **Functional relationship** | **Block symbol** |
| Tt -member |  |  |

## Artificial intelligence

The term artificial intelligence (AI) was first defined in 1956 at the “Dartmouth Summer Research Project on Artificial Intelligence” conference[6]. This conference marked the beginning of AI research and brought together leading scientists to explore the potential of “intelligent machines”.[6]. An early practical application of AI was Newell and Simon's 1956 Logic Theorist[7]. This was a program that was able to independently prove some theorems from Whitehead and Russel's “Principia Mathematica”.[7]. Since then, the field of AI has continued to evolve, with periods of euphoria and periods of disappointment. This development is inIllustration7shown.

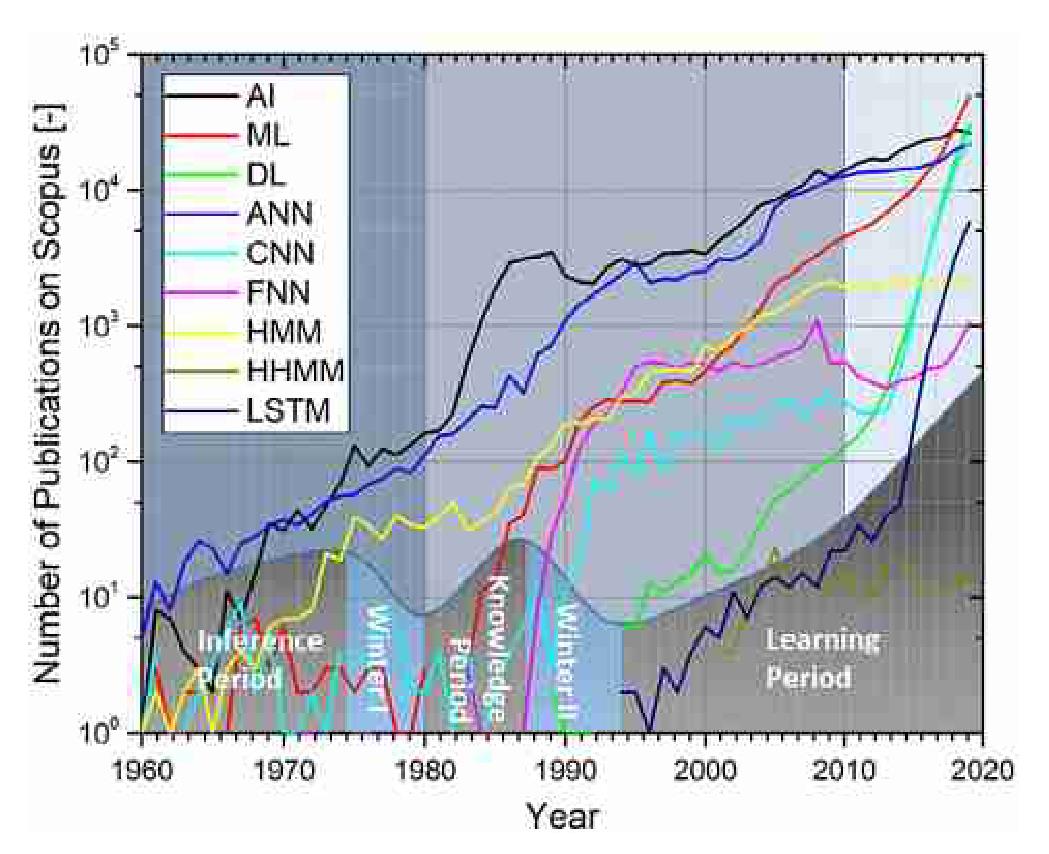


Illustration7: Number of publications of different ML methods on Scopus and qualitative representation of the respective AI eras over time[8th]

One of the most widely used methods of AI is machine learning and deep learning, which uses artificial neural networks as a model[8th]. The following chapters explain machine learning and the sub-learning techniques that can be used to train a model. The construction of artificial neural networks is described in Section2.2.3explained in more detail.

### Machine learning

According to Arthur Samuel, machine learning is the field that gives computers the ability to learn and solve problems on their own without being explicitly programmed to do so. Arthur Samuel became famous in 1959 with a program that was able to learn the game of “checkers” independently and, after a short training period, surpass the developer in this game.[9]

The goal of machine learning is to learn from existing data and generalize new data as efficiently as possible. A model that can generalize effectively is able to make correct predictions on unknown data that it has not seen during the learning process. Machine learning is used when no relevant information and connections can be derived from the existing data, for example by an expert. Its use is particularly advantageous for complex, nonlinear problems. There are many different algorithms in the field of machine learning, the selection of which depends on the respective problem. There is no universally valid algorithm, so it must be checked in advance which one is suitable for the problem at hand. A complex model often requires large amounts of data and high computing power, which are increasingly available due to the exponential growth in data volumes and computing and storage power over the last decade. Interest in machine learning has therefore increased significantly in recent years.[8, 10, 11]

### Learning methods

Machine learning can be divided into three main learning methods: supervised learning, unsupervised learning and reinforcement learning.[10]. The different learning methods with the respective algorithms are inIllustration8and are explained in this section.

Illustration8: Classes of machine learning with the respective algorithms. Created based on[8, 12, 13].

**Supervised learning:**The goal of supervised learning is to learn a function that maps specific inputs to associated outputs. This requires labeled data sets for which the associated outputs are already known. Before training, the data is divided into a training data set and a test data set to later check the accuracy of the learned function. Among other things, classification and regression are assigned to this learning method. Classification is a problem in which the output is discrete, meaning there is only one solution from a known solution space. It is used, for example, in image recognition. Regression is used when the output needs to be continuous and numerical, for example in weather forecasting. Supervised learning is used for predictions based on historical data and is used by about 70% of all AI programs.[8, 10, 11, 13]

An example of the application of supervised learning is the MNIST dataset, which contains handwritten numbers from zero to nine represented by a 28 × 28 pixel grayscale image. Each image is labeled with the corresponding number. It is therefore a classification problem in which a function is to be learned that assigns a number between zero and nine to the 28 × 28 pixels (784 inputs) with different gray levels.[14]

**Unsupervised learning:**Unlike supervised learning, unsupervised learning does not require labeled data, reducing the additional effort for experts. With this learning method, the algorithm must explore the data independently to recognize inherent connections.[8, 10]

Unsupervised learning is mainly used for “clustering” and “dimension reduction”. Clustering involves grouping data with similar structures and properties. Dimensionality reduction, on the other hand, attempts to reduce the dimensions of the input data by filtering out inputs that have little or no influence on the output.[8, 10, 11]

**Reinforcement learning:**In reinforcement learning, a learning agent carries out actions and receives feedback from the environment about how these actions affect goal achievement.[15]

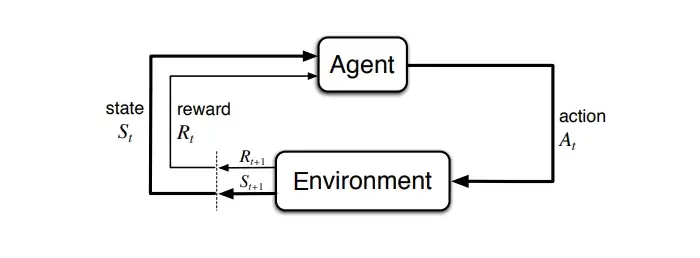


Illustration9: Reinforcement learning process[16]

The agent is programmed to achieve a specific goal using rewards and punishments, without explicitly specifying how the goal should be achieved. To do this, the algorithm must independently learn the optimal actions for the given situations through trial and error. This creates a conflict between “exploration” and “exploitation”. To get a high reward, the agent should exploit the successful strategies. However, to develop and improve successful strategies, the agent must try new actions, with the risk of undesirable outcomes. Another challenge with reinforcement learning is that in most complex systems, actions affect not only the next reward, but also future states and thus future rewards. An action that does not seem immediately effective at first glance may be necessary in the future to achieve the goal, which is difficult to determine in retrospect. Reinforcement learning is based on a continuous flow of new training data and is therefore also referred to as online learning.[8, 15, 17]

There are two different approaches to solving a reinforcement learning problem. The first approach is to select and develop the behaviors that are most effective in the environment. This approach is used, for example, by genetic algorithms, which work similarly to Darwinian evolution theory in biology. The second option is to use statistical techniques and dynamic programming to evaluate the utility of various actions in the environment at any point in time and, among possible strategies, to execute the one that is likely to bring the best future reward.[17, 18]

### Artificial neural networks

The most commonly used method in machine learning is the artificial neural network (ANN). The way an ANN works is loosely based on the understanding of the human brain from the 1960s[19]. The following section explains how ANNs work and addresses various activation functions as well as the backpropagation algorithm (error feedback).

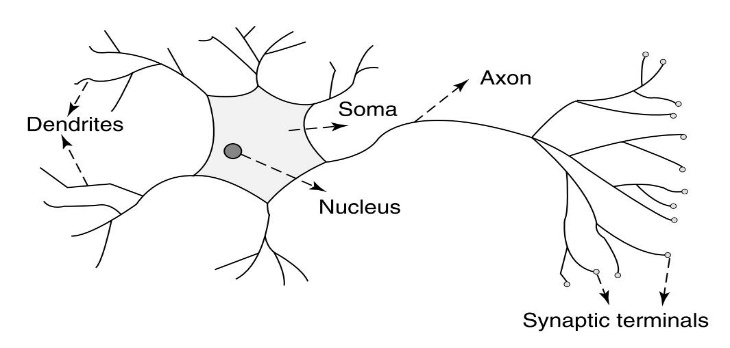


Illustration10: neuron of a mammal[20]

The human brain is made up of over 80 billion interconnected neurons, each neuron being a cell that can receive, process and send information through biochemical reactions. Each neuron has an axon that connects the cell to other neurons through axon terminals. The transmission of signals between neurons occurs through a complex chemical process.[20]

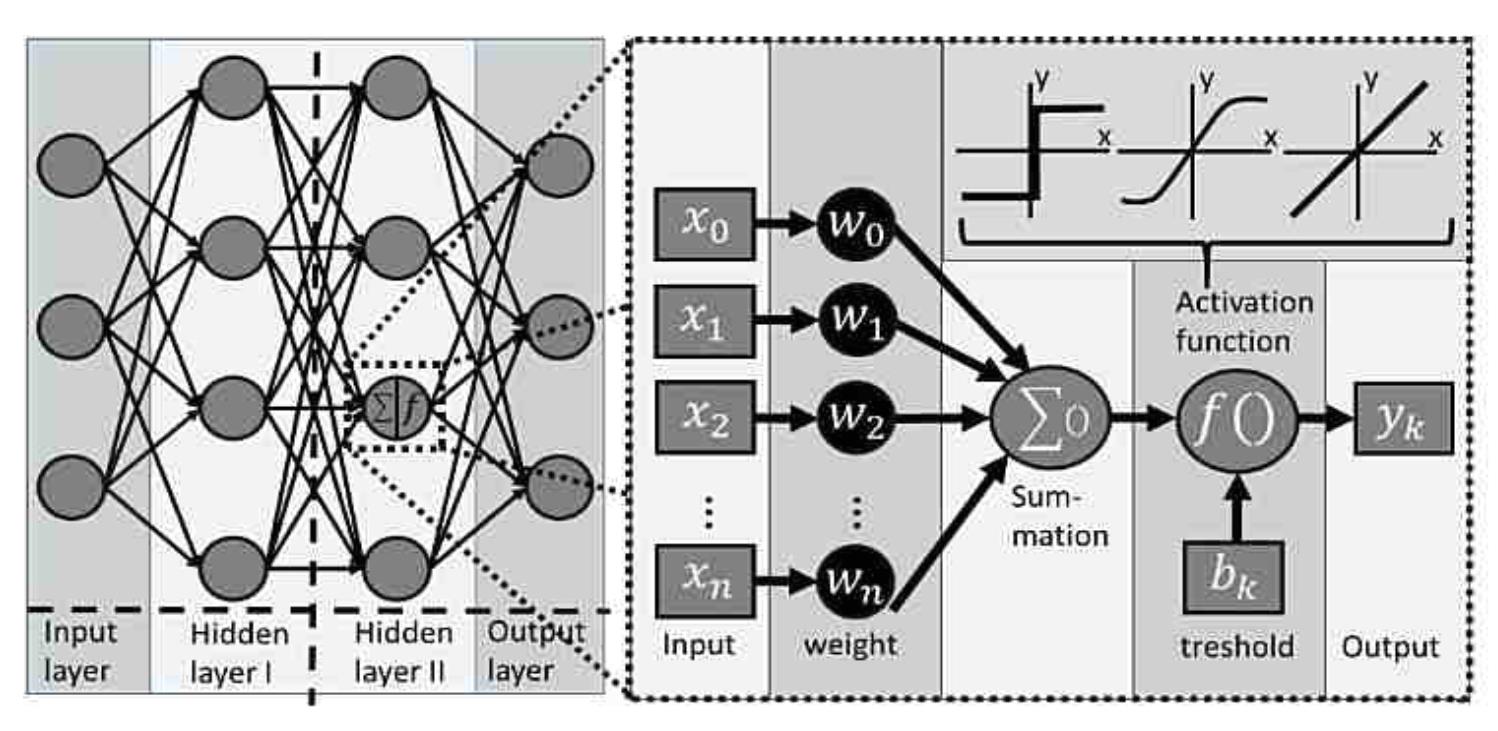


Illustration11: Left: Representation of an artificial neural network. Right: Representation of a single neuron with the weighted inputs and the activation function.[18]

Artificial neural networks consist of neurons arranged in layers and connected to each other, with each connection having a different weight. The first layer of the network is called the input layer and the last is called the output layer. Each input neuron represents an input parameter and each output neuron represents an output parameter. Input parameters can be numeric or binary values, for example process parameters or the color of a pixel in an image. Layers that lie between the input layer and the output layer are called hidden layers. The threshold at which a neuron becomes active is determined by a so-called activation function. It uses the summed and weighted inputs of the neuron and a constant “bias” to determine whether a neuron is active and how large its output is. The output is then passed on to the neurons of the next layer.[18, 21]

As inputs, a single neuron from the layer receives the outputs of all neurons from the previous layer[22](please referIllustration11right).

|  |  |
| --- | --- |
|  | (10) |

The output of the neuron in the layer of the ANN is then calculated using the activation function[22]

|  |  |
| --- | --- |
|  | (11) |

where the sum is formed over the weighted inputs of the neuron in the layer. is the output of the neuron from the previous layer and the weight of this connection. is the bias. In vector notation:

|  |  |
| --- | --- |
|  | (12) |

Depending on the activation function, linearities, nonlinearities or other properties can be introduced into the model[18]. They are therefore necessary to recognize complex, non-linear relationships in the data. The choice of activation function depends on the problem and has a major influence on the prediction accuracy of the neural network. The most commonly used activation functions are nonlinear because they are less sensitive to erroneous data than linear activation functions. Well-known examples of nonlinear activation functions are Sigmoid, Tanh and ReLU.[21]The functions are inIllustration12shown.

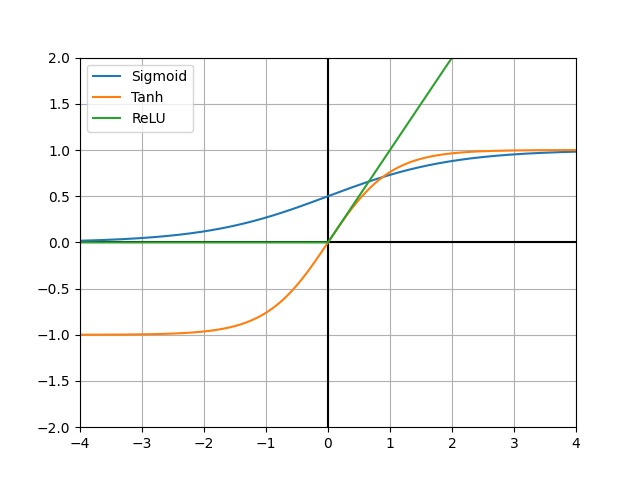


Illustration12: Representation of the Sigmoid, Tanh and ReLU activation functions

**Sigmoid function:**The sigmoid function scales values ​​to the range between zero and one. It is continuous and differentiable, which allows the use of “backpropagation”. The sigmoid function is often used as an activation function to limit the outputs of neurons to this area.[21]

|  |  |
| --- | --- |
|  | (13) |

**Tanh function:**The tangent hyperbolic function is similar to the sigmoid function, but compared to it it is point-symmetrical about the origin. The values ​​of the Tanh function are between negative one and one, which allows negative signs in the outputs. Like the sigmoid function, the Tanh function is continuous and differentiable, which makes “backpropagation” possible. It is often used as an activation function when the outputs of the neurons should also be negative.[21]

|  |  |
| --- | --- |
|  | (14) |

**ReLU function:**The rectified linear unit function is one of the most commonly used activation functions because it has a biological reference to a real neuron and is one of the most efficient activation functions[23]. It works in such a way that the neuron remains inactive for negative values ​​and only becomes active for positive values. This function is more efficient than all other activation functions because not all neurons are active at the same time[21]. However, when using ReLU, neurons may “die” because the gradient for negative values ​​is zero and thus the weights for negative values ​​are not updated through “backpropagation”, which complicates the learning process of the artificial neural network[23]. To get around this problem, you can use the Leaky-ReLU function, for example, which has a small linear component instead of zero for negative values ​​and thus also a gradient that is not zero for negative values. The problem of “dead” neurons does not occur here, but all neurons are active again.

Both ReLU and Leaky ReLU are for continuous and differentiable. Since the derivative for is not defined, it is set either to one or to zero, backpropagation is also possible here.[21, 23]

|  |  |
| --- | --- |
|  | (15) |
|  | (16) |

**Backpropagation (error feedback):**In order for an ANN to learn and improve its prediction accuracy, the weights of the individual connections between the neurons must be adjusted as optimally as possible. For this purpose, the backpropagation algorithm is used, which minimizes a cost or error function by changing the weights of the ANN in a certain way. The error refers to the deviation of the ANN's prediction from the actual value.[22]

Local error minimum

Weight

Mistake

Global error minimum

Reduce weight

Increase weight

Illustration13: Influence of weighting on error. Created based on[24]

To adjust the weights, the partial derivative of the error function with respect to the weighting of the neural network is calculated (). It indicates how quickly the error changes when the weights are changed.Illustration13shows an example of a function of error as a function of weight. The goal of the backpropagation algorithm is to adjust the weights in the direction of the descending gradient or in the direction of the error minimum, where the size of the respective weight change is proportional to the influence of the weight on the error. The weights are adjusted iteratively using the training data. It can happen that the weight is adjusted towards a local minimum, which means that the global optimum is never reached.[22]

## Error calculation in prediction

To assess the accuracy of a model's prediction, a comparison between the predicted value and the actual value is necessary. This chapter explains the error calculation in classification and regression.

### Calculation during classification

With binary classification (positive or negative) there are four different results of a prediction[25].

**True positive (RP):**The model predicted “positive” and the actual value is positive.

**True negative (RN):**The model predicted “negative” and the actual value is negative.

**False positive (FP):**The model predicted “positive” but the actual value is negative.

**False negative (FN):**The model predicted “negative” but the actual value is positive.

Table5: Possible results of a binary classification

|  |  |
| --- | --- |
| **True positive (RP)**  Actually: positive | **False positive (FP)**  Actually: negative |
| **False negative (FN)**  Actually: positive | **True negative (RN)**  Actually: negative |

The accuracy A of the prediction is with(17)calculated[26].

|  |  |
| --- | --- |
|  | (17) |

When working with an imbalanced data set, the overall accuracy may be high, but the model may not necessarily be able to distinguish between different cases. For example, from 100 images you want to identify which one shows an apple. An apple can be seen in 10 pictures, but not in the remaining 90. If every image were classified as “not an apple,” the accuracy would be 90%, even though no distinction is made. Therefore, in most cases the accuracy is not sufficient to make a statement about the effectiveness of the model.[26]

To get around this problem, precision and hit rate are used. Precision is defined as the proportion of positive predictions that were actually correct[27].

|  |  |
| --- | --- |
|  | (18) |

The hit rate (recall) is defined as the proportion of actually positive results that were correctly identified[27].

|  |  |
| --- | --- |
|  | (19) |

In order to make a statement about the efficiency of a model, both the accuracy P and the hit rate R must be taken into account. Both are in conflict with each other. An improvement in precision usually leads to a reduction in the hit rate and vice versa. The aim is to keep precision and hit rate as balanced as possible.[27]

### Calculation in regression

Various error calculation methods are available for regression problems. Which method is ultimately used depends on the problem statement and is another hyperparameter that needs to be tested for the problem at hand[28].

**Scale-dependent errors:**The commonly used methods for error calculation are scale dependent. They are useful for comparing different methods applied to the same data set, but not for comparing different data sets scaled differently. Well-known examples of scale-dependent error calculations are Mean Square Error (MSE), Root Mean Square Error (RMSE) and Mean Absolute Error (MAE).[28, 29]

The absolute prediction error at time is defined as

|  |  |
| --- | --- |
|  | (20) |

where is the actual value at time and is the model's predicted value at time. Over a certain horizon, the total deviation can be determined using various methods.[29]

|  |  |
| --- | --- |
|  | (21) |
|  | (22) |
|  | (23) |

Historically, RMSE and MSE have been popular methods for calculating error due to their relevance in statistical modeling, but have met with criticism when evaluating temporal predictions[28].

**Percentage errors:**The advantage of percentage errors is their scale independence, which is why they are often used to evaluate data sets with different scales. Commonly used methods are Mean Absolute Percentage Error (MAPE) and Median Absolute Percentage Error (MdAPE).[28]

|  |  |
| --- | --- |
|  | (24) |
|  | (25) |
|  | (26) |

However, they are unsuitable for data with small values, especially if the data set contains, since in this case the percentage error is undefined. For values ​​close to zero there are large fluctuations in the error calculation. Therefore, MAPE can be significantly larger than MdAPE in these cases because the median is generally more resistant to outliers than the mean. Another disadvantage is that both methods give more weight to positive errors than to negative ones. This leads to an asymmetry in the calculation.[28]

## Predictive modeling over a sliding time window

Predictive modeling is a form of machine learning whose goal is to predict the future state of a system based on known historical data[30]. Statistical models are usually used for this purpose, but they have difficulties when modeling high-dimensional, nonlinear systems[31]. ANNs that can learn the inherent system properties only with the help of historical data are suitable for modeling and predicting complex nonlinear systems. A sliding time window is used in this work to model time sequences. A time sequence is a sequence of historical measurements of a system state at equal time intervals[30].

**Sliding time window:**The sliding time window method converts sequential data in the form of a time sequence into a classic supervised learning problem. This makes it possible to train an artificial neural network using the backpropagation algorithm. For each time window, the most recent states are used to predict one or more future states. Various methods for forecasting over multiple time steps are discussed in Chapter2.4.1presented.[32]

Window 1

Window 2

y(t) [-]

y(t) [-]

Lay

horizon

time [s]

time [s]

Illustration14: Example of a sliding time window

Illustration14shows an example of a sliding time window. The delay refers to the period of the past that is taken into account for the prediction[30]. The horizon indicates how many future states should be predicted and in this example is equal to one[30]. The first time window is between and , whereby the values ​​for , and are used to predict the value at the time. For the following steps, the time window is shifted by one to the right and this process is repeated.

For the learning process, all values ​​must be known at the respective points in time so that supervised learning can be applied. Each time window then represents an entry in the training data set. With continuous systems, it is possible to further train the model with the new data and thus improve the accuracy of the model over time. The trained model with the last states as input can then be used to predict unknown future states.

### Methods for forecasting over multiple time steps

A multiple time step forecast has the task of predicting the next values ​​(,…,) using the historical time series of the last values ​​(,…, ), where is the forecast horizon. The problem here is that each value has a time dependency on the previous values. For example, the prediction of the value also depends on the value, which is not known but also needs to be predicted. This chapter introduces three different methods to solve this problem. This is the model that maps the past values ​​to the future values ​​and the deviation from the model including disturbances.[33]

**Recursive method:**The recursive method trains a model that can only make a prediction about the next time step[33].

|  |  |
| --- | --- |
|  | (27) |

The next values ​​are then predicted recursively, with the predicted value being used as an input parameter for predicting the following values. This is repeated until the entire horizon is predicted. A disadvantage of this method is the sensitivity to forecast errors, as these are passed on to subsequent forecasts and thus become larger over time.[33]

**Direct method:**In contrast to the recursive method, the direct method trains different models, each of which predicts a future value[33].

|  |  |
| --- | --- |
|  | (28) |

In this method, the predicted values ​​are not used to predict new values, so there is no accumulation of errors. However, a disadvantage of this method is that the complex relationships between the predicted values ​​(, ...,) are not modeled because these values ​​are predicted by different models. In addition, this method is significantly more inefficient and computationally intensive than other methods because different models have to be trained.[33]

**MIMO (Multiple Input Multiple Output) method:**In contrast to the recursive and direct method, the MIMO method enables the output of several values ​​at once, which means that complex relationships between future values ​​can also be modeled in longer-term predictions[33]. A single model is trained that has outputs.

|  |  |
| --- | --- |
|  | (29) |

with the vector function and the noise vector. A disadvantage of this method is that the horizon cannot be changed afterwards without training a completely new model.[33]

## Model Predictive Control (MPC)

MPC is a modern control method that uses predictive modeling to regulate a system or process as optimally as possible. The goal is to predict the reaction of a system to various manipulated variables over a certain time horizon in real time and to find the best control strategy while minimizing a cost function and certain additional conditions. A control strategy is a combination of manipulated variables at discrete times over a specific control horizon. MPC enables the autonomous control of complex processes without expert intervention over a longer period of time and is flexible, which allows it to be applied to different systems with different characteristics.[3, 34]

**Model**

Illustration15: Basic structure of MPC. Created based on[3].

MPC requires a suitable process model in order to be able to predict the future behavior of the process based on past values ​​for manipulated and controlled variables and possible control strategies. The effectiveness of the control therefore depends heavily on the quality of this model. With the future control deviation(30)The optimizer calculates the optimal control strategy taking into account the cost function and certain additional conditions.[3]

### Cost function and additional conditions

The possible control strategies are evaluated using a cost function and, if necessary, additional constraints. The general goal is to select the control strategy in which the predicted values ​​optimally approximate the reference variable. To calculate the total costs, the future control deviations are required at every point in time over the forecast horizon (see formula(1)).[3]

|  |  |
| --- | --- |
|  | (30) |
|  | (31) |

The first term of the cost function is a weighted square sum of the predicted control deviations with the respective weight. The weight can change over the forecast horizon, so values ​​at different points in time have a different impact on costs. For example, an exponential function can be used for this.[3]

|  |  |
| --- | --- |
| with | (32) |

If , predicted errors that are further in the future are weighted higher than the first ones, which leads to a gentler control with less effort. The first errors are weighted higher, which leads to a stricter regulation.[3]

The second term of refers to the control effort that arises when the manipulated variable changes; this could, for example, include energy costs, which should be kept as minimal as possible. If the control effort is not significant for the problem, the second term can be ignored.[3]

In addition to the cost function, certain additional conditions are also taken into account in practice. For example, the actuator only has a limited range of action and a maximum slew rate, which must be taken into account in the various control strategies. For example, a valve is fully open or closed by its position and its opening and closing speed is limited. Additional conditions can also arise from safety or environmental requirements where process parameters such as temperature or pressure must be limited. In most cases there will be a lower and upper limit of(33), a limit on the rate of increase of(34)and a limitation of(35)taken into account.[3]

|  |  |
| --- | --- |
|  | (33) |
|  | (34) |
|  | (35) |

### Optimal control strategy

The optimal control strategy is the strategy with the lowest total costs that also fulfills all additional conditions. In order to identify these, all possible control strategies must first be created. For this purpose, all possible combinations of the manipulated variables are formed at discrete times over a control horizon. For each control strategy, the future values ​​of the controlled variable are then predicted over the forecast horizon using the model. These predictions are then used to determine the associated costs and the one that causes the lowest costs and meets additional constraints is selected from all strategies. From the optimal control strategy at the time, only the first control signal is then applied. The calculation of the optimal control strategy is repeated for each time step.[34]

**…**

**…**

**…**

Reference variable w(t)

Output size

manipulated variable u(t)

Rule horizon

Forecast horizon

Control deviation e(t)

Illustration16: Prediction of a control strategy over a sliding time window in MPC

Illustration16shows an example of predicting a control strategy. A sliding time window is used, except that the manipulated variable is used for modeling in addition to the system state. In addition to the actual forecast horizon, a control horizon is introduced over which all possible control strategies are formed. The strategies are then evaluated over the actual forecast horizon. Since the system reacts to a manipulated variable with a time delay, the forecast horizon should be larger than the control horizon in order to take the time dependencies into account. In addition, the control horizon can be chosen to be smaller in order to reduce the computational effort.

# implementation

This chapter describes how to implement automatic process control using AI and predictive modeling in Python. In addition, a virtual proxy system will be implemented to test the efficiency of the AI ​​control.

The goal of control AI is to approximate the system's controlled variable to a specified reference value using MPC. An ANN is used as a predictive model in this work, which is intended to learn the connection between past manipulated and controlled variables through a sliding time window and can make predictions about future system states. During the virtual process, the ANN is continuously trained with new data in order to improve the prediction accuracy and thus the effectiveness of the control over time. For this purpose, the AI ​​framework “HyREN” (Hybrid Regression Evolutionary Network) developed at iPAT was used in the first version, which uses a genetic algorithm to train an ANN[18]. However, this method requires more computing power than classic supervised learning with the backpropagation algorithm, which means the entire training process takes significantly longer. When simulating a virtual system, a longer training period is not necessarily a disadvantage, as in this case the process does not continue until the training of the rule AI is completed. However, when transferring the rule AI to a real process, it must be taken into account that the process continues to run during training. It is crucial to keep the training time of the control AI as short as possible to enable real-time control. It was found that with the HyREN AI framework, one training iteration takes several minutes. In comparison, supervised learning with the backpropagation algorithm takes less than a second, depending on the choice of hyperparameters such as “batch\_size” and “epochs” and the available computing power. For this reason, HyREN is not used in this work.

The main program that interfaces between the rule AI and the virtual system runs in the main.py file. Both the rule AI and the virtual system are configured in this file. After each time step, the current system state of “main.py” is passed to the rule AI. At regular intervals, the control AI calculates the optimal control strategy using MPC and passes it on to “main.py”. There the first manipulated variable of the optimal control strategy is applied to the virtual system. The virtual system then calculates the next system state depending on the applied manipulated variable and the disturbance variable and returns this to “main.py”. The rule AI is also trained with the new data at regular intervals. The cycle described is repeated until a specified step limit is reached. The algorithm flow is inIllustration17shown.

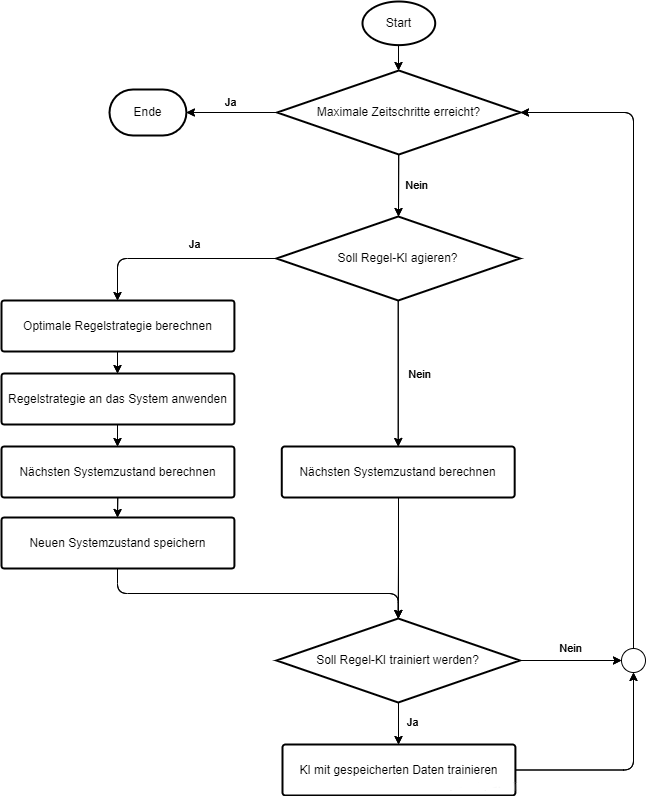


Illustration17: Process of the main program

## Transforming a time series into supervised learning

In order for the rule AI to be trained, the known time series must first be converted into classic supervised learning. For this purpose, the sliding time window method is used, in which the values ​​of previous time steps are used to predict the values ​​of subsequent time steps. An example of a univariate time series, or a time series in which only one variable is considered, is inTable6shown.[35]

Table6: Example of a univariate time series

|  |  |
| --- | --- |
| **Time** | **Value** |
| 1 | 64 |
| 2 | 21 |
| 3 | 45 |
| 4 | 89 |

Table7: Example of converting a univariate time series into supervised learning

|  |  |
| --- | --- |
| **X** | **y** |
| ? | 64 |
| 64 | 21 |
| 21 | 45 |
| 45 | 89 |
| 89 | ? |

Conversion to supervised learning is inTable7shown. X are the values ​​used as inputs to the ANN and y are the values ​​used as outputs to the ANN. The first and last lines cannot be used because a value is missing in each. During the conversion, the column with the values ​​is copied and moved by one row.[35]

In Python, this transformation can be applied to a DataFrame using the .shift(x) function, which shifts all columns of the DataFrame by x rows[36]. For a prediction over multiple time steps, the time series must be copied for each step and moved accordingly. The conversion into training data is done in this program using the “convert\_input\_data\_training()” function from “utils.py”. The function allows you to vary the number of past values, the control horizon and the forecast horizon and thus create different time windows. For the transformation into training data, time steps are required for a time window

|  |  |
| --- | --- |
|  | (36) |

Table8: Multivariate time series with manipulated and controlled variables

|  |  |  |
| --- | --- | --- |
| Time | manipulated variable | controlled variable |
| 0 |  |  |
| 1 |  |  |
| **…** | | |
|  |  |  |
| **…** | | |
|  |  |  |
|  |  |  |
|  |  |  |
| **…** | | |
|  |  |  |
| **…** | | |
|  |  |  |
| **…** | | |
|  |  |  |

…

Present

Future

Past

Inputs

expenditure

…

…

…

Time

…

Illustration18: Structure of a time window at time t with manipulated and controlled variables

To predict the future controlled variable (both the controlled variable and the manipulated variable of the past points in time are required. This means that there is a multivariate time series.Table8shows the recorded time series of the system, in which the manipulated variable and the controlled variable were recorded for each time step. This is the last recorded time. In order for there to be at least one valid time window for training, this must be the case, otherwise there will not be enough values ​​for a time window. For any point in time, a time window can be created that uses past times to predict future times, with all values ​​known. The conversion of the time series intoTable8into a time window at time , with the respective inputs and outputs for the ANN, is inIllustration18shown. The last values ​​of the manipulated and controlled variables, the current manipulated and controlled variables at the time and the next manipulated variables are used as inputs. The red dotted box represents a matrix of all possible control strategies, used only for prediction. It does not play a role when creating training data because the values ​​( are known in this case.

## Rules AI

The main part of the Rules AI is located in the Agent class in the control\_ai.py file. This class stores the ANN, which is used as a predictive model and is built using the static build\_network() function. The ANN consists of an input layer, an output layer and two hidden layers, the number of neurons of which can each be adjusted by the input parameters of the function.

The “Keras” module is used to create an ANN. Keras is a powerful library based on the TensorFlow machine learning platform developed by Google that enables the creation of complex models with minimal effort. Due to its ease of use, Keras is one of the most popular libraries for deep learning.[37]

The “deque” data type from the “collections” module is used to store the time series. This data type is similar to a list, but a maximum size can be defined[38]. If the maximum size is exceeded after inserting a new element, the oldest element is automatically deleted[38]. This property is advantageous in this context because in a continuous process the amount of data can be very high over a longer period of time. Using the “deque” data type prevents the rule AI from running out of storage space after a longer period of time.

In this work this will be discussed in chapter2.4.1The MIMO method described is used to make predictions over multiple time steps because this method is efficient with a high number of predictions. In comparison, the recursive method was initially used, but with a high number of predictions it requires significantly more time because each time step of each control strategy has to be predicted individually. The recursive strategy is therefore not suitable for real-time control. When using the MIMO method, the ANN has as many output neurons as the number of time steps that need to be predicted. The structure of the input and output layers of an ANN that calculates the next points in time using the reshaped time windowIllustration18predicts, is inIllustration19shown. The number of input neurons is with(37)calculated.

|  |  |
| --- | --- |
|  | (37) |

Illustration19: Structure of the input and output layer of the rule AI

The red box in this figure also represents the matrix of possible control strategies. All input values ​​above the box are known for the prediction, as they are past values ​​for the manipulated and controlled variables. As inIllustration15shown, the model predicts the course of the future controlled variable with the past values ​​for manipulated and controlled variables and the future manipulated variables from every possible control strategy.

At the beginning of process control, the control AI is in an exploration phase because it has no data available for training. Similar to reinforcement learning, there is a conflict between exploration and exploitation (see chapter2.2.2). In the initial phase, the rule AI should gain as much experience as possible to gain a solid understanding of the process. Over the course of the process, the prediction accuracy of the ANN used improves as the amount of data increases, so that the control AI should increasingly exploit the best possible control strategy.

To balance exploration and exploitation, the Epsilon-Greedy algorithm is used. The value of Epsilon is a number between zero and one that describes the probability of exploration through a random action. To determine whether the rule AI should explore or exploit, a random number between zero and one is generated. If is, a random action is executed, otherwise the calculated optimal control strategy is executed. A value of  
means 100% random action will be performed while  
means that the calculated optimal control strategy is executed 100%. At the beginning of the process is to explore the system. As the process progresses, reductions should be made continuously to reduce the likelihood of random actions. To do this, after each training iteration of the ANN, the value for is multiplied by a constant, which leads to a reduction of over time. Additionally, a lower bound can be set for so that the probability of exploration never becomes zero.[39]

The rule AI acts at fixed time intervals of milliseconds and is trained at fixed time intervals of milliseconds. In a virtual system, and should be a multiple of the simulation time step size to ensure predictable and consistent behavior of the control AI. The size of the time window can be determined using the equation(38)and the amount of new data per training iteration with the equation(39)be calculated.

|  |  |
| --- | --- |
|  | (38) |
|  | (39) |

## Calculation of the optimal control strategy

To determine the optimal control strategy, it is first necessary to predict the course of the future controlled variable depending on all possible control strategies. To do this, a matrix must first be created that contains all possible control strategies over a specific control horizon. To calculate the matrix, the Cartesian product is used, which can be used to calculate all combinations of two or more quantities. Assuming the quantities and are present, the Cartesian product of A and B results in the quantity .[40]

With the set of possible discrete values ​​for the manipulated variable and the control horizon, all possible control strategies can be calculated

|  |  |
| --- | --- |
|  | (40) |

A matrix is ​​created. For results . It should be noted that the control horizon in this work only takes future manipulated variables () into account. The current manipulated variable is always considered. For the control horizon, only the current manipulated variable is used for prediction. The number of possible control strategies increases exponentially as the number increases. For example, if there are ten possible values ​​for the manipulated variable, this results in 100,000 different control strategies.

In order to list all possible control strategies, the possible discrete values ​​for the manipulated variable must be passed to the control AI in the form of a list at the start. All possible control strategies are then listed in a “numpy.array” using the “compute\_all\_possible\_strategies()” function from the “utils.py” file. The function receives the control horizon and a list of values ​​for the manipulated variable as parameters.

Data for prediction is put into the correct form using the “\_get\_prediction\_data()” function. The function receives the current value of the controlled variable as a parameter because it is not yet available in memory. In this function, the data from the last points in time and the current value of the controlled variable are first converted into a “numpy.array” (,). The row of the list is then copied as many times as there are control strategies so that the array just created can be merged with the array of control strategies so that each row represents the input structureIllustration18has. Finally, the predicted control variables are combined with the cost function(31)rated. In this work, only the secondary condition is used(34)considered, with . In the rule AI, a “du\_max” can be defined in order to avoid large fluctuations in the manipulated variable. Control strategies that do not meet this criterion are filtered out for the prediction. This has the advantage that the number of predictions does not become too large. In most cases, the control strategy with the lowest cost is used. To avoid the rule AI falling into a local minimum, a random rule strategy is sometimes selected from the best.

## Virtual system

In this chapter, the virtual proxy system is implemented, which is used to test the effectiveness of the rule AI. Since the control AI is to be applied to a bioreactor for pH control in a later work, a proxy system is initially used to roughly model the pH value in a bioreactor. This allows the transferability of the rule AI to a real system to be efficiently tested in advance.

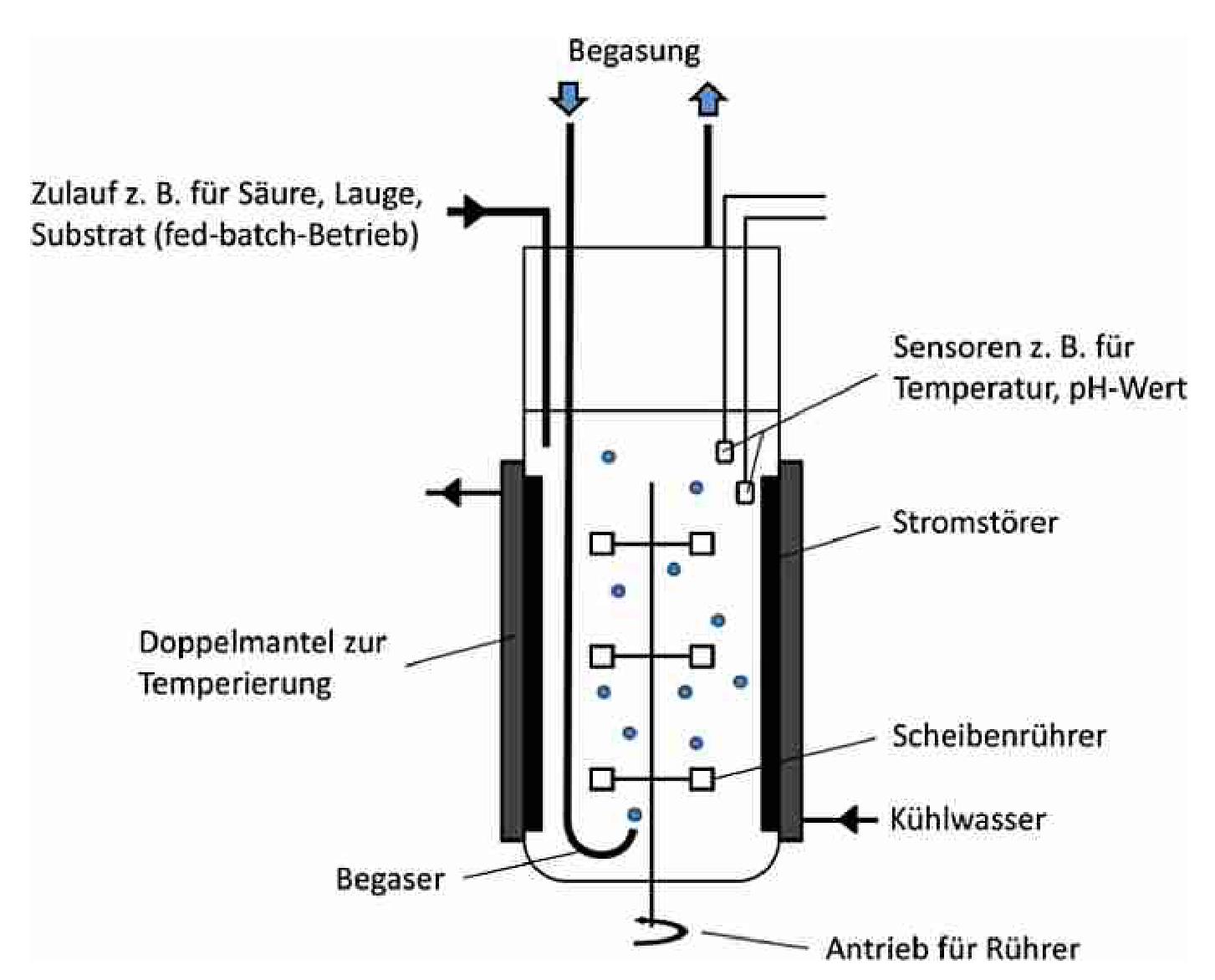


Illustration20: Schematic representation of a bioreactor in the form of a stirred tank[41]

The aim of future work will be to test the effectiveness of control AI in pH control in a bioreactor. In this case, the manipulated variable is represented by the valve position, which influences the inflow rate of an acid or alkali and thus changes the pH value. The controlled variable here is the pH value, which is measured via sensors in the boiler. In this work, a proxy system is used that provides a pH value with equation(41)should be modeled approximately. The proxy system enables rapid implementation and testing of various control procedures.

|  |  |
| --- | --- |
|  | (41) |

Here is the controlled variable of the system, i.e. the pH value, which is influenced by the manipulated variable. and are internal state variables of the system that additionally influence the controlled variable and can change over time. describes the most stable value of the system, where there is no influence and the slope of is minimal. describes the external disturbance variables and is divided into random noise, a possible disturbance impulse and a continuous, dynamic disturbance behavior.

|  |  |
| --- | --- |
|  | (42) |

The random noise is generated by a random number using the “numpy” function “random.normal()” and is intended, among other things, to model the measurement uncertainty. The random number is normally distributed with the expected value and a specified standard deviation. The value for the standard deviation can be passed as a parameter to the simulation before starting. The random value is generated anew in each time step of the simulation. The disturbance pulse can occur at any time in order to test the stability of the control. The dynamic disturbance variable can be specified at the beginning of the simulation as a function of time.

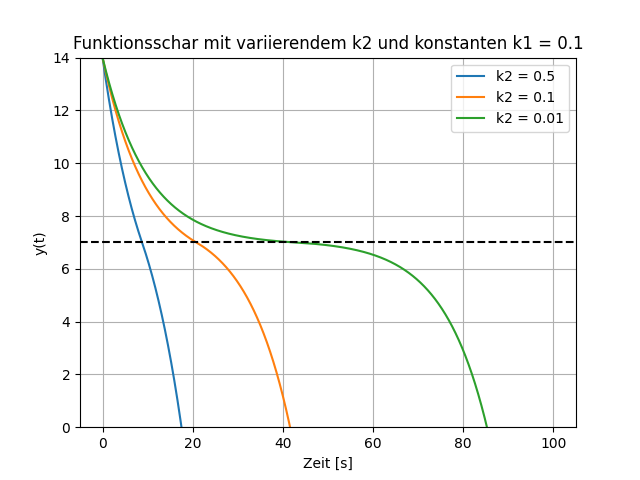
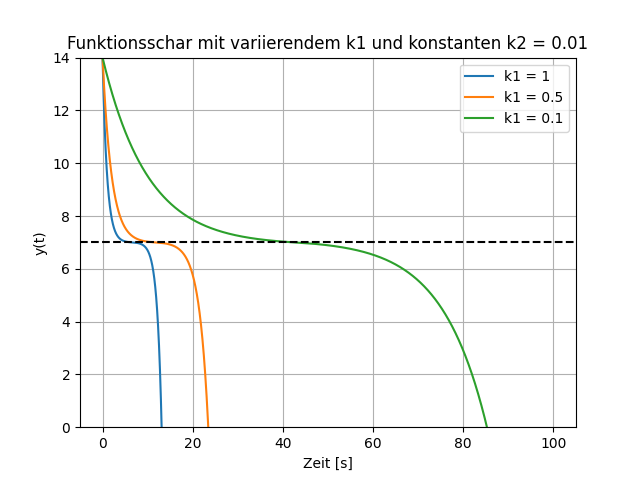


Illustration21: Influence of k1 and k2 on the system with u(t) = 0 and z(t) = 0

The system is limited by to approximately model a pH value.Illustration21shows the influence of and on the system with . It can be seen that the slope depends on the distance to the value. The larger is, the faster the system becomes unstable if it deviates slightly from. The value of describes a constant decrease in the system and influences how stable the system is in the range of. The values ​​of and contain all physical properties that influence pH. For this can be, among other things, the reaction kinetics and for, for example, the diffusion rate. Other physical properties such as temperature and pressure can also influence. The summary in and serves to simplify the proxy system.

The virtual system is simulated in the “simulation.py” file. This file contains the Simulation class in which the virtual system is simulated. The simulation can be configured with the “SimConfig” class, in which the system parameters can be set. The “step()” function is used to calculate the state of the system at the next time step depending on the manipulated variable and the disturbance pulse. The differential equation(41)is used to calculate the change using the “\_calculate\_change()” function, which is then added to the controlled variable. The old system state is then saved in a “dict” object and inserted into a list and the time is increased by. At the end it is ensured that it is within the specified limits. The function returns the saved system state and the recalculated value for.

## Data evaluation and visualization

To evaluate the performance of the control AI, for each control strategy the actual values ​​that the system would take are compared with the differential equation(41)calculated and compared with the predicted values. The actual values ​​( for each control strategy can be calculated using the “calculate\_strategies()” function from “simulation.py”. To calculate the differential equation, the “odeint()” function from the “scipy.integrate” module is used. It is Please note that the calculation of the actual values ​​is only used to check the accuracy of the prediction and these are not used to train the rule AI.

In the “Data” class from the “export.py” file, the error is calculated using the MSE, MAPE and MAE methods (Section2.3.2). The total error is calculated across all control strategies and times. In addition, the total error is calculated across all control strategies at each individual point in time (. The results are visualized graphically using the functions of “export.py” and saved as “.png” files in a folder. The configuration settings of each simulation are stored in the “data " of the main project. The graphs for visualizing the prediction accuracy of the ANN are also saved at regular intervals in the "Graphs" folder. At the end of the simulation, the course of the simulation and the time course of the error over time are saved in the "summary" folder .

# Results and discussion

This chapter deals with the evaluation of the implemented rule AI and the problems that occurred. First, the effectiveness of the control in a static system is examined, whereby the influence of the size of the time window, the process parameters and the disturbances on the control success is examined. In all cases there is random, normally distributed noise, the standard deviation of which is set at the beginning of the simulation.

The control success in a dynamic system is then examined by varying both the reference variable and the state variables over time. Furthermore, the accuracy of the predictive model is examined by comparing the predicted values ​​with the actual values, taking all control strategies into account.

Finally, it is discussed whether the control AI can be transferred to a real process and for which processes this control method is unsuitable.

## Problem with the ReLU activation function

One problem that arose during the evaluation was that one or more output neurons of the ANN with the ReLU activation function often remained zero throughout the runtime. This is probably due to the one in chapter2.2.3This is due to the dead neuron problem mentioned in the ReLU function section.

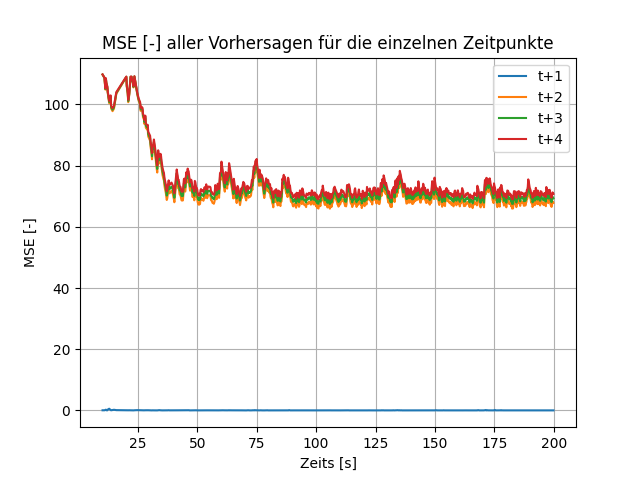
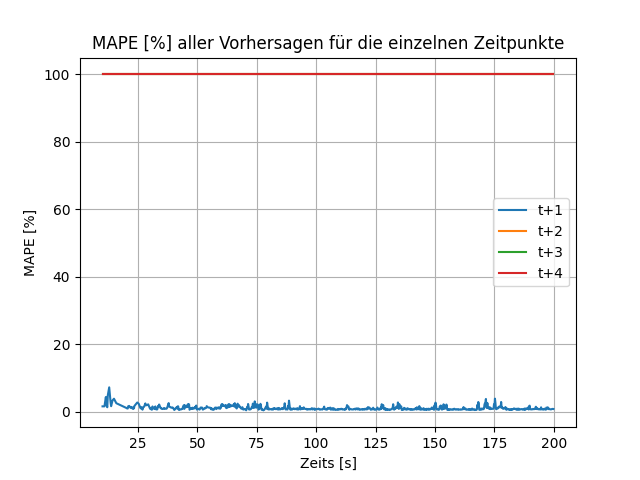


Illustration22: MSE and MAPE for the individual time points at which three output neurons are permanently zero

Illustration22shows the MSE and MAPE of each time point using the ReLU activation function. It can be seen that only the prediction for is accurate. The remaining three time points have an MSE of approximately 70 and a MAPE of 100%. This is because the output neurons that make the prediction for the second, third, and fourth time steps are permanently turned off.

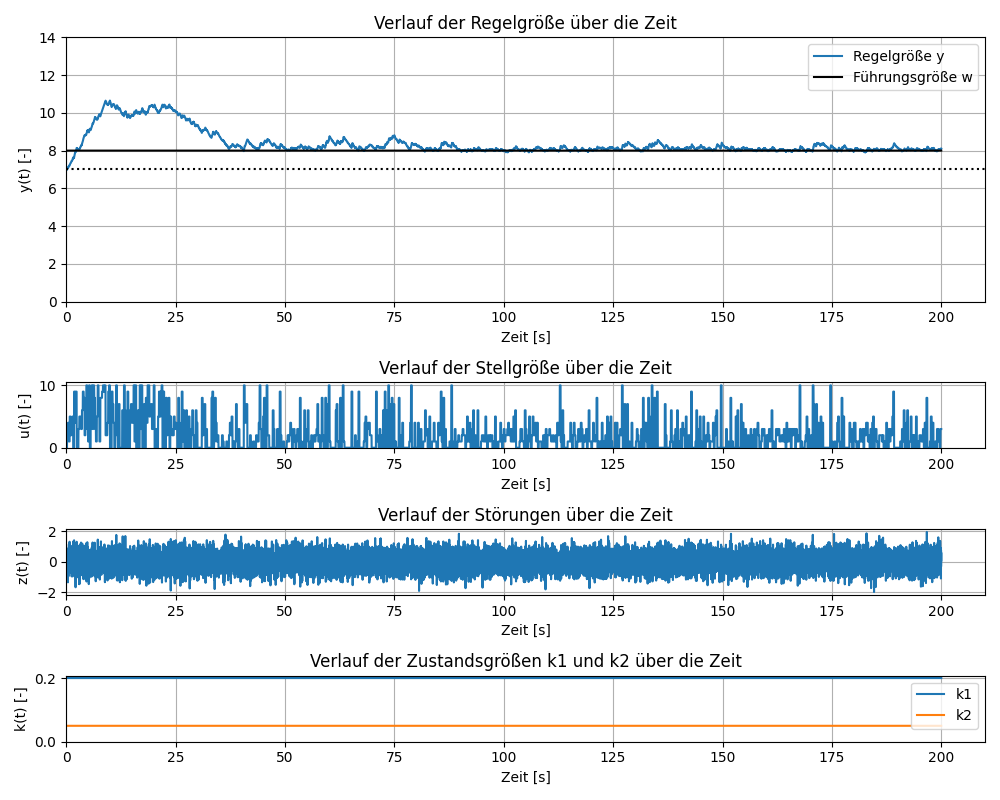


Illustration23: Simulation progression for a control AI in which three output neurons are permanently zero

Despite this erroneous prediction showsIllustration23that the regulation is successful. This is because the cost of the incorrectly predicted time steps remains constant since neither the output nor the weight of the time steps changes. If the cost of a time step remains the same for all predictions, it can be practically ignored since it has no influence on the choice of the cost-minimum strategy. Successful control is theoretically possible as long as at least one output neuron can make correct predictions. However, to avoid this problem, the leaky ReLU activation function with a slope of 0.1 in the negative range is used instead of the ReLU function for each layer of the ANN. This fixes the problem and leads to better predictions.

## Control of a static system

This chapter examines the effectiveness of the control AI in the virtual system with constant state variables and a constant reference variable. For this purpose, the influence of the window size, the process parameters and the disturbances on the control success is examined. The simulation time is set to 200 seconds. The system parameters are set as follows: ,  
, , and . The simulation time is set to 10ms. The weighting for the cost function(32)is also calculated, later values ​​are therefore weighted higher than earlier values. The secondary condition(34)is not initially taken into account, which means that the manipulated variable can exhibit high fluctuations. The simulation starts at . When it comes to interference behavior, only random noise is initially taken into account. The influence of and is discussed in chapter4.2.3examined. The value of is 0.7, which means that the probability of a random action after 10 training iterations is only 3%. This should give the rule AI enough time to learn the system behavior.

### Influence of the window size on the control success

The following section examines how varying , , , and affects the control and prediction accuracy of the rule AI. The amount of training data per training iteration depends on , , and. The larger the time window, the less training data is available. If the time window is chosen too small, it can happen in complex systems that temporal connections are not recognized. A time window must be chosen that is large enough to reflect the temporal relationships without the amount of training data becoming too small.

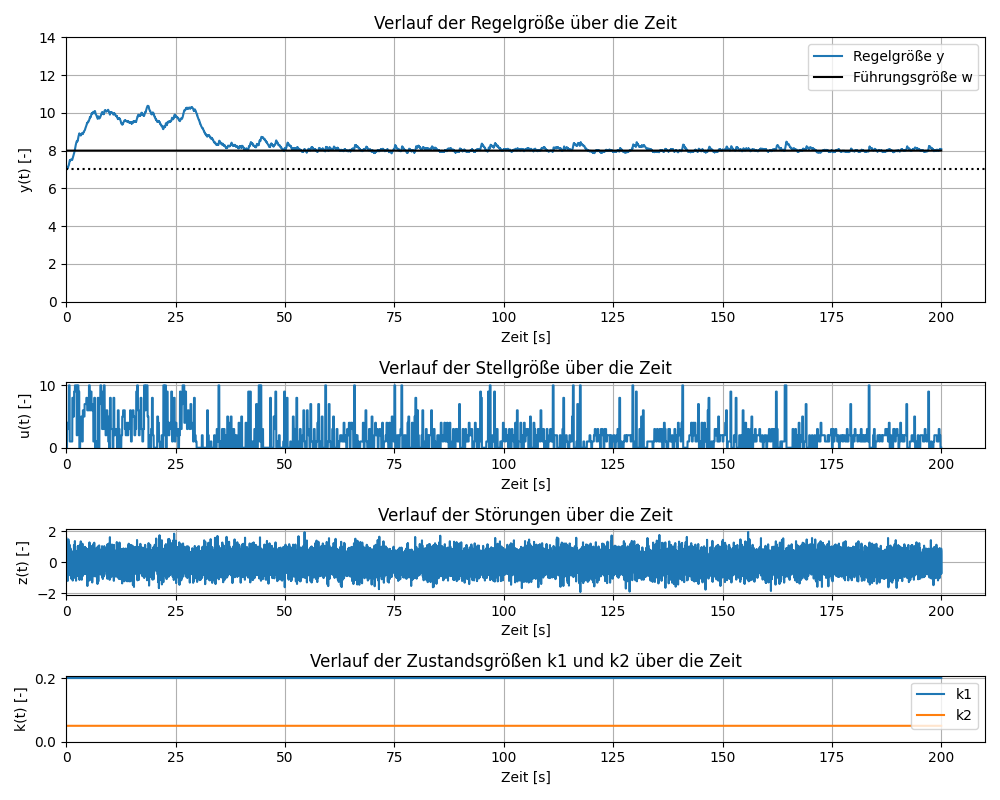


Illustration24: Simulation course for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s

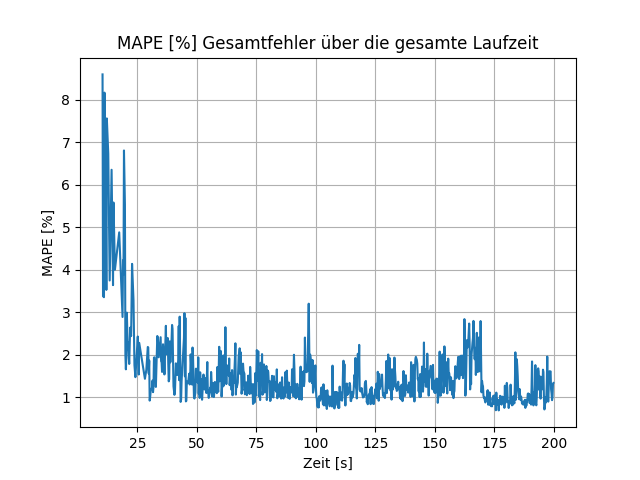


Illustration25: MSE and MAPE for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s

First, the control AI with a symmetrical time window with , , , and is examined. Choosing these parameters results in a window size of 1.6 seconds(38)and 42 new data per training iteration(39). The course of a simulation with these parameters is inIllustration24It can be seen that the control AI reaches the reference value after about 50 seconds and maintains it over the entire period. InIllustration25It can be seen that after just a few predictions the total error (MSE and MAPE) drops below 0.1 and 3%, respectively. This is because the virtual system with constant state variables is not complex and the control AI therefore quickly learns the temporal relationships. In more complex systems, the rule AI would probably take longer to learn the temporal relationships. The reason why the rule AI only reaches the reference value after 50 seconds, even though the prediction error is small after just a few seconds, is due to the Epsilon Greedy algorithm, which often selects random actions in the range from 0 seconds to 50 seconds. The probability of a random action is , since the AI ​​has been trained twice up to this point. After 30 seconds, the AI ​​is trained a third time, dropping to 34.3%. From this point on it can be seen that the control AI is slowly approaching the reference value. Furthermore, it can be seen from the course of the manipulated variable that small deflections occur after the reference variable is reached. In most cases, this is due to the fact that in this simulation there is a 10% probability that one of the 10 best control strategies will be selected and that the secondary condition(34), which limits the maximum rate of increase of the manipulated variable, is not taken into account.

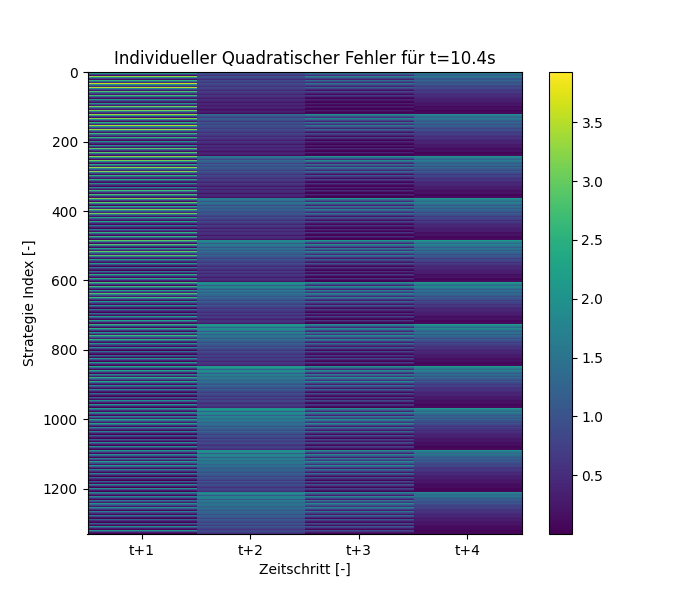
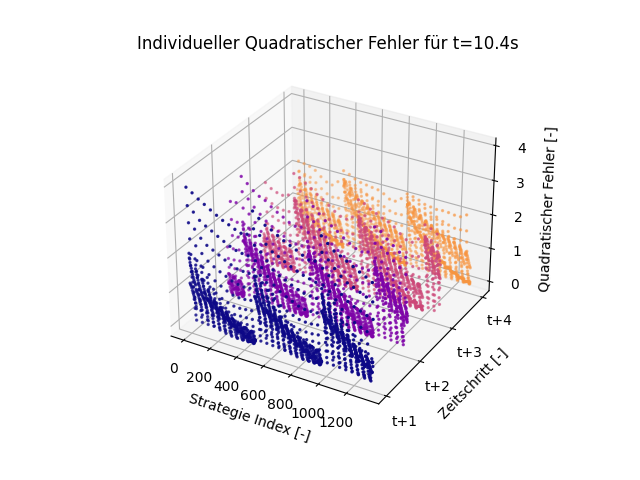


Illustration26: Squared error across all control strategies and time steps of the first prediction for t = 11.2s

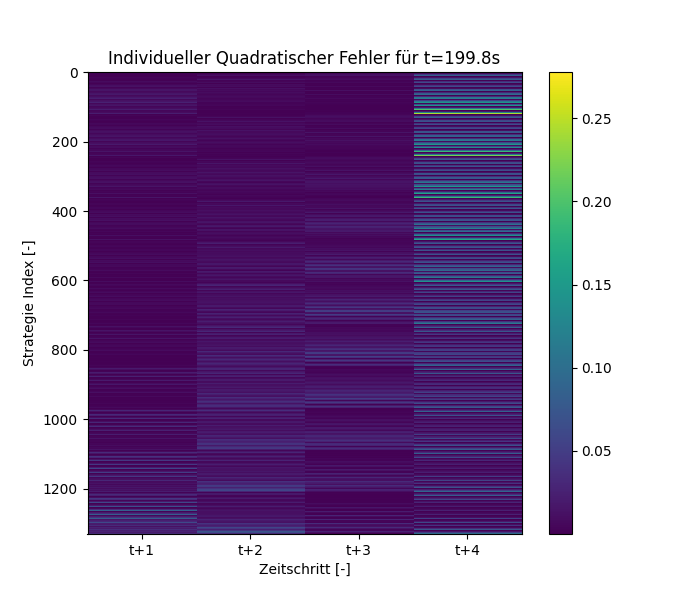
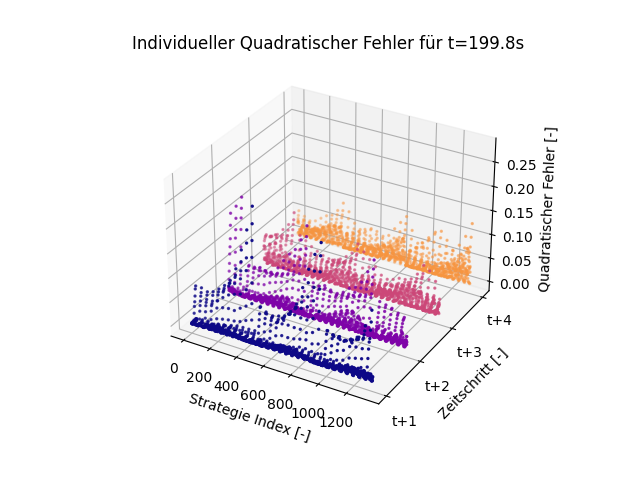
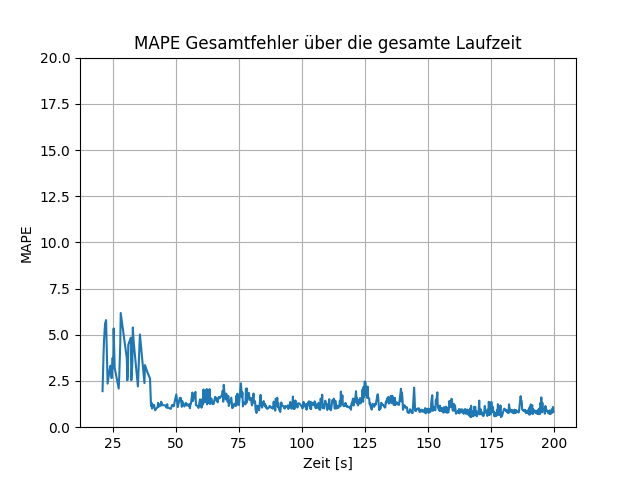
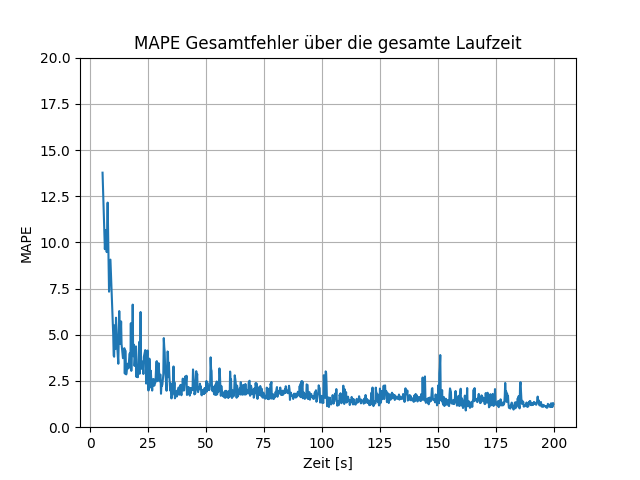


Illustration27: Squared error across all control strategies and time steps of the last prediction for t = 199.8s

InIllustration26andIllustration27the squared errors of the predictions for each time step and for each control strategy are shown in a 3D scatterplot (left) and a heatmap (right). Each point in the 3D scatterplot represents the prediction of a control strategy for a specific time step. The heatmap can be interpreted as a bird's eye view of the 3D scatterplot, with the size of the error represented by a color gradient, similar to a map depicting mountains.

The first of the two figures refers to the first prediction of the rule AI. It can be seen that the prediction at this point is inaccurate, with a maximum root mean square error of about 3.5. The second of the two figures shows the error of the last prediction. It can be seen that the prediction error is significantly lower than at the beginning of the simulation. In the end, the maximum squared error is only about 0.25. Furthermore, it can be seen that the prediction in the last time step is the least accurate, which is due to the difference between the control horizon and the forecast horizon. This effect is explained in more detail in the section “Influence of h and m”.

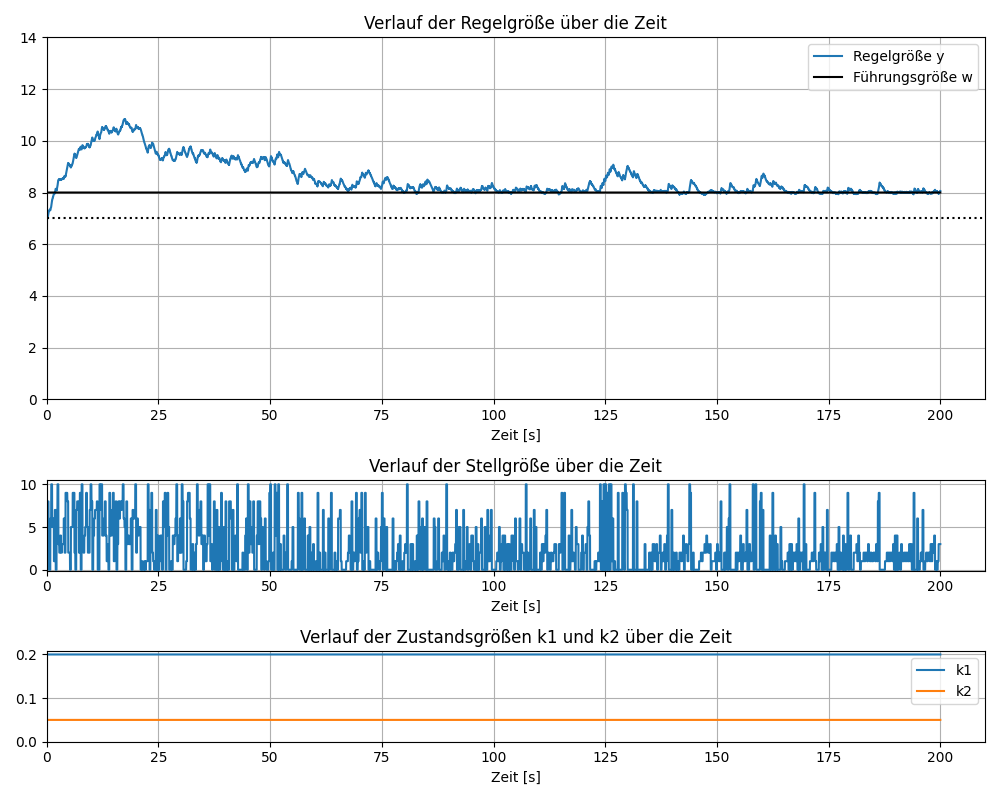
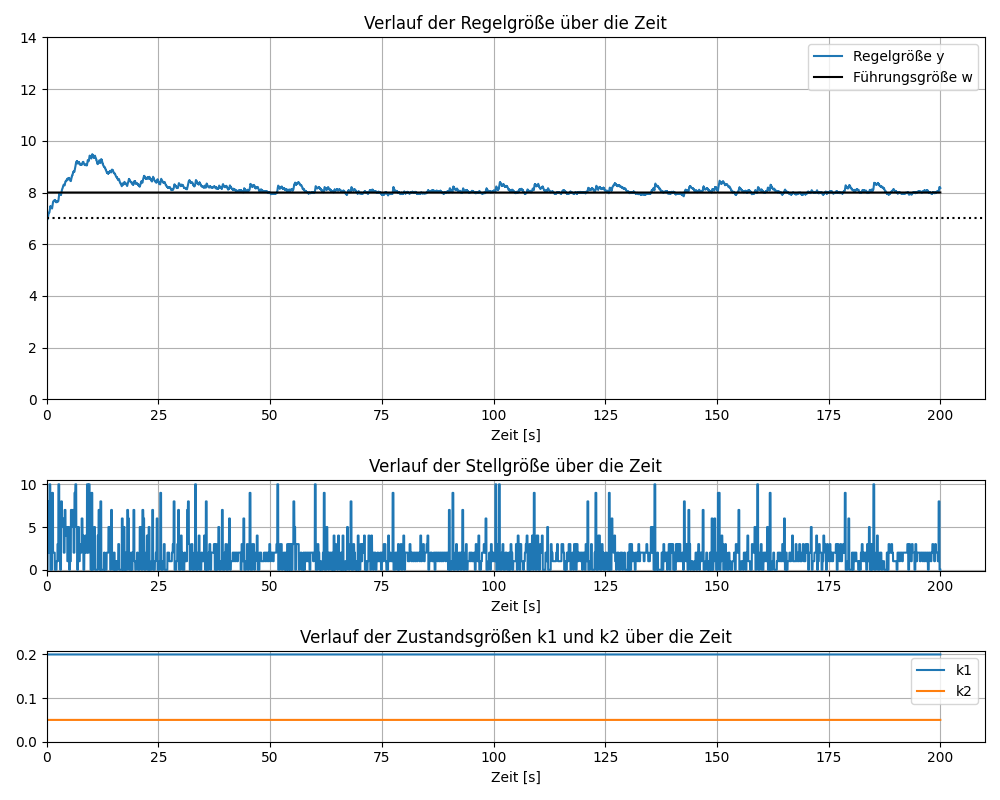
**Influence of training interval ():**The variation of is influences how often the rule AI is trained and therefore also how quickly it decreases, since after each training iteration the value of is multiplied by . The more frequently the rule AI is trained, the faster the probability of random actions being carried out decreases.



[%]

[%]

Illustration28: MAPE for ttrain = 5s (left) and ttrain = 20s (right)



u(t) [-]

u(t) [-]

Illustration29: Course of the manipulated variable for ttrain = 5s (top) and ttrain = 20s (bottom)

The comparison of the prediction error at and is inIllustration28shown, in which the MAPE error of the prediction is shown over the entire runtime. The timeline of the graphs starts after 5s and 20s, as the rule AI does not make any predictions before the first training run, as it is 100% until the first training run. An interesting result can be seen in the right graph, where the prediction error drops sharply after 40 seconds, i.e. after the second training iteration. After a runtime of 20 seconds, the progression of the two errors is similar. It follows that this system does not have a major influence on the control success. The influence of on the manipulated variable is inIllustration29shown. It can be seen that the fluctuations in the manipulated variable are smaller than with , as it decreases quickly and therefore fewer random actions are carried out.

**Influence of control interval ():**The value of determines the intervals at which the rule AI should intervene. A larger value for leads to a larger distance between the points in time and thus a larger time window. This means that the manipulated variable can be changed less frequently, which can lead to greater fluctuations in the controlled variable if the optimal control strategy is incorrectly calculated. In addition, less training data is stored. Larger prediction errors can also occur because larger , values ​​must be predicted further into the future. For this purpose, longer temporal relationships between the manipulated variable and the controlled variable are recorded.

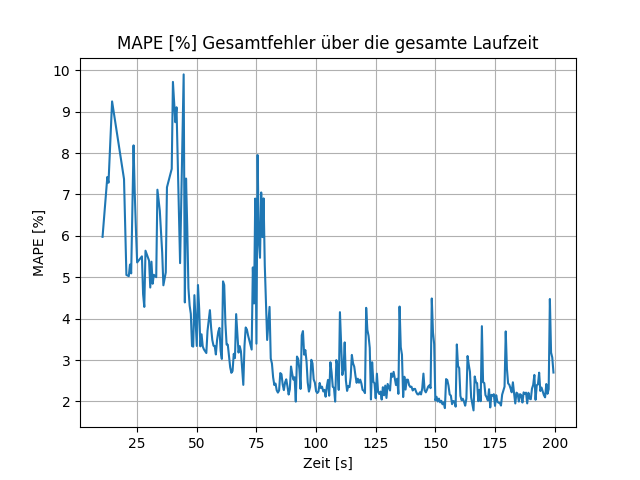
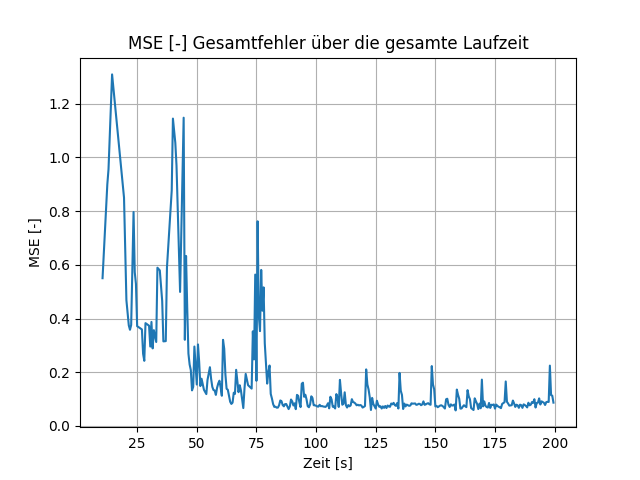


Illustration30: MSE and MAPE for tact = 500ms

The course of the prediction error for is inIllustration30shown. This is compared to the error for offIllustration25higher over the entire term and shows greater fluctuations. This is probably because the 4s time window is more than twice as large as the time window for , requiring values ​​to be predicted further into the future, leading to larger errors. Nevertheless, the regulation is successful in both cases.

**Influence of and:**The larger the control and forecast horizon is chosen, the more predictive the control AI can act, but the computational effort required to evaluate the possible control strategies also increases. The choice of forecast horizon also depends on the complexity of the system. For less complex systems, it makes sense to choose a smaller horizon in order to reduce the computational effort. For more complex systems with strongly delayed transmission behavior, a larger horizon is an advantage.

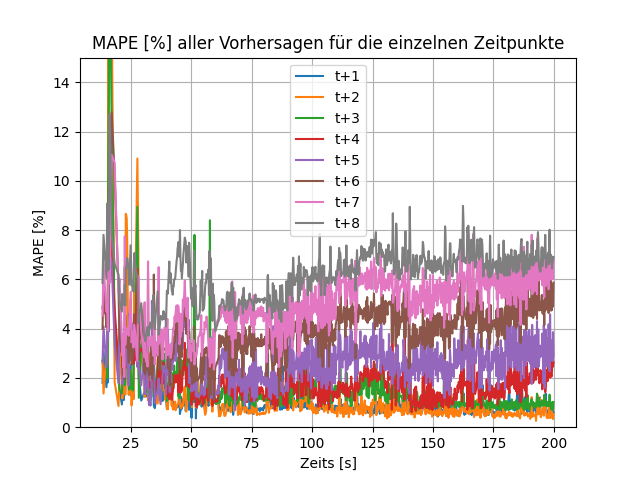
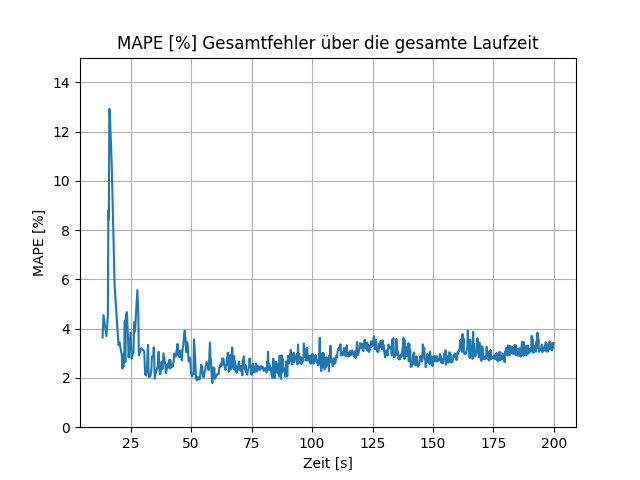


Illustration31: MAPE total error and MAPE for each time point for h = 8 and m = 2

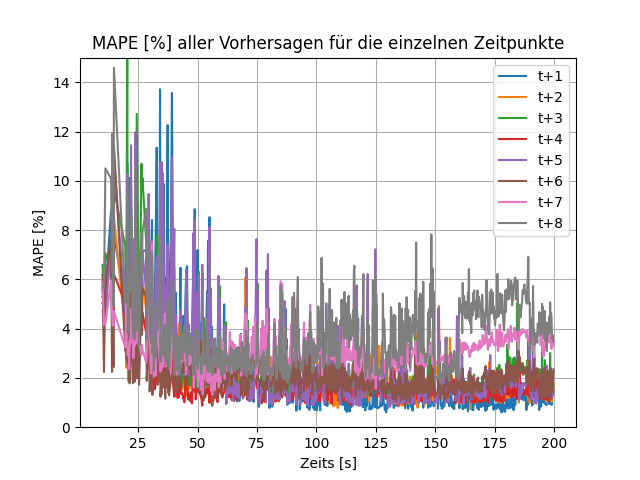
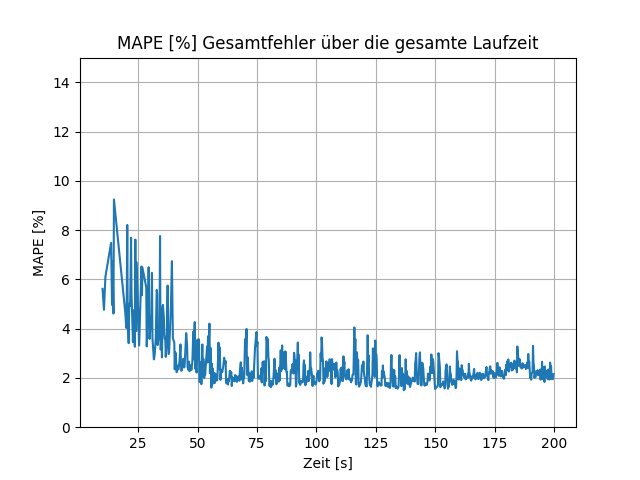


Illustration32: MAPE total error and MAPE for each time point for h = 8 and m = 4

Illustration31shows that predictions for values ​​further in the future have higher errors than values ​​closer in the future, resulting in a higher overall error. This is due to remaining, which requires the rule AI to predict more values ​​with the same inputs. There is a lack of data here to establish the temporal relationship between the values ​​of the points in time  
and to learn. The larger the difference between and is, the less accurate the prediction becomes for points in time further in the future, such as the comparison betweenIllustration31andIllustration32shows. On the other hand, the larger is, the larger the number of possible control strategies and the greater the computational effort to evaluate all strategies. Basically, the choice of size is a compromise between control performance and computing effort. It is necessary to find a balance that meets the needs of each application. The forecast horizon should be chosen so that the temporal relationships between the manipulated and controlled variables are recorded as optimally as possible, without the difference between the forecast horizon and the control horizon becoming too large.

**Influence of :**The larger it is, the more past values ​​are used for the prediction and the better temporal relationships can be mapped. A larger one is expected to result in a smaller prediction error as more time points are considered for the prediction.

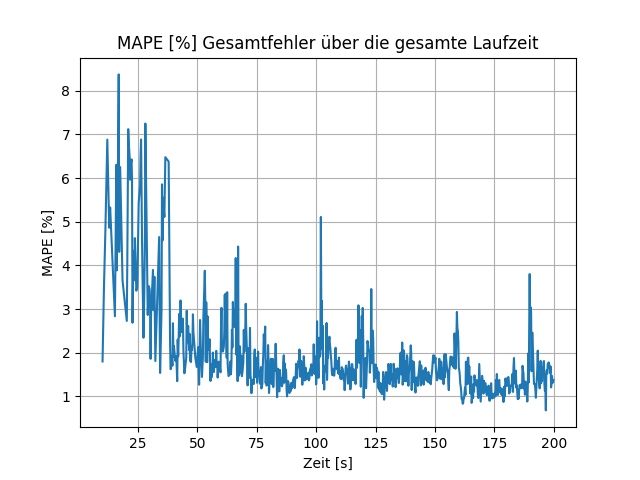
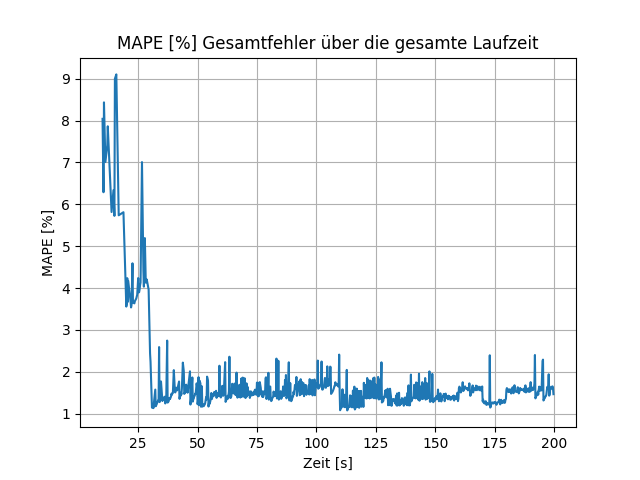


Illustration33: MAPE total error for n = 1 (left) and n = 8 (right)

Illustration33shows the comparison of the MAPE total error for and . In this case, the prediction for is more accurate. This is probably due to the fact that in this case a static system is considered in which the internal state variables remain constant over the entire runtime and thus no long-term, dynamic relationships between the manipulated variable and the controlled variable exist. In addition, more training data is available with a smaller time window. The information about the slope is not recorded for a time window with , which is why in most cases there should be at least two. In both cases the control is stable.

Ultimately, the choice of parameters for the window size in this virtual system is not crucial for the control success, since the reference variable is achieved and maintained in any case. The parameters only affect the stability of the control or the size of the fluctuations in the controlled variable and the prediction error of the control AI.

### Influence of the process parameters on the control success

The choice of the set of possible values ​​for the manipulated variable depends on , and. In any case, there should be a possible manipulated variable that can move the system in the direction of the reference variable. For example, the largest possible value of the manipulated variable should be able to change the controlled variable when it reaches the minimum value. For and  
is the slope with(41). In this case, in order for the system to leave the minimum value. For there must be at least one that is greater than 9.66 to bring the slope into the positive range. If this condition is not met, the controlled variable can no longer be influenced when reached. The general condition for successful regulation in this system is

|  |  |
| --- | --- |
|  | (43) |

A larger value or larger values ​​for the manipulated variable can lead to larger fluctuations in the controlled variable, especially if the reference variable is close to , since the slope is then small and a large value for can lead to a large deflection of the controlled variable.

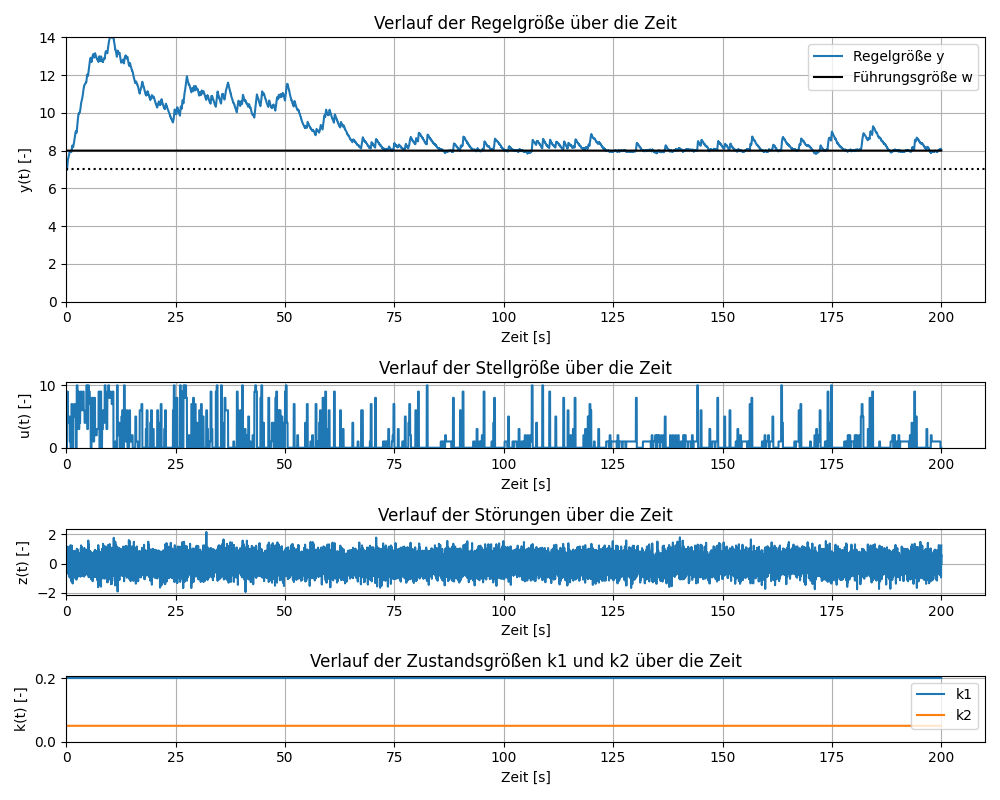


Illustration34: Simulation course for ks = 0.3

Illustration34shows the course of a simulation for , with the same window size (, , ). In comparison to a simulation with (seeIllustration24), the control is more unstable and has higher fluctuations. This is due to the fact that a larger deflection of the manipulated variable has a greater influence on the controlled variable.

### Influence of disruptions on the control success

This section examines the response of the rule AI to various disturbances. First, the influence of high noise is examined. The stability of the control AI is then checked by stimulating the system with discrete interference pulses. Finally, the reaction of the rule AI to a continuous, dynamic disturbance is examined.

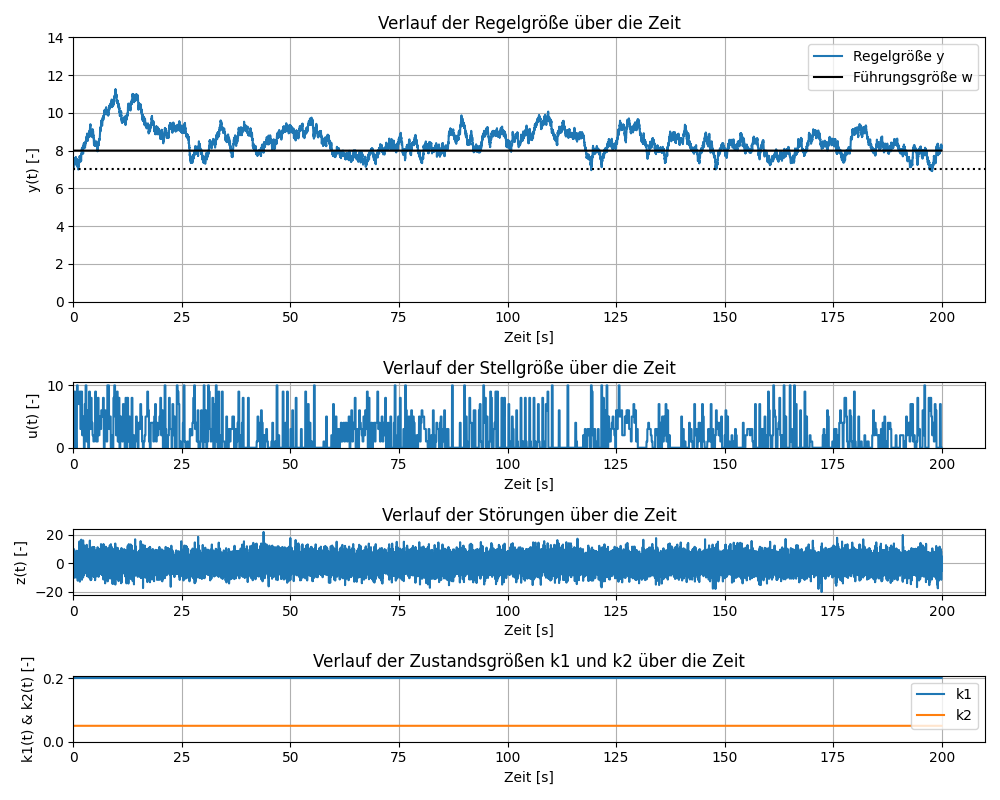


Illustration35: Simulation history for high noise

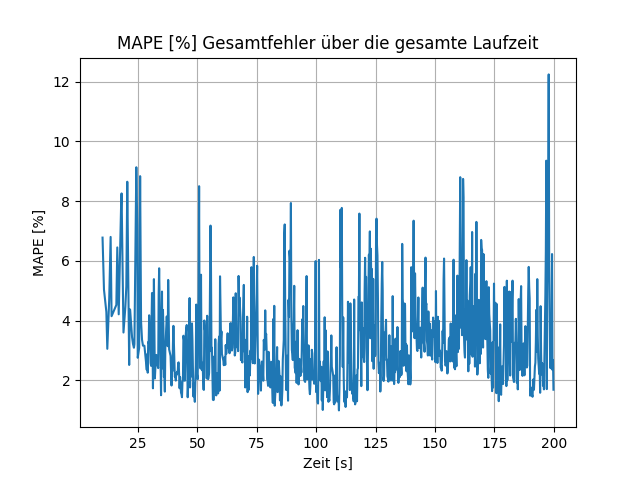
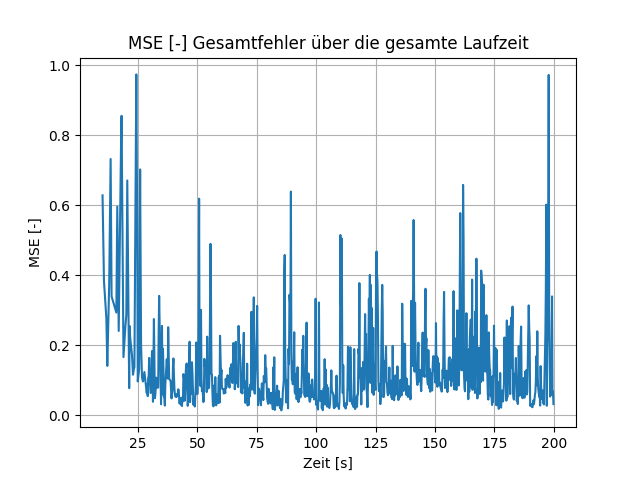


Illustration36: MSE and MAPE for high noise

Illustration35shows the course of the simulation for high noise  
. Basically, the noise represents the effect of small disturbance pulses at each time step of the simulation. It can be seen that there are high fluctuations and the control AI is not able to sufficiently stabilize the system. Nevertheless, the controlled variable roughly approximates the reference variable.Illustration36shows that the prediction error is volatile, which can be explained by the high random noise. Since these are random disturbances, the control AI is not able to accurately predict the control variable.

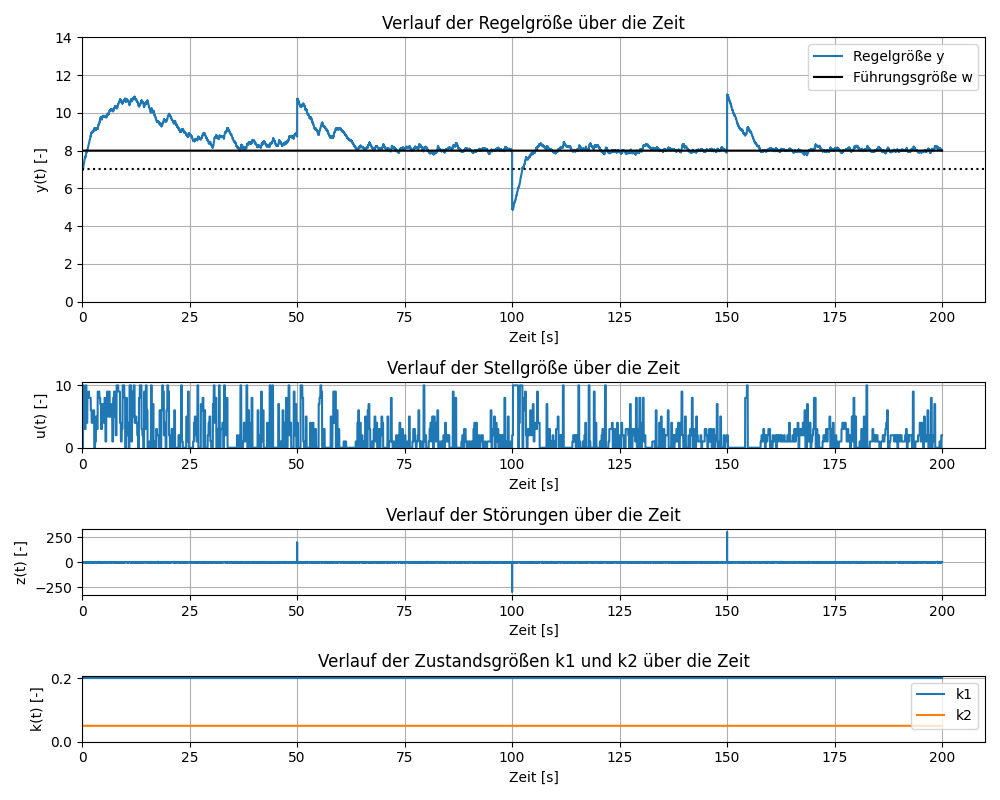


Illustration37: Simulation process when excited by interference pulses

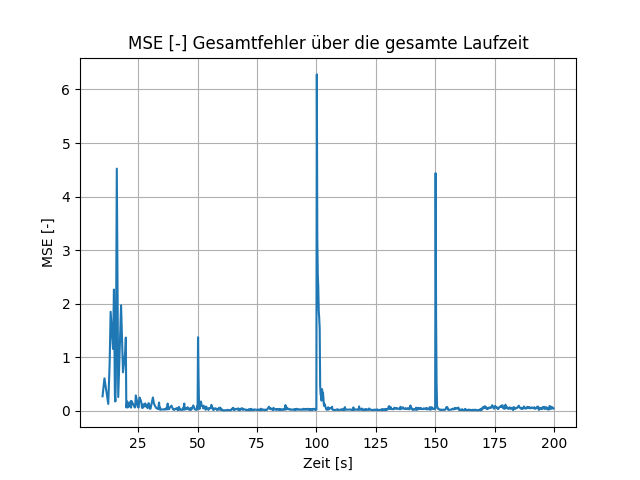


Illustration38: MSE and MAPE when excited by interference pulses

Illustration37shows the course of a simulation in which an interference pulse occurs at times , and . In this case is . It can be seen that the controlled variable shows a clear deflection at these times. However, after a few seconds the control stabilizes again.Illustration38shows that at the times when a glitch occurs, the prediction error increases sharply because the control AI cannot predict it. However, after a few seconds the error returns to its original size. A disturbance pulse therefore has no future influence on the prediction error, which enables the control to stabilize after a disturbance pulse.

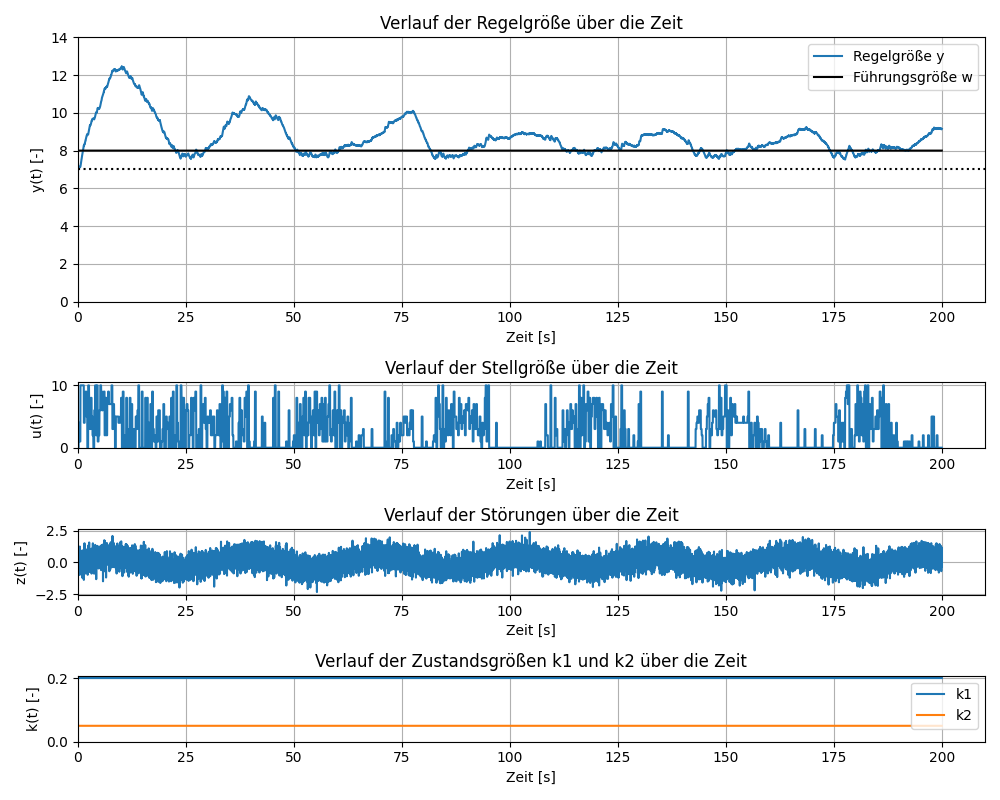


Illustration39: Simulation course with a dynamic disturbance behavior with A = 0.5

InIllustration39The course of a simulation with dynamic disturbance behavior is shown. The dynamic disturbance in this case is caused by the function

|  |  |
| --- | --- |
|  | (44) |

determined with the amplitude. The value for here is again 0.5. The simulation process shows that the dynamic disturbance shifts the expected value of the random noise because and add up to an overall disturbance. It can be seen that the control is unstable and exhibits strong fluctuations. This is due to the condition(43)is not fulfilled. If the reference value is to hold, then the following applies. The minimum expected value of the disturbance here is -0.5. This results in the condition , and . There, this condition is not met. The influence of the dynamic disturbance therefore exceeds the ability of the control AI to compensate for it. If the maximum amplitude of the dynamic disturbance is less than 0.25, the condition is met.

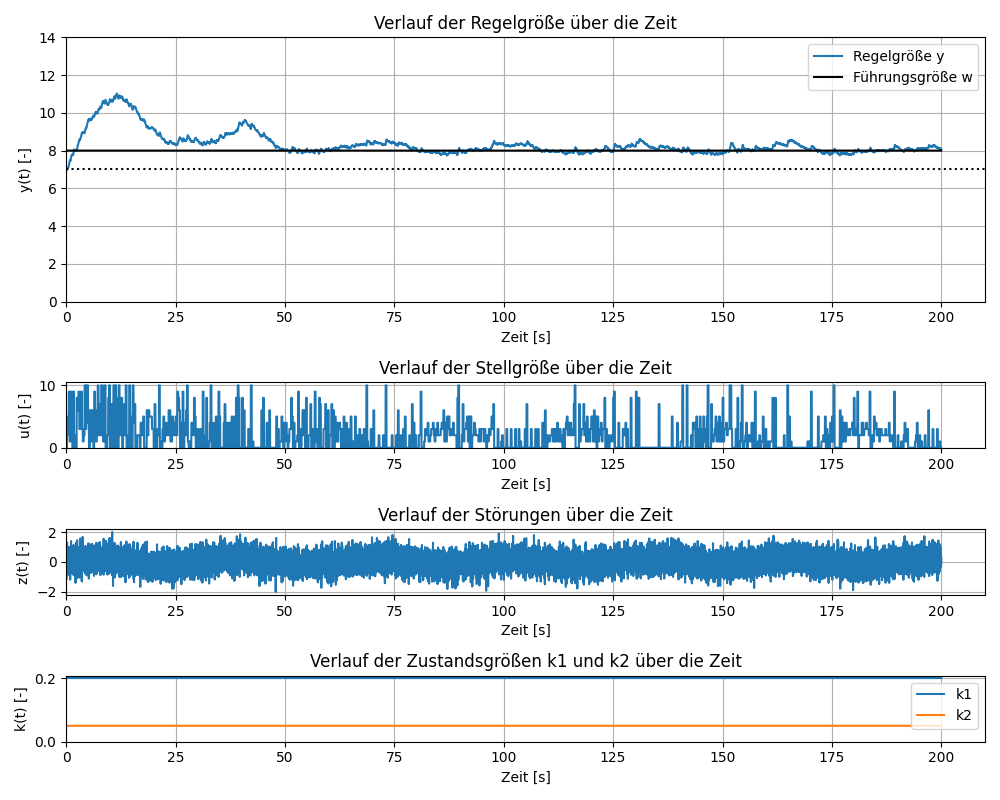


Illustration40: Simulation course with a dynamic disturbance behavior with A = 0.2

The simulation course for the dynamic disturbance(44)with is inIllustration40shown. It can be seen that the regulation if the condition is met(43)is stable.

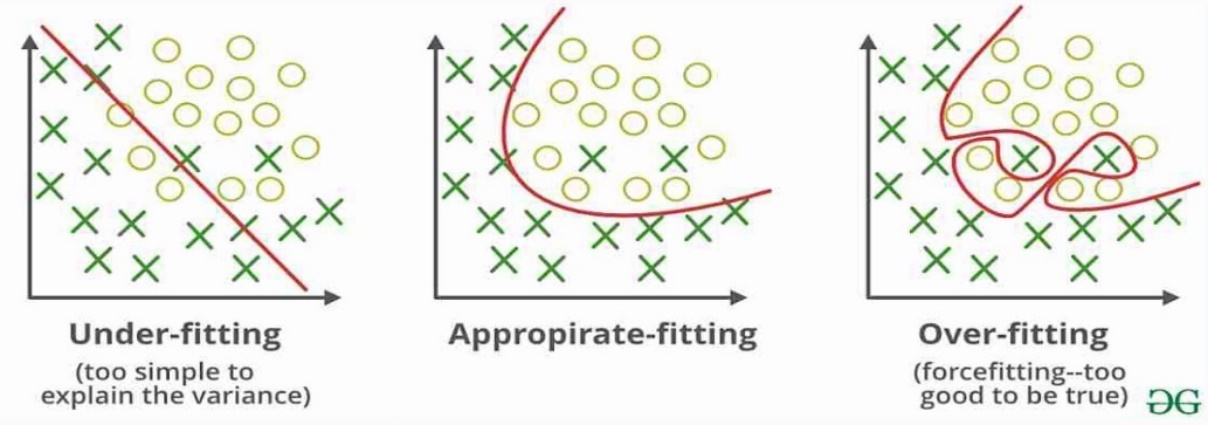


Illustration41: Representation of underfitting and overfitting[42]

A high level of random perturbation can lead to “overfitting” when training an ANN. This happens when the model tries to recognize connections that do not exist in reality. This causes the model to overfit the training data, thereby losing its ability to generalize new and unknown data.Illustration41shows the difference between underfitting and overfitting. One way to avoid overfitting is to reduce the complexity of the model by reducing the number of hidden layers. The “dropout” method can also be used, in which neurons are randomly deleted during training. It may also be useful to filter and normalize high-noise data before training.[8th]

Care should also be taken to ensure that the “Epoch” hyperparameter is not chosen too large. The number of epochs indicates the number of times a training data set is iterated, with the ANN's weights updated at each iteration. A high number of epochs can lead to overfitting, while a too low number of epochs can lead to underfitting. This is especially true for this use case, where the ANN is continuously trained with new and small data sets. The number of epochs was set to 50 in all simulation runs.

### Effect of the secondary condition Δumax on the control success

In the following section the control strategies are discussed taking the secondary condition into account(34)selected. Only those control strategies are considered in which the maximum change in the manipulated variable between two points in time is smaller than a specified one. This reduces the number of possible control strategies and thus also the computational effort for the prediction. In addition, there can no longer be large jumps in the manipulated variable, which makes the control more stable.

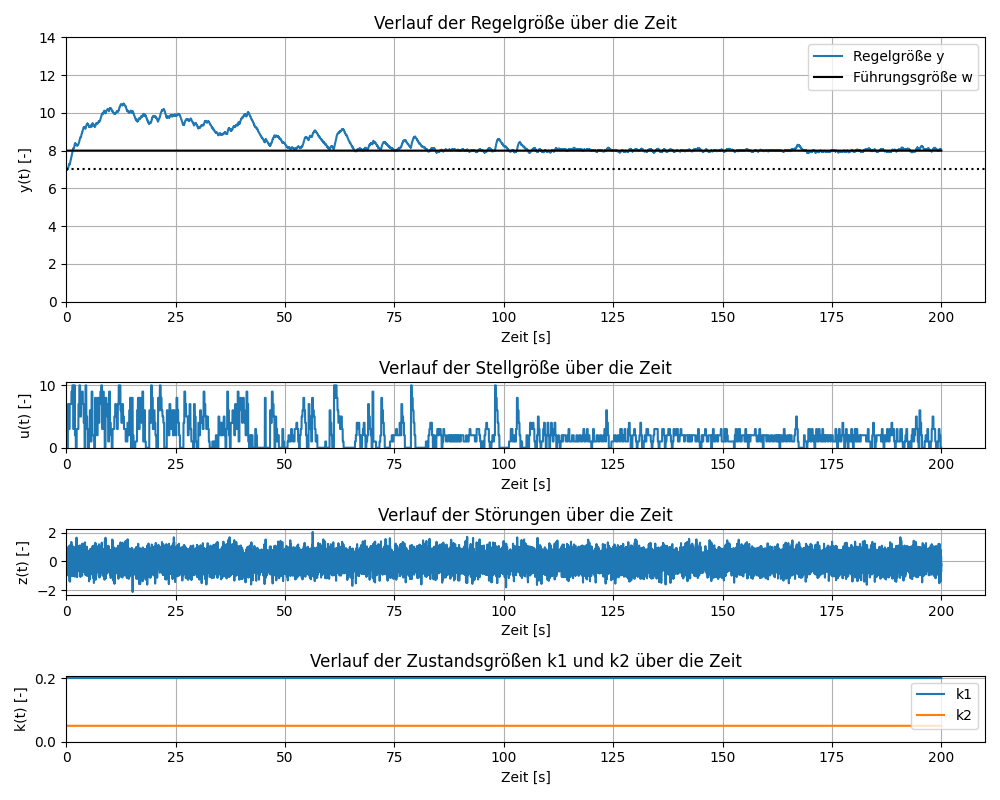


Illustration42: Simulation curve for Δumax = 2

Illustration42shows the course of the simulation for , , and . It can be seen that from then on there are hardly any fluctuations in the manipulated variable and the controlled variable maintains the reference variable almost exactly. For a control horizon of  
and the given quantity, possible control strategies arise without using the secondary condition. The number of possible control strategies drops to a maximum of 123. By introducing the secondary condition, the control becomes more stable while the computational effort is reduced.

### Classification of the influencing factors on the success of the regulation

This section summarizes the influence of the various parameters on the control success. For this purpose, the parameters are ranked according to the size of their influence on the control success.

Table9: Ranking of the factors influencing the success of the regulation

|  |  |
| --- | --- |
| **Ranking** | **Influencing variables** |
| 1 | Process parameters (, , , ) |
| 2 | Additional condition |
| 3 | Size of the noise |
| 4 | Difference between and |
| 5 | control interval |
| 6 | Dynamic & Impulsive Disturbances |
| 7 | Training interval |
| 8th | size of |

Table9shows the ranking of the various factors influencing the success of the regulation. The process parameters have the greatest influence on the control success, which depends on the condition(43)is attributable. In order for a control to be successful at all, there must be at least one discrete manipulated variable that moves the controlled variable in the direction of the reference variable.

The secondary condition has the second greatest influence(34), which limits the maximum rate of increase of the manipulated variable. This was found to have a significant impact on the stability of the control and the required computational effort.

The third biggest influence is the size of the noise. If the noise is too great, the control AI cannot sufficiently learn the temporal relationships between the manipulated variable and the controlled variable, which makes the control unstable.

The fourth largest influence is the difference between the forecast horizon and the control horizon. The larger the difference between the forecast horizon and the control horizon, the larger the prediction error becomes, especially for points in time further in the future.

The fifth largest influence is the size of the rule AI's rule interval. The larger the control interval, the less frequently the manipulated variable can be changed, which can lead to greater fluctuations. However, a larger control interval also enables a larger time window, which can be an advantage if transmission behavior is significantly delayed.

The sixth-largest influence is impulsive and dynamic interference behavior. It has been observed that the rule AI can successfully adapt to impulsive disturbances. The adaptability depends on the frequency and size of the interference pulses that occur. Adaptation to dynamic disturbances is also possible if the condition(43)is fulfilled in any case.

The seventh largest influence is the size of the training interval, which only has an influence on how quickly the control reaches the reference value.

The number of values ​​from the past that are used for the prediction has the least influence on the control success. The scheme itself was found to be successful.

The results refer to the proxy system used in this work and may be different for other systems.

## Control of a dynamic system

This chapter examines the adaptability of rule AI to dynamic changes in system properties. On the one hand, the reaction of the control AI to a change in the reference variable is examined. On the other hand, the adaptability of the control AI is tested by changing the internal state variables and the system over time.

### Change in the reference variable

The control AI should be able to adapt to a change in the reference variable as quickly as possible. For this purpose, the following section considers a simulation in which the reference variable is changed suddenly at discrete points in time.

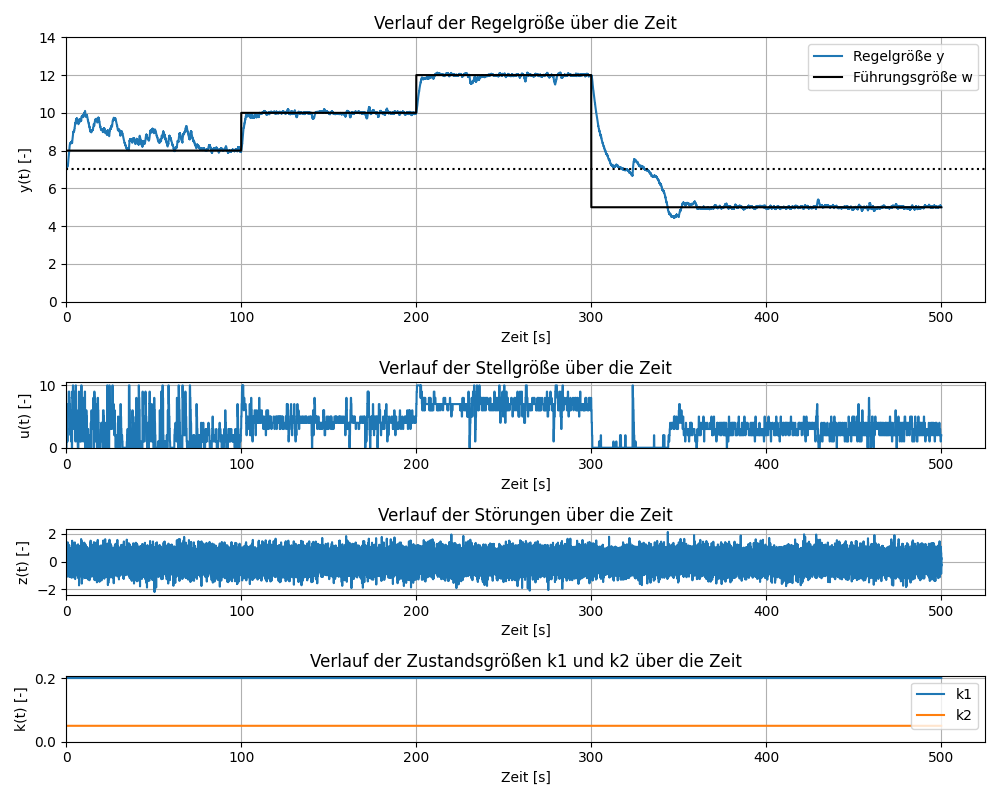


Illustration43: Simulation course when the reference variable changes

InIllustration43The course of a simulation is shown in which the reference variable is changed abruptly to 10, 12 and 5 at times , and . The simulation is based on those in chapter4.2parameters used with additional consideration of and the window size  
. As can be seen from the figure, the rule AI can successfully adapt to any change in the reference variable after a short period of time. This indicates that the control AI learns the temporal relationship between the manipulated variable and the controlled variable. However, it should be noted that the success rate of adapting the rule AI to a change in the command variable depends on the complexity of the system. With a more complex system, an adjustment may not be successful.

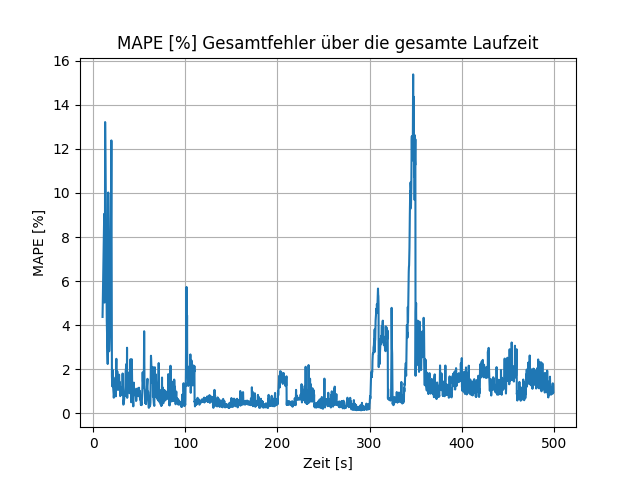
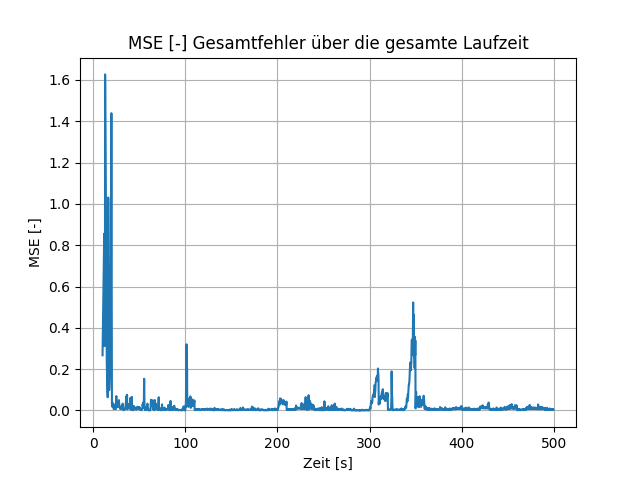


Illustration44: MSE and MAPE when changing the reference variable

Out ofIllustration44It can be seen that the total error increases at the times when the reference variable is changed and then decreases again shortly afterwards. This is an indicator that the rule AI needs some time to learn the new system state. In any case, the prediction error decreases again after a change in the reference variable.

### Dynamic state variables k1 and k2

The rule AI should also be able to adapt to dynamic changes in system properties. In order to test this adaptability, this section examines the control behavior in the event of an unpredictable and a predictable, dynamic change of and .

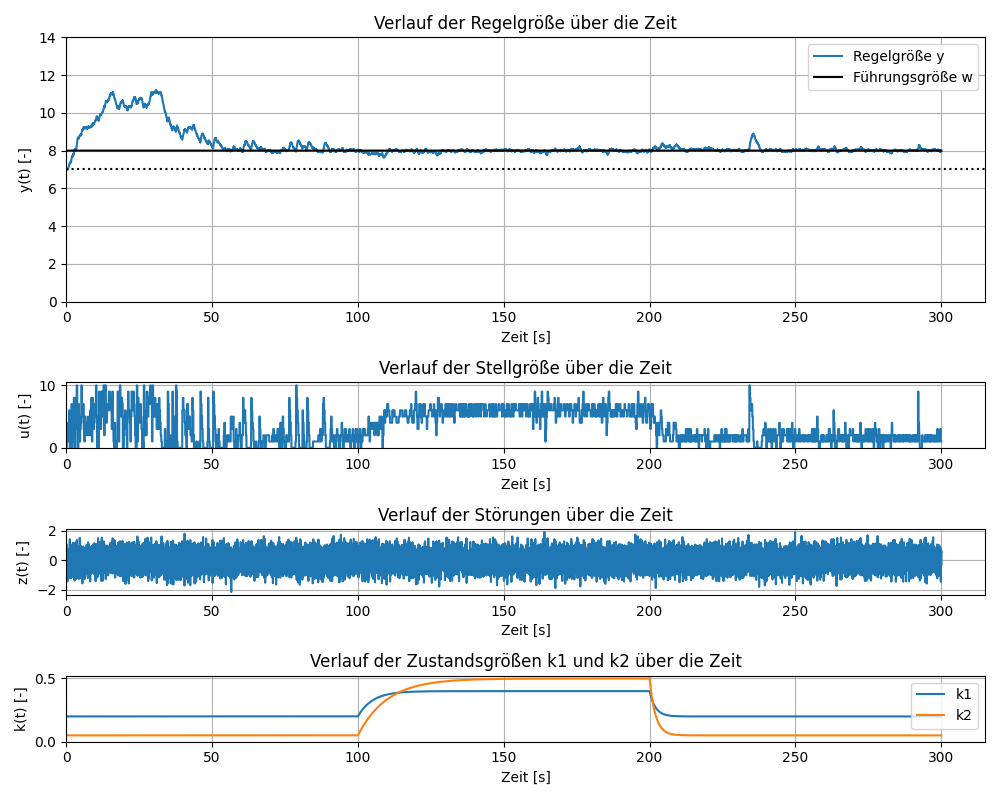


Illustration45: Simulation course with delayed jump change of k1 and k2

In order to investigate an unpredictable change in the system properties, and are changed at discrete points in time. This change is delayed by a PT1 element.Illustration45shows the course of the simulation for a change in the state variables and at the times and . At the moment it is set to 0.4 and 0.5. At this point and are set back to their original values. The delayed step response of the PT1 element in this case is

|  |  |
| --- | --- |
|  | (45) |

For the first jump, the delay time for the change of is set to 5s and for the change of to 10s. So the value of is delayed longer. When resetting the values, both are reset with a delay of  
changed to examine how the rules AI reacts to sudden changes.Illustration45shows that the control AI is able to adapt to an unpredictable change in the internal state variables and continue to maintain the reference variable. In the case of dynamic changes to the system state, care must also be taken to ensure that the condition(43)is satisfied. In this case, the change from to 0.4 and to 0.5 and the condition that is satisfied results in .

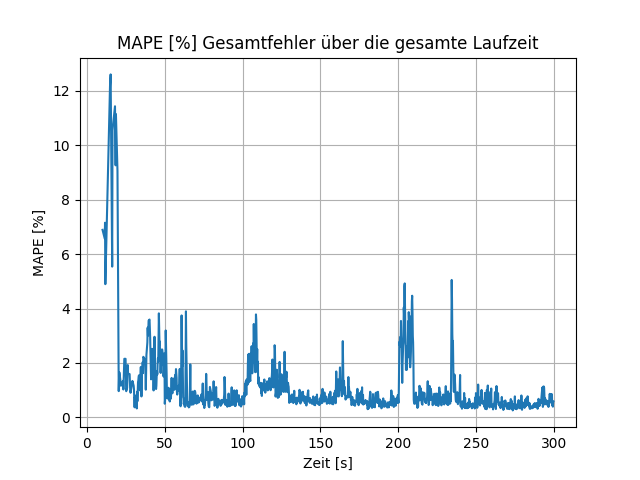
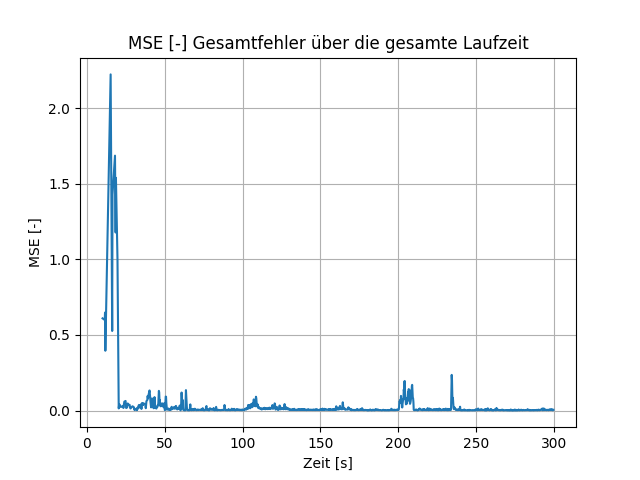


Illustration46: MSE and MAPE with delayed jump change of k1 and k2

Illustration46shows that the prediction error increases briefly after changing from and to and then drops back to the original value after a few seconds. This suggests that the rule AI can adapt to an unpredictable dynamic change in system properties.

In addition to unpredictable changes in the system state, there can also be predictable changes, for example if a state variable runs periodically. If the rule AI can successfully adapt to a predictable change, the prediction error can be expected to remain constant and not exhibit large fluctuations.

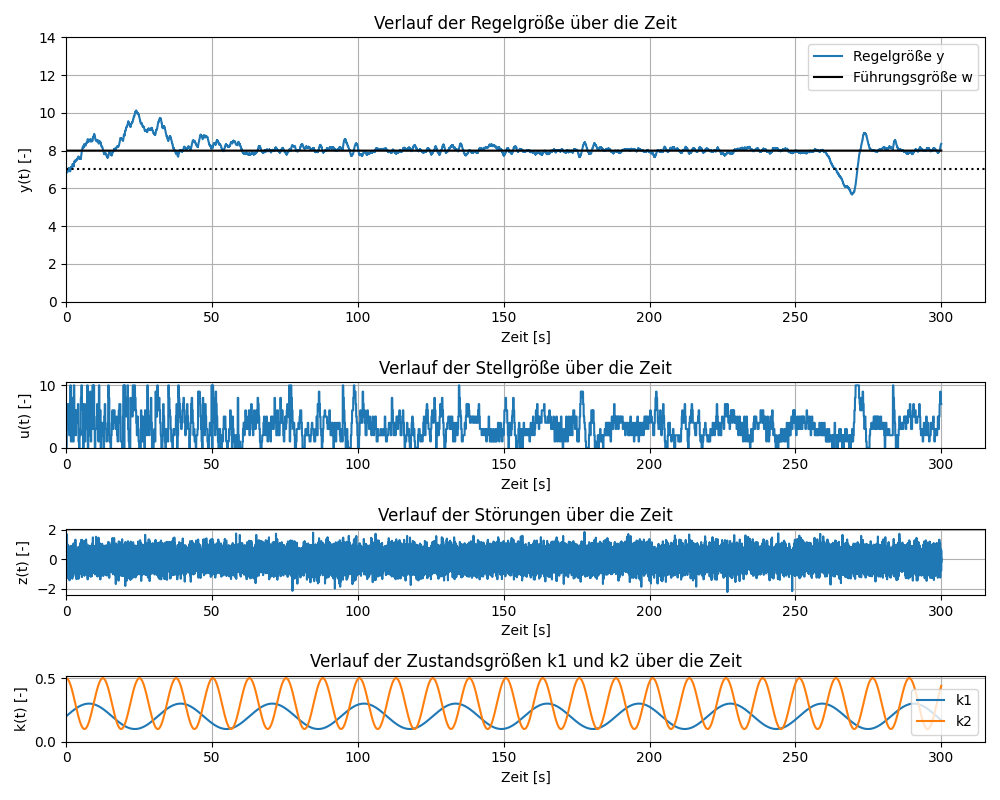


Illustration47: Simulation course with periodic changes of k1 and k2

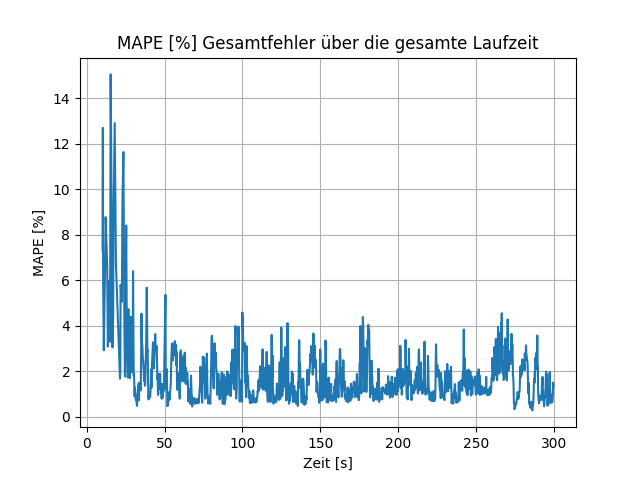
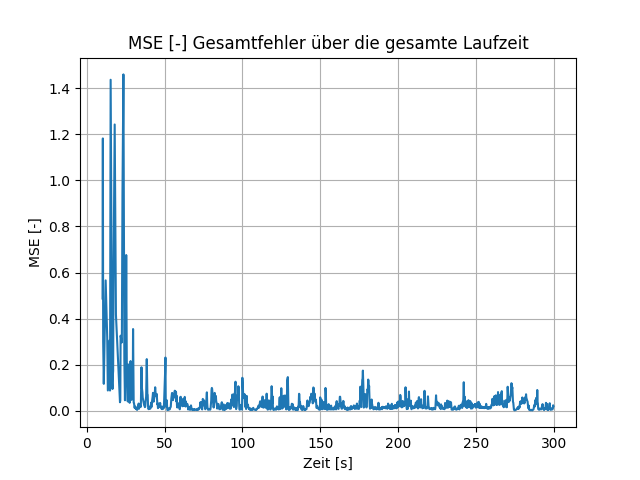


Illustration48: MSE and MAPE with periodic change of k1 and k2

Illustration47shows the course of a simulation, which is represented by a sine function and a cosine function. From the figure it can be seen that the reference variable is reached and is kept stable except for a deflection. Even after a large swing, the control AI is able to stabilize the control variable. Compared to an unpredictable change in and , the prediction error is inIllustration48more volatile overall, suggesting that the rules AI cannot adapt perfectly to complex periodic changes. In this case, varying the window parameters has no significant influence on the prediction error. In order for the control AI to successfully adapt to a dynamic change in the internal state variables, it makes sense to measure these and use them as input parameters for the ANN. This means that the control AI is able to recognize more complex temporal relationships between the individual variables.

### Change in the reference variable and dynamic state variables

In this section, the change in the reference variable is combined with the periodic behavior of and . In addition, all disruptive influences from Chapter4.2.3(noise, impulse, dynamic) are taken into account. The standard deviation of the noise is still 0.5. The system is stimulated every 50 seconds with a random-sized perturbation pulse and the dynamic perturbation behavior is determined by the function(44)with certainty.

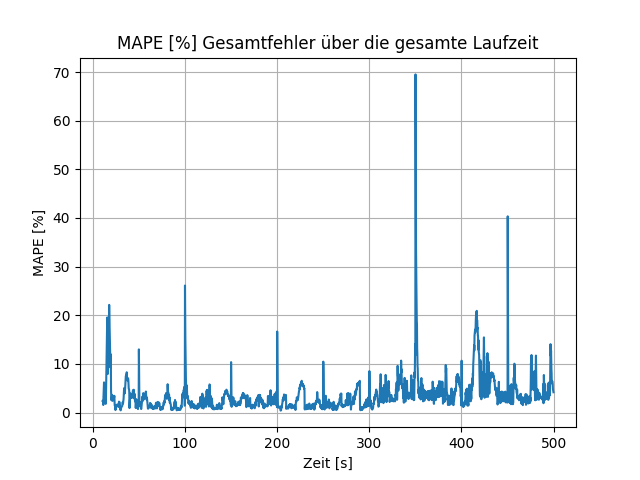
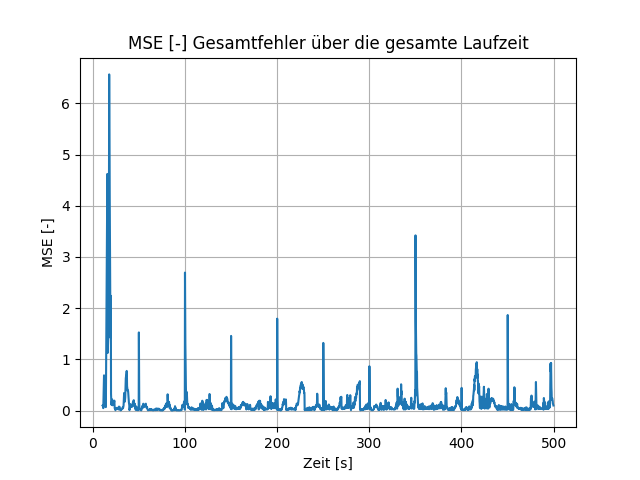


Illustration49: MSE and MAPE for changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 4, m = 3, h = 4

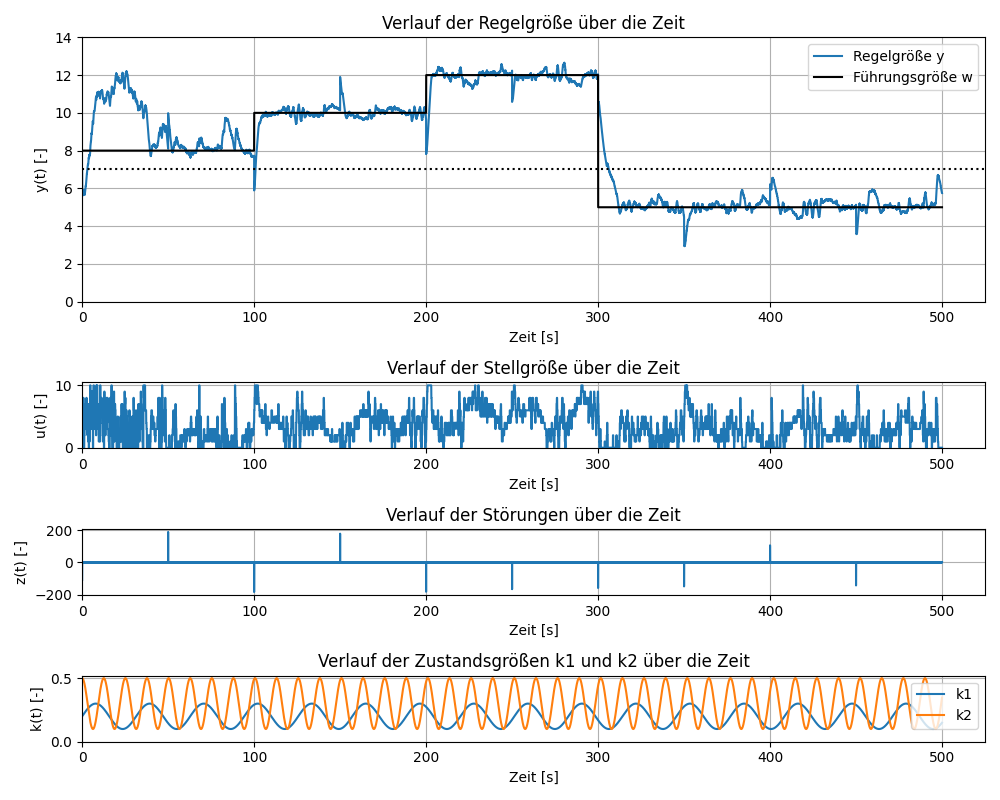


Illustration50: Simulation course with changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 4, m = 3, h = 4

InIllustration50The course of the simulation is shown, whereby the change in the reference variable corresponds to that in section4.3.1and the periodic course of and that in section4.3.2corresponds. The only change is the value of , which is increased from 0.15 to 0.25, making the maximum guide size of 12 for the case  
, and can be held, or with it(43)is satisfied. Otherwise, in this case the maximum value for the manipulated variable would not be sufficient to maintain the reference variable.Illustration49shows that the prediction error is volatile and exhibits large swings at the times when the system is excited with a glitch. For most predictions, the error is less than 0.2% with MSE and less than 5% with MAPE. This indicates that the control AI sufficiently learns the temporal relationships between manipulated, disturbing, state and controlled variables and thus enables successful control with some fluctuations.

In a dynamic system, the choice of window size has a major influence on the stability of the control. If the time window is too small, the dynamic relationships between the individual variables are not adequately recorded.

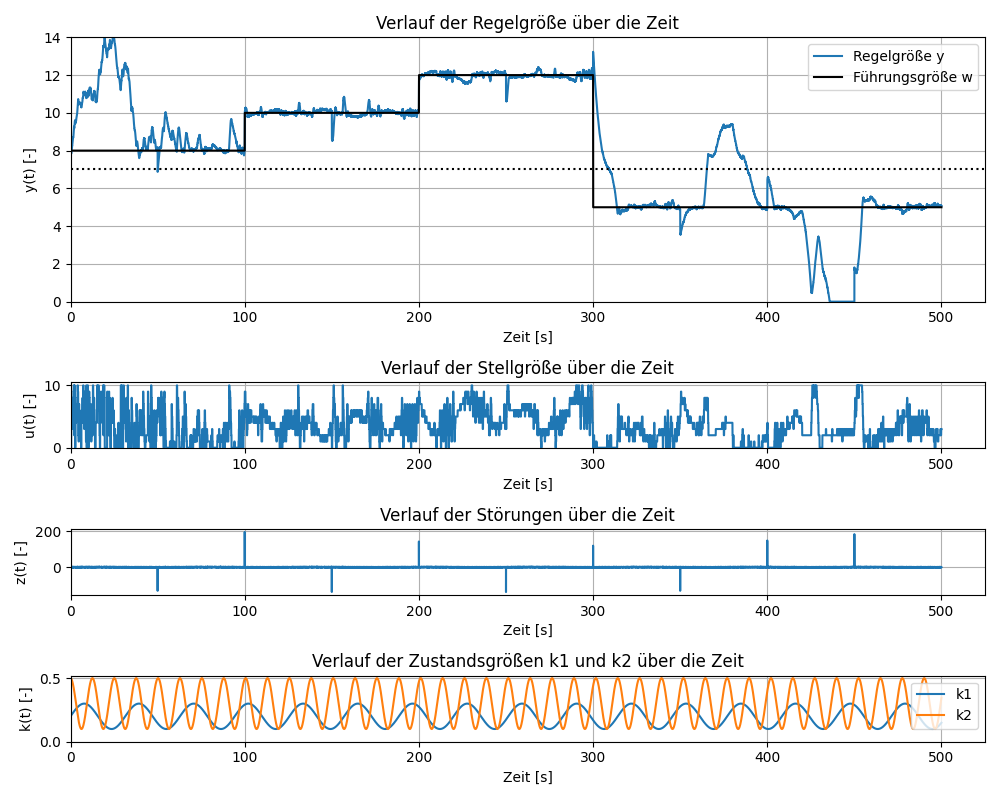


Illustration51: Simulation course with changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 1, m = 1, h = 2

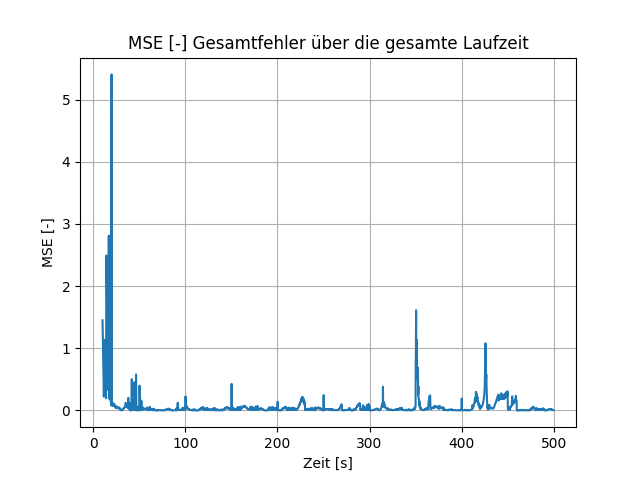


Illustration52: MSE for changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 1, m = 1, h = 2

Based onIllustration51It becomes clear that the control can become unstable with a small time window and sometimes fail completely. It can be seen that the reference variable is reached and maintained, but when it is held, the deflections are too high, which is because too few past values ​​are used for the prediction.Illustration52shows that the MSE prediction error is smaller and more constant for a smaller time window than for a larger time window. The MAPE error cannot be used in this case because the controlled variable reaches 0. The low prediction error is due to the fact that few control strategies have to be predicted, which leads to fewer incorrect predictions. Despite the lower prediction error, the control with a small time window is overall more unstable. Care should therefore be taken to ensure that the time window is not chosen too small, depending on the application of the rule AI.

## Transfer to a real process

Transferring this control system to real processes should be done with caution, as the effectiveness of control AI in these cases has not yet been tested. The virtual proxy system used in this work is not very complex compared to most real processes, which facilitates successful regulation. Rule AI is unsuitable for processes that can easily become unstable and potentially fail completely, as Rule AI learning is based on exploring the system through random actions. An example of this would be the use of this rule AI to control controls in an aircraft, which is not suitable due to the high requirements for safety and reliability.

The implemented control AI can be transferred to a bioreactor for pH value control, taking any adjustments into account. In order to obtain a better process model, it makes sense to measure, in addition to the controlled variable and the manipulated variable, all relevant variables that have an influence on the pH value and to use them as input variables for the prediction. To comply with safety conditions, a separate controller can be installed if necessary, which takes over control for a short time in the event of excessive fluctuations in order to avoid damage to the bioreactor due to pH values ​​that are too high or too low. Otherwise, additional safety mechanisms can be installed, such as switching off the inlet valve if the pH value exceeds a certain limit. Care should also be taken to select a sufficiently large time window to take into account the diffusion of the influent.

By using an ANN as a model for MPC, complex relationships and non-linear relationships between manipulated and controlled variables can be captured without prior knowledge of the system properties. ANNs are also robust to changes in process parameters and can adapt to dynamic processes. However, there is a risk that an ANN adapts too closely to old data and cannot generalize new system states, which can lead to control failure. In addition, the computational effort when using an ANN as a model is higher than with statistical models, which can lead to higher energy costs for control. With regard to transferability to real processes, economic efficiency must also be taken into account. The adaptability of a control AI to different systems can save the costs of designing a specific controller, but requires the use of more expensive computers, which also require higher energy. The transfer of this control AI is probably unsuitable for process controls that are operated with older computers, as in these cases there is a lack of computing power to evaluate a large number of control strategies in real time and at the same time to continuously train the ANN with new data.

# Summary and Outlook

In this work, an adaptive control AI was implemented that uses a sliding time window to learn the temporal relationships between the manipulated and controlled variables of a system and can identify the optimal one from possible control strategies. To this end, the theoretical basics of process control, AI and the MPC control method used were explained. The rule AI was then implemented in Python and tested on a virtual proxy system.

It has been found that using the ReLU activation function can result in dead neurons. To avoid this, the leaky ReLU function was used. Furthermore, it was observed that the size of the time window in this system does not have a major influence on the control success. In order to reduce the computational effort and increase the stability of the control, a secondary condition was introduced when calculating the optimal control strategy that limits the change in the manipulated variable between two points in time. It was observed that the rule AI is able to stabilize itself again after disruptions in the system. In addition, the adaptability of the control AI to a change in the reference variable was tested in the event of dynamic changes in the internal state variables and under the influence of various disturbance variables. It was found that the control AI is able to adapt to dynamic changes and to achieve and maintain the reference variable in a stable manner with only minor fluctuations. The process parameters of the proxy system have the greatest influence on successful control, and they must be selected in such a way that successful control is possible in the first place. Compared to a static system, size is more important in a dynamic system, since temporal relationships are only learned to a limited extent in a dynamic system with a small size.

These results show the potential for integrating control AI into certain physical processes. For real-time control, it is important that sufficient computing power is available. A graphics processing unit (GPU) with high memory capacity, a CPU with multiple cores and a sufficiently large RAM for storing the time series should be used. The graphics processor plays the most important role as it allows many computationally intensive matrix operations to be executed in parallel, which ensures fast prediction and training of the ANN.

However, it should be noted that further testing and analysis needs to be carried out to confirm the effectiveness and applicability of rule AI in real processes. For this purpose, the control AI can be used and tested in future work on a bioreactor for pH control, taking into account possible adjustments by incorporating additional safety conditions. Additionally, since LSTM networks are optimized for predicting sequential data, it can be tested whether an LSTM network is better suited to predicting a time series than a normal DNN.

1. bibliography

[1] Corriou, J.-P., Process Control: Theory and Applications, 2nd edn., Springer International Publishing; Imprint: Springer, Cham, 2018.

[2] Hewing, L., Wabersich, KP, Menner, M., Zeilinger, MN, Learning-Based Model Predictive Control: Toward Safe Learning in Control. Annu. Rev. Control Robot. Auton. Syst. 2020, 3, 269–296, DOI: 10.1146/annurev-control-090419-075625.

[3] Grimble, Michael J., Johnson, Michael A., Camacho, EF, Bordons, C., Model Predictive control, Springer London, London, 2007.

[4] Lunze, J., Control engineering 1: System theoretical foundations, analysis and design of single-loop controls, 5th edn., Springer, Berlin, 2006.

[5] Horn, J., structural image and transmission elements, in: Plaßmann, W., Schulz, D. (Ed.). Manual Electrical Engineering, Springer Fachmedien Wiesbaden, Wiesbaden, 2016, pp. 847–852.

[6] John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon, A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence.

[7] Gugerty, L., Newell and Simon's Logic Theorist: Historical Background and Impact on Cognitive Modeling.

[8] Thon, C., Finke, B., Kwade, A., Schilde, C., Artificial Intelligence in Process Engineering. Advanced Intelligent Systems 2021, 3, 2000261, DOI: 10.1002/aisy.202000261.

[9] Samuel, AL, Some Studies in Machine Learning Using the Game of Checkers. IBM J. Res. & Dev. 1959, 3, 210-229, DOI: 10.1147/rd.33.0210.

[10] Ayon Dey, Machine Learning Algorithms: A Review. International Journal of Computer Science and Information Technologies 2016, 1174–1179.

[11] Alzubi, J., Nayyar, A., Kumar, A., Machine Learning from Theory to Algorithms: An Overview. J. Phys.: Conf. Ser. 2018, 1142, 12012, DOI: 10.1088/1742-6596/1142/1/012012.

[12] Steven L. Brunton, J. Nathan Kutz, Data Driven Science & Engineering: Machine Learning, Dynamical Systems, and Control.

[13] Lorberfeld, A., Machine Learning Algorithms In Layman's Terms, Part 1. Towards Data Science, March 2, 2019. https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-1- d0368d769a7b. Accessed December 8, 2022.

[14] Cohen, G., Afshar, S., Tapson, J., van Schaik, A., EMNIST: Extending MNIST to handwritten letters, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp . 2921-2926.

[15] Sutton, RS, Barto, A., Reinforcement learning: An introduction, The MIT Press, Cambridge, Massachusetts, London, England, 2018.

[16] blackburn, Reinforcement Learning Markov Decision Process (Part 1). Towards Data Science, July 18, 2019. https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da. Accessed December 9, 2022.

[17] Kaelbling LP, Littman ML, Moore AW, Reinforcement Learning: A Survey. jair 1996, 4, 237–285, DOI: 10.1613/jair.301.

[18] Christoph Thon, Marvin Röhl, Somayeh Hosseinhashemi, Arno Kwade, Carsten Schilde, Artificial Intelligence and Evolutionary Approaches in Particle Technology. KONA.

[19] Anders Krogh, What are artificial neural networks? computational biology.

[20] Sydenham, PH, Thorn, R., Handbook of measuring system design: 129: Artificial Neural Networks, Wiley, Chichester England, 2005.

[21] Sharma, S., Sharma, S., Athaiya, A., Activation Functions In Neural Networks. IJEAST 2020, 04, 310-316, DOI: 10.33564/ijeast.2020.v04i12.054.

[22] Nielsen, MA, Neural Networks and Deep Learning, Determination Press, 2015.

[23] Agarap, AF, Deep Learning using Rectified Linear Units (ReLU), 2018.

[24] Saurabh, Backpropagation – Algorithm For Training A Neural Network. Edureka, December 7, 2017. https://www.edureka.co/blog/backpropagation/. Accessed December 12, 2022.

[25] Google Developers, Classification: True vs. False and Positive vs. Negative  |  Machine Learning  |  Google Developers. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative?hl=en. Accessed December 13, 2022.

[26] Google Developers, Classification: Accuracy  |  Machine Learning  |  Google Developers. https://developers.google.com/machine-learning/crash-course/classification/accuracy?hl=en. Accessed December 13, 2022.

[27] Google Developers, Classification: Precision and Recall  |  Machine Learning  |  Google Developers. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall?hl=en. Accessed December 13, 2022.

[28] Hyndman, RJ, Koehler, AB, Another look at measures of forecast accuracy. International Journal of Forecasting 2006, 22, 679–688, DOI: 10.1016/j.ijforecast.2006.03.001.

[29] MV Shcherbakov, A. Brebels, NL Shcherbakova, AP Tyukov et al., A survey of forecast error measures. World Applied Sciences Journal 2013, 24, 171–176, DOI: 10.5829/idosi.wasj.2013.24.itmies.80032.

[30] Bontempi, G., Ben Taieb, S., Le Borgne, Y.-A., Machine Learning Strategies for Time Series Forecasting, in: Aufaure, M.-A., Zimányi, E. (Ed.). Business Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 62–77.

[31] HS Hota, Richa Handa, AK Shrivas, Time Series Data Prediction Using Sliding Window Based RBF Neural Network. International Journal of Computational Intelligence Research 2017.

[32] Thomas G. Dietterich, Machine Learning for Sequential Data: A Review.

[33] Taieb, SB, Bontempi, G., Atiya, A., Sorjamaa, A., A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition, 2011.

[34] KS Holkar, LM Waghmare, An Overview of Model Predictive Control. International Journal of Control and Automation International Journal of Control and Automation 2010.

[35] Brownlee, J., Time Series Forecasting as Supervised Learning. Machine Learning Mastery, December 4, 2016. https://machinelearningmastery.com/time-series-forecasting-supervised-learning/. Accessed December 1, 2022.

[36] Brownlee, J., How to Convert a Time Series to a Supervised Learning Problem in Python. Machine Learning Mastery, May 7, 2017. https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/. Accessed December 1, 2022.

[37] Team, K., Keras documentation: About Keras. https://keras.io/about/. Accessed January 12, 2023.

[38] GeeksforGeeks, Deque in Python - GeeksforGeeks. https://www.geeksforgeeks.org/deque-in-python/. Accessed January 9, 2023.

[39] GeeksforGeeks, Epsilon-Greedy Algorithm in Reinforcement Learning - GeeksforGeeks. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/. Accessed January 12, 2023.

[40] ECOOP '95 - object-oriented programming: 9th European conference, Åarhus, Denmark, August 7 - 11, 1995 ; proceedings, Springer, 1995.

[41] Jastram, A., Langschwager, F., Kragl, U., Reactors for special technical-chemical processes: Biochemical reactors, in: Reschetilowski, W. (Ed.). Handbook of Chemical Reactors, Springer Berlin Heidelberg, Berlin, Heidelberg, 2019, pp. 1–39.

[42] ML | Underfitting and overfitting. GeeksforGeeks, November 23, 2017. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. Accessed January 21, 2023.

# List of illustrations

[Figure 1: Block diagram of a process 12](#_Toc126152034)

[Figure 2: Structure of a control loop. Created based on [4] 13](#_Toc126152035)

[Figure 3: Step response of a second-order system [4] 15](#_Toc126152036)

[Figure 4: Representation of the square wave pulse and the Dirac pulse [4] 16](#_Toc126152037)

[Figure 5: Impulse response of a second-order system [4] 16](#_Toc126152038)

[Figure 6: Step response of oscillatory PT2 elements with small damping [4] 18](#_Toc126152039)

[Figure 7: Number of publications of different ML methods on Scopus and qualitative representation of the respective AI eras over time [8] 20](#_Toc126152040)

[Figure 8: Machine learning classes with the respective algorithms. Created based on [8, 12, 13]. 22](#_Toc126152041)

[Figure 9: Reinforcement learning process [16] 23](#_Toc126152042)

[Figure 10: Neuron of a mammal [20] 25](#_Toc126152043)

[Figure 11: Left: Representation of an artificial neural network. Right: Representation of a single neuron with the weighted inputs and the activation function. [18] 25](#_Toc126152044)

[Figure 12: Illustration of the Sigmoid, Tanh and ReLU activation function 27](#_Toc126152045)

[Figure 13: Influence of weighting on error. Created based on [24] 29](#_Toc126152046)

[Figure 14: Example of a sliding time window 34](#_Toc126152047)

[Figure 15: Basic structure of MPC. Created based on [3]. 37](#_Toc126152048)

[Figure 16: Prediction of a control strategy over a sliding time window at MPC 39](#_Toc126152049)

[Figure 17: Flow of the main program 42](#_Toc126152050)

[Figure 18: Structure of a time window at time t with manipulated and controlled variables 45](#_Toc126152051)

[Figure 19: Structure of the input and output layer of the rule AI 47](#_Toc126152052)

[Figure 20: Schematic representation of a bioreactor in the form of a stirred tank [41] 51](#_Toc126152053)

[Figure 21: Influence of k1 and k2 on the system with u(t) = 0 and z(t) = 0 52](#_Toc126152054)

[Figure 22: MSE and MAPE for the individual time points at which three output neurons are permanently zero 55](#_Toc126152055)

[Figure 23: Simulation progression for a control AI in which three output neurons are permanently zero 56](#_Toc126152056)

[Figure 24: Simulation progression for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s 58](#_Toc126152057)

[Figure 25: MSE and MAPE for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s 58](#_Toc126152058)

[Figure 26: Squared error across all control strategies and time steps of the first prediction for t = 11.2s 60](#_Toc126152059)

[Figure 27: Squared error across all control strategies and time steps of the last prediction for t = 199.8s 60](#_Toc126152060)

[Figure 28: MAPE for ttrain = 5s (left) and ttrain = 20s (right) 61](#_Toc126152061)

[Figure 29: Course of the manipulated variable for ttrain = 5s (top) and ttrain = 20s (bottom) 62](#_Toc126152062)

[Figure 30: MSE and MAPE for tact = 500ms 63](#_Toc126152063)

[Figure 31: MAPE total error and MAPE for each time point for h = 8 and m = 2 64](#_Toc126152064)

[Figure 32: MAPE total error and MAPE for each time point for h = 8 and m = 4 64](#_Toc126152065)

[Figure 33: MAPE total error for n = 1 (left) and n = 8 (right) 65](#_Toc126152066)

[Figure 34: Simulation progression for ks = 0.3 67](#_Toc126152067)

[Figure 35: Simulation curve for high noise 68](#_Toc126152068)

[Figure 36: MSE and MAPE for high noise 68](#_Toc126152069)

[Figure 37: Simulation process when excited by interference pulses 69](#_Toc126152070)

[Figure 38: MSE and MAPE when excited by interference pulses 69](#_Toc126152071)

[Figure 39: Simulation process with dynamic disturbance behavior with A = 0.5 70](#_Toc126152072)

[Figure 40: Simulation progression with dynamic disturbance behavior with A = 0.2 71](#_Toc126152073)

[Figure 41: Illustration of underfitting and overfitting [42] 72](#_Toc126152074)

[Figure 42: Simulation curve for Δumax = 2 73](#_Toc126152075)

[Figure 43: Simulation process when changing the reference variable 76](#_Toc126152076)

[Figure 44: MSE and MAPE when changing the reference variable 77](#_Toc126152077)

[Figure 45: Simulation process with delayed jump change of k1 and k2 78](#_Toc126152078)

[Figure 46: MSE and MAPE with delayed jump change of k1 and k2 79](#_Toc126152079)

[Figure 47: Simulation progression with periodic changes to k1 and k2 80](#_Toc126152080)

[Figure 48: MSE and MAPE with periodic changes in k1 and k2 80](#_Toc126152081)

[Figure 49: MSE and MAPE with changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 4, m = 3, h = 4 81](#_Toc126152082)

[Figure 50: Simulation process with changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 4, m = 3, h = 4 82](#_Toc126152083)

[Figure 51: Simulation process with changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 1, m = 1, h = 2 83](#_Toc126152084)

[Figure 52: MSE for changes in the reference variable, dynamic changes in the state variables and the influence of disturbances for n = 1, m = 1, h = 2 84](#_Toc126152085)

# Table directory

[Table 1: Functional relationships and block symbols of proportional elements [4, 5] 17](#_Toc126152086)

[Table 2: Functional relationship and block symbol of the I-member [4, 5] 18](#_Toc126152087)

[Table 3: Functional relationships and block symbols of differentiating elements [4, 5] 19](#_Toc126152088)

[Table 4: Functional relationship and block symbol of the dead time element [4, 5] 19](#_Toc126152089)

[Table 5: Possible results of a binary classification 30](#_Toc126152090)

[Table 6: Example of a univariate time series 43](#_Toc126152091)

[Table 7: Example of converting a univariate time series into supervised learning 43](#_Toc126152092)

[Table 8: Multivariate time series with manipulated and controlled variables 44](#_Toc126152093)

[Table 9: Ranking of the factors influencing the success of the regulation 74](#_Toc126152094)

# Attachment

## Python script from “main.py”

fromsimulation.simulationimportSimKeys  
fromsimulation.simulationimportsimulation  
fromsimulation.simulationimportSimConfig  
fromexportimportData  
fromcontrol.control\_aiimportAgent, AgentConfig  
  
  
defrun\_simulation(s\_config:SimConfig, agent\_configs): sim\_time =200,000sim\_steps =int(sim\_time / s\_config.dt)  
  
 foragent\_configinagent\_configs: sim = Simulation(s\_config) agent = Agent(agent\_config, SimKeys.ACTION, SimKeys.VALUE) data = Data(plots\_every=50\*pow(10,3),agent\_config=agent\_config,sim\_config=s\_config)  
  
target, time = sim.reset()  
  
 for\_inrange(sim\_steps):*# Change reference here if needed# ##################################*action, predictions = agent.choose\_action(target, time, sim.current\_u)  
  
 ifpredictionsis not None:*# Calculate actual trajectories*values ​​= sim.calculate\_strategies(agent.get\_allowed\_strategies(sim.current\_u), agent.h, agent.act\_every) data.record\_errors(predictions, values, time)  
  
zi =0*# Add disturbance impulse z here if needed # ######################################### ###*target\_, time, temp = sim.step(action, zi, agent.reference)ifactionis not None: agent.record(temp) agent.train\_model(time) target = target\_ data.finish(sim)  
  
  
defget\_agent\_configs():*# Create all the Configs which should be compared* c1 = AgentConfig(reference=8th,action\_space=[0,1,2,3,4,5,6,7,8th,9,10]) c1.n =4  
 c1.m =2c1.h =4c1.act\_every =200c1.train\_every =10\_000  
  
 return[c1]  
  
  
if\_\_name\_\_ =='\_\_Main\_\_': sim\_config = SimConfig() sim\_config.noise =0.5sim\_config.dt =10*# Time in ms*sim\_config.start\_value =7sim\_config.ks =0.15sim\_config.stable\_value =7  
  
 run\_simulation(sim\_config, get\_agent\_configs())

## Python script of the rule AI from “control\_ai.py”

importnumpyasnp  
importcontrol.utilsasutils  
importpandasaspd  
fromcollectionsimportdeque  
fromtensorflow.python.keras.modelsimportSequential  
fromtensorflow.python.keras.layersimportDense, LeakyReLU  
fromtensorflow.python.keras.optimizer\_v2.adamimportAdam  
  
  
classAgentConfig:  
  
 def\_\_init\_\_(self, reference, action\_space, du\_max=None, sys\_lim=None):self.max\_mem\_size =10\_000self.n =2*# h needs to be greater than m and > 0, m can be 0 # Control horizon*self.m =2*# Prediction horizon*self.h =4self.epsilon =1self.epsilon\_dec =0.7self.epsilon\_min =0.01self.batch\_size =10self.epochs =50*# Train model every 10 seconds*self.train\_every =10\_000*# Act every 200ms*self.act\_every =200self.alpha =0.7self.reference = referenceself.action\_space = action\_spaceself.du\_max = du\_maxself.sys\_lim = sys\_lim  
  
  
defbuild\_network(input\_size, output\_size, n\_hidden1, n\_hidden2): model = Sequential([ Dense(n\_hidden1,input\_shape=(input\_size,)), LeakyReLU(alpha=0.1), Dense(n\_hidden2), LeakyReLU(alpha=0.1), Dense(output\_size), LeakyReLU(alpha=0.1) ]) model.compile(optimizer=Adam(learning\_rate=0.0001),loess='mse')  
  
 returnmodel  
  
  
classagent:def\_\_init\_\_(self, config:AgentConfig, c\_key, t\_key):assertconfig.h > config.mself.input\_size = (config.n +1) \*2+ config.m*# System limits (upper limit and lower limit) -> if not None the data will be normalized to that range*self.sys\_lim = config.sys\_limself.action\_space = config.action\_spaceself.du\_max = config.du\_maxself.alpha = config.alphaself.epsilon = config.epsilonself.epsilon\_dec = config.epsilon\_decself.epsilon\_min = config.epsilon\_minself.batch\_size = config.batch\_sizeself.epochs = config.epochsself.reference = config.referenceself.n = config.nself.control\_key = c\_keyself.target\_key = t\_keyself.m = config.mself.h = config.hself.train\_every = config.train\_everyself.last\_trained =0self.act\_every = config.act\_everyself.last\_acted =0self.memory = deque(maxlen=config.max\_mem\_size)self.training\_history = []*# Temporary memory is cleared after every training iteration*self.temp\_memory = []self.model = build\_network(self.input\_size,self.H,256,128)self.control\_strategies = utils.compute\_all\_possible\_strategies(self.m,self.action\_space,you\_max=self.du\_max)  
  
 def\_get\_prediction\_data(self, current\_y, allowed\_strategies):*# Get the target and control values ​​for the last n time steps from the memory*data = [[self.memory[i][self.target\_key],self.memory[i][self.control\_key]]foriinrange(len(self.memory) -self.n,len(self.memory))] data = np.array(data,dtype=np.float)  
  
 ifself.sys\_limis not None:*# Normalize Data to system limits*data[:,0] = utils.normalize(data[:,0],self.sys\_lim[0],self.sys\_lim[1]) data[:,1] = utils.normalize(data[:,1],min(self.action\_space),Max(self.action\_space)) allowed\_strategies = utils.normalize(allowed\_strategies,min(self.action\_space),Max(self.action\_space))  
  
 *# Flatten array since its currently nested*data = data.flatten()*# Add current y at the end of the array*ifself.sys\_limis not None: data = np.append(data, utils.normalize(current\_y,self.sys\_lim[0],self.sys\_lim[1]))else: data = np.append(data, current\_y)  
  
 *# Expand dimensions of array so that the row can be copied*data = np.expand\_dims(data,axis=0)*# Copy the row to fit the same shape as the control strategies so that they can be concatenated*data = np.repeat(data,repeats=allowed\_strategies.shape[0],axis=0) data = np.concatenate((data, allowed\_strategies),axis=1)  
  
 returndata  
  
 defget\_allowed\_strategies(self, current\_u):ifself.you\_maxis None:returnself.control\_strategies  
  
 *# Filter possible control strategies with constraint du <= du\_max*diff = np.abs(current\_u -self.control\_strategies[:,0])returnself.control\_strategies[diff <=self.du\_max]  
  
 defrecord(self, state):*# Save current target and control values ​​in the long term and temporary memory*inputs = [self.target\_key,self.control\_key] y = {k: state[k]forkininputs}self.memory.append(y)self.temp\_memory.append(y)  
  
 defchoose\_action(self, current\_y, t, u):*# Returns next action and all the predictions the ANN made*ift <self.last\_acted +self.act\_every:return None,None  
  
 self.last\_acted = t rand = np.random.random()iflen(self.memory) <self.noredge <self.epsilon:*# Pick random action*action = np.random.choice(self.action\_space)returnaction,None  
  
 allowed\_strategies =self.get\_allowed\_strategies(u) inputs =self.\_get\_prediction\_data(current\_y, np.copy(allowed\_strategies)) predictions =self.model.predict(inputs,batch\_size=4096)ifself.sys\_limis not None: predictions = utils.denormalize(predictions,self.sys\_lim[0],self.sys\_lim[1])  
  
 *# Evaluate control strategies and find the best one (minimal cost)*costs = \_cost\_function(predictions,self.reference,self.alpha) rand = np.random.random()ifedge <=0.1:*# Pick a random strategy of the best 10*idx = np.argpartition(costs,min(10, allowed\_strategies.shape[0] -1)) min\_cost\_idx = np.random.choice(idx)else: min\_cost\_idx = np.argmin(costs)  
  
best\_strategy = allowed\_strategies[min\_cost\_idx]print('BEST STRATEGY:', best\_strategy,'COST:', costs[min\_cost\_idx],'EPSILON:',self.epsilon)  
  
 *# Only use first action of control strategy for the next timestep, after that repeat this process*returnbest\_strategy[0], predictions  
  
 deftrain\_model(self, t):iflen(self.memory) <self.n+self.h+1+self.batch\_sizeort <self.last\_trained +self.train\_every:return  
  
 df = pd.DataFrame.from\_records(self.temp\_memory)  
  
 *# Normalize data*ifself.sys\_limis not None:df[self.target\_key] = utils.normalize(df[[self.target\_key]],self.sys\_lim[0],self.sys\_lim[1]) df[self.control\_key] = utils.normalize(df[[self.control\_key]],min(self.action\_space),Max(self.action\_space))  
  
x\_data, y\_data = utils.convert\_input\_data\_training(df,self.n,self.m,self.H,self.control\_key,self.target\_key)  
  
history =self.model.fit(x\_data, y\_data,batch\_size=self.batch\_size,epochs=self.epochs,verbose=0)self.training\_history.append(history)  
  
 self.epsilon =self.epsilon \*self.epsilon\_decifself.epsilon >self.epsilon\_minelseself.epsilon\_minself.temp\_memory.clear()self.last\_trained = t  
  
  
def\_cost\_function(trajectories, reference, alpha):*''' Calculate cost of trajectories :param trajectories: Predicted trajectories over a given horizon of len(trajectory) :param reference: Value where the system should end up :return: cost of all trajectories '''*h = trajectories.shape[1]  
  
 *# Calculate weights for each element in the trajectory*weights = np.power(alpha, np.flip(np.arange(h)))returnnp.sum((trajectories - reference) \*\*2\* weights,axis=1)

## Python script from “utils.py”

importitertools  
importnumpyasnp  
importpandasaspd  
  
  
defconvert\_input\_data\_training(df:pd.DataFrame, n, m, h, c\_key, t\_key):asserth >= masserth >=1  
  
 inputs = [t\_key, c\_key] cols, names = [], []*# Input sequence (tn, ... t-1)*foriinrange(n, -1, -1): cols.append(df.shift(i))ifi ==0: names += [f'{x}(t)'forxininputs]else: names += [('%hours)'% (x, i))forxininputs]  
  
 *# Input sequence (u\_t, u\_t+1 ..., u\_t+m)*foriinrange(1, m +1): cols.append(df[c\_key].shift(-i)) names += ['%s(t+%d)'%(c\_key,i)]  
  
target\_index =len(name)  
  
 *# Output sequence (t+1, ... t+h)*foriinrange(1, h +1): cols.append(df[t\_key].shift(-i)) names += ['%s(t+%d)'%(t\_key,i)]  
  
 *# Put it all together*data = pd.concat(cols,axis=1) data.columns = names*# Drop rows with NaN values*data.dropna(inplace=True) data.reset\_index(drop=True,inplace=True) data = data.applymap(float.float)  
  
 returndata.iloc[:, :target\_index].copy(), data.iloc[:, target\_index:].copy()  
  
  
defcompute\_all\_possible\_strategies(m, action\_space:list, you\_max=None): strategies = np.array(list(itertools.product(action\_space,repeat=m +1)),dtype=np.float)  
  
 ifyou\_maxis not None:*# Filter out strategies that don't match the constraint*diff = np.abs(np.diff(strategies,axis=1)) strategies = strategies[np.max(diff,axis=1) <= you\_max]  
  
 returnstrategies  
  
  
defnormalize(x, min\_x, max\_x):return(x - min\_x) / (max\_x - min\_x)  
  
  
defdenormalize(y, min\_x, max\_x):returny \* (max\_x - min\_x) + min\_x

## Python script of the simulation from “simulation.py”

importnumpyasnp  
fromscipy.integrateimportodeint  
  
  
classSimKeys: TIME ="time"VALUE ="value"CHANGE ="change"DELTA ="delta"ACTION ="action"REFERENCE ="reference"K1 ="k1"K2 ="k2"Z ="z"  
  
  
defk1\_function(t):*# if t < 100: # return 0.2 # elif t < 200: # return 0.4 - 0.2 \* math.exp(-(t - 100) / 5) # else: # return 0.2 + 0.2 \* math.exp(-(t - 200) / 2)  
  
#return 0.1 \* math.sin(0.2\*t) + 0.2*return0.2  
  
  
defk2\_function(t):*# if t < 100: # return 0.05 # elif t < 200: # return 0.5 - 0.45 \* math.exp(-(t - 100) / 10) # else: # return 0.05 + 0.45 \* math.exp(-(t - 200) / 2)  
  
#return 0.2 \* math.cos(0.5\*t) + 0.3*return0.05  
  
  
defzd\_function(t):*#return 0.2 \* math.sin(0.1 \* t)*return0  
  
  
classSimConfig:*# dt in ms*def\_\_init\_\_(self):self.dt =10self.k1 =lambdat: k1\_function(t)self.k2 =lambdat: k2\_function(t)self.zd =lambdat: zd\_function(t)self.ks =0.15self.min\_value =0  
 self.max\_value =14self.stable\_value =7self.start\_value =7self.noise =0.5  
  
  
classsimulation:def\_\_init\_\_(self, simulation\_config:SimConfig):self.dt = simulation\_config.dtself.start\_val = simulation\_config.start\_valueself.current\_val = simulation\_config.start\_valueself.stable\_val = simulation\_config.stable\_valueself.k1 = simulation\_config.k1self.k2 = simulation\_config.k2self.ks = simulation\_config.ksself.zd = simulation\_config.zdself.min\_val = simulation\_config.min\_valueself.max\_val = simulation\_config.max\_valueself.noise = simulation\_config.noiseself.df = []self.current\_t =0self.current\_u =0  
  
 defstep(self, u, zi, w):ifuis not None:self.current\_u = u  
  
 *# Calculate disturbance*noise = np.random.normal(0,self.noise,1) z = noise[0] + zi +self.zd(self.current\_t/1000) change =self.\_calculate\_change(self.current\_u, z)  
  
 *# Old system state (before step)*temp = { SimKeys.TIME:self.current\_t, SimKeys.VALUE:self.current\_val, SimKeys.DELTA:self.current\_val - w, SimKeys.CHANGE: change, SimKeys.ACTION:self.current\_u, SimKeys.REFERENCE: w, SimKeys.K1:self.k1(self.current\_t/1000), SimKeys.K2:self.k2(self.current\_t/1000), SimKeys.Z: z }self.df.append(temp)  
  
 self.current\_val += change  
  
 *# Cap value between min and max*ifself.current\_val <self.min\_val:self.current\_val =self.min\_valelifself.current\_val >self.max\_val:self.current\_val =self.max\_val  
  
 self.current\_t +=self.dt  
  
 returnself.current\_val,self.current\_t, temp  
  
 defreset(self):self.current\_t =0self.current\_u =0self.df.clear()self.current\_val =self.start\_val  
  
 returnself.current\_val,self.current\_t  
  
 def\_calculate\_change(self, u, z):*# Divide by 1000 to convert from ms -> s*return(-self.k1(self.current\_t/1000) \*Section(self.stable\_val -self.current\_val) -self.k2(self.current\_t/1000) +self.ks \* u + z) \*self.dt /1000  
  
 defcalculate\_strategies(self, strategies:np.array, p\_horizon, act\_every):*# Calculate the actual influence of the different control strategies on the system*defmodel(y, t, u, dt, t0):return-(self.k1(t0 + t) \*Section(y -self.stable\_val) +self.k2(t0 + t)) +self.ks \* u[:,min(int(t / dt), u.shape[1] -1)] +self.zd(t0 + t)  
  
delta\_t = act\_every /1000t\_eval = np.arange(0, p\_horizon +1,1) \* delta\_t y0 = np.ones(shape=strategies.shape[0]) \*self.current\_val  
  
values ​​= odeint(model, y0, t\_eval,bad=(strategies, delta\_t,self.current\_t/1000))*# Transpose matrix, because somehow it is in the wrong order*values ​​= np.array(values).T*# Drop first column since it is the one for t=0 which we don't need*values ​​= values[:,1:]  
  
 returnvalues

## Python script for visualizing the results from “export.py”

importnumpyasnp  
importpandasaspd  
importos  
importpathlib  
importdatetime  
importos.path  
importmatplotlib.pyplotasplt  
fromsimulation.simulationimportSimKeys, simulation  
importmatplotlib.cmascm  
  
  
classExpKeys: TIME ="Time (ms)"MSE\_TOTAL ="mse\_total"MAPE\_TOTAL ="mape\_total"MAE\_TOTAL ="mae\_total"  
  
  
classData:def\_\_init\_\_(self, plots\_every, agent\_config, sim\_config):self.plots\_every = plots\_everyself.last\_plotted =Noneself.last\_individual\_error =Noneself.last\_individual\_time =0self.total\_errors = []self.mse\_tsteps =Noneself.mape\_tsteps =Noneself.mae\_tsteps =Noneself.agent\_config = agent\_configself.sim\_config = sim\_config  
  
 self.path,self.image\_path =self.\_create\_folder()self.\_create\_meta\_data()  
  
 def\_create\_folder(self): parent\_path = os.path.join(pathlib.Path().resolve(),"data")if notos.path.exists(parent\_path): os.makedirs(parent\_path)  
  
folder\_name ="Simulation\_"+ datetime.datetime.now().strftime('%Y-%m-%d\_%H-%M-%S') dir\_name = os.path.join(parent\_path, folder\_name) os.makedirs(dir\_name)  
  
im\_path = os.path.join(dir\_name,"Graphs") os.makedirs(im\_path)  
  
 returndir\_name, im\_path  
  
 def\_create\_meta\_data(self): path1 = os.path.join(self.path,"sim\_config.txt") path2 = os.path.join(self.path,"agent\_config.txt")  
  
 withopen(path1,"w")asf1,open(path2,"w")asf2: f1.write(''.join(["%s = %s\n"% (k, v)fork, vinself.sim\_config.\_\_dict\_\_.items()])) f2.write(''.join(["%s = %s\n"% (k, v)fork, vinself.agent\_config.\_\_dict\_\_.items()]))  
  
 deffinish(self, sim:simulation): dir\_name = os.path.join(self.image\_path,"summary") os.makedirs(dir\_name)  
  
create\_sim\_linechart(sim, dir\_name)  
  
 iflen(self.total\_errors) ==0:returntotal\_error = pd.DataFrame.from\_records(self.total\_errors) time = total\_error[[ExpKeys.TIME]].div(1000)ifself.mse\_tstepsis not None andself.mape\_tstepsis not None andself.mae\_tstepsis not None: create\_error\_linechart\_steps(time.to\_numpy(),self.mse\_tsteps,"MSE[-]", dir\_name) create\_error\_linechart\_steps(time.to\_numpy(),self.mape\_tsteps,"MAPE [%]", dir\_name) create\_error\_linechart\_steps(time.to\_numpy(),self.mae\_tsteps,"MAE [-]", dir\_name)  
  
create\_error\_linechart\_total(time.to\_numpy(), total\_error[[ExpKeys.MSE\_TOTAL]].to\_numpy(),"MSE[-]", dir\_name) create\_error\_linechart\_total(time.to\_numpy(), total\_error[[ExpKeys.MAPE\_TOTAL]].to\_numpy(),"MAPE [%]", dir\_name) create\_error\_linechart\_total(time.to\_numpy(), total\_error[[ExpKeys.MAE\_TOTAL]].to\_numpy(),"MAE [-]", dir\_name)  
  
 ifself.last\_individual\_erroris not None: f\_name\_heatmap ="error\_heatmap.png"f\_name\_scatter ="3d\_scatter\_error.png"create\_error\_scatter\_3d(self.last\_individual\_error,self.last\_individual\_time, os.path.join(dir\_name, f\_name\_scatter)) create\_error\_heatmap(self.last\_individual\_error,self.last\_individual\_time, os.path.join(dir\_name, f\_name\_heatmap))  
  
 defrecord\_errors(self, predictions:np.array, actual\_values:np.array, time): mse\_total = np.mean((predictions - actual\_values) \*\*2) mape\_total = np.mean(np.abs((actual\_values ​​- predictions) / actual\_values)) \*100mae\_total = np.mean(np.abs(actual\_values ​​- predictions))  
  
total = { ExpKeys.TIME: time, ExpKeys.MSE\_TOTAL: mse\_total, ExpKeys.MAPE\_TOTAL: mape\_total, ExpKeys.MAE\_TOTAL: mae\_total  
}self.total\_errors.append(total)  
  
mse\_tsteps = np.mean((predictions - actual\_values) \*\*2,axis=0) mape\_tsteps = np.mean(np.abs((actual\_values ​​- predictions) / actual\_values),axis=0) \*100mae\_tsteps = np.mean(np.abs(actual\_values ​​- predictions),axis=0)  
  
 *# Add different error measures for each time step to an array*ifself.mse\_tstepsis None:self.mse\_tsteps = np.copy(mse\_tsteps)self.mse\_tsteps = np.expand\_dims(self.mse\_tsteps,axis=0)else:self.mse\_tsteps = np.vstack([self.mse\_tsteps, mse\_tsteps])  
  
 ifself.mape\_tstepsis None:self.mape\_tsteps = np.copy(mape\_tsteps)self.mape\_tsteps = np.expand\_dims(self.mape\_tsteps,axis=0)else:self.mape\_tsteps = np.vstack([self.mape\_tsteps, mape\_tsteps])  
  
 ifself.mae\_tstepsis None:self.mae\_tsteps = np.copy(mae\_tsteps)self.mae\_tsteps = np.expand\_dims(self.mae\_tsteps,axis=0)else:self.mae\_tsteps = np.vstack([self.mae\_tsteps, mae\_tsteps])  
  
quad\_individual = (predictions - actual\_values) \*\*2self.last\_individual\_error = quad\_individualself.last\_individual\_time = time  
  
 *# Graphs are only saved after a set interval*ifself.last\_plottedis None ortime >=self.last\_plotted +self.plots\_every:self.last\_plotted =0ifself.last\_plottedis None elsetime  
  
mse\_strategies = np.mean((predictions - actual\_values) \*\*2,axis=1) mape\_strategies = np.mean(np.abs((actual\_values ​​- predictions) / actual\_values),axis=1) \*100mae\_strategies = np.mean(np.abs(actual\_values ​​- predictions),axis=1)  
  
f\_name =f"Time\_{str(time /1000)}s"dir\_name = os.path.join(self.image\_path, f\_name) os.makedirs(dir\_name) f\_name\_heatmap ="error\_heatmap.png"f\_name\_scatter ="3d\_scatter\_error.png"  
  
 create\_error\_heatmap(quad\_individual, time, os.path.join(dir\_name, f\_name\_heatmap)) create\_error\_scatter\_3d(quad\_individual, time, os.path.join(dir\_name, f\_name\_scatter))  
  
create\_error\_bar\_chart\_strategies(mse\_strategies,"MSE[-]", time, dir\_name)  
create\_error\_bar\_chart\_strategies(mape\_strategies,"MAPE [%]", time, dir\_name) create\_error\_bar\_chart\_strategies(mae\_strategies,"MAE [-]", time, dir\_name)  
  
create\_error\_bar\_chart\_steps(mse\_tsteps,"MSE[-]", time, dir\_name) create\_error\_bar\_chart\_steps(mape\_tsteps,"MAPE [%]", time, dir\_name) create\_error\_bar\_chart\_steps(mae\_tsteps,"MAE [-]", time, dir\_name)  
  
  
defcreate\_error\_heatmap(errors:np.array, time, path):*# First Axis are the different strategies, second axis are the future time step*plt.ioff() x\_labels = [f"t+{str(i +1)}"foriinrange(errors.shape[1])] fig, ax = plt.subplots(figsize=(7,6)) im = ax.imshow(errors,interpolation='nearest',cmap='viridis',aspect='automobile') ax.set\_xticks(np.arange(len(x\_labels)),labels=x\_labels) ax.set\_xlabel("Time step [-]") ax.set\_ylabel("Strategy Index [-]") plt.colorbar(im) plt.title(f"Individual squared error for t={str(time /1000)}s") plt.savefig(path) plt.clf() plt.close()  
  
  
defcreate\_error\_scatter\_3d(errors:np.array, time, path): plt.ioff()fig= plt.figure() ax = plt.axes(projection="3d") x\_data = np.arange(0, errors.shape[0],1) y\_data = np.arange(0, errors.shape[1],1) X, Y = np.meshgrid(x\_data, y\_data) y\_labels = [f"t+{str(i +1)}"foriinrange(errors.shape[1])] ax.scatter(X, Y, errors,s=2,c=Y,vmin=0,vmax=errors.shape[1],cmap=cm.get\_cmap("plasma")) ax.set\_yticks(np.arange(len(y\_labels)),labels=y\_labels) ax.set\_xlabel("Strategy Index [-]") ax.set\_ylabel("Time step [-]") ax.set\_zlabel("Square error [-]")  
  
plt.title(f"Individual squared error for t={str(time /1000)}s")  
  
plt.savefig(path) plt.clf() plt.close()  
  
  
defcreate\_error\_bar\_chart\_strategies(errors:np.array, error\_type, time, path): plt.ioff() plt.figure() plt.bar(np.arange(0, errors.shape[0],1), errors) plt.ylabel(error\_type) plt.xlabel("Strategy Index") plt.title(error\_type +" for predicting all strategies for t="+str(time /1000) +"s") f\_name = error\_type +"\_strategies.png"plt.savefig(os.path.join(path, f\_name)) plt.clf() plt.close()  
  
  
defcreate\_error\_bar\_chart\_steps(errors:np.array, error\_type, time, path): x\_labels = [f"t+{str(i +1)}"foriinrange(errors.shape[0])]  
  
plt.ioff() plt.figure() plt.bar(x\_labels, errors) plt.ylabel(error\_type) plt.xlabel("time step") plt.title(error\_type +" for predicting all time steps for t="+str(time /1000) +"s") f\_name = error\_type +"\_steps.png"plt.savefig(os.path.join(path, f\_name)) plt.clf() plt.close()  
  
  
defcreate\_error\_linechart\_steps(time:np.array, errors:np.array, error\_type, path, ylim=None):*# Here errors is a 2d array that includes the error of every time step over all predictions*plt.ioff() plt.figure() legend = [f"t+{str(i +1)}"foriinrange(errors.shape[1])]  
  
 fortstepinerrors.T: plt.plot(time, tstep)  
  
 ifylimis not None: plt.ylim(ylim) plt.grid(visible=True) plt.ylabel(error\_type) plt.xlabel("time[s]") plt.title(error\_type +"all predictions for the individual points in time") plt.legend(legend) f\_name = error\_type +"\_line\_steps.png"plt.savefig(os.path.join(path, f\_name)) plt.clf() plt.close()  
  
  
defcreate\_error\_linechart\_total(time:np.array, errors:np.array, error\_type, path, ylim=None):*# Here errors is a 2d array that includes the error of every time step over all predictions*plt.ioff() plt.figure() plt.plot(time, errors) plt.grid(visible=True) plt.ylabel(error\_type) plt.xlabel("time[s]")ifylimis not None: plt.ylim(ylim) plt.title(error\_type +"Total error over the entire runtime") f\_name = error\_type +"\_line\_total.png"plt.savefig(os.path.join(path, f\_name)) plt.clf() plt.close()  
  
  
defcreate\_sim\_linechart(sim:simulation, path): plt.ioff() df = pd.DataFrame.from\_records(sim.df) time = df[SimKeys.TIME].div(1000)  
  
fig, axs = plt.subplots(4,1,sharex='row',figsize=(10,8th),gridspec\_kw={'height\_ratios':[4,1,1,1]})  
  
axs[0].plot(time, df[SimKeys.VALUE]) axs[0].plot(time, df[SimKeys.REFERENCE],c='k') axs[0].set\_ylim(bottom=sim.min\_val,Top=sim.max\_val) axs[0].set\_xlim(left=0) axs[0].axhline(sim.stable\_val,color='black',linestyle='dotted') axs[0].legend(["Control variable y","Reference variable w"]) axs[0].set\_xlabel("time[s]") axs[0].set\_ylabel("y(t) [-]") axs[0].grid(visible=True) axs[0].set\_title(“Change of the controlled variable over time”)  
  
axs[1].plot(time, df[SimKeys.ACTION]) axs[1].set\_xlabel("time[s]") axs[1].set\_ylabel("u(t) [-]") axs[1].set\_ylim(bottom=-0.1) axs[1].set\_xlim(left=0) axs[1].grid(visible=True) axs[1].set\_title("Current of the manipulated variable over time")  
  
axs[2].plot(time, df[SimKeys.Z]) axs[2].set\_xlabel("time[s]") axs[2].set\_ylabel("z(t) [-]") axs[2].set\_xlim(left=0) axs[2].grid(visible=True) axs[2].set\_title("Course of disturbances over time")  
  
axs[3].plot(time, df[SimKeys.K1]) axs[3].plot(time, df[SimKeys.K2]) axs[3].set\_ylim(bottom=0) axs[3].set\_xlabel("time[s]")  
axs[3].set\_ylabel("k(t) [-]") axs[3].set\_xlim(left=0) axs[3].legend(["k1","k2"]) axs[3].grid(visible=True) axs[3].set\_title("Current of the state variables k1 and k2 over time")  
  
plt.tight\_layout()  
  
fname ="sim\_linechart.png"plt.savefig(os.path.join(path, fname)) plt.clf() plt.close()