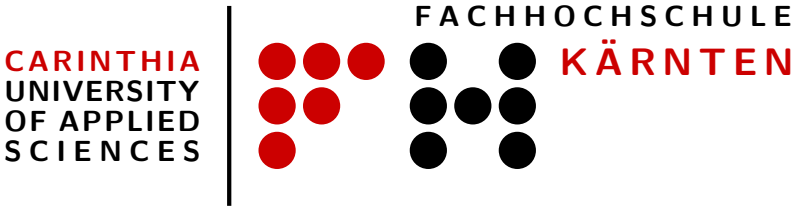

Carinthia University of Applied Sciences

Degree Program: Systems Design

LAB REPORT

"Control Systems Special Topics 1"

Student:	Patrick Hofer
ID:	1410528006
Lecturer:	Dr. Andreja Rojko
Submitted:	November 24, 2015
Submitted:	
Grade:	

Contents

List of Figures

List of Tables

List of Acronyms

ANN Artificial Neural Network

FLS Fuzzy Logic System

GA Genetic Algorithm

MF Membership Function

1 Fuzzy Logic

1.1 Exercise 1: Liquid level tank

The first exercise is to calculate the output of a given Fuzzy Logic System (FLS), which is used to control the liquid level. Table 1.1 shows the two input values and their respective fulfilment rates, which are given.

Table 1.1: Input values and the fulfilment rates

Input	Value	LOW/ negative	OK OK	HIGH/ positive
LEVEL:	-0.2	0.03	0.8	0
RATE:	-0.03	0.07	0.6	0

For the calculation of the output, the product/sum reasoning method and the simplified centre-of-gravity fuzzification method have to be used. Therefore, the fulfilment rate of the different rules for the two input variables has to be found. The following five rules are used in this example:

R^1 : IF LEVEL = OK THEN valve = no change

R^2 : IF LEVEL = LOW THEN valve = open fast

R^3 : IF LEVEL = HIGH THEN valve = close fast

R^4 : IF LEVEL = OK AND RATE = positive THEN valve = slowly close

R^5 : IF LEVEL = OK AND RATE = negative THEN valve = slowly open

Applying the two inputs to the rules results in the following fulfilment rates of the rules:

$$\begin{aligned} \mu_{A^1} &= 0.8 & \mu_{B^1} &= \{\} & \mu_{R^1} &= 0.8 \\ \mu_{A^2} &= 0.03 & \mu_{B^2} &= \{\} & \mu_{R^2} &= 0.03 \\ \mu_{A^3} &= 0 & \mu_{B^3} &= \{\} & \mu_{R^3} &= 0 \\ \mu_{A^4} &= 0.8 & \mu_{B^4} &= 0 & \mu_{R^4} &= 0 \\ \mu_{A^5} &= 0.8 & \mu_{B^5} &= 0.07 & \mu_{R^5} &= 0.056 \end{aligned}$$

The output can be calculated with the following formula for the simplified centre-of-gravity fuzzification method:

$$y = \frac{\sum_{l=1}^{M} \bar{y}_l \cdot \mu_{R^l}}{\sum_{l=1}^{M} \mu_{R^l}}$$

In this formula, $\bar{y}_l$ is the centre point of the according output fuzzy set. The result of this calculation is the output of the FLS:

$$y = \frac{0.8 \cdot 0 + 0.03 \cdot 0.54 + 0 \cdot (-0.54) + 0 \cdot (-0.27) + 0.056 \cdot 0.27}{0.8 + 0.03 + 0 + 0 + 0.056} = 0.0353$$

So the output for the given input values is $y = 0.0353$.

1.2 Exercise 2: Fuzzy PID controller

The task of this exercise is to write a rule base for a fuzzy controller, which adapts the parameters of a PID controller. Inputs of the FLS are the overshoot and the steady-state error and the outputs are the changes of the parameters k_p, k_i and k_d. The main purpose of this FLS is to minimize the overshoot and the steady-state error. Table 1.2 shows the influence of the different parameters on the characteristics of the closed loop system.

Table 1.2: Influence of the PID parameters on the closed loop characteristics

Parameter	Rise time	Overshoot	Settling time	Steady-state error
k_p:	Decrease	Increase	No influence	Decrease
k_i:	Decrease	Increase	Increase	Eliminates
k_d:	No influence	Decrease	Decrease	No influence

The first step is to decide which Membership Function (MF)s are used for the inputs. As both inputs, overshoot and steady-state error, are both non-negative values, the range of the inputs is set from 0 to 1. Furthermore, the inputs are divided by three MFs, which are *zero*, *middle* and *high*. Figure 1.1 shows the graphs with the used MFs.

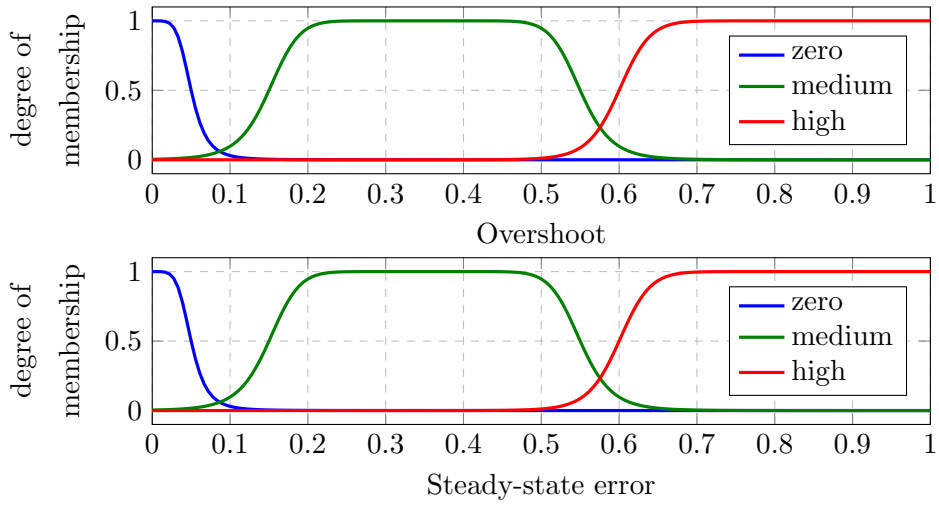

Figure 1.1: MFs for the inputs overshoot (top) and steady-state error (bottom)

The next step is to define the MFs for the output variables. As the change of the controller parameter can either be positive, negative or zero, the range is set from -1 to 1. To keep the system simple, three MFs are used for the output variables, which are *decrease*, *ok* and *increase*. Figure 1.2 shows the actual used MFs for the output variables.

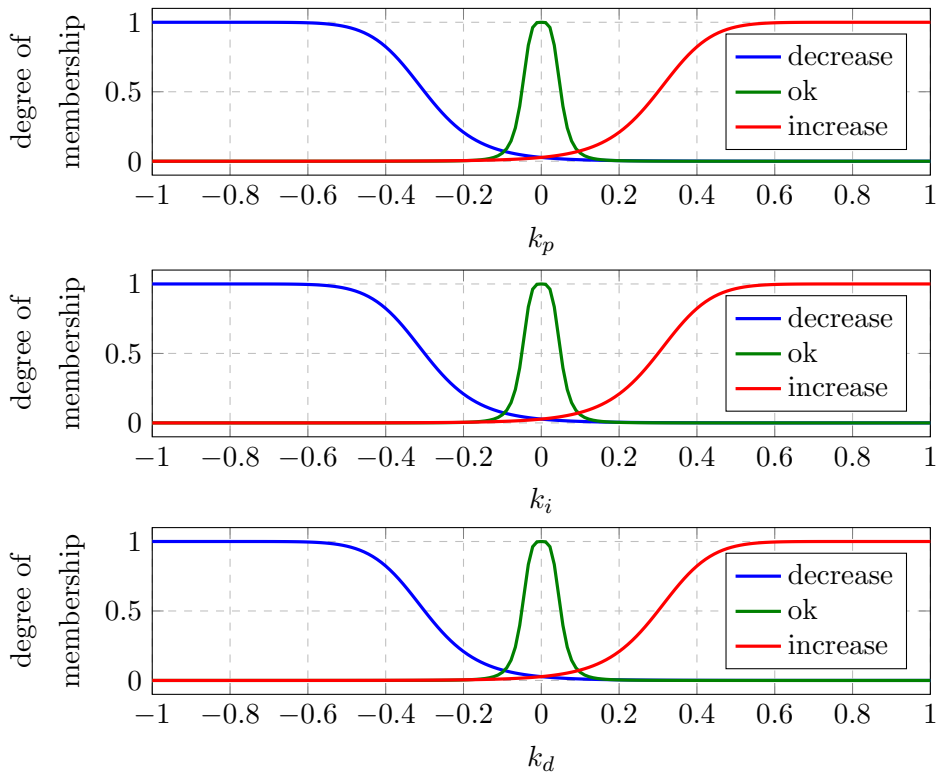

Figure 1.2: MFs for the outputs k_p (top), k_i (mid) and k_d (bottom)

Using the information of Table 1.2 and the previously mentioned MFs, the rules of the FLS can be set up.

R^1 : IF Overshoot = zero & Steady-state error = zero THEN k_p = ok & k_i = ok & k_d = ok

R^2 : IF Overshoot = middle THEN k_p = decrease & k_i = ok & k_d = ok

R^3 : IF Overshoot = high THEN k_p = decrease & k_i = decrease & k_d = increase

R^4 : IF steady-state error = middle THEN k_p = ok & k_i = increase & k_d = ok

R^5 : IF steady-state error = high THEN k_p = increase & k_i = increase & k_d = ok

This rulebase is kept very simple, and therefore contains only five rules. The main idea is, if one of the inputs is greater than zero, then reduce the controller parameters which cause an increase of the input and increase the parameters which reduce the input. And finally, if both inputs are ok, than the FLS does not change the controller parameters.
Figure 1.3 shows the three output variables in dependency of the two inputs. The plot for k_p shows, that this parameter is only increased, if the steady-state error is high and the overshoot is close to zero. Otherwise it is always decreased. The value of k_i on the other hand, is only decreased, if the overshoot is very high and the steady-state error is zero. Finally, the parameter k_d is increased if the overshoot is high but is never decreased.

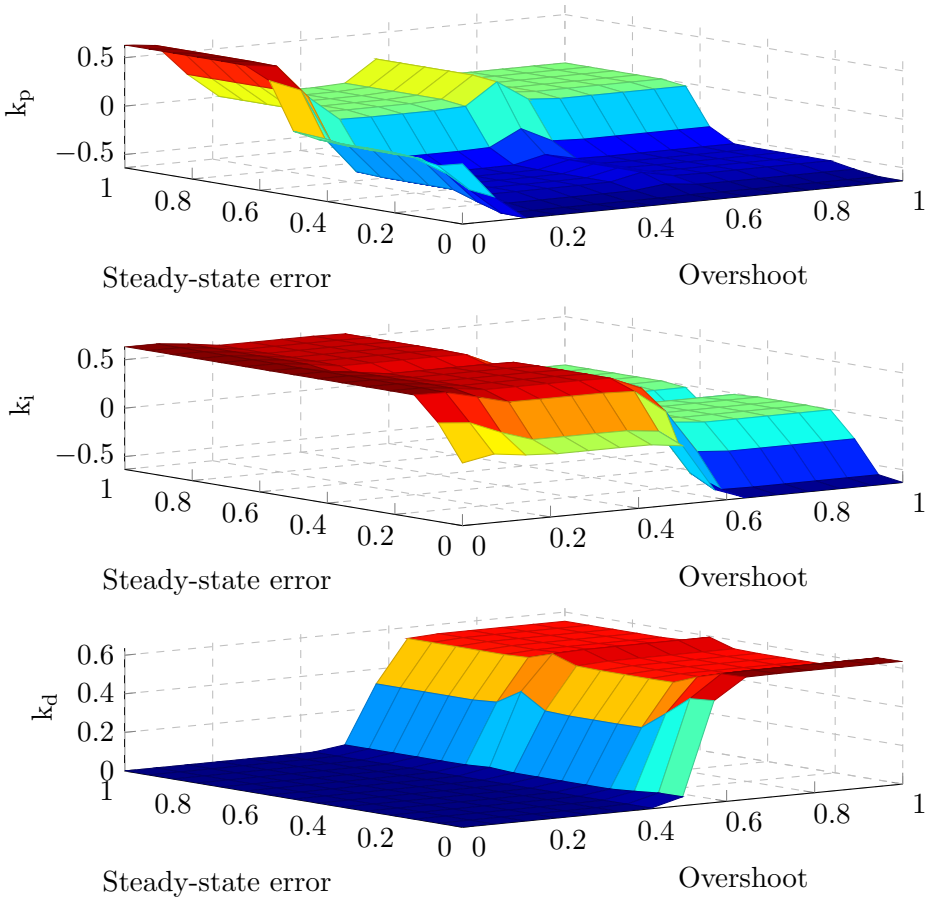

Figure 1.3: Surface of the three output variables k_p (top), k_i (mid) and k_d (bottom)

1.3 Exercise 4: Fuzzy control of an inverted pendulum

This task is to control an inverted pendulum with a FLS. The system, as shown in Figure 1.4, consists of a mass which is connected to a motor via a rod. Goal of the controller is to bring the system from the stable, downward equilibrium point to the unstable, upright equilibrium and stabilize the mass there.

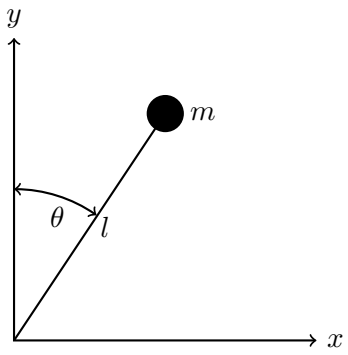

Figure 1.4: System schematic of the inverted pendulum

Although the design of a FLS controller is not model based, the model is still being derived to

be able to simulate the performance of the control algorithm. The parameters of the system are summarised in Table 1.3. The mass of the rod is considered to be much smaller than the mass and can therefore be neglected.

Table 1.3: Parameters of the given system

Parameter	Value	Unit
m:	1	kg
l:	1	kg
B:	0.4	Nms

For the derivation of the model, the Lagrange formalism is used. As the system has only a single degree of freedom, only one generalised coordinate is needed. The angle of the rod θ is used as the generalised coordinate. The next step is to find the kinetic and the potential energy of the system. Therefore, the position vector of the mass and its time derivative are introduced:

$$\mathbf{r} = \begin{pmatrix} l\sin(\theta) \\ l\cos(\theta) \end{pmatrix} \qquad \dot{\mathbf{r}} = \begin{pmatrix} l\cos(\theta)\dot{\theta} \\ -l\sin(\theta)\dot{\theta} \end{pmatrix}$$

As the movement of the system is only in the $x-y$ plane, the z component is always zero and is therefore not mentioned in the position vectors. For the kinetic energy T, the velocity vector is inserted into following equation:

$$T = \frac{m}{2}\dot{\mathbf{r}}^T\dot{\mathbf{r}} = \frac{ml^2}{2}\dot{\theta}^2$$

The potential energy of the system is given by the height of the mass. The potential has its maximum at the unstable equilibrium point, which is defined as $\theta = 0$ and its minimum at the stable equilibrium $\theta = \pi$. The potential energy V is described by the product of the height, which is the y component of the position vector, and the gravitation force:

$$V = mg\mathbf{r}^T\mathbf{e}_y = mgl\cos(\theta)$$

Out of the kinetic energy T and the potential energy V, the Lagrangian L can be calculated:

$$L = T - V = \frac{ml^2}{2}\dot{\theta}^2 - mgl\cos(\theta) \tag{1.1}$$

The formalism of Lagrange is defined as:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} = Q_k \tag{1.2}$$

where:

q_k : k$^{\text{th}}$ generalised coordinate
$\dot{q}_k$: k$^{\text{th}}$ generalised velocity
Q_k : k$^{\text{th}}$ generalised force

The generalised force Q_k for this system contains the input torque τ and the viscous friction:

$$Q = \tau - B\dot{\theta} \tag{1.3}$$

Inserting 1.1 and 1.3 into 1.2 results in the complete equation of motion of the given system:

$$ml^2\ddot{\theta} - mgl\sin(\theta) = \tau - B\dot{\theta}$$

Now that the model description is obtained, the model has to be verified. Therefore, some simulations with different initial conditions are done, where the behaviour of the system is pre-known (e.g. a system in an equilibrium point stays in the equilibrium point if there is no influence). The following four sets of initial conditions $\mathbf{x}_0$ and input torques τ are used for the verification of the model:

$$\mathbf{x}_{0,1} = \begin{pmatrix} \theta = 0 \\ \dot{\theta} = 0 \end{pmatrix} \quad \mathbf{x}_{0,2} = \begin{pmatrix} \theta = \pi \\ \dot{\theta} = 0 \end{pmatrix} \quad \mathbf{x}_{0,3} = \begin{pmatrix} \theta = 0.1 \\ \dot{\theta} = 0 \end{pmatrix} \quad \mathbf{x}_{0,4} = \begin{pmatrix} \theta = \pi \\ \dot{\theta} = 0 \end{pmatrix}$$
$$\tau_1 = 0 \qquad \tau_2 = 0 \qquad \tau_3 = 0 \qquad \tau_4 = 8$$

These sets contain both equilibria, one initial condition close to the unstable equilibrium point and one set with an input torque. The results of the according simulations are shown in Figure 1.5. The upper left and the upper right figures show the system response at the two equilibrium states. This shows, that without any disturbances, the system stays in the equilibria. The lower left figure shows the behaviour of the system, close to the unstable, upper equilibrium point. The mass immediately starts to fall down, and than oscillates around the stable equilibrium point. The final figure shows the system response to an input torque. After the steady state is reached, the velocity of the system is constant and therefore the mass keeps turning. As all of these responses were expected, the model behaviour is assumed to be correct and it can be used for the verification of the controller.

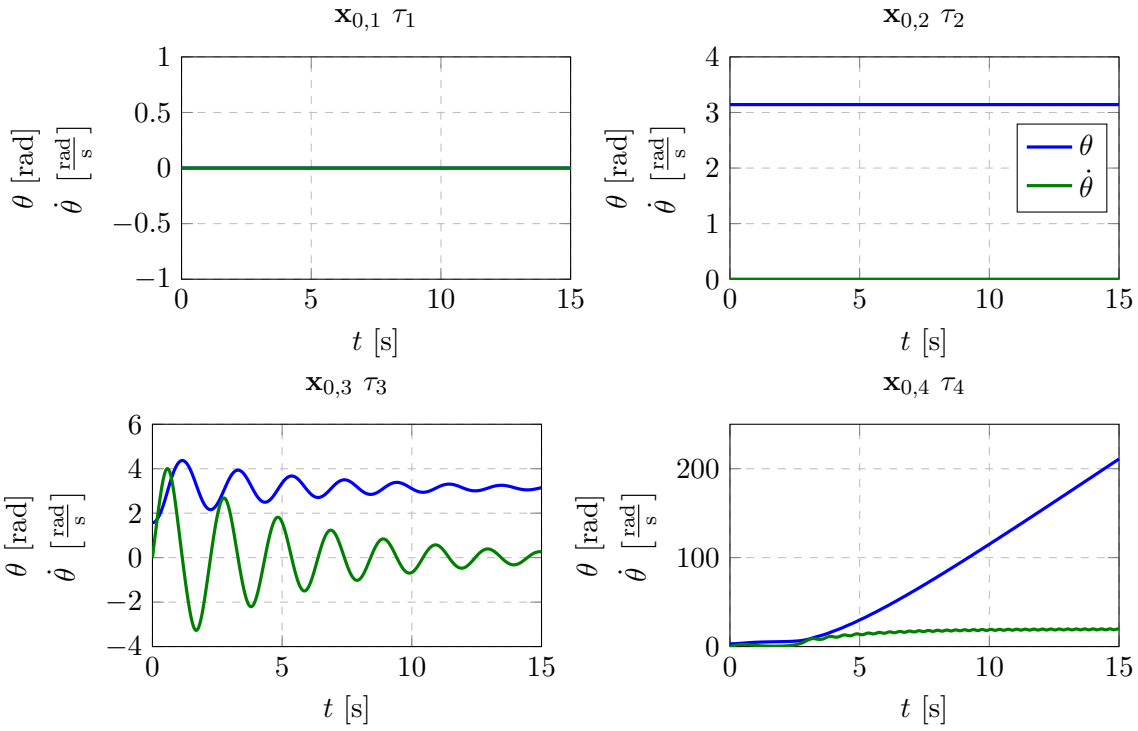

Figure 1.5: Simulations for the verification of the model

The next step is to design a FLS which swings the system from the stable, downright equilibrium point to the unstable, upper one and stabilises it there. As inputs for the fuzzy controller, the angle θ and the angular velocity $\dot{\theta}$ are used. The output is the motor torque τ. Figure 1.6 shows

the MFs for the input variables. The angle θ is divided into five different categories, which are, *BN*, *N*, *zero*, *P* and *BP*. The angle θ ranges from $-\pi$ to π, which is therefore chosen as the range for the input variable. The angular velocity $\dot{\theta}$ is only divided into three groups, which are *N*, *zero* and *P*. As only the sign is considered, the input range is defined from -1 to 1. The types of the MFs are set to *bell* shape, such that the crossover between the different MFs is smoother.

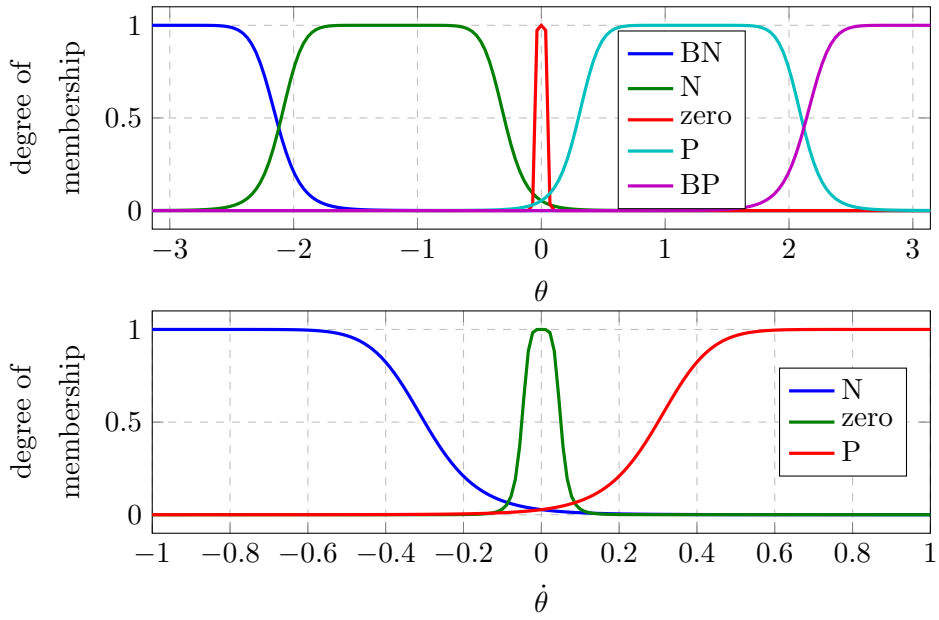

Figure 1.6: MFs for the inputs θ and $\dot{\theta}$ of the fuzzy controller

Furthermore, the MFs for the output variable τ need to be defined. In order to overcome the gravitation force of the mass, τ has to be bigger than 8 Nm. To create a smoother upswing, the range of the output torque is set to $|\tau| \leq 30$. This range is also divided into five MFs, which also are labelled as *BN*, *N*, *zero*, *P* and *BP*. Again, *bell* shaped MFs are used for the output MFs.

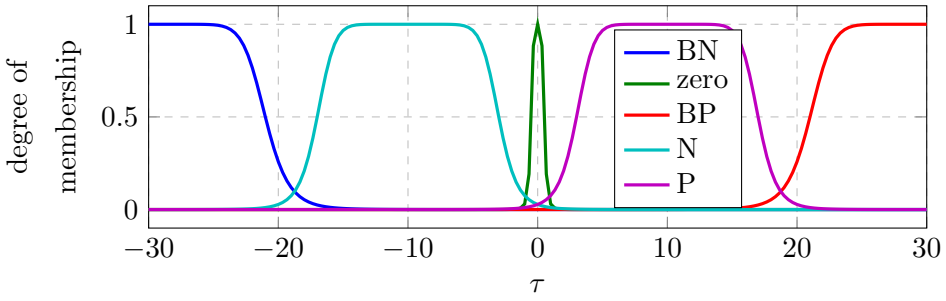

Figure 1.7: MFs for the output τ of the fuzzy controller

Setting up the ruleset for the fuzzy controller can be done intuitive. As long as the rod stays in the upper equilibrium point, no output torque is needed. If the mass is far away from the equilibrium point, the output torque has to be high and the direction has to point towards the equilibrium point. The closer the angle gets to the unstable equilibria, the lower the output torque has to be, in order to create smaller overshoot.

The following ruleset is used for the implementation of the controller:

R^1 : IF θ = zero THEN $\tau = 0$

R^2 : IF θ = BN & $\dot{\theta}$ = N THEN τ = BP

R^3 : IF θ = BP & $\dot{\theta}$ = P THEN τ = BN

R^4 : IF θ = N & $\dot{\theta}$ = N THEN τ = P

R^5 : IF θ = P & $\dot{\theta}$ = P THEN τ = N

R^6 : IF θ = N THEN τ = P

R^7 : IF θ = P THEN τ = N

With the MFs for the input and output variables and the ruleset, the fuzzy controller can be implemented. Figure 1.8 shows the surface of the output variable in dependency of the two inputs. If both inputs are at their lowest value, the output torque τ is at approximately 25 Nm and if the inputs are at their maximum, the output torque is -25 Nm.

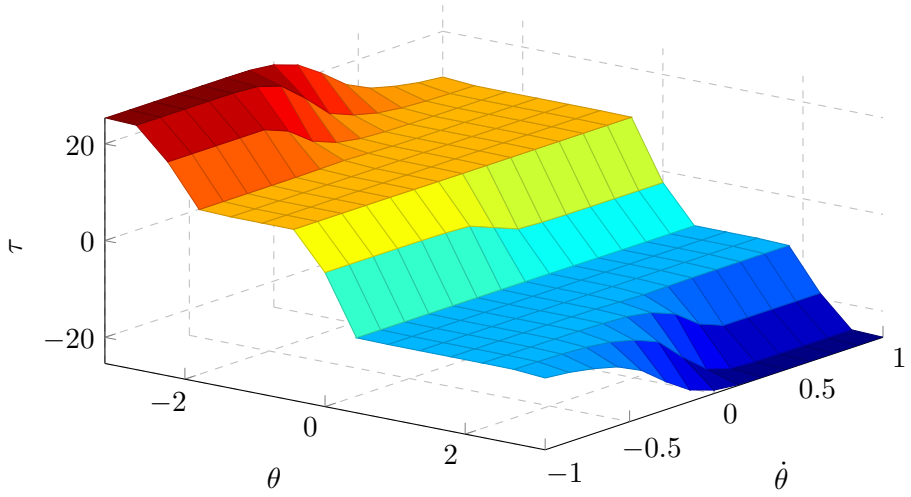

Figure 1.8: Surface of the output variable τ

The simulation of the control behaviour is done in Simulink. Figure 1.9 shows the Simulink schematic, which implements the control loop. The system, called *Pendulum*, has three inputs which are the control torque τ, the parameters of the model and the initial condition $\mathbf{x}_0$. To ensure, that the angle θ is limited to $-\pi \leq \theta \leq \pi$, a modulo division by 2π is included after the system.

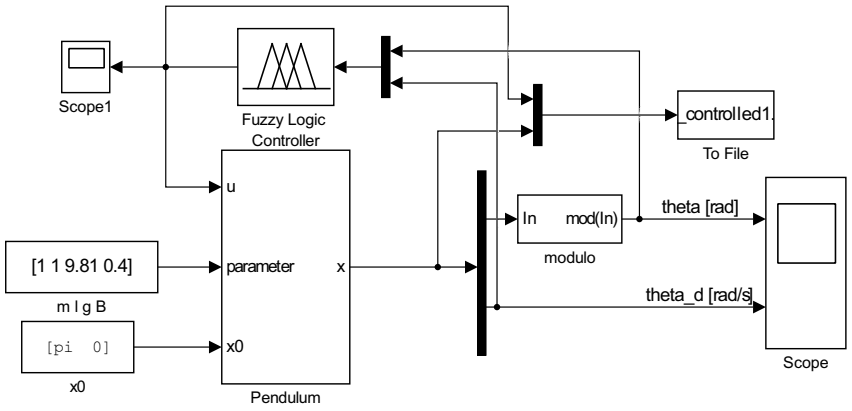

Figure 1.9: Simulink schematic for the simulation of the fuzzy controller

The final step is to simulate the behaviour of the closed loop system. Figure 1.10 shows the angle θ, the angular velocity $\dot{\theta}$ and the according control torque τ for a simulation with an initial condition at the stable, downward equilibrium point. After approximately 1.2 s, the mass first crosses the unstable equilibrium point and has an overshoot of roughly 70 %. After about 8 s the system has reached the upper equilibrium and is settled. The output torque lies between from 15 Nm to −15 Nm until approximately 8 s and then converges slowly towards 0 Nm.

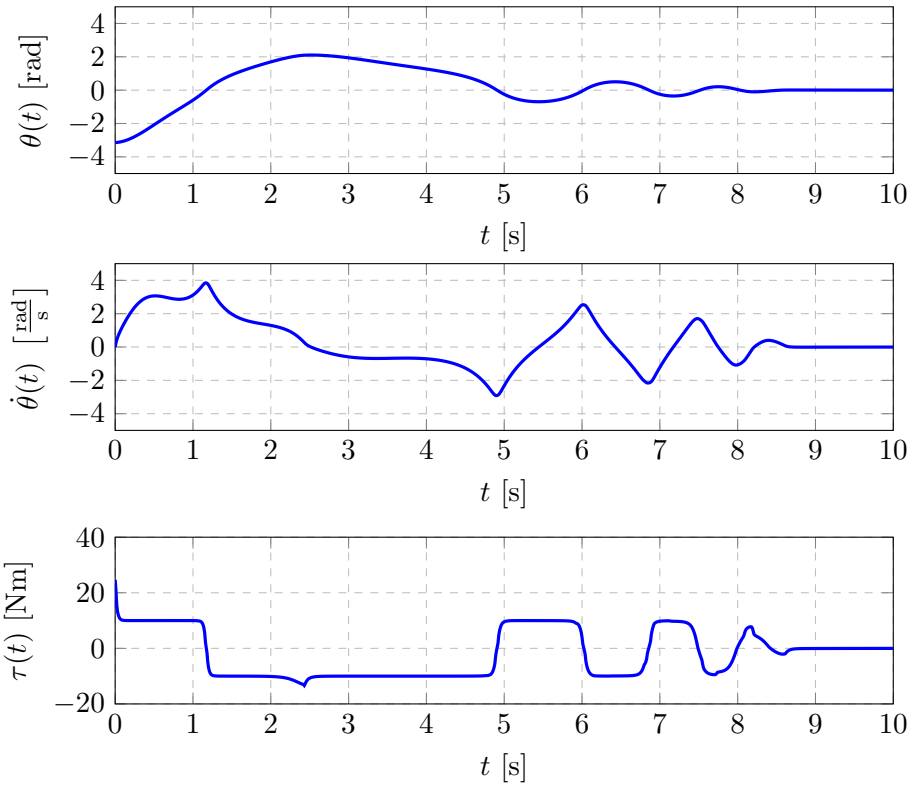

Figure 1.10: Response of the controlled system when starting at the lower equilibrium

2 Genetic Algorithm

2.1 Exercise 1: Autonomous vehicle

Four autonomous vehicles are put in a warehouse with fixed obstacles and known geometries. The starting point of the vehicles, which is called *depot* is fixed. Each vehicle has the same, constant velocity. Placed inside the warehouse are so called *workstations*, which also are fixed in position. The task of the vehicles is to visit every workstation at least once and then return to the depot. To find the fastest solution to this problem, a Genetic Algorithm (GA) has to be designed. Therefore, a suitable coding strategy, a fitness function, crossover, mutation and selection strategy have to be defined.
First, the coding scheme for the definition of the chromosomes and the genes needs to be applied to the problem. For this approach, one gene is set to be a single workspace. As every workspace needs to be visited only once, a total of 16 genes would be needed to include every workspace. As it is possible, that the optimal solution is not that every vehicle travels to four workspaces, but maybe one drives to five and another only to three, four additional slots including 0 are added. This 0 then represents the return to the depot and the beginning of the next vehicle. This makes a total of twenty genes in a single chromosome. So it is possible to enable a varying number of workstations, that are visited by a single vehicle. A possible chromosome is shown in Figure 2.1.

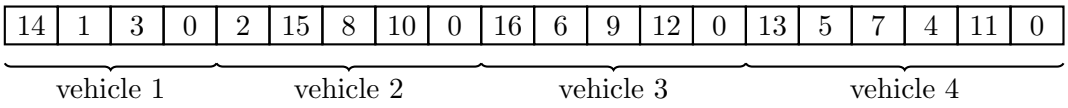

Figure 2.1: Example of a possible chromosome

The next part that is needed, is the fitness function to decide which chromosome has the best potential to be close to the optimal solution. The goal is to minimize the time until every workstation has been served and every vehicle has returned to the depot. If $x_{i,j}$ denotes the length of the path the i^{th} vehicle in the j^{th} chromosome has to drive, than the time of the solution is given by:

$$t_j = \max_i \left(\frac{x_{i,j}}{v} \right)$$

where:

t_j : time to serve all workspaces with j^{th} chromosome
v : velocity of the cars

Ranking all times of the possible solutions, the fastest equals the fittest. So the smaller t_j is, the better is the solution.
In order to find the optimal solution to a problem with a GA, the current solutions, called *population*, have to generate the next population. This process is called crossover. Two solutions of the current population are used to generate a number of new solutions. For the given task, a standard crossover, where parts of the two used solutions are exchanged, is not possible as it would most likely lead to a solution where one workstation is not served at all. A possible crossover strategy would be, to take a part of one of the solutions and take the missing workstations out of the second parent solution. Another possibility would be, to combine two solutions

and then to check for missing and double workstations and change the new chromosome to include every workstation only once.
Just as in real evolution, sometimes mutations occur in a GA. This can be done by changing the values of different genes, by shifting chromosomes, etc. For the given task, two mutation strategies can be implemented. The first, is to choose a random gene in a chromosome and change its location with a different, randomly chosen workstation. The other possibility would be, to take the segment which represents a single vehicle, except for the 0 at the end and shift the genes to the left or to the right. One example of those two mutation strategies is shown in Figure 2.2. The upper part shows the exchange of two genes and the lower part shows the right shift of the genes of vehicle 3.

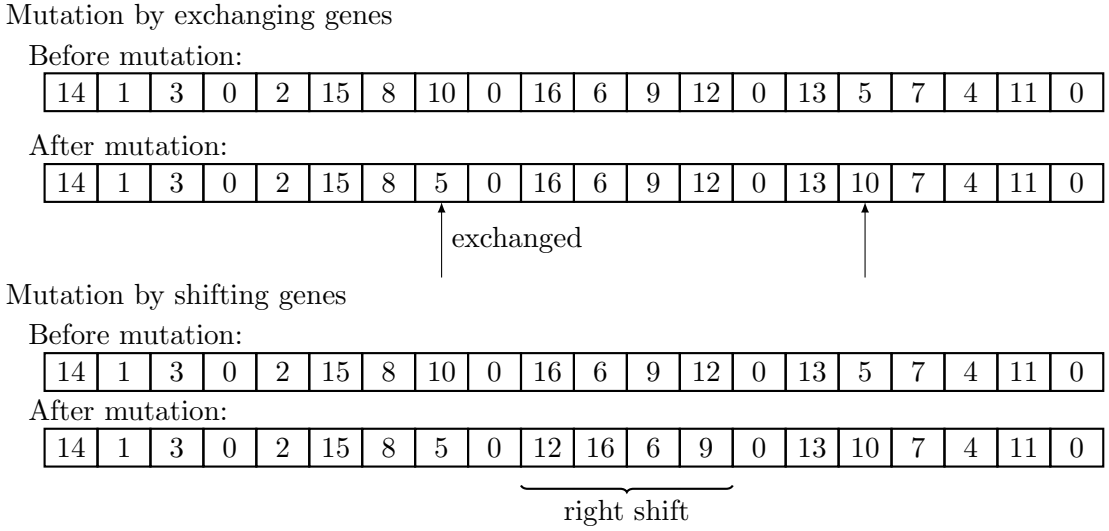

Figure 2.2: Examples of mutation strategies, above the exchange of genes, below a right shift of the genes of vehicle 3

The last part which is needed for the GA is the selection strategy for the selection of the parents of the new generation and which individuals are passed on to the next cycle. An approach, where the amount of solutions is constant is chosen. This means, the parent generation creates a number of new solutions and then, the same amount of solutions, as before, is chosen, according to the ranking of the solutions. So if the initial generation has for example 20 individuals, then each pair creates 2 children, resulting in 20 new solutions. Out of these 40, the 20^{th} best solutions are selected and fed to the next generation. So, the best individuals are always part of the new generation and the calculation time is not increased with each generation step.

3 Artificial Neural Network

3.1 Exercise 1: Basic artificial neural network

This exercise is about a basic two layer Artificial Neural Network (ANN) with two inputs and one output. The first task is to fully label the graph of such a ANN. This graph can be seen in Figure 3.1. Not all weights between the input layer and the hidden layer are labelled to keep the graph readable. The weight $w_{J,ij}$ shows the general form of the nomenclature, while the other three serve as examples.

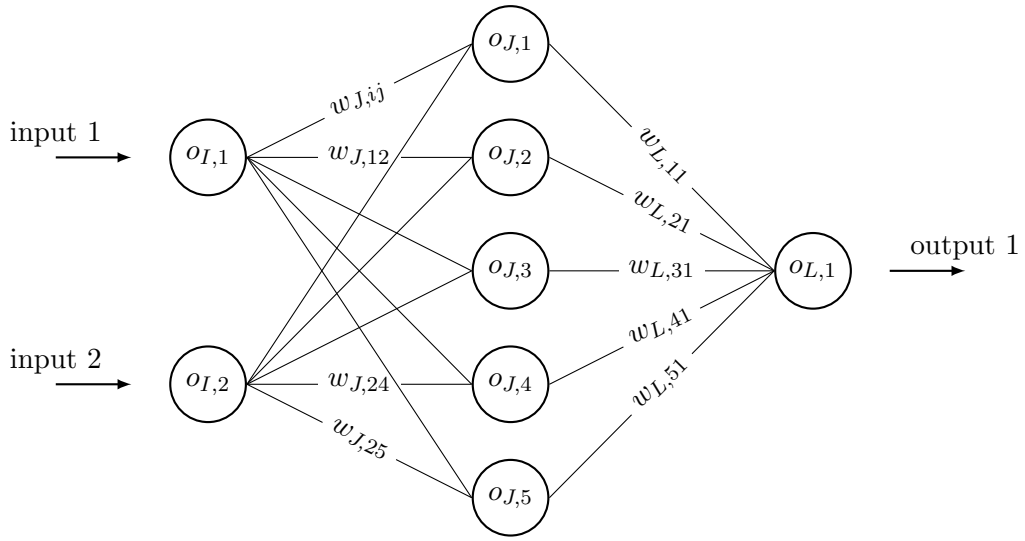

Figure 3.1: Two layer ANN with two inputs, five neurons in the hidden layer and one output

The second task is to calculate the output of this ANN with the following specifications:

1. both layers have linear activation functions,
2. input 1 has value 2,
3. input 2 has value -4,
4. $w_{J,1j} = 1$; $j = 1..5$,
5. $w_{J,1j} = 2$; $j = 1..5$,
6. All weights in output layer have value -0.5, except $w_{L,41} = 10$.

For the calculation of the output, a Matlab function has to be written. For the calculation, the weights are collected to a matrix with one row per input and one column per neuron in the hidden layer. The following matrix and vector are obtained:

$$\mathbf{W}_J = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix} \qquad \mathbf{w}_L = \begin{pmatrix} -0.5 & -0.5 & -0.5 & 10 & -0.5 \end{pmatrix}^T$$

The Matlab function for the calculation of the output is realised with three inputs: the actual inputs of the ANN and the two weighting matrices. The single output is a vector containing the outputs of the output layer. Listing 3.1 shows the code of the created function. First, the function checks if the dimensions of the different inputs matches with the other inputs. After these checks, the output is calculated with two nested for loops. The outer loop is used to

calculate the output of the different neurons, while the inner loop calculates the output of a single neuron.

```
function out = calc_ann_output( in, WJ, WL )
%CALC_ANN_OUTPUT calculates output of two layer ANN
%   Calculates the output of a two layer artificial neural network
%   Input arguments are a vector with the inputs, a matrix with the weights
%   from the hidden layer and a matrix with the weigths of the output layer
n = length(in);
if n ~= size(WJ,1) && n ~= size(WJ,2)
    error('Number of inputs does not match the number of weights of the hidden layer');
end
if n == size(WJ,2)
    WJ = WJ.';
end
m = size(WJ, 2);
if m ~= size(WL,1) && m ~= size(WL,2)
    error('Number of hidden layer neurons does not math the number of weigths of the output layer');
end
if m == size(WL,2)
    WL = WL.';
end
l = size(WL, 2);

oJ = zeros(m,1);
for a = 1:m
    for b = 1:n
        oJ(a) = oJ(a) + (WJ(b,a)*in(b));
    end
end
out = zeros(l,1);
for a = 1:l
    for b = 1:m
        out(a) = out(a) + (WL(b,a)* oJ(b));
    end
end
end
```

Listing 3.1: Matlab function for the calculation of the output of an ANN

To test the functionality of this script the given vales of the example are used. This results in an output of the ANN of $o_L = -48$.

3.2 Exercise 2: Artificial neural network as universal approximator

The task of this exercise is to approximate the following function with an ANN:

$$y_c(i) = 200 \cdot \left(\frac{1}{(i-10)^2+1}\right)\left(\sin\left(\frac{i-10}{2}\right)\cos(i-10)\right) \quad i = [0, 20]$$

For the design process, the Matlab ANN toolbox is used. This toolbox is used to create a two layer ANN with a sigmoid activation function in the hidden layer and a linear activation function in the output layer. The designed ANN can also be trained to input/output pairs and can then be exported as a Matlab function or a Simulink schematic.
For the approximation of the function y_c, 25 neurons are used in the hidden layer. As trainings algorithm, the *Bayesian Regularization* was used, as the best results were received with this algorithm. Figure 3.2 shows the realisation of the hidden layer and the output layer in Simulink. Furthermore, this figure shows the two activation functions: the sigmoid function for the hidden layer and the linear activation function for the output layer.

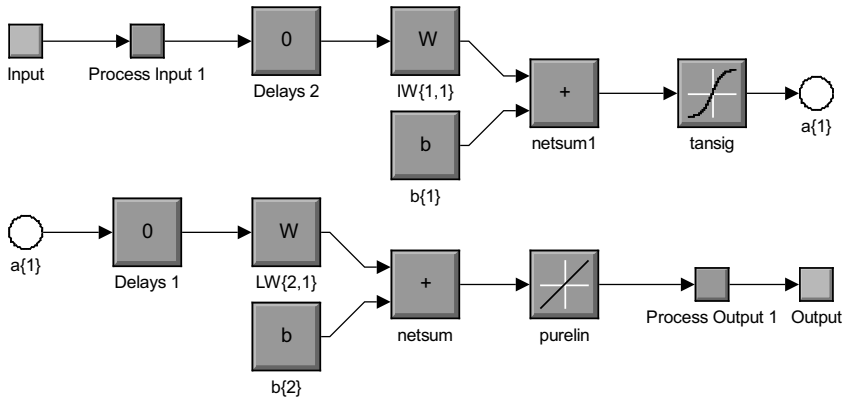

Figure 3.2: Simulink schematic which shows the two layers of the artificial neural network

Figure 3.3 shows the original signal and the output of the ANN. It can be seen, that there is almost no difference between the two curves.

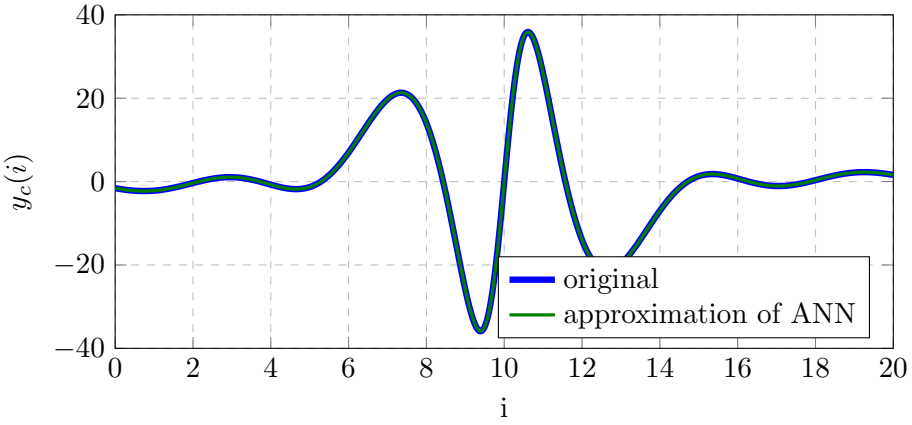

Figure 3.3: Comparison of the original function y_c and the output of the artificial neural network

To show the actual difference between the two signals, Figure 3.4 shows the error signal. It can be seen, that the error is in the range $\pm 10^{-4}$. Compared to the amplitude of the original signal,

which is at approximately ±30 at the maximum/minimum, the relative error is almost zero. So the output of the ANN can be seen as a useful approximation of the original signal y_c.

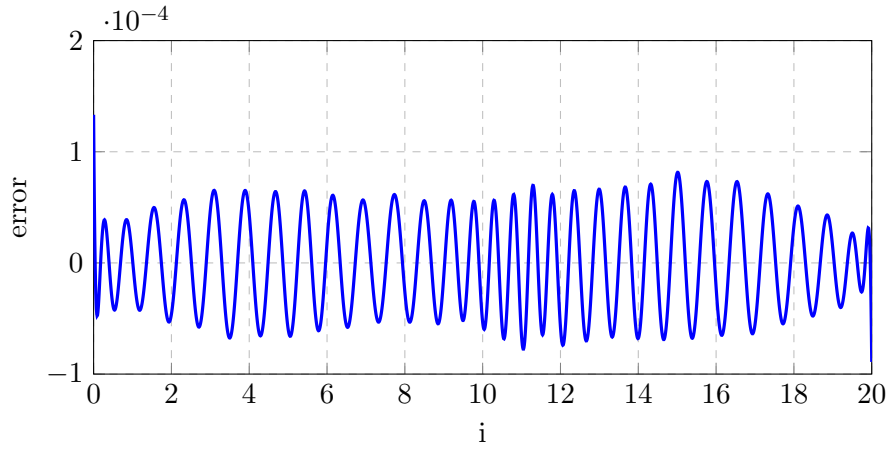

Figure 3.4: Error of the neural network compared to the original signal

The main advantage of the ANN toolbox of Matlab is, that the design and the learning process of a two layer neural network is very easy. All that has to be done, is to define the input, the desired output, the number of neurons in the hidden layer and the learning algorithm and then the process is finished and can be exported as Matlab function or as Simulink schematic. The problem is, that the received code is very complicated and untransparent, making it difficult to change or adept anything after the design process.

4 Continuous sliding mode neural network controller

This chapter describes the design and the simulation of a continuous sliding mode neural network controller for a nonlinear system with partly unknown parameters.

4.1 System description, dynamic model and model verification

The given system, as shown in Figure 4.1, consists of a motor which rotates a cylinder which is connected to a spring. The angular position of the cylinder is denoted by θ, the according angular velocity with $\dot{\theta}$ and the angular acceleration with $\ddot{\theta}$. When the motor starts rotating, the cylinder moves as well, also extending the spring. The spring reaches its maximum extension at $\pm\pi$.

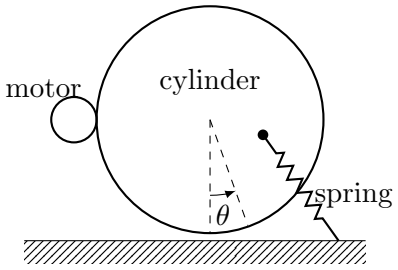

Figure 4.1: Schematic of the given system

As the system has only one degree of freedom, it can be described by the following nonlinear differential equation of second order:

$$\tau = \mathbf{M}(\theta)\ddot{\theta} + \tau_c(\theta, \dot{\theta}) + \tau_g(\theta) + \tau_{fr}(\dot{\theta}) + \tau_d(\theta, \dot{\theta})$$

where:

τ : input torque generated by the motor
$\mathbf{M}$: matrix of moment of inertia
τ_c : torques generated by coreolis/centripedal/centrifugal force
τ_g : torques caused by gravitation force
τ_{fr} : viscous friction
τ_d : disturbances to the system

For the simulation of the controller, the dynamic model of the system is given:

$$\tau = \frac{mr_c^2}{2}\ddot{\theta} + rk\left(\sqrt{(r_c+l)^2 + r^2 - 2r(r_c+l)\cos(\theta)} - l\right) \cdot \frac{(r_c+l)\sin(\theta)}{d} + B\dot{\theta}$$

where:

m : mass of the cylinder
r_c : radius of the cylinder
r : distance from cylinder axis to spring fastening
k : spring constant
l : length of the spring, when $\theta = 0$
d : actual length of the spring

For the simulation of the system dynamics, a Simulink schematic, which implements the differential equations, is given. To check the plausibility of the dynamic model, some simulations with different initial conditions and input torques are executed. The first simulation is with the initial condition set to the stable equilibrium point, where the spring is not extended. The second simulation begins at the unstable equilibrium point, with a maximum extension of the spring. And also, these simulations are repeated with no viscous friction and a longer simulation time. A final simulation, with the initial condition set to the stable equilibrium point is done with a constant input torque τ. The results of these simulations are shown in Figure 4.2, 4.3 and 4.4. The first figure shows the simulation of the two equilibria with viscous friction. On the left hand side of the figure, the stable equilibrium point is shown. According to the expectation, the system does not move at all. On the right side, the unstable equilibrium point is shown. Although the angle θ and the extension of the spring Δx do not change, the angular velocity $\dot{\theta}$ shows a falling behaviour. The values next to the plot are 0 because the actual value is around 10^{-13}. This small values are caused by numeric errors of Matlab.

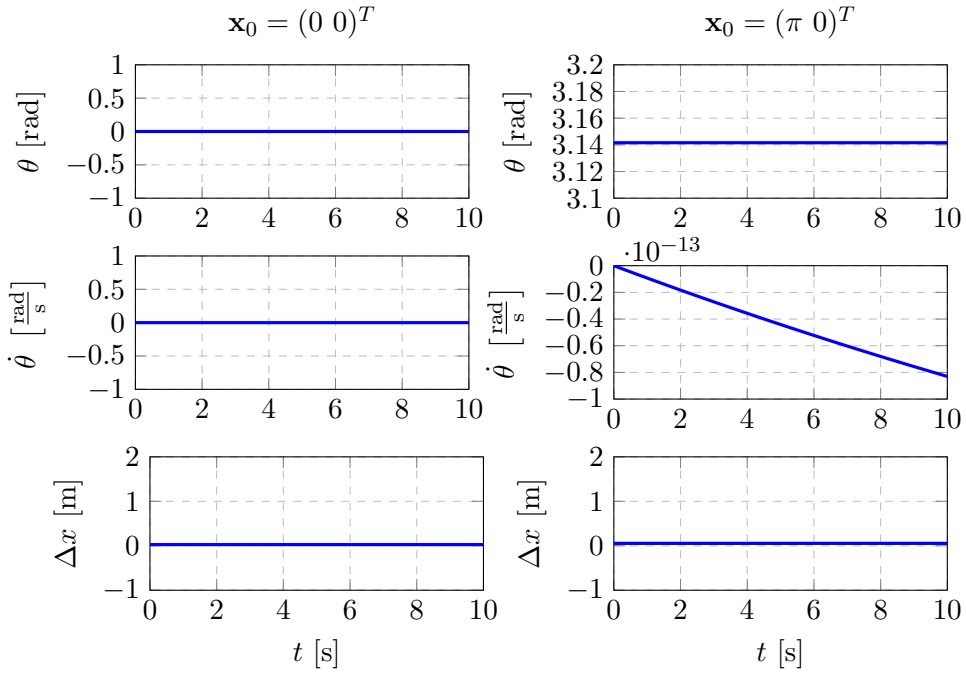

Figure 4.2: Simulation of the system dynamics with two different initial conditions

Figure 4.3 shows the simulation of the two equilibria with no viscous friction and a simulation time of 100 s. On the left side is also the stable equilibrium point, which, again, shows no movement. On the other hand, on the right side, the unstable equilibrium point shows some unwanted behaviour. After approximately 50 s, the cylinder starts rotating. The problem is the numerical error, which was shown in the previous simulation. As the integrator keeps integrating the error, it increases to the point, where the cylinder starts moving. As the viscous friction was removed for this simulation, no energy is removed the system, so the cylinder keeps spinning. This results is only valid in simulation, as the real system will always have some kind of friction and therefore, the real system will only move to the unstable equilibrium point and after some time, it will remain there.

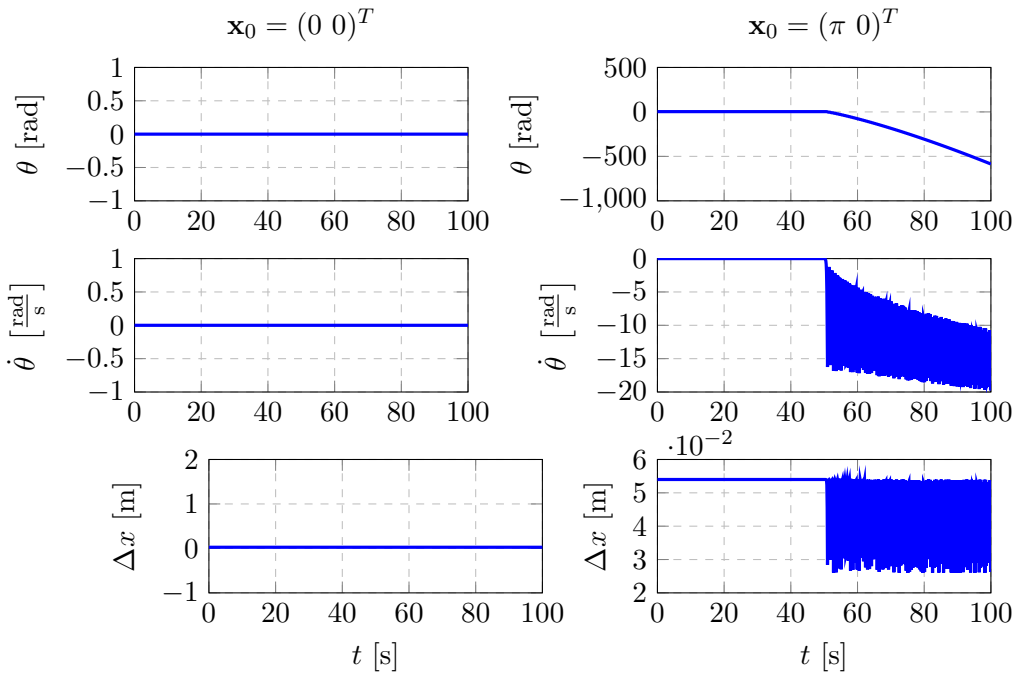

Figure 4.3: Simulation of the system dynamics with two different initial conditions, no viscous friction and longer simulation time

Figure 4.4 shows the simulation results with the constant input torque $\tau = 0.001$ Nm. The cylinder starts to rotate and reaches its maximum deflection at approximately $\theta = 0.8$ rad. Afterwards, the cylinder starts oscillating because of the spring. If the input torque is above 0.003 Nm, the cylinder does not oscillate but starts to move into a single direction.

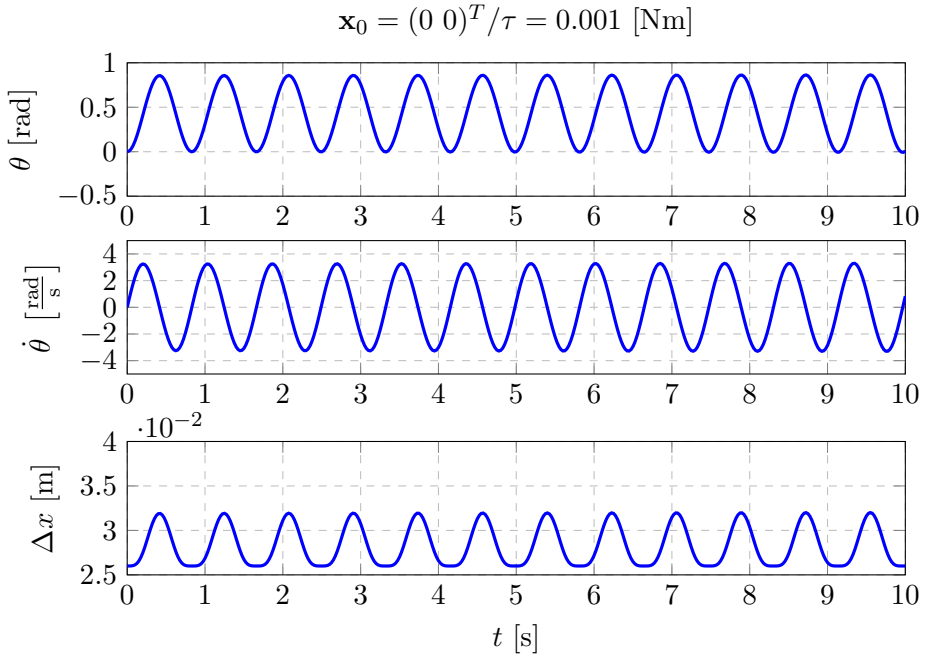

Figure 4.4: Simulation of the system dynamics with two different initial conditions, no viscous friction and longer simulation time

4.2 Velocity profile

The next part, which is needed for the simulation of the system, is a so called velocity profile. A velocity profile is used as trajectory, containing position, velocity and acceleration, which the controlled system uses as reference values. There are a lot of different velocity profiles, for example the trapezoidal, the exponential or the constant velocity profiles. For the given system, the so called $\sin^2$ profile has already been prepared. The advantages of this profiles are the smooth acceleration and deceleration and on the other hand, less jerk is generated. A disadvantage is, that a higher maximum acceleration is needed, compared to other profiles [1]. As already mentioned, a Simulink schematic, which calculates a $\sin^2$ profile, is given. Figure 4.5 shows two generated references with different path lengths, maximum velocities and maximum accelerations. On the left side of the figure, the end point of the movement is at π rad. At the beginning, a small peak occurs at the acceleration until the velocity reaches its maximum value of 1 $\frac{rad}{s}$. At the end, a negative acceleration peak reduces the velocity back to zero. On the right side of the figure, a shorter path with a higher maximum velocity and acceleration, is shown. The acceleration takes a much longer time, than compared to the first path. Therefore, the velocity is only for a short duration at its maximum.

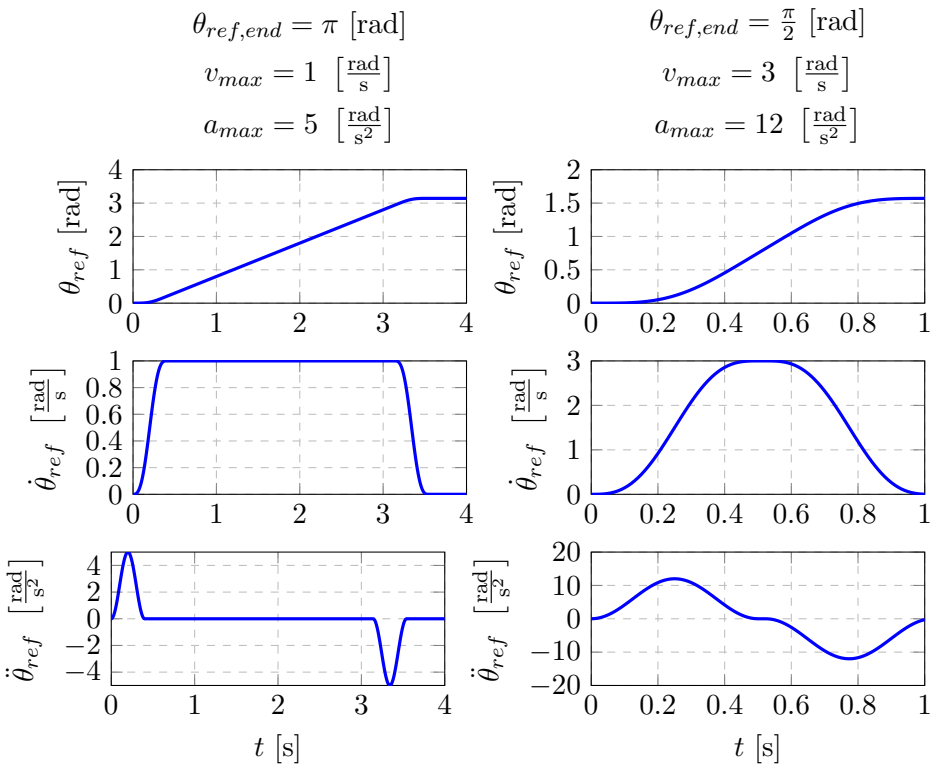

Figure 4.5: Two different sets of reference values, generated by $\sin^2$ profile

The given concept only calculates trajectories for positive movement directions. Even if the ending position is below the starting position, the algorithm computes the absolute values of both positions. To enable a movement into the negative direction, a small adaptation has to be made. As a first step, the sign of the difference between the initial position and the ending position is calculated and stored. For the negative movement, this sign is multiplied to the output of the function.

```
function [output_sin] = reference_sin2_calculation(input)

t = input(1);
qp0 = input(2);
qpf = input(3);
vmax = input(4);
amax = input(5);

sign_qp = sign(qpf - qp0); % Storage of sign of movement

... % Here all needed calculations are done

output_sin(1) = rs * sign_qp; % Output with sign, for negative movements
output_sin(2) = rsi * sign_qp;
output_sin(3) = rsii * sign_qp;
```

Listing 4.1: Needed adaptation of the $\sin^2$ profile calculation for negative moving direction

4.3 Neural network and controller design

As already stated in the beginning, a sliding mode controller with a neural network, for the adaptation of the unknown system parameters, will be designed for the control purpose of the given system. As controller, a sliding mode controller is used. As first step, the wanted sliding surface σ needs to be defined:

$$\sigma(\mathbf{x}) = \mathbf{G}(\mathbf{x} - \mathbf{x}_r) = \begin{pmatrix} k_p & k_v \end{pmatrix} \begin{pmatrix} e \\ \dot{e} \end{pmatrix}$$

where:

$\mathbf{G}$: weighting matrix containing k_p and k_v
k_p : position error gain
k_v : velocity error gain
$\mathbf{x}$: state vector
$\mathbf{x}_r$: reference vector
e : position error
$\dot{e}$: velocity error

The next step is to find a Lyapunov function to prove stability of the closed loop system. The following function is used as a Lyapunov function candidate:

$$V(\mathbf{x}) = \frac{1}{2}\sigma^T\sigma \tag{4.1}$$

Deriving equation 4.1 leads to:

$$\dot{V}(\mathbf{x}) = \sigma^T\dot{\sigma} \tag{4.2}$$

As we cannot prove Equation 4.2 to be negative definite, this derivative is set to a wanted function, which is negative definite. Afterwards the input of the system u is calculated in a way, such that the derivative of the Lyapunov function has the wanted form.

First, a negative definite, scalar function has to be chosen:

$$\dot{V}(\mathbf{x})_{wanted} = -\sigma^T \mathbf{D}\sigma \tag{4.3}$$

where:

$\mathbf{D}$: diagonal matrix which is positive definite

Setting the Equation 4.2 and 4.3 equal, results in the following equation:

$$\sigma^T \dot{\sigma} \stackrel{!}{=} -\sigma^T \mathbf{D}\sigma$$

Bringing one side to the other:

$$\sigma^T(\dot{\sigma} + \mathbf{D}\sigma) = 0$$

As there is no direct influence on σ^T, the following condition needs to be true so that the closed loop is stable:

$$\dot{\sigma} + \mathbf{D}\sigma = 0 \tag{4.4}$$

Next, the function σ needs to be derived by time:

$$\dot{\sigma}(\mathbf{x}) = \mathbf{G}(\dot{\mathbf{x}} - \dot{\mathbf{x}}_r) \tag{4.5}$$

$\dot{\mathbf{x}}_r$ is the derivative of the reference vector, which contains the reference velocity and the reference acceleration. $\dot{\mathbf{x}}$ is the mathematical model of the system. The given nonlinear system can be described by following system of differential equations:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \tilde{\mathbf{B}}(\mathbf{x})u + \Delta\mathbf{B}(\mathbf{x})u + \mathbf{d}(\mathbf{x}) \tag{4.6}$$

where:

$\mathbf{f}(\mathbf{x})$	: system vector
$\tilde{\mathbf{B}}(\mathbf{x})$	: known part of the input vector
$\Delta\mathbf{B}(\mathbf{x})$	: unknown/changing part of the input vector
$\mathbf{d}(\mathbf{x})$	: disturbances to the system

Putting Equation 4.6 into 4.5, results in:

$$\dot{\sigma}(\mathbf{x}) = \mathbf{G}(\mathbf{f}(\mathbf{x}) + \tilde{\mathbf{B}}(\mathbf{x})u + \Delta\mathbf{B}(\mathbf{x})u + \mathbf{d}(\mathbf{x}) - \dot{\mathbf{x}}_r) \tag{4.7}$$

The final control law can be derived by inserting 4.7 into 4.4 and solving this for u:

$$u = -(\mathbf{G}\tilde{\mathbf{B}})^{-1}(\mathbf{G}(\mathbf{f}(\mathbf{x}) + \Delta\mathbf{B}(\mathbf{x})u + \mathbf{d}(\mathbf{x}) - \dot{\mathbf{x}}_r) + \mathbf{D}\sigma) \tag{4.8}$$

The part $\mathbf{f}(\mathbf{x}) + \Delta\mathbf{B}(\mathbf{x})u + \mathbf{d}(\mathbf{x})$ is only known partially or not at all. Therefore, an ANN is used to approximate this unknown part.
For the ANN, a two layer neural network, with a sigmoid activation function in the hidden layer and a linear activation function in the output layer will be implemented. This activation functions are used to enable nonlinear behaviour of the ANN and to give the output layer the range of $-\infty \leq output \leq \infty$. As learning algorithm, a backpropagation learning algorithm will be used. The inputs of the ANN are the angle θ, the angular velocity $\dot{\theta}$ and the error $\theta - \theta_{ref}$. In the hidden layer, five neurons will be used in the beginning. This number can be varied, up to a total number of thirty neurons in the hidden layer. The output of the network is the approximated system. As the system is of order two, there are two outputs needed. A schematic

of the described neural network is shown in Figure 4.6. This figure has all needed nodes labelled with the according variable names and some of the weights are labelled as examples. The values of the different nodes are stored in vectors and the weights between the layers are stored in matrices.

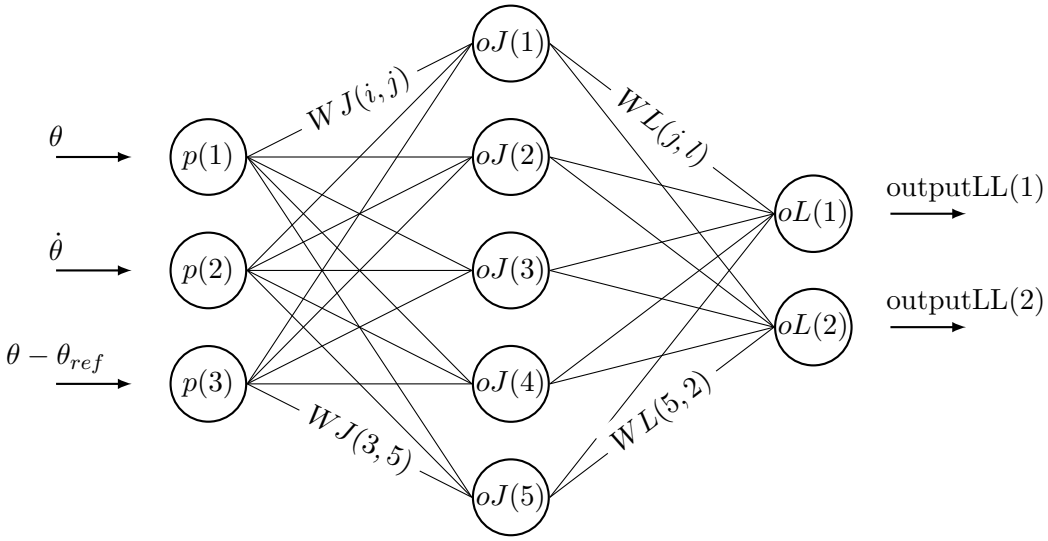

Figure 4.6: Schematic of the used artificial neural network with variable names

As the control law and the structure of the ANN are finished, the next step is to implement the overall controller. First part of the algorithm, is a Matlab function called *NN_init*. This script declares the global variables WJ, WL and *cycle*, which are needed for the final implementation. WJ is initialised as matrix with thirty rows and three columns, which are all set to 0. WL is also a 0 matrix, with 2 rows and thirty columns. And the variable cycle is simply set to 1. Furthermore, the sampling time is defined to $T_s = \frac{1}{2}$ ms.
The next part of the system is the code for the neural network and the controller implementation. In Listing 4.2 shows the complete Matlab function implementing the neural network and the sliding mode controller. At the beginning of the function, the global variables are declared. Afterwards, the different variables are extracted from the input vector. Afterwards, some definitions, which are needed later on, are done. As a next step, the matrices WJ and WL are initialised with values different from 0, so that the ANN works. Now the real calculations begin. First, the values of the hidden layer are calculated, using two nested for loops. After this, the output of the neural network *outputLL* is calculated. After these calculations, some definitions, like the gain vector $\mathbf{g}$, the derivative of the reference vector $\dot{\mathbf{x}}$, etc. are done. The next part of the function includes the backpropagation learning algorithm. The last part calculates the output torque $control_torque$ of the sliding mode controller, using the previously described control law (Equation 4.8). Output of the function are the actual control value $control_torque$ and four values of the weighting matrices to check, if the neural network is actually adapting to the system.

```
1 function [output] = CSMNN_control_sim(input)
2
3 global WJ; %Weights are global variables
4 global WL;
5 global cycle;
6
7 %Inputs  !INCLUDE EXACTLY LIKE IT IS WRITTEN HERE
8 theta_ref = input(1);
9 theta_dot_ref = input(2);
```

```
theta_ddot_ref = input(3);
theta = input(4);
theta_dot = input(5);
theta_ddot = input(6);
Kp = input(7);
Kv = input(8);
M = input(9);
D = input(10);
m = input(11); %Number of neurons in hidden layer
epsilon = input(12); %Learning rate, epsilon
enableNN = input(13); %Enables NN

p = [theta; theta_dot; theta-theta_ref];
n = length(p);
l = 2;
output = zeros(5,1);
%Initialization of WJ, WL only in first cycle, to different small numbers
if (cycle==1)
    cycle=cycle+1; %Counter

    for row=1:m
        for column=1:n
            WJ(row, column) = 1 + (row*0.16)-(column*0.23); %Small numbers
        end
    end

    for row=1:m
        for column=1:l
            WL(column, row)= -1 - (row*0.14)+(column*0.25);
        end
    end
end %End of initialization
% Begin algorithm
outputLJ = zeros(m,1);
for a = 1:m
    for b = 1:n
        outputLJ(a) = outputLJ(a) + (WJ(a,b)*p(b));
    end
    outputLJ(a) = 2/(1+exp(-outputLJ(a)))-1;
end
outputLL = zeros(l,1);
for a = 1:l
    for b = 1:m
        outputLL(a) = outputLL(a) + (WL(a,b)*outputLJ(b));
    end
end
% additional definitions
G = [Kp Kv]';
x_dot_ref = [theta_dot_ref; theta_ddot_ref];
sigma = Kp * (theta - theta_ref) + Kv * (theta_dot - theta_dot_ref);
sigma_dot = Kp * (theta_dot - theta_dot_ref) + Kv * (theta_ddot - theta_ddot_ref);
deltaL = G*(D*sigma + sigma_dot);
% backpropagation learning algorithm
dWL = zeros(l,m);
deltaJ = zeros(m,1);
for a = 1:m
    for b = 1:l
        dWL(b,a) = epsilon * deltaL(b) * outputLJ(a);
        deltaJ(a) = deltaJ(a) + deltaL(b) * WL(b,a);
    end
end
dWJ = zeros(m,n);
```

```
72 for a = 1:m
73     for b = 1:n
74         dWJ(a,b) = epsilon * outputLJ(a)*(1-outputLJ(a)) * deltaJ(a) * p(b);
75     end
76 end
77 % calculation of output torque
78 B = [0; 1/M];
79 if enableNN
80     control_torque = -1/(G'*B)*(G'*(outputLL-x_dot_ref)+D*sigma);
81     WL(1:size(dWL,1), 1:size(dWL,2)) = WL(1:size(dWL,1), 1:size(dWL,2)) + dWL;
82     WJ(1:size(dWJ,1), 1:size(dWJ,2)) = WJ(1:size(dWJ,1), 1:size(dWJ,2)) + dWJ;
83 else
84     control_torque = 0;
85 end
86 %Function output, control torque and some weights
87 output(1)=control_torque;
88 output(2)=WJ(1,2);
89 output(3)=WJ(2,1);
90 output(4)=WL(2,1);
91 output(5)=WL(2,2);
```

Listing 4.2: Matlab function implementing the neural network and the sliding mode controller

Now that all parts are prepared, the controller can be implemented and verified by simulation with the actual model.

4.4 Controller implementation

The implementation of the control loop is done in Simulink. Figure 4.7 shows a schematic, which explains the principle of the signal flow of the previously described subsystems. The reference generator calculates the values of the $\sin^2$ velocity profile and feeds them to the neural network. The control torque of the controller drives the system, which has a feedback loop to the neural network.

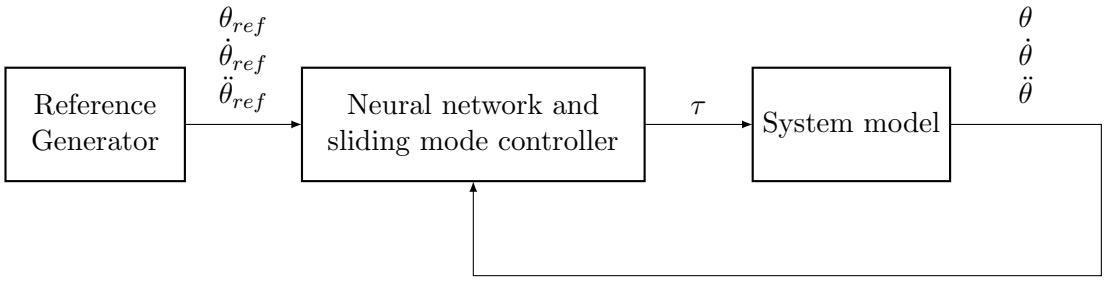

Figure 4.7: Schematic of the implemented control loop containing the $\sin^2$ velocity profile, the neural network and the dynamic model of the system

For the implementation of the controller, some parameters have to be defined first. Table 4.1 shows a set of parameters, which provides satisfying results.

Table 4.1: Parameters of the ANN sliding mode controller which provided satisfying results

Parameter	Value	Description
k_p	5	position gain
k_v	1.3	velocity gain
M	$80 \cdot 10^{-6}$	estimated moment of inertia
D	60	σ weighting in Lyapunov function
ϵ	10^{-4}	learning rate

Figure 4.8 shows the results of the simulation. The reference has the ending position at 5π, a maximum angular velocity of 1 $\frac{\text{rad}}{\text{s}}$ and a maximum acceleration of 5 $\frac{\text{rad}}{\text{s}^2}$. The upper plot of the figure shows the actual and the reference position. In this plot it can be seen, that there is almost no error. The plot in the middle shows the position error of the cylinder. The error has a peak of approximately 0.008 at the beginning and oscillates around 0.001. The steady state error is 0. The bottom plot shows the applied torque. As the cylinder moves 5π, it turns $\frac{5}{2}$ times. When moving to the unstable equilibrium point, the applied torque is positive. When moving back to the stable equilibrium point, the applied torque is negative, to brake the cylinder.

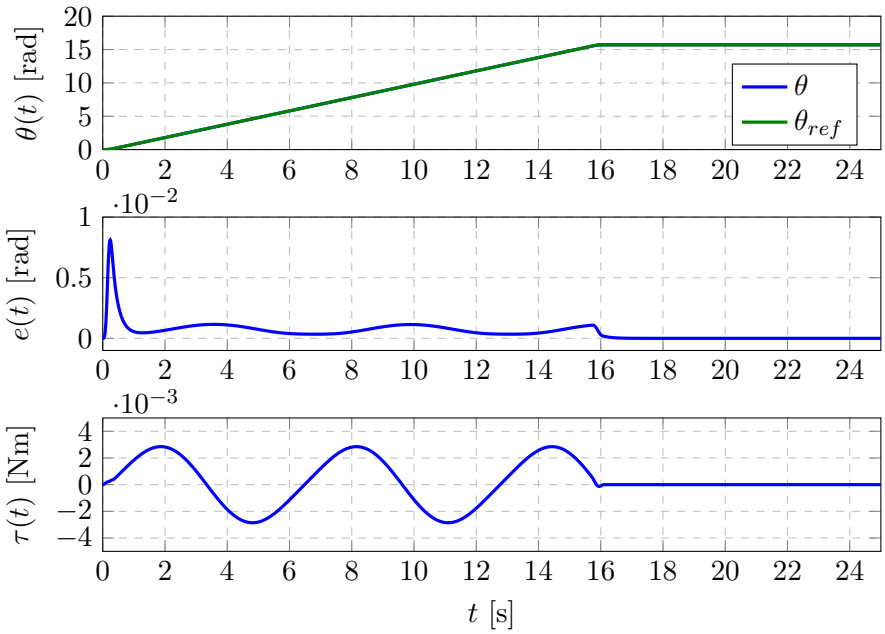

Figure 4.8: Simulation of the sliding mode neural network controller with $k_p = 5$, $k_v = 1.3$, $M = 80 \cdot 10^{-6}$, $D = 60$ and $\epsilon = 10^{-4}$

To show the influence of the parameters on the position error, a second simulation with different parameters is performed. k_p, k_v and D are increased for the second simulation. The results are shown in Figure 4.9. Compared to the previous results, the initial peak of the position error is much smaller.

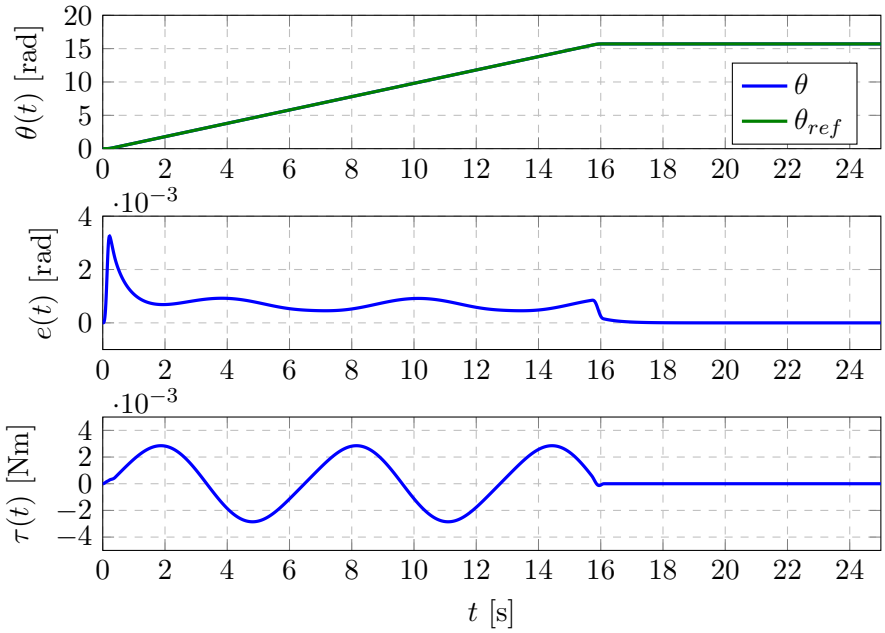

Figure 4.9: Simulation of the sliding mode neural network controller with $k_p = 7$, $k_v = 4$, $M = 80 \cdot 10^{-6}$, $D = 80$ and $\epsilon = 10^{-4}$

Finally, the influences of the different parameters on the system response are examined. First, the number of neurons in the hidden layer is changed. Increasing the number or neurons results in a higher initial peak of the position error but reduces it afterwards. Setting the number too high, results in unstable behaviour. On the other hand, decreasing the number of neurons in the hidden layer decreases the initial peak of the error but results in a higher amplitude of the oscillation afterwards. Next, the value of the position gain k_p is changed. Increasing this parameter reduces the overall position error. Also, if the value is set too high, the closed loop system gets unstable. Reducing k_p results in an increase of the position error and can even result in a steady state error if it gets too low. As next parameter to be varied k_v is chosen. Similar to the number of neurons, increasing this parameter decreases the initial peak of the position error but slightly increases it afterwards. Again, raising the parameter too high results in unstable behaviour. Setting k_v very low greatly increases the initial peak of the position error and results in a decreased error later on. The last parameter which is varied, is the learning rate ϵ. Trying to increase this parameter above 10^{-4} results in unstable behaviour almost immediately. Reducing the learning rate, increases the oscillation of the position error and also results in a steady state error, once it is reduced below 10^{-6}.

Bibliography

[1] K.H. Kim and A. Kern. Comparison of different methods of trajectory calculation for an industrial robot. In *Power Electronics and Drive Systems, 1995., Proceedings of 1995 International Conference on*, pages 449–453 vol.1, Feb 1995.