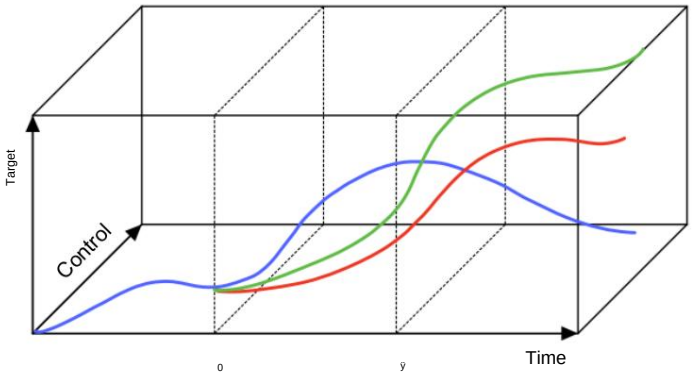
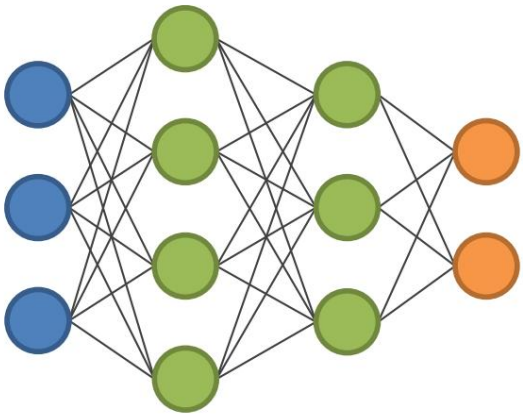
Technische Universität Braunschweig

iPAT Institut für Partikeltechnik

IMPLEMENTATION OF AN ADAPTIVE AUTONOMOUS PROCESS CONTROL USING PREDICTIVE AI MODELING





BACHELOR THESIS

NUMERICAL

PREPARED BY: JONAS BRINKMANN, 5005717

BRAUNSCHWEIG, NOVEMBER 2022

SUPERVISOR: M. SC. CHRISTOPH THON

FIRST EXAMINATOR: PROF. DR.-ING. CARSTEN SCHILDE

SECOND EXAMINATOR: PROF. DR.-ING. ARNO KWADE

statutory declaration

I, Jonas Brinkmann, born on 10.03.2000, hereby declare that I
This bachelor thesis was written independently and only the indicated resources and sources. All literal or analogous quotations that
taken from published or unpublished writings are considered as such
marked.

Brunswick, _____

Jonas Brinkmann

Table of contents

List of abbreviations

abbreviation	Explanation
AI	Artificial Intelligence
ML	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
TO	Artificial intelligence
KNN	Artificial Neural Network
SVM	Support Vector Machines
resume	Rectified Linear Unit
RP	Really positive
FP	False Positive
FN	False negative
RN	Really negative
MSE	Mean Square Error
RMSE	Root Mean Square Error
THERE IS	Mean Absolute Error
MAP	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
( )	-	Control variable
( )	-	Regelabweichung
( )	-	Disturbance
( )	-	Control variable
( )	-	Leading variable
	-	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 a KNN
	-	Activation function of the layer of a KNN
	-	Absolute deviation of a prediction
	-	Actual value at the time
y	-	Predicted value for the time
ÿ	-	Forecast horizon
	%	Percent error of a prediction at the time
	-	Number of past points in time that are used for the prediction can be used
	-	Control horizon
ÿ( )	-	Predicted control deviation for time
(ÿ,)	-	Total costs of a control strategy
	-	Smallest possible value of the manipulated variable
	-	Maximum possible value of the manipulated variable
ÿ	-	Minimum rate of change of the manipulated variable
ÿ	-	Maximum rate of change of the manipulated variable
	-	Number of time steps required for conversion to Time window
	ms	Line interval
	ms	Trainingsintervall
	ms	Size of the time window
	-	Number of new data per training iteration
	-	Set of all possible discrete values for the Control variable
	-	Matrix containing all possible control strategies
	-	Value at which the slope of the virtual System is minimal
1	-	Internal state variable of the virtual system
2	-	Internal state variable of the virtual system

	Random noise of the system
	Size of a discrete disturbance pulse
	Dynamic disturbance behavior of the system

1 Introduction and motivation

In today's industrial production it is crucial
To manage processes efficiently and reliably to ensure high productivity and quality
Process control plays an important role in this, as it helps
contributes to ensuring the stability of the process and minimizing the influence of
disturbances [1]. Especially in process engineering, there are many complex
Processes that are difficult to regulate. Manually implemented, static
Control concepts are often costly and inflexible when it comes to systematic changes.
An ideal control system should be able to adapt itself
to adapt to the requirements of the process, the process without prior knowledge
to regulate the system properties as optimally as possible and to adapt them without great effort to
different systems. The availability of higher computing power
and advances in machine learning have led to a strong
increased interest in the automation of controller design and adaptation
based on historical process data [2].

In this work, a control AI is developed that implements the control procedure Model
Predictive Control (MPC) is used to predict in real time the effects of various
Control strategies for a process over a certain time horizon
and to identify the optimal strategy [3]. The model used is a
artificial neural network is used, which is continuously trained with a
sliding time window is able to increase the forecast accuracy and thus the
To improve control success over time and to adapt to dynamic changes in
Finally, the implemented rule AI is adapted to
a virtual proxy system in different scenarios and discussed whether
the application of rule AI to a real process is possible.

2 Theoretical foundations

2.1 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 by influencing the process from the outside so that it follows a desired Default expires [4].

2.1.1 Process

A process in the process engineering context is a dynamic system that changed by certain input and disturbance variables and by internal State variables such as temperature, pressure or concentration are described. The internal state variables show the temporal change of the system. The The system's response to the input and disturbance variables is determined by the The effect of an input variable on a certain Output variable is usually delayed and can be non-linear. This must be taken into account when designing a controller. A dynamic system is also referred to as a controlled system. The representation of a process in the block diagram is in Figure 1. [1, 4]

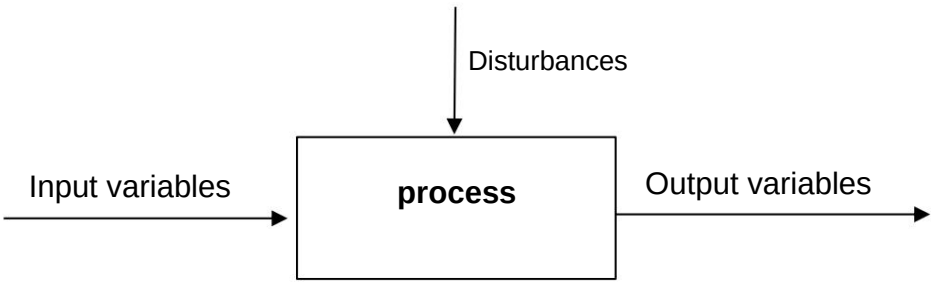


Figure 1: Block diagram of a process

A process can be, for example, a water heating system 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 determines the flow rate and thus the temperature of the

Heating. The temperature change when adjusting the
The temperature controller is delayed. A disturbance could be the fluctuation of the room temperature, for example due to opening a window. A
In this case, the output value could be the measured temperature of the heater.
[4]

2.1.2 Control loop

The aim of a control loop is to control a controlled variable ( ) of a process or
Control system, as close as possible to a given reference variable ( )
To do this, the controlled variable must either be measured directly or obtained from other Measured quantities are calculated.

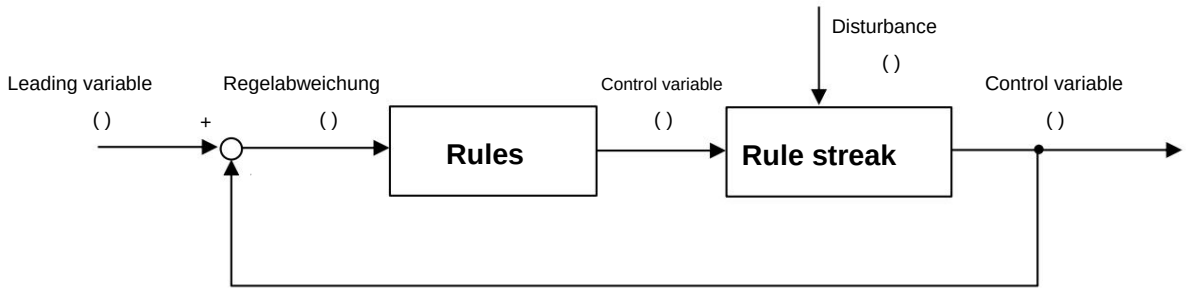


Figure 2: Structure of a control loop. Created based on [4]

Figure 2 shows the structure of a control loop. In its simplified form
It consists of a controller and the system to be controlled, which is controlled by the manipulated variable ( ) and the disturbance variable ( ). The controlled variable becomes negative fed back and compared with the reference variable. The difference between
Controlled variable and reference variable is the control deviation ( ), which is determined by a appropriate controller should be minimized. [4]

$$( ) = ( ) \ddot{y} ( ) \quad (1)$$

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

$$( ) = (( )) \quad (2)$$

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 human is the Controller, the car is the system to be controlled and the speed is the controlled variable, which is measured and displayed on the speedometer. The control variable here is the angle of the accelerator pedal, which affects 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 continue to press the accelerator pedal to maintain the speed.
If it is driving too fast, the driver has to take the accelerator pedal back to to reduce the speed. In this way, the desired Speed reached and maintained.

2.1.3 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 used, for example, of the step and impulse response. The analysis of the step and impulse Impulse response can help to determine the dynamic behavior of a system to predict, assess the stability of the system and understand the system's response to disturbances. The transfer behavior can be modeled by so-called transfer elements. [4]

Step response: The step response describes the temporal reaction of the system to a step-like change in the control variable. It can be used to develop and Optimization of control strategies and thus improvement of system behavior The manipulated variable is determined by the step function with the step height 0. [4]

$$( ) = \begin{cases} 0, & < 0 \\ 1, & \ddot{y}\ 0 \end{cases} \quad (3)$$

$$( ) = \quad _0 \quad ( ) \tag{4}$$

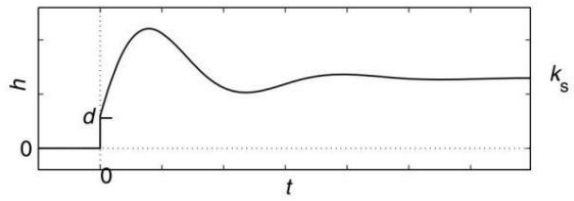


Figure 3: Step response of a second order system [4]

The step response of a step function with the step height 0 = 1 is called the transition function ÿ( ). Figure 3 shows an example of a

Step response or transition function of a second order system. It can be seen that the transition function for ÿ ÿ approaches a certain limit

which represents the static gain of the system. The static gain indicates how strongly the system responds to a change in the input signal

reacts. The value of indicates the extent to which the output variable corresponds to the input variable immediately and without delay. Systems for which ÿ 0 are called

jump-capable systems. [4]

Impulse response: The impulse response describes the temporal response of the system to a pulse-like change in the input variable. It is often used to understand how the system reacts to disturbances. The system is excited by a short pulse, which can be represented by the square pulse ( ).

[4]

$$( ) = \begin{cases} \frac{1}{\quad .} & 0 \ ÿ \ ÿ \\ 0, & \end{cases} \tag{5}$$

For each, the area of the rectangular pulse is equal to one. A system is

excited with a Dirac pulse ( ), which is composed of the rectangular pulse for ÿ 0

results. [4]

$$( ) = \lim_{ÿ0} \quad (6)$$

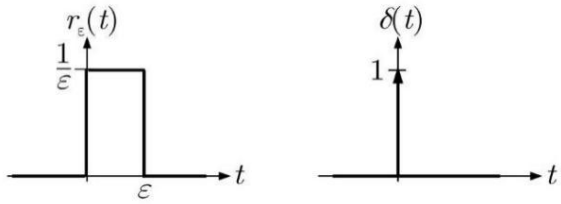


Figure 4: Representation of the rectangular pulse and the Dirac pulse [4]

The Dirac impulse 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 the In reality, the Dirac impulse is not realizable, a similar system behavior can but with the square pulse it can be achieved for a small one. [4]

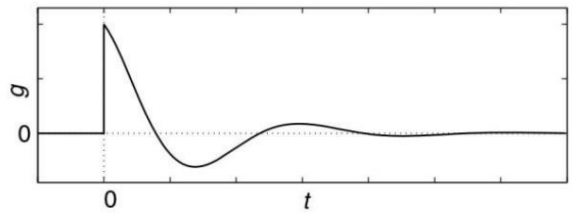


Figure 5: Impulse response of a second order system [4]

The impulse response of a system excited by the Dirac impulse is also called weight function ( ). Figure 5 shows an example of a Impulse response or weight function. A stable system returns after the excitation with a pulse with a time delay back to its initial state.

2.1.4 Transmission elements

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 diagrams and can be classified according to the qualitative course of their Step response can be divided into proportional, integrating, differentiating and dead-time elements. The type of transfer element determines the behavior of the modeled system when the input variable changes. [4, 5]

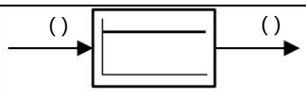
Proportional elements (P-elements): are dynamic transmission elements which, at constant input variable ( ) = in stationary state have an output variable proportional to the value of the input variable. [4]

( ) ~ (7)

Proportional elements can be divided into delayed and undelayed elements. A P-element without delay is a static transfer element in which the Output signal at time t times as large as the input signal. A P-The element with delay is called PTn element, where n is 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 delay-free P-element, the PTn elements have Output size only for ÿ ÿ times as large as the input size. The Output variable does not take the final value immediately, but only after a delay PTn elements can also be realized by connecting n PT1 elements in series . The more elements are connected in series, the longer the The more the input signal is delayed, the smaller the amplitude of the impulse response becomes. [4]

Table 1: Functional relationships and block symbols of proportional elements [4, 5]

Transmission link functional relationship		Blocksymbol
P-element	( ) = ( )	
PT1 element	ÿ( ) + ( ) = ( )	
PT2 element	² ÿ( ) + 2 ÿ( ) + ( ) = ( )	

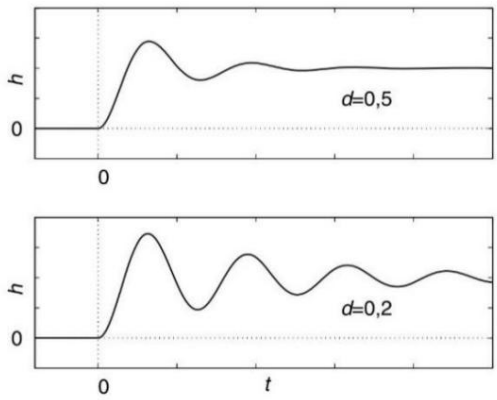


Figure 6: Step response of oscillating PT2 elements with small damping [4]

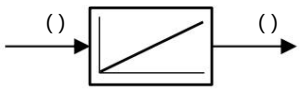
For delay elements, a time constant that determines the delay time and is a damping factor. The variation of and can lead to very different step responses. The influence of the damping factor on the transition function of a PT2 element is shown in Figure 6. If the damping factor is in the range 0 < < 1, the step response overshoots. The smaller the damping factor, the greater the overshoot. [4]

Integrating elements (I-elements): In integrating elements, the output variable is Integration of the input variable. For a constant input variable ( ) = the output variable takes the form of a ramp function. The output variable only takes a constant value if the input variable is zero. stationary system behavior, the output variable is proportional to the integrated Input variable. [4]

$$( ) \sim \ddot{y} \ddot{y}_0 = \quad (8)$$

Table 2: Functional relationship and block symbol of the I-element [4, 5]

Transmission link functional relationship	Blocksymbol

I-Glied	$( ) = \frac{1}{} \int_0 ( ) + (0)$	

Differentiating elements (D-elements): In differentiating elements, the output variable is determined by the change in the input variable. If the input variable is constant, the output variable converges to zero. In stationary

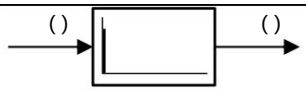
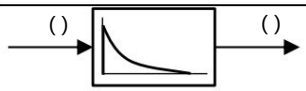
In system behavior, the output variable is proportional to the derived input variable.

[4]

$$( ) \sim \frac{}{} \quad (9)$$

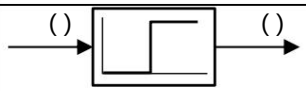
The step response of the delay-free D-element contains a Dirac pulse and is therefore only theoretically possible. In real systems, D-behavior only ever occurs with delay. The delayed D-element is called DTn-element, where n describes the number of delay elements. A DT1 element is replaced by a PT1 Limb additionally dampened. [4]

Table 3: Functional relationships and block symbols of differentiators [4, 5]

Transmission link functional	**relationship**	**Blocksymbol**
D-member	$( ) =$	
DT1 element	$+ ( ) =$	

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 elements combined. [4]

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

Transmission link functional relationship		Blocksymbol
Tt -member	( ) = (ÿ)	

2.2 Artificial Intelligence

The term Artificial Intelligence (AI) was first coined in 1956 at the conference "Dartmouth Summer Research Project on Artificial Intelligence" [6]. This Conference marked the beginning of AI research and brought together leading Scientists come together to explore the potential of "intelligent machines" [6]. An early practical application of AI was the "Logic Theorist" by Newell and Simon from 1956 [7]. This was a program that was developed in the Was able to use some theorems from Whitehead and Russell's "Principia Mathematica" independently [7]. Since then, the field of AI has steadily developed, with phases of euphoria and phases of disappointment. These Development is shown in Figure 7.

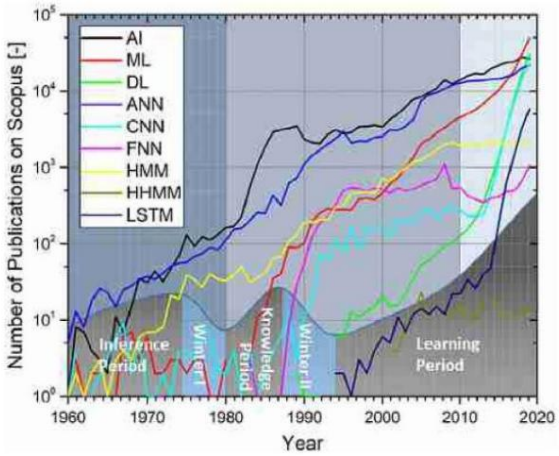


Figure 7: Number of publications of different ML methods on Scopus and qualitative representation of the respective AI epochs over time [8]

One of the most widely used methods of AI is machine learning and
the subordinate Deep Learning, which uses artificial neural networks
as a model [8]. In the following chapters, machine learning
and the sub-learning procedures used to train a model
The construction of artificial neural networks is explained in
Section 2.2.3 explains this in more detail.

2.2.1 Machine Learning

According to Arthur Samuel, machine learning is the field that gives computers the ability
to learn independently and solve problems without explicit
Arthur Samuel became famous in 1959 through a program
who was able to learn the game of checkers independently and after a short
Training time to surpass the developer in this game. [9]

The aim of machine learning is to use existing data as efficiently as possible
learn and generalize new data. A model that can generalize effectively,
is able to make correct predictions for unknown data that it receives during
of the learning process. Machine learning is then used
If no conclusions can be drawn from the available data, for example from an expert
relevant information and contexts can be derived.
Its use is particularly important for complex, non-linear problems.
Advantage. In the field of machine learning, there are many different algorithms,
The choice depends on the specific problem. There is no
generally valid algorithm, so it must be checked in advance which one is suitable for the
suitable for the problem at hand. For a complex model, large
Large amounts of data and high computing power are required, which due to the exponential
Growth in data volumes as well as computing and storage power in recent
ten years. Interest in machine learning has increased
Learning has therefore increased significantly in recent years. [8, 10, 11]

2.2.2 Learning methods

Machine learning can be divided into three main learning methods:

supervised learning, unsupervised learning

Learning) and reinforcement learning [10]. The

different learning methods with the respective algorithms are shown in Figure 8

and are explained in this section.

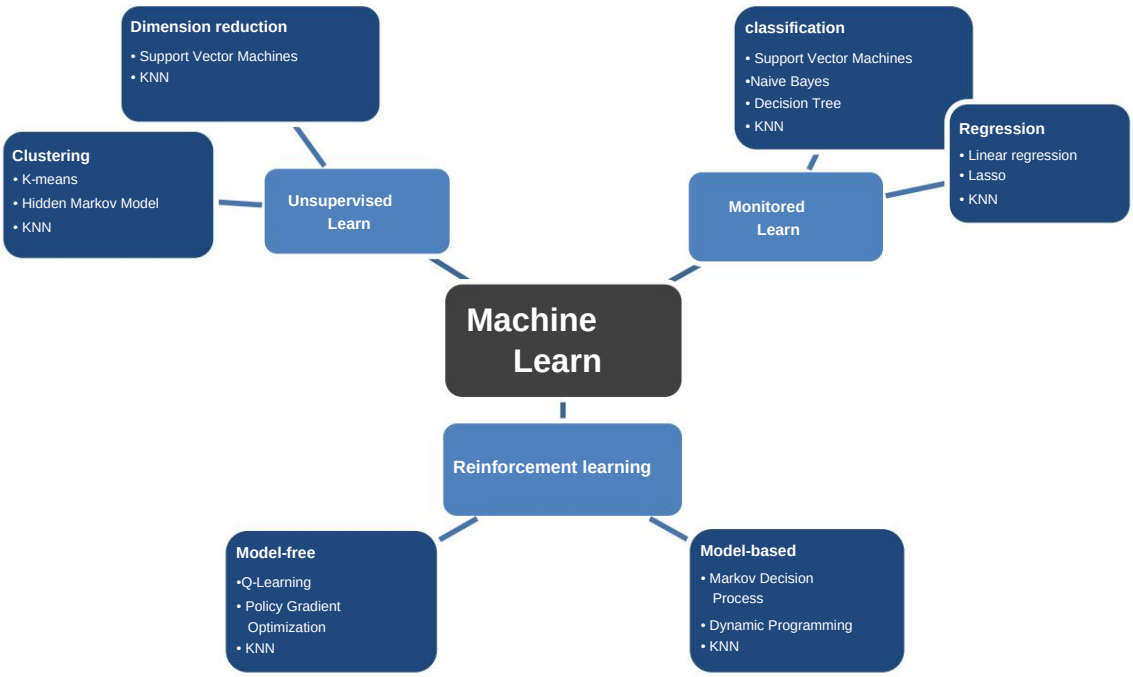


Figure 8: Classes of machine learning with the respective algorithms. Created on Basis of [8, 12, 13].

Supervised learning: The goal of supervised learning is to

that maps certain inputs to corresponding outputs.

labeled data sets are required for which the associated outputs are already known

Before training, the data is divided into a training dataset and a

test data set to later determine the accuracy of the learned function

This learning method is used, among other things, to classify and
Regression. Classification is a problem in which
the output is discrete, so there is only one solution from a known solution space
It is used, for example, in image recognition. Regression is
used when the output should 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, separated by a 28 × 28 pixel
large grayscale image. Each image is assigned the corresponding
This is therefore a classification problem in which a
Function is to be learned that corresponds to the 28 × 28 pixels (784 inputs) with different
Assigns a number between zero and nine to shades of gray. [14]

Unsupervised learning: In contrast to supervised learning,
Unsupervised learning does not require labeled data, which requires additional effort for
experts. In this learning method, the algorithm must
explore independently to recognize inherent connections. [8, 10]

Unsupervised learning is mainly used for “clustering” and “dimensionality reduction”
used. Clustering is the process of grouping data with similar structures and properties
Dimension reduction, on the other hand, tries to reduce the dimensions of the input data
by reducing inputs that have little or no impact on the output
are filtered out. [8, 10, 11]

Reinforcement learning: In reinforcement learning,
a learning agent performs actions and receives feedback from the environment, such as
these actions affect the achievement of the goals. [15]

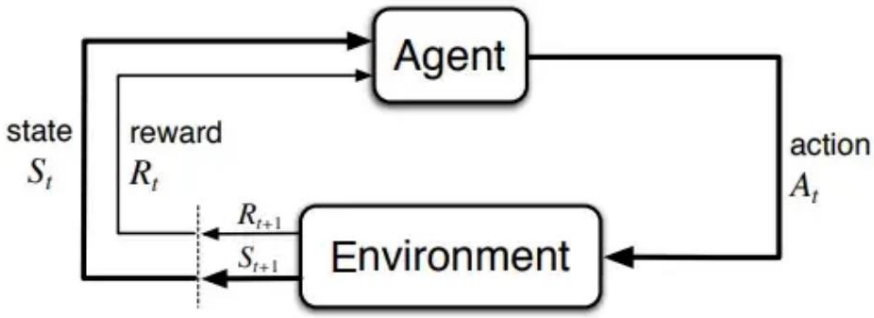


Figure 9: Process of reinforcement learning [16]

The agent is programmed using rewards and punishments, a
to achieve a specific goal without explicitly prescribing how the goal is to be achieved
To do this, the algorithm must use trial and error to find the optimal
Learn actions for the given situations independently. This creates a
Conflict between “exploration” and “exploitation”. To achieve a
To obtain high reward, the agent should exploit the successful strategies.
However, to develop and improve successful strategies, the agent must
try new actions, although the risk of undesirable results
Another challenge in reinforcement learning is
that in most complex systems, actions are not limited to the next
reward, but also on future conditions and thus on future
rewards. An action that at first glance does not seem to be immediately
appears to be expedient, 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 known 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 are used, which, like Darwin's theory of evolution,

work in biology. The second possibility is to use statistical
techniques and dynamic programming to maximize the benefits
different actions in the environment at any time and under the
possible strategies to implement the one that is likely to produce the best future
brings reward. [17, 18]

2.2.3 Artificial neural networks

The most commonly used method in machine learning is artificial
neural network (KNN). The functionality of a KNN is loosely linked to the understanding of the
human brain from the 1960s [19]. The following section
explains how ANNs work and discusses various
Activation functions and the backpropagation algorithm (error feedback)
a.

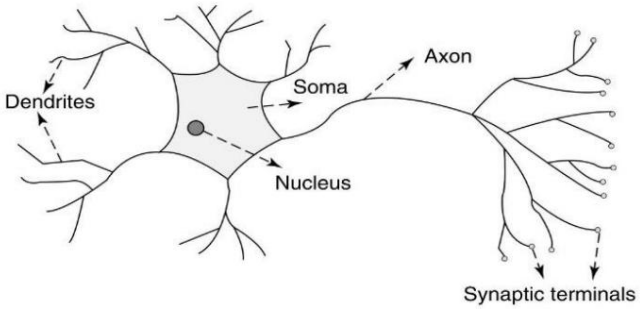


Figure 10: Neuron of a mammal [20]

The human brain consists of over 80 billion interconnected
Neurons, where each neuron is a cell that through biochemical reactions
can receive, process and send information. 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]

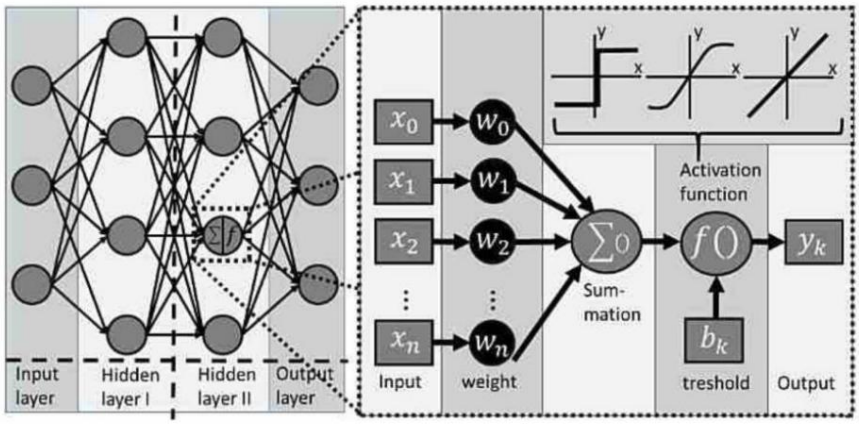


Figure 11: Left: Representation of an artificial neural network. Right: Representation of a single neuron with the weighted inputs and the activation function. [18]

Artificial neural networks consist of neurons arranged in layers and are connected to each other, with each connection having a different weighting. The The first layer of the network is called the input layer and the last layer is called the output layer Each input neuron represents an input parameter and each Output neuron has an output parameter. Input parameters can be numerical 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 determines the the summed and weighted inputs of the neuron and a constant “bias”, whether a neuron is active and how large its output is. The output is then sent to the neurons of the next layer. [18, 21]

As inputs a single neuron from the layer receives the output ÿ1 of all neurons from the previous layer ÿ 1 [22] (see Figure 11 right).

$$= \quad \text{ÿ1} \tag{10}$$

The edition of the neuron in the layer of the KNN is then compared with the Activation function calculated [22]

$$= (\ddot{y} , , \quad \ddot{y}1 \quad + \quad ,) \tag{11}$$

where the sum over the weighted inputs of the neuron in the layer
is formed. $\ddot{y}1$ is the output of the neuron from the previous layer ÿ 1
and the weight of this connection. is the bias. In vector notation:

$$= (\quad \ddot{y}1 \quad +) \tag{12}$$

Depending on the activation function, linearities, nonlinearities or other
Properties are introduced into the model [18]. They are therefore necessary to
to recognize complex, non-linear relationships in the data. The choice
the activation function depends on the problem and has a great influence
on the prediction accuracy of the neural network. The most common
The activation functions used are nonlinear because they are less sensitive
against faulty data are known as linear activation functions.
Examples of nonlinear activation functions are Sigmoid, Tanh and ReLU. [21]
The functions are shown in Figure 12.

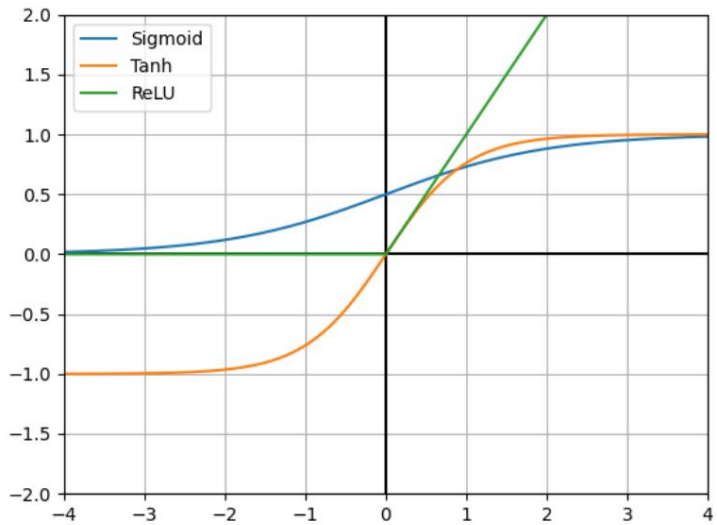


Figure 12: Representation of the Sigmoid, Tanh and ReLU activation function

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 to limit the output of neurons to this area. [21]

$$( ) = \frac{1}{1 + ÿ} \tag{13}$$

Tanh function: The hyperbolic tangent function is similar to the sigmoid function, but is but compared to this point-symmetric to the origin. The values of the tanh-Function are between minus one and one, which means negative signs in the outputs. Like the sigmoid function, the tanh function is continuous and differentiable, which makes backpropagation possible. It is often referred to as Activation function is used when the outputs of the neurons can also be negative should. [21]

$$ÿ ( ) = \frac{2}{1 + ÿ2} ÿ 1 = 2 \qquad (2) ÿ 1 = \tanh( ) \tag{14}$$

ReLU function: The Rectified Linear Unit function is one of the most commonly used activation functions, as they have a biological reference to a real Neuron and is one of the most efficient activation functions [23]. It works so that for negative values the neuron remains inactive and only for positive values becomes active. This function is more efficient than all other activation functions because not all neurons are active at the same time [21]. When using ReLU, However, it can happen that neurons “die” because the gradient for negative values is zero and thus the weights for negative values are not backpropagated. updated, which speeds up the learning process of the artificial neural network [23]. To circumvent this problem, the Leaky-ReLU Function can be used which for negative values instead of zero a small linear component and thus also has a gradient that is not zero for negative values The problem of “dead” neurons does not occur here, but all Neurons active.

Both ReLU and Leaky ReLU are continuous and differentiable for ÿ 0. Since the derivative is not defined for = 0, it is set to either one or zero,
Backpropagation is also possible here. [21, 23]

$$( ) = \{ \quad \begin{matrix} \text{ÿ } 0 \\ 0, \end{matrix} \quad = \max(0,) < 0 \tag{15}$$

$$( ) = \{ \quad \begin{matrix} \text{ÿ } 0 \\ \end{matrix} \quad = \max(,) < 0 \tag{16}$$

Backpropagation: In order for an ANN to learn and its
To improve prediction accuracy, the weights of the individual
Connections between the neurons are adjusted as optimally as possible.
The backpropagation algorithm is used, which calculates a cost or
Error function minimized by adjusting the weights of the ANN in a certain way
The error refers to the deviation of the prediction
of the KNN from the actual value. [22]

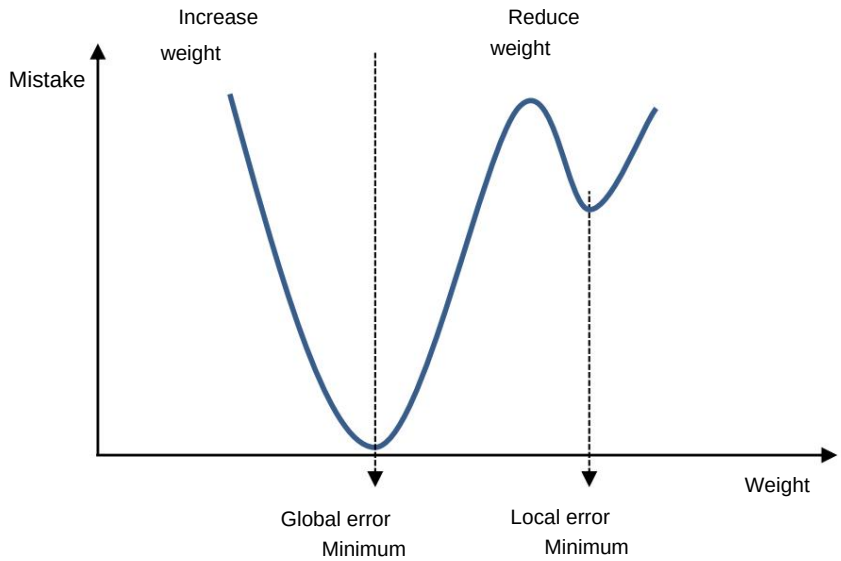


Figure 13: Influence of weighting on the error. Created based on [24]

To adjust the weights, the partial derivative of the error function with respect to
calculated on the weighting of the neural network (—). It indicates how fast

the error changes when the weights are changed. Figure 13 shows a
Example of a function of the error depending on the weighting. The goal
The aim of the backpropagation algorithm is to increase the weights in the direction of descending gradients or in the direction of the error minimum, whereby the
Size of the respective weight change proportional to the influence of the weight on the error. The weights are adjusted iteratively over the training data.
It can happen that the weight moves towards a local minimum
adjusted, meaning that the global optimum is never reached. [22]

2.3 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. In this
Chapter 1 describes the error calculation in classification and regression
explained.

2.3.1 Calculation for classification

In the binary classification (positive or negative) there are four different results
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" and the actual value
is, however, negative.

False negative (FN): The model predicted "negative" and the actual value
is however positive.

Table 5: Possible results of a binary classification

True positive (RP) ÿ Actually: Positive	**False positive (FP)** ÿ Actually: Negative
False negative (FN) ÿ Actually: Positive	**True negative (RN)** ÿ Actually: Negative

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

$$= \frac{\ddot{y}\,\ddot{y} \qquad \ddot{y}}{\ddot{y} \qquad \ddot{y}} = \frac{+}{+ + +} \tag{17}$$

When working with an unbalanced data set, it can happen that
that the overall accuracy is high, but the model is not necessarily in the
Able to distinguish between different cases. For example, out of 100
pictures in which an apple is visible. In 10 pictures an apple is
to see, not on the remaining 90. If every image was classified as “not an apple”, the accuracy would be 90%,
although 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 avoid this problem, precision and hit rate are used. Precision is defined as the proportion of positive
predictions that are actually correct.
were [27].

$$= \frac{}{+} \tag{18}$$

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

$$= \frac{}{+} \quad (19)$$

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

2.3.2 Calculation in regression

For regression problems, various methods for calculating errors are available Which method is ultimately used depends on the problem and is another hyperparameter that is relevant for the present Problem needs to be tested [28].

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

The absolute prediction error at time is defined as

$$= \quad (20)$$

where is the actual value at time and ÿ is the predicted value of the model for the time . Over a certain horizon ÿ, the Determine total deviation using the different methods. [29]

$$= \frac{1}{} \sum_{=1}^{ÿ} 2 \quad (21)$$

$$= \ddot{y} \quad \text{(22)}$$

$$= \frac{1}{\ddot{y}} \sum_{=1}^{\ddot{y}} |\ddot{y} \ddot{y}| \quad \text{(23)}$$

Historically, RMSE and MSE have been used in statistical analysis due to their relevance Modeling popular methods to calculate the error, but meet criticism in the evaluation of 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) und Median Absolute Percentage Error (MdAPE). [28]

$$= 100 \ddot{y} \quad \text{(24)}$$

$$= \frac{1}{\ddot{y}} \sum_{=1}^{\ddot{y}} \ddot{y} \ddot{y}| | \quad \text{(25)}$$

$$= (| |) \quad \text{(26)}$$

However, they are unsuitable for data with small values, especially if the Data set = 0, because in this case the percentage error is undefined. For Values close to zero cause large fluctuations in the error calculation. Therefore, MAPE in these cases may be significantly larger than MdAPE, since the median is generally is 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 a Asymmetry in the calculation. [28]

2.4 Predictive modelling over a sliding time window

Predictive modeling is a form of machine learning whose goal is to
to predict the future state of a system based on known historical data [30]. Typically,
statistical models are used to
However, difficulties in modelling high-dimensional, nonlinear
systems [31]. For modeling and predicting complex nonlinear
Systems are suitable for ANNs that can only describe the inherent system properties with the help of
historical data. For modelling time series, this
Work uses a sliding time window. 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 series into a classical problem of supervised
This makes it possible to create an artificial neural network using the
Backpropagation algorithm. For each time window, the last
States are used to predict one or more future states.
Different methods for forecasting over multiple time steps are presented in Chapter
2.4.1 presented. [32]

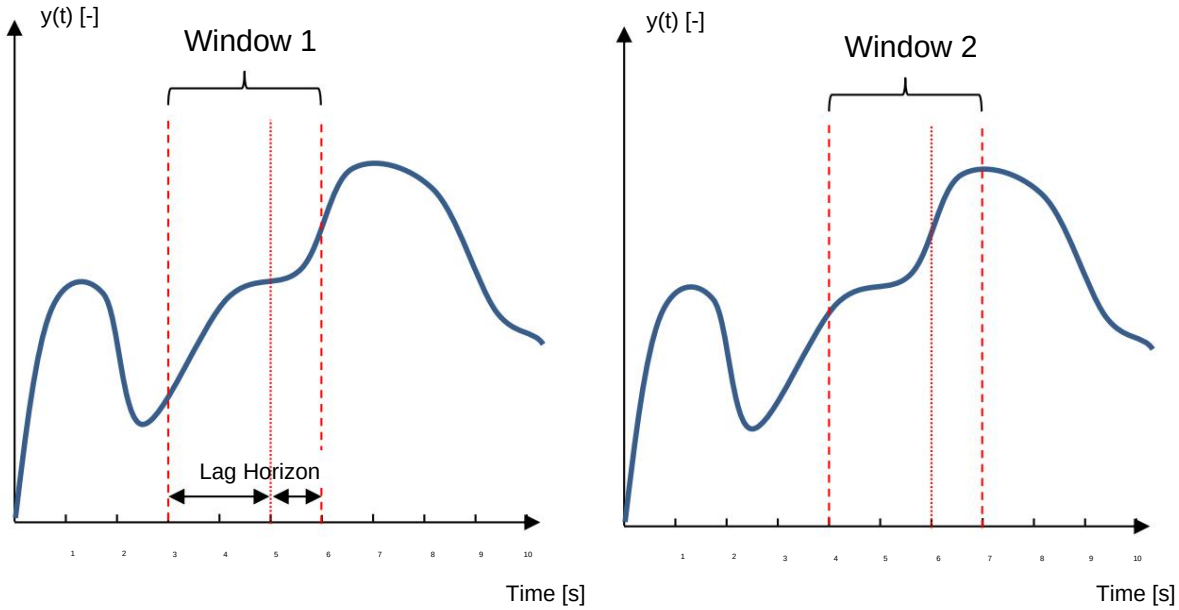


Figure 14: Example of a sliding time window

Figure 14 shows an example of a sliding time window. The delay (lag) refers to the period of the past that is taken into account for the prediction [30]. The horizon ÿ indicates how many future states are predicted and in this example is equal to one [30]. The first time window is between $_{3}$ and $_{6,}$ where the values for $_{3, 4}$ and $_{5}$ the value at the time $_{6}$ should be predicted. For the following steps, the time window is increased 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 slot then represents an entry in the training data set. In continuous systems it is possible to to further train the model with the new data and thus improve the accuracy of the model over time. To predict unknown future States can then be trained using the last states as input, be used.

2.4.1 Methods for multi-time-step prediction

A prediction over several time steps has the task of predicting the next ÿ values
(+1,…, +ÿ) using the historical time series of the last values (ÿ +1,…,)
where ÿ > 1 is the forecast horizon. The problem here is that
each value has a temporal dependence on the previous values.
For example, the prediction of the value $_{+2}$ also the value $_{+1}$ from the
is not known but must also be predicted. In this chapter
Three different methods are presented to solve this problem. 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 only
can make a prediction about the next time step [33].

$$_{+1} = (ÿ +1, … ,) + _{+1} \quad (27)$$

The prediction of the next ÿ values is then done recursively, with the predicted
Value each with as input parameter for the prediction of the following values
This is repeated until the entire horizon ÿ
predicted. A disadvantage of this method is the sensitivity to
forecast errors, as these are passed on to the following forecasts and
thus becoming larger and larger above ÿ. [33]

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

$$_{+} = (ÿ +1, … ,) + _{+} \quad (28)$$

In this method, the predicted values are not used to predict new
values are used so that no error accumulation occurs. A disadvantage of this
However, the method is that the complex relationships between the predicted
Values (ÿ $_{+1, ...,}$ ÿ +ÿ) cannot be modelled because these values are

different models. In addition, this method is significantly
less efficient and more computationally intensive than other methods, since ÿ different models need to be trained. [33]

MIMO (Multiple Input Multiple Output) method: The MIMO method enables
In contrast to the recursive and direct method, the output of multiple values
at once, which means that even complex relationships can be
between the future values can be modelled [33]. A single
Model trained that has ÿ outputs.

$$(\quad_{+1, \dots, +ÿ} \quad) = (ÿ , \dots,) + \tag{29}$$

with the vector function ÿ ÿ $^{+1}$ ÿ ÿÿ and the noise vector ÿ ÿÿ . A disadvantage
This method is that the horizon ÿ cannot be changed afterwards
can be done without training a completely new model. [33]

2.5 Model Predictive Control (MPC)

MPC is a modern control method that uses predictive modeling to
to regulate a system or process as optimally as possible. The aim is to
Reaction of a system to different control variables over a certain
time horizon in real time and to find the best control strategy while minimizing
a cost function and certain constraints. A
The control strategy is a combination of control variables at discrete points in time
over a certain control horizon (, +1, … , +). MPC enables the
autonomous control of complex processes without expert intervention via a
longer period of time and is flexible, which allows application to different systems with
different properties allowed. [3, 34]

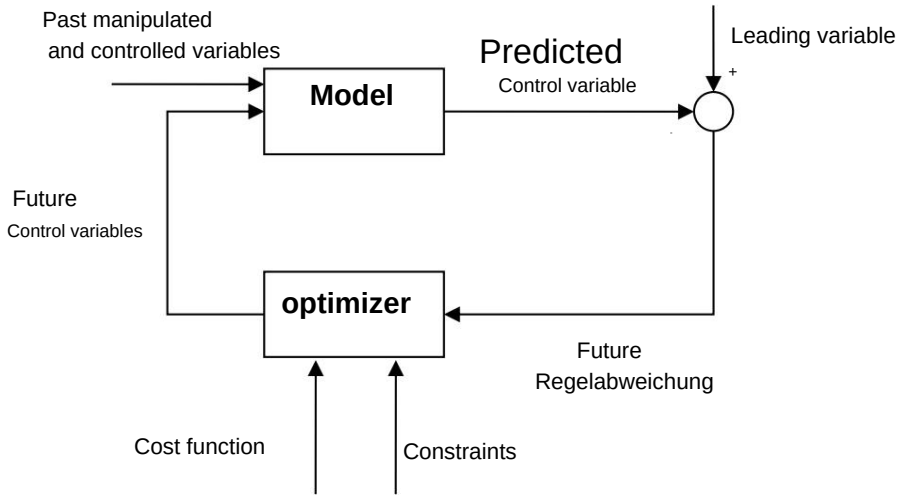


Figure 15: Basic structure of MPC. Created based on [3].

MPC requires a suitable process model to calculate the performance based on past values Control variables and possible control strategies determine the future behavior of process. The effectiveness of the control therefore depends heavily on the quality of this model. With the future control deviation **(30)** calculated the optimizer taking into account the cost function and certain Constraints determine the optimal control strategy. [3]

2.5.1 Cost function and constraints

The possible control strategies are defined with a cost function and, if necessary, with additional constraints. The general objective is that the control strategy should be selected in which the predicted values ÿ optimally approach the reference value. To calculate the total costs the future control deviations ÿ( ) are calculated at any time via the Forecast horizon ÿ is required (see formula (1)). [3]

$$ÿ( ) = ( ) ÿ ÿ( ) \quad (30)$$

$$(\ddot{y},) = \ddot{y} ( )[\ddot{y}(+)] \quad ^2 + \ddot{y} ( )[\ddot{y} (+ \ddot{y} 1)] \quad ^2 \qquad (31)$$

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

$$( ) = \quad \ddot{y}\ddot{y} \text{ mit } \ddot{y} [1, \dots , \ddot{y}] \qquad (32)$$

If 0 < < 1, predicted errors further in the future will be higher
weighted than the first, resulting in a gentler control with less effort
If > 1, the first errors are weighted higher, which leads to a stricter
regulation. [3]

The second term of (ÿ,) refers to the control effort that occurs when the manipulated variable changes, this could include, for example, energy costs,
which should be kept as minimal as possible. If the standard expenditure for the
If the problem is not important, the second term can be neglected.
[3]

In addition to the cost function, certain constraints are also taken into account in practice taken into account. For example, the actor has only a limited range of action and a maximum rate of increase that can be achieved with the different control strategies must be taken into account. For example, a valve is fully open or closed by its position, the opening and closing speed
Side conditions may also be imposed by safety or environmental regulations
where process parameters such as temperature or pressure are limited
In most cases, a lower and upper limit of **(33),**
a limitation of the rate of increase of **(34)** and a limitation of **(35)**
taken into account. [3]

ÿ ( ) ÿ (33)

ÿ ÿ ( ) ÿ (ÿ 1) ÿ ÿ (34)

ÿ ( ) ÿ (35)

2.5.2 Optimal Control Strategy

The optimal control strategy is the strategy with the lowest total cost, which all constraints are also fulfilled. To identify these, all constraints must first be possible control strategies are formed. For this purpose, all possible Combinations of control variables at discrete points in time via a Control horizon (, +1, … , +) is formed. For each control strategy, Using the model, the future values of the controlled variable over the forecast horizon predicted. These predictions are then used to determine the associated costs and from all strategies the one with the lowest costs is selected and additional constraints are met. From the optimal control strategy to At this point in time, only the first control signal is applied. The calculation of the optimal control strategy is repeated for each time step. [34]

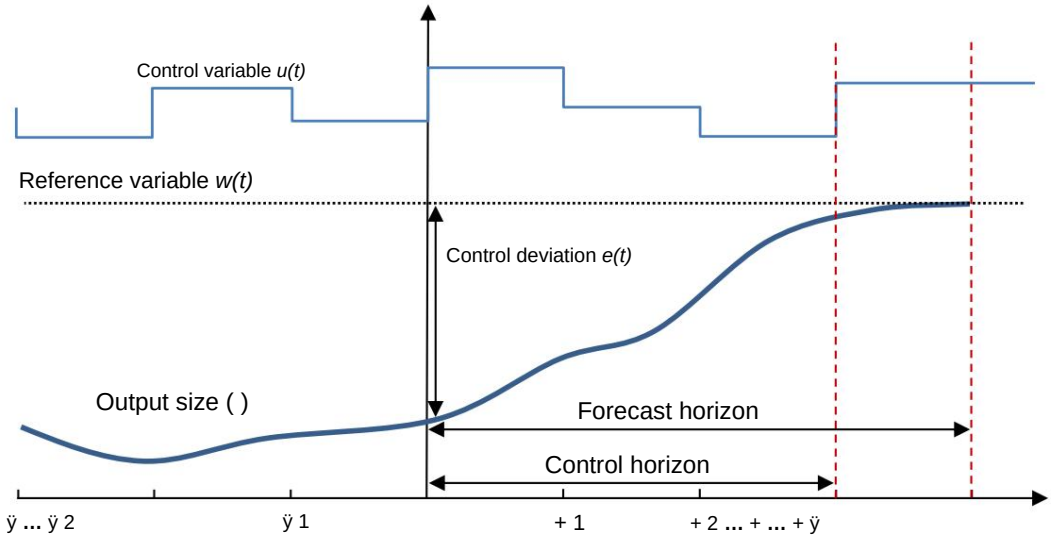


Figure 16: Prediction of a control strategy over a sliding time window at MPC

Figure 16 shows an example of the prediction of a control strategy. A sliding time window is used, only that here in addition to the system state the Control variable is used for modelling. In addition to the actual Forecast horizon ÿ introduces a control horizon over which all possible Control strategies are formed. The strategies are then evaluated via the actual forecast horizon. Since the system responds to a control variable with a time delay reacts, the forecast horizon should be longer than the control horizon in order to time dependencies must be taken into account. In addition, the control horizon chosen smaller to reduce the computational effort.

3 Implementation

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

The goal of control AI is to use MPC to adapt the system's controlled variable to a to approximate a given reference value. The predictive model used in this work is A KNN is used, which is supposed to learn the relationship between past manipulated and controlled variables through a sliding time window and to make predictions about future system states. During the virtual process, the KNN is continuously trained with new data to improve prediction accuracy and thus to improve the effectiveness of the regulation over time. To this end, the first Version of the AI framework "HyREN" (Hybrid Regression Evolutionary Network), which uses a genetic algorithm to train of an ANN [18]. However, this method requires more computing power than classical supervised learning with the backpropagation algorithm, whereby the entire training process takes significantly longer. When simulating a virtual system, a longer training period is not necessarily a disadvantage, since the In this case, the process does not continue until the training of the rule AI is completed. When transferring the rule AI to a real process, however, It must be taken into account that the process continues during training. It is crucial to keep the training time of the rule AI as short as possible to enable real-time control. It was found that with the AI framework HyREN, a training iteration takes several minutes. In In comparison, supervised learning with the backpropagation algorithm requires 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 provides the interface between the rule AI and the virtual system is executed in the file “main.py”. In this file, both the rule AI and the virtual system are configured. After each time step, the current system state of “main.py” is passed to the rule AI. In regular At regular intervals, the control AI uses MPC to calculate the optimal control strategy and passes this to “main.py”. There the first control variable of the optimal Control strategy applied to the virtual system. The virtual system calculates then the next system state depending on the applied control variable and the disturbance variable and returns it to “main.py”. The control AI is also trained at regular intervals with the new data. The cycle described is repeated until a predetermined step limit is reached. The sequence of Algorithm is shown in Figure 17.

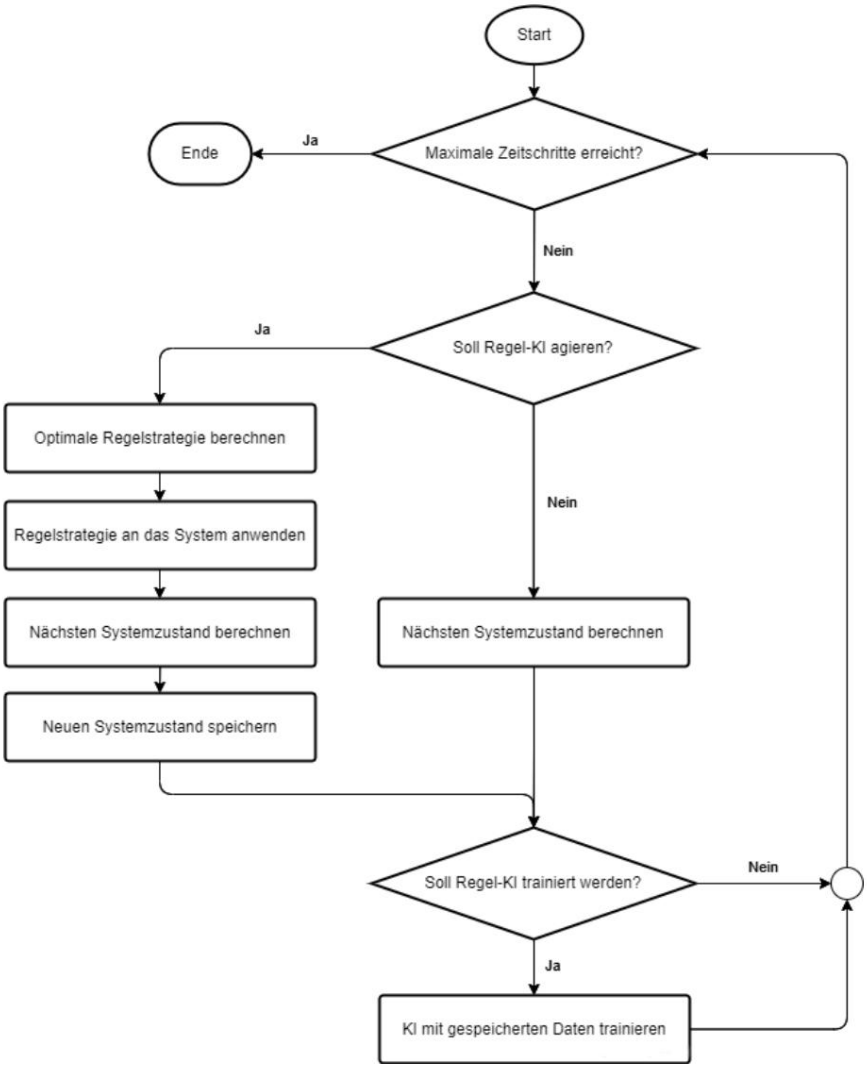


Figure 17: Main program flow

3.1 Transforming a time series into supervised learning

In order to train the rule AI, the known time series must first be
classical supervised learning. For this purpose, the method of
sliding time window, where 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 shown in Table 6. [35]

Table 6: Example of a univariate time series

Time	Value
1	64
2	21
3	45
4	89

Table 7: Example of converting a univariate time series into supervised learning

X	and
?	64
64	21
21	45
45	89
89	?

The conversion to supervised learning is shown in Table 7. X are the values, which are used as inputs for the KNN and y are the values which are used as outputs for the KNN. The first and last rows cannot be used because one value is missing. During the conversion, the column with the values copied and moved by one line. [35]

In Python, this transformation can be applied to a DataFrame using the function ".shift(x)", which shifts all columns of the DataFrame by x rows [36].

For a prediction over several time steps, the time series for each step must copied and moved accordingly. The transformation into training data is done in this program with the function "convert_input_data_training()" from "utils.py". The function allows to count the number of past values the

control horizon and forecast horizon ÿ to achieve different

To convert the data into training data, a time window is created for

Time window time steps required, with

$$= + ÿ + 1 \quad (36)$$

Table 8: Multivariate time series with manipulated and controlled variables

Time	Control variable	Control variable
0	0	1
1	1	1
	...	
	...	
ÿ 1	ÿ1	ÿ1
+ 1	+1	+1
	...	
+	+	+
	...	
+ ÿ	+ÿ	+ÿ
	...	

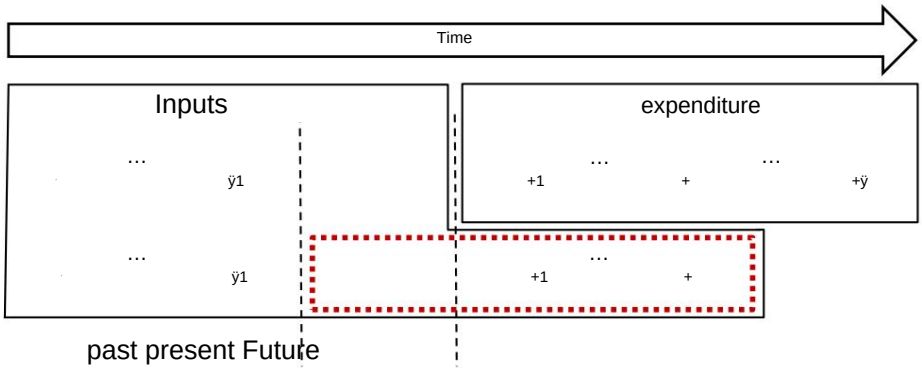


Figure 18: Structure of a time window at time t with manipulated and controlled variables

For a prediction of the future controlled variable (ÿ +1, ÿ +2, … , ÿ +ÿ) are both the controlled variable as well as the manipulated variable of the past points in time are required. a multivariate time series. Table 8 shows the recorded time series of the System in which the manipulated variable and the controlled variable are recorded for each time step was. The last recorded time is. So that at least one valid Time window for training exists, must be ÿ, otherwise not enough values for a time window are available. For any point in time ÿ [, ÿ ÿ] with ÿ can a time window can be created that uses past points in time to ÿ predict future to predict the time at which all values are known. The conversion of the Time series in Table 8 in a time window at time , with the respective inputs and outputs for the KNN is shown in Figure 18. The inputs are the last values of the manipulated and controlled variables, the current manipulated and controlled variables for time and the next control variables are used. The red dotted box represents a matrix of all possible control strategies that are only used for the prediction of = is used. It does not play a role when creating training data, since the Values (, +1, … , +) are known in this case.

3.2 Regel-KI

The main component of the rule AI is located in the class “Agent” in the file “control_ai.py”. This class stores the KNN, which is used as a predictive model and is created with the static function “build_network()”. The KNN consists of an input layer, an output layer and two hidden Layers, the number of neurons of which is determined by the input parameters of the function can be customized.

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

To store the time series, the data type “deque” from the module “collections” is used. This data type is similar to a list, but it can be a maximum size can be defined [38]. If the maximum size is defined according to the Inserting a new element, the oldest element is automatically deleted [38]. This property is advantageous in this context, since a continuous process, the amount of data over a longer period of time is can be high. Using the data type “deque” avoids After a longer period of time, the AI rule runs and the storage space is no longer sufficient.

In this work, the MIMO method described in Chapter 2.4.1 is used, to make predictions over several time steps, since this method is high number of predictions. In comparison, the first step was to recursive method is used, but with a high number of predictions requires significantly more time, since each time step of each control strategy is individually must be predicted. For real-time control, the recursive strategy therefore not suitable. When using the MIMO method, the KNN has so many Output neurons, such as time steps must be predicted. The structure of the

Input and output layer of a KNN that calculates the next ÿ points in time under
Using the transformed time window from Figure 18, is in
Figure 19. The number of input neurons is calculated using **(37) .**

$$= (+ 1) ÿ 2 + \quad (37)$$

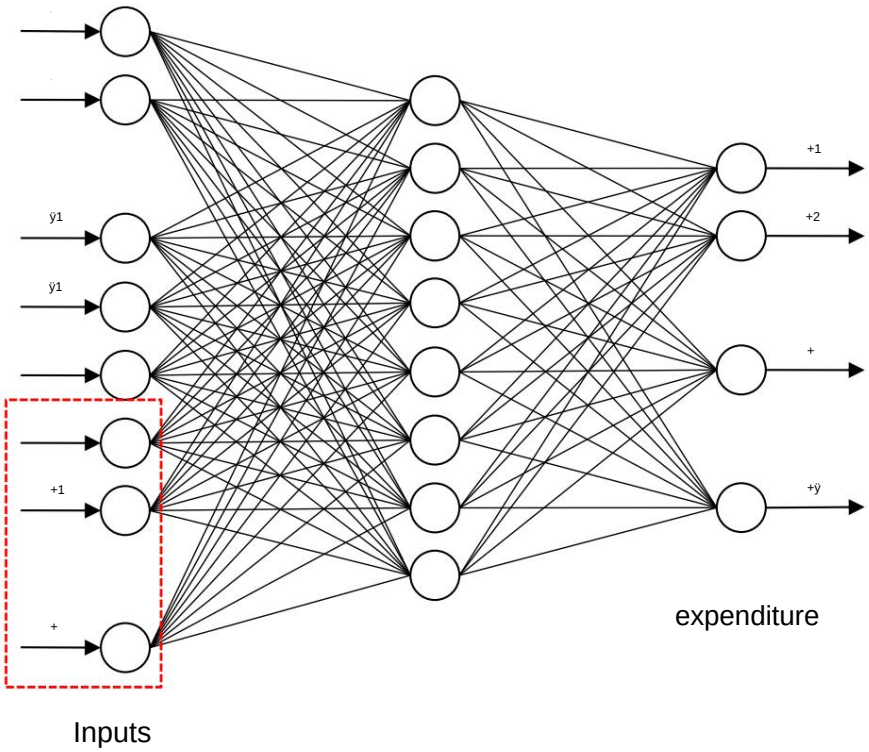


Figure 19: Structure of the input and output layer of the rule AI

The red box in this figure also represents the matrix of possible
Control strategies. For the prediction, all input values above the box
known, since these are past values for the manipulated and controlled variables. How
As shown in Figure 15, the model predicts with the past values for positioning and
controlled variable and the future manipulated variables from each possible control strategy.
Predicts the course of the future controlled variable.

At the beginning of the process control, the control AI is in an exploration phase because it has no data available for training. Similar to Reinforcement Learning is a conflict between exploration and exploitation (see chapter 2.2.2). In the initial phase, the rule AI should have as much experience as possible to gain a solid understanding of the process. In the course of the process, the prediction accuracy of the KNN used improves with increasing amount of data, so that the rule AI increasingly finds the best possible control strategy should be exploited.

To balance exploration and exploitation, the Epsilon-Greedy algorithm The value of epsilon is a number between zero and one, which represents the probability of exploration by a random action. To To determine whether the rule AI should explore or exploit, a random number is between zero and one. If <, a random action is performed, otherwise the calculated optimal control strategy is executed. A value of

= 1 means that 100% of the time a random action is performed, while

= 0 means that the calculated optimal control strategy is executed 100%.

At the beginning of the process, = 1 to explore the system. In the course of the Process should be continuously reduced to reduce the probability of random actions. To do this, after each training iteration of the KNN, the value for with a constant ÿ (0,1), resulting in a reduction of over the time. In addition, a lower bound for be set so that the probability of exploration never becomes zero. [39]

The rule AI acts in fixed time intervals of milliseconds and is measured in fixed Time intervals of milliseconds. In a virtual system, and be a multiple of the time step size ÿ of the simulation in order to to ensure predictable and consistent behavior of the rule AI. The size of the time window can be calculated using equation **(38)** and the amount of new data per Trainingsiteration can be calculated using equation **(39) .**

$$= (+ ÿ) ÿ \quad (38)$$

$$= \frac{\qquad}{\qquad} \quad (39)$$

3.3 Calculation of the optimal control strategy

To determine the optimal control strategy, it is first necessary to examine the course the future controlled variable depending on all possible control strategies To do this, first a matrix created that contain all possible control strategies over a certain control horizon. To calculate the Matrix, the Cartesian product is used, which calculates all combinations of two or more quantities can be calculated. Assuming that the quantities = { , , } and = {1, 2}, the Cartesian product of A and B gives the Set {(, 1), (, 2), (, 1), (, 2), (, 1), (, 2)}. [40]

With the set of possible discrete values for the manipulated variable and the Control horizon allows you to calculate all possible control strategies with

$$= \quad ^{+1} \quad (40)$$

The result is a | $^{+1}$ × (+ 1) matrix. For = 2 we get = × × . It is It should be noted that the control horizon in this work only includes future control variables (+1, … , +) is taken into account. The current control variable is always considered. For the control horizon = 0, only the current control variable is used to predict The number of possible control strategies increases with increasing exponentially. For example, if there are ten possible values for the manipulated variable, At = 4 there are 100,000 different control strategies.

To list all possible control strategies, the possible discrete Values for the control variable are passed to the control AI in the form of a list at start-up Then the function “compute_all_possible_strategies()” from the file “utils.py” all possible rule strategies in a “numpy.array”

The function receives as parameters the control horizon and a list of
Values for the manipulated variable.

Data for the prediction are entered into the
The function receives as a parameter the current value of the
Control variable, since it is not yet available in the memory. In this function
First, the data from the last time and the current value of the
Control variable converted into a “numpy.array” (ÿ , ÿ , … , ÿ1, ÿ1,). The line
The list is then copied as often as there are rule strategies, so that the
generated array can be merged with the array of control strategies, so
that each line has the input structure shown in Figure 18. Finally, the
predicted control variables (ÿ +1, … , ÿ +ÿ) is evaluated with the cost function **(31) .** In
of this work, only the constraint **(34)** is considered, with ÿ = 0. In the
Control AI can define a “du_max” to compensate for 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
not too large. In most cases, the control strategy with the smallest
costs are applied. To avoid the rule AI falling into a local minimum,
Sometimes a random rule strategy is chosen from the best ones.

3.4 Virtual system

This chapter implements the virtual proxy system, which serves to
to test the effectiveness of the rule AI. Since the rule AI will be used in a later work on
a bioreactor for pH control, a
A proxy system is used to roughly model the pH value in a bioreactor. This allows the
transferability of the control AI to a real system
be efficiently tested in advance.

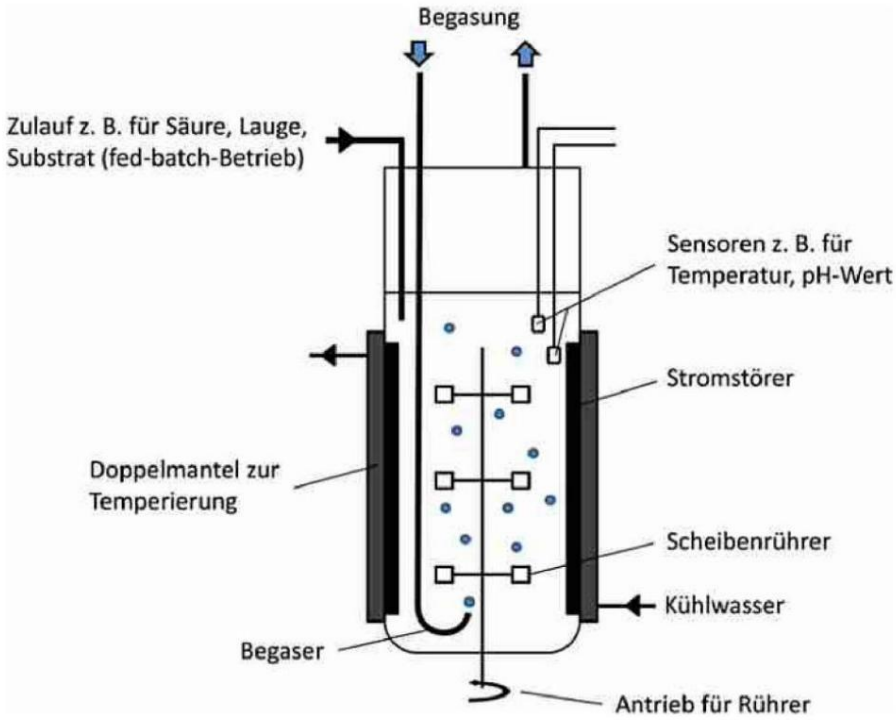


Figure 20: Schematic representation of a bioreactor in the form of a stirred tank [41]

The aim of future work will be to investigate the effectiveness of the control AI in the pH Value control in a bioreactor. In this case, the control variable is represents the valve position that influences the flow rate of an acid or alkali and thus changes the pH value. The controlled variable here is the pH value, which is sensors in the boiler. In this work, a proxy system is used, which is intended to approximately model a pH value using equation **(41) .** Proxy system enables rapid implementation and testing of various regulatory procedures.

$$\ddot{y}(\,) + {}_{1}(\,)\,|\,(\,)\,\ddot{y}\,|\, + {}_{2}(\,) = (\,) + (\,) \qquad (41)$$

Here ( ) is the controlled variable of the system, i.e. the pH value, which is determined by the manipulated variable ( ) is influenced. 1( ) and 2( ) are internal state variables of the system, which can additionally influence the controlled variable and change over time. describes the most stable value of the system at which $_{1}$ has no influence and the

Slope of is minimal. ( ) describes the external disturbances and is divided into into a random noise a possible interference pulse and a continuous, dynamic disturbance behavior .

$$( ) = ( ) + ( ) + ( ) \quad (42)$$

The random noise is generated by a random number using the “numpy” function “random.normal()” is generated and is intended, among other things, to model the measurement uncertainty. The random number is normally distributed with the expected value = 0 and a specified standard deviation . The value for the standard deviation can 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 lead to at any time to test the stability of the control system. The dynamic disturbance ( ) can be calculated at the beginning of the simulation as a function of time be specified.

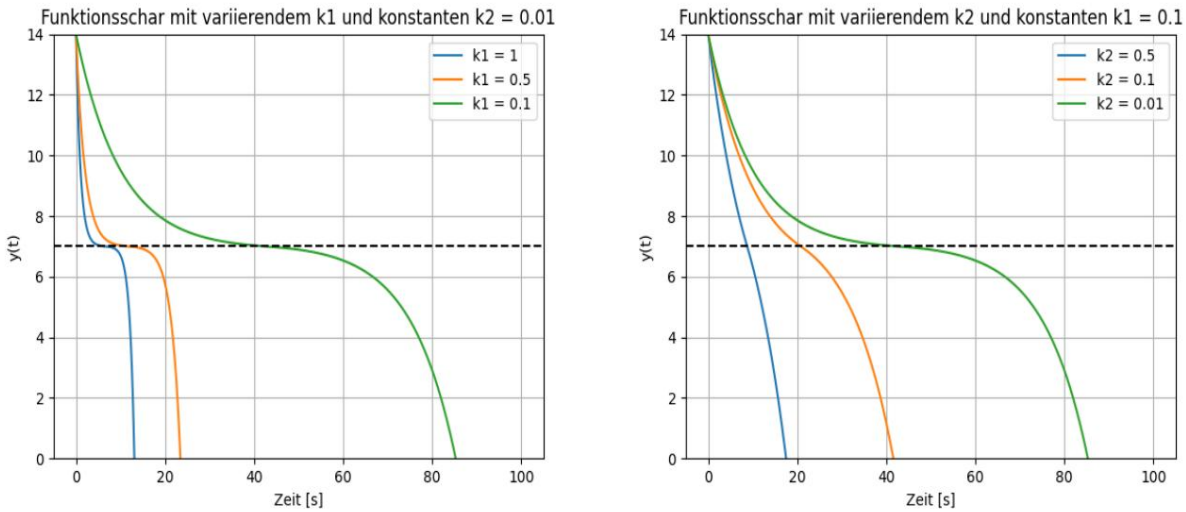


Figure 21: Influence of k1 and k2 on the system with u(t) = 0 and z(t) = 0

The system is limited by ÿ [0,14] to approximately model a pH value. Figure 21 shows the influence of 1 on the system with = 7. and 2

It can be seen that 1 the slope depending on the distance to the value

The larger 1 is, the faster the system becomes unstable if it deviates slightly from . The value of $_2$ describes a constant decrease in System and affects how stable the system is in the range of. The values of $_1$ and $_2$ contain all physical properties that influence the pH value. For $_1$ This can, among other things, affect the reaction kinetics and $_2$ for example the diffusion rate. Other physical properties such as temperature and Pressure can $_1$ and $_2$ The summary in $_1$ and $_2$ serves the Simplifying the proxy system.

The virtual system is simulated in the file “simulation.py”. This file contains the Class “Simulation”, in which the virtual system is simulated. The simulation can be the class “SimConfig”, in which the system parameters are set
The function “step()” changes the state of the system to the next
Time step depending on the manipulated variable and the disturbance pulse. The differential equation **(41)** is used to calculate the change ÿ with the function “_calculate_change()”, which is then added to the controlled variable.
The old system state is then saved in a “dict” object and
a list is inserted and the time is increased by ÿ. At the end it is ensured that that is within the specified limits. The function returns the stored system state and the newly calculated value for .

3.5 Data analysis and visualization

To evaluate the performance of the rule AI, for each rule strategy the actual values that the system would assume with the differential equation **(41)** and compared with the predicted values. The actual Values (+1, … , +ÿ) for each control strategy can be set using the function “calculate_strategies()” from “simulation.py”. The function “odeint()” from the module “scipy.integrate” is used to calculate the differential equation.
It should be noted that the calculation of the actual values is only for

Checking the accuracy of the prediction and not for training the
Rule AI can be used.

In the class “Data” from the file “export.py” the error is reported using the methods MSE, MAPE and MAE are calculated (section 2.3.2). The total error over all
control strategies and points in time. In addition, the total error over all
Control strategies at each individual point in time (+ 1, … , + ÿ). The
Results are visualized graphically using the functions of “export.py” and are stored as “.png” files are stored in a folder. The configuration settings of each
Simulations are stored in the “data” folder of the main project.
The graphs for visualizing the prediction accuracy of the KNN are saved at regular intervals in the “Graphs” folder. At the end of the simulation, the course
the simulation and the temporal progression of the error over time in the folder “summary” saved.

4 Results and discussion

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

Subsequently, the control success in a dynamic system is examined, by both the reference variable and the state variables 1 and $_2$ over time Furthermore, the accuracy of the predictive model is examined, by comparing the predicted values with the actual values taking into account all control strategies are compared.

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

4.1 Problem with the ReLU activation function

A problem that arose during the evaluation was that often one or more Output neurons of the KNN with the ReLU activation function during remained zero throughout the entire term. This is probably due to the fact that in Chapter 2.2.3 in section This is due to the problem of dead neurons mentioned in the ReLU function.

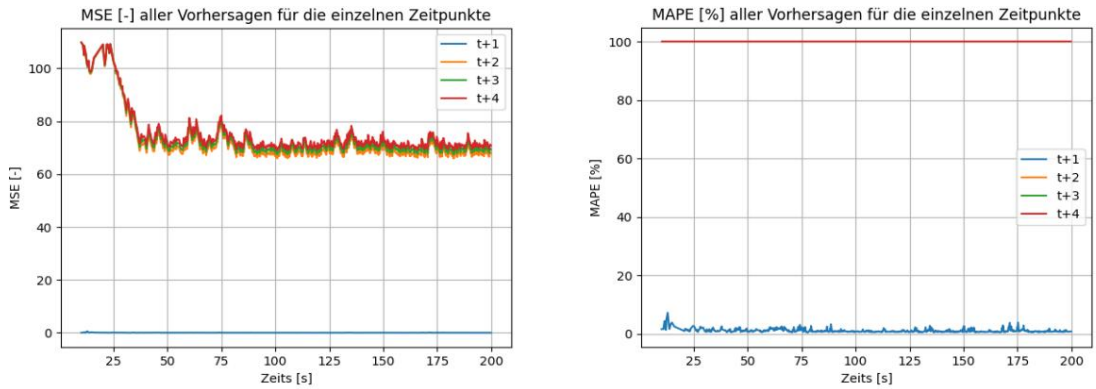


Figure 22: MSE and MAPE for the individual time points, where three output neurons are permanently zero

Figure 22 shows the MSE and MAPE of the individual time points using the ReLU activation function. It can be seen that only the prediction for + 1 is accurate. The remaining three time points have an MSE of about 70 and a MAPE of 100%. This is due to the fact that the output neurons that control the Make predictions for the second, third and fourth time steps, permanently are turned off.

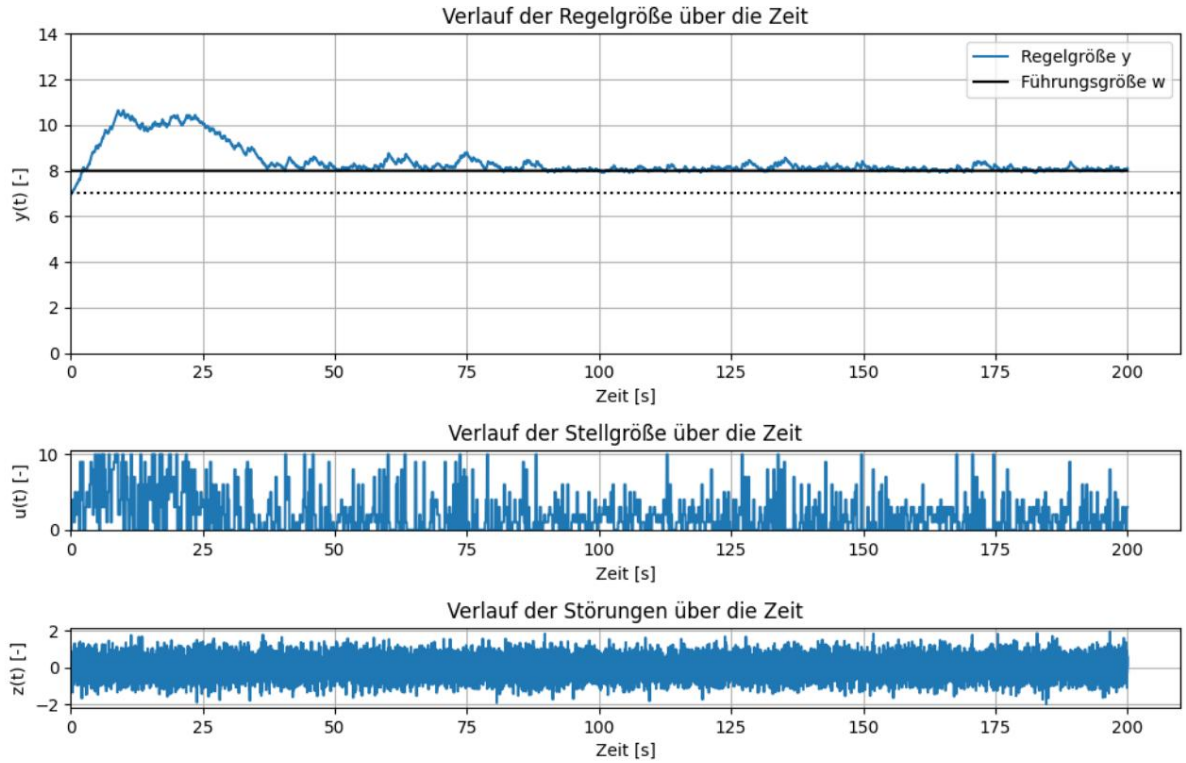


Figure 23: Simulation process for a rule AI in which three output neurons are permanently are zero

Despite this incorrect prediction, Figure 23 shows that the control is successful
This is because the costs for the incorrectly predicted
Time steps remain constant, since neither the output nor the weighting of the
time steps. If the cost of a time step for all predictions
remain the same, they can practically be neglected, since they have no influence on
the choice of the cost-minimizing strategy. A successful regulation is
theoretically possible, as long as at least one output neuron makes correct predictions
However, to avoid this problem, instead of the
ReLU function for each layer of the KNN the Leaky ReLU activation function with
a slope of 0.1 in the negative range. This fixes the problem and
leads to better predictions.

4.2 Control of a static system

In this chapter, the effectiveness of the rule AI in the virtual system with constant state variables 1 and $_2$ and a constant control variable The influence of window size, process parameters and the disturbances are examined for the control success. The simulation time is set to 200 seconds. The system parameters are set as follows: = 7, $_1$ = 0,2, $_2$ = 0.05, = 0.15 and = {0.1, … ,10}. The ÿ of the simulation is set to 10ms. The weighting for the cost function **(32)** is calculated with = 0.7, later values are therefore weighted higher than earlier values. The constraint **(34)** is initially not taken into account, which causes the manipulated variable to fluctuate greatly The simulation starts at (= 0) = 7. For the disturbance behavior, initially only the random noise with = 0.5 is taken into account. The influence of and is examined in Chapter 4.2.3. 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 control AI enough time to determine the system behavior learn.

4.2.1 Influence of window size on control success

The following section examines how a variation of , ÿ, and on the control and prediction accuracy of the control AI. The The amount of training data per training iteration depends on , ÿ, and The larger the time window, the less training data is available. If the time window is chosen too small, complex systems may experience that temporal connections are not recognized. A time window must be selected large enough to reflect the temporal relationships without the amount of training data becomes too small.

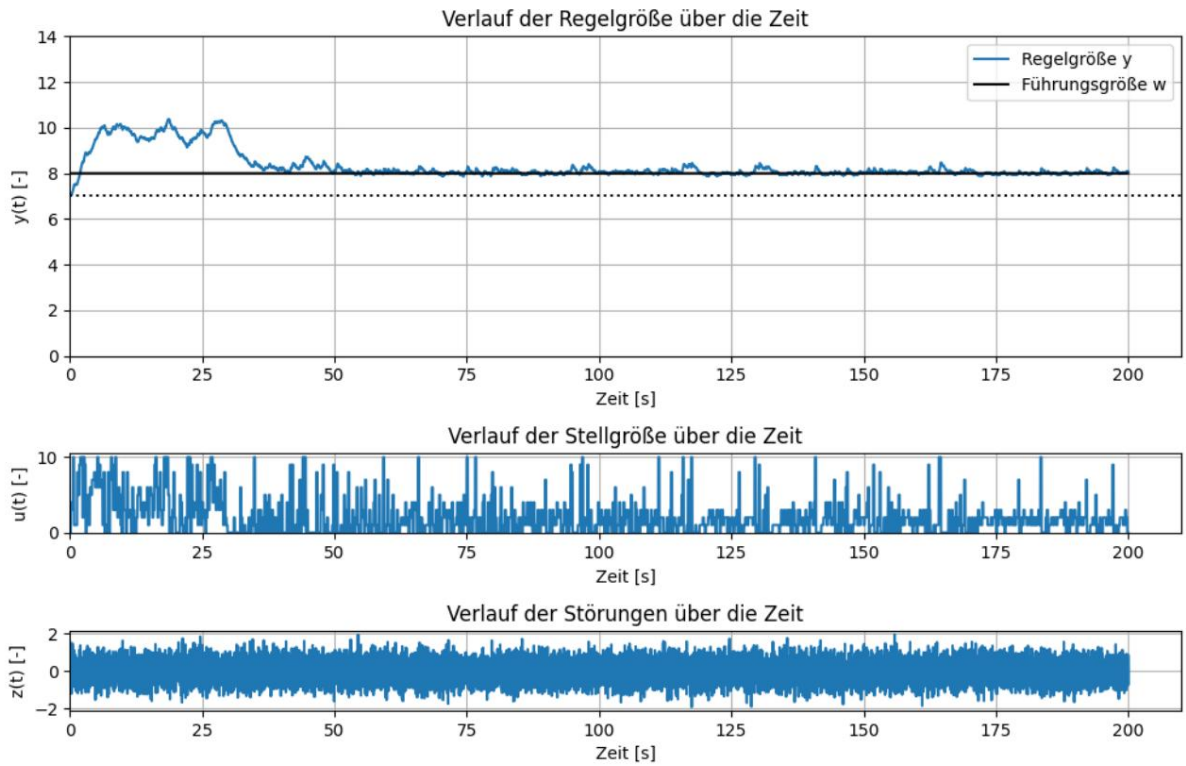


Figure 24: Simulation course for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s

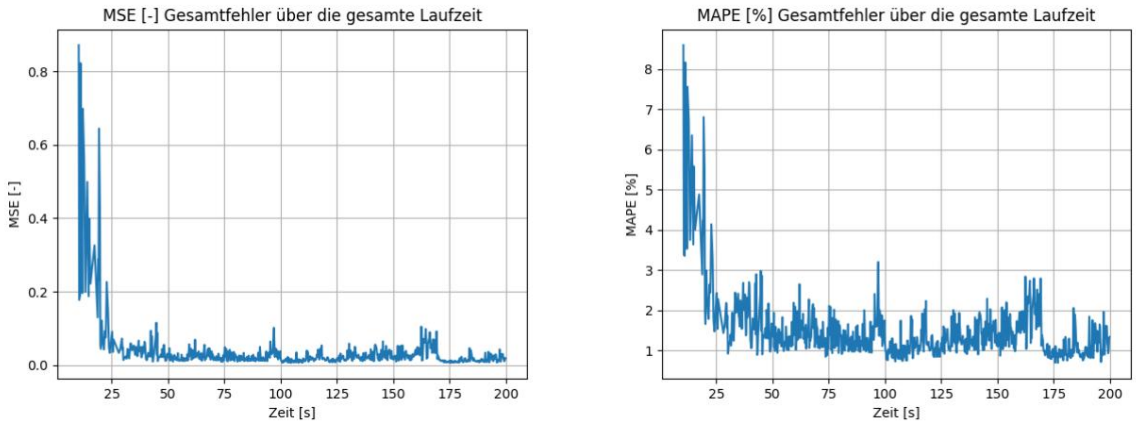


Figure 25: MSE and MAPE for n = 4, m = 2, h = 4, tact = 200ms, ttrain = 10s

First, the rule AI is simulated with a symmetric time window with = 4, = 2, ÿ = 4,

= 200 and = 10. The choice of these parameters leads to 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 shown in Figure 24. It

it can be seen that the control AI reaches the reference value after about 50 seconds and holds over the entire period. Figure 25 shows that after a few predictions the total error (MSE and MAPE) is below 0.1 or 3% respectively. This is due to the fact that the virtual system with constant state variables is not complex and the control AI therefore the temporal relationships quickly. In more complex systems, the rule AI would probably need longer to learn the temporal relationships. The reason why the control AI only reaches the reference value after 50 seconds, although the The fact that the prediction error is small after just a few seconds is due to the Epsilon-Greedy Algorithm that frequently generates random values in the range of 0 seconds to 50 seconds. Actions selected. With = 0.7 is the probability of a random Action at = 25 0.7 ^{2} = 49%, because the AI has 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 rule AI is slowly becoming Furthermore, the course of the manipulated variable shows that there are small fluctuations after the reference value is reached. This is due to In most cases, this is because in this simulation, with a probability of 10% one of the 10 best control strategies is selected and that the Constraint **(34),** which determines the maximum rate of increase of the manipulated variable limited, is not taken into account.

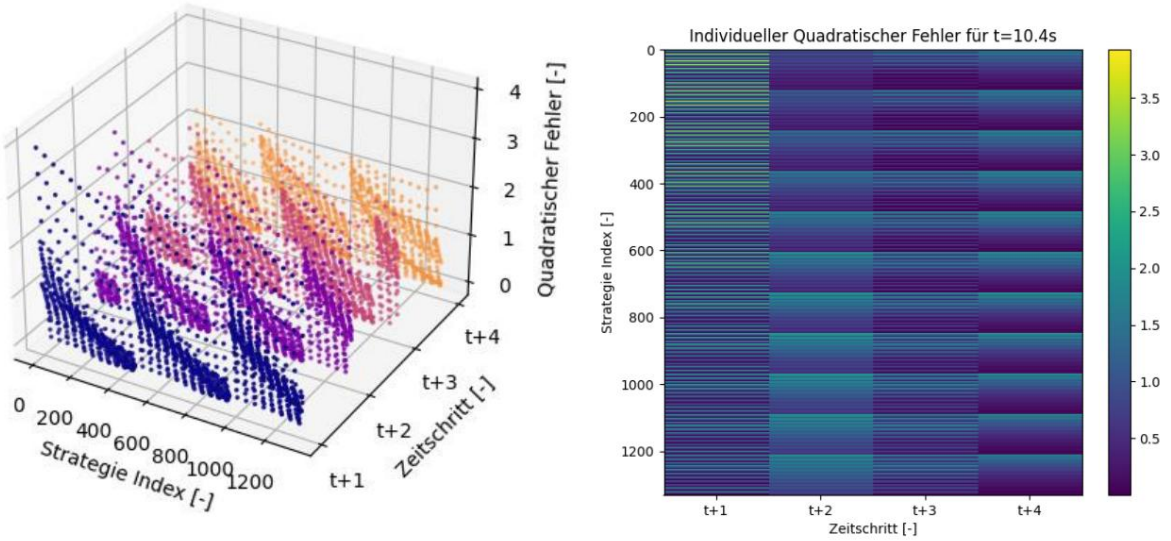


Figure 26: Square error over all control strategies and time steps of the first Prediction for t = 11.2s

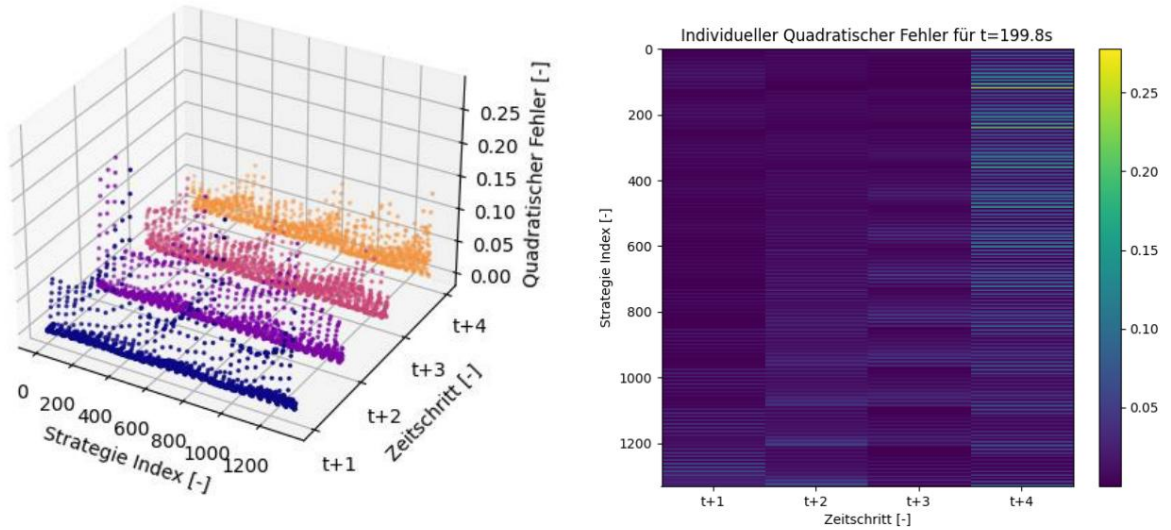
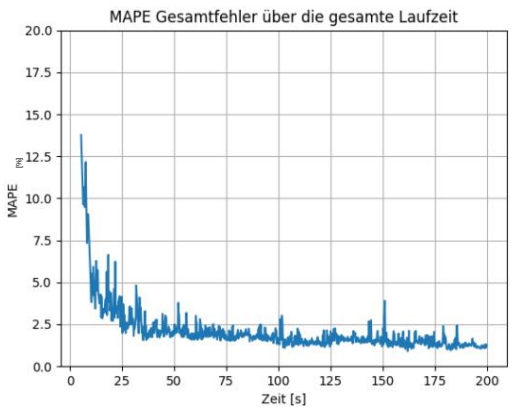


Figure 27: Quadratic error over all control strategies and time steps of the last Prediction for t = 199.8s

Figure 26 and Figure 27 show the squared errors of the predictions for each time step and for each control strategy in a 3D scatter plot (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 can be interpreted as a bird's eye view of the 3D scatter plot, whereby the size of the error is represented by a color gradient, similar to a map, which represents mountains.

The first of the two figures refers to the first prediction of the rule AI. It can be seen that the forecast is inaccurate at this time, with a maximum squared 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. The maximum The squared error is only about 0.25 at the end. It can also be seen that 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 The effect is explained in more detail in the section “Influence of h and m”.

Influence of the training interval ( ): The variation of is has influence on how often the rule AI is trained and thus also on how quickly it decreases, since after each training iteration the value of with The more often the rule AI is trained, the faster the probability decreases that random actions are performed.



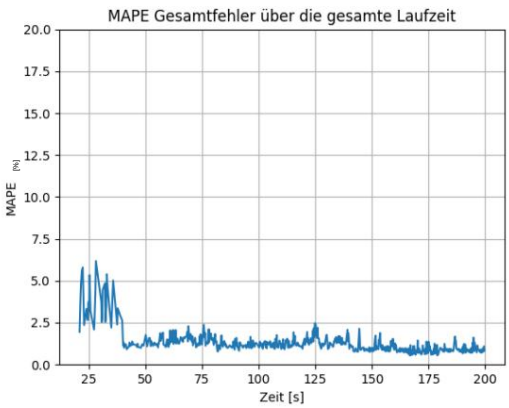


Figure 28: MAPE for ttrain = 5s (left) and ttrain = 20s (right)

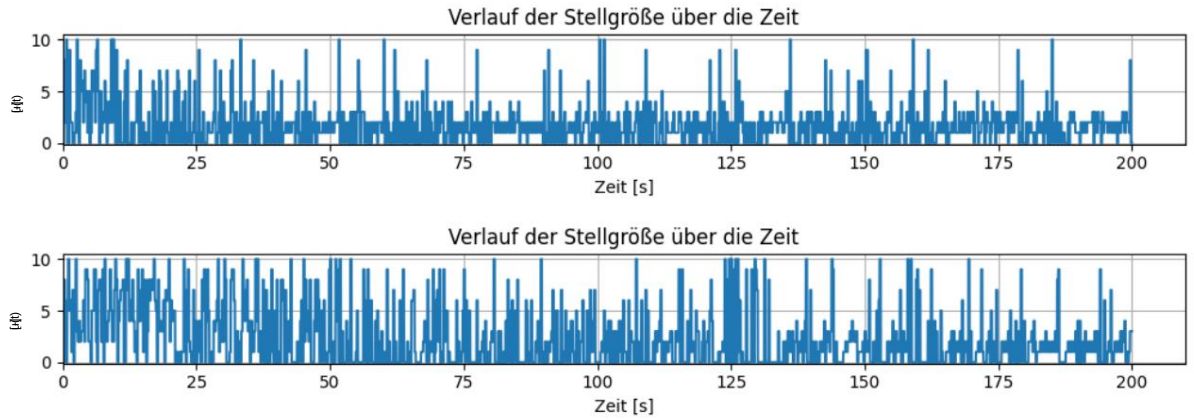


Figure 29: Course of the manipulated variable for ttrain = 5s (top) and ttrain = 20s (bottom)

The comparison of the prediction error at = 5 and = 20 is in Figure 28, in which the MAPE error of the prediction over the entire runtime is shown. The time axis of the graphs starts after 5s and 20s, because the rule AI does not make any predictions before the first training run, since until first training is 100%. An interesting result can be seen in the right graph see where the prediction error is after 40 seconds, i.e. after the second training iteration, drops sharply. After a runtime of 20 seconds, the course of the two errors are similar. It follows that with this system no major influence on the success of the regulation. The influence of on the manipulated variable is in Figure 29 shows that = 5 the fluctuations the manipulated variable are lower than = 20 because it decreases quickly and thus fewer random actions are performed.

Influence of the control interval ( ): The value of determines in which Time intervals 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 control variable can be changed less frequently, which can lead to a faulty Calculating the optimal control strategy leads to a larger fluctuation of the controlled variable. In addition, less training data is stored.

can also lead to larger prediction errors, since a larger one must predict further into the Values future. For this reason, for a larger longer temporal relationships between manipulated variable and controlled variable are recorded.

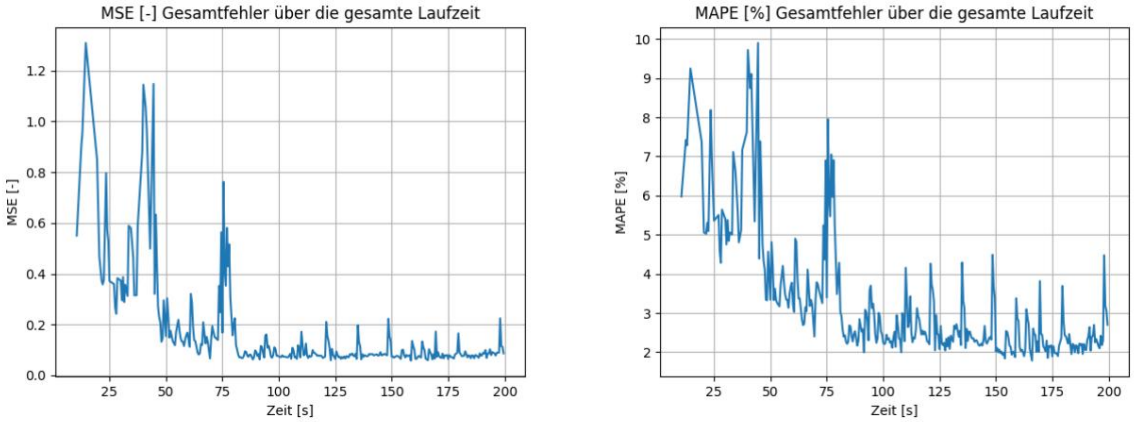


Figure 30: MSE and MAPE for tact = 500ms

The course of the prediction error for = 500 is shown in Figure 30. This is compared to the error for = 200 in Figure 25 over the higher and exhibits greater fluctuations over the entire term. This is probably due to that the time window of 4s is more than twice as large as the time window for = 200 which means that values have to be predicted further into the future, which leads 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 proactive the rule AI can act, but this also increases the Computational effort to evaluate the possible control strategies. The choice of The forecast horizon also depends on the complexity of the system. With less complex systems, it is useful to choose a smaller horizon in order to To reduce computational effort. For more complex systems with significantly delayed transmission behavior, a larger horizon is advantageous.

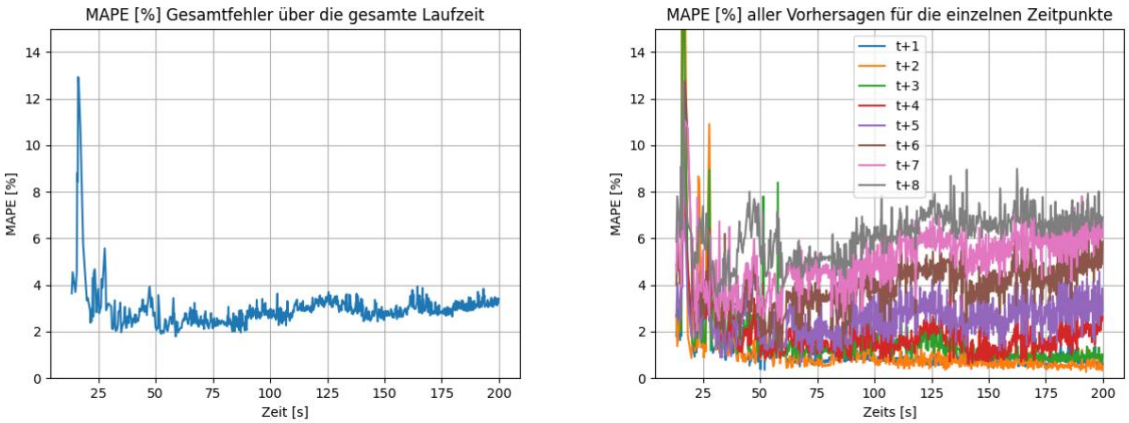


Figure 31: MAPE total error and MAPE for each time point for h = 8 and m = 2

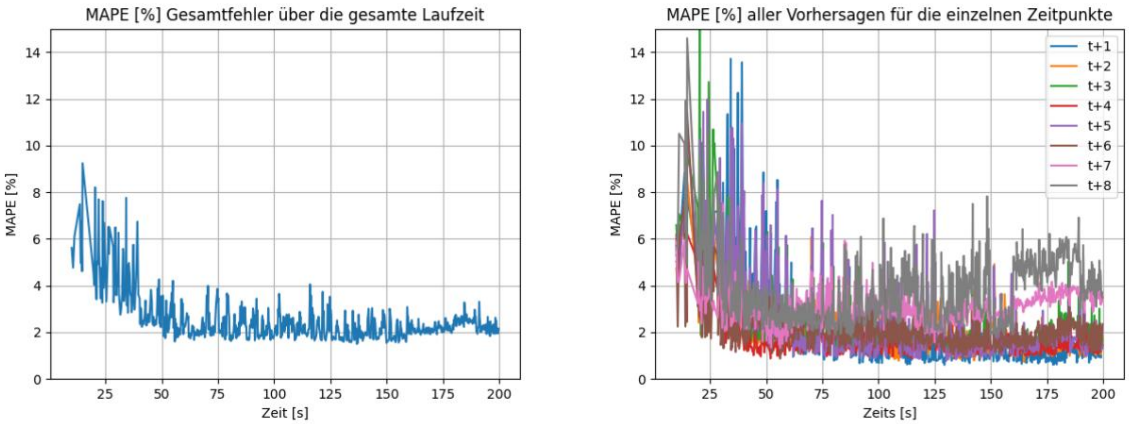


Figure 32: MAPE total error and MAPE for each time point for h = 8 and m = 4

Figure 31 shows that predictions for values further in the future
have higher errors than values that are closer in the future, which leads to a
This is due to the fact that = 2 remains, which means
the rule AI has to predict more values with the same inputs. Here,
Data to determine the temporal relationship for the values of the points in time between
+ and + ÿ. The bigger the difference between and ÿ, the
the prediction becomes less accurate for points in the future, such as the
Comparison between Figure 31 and Figure 32 shows. On the other hand, the larger
is, the greater the number of possible control strategies and the greater

the computational effort to evaluate all strategies. Basically, the choice of
Size of a compromise between control performance and computational effort. It is
necessary to find a balance that meets the requirements of the respective
application. The forecast horizon should be chosen so that
the temporal relationships between manipulated and controlled variables are as optimal as possible
without the difference between the forecast horizon and the
Control horizon becomes too large.

Influence of : The larger is, the more past values are used for the
prediction is used and the better the temporal relationships
It is expected that a larger one will lead to a smaller
prediction errors because more time points are taken into account for the prediction.

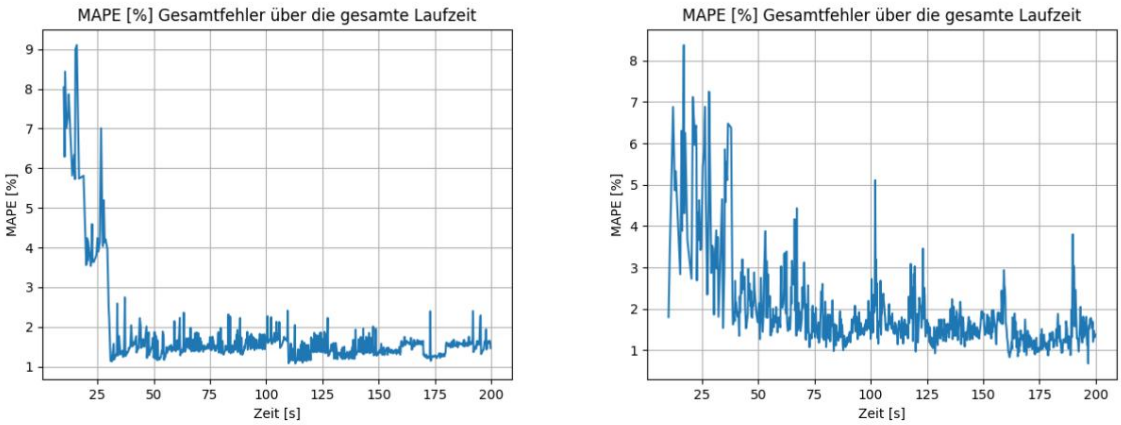


Figure 33: MAPE total error for n = 1 (left) and n = 8 (right)

Figure 33 shows the comparison of the MAPE total error for = 1 and = 8. In
In this case, the prediction for = 1 is more accurate. This is probably due to
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 manipulated and controlled variables
exist. In addition, more training data is available with a smaller time window. The
information about the slope is not available for a time window with = 1

recorded, 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 control variable in each case is reached and maintained. The parameters only affect the stability the control or the size of the fluctuations of the controlled variable and the prediction error of the rule AI.

4.2.2 Influence of process parameters on control success

The choice of the set of possible values for the manipulated variable depends on $_{1,2}$ and In any case, there should be a possible control variable ÿ that system can move towards the reference value. For example, the largest possible value of the manipulated variable must be able to change the controlled variable if it the minimum value is reached. For = 7, $_{1}$ = 0,2, $_{2}$ = 0.05, ( ) = 0 and ( ) = = 0 is the slope with **(41)** ÿ = ÿ1.45. In order for the system to Minimum value must be exceeded in this case ( ) > 1.45. For = 0.15 there must be at least one ÿ that is greater than 9.66 to determine the slope in the positive range. If this condition is not met, the controlled variable can no longer be influenced when = 0 is reached. The general condition for a successful regulation in this system is

$$> \ _{1} | ( ) ÿ | + \ _{2} ÿ ( ) > \quad (43)$$

A larger value for or larger values for the manipulated variable can lead to larger Fluctuations in the controlled variable, especially if the reference variable is in the Near , since then the slope is small and a large value for ( ) to which can lead to a large deflection of the controlled variable.

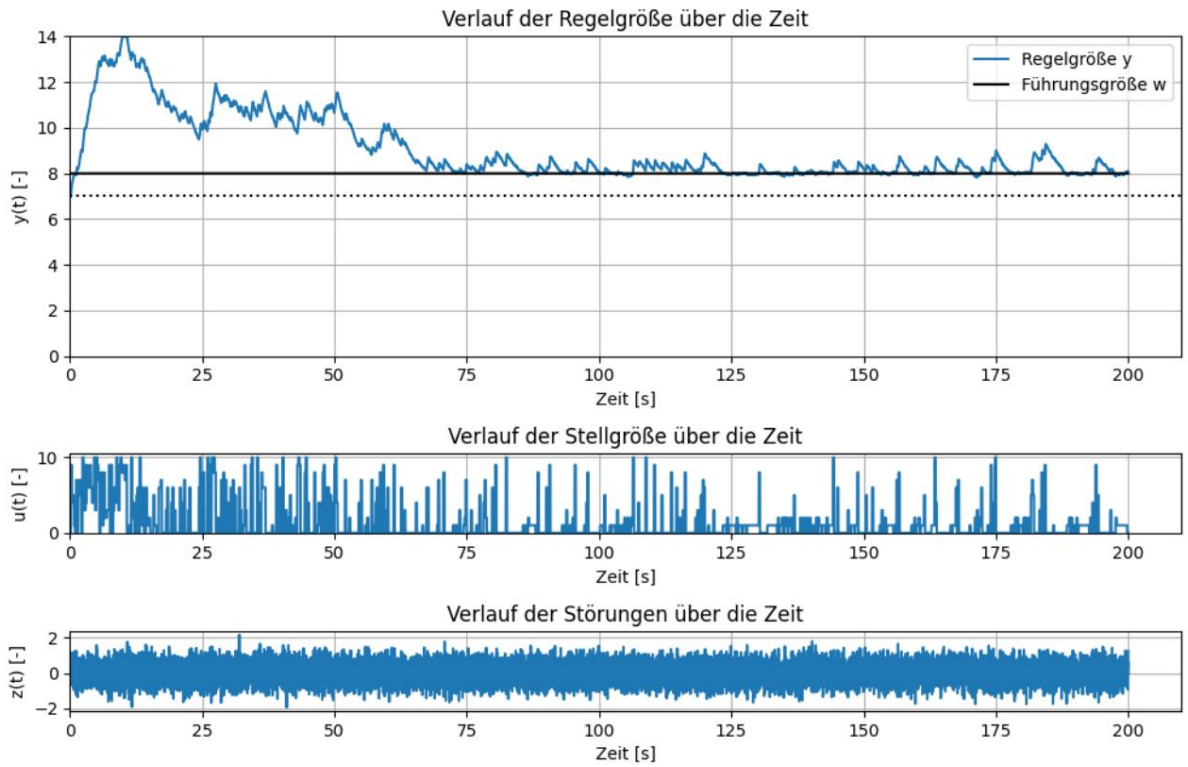


Figure 34: Simulation curve for ks = 0.3

Figure 34 shows the course of a simulation for = 0.3, with the same window size (= 4, = 2, ÿ = 4). In comparison with a simulation with = 0.15 (see Figure 24) the control with = 0.3 is more unstable and shows higher fluctuations. This is due to the fact that a deflection of the control variable with a larger has a greater influence on the controlled variable.

4.2.3 Influence of disturbances on control success

This section describes the response of the control AI to various disturbances

First, the influence of high noise is investigated.

The stability of the control AI is then tested by exciting the system with discrete disturbance pulses. Finally, the reaction of the control AI to a continuous, dynamic disturbance is examined.

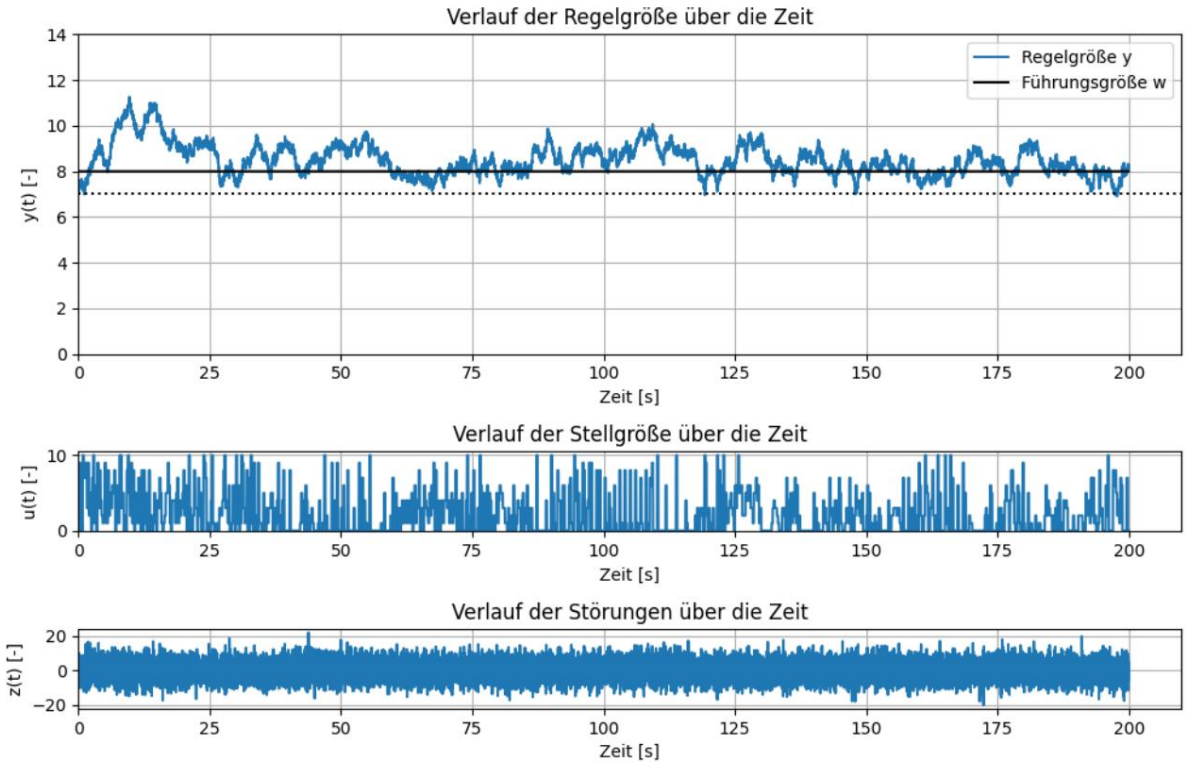


Figure 35: Simulation curve for high noise

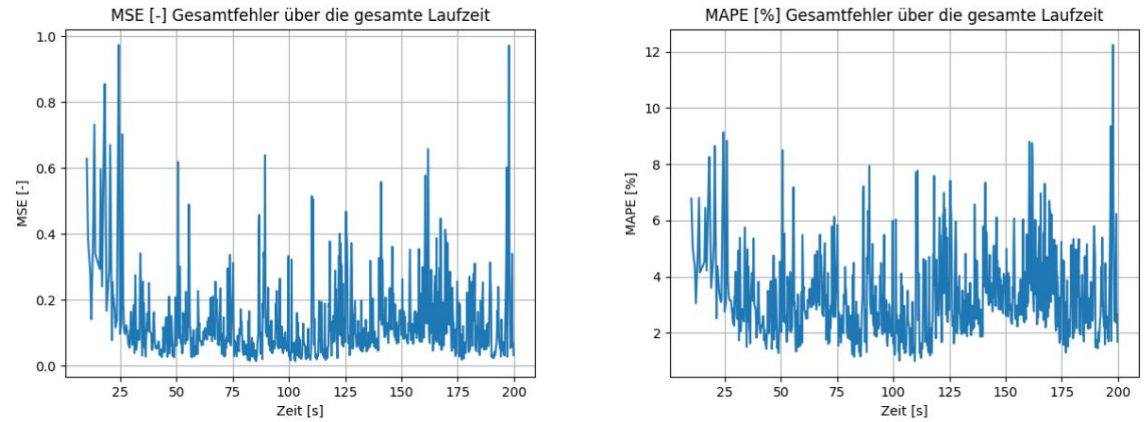


Figure 36: MSE and MAPE for high noise

Figure 35 shows the simulation for a high noise level with

= 5. Basically, noise represents the effect of small disturbances on each time step of the simulation. It is clear that high fluctuations and the control AI is not able to adequately control the system.

stabilize. Nevertheless, the controlled variable roughly approaches the reference variable. Figure 36 shows that the prediction error is volatile, which is due to the high random noise. Since these are random disturbances, the rule AI not able to accurately predict the controlled variable.

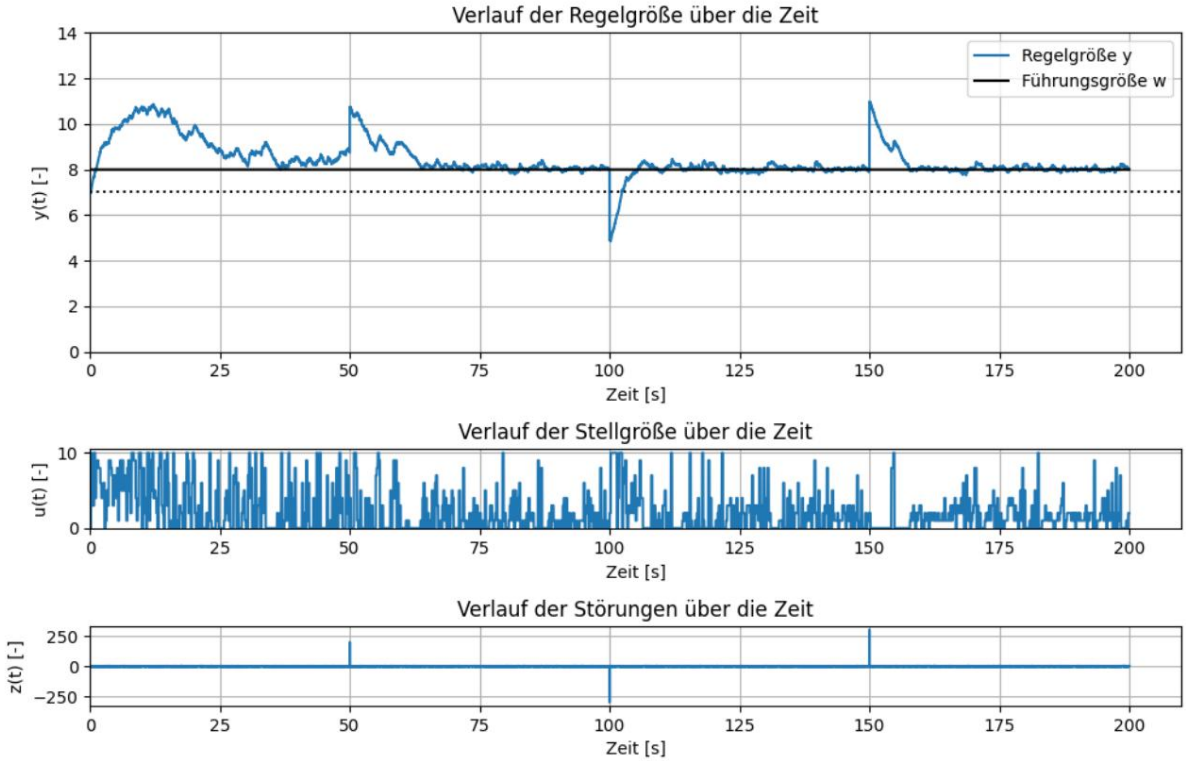


Figure 37: Simulation progression when excited by interference pulses

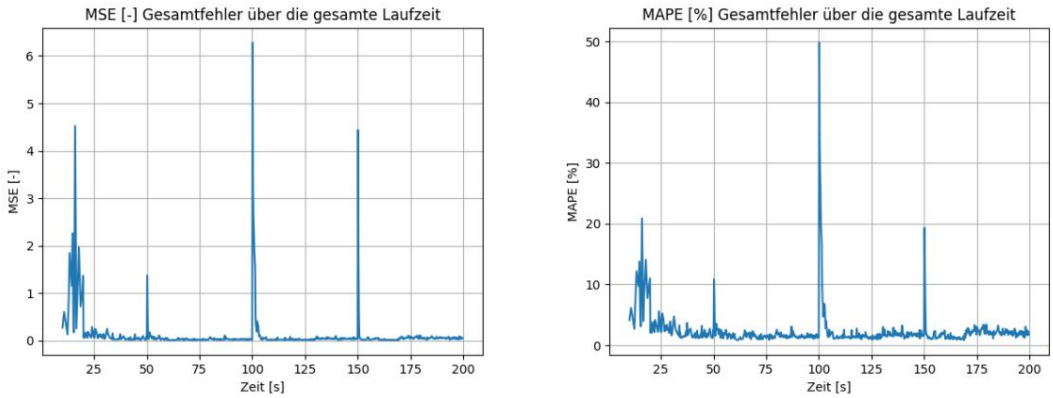


Figure 38: MSE and MAPE when excited by interference pulses

Figure 37 shows the course of a simulation in which at the times = 50
= 100 and = 150 an interference pulse occurs. In this case = 0.5.
It can be seen that the controlled variable at these times has a significant
After a few seconds, however, the control stabilizes
Figure 38 shows that at the time when a disturbance pulse occurs,
the prediction error increases sharply because the rule 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 allows stabilization of the control after a disturbance
enabled.

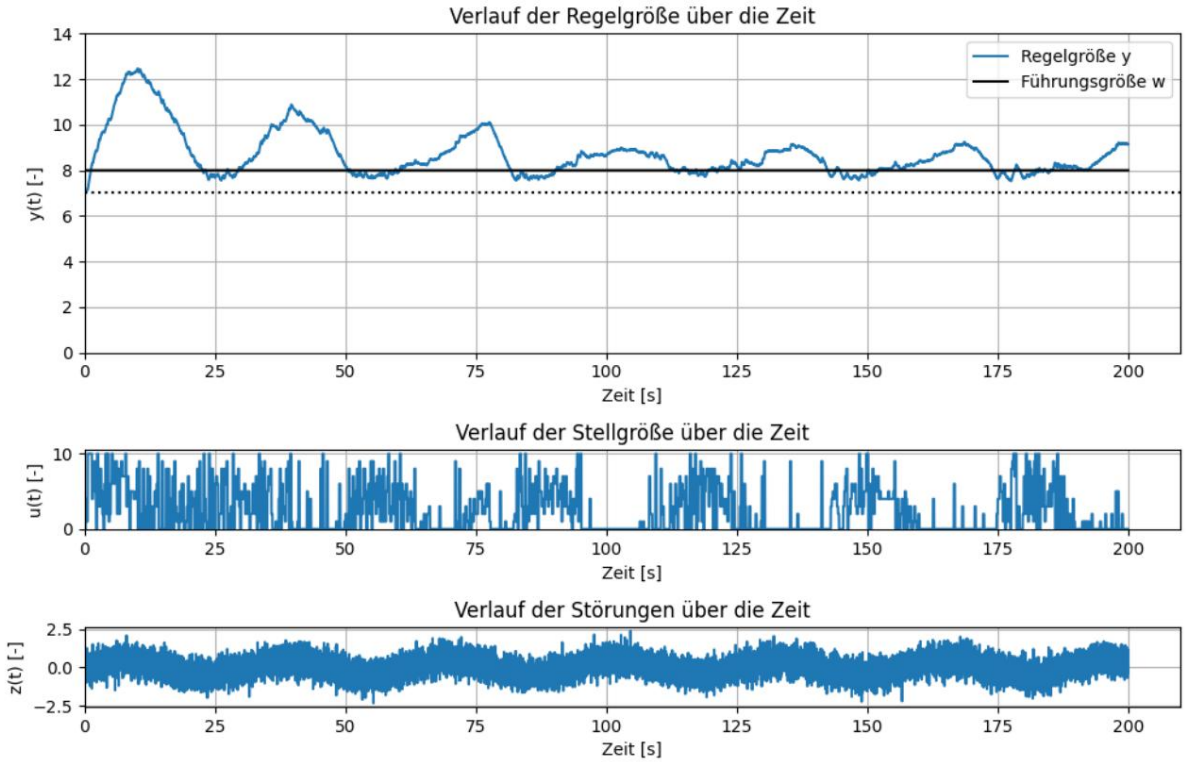


Figure 39: Simulation course for a dynamic disturbance behavior with A = 0.5

Figure 39 shows the course of a simulation with a dynamic disturbance behavior
The dynamic disturbance is represented in this case by the function

$$( ) = ÿ \sin(0{,}2) \quad (44)$$

with the amplitude = 0.5. The value for is again 0.5. The
Simulation history shows that the dynamic disturbance exceeds the expected value of the random noise, since ( ) and ( ) result in a total disturbance ( )
It can be seen that the control is unstable and strong
This is due to the fact that the condition **(43)**
is not fulfilled. If ( ) is to hold the reference variable, then ( ) = ( ) = 8 applies. The
The minimum expected value of the disturbance is -0.5. This gives us = 7,
$_1$ = 0.2 and $_2$ = 0.05 the condition < -0.25. And = 0, this
Condition is not fulfilled. The influence of the dynamic disturbance thus exceeds the Ability of the control AI to compensate for this. If the maximum amplitude of the dynamic disturbance is less than 0.25, the condition is met.

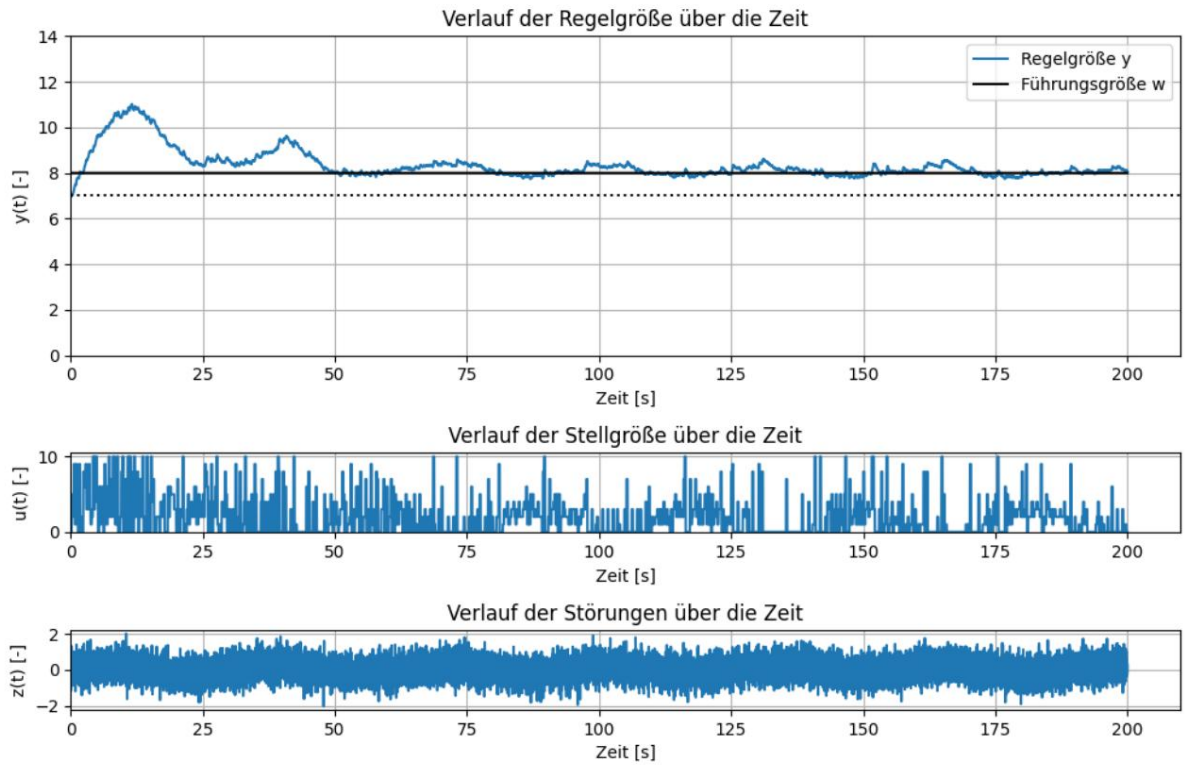


Figure 40: Simulation course for a dynamic disturbance behavior with A = 0.2

The simulation curve for the dynamic disturbance **(44)** with = 0.2 is shown in Figure 40. It can be seen that the regulation is valid if the condition **(43)** is stable.

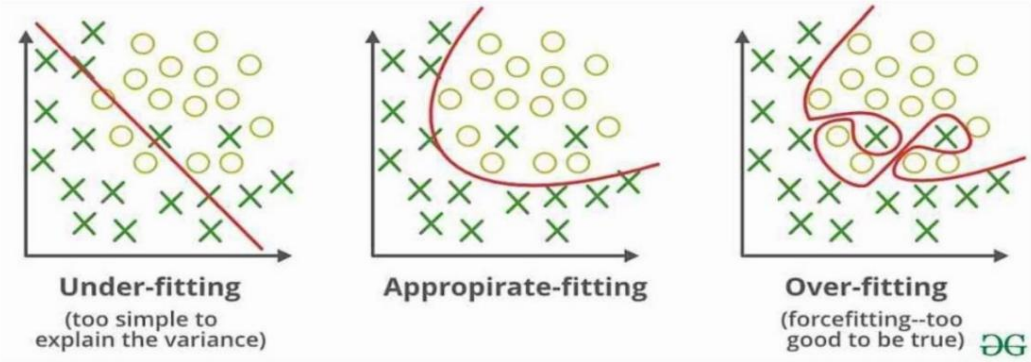


Figure 41: Representation of underfitting and overfitting [42]

A high degree of random disturbances can lead to “overfitting” when training an ANN. This happens when the model tries to identify relationships that do not exist in reality. This leads to the model being too closely tied to the training data and thereby loses its ability to develop new and unknown data. Figure 41 shows the difference between underfitting and overfitting. One method to Avoiding overfitting is to reduce the complexity of the model, by reducing the number of hidden layers. It can also The “dropout” method can be used, where random neurons are deleted. It can also be useful to store data with high To filter and normalize noise before training. [8]

Care should also be taken that the hyperparameter “Epochs” is not too large. The number of epochs indicates how often a training data set iterated, with the weights of the KNN being updated at each iteration. A A high number of epochs can lead to overfitting, while too low a number of epochs can lead to underfitting. This is especially true for this

Use case in which the KNN is continuously trained with new and small data sets
is trained. In all simulation runs, the number of epochs was set to 50.

4.2.4 Effect of the constraint ÿumax on the control success

In the following section, the control strategies are discussed taking into account the
Constraint **(34)** is selected. Only those control strategies are considered where the maximum
change in the manipulated variable between two points in time is less than
a given ÿ This reduces the number of possible control strategies
reduced and thus also the computational effort for the prediction. In addition, it cannot
There will be no more large jumps in the manipulated variable, which makes the control more stable.

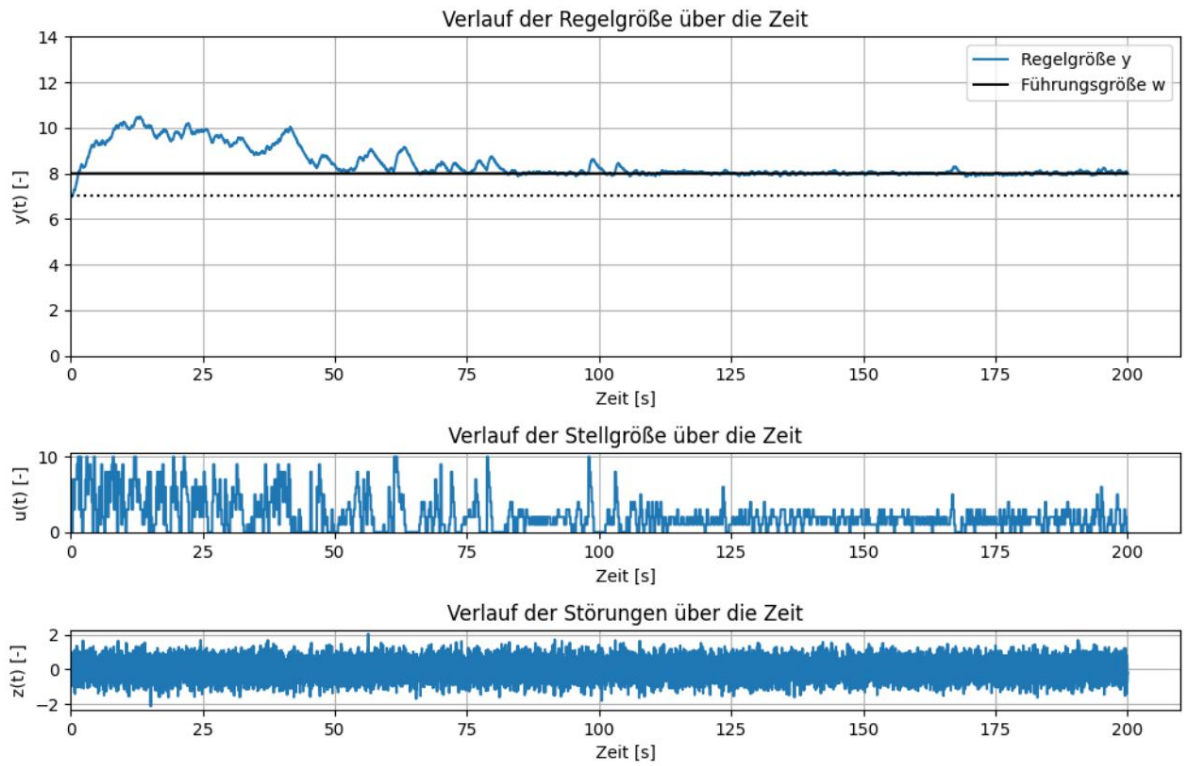


Figure 42: Simulation curve for ÿumax = 2

Figure 42 shows the course of the simulation for = 4, = 2, ÿ = 4 and ÿ = 2.
It is clear that from > 110 there are hardly any more deviations in the control variable and

the controlled variable maintains the reference variable almost exactly. For a control horizon of = 2 and the given quantity arise without using the Constraint 113 = 1331 possible control strategies. With ÿ = 2 the Number of possible control strategies to a maximum of 123. By introducing the As a result, the control becomes more stable while the computational effort is reduced.

4.2.5 Classification of factors influencing the success of regulation

This section describes the influence of the various parameters on the Control success is summarized. For this purpose, the parameters are sorted according to the size of their Influence on the success of the regulation is ranked.

Table 9: Ranking of factors influencing regulatory success

Ranking	Influencing factors
1	Process parameters (1, 2,)
2	Constraint ÿ
3	Size of noise
4	Difference between ÿ and
5	Line interval
6	Dynamic & Impulse Disturbances
7	Trainingsintervall
8	size of

Table 9 shows the ranking of the various factors influencing the Regulatory success. The greatest influence on regulatory success is the process parameters, which is due to condition **(43) .**

For regulation to be successful at all, there must be at least one discrete
Provide a manipulated variable that moves the controlled variable towards the reference variable.

The second largest influence is the constraint **(34),** which determines the maximum
The rate of increase of the manipulated variable is limited. It was found that this
significant influence on the stability of the control and the required
computational effort.

The third biggest influence is the amount of noise. If the noise is too large
the control AI can determine the temporal relationships between the manipulated variable and
Controlled variables are not learned sufficiently, which makes the control unstable.

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

The fifth largest influence is the size of the control interval the rule AI. The larger
the control interval, the less often the control variable can be changed, which leads to
can lead to larger fluctuations. A larger control interval also allows a larger time window, which can be useful
in the case of significantly delayed transmission behavior of
can be an advantage.

The sixth largest influence is the impulsive and dynamic disturbance behavior. It
It was observed that the control AI successfully adapts to impulse-like disturbances
The adaptability depends on the frequency and the
Size of the interference pulses occurring. Adaptation to dynamic disturbances
is also possible if condition **(43)** is fulfilled in every 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 variable.

The number of values from the
past that are used for prediction. It was found that
the control is successful even at = 1.

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

4.3 Control of a dynamic system

This chapter examines the adaptability of the control AI to dynamic changes in the 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 was tested by changing the internal state variables 1 and $_2$ of the system over time.

4.3.1 Change of the reference variable

The control AI should adapt as quickly as possible to a change in the control variable To this end, the following section considers a simulation in which
in which the reference variable is changed abruptly at discrete points in time.

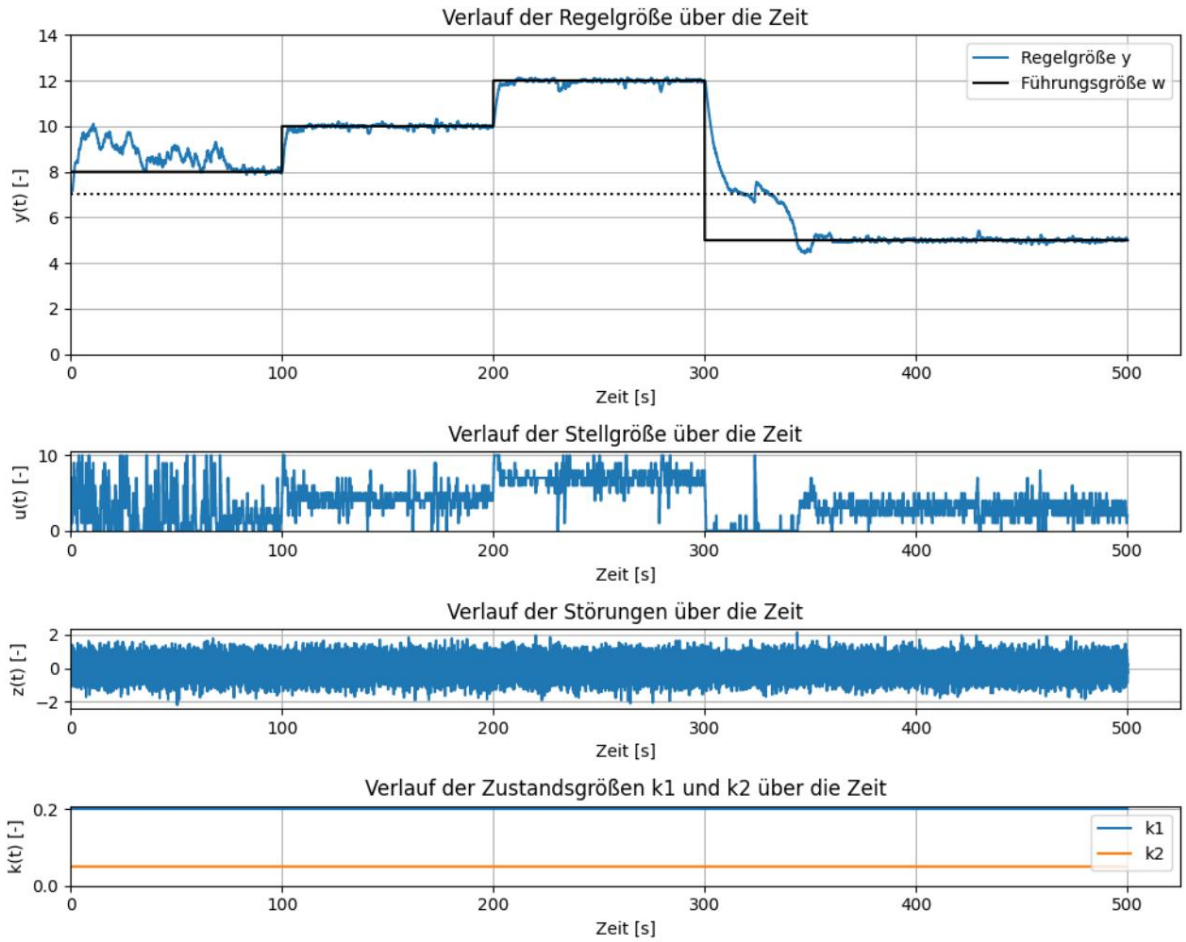


Figure 43: Simulation progression when changing the reference variable

Figure 43 shows the course of a simulation in which the reference variable at the times = 100 = 200 and = 300 jump to 10, 12 and 5 is changed. The simulation is based on the parameters used in Chapter 4.2 with additional consideration of ÿ = 2 and the window size = 4, = 2, ÿ = 4, = 200 = 10 . As can be seen from the figure, the control AI can successfully adapt to any change in the reference variable after a short time. This indicates that the control AI has relationship between manipulated variable and controlled variable. However, it should be noted that the success rate of adapting the control AI to a change in the The control variable depends on the complexity of the system. For a more complex System, it may happen that an adjustment is not successful.

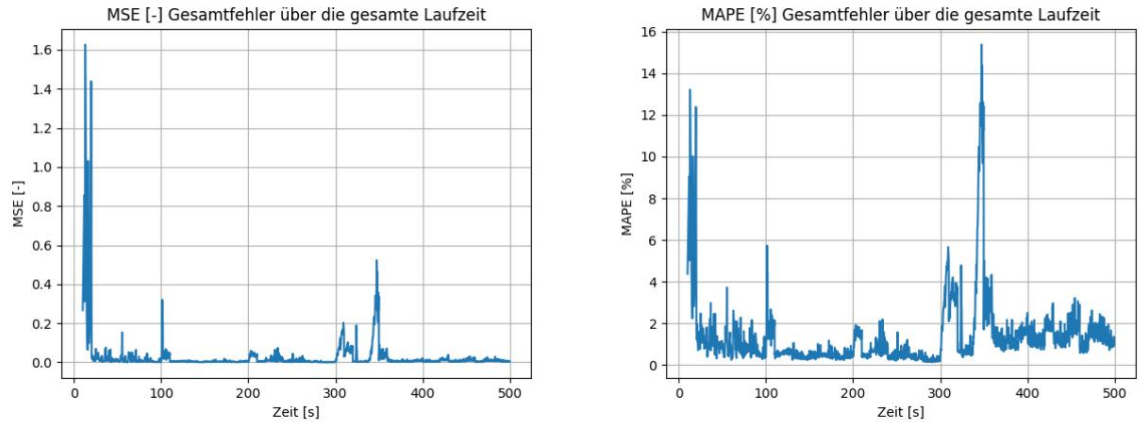


Figure 44: MSE and MAPE when changing the reference variable

Figure 44 shows that the total error at the times when
the reference variable is changed, increases and then falls again shortly afterwards. This is a
Indicator that the rule AI needs a certain amount of time to adapt to the new
system state. In any case, the prediction error decreases after a
Change of the reference variable back again.

4.3.2 Dynamic state variables k1 and k2

The rule AI should also be able to adapt to dynamic changes in the
To test this adaptability,
This section describes the control behaviour in the case of an unpredictable and a
predictable, dynamic change of $_1$ and $_2$ examined.

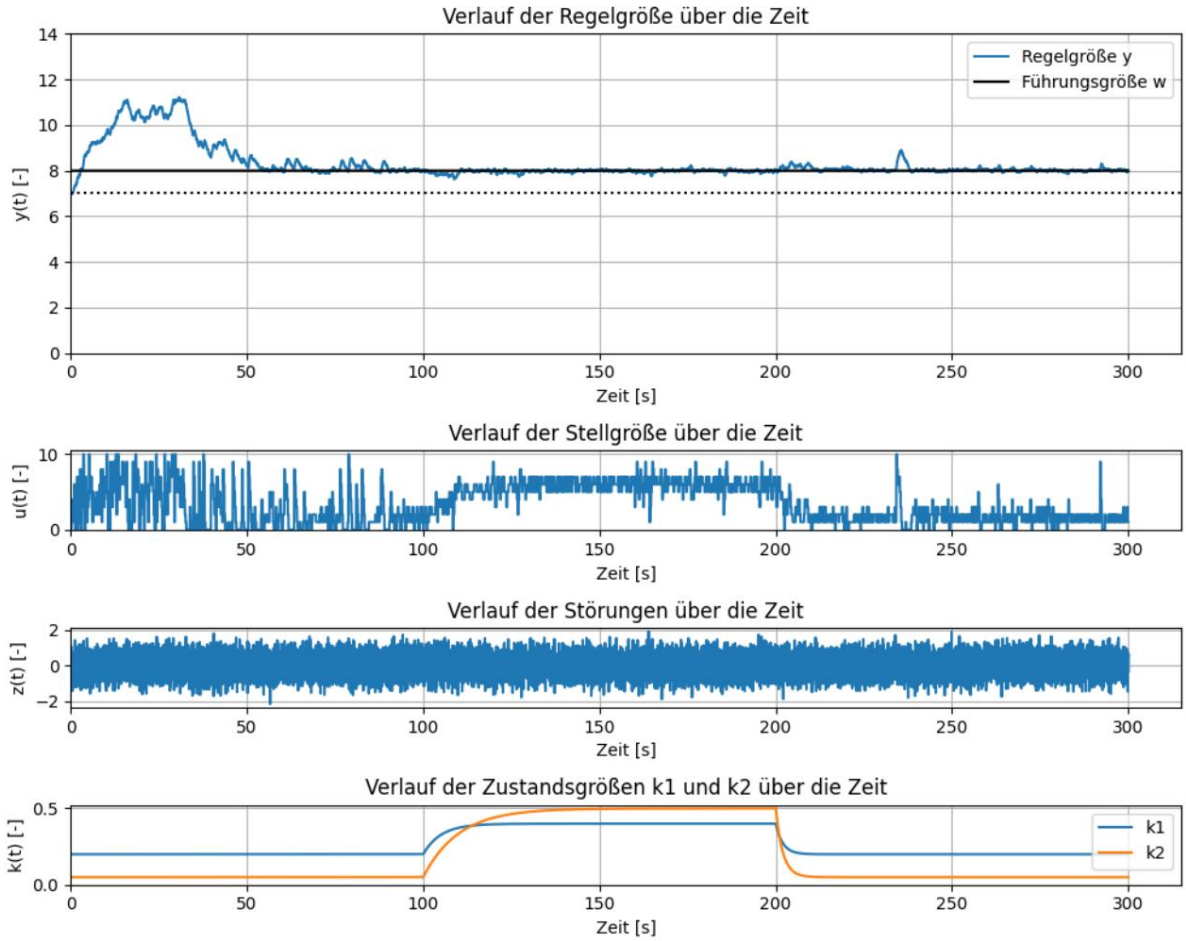


Figure 45: Simulation course with delayed step change of k1 and k2

To investigate an unpredictable change in the system properties, the system is changed become $_1$ and $_2$ at discrete points in time. This change is delayed by a PT1 element. Figure 45 shows the course of the simulation for a change of state variables 1 and $_2$ at the times = 100 and = 200 . Time = 100 is $_1$ to 0.4 and $_2$ set to 0.5. At time = 200 become $_1$ and $_2$ reset to their original values. The delayed Step response of the PT1 element is in this case

$$( ) = + (ÿ) ÿ \quad – \qquad (45)$$

For the first jump, the delay time for the change of for the change of $_1$ to 5s and $_2$ set to 10s. The value of $_2$ is therefore delayed for longer.

When resetting the values, both are changed with a delay of = 2 to slow down the response of the rule AI to sudden changes.

Figure 45 shows 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 in the system state, it must also be ensured that the condition **(43)**

is fulfilled. In this case, the change of $_1$ to 0.4 and $_2$ to 0.5 and = 0.15 the condition > 6 > which is satisfied with = {0,1, … ,10}.

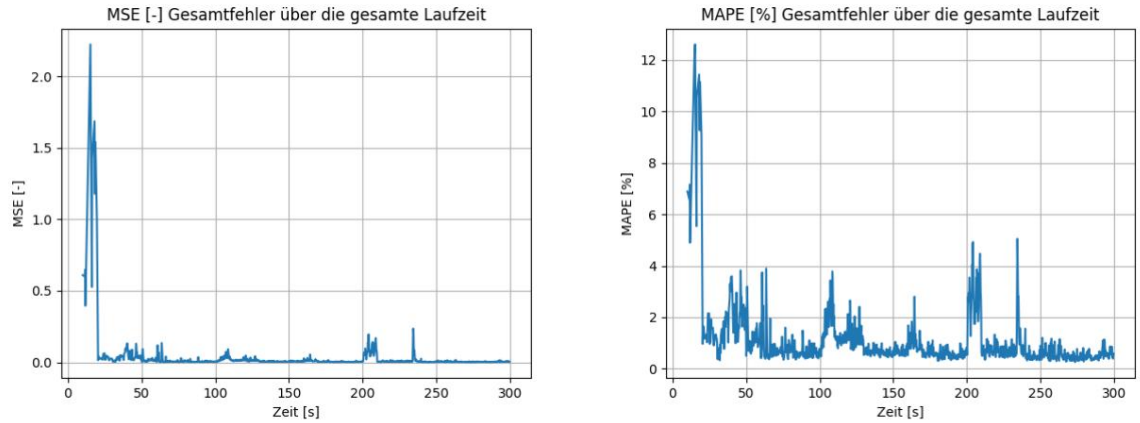


Figure 46: MSE and MAPE with delayed step change of k1 and k2

Figure 46 shows that the prediction error after changing $_1$ and $_2$ at = 100 and = 200 briefly and after a few seconds returns to the original value. This indicates that the control AI can adapt to an unpredictable dynamic change in the system properties.

In addition to unpredictable changes in the system state, there can also be predictable changes, for example when a state variable is periodic. If the control AI can successfully adapt to a predictable change, it can be expected that the prediction error will remain constant and not show large fluctuations.

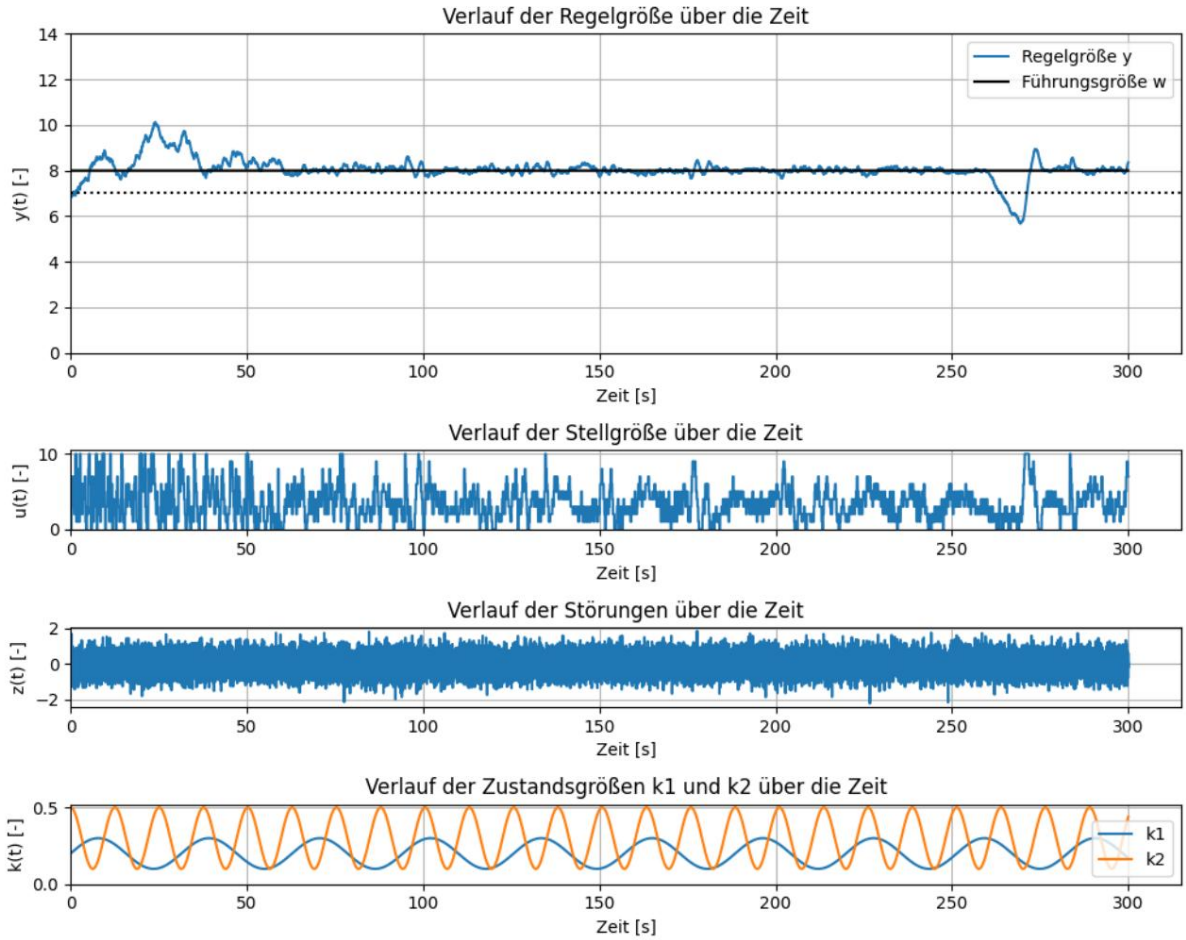


Figure 47: Simulation course with periodic change of k1 and k2

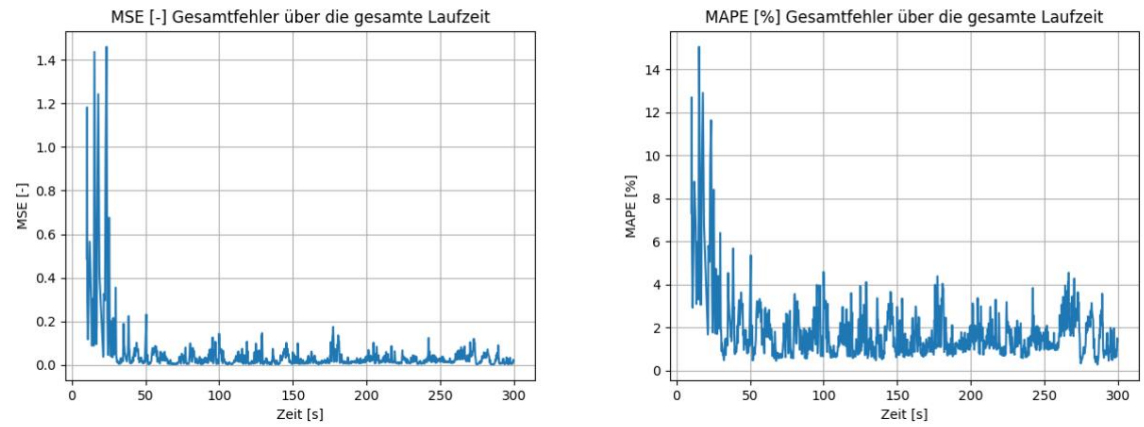


Figure 48: MSE and MAPE with periodic variation of k1 and k2

Figure 47 shows the course of a simulation in which $_1$ by a sine function and $_2$ is represented by a cosine function. From the figure it can be seen that the reference value is reached and remains stable except for a deflection at = 260 Even after a large swing, the control AI is able to maintain the to stabilize the controlled variable. Compared to an unpredictable change in $_1$ and $_2$ the prediction error in Figure 48 is more volatile overall, indicating that the rule AI does not adapt perfectly to complex periodic changes A variation of the window parameters has no effect in this case. significant influence on the prediction error. In order for the rule AI to successfully adapt to can adapt to a dynamic change of the internal state variables, it is It makes sense to measure these and use them as input parameters for the KNN. This enables the rule AI to understand more complex temporal relationships between the individual sizes.

4.3.3 Change of the reference variable and dynamic state variables

In this section, the change of the reference variable is compared with the periodic Behavior of $_1$ and $_2$ In addition, all interferences from chapter 4.2.3 (Noise, Impulse, Dynamic) are taken into account. The standard deviation of the noise is still 0.5. The system is checked every 50 seconds with a A disturbance pulse with random size is excited and the dynamic disturbance behavior is determined by the function **(44)** with = 0.2.

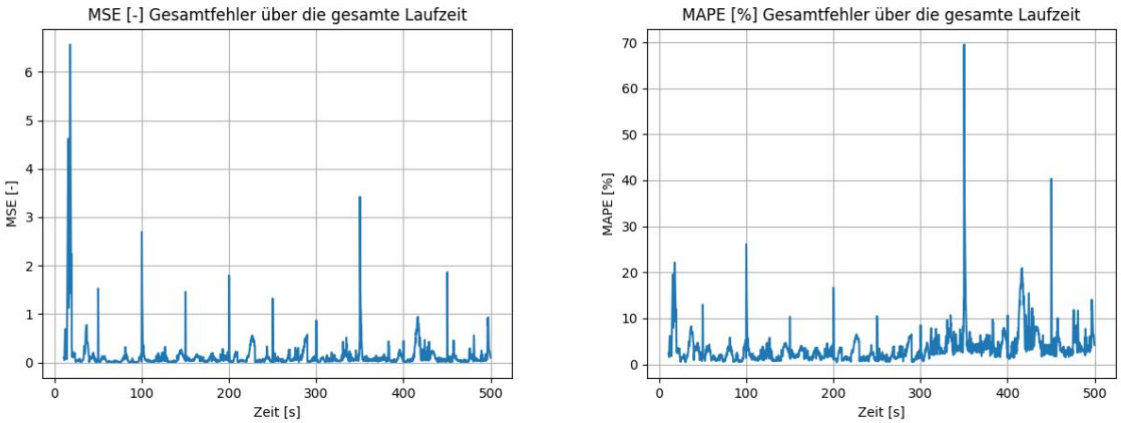


Figure 49: MSE and MAPE when changing the control variable, dynamic change of the state variables and influence of disturbances for n = 4, m = 3, h = 4

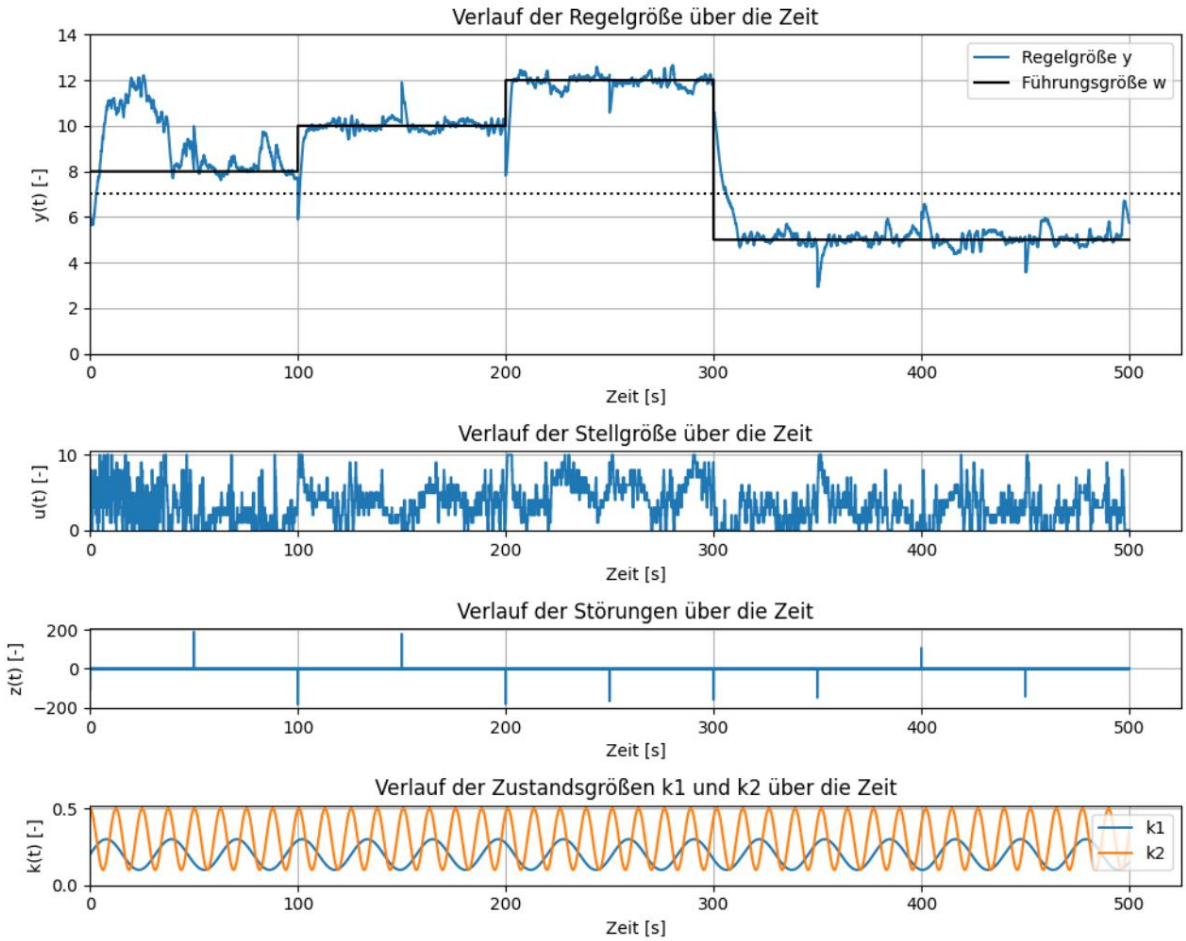


Figure 50: Simulation process with change of the reference variable, dynamic change of the state variables and influence of disturbances for n = 4, m = 3, h = 4

Figure 50 shows the course of the simulation, where the change in the Reference variable in section 4.3.1 and the periodic course of $_1$ and $_2$ dem in section 4.3.2. The only change is the value of $$ of 0.15 is increased to 0.25 so that the maximum reference value of 12 for the case $_{1,}$ = 0,3, $_{2,}$ = 0.5 and $$ = 0.2 can be maintained, or so that **(43)** is satisfied. Otherwise, in this case the maximum value for the Control variable $$ = 10 is not enough to maintain the reference value. Figure 49 shows that the forecast error is volatile and at the times when the System is stimulated with a disturbance pulse, exhibits strong oscillations. For the most forecasts, the error with MSE is less than 0.2 and with MAPE less than 5%. This

indicates that the control AI understands the temporal relationships between positioning, disturbance, state and control variables sufficiently learns and thus a successful Regulation with some fluctuations possible.

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

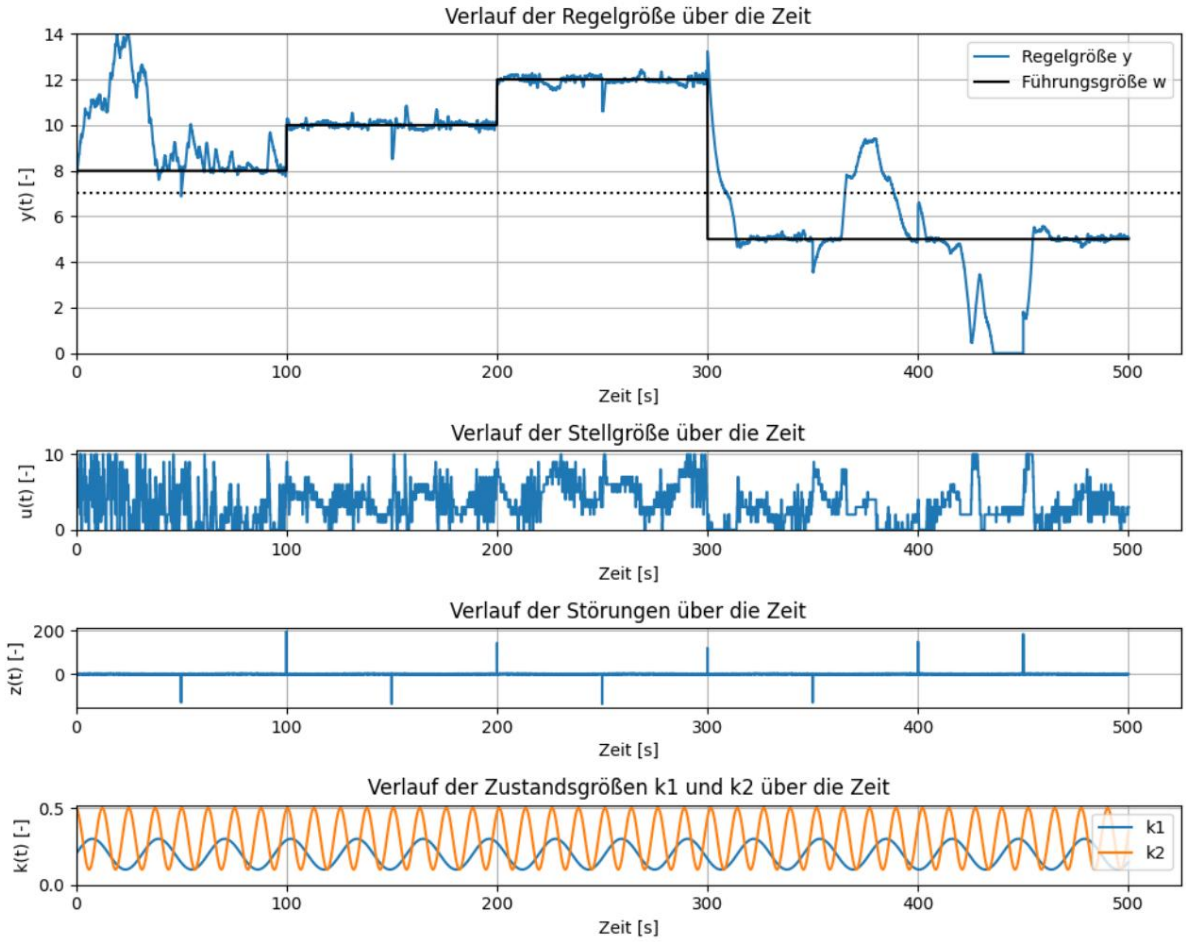


Figure 51: Simulation process when changing the control variable, dynamic change of the state variables and influence of disturbances for n = 1, m = 1, h = 2

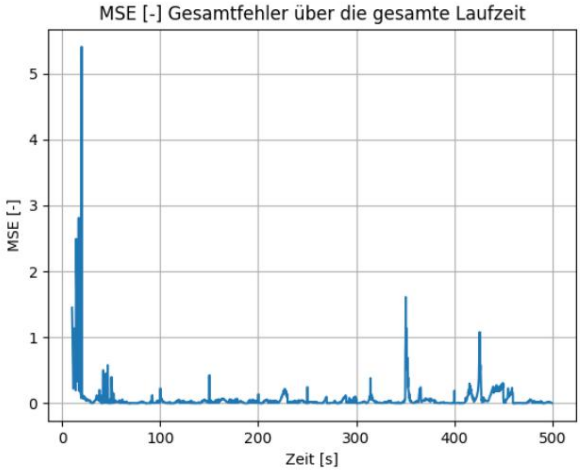


Figure 52: MSE when changing the reference variable, dynamic change of the State variables and influence of disturbances for n = 1, m = 1, h = 2

Figure 51 shows that the control with a small
time frame can become unstable and sometimes fail completely. It can be seen that
that the reference value is reached and maintained up to = 300, but at
Holding ( ) = 5 leads to high swings, which is because too little
past values are used for the prediction. Figure 52 shows that the
MSE prediction error for a smaller time window is smaller and more constant than for
a larger time window. The MAPE error cannot be used in this case
because the controlled variable reaches 0. The small prediction error is due to
due to the fact that with = 1 and ÿ = 2 few control strategies predicted
which leads to fewer erroneous predictions. Despite the lower
Due to the prediction error, the control with a small time window is overall more unstable.
It should therefore be ensured that the time window, depending on the application
the rule AI, is not chosen too small.

4.4 Transfer to a real process

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

The implemented rule AI can be adapted to any adjustments to a bioreactor for pH control. To achieve a better To obtain a process model, it is useful to include the controlled variable and the manipulated variable to measure 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 may require a separate controller to be installed, which temporarily takes over control in case of excessive fluctuations in order to avoid damage to the bioreactor due to pH values that are too high or too low. Otherwise Additional security mechanisms can be incorporated, such as Switching off the inlet valve when the pH value exceeds a certain limit Care should also be taken to ensure that a sufficiently large Time windows should be selected to take into account the diffusion of the inflow.

By using an ANN as a model for MPC, complex Interrelationships and non-linear relationships between manipulated and controlled variables without prior knowledge of the system properties. ANNs are also robust to changes in process parameters and can adapt to dynamic However, there is a risk that an ANN will become too attached to old data and cannot generalize new system states, which leads to failure of the control system. In addition, the computational effort for the

Using an ANN as a model higher than with statistical models, which leads to higher energy costs for the control. With regard to transferability to real processes, economic efficiency must also be taken into account.

The adaptability of a rule AI to different systems can indeed Save costs for the design of a specific controller, but requires the Use of more expensive computers, which also have higher energy requirements. For process controls that are operated with older computers, the transmission this rule AI is probably unsuitable, since in these cases the computing power is missing to evaluate a large number of control strategies in real time and at the same time continuously train the KNN with new data.

5 Summary and outlook

In this work, an adaptive rule AI was implemented, which uses a sliding time window the temporal relationships between the setting and controlled variable of a system and from possible control strategies the optimal To do this, the theoretical foundations of the Process control, AI and the control method MPC used are explained. The rule AI was then implemented in Python and tested on a virtual Proxy system tested.

It was found that the use of the ReLU activation function leads to dead neurons. To avoid this, the Leaky ReLU function was Furthermore, it was observed that the size of the time window in this System has no major influence on the control success. In order to reduce the computational effort and to increase the stability of the control, a constraint was introduced in the calculation of the optimal control strategy that limits the change in the manipulated variable between two points in time. It was observed that the control 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 in the event of dynamic changes in the internal state variables and under the influence of various disturbances was tested. It was found that the control AI is able to adapt to dynamic changes and To achieve and maintain a stable control value with only minimal fluctuations. The process parameters of the Proxy systems, which must be chosen so that a successful regulation is possible in the first place. Compared to a static system, the Size in a dynamic system has a greater importance, since temporal Relationships in a dynamic system with small only partially learned become.

These results show the potential for the integration of control AI into certain physical processes. For real-time control, it is important that sufficient

computing power is available. A graphics processor (GPU) with high memory capacity, a CPU with multiple cores and a sufficiently large RAM should be used to store the time series. The Graphics processor plays the most important role as it enables the parallel execution of many computationally intensive matrix operations, enabling fast prediction and rapid training of the KNN is ensured.

However, it should be noted that further tests and analyses are being carried out to ensure the effectiveness and applicability of rule AI in real processes To this end, the rule AI can be used in a future work on a bioreactor for pH control and tested, whereby possible adjustments by incorporating additional safety conditions In addition, it can be tested whether an LSTM network is suitable for the Predicting a time series is better suited than a normal DNN, because LSTM networks optimized for predicting sequential data.

6 Bibliography

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

[2] Hewing, L., Wabersich, K. P., Menner, M., Zeilinger, M. N., 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 Fundamentals, Analysis and Draft single-loop regulations,* 5th edn., Springer, Berlin, 2006.

[5] Horn, J., Structural image and transmission elements, in: Plaßmann, W., Schulz, D. (Ed.). *Handbook of 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, A. L., 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,* 2 March 2019. https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-1-d0368d769a7b. Accessed 8 December 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, R. S., 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,* 18 July 2019. https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da. Accessed 9 December 2022.

[17] Kaelbling, L. P., Littman, M. L., Moore, A. W., 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. *THERE.*

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

[20] Sydenham, P. H., 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, M. A., *Neural Networks and Deep Learning,* Determination Press, 2015.

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

[24] Saurabh, Backpropagation – Algorithm For Training A Neural Network. *Edureka,* 7 December 2017. https://www.edureka.co/blog/backpropagation/. Accessed 12 December 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 13 December 2022.

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

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

[29] M.V. Shcherbakov, A. Brebels, N.L. Shcherbakova, A.P. Tyukov *et al.,* A survey of forecast error measures. *World Applied Sciences Journal* 2013, *24,* 171–176, DOI: 10.5829/dosi.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. , Sorjama , *A.* *strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition,* 2011.

[34] K. S. Holkar, L. M. 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,* 4 December 2016. https://machinelearningmastery.com/time-series-forecasting-supervised-learning/. Accessed 1 December 2022.

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

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

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

[39] GeeksforGeeks, Epsilon-Greedy Algorithm in Reinforcement Learning - GeeksforGeeks. https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/. Accessed 12 January 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,* 23 November 2017. https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/. Accessed 21 January 2023.

7 List of figures

8 List of tables

9 Appendix

9.1 Python script from “main.py”

```
from simulation.simulation import SimKeys from simulation.simulation import
Simulation from simulation.simulation import SimConfig from export import Data from
control.control_ai import Agent, AgentConfig




def run_simulation(s_config: SimConfig, agent_configs): sim_time = 200_000 sim_steps = int(sim_time /
        s_config.dt)


        for agent_config in agent_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       in range(sim_steps):
                # Change reference here if needed#
                ####################################


                action, predictions = agent.choose_action(target, time,
sim.current_u)

                if predictions is 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)

                day = 0
                # Add disturbance impulse z here if needed #
                ############################################## target_, time, temp =
                sim.step(action, zi, agent.reference)
                if action is not None:
                        agent.record(temp)
                agent.train_model(time) target = target_
                data.finish(sim)



def get_agent_configs():
        # Create all the Configs which should be compared

        c1 = AgentConfig(reference=8, action_space=[0, 1, 2, 3, 4, 5, 6, 7, 8,
```

```
9, 10])
        c1.n = 4
        c1.m = 2 c1.h
        = 4 c1.act_every
        = 200 c1.train_every = 10_000


        return [c1]


if __name__ == '__main__':
        sim_config = SimConfig()
        sim_config.noise = 0.5 sim_config.dt
        = 10 # Time in ms sim_config.start_value = 7
        sim_config.ks = 0.15 sim_config.stable_value
        = 7


        run_simulation(sim_config, get_agent_configs())
```

9.2 Python script of the rule AI from “control_ai.py”

```
import numpy as np import
control.utils as utils import pandas as pd from
collections import deque from
tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense, LeakyReLU from
tensorflow.python.keras.optimizer_v2.adam import Adam



class AgentConfig:

    def __init__(self, reference, action_space, du_max=None, sys_lim=None): self.max_mem_size = 10_000 self.n = 2 # h needs to be greater
        than m and > 0, m can be 0 # Control horizon self.m
        = 2



        # Prediction horizon
        self.h = 4
        self.epsilon = 1 self.epsilon_dec
        = 0.7 self.epsilon_min = 0.01 self.batch_size
        = 10 self.epochs = 50 # Train model every 10
        seconds self.train_every = 10_000 #
        Act every 200ms self.act_every
        = 200 self.alpha = 0.7 self.reference = reference
        self.action_space = action_space self.du_max =
        du_max self.sys_lim = sys_lim




def build_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), loss='mse')

    return model


class Agent:
    def __init__(self, config: AgentConfig, c_key, t_key):
        assert config.h > config.m self.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_lim
            self.action_space = config.action_space
            self.du_max = config.du_max self.alpha = config.alpha self.epsilon
            = config.epsilon self.epsilon_dec =
            config.epsilon_dec self.epsilon_min =
            config.epsilon_min self.batch_size =
            config.batch_size self.epochs = config.epochs self.reference =
            config.reference self.n = config.n self.control_key = c_key
            self.target_key = t_key self.m = config.m self.h = config.h
            self.train_every = config.train_every
            self.last_trained = 0 self.act_every = config.act_every
            self.last_acted = 0
            self.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,

du_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]] for i in range(len(self.memory) - self.n,
                                len(self.memory))]
            data = np.array(data, dtype=np.float)

            if self.sys_lim is 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 if self.sys_lim is 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)


            return data

        def get_allowed_strategies(self, current_u): if self.du_max is None: return
            self.control_strategies


            # Filter possible control strategies with constraint du <= du_max diff = np.abs(current_u - self.control_strategies[:,
            0]) return self.control_strategies[diff <= self.du_max]


        def record(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] for k in inputs} self.memory.append(y) self.temp_memory.append(y)




        def choose_action(self, current_y, t, u):
            # Returns next action and all the predictions the ANN made if t < self.last_acted + self.act_every:

                return None, None

            self.last_acted = t rand =
            np.random.random() if len(self.memory) <
            self.n or rand < self.epsilon:
                # Pick random action action =
                np.random.choice(self.action_space) return action, 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) if self.sys_lim is 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() if rand <= 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

            return best_strategy[0], predictions

        def train_model(self, t):
            if len(self.memory) < self.n + self.h + 1 + self.batch_size or t < self.last_trained + self.train_every:
                return

            df = pd.DataFrame.from_records(self.temp_memory)

            # Normalize data
            if self.sys_lim is 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_dec if self.epsilon >
self.epsilon_min else self.epsilon_min
            self.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)))
    return np.sum((trajectories - reference) ** 2 * weights, axis=1)
```

9.3 Python script from “utils.py”

```
import itertools import numpy
as np import pandas as pd



def convert_input_data_training(df: pd.DataFrame, n, m, h, c_key, t_key):
        assert h >= m
        assert h >= 1

        inputs = [t_key, c_key] cols, names = [], []

        # Input sequence (t-n, ... t-1) for i in range(n, -1, -1):

                cols.append(df.shift(i)) if i == 0:

                        names += [f'{x}(t)' for x in inputs] else:

                        names += [('%s(t-%d)' % (x, i)) for x in inputs]

        # Input sequence (u_t, u_t+1 ..., u_t+m) for i in range(1, m + 1):
        cols.append(df[c_key].shift(-i)) names += ['%s(t+
                %d)' % (c_key, i)]


        target_index = len(names)

        # Output sequence (t+1, ... t+h) for i in range(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)


        return data.iloc[:, :target_index].copy(), data.iloc[:, target_index:].copy()



def compute_all_possible_strategies(m, action_space: list, du_max=None):
        strategies = np.array(list(itertools.product(action_space, repeat=m + 1)), dtype=np.float)


        if du_max is not None:
                # Filter out strategies that dont match the constraint diff = np.abs(np.diff(strategies, axis=1)) strategies =
                strategies[np.max(diff, axis=1) <= du_max]


        return strategies
```

```
def normalize(x, min_x, max_x):
    return (x - min_x) / (max_x - min_x)

def denormalize(y, min_x, max_x):
    return y * (max_x - min_x) + min_x
```

9.4 Python script of the simulation from “simulation.py“

```
import numpy as np from
scipy.integrate import odeint


class SimKeys:
        TIME = "time"
        VALUE = "value"
        CHANGE = "change"
        DELTA = "delta"
        ACTION = "action"
        REFERENCE = "reference"
        K1 = "k1"
        K2 = "k2"
        Z = "z"


def k1_function(t): # if t < 100: # # elif
        t < 200: # return 0.4 - 0.2
        *               return 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 return 0.2


def k2_function(t): # if t < 100: # # elif
        t < 200: # return 0.5 -
                return 0.05
        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 return 0.05


def zd_function(t): #return 0.2 *
        math.sin(0.1 * t) return 0


class SimConfig: # dt in ms def
        __init__(self): self.dt
        = 10

                self.k1 = lambda t: k1_function(t) self.k2 = lambda t: k2_function(t)
                self.zd = lambda t: zd_function(t)

                self.ks = 0.15
```

```
                self.min_value = 0 self.max_value
                = 14 self.stable_value = 7
                self.start_value = 7 self.noise = 0.5



class Simulation:
    def __init__(self, simulation_config: SimConfig): self.dt = simulation_config.dt self.start_val =
                simulation_config.start_value self.current_val =
                simulation_config.start_value self.stable_val = simulation_config.stable_value self.k1 =
                simulation_config.k1 self.k2 = simulation_config.k2 self.ks = simulation_config.ks self.zd =
                simulation_config.zd self.min_val = simulation_config.min_value self.max_val =
                simulation_config.max_value self.noise =
                simulation_config.noise self.df = [] self.current_t = 0
                self.current_u = 0









    def step(self, u, zi, w): if u is 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 if self.current_val <
                self.min_val: self.current_val = self.min_val elif self.current_val >
                        self.max_val: self.current_val = self.max_val
```

```
            self.current_t += self.dt

            return self.current_val, self.current_t, temp

        def reset(self):
            self.current_t = 0
            self.current_u = 0
            self.df.clear()
            self.current_val = self.start_val

            return self.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) * abs(self.stable_val -
self.current_val) - self.k2(
                    self.current_t / 1000) + self.ks *                    u + z) * self.dt / 1000

        def calculate_strategies(self, strategies: np.array, p_horizon,
act_every):
            # Calculate the actual influence of the different control
strategies on the system
            def model(y, t, u, dt, t0):
                return -(self.k1(t0 + t) * abs(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 / 1000
            t_eval = np . range ( 0 , p_horizon + 1 , 1 ) * delta_t
            y0 = np.ones(shape=strategies.shape[0]) * self.current_val

            values = odeint(model, y0, t_eval, args=(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:]

            return values
```

9.5 Python script for visualizing the results from „export.py“

```
import numpy as np import
pandas as pd import os import
pathlib import
datetime import os.path
import matplotlib.pyplot
as plt from
simulation.simulation import SimKeys, Simulation
import matplotlib.cm as cm



class ExpKeys:
    TIME = "Time (ms)"
    MSE_TOTAL = "mse_total"
    MAPE_TOTAL = "mape_total"
    MAE_TOTAL = "mae_total"


class Data:
    def __init__(self, plots_every, agent_config, sim_config):
        self.plots_every = plots_every self.last_plotted =
        None self.last_individual_error = None
        self.last_individual_time = 0 self.total_errors = []
        self.mse_tsteps = None self.mape_tsteps =
        None self.mae_tsteps = None
        self.agent_config = agent_config
        self.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 not os.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)


        return dir_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")

            with open(path1, "w") as f1, open(path2, "w") as f2:
                f1.write(''.join(["%s = %s\n" % (k, v) for k, v in
self.sim_config.__dict__.items()]))
                f2.write(''.join(["%s = %s\n" % (k, v) for k, v in self.agent_config.__dict__.items()]))


    def finish(self, sim: Simulation):
            dir_name = os.path.join(self.image_path, "summary")
            os.makedirs(dir_name)

            create_sim_linechart(sim, dir_name)

            if len(self.total_errors) == 0:
                return
            total_error = pd.DataFrame.from_records(self.total_errors)
            time = total_error[[ExpKeys.TIME]].div(1000)
            if self.mse_tsteps is not None and self.mape_tsteps is not None and
self.mae_tsteps is 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)

            if self.last_individual_error is 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))

    def record_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)) * 100
            mae_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) * 100 mae_tsteps = np.mean(np.abs(actual_values - predictions),
axis=0)


            # Add different error measures for each time step to an array if self.mse_tsteps is 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])

            if self.mape_tsteps is 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])

            if self.mae_tsteps is 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) ** 2 self.last_individual_error =
            quad_individual self.last_individual_time = time


            # Graphs are only saved after a set intervall if self.last_plotted is None or
            time >= self.last_plotted + self.plots_every: self.last_plotted = 0 if self.last_plotted is None else time



                mse_strategies = np.mean((predictions - actual_values) ** 2,
axis=1)
                mape_strategies = np.mean(np.abs((actual_values - predictions)
/ actual_values), axis=1) * 100
                mae_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)


def create_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)}" for i in range(errors.shape[1])]
    fig, ax = plt.subplots(figsize=(7, 6))
    im = ax.imshow(errors, interpolation='nearest', cmap='viridis', aspect='auto')

    ax.set_xticks(np.arange(len(x_labels)), labels=x_labels)
    ax.set_xlabel("Timestep [-]")
    ax.set_ylabel("Strategie Index [-]")
    plt.colorbar(im)
    plt.title(f"Individual squared error for t={str(time / 1000)}s")

    plt.savefig(path)
    plt.clf()
    plt.close()


def create_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)}" for i in range(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("Strategie Index [-]")
    ax.set_ylabel("Timestep [-]")
    ax.set_zlabel("Squared error [-]")

    plt.title(f"Individual squared error for t={str(time / 1000)}s")


    plt.savefig(path)
```

```
        plt.clf()
        plt.close()


def create_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()


def create_error_bar_chart_steps(errors: np.array, error_type, time, path):
        x_labels = [f"t+{str(i + 1)}" for i in range(errors.shape[0])]

        plt.ioff()
        plt.figure()
        plt.bar(x_labels, errors)
        plt.ylabel(error_type)
        plt.xlabel("timestep")
        plt.title(error_type +                  " for the prediction of 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()


def create_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)}" for i in range(errors.shape[1])]

        for tstep in errors.T:
                plt.plot(time, tstep)

        if ylim is not None:
                plt.top(top)
        plt.grid(visible=True)
        plt.ylabel(error_type)
        plt.xlabel("Time [s]")
        plt.title(error_type + " of 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()



def create_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("Zeit
        [s]") if ylim is not None: plt.ylim(ylim)
        plt.title(error_type + f_name =
                error_type +
        "_line_total.png"                    "Total error over the entire runtime")
        plt.savefig(os.path.join(path, f_name)) plt.clf() plt.close()



def create_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, 8),
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(["Controlled 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("Course 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("Course of the manipulated variable over time")

        axs[2].plot(time, df[SimKeys.Z]) axs[2].set_xlabel("Zeit
        [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("Course of 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()
```