HSB
Hochschule Bremen
City University of Applied Sciences

AKI-KNN-Dokumentation

Part 1

Semester: Sommersemester 2016

Name:	Andre Bode
Matrikelnummer:	358254
Name:	Philipp Sommer
Matrikelnummer:	355983
Name:	Hermann Wafo
Matrikelnummer:	5006146
Name:	Martin Müller
Matrikelnummer:	358740

1 Einleitung

In diesem Dokument soll die Bearbeitung des Projektes im Modul „AKI-KNN“ im Sommersemester 2016 bei Prof. Dr. Mevenkamp dokumentiert werden.
Festgehalten wird im Folgenden der theoretische, mathematische und physikalische Hintergrund des Projektes, die benötigten Formeln und die Umsetzung in MATLAB / Simulink.

2 Formeln aus der Vorlesung

MLP:

$$e^{(j)} = W^{(j)} \cdot o^{(j-1)} \tag{1}$$

$$o^{(j)} = \tanh(c_F \cdot e^{(j)}) \qquad \text{bzw.} o^{(j)} = \tanh(c_F \cdot e^{(j)}) \tag{2}$$

$$o^{(j)} = \tanh(c_F \cdot W^{(j)} \cdot o^{(j-1)}) \tag{3}$$

3 Theoretische Herangehensweise / Formeln

Um die Regelung des Balancierens eines Balls auf einer Felge in Simulink simulieren zu können, müssen mathematische Modelle für das Beschreiben der Situation entwickelt werden.
Interessant ist hierbei vor allem, wie die Bewegung des Balles auf der Felge beschrieben werden kann, denn hierdurch ist es möglich Rückschlüsse auf die Position des Balles zu einer bestimmten Zeit zu ziehen.

Die Physik liefert hier den Zusammenhang zwischen den Formeln zur Berechnung von Beschleunigung, Geschwindigkeit und Ort. Sie werden auch **Impulssatz** genannt. Die Geschwindigkeit kann als Integration der Beschleunigung angegeben werden.

$$v(t) = \int_{t_0}^{t} a(t)\ dt \tag{4}$$

Unter Berücksichtigung eines Anfangwertes v_0 kommt man somit zur Formel :

$$v(t) = v_0 + \int_{t_0}^{t} a(t)\ dt \tag{5}$$

Ebenso kann nun der Ort als Integration der Geschwindigkeit angegeben werden.

$$x(t) = \int_{t_0}^{t} v(t)\ dt \tag{6}$$

Unter Berücksichtigung eines Anfangwertes x_0 kommt man somit zur Formel :

$$x(t) = x_0 + \int_{t_0}^{t} v(t)\ dt \tag{7}$$

Da die Beschleunigung a als Quotient aus Summe aller Kräfte und der Masse angegeben werden kann,

$$a = \frac{F_{gesamt}}{m} \tag{8}$$

besteht daher ein direkter Zusammenhang zwischen den Kräften die auf den zu modellierenden Ball wirken, geteilt durch seine Masse und dem Ort, an dem sich der Ball zu einer bestimmten Zeit befindet :

$$x(t) = x_0 + \int_{t_0}^{t}\int_{t_0}^{t} a(t)\ dt\ dt \tag{9}$$

$$= x_0 + \int_{t_0}^{t}\int_{t_0}^{t} \frac{F_{gesamt}}{m}\ dt\ dt \tag{10}$$

3.1 Wichtige physikalische Kräfte und Kenngrößen

Im nächsten Schritt müssen somit die Kräfte ermittelt werden, die auf den Ball wirken.
Diese Kräfte und weitere, für das Verständnis wichtige physikalische Größen, sollen im folgenden kurz erläutert werden

3.1.1 Drehmoment

Das Drehmoment ist eine Kraft der Rotation in der klassischen Mechanik, die Einfluss auf die Beschleunigung eines rotierenden Körpers hat.

Die allgemeinen Formeln lauten:
M = Drehmoment
r = Radius
F = Kraft der Bewegung

$$M = rF \tag{11}$$

$$\vec{M} = \vec{r} \times \vec{F} \tag{12}$$

Die Formel 12 ist für ein mehr als zweidimensionalen Raum gedacht und beinhaltet ein Kreuzprodukt.

3.1.2 Massenträgheitsmoment

Beschreibt den Widerstand eines Körpers bei Änderung seiner Rotationsbewegung. Drehimpulserhaltung bedeutet eine Änderung des Trägheitsmoments oder der Winkelgeschwindigkeit (wenn kein Drehmoment einwirkt).

Die allgemeinen Formeln lauten:
J = Trägheitsmoment
M = Drehmoment

Stab

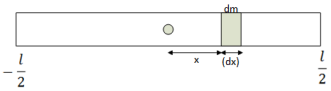


Abbildung 1: Traegheitsmoment

$$dJ = x^2 \cdot dm$$
$$dm = \frac{M}{l} \cdot x$$

$$J = \int_{-\frac{l}{2}}^{\frac{l}{2}} \frac{M}{l} x^2 dx = \frac{M}{l} \left[\frac{1}{3}x^3\right]_{-\frac{l}{2}}^{\frac{l}{2}} \tag{13}$$

$$= \frac{2}{3}\frac{M}{l}\frac{l^3}{8} = \frac{1}{12}Ml^2 \tag{14}$$

Kugel

$$J = \frac{2}{5}M \cdot r^2 \tag{15}$$

Zylinder / Scheibe

$$J = \frac{1}{2}M \cdot r^2 \tag{16}$$

3.1.3 Drehimpuls-Satz / Drall-Satz

Der Drehimpuls ist die „Wucht“ einer Bewegung, die kreisförmig gerichtet ist. Diese „Wucht“ entspricht der Geschwindigkeit und Masse des Körpers.

Die allgemeinen Formeln lauten:
$\vec{P}$ = Drehimpuls
M = Drehmoment
$\vec{w}$ = Winkelgeschwindigkeit
$\dot{w}$ = Winkelbeschleunigung
J = Trägheitsmoment

$$\vec{P} = J \cdot \vec{w} \tag{17}$$

$$\frac{dP}{dt} = J \cdot \dot{w} = \sum M_i \tag{18}$$

$$[Nm] = [kg \cdot m^2] \cdot [\frac{1}{s^2}] = [kg \cdot \frac{m^2}{s^2}] \tag{19}$$

3.1.4 Hangabtriebskraft

Die Hangabtriebskraft ist eine Kraft, welche auf eine Masse auf einer schiefen Ebene einwirkt. Hierbei wird die Masse z.B. durch die Gravitation (F_g) nach unten gezogen, durch das Hinderniss der schiefen Eben (Winkel: α) wird diese Kraft „umgelenkt“.

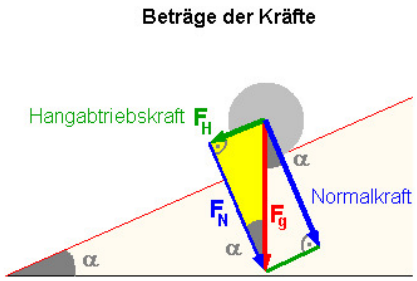


Abbildung 2: Kräfte einer schiefen Ebene (Quelle: `http://schule-bw.de`)

Die allgemeinen Formeln lauten:

$$F_H = F_g \sin(\alpha) \tag{20}$$
$$F_N = F_g \cos(\alpha) \tag{21}$$

3.2 Ball auf beschleunigter und geneigter Fläche

Ein Ball der auf einer beschleunigten Fläche mit dem Neigungswinkel α und der Beschleunigung a_F liegt, ist neben der Hangabtriebskraft auch der Beschleunigung der Fläche und dem Trägheitsmoments der Balles ausgesetzt.

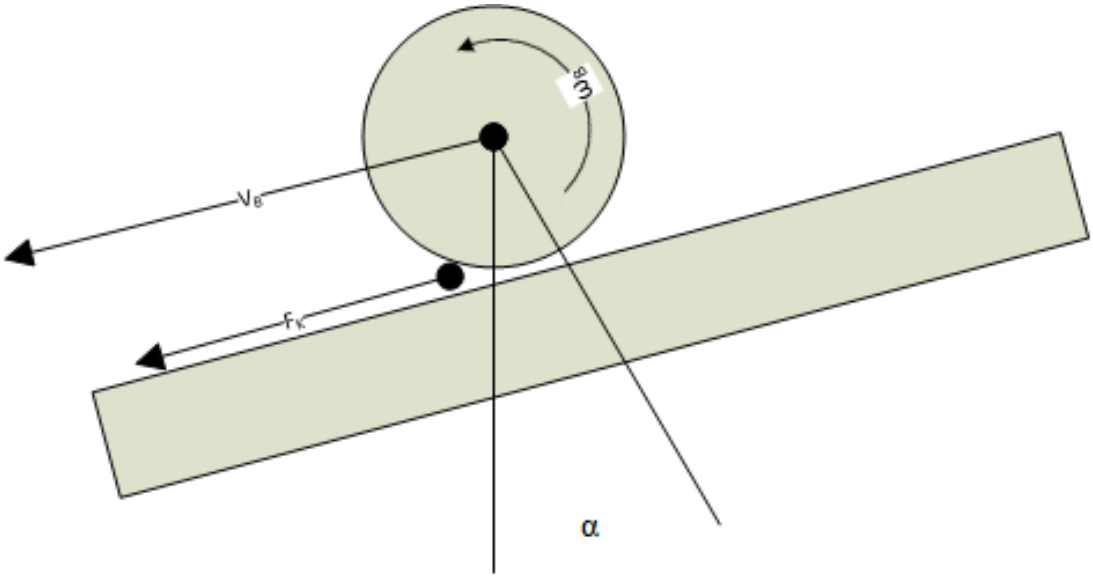


Abbildung 3: Ball auf beschleunigter und geneigter Fläche

Die Beschleunigung des Balles wird daher beschrieben durch :

$$m_B \cdot \dot{v}_B = F_K + m_B \cdot g \cdot sin(\alpha) \tag{22}$$

$$a_B = \frac{F_K}{m_B} + g \cdot sin(\alpha)) \tag{23}$$

wobei F_K die Kraft die durch die Beschleunigung auf den Ball wirkt darstellt und $g \cdot sin(\alpha)$ die Hangabtriebskraft.

3.3 Rotation der Felge

Die oben erwähnte beschleunigte Fläche, auf der Ball liegt, ist in diesem Fall eine rotierende Felge.

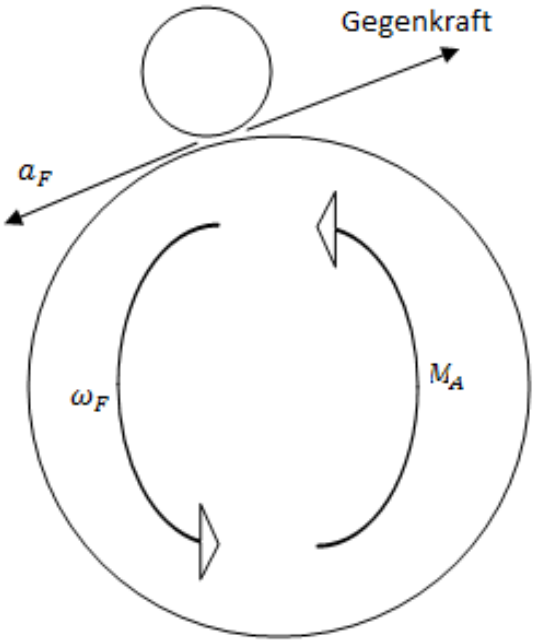


Abbildung 4: Rotierende Felge

Die Bewegung der Felge kann beschrieben werden durch den folgenden Zusammenhang :

$$J_F \cdot \dot{\omega}_F = M_A - F_K \cdot R \tag{24}$$

wobei J_F das Trägheitsmoment der Felge, $\dot{\omega}_F$ die Kreisbeschleunigung der Felge, M_A das Drehmoment der Felge und $-F_K \cdot R$ die Gegenkraft beschreibt.
Die Außengeschwindigkeit am Rand der Felge ergibt sich durch die Multiplikation der Winkelgeschwindigkeit der Felge ω_F mit dem Radius der Felge R.

$$v_F = \omega_F \cdot R \tag{25}$$

Durch Ableiten dieser Geschwindigkeit kann auf die Beschleunigung geschlossen werden. Denn :

$$a_F = \dot{v}_F = \dot{\omega}_F \cdot R \tag{26}$$

Stellt man nun die obige Formel nach $\dot{\omega}_F$ um und setzt sie hier ein, erreicht man die folgende Formel zur Berechnung der Beschleunigung :

$$a_F = \frac{R}{J_F} \cdot M_A - \frac{F_K \cdot R^2}{J_F} \tag{27}$$

Hierbei ist der hintere Term in unserem Versuch $-\frac{F_K \cdot R^2}{J_F}$ allerdings vernachlässigbar und die Formel vereinfacht sich zu :

$$a_F = \frac{R}{J_F} \cdot M_A \tag{28}$$

3.4 Ball auf rotierender Felge

Setzt man nun die obigen Formeln zusammen, so ergibt sich für die Beschleunigung des Balles die folgende Formel :

$$a_B = \frac{m_B \cdot g}{(R+r)(m_B + \frac{J_B}{r^2})} + \frac{J_B}{r^2(m_B + \frac{J_B}{r^2})} \cdot a_F \tag{29}$$

Hierbei beschreibt der Teil

$$\frac{m_B \cdot g}{(R+r)(m_B + \frac{J_B}{r^2})} \tag{30}$$

den Anteil der Beschleunigung, auf den die Gewichtskraft einwirkt. Dies wird deutlich klarer, wenn man darauf hinweist, dass der Term $sin(\alpha)$ aus der Formel der Hangabtriebskraft hier durch :

$$sin(\alpha) = \frac{x_B}{R+r} \tag{31}$$

ersetzt wurde. Der zweite Teil :

$$\frac{J_B}{r^2(m_B + \frac{J_B}{r^2})} \cdot a_F \tag{32}$$

beschreibt hierbei den Anteil der Beschleunigung des Balles, den die Beschleunigung der Felge ausmacht.
Hierbei wird davon ausgegangen, dass es zwischen Felge und Ball keinen Schlupf gibt. Da die Beschleunigung der Felge nicht mit dem Verhältnis 1:1 in Beschleunigung des Balles mitüber geht, beschreibt

$$\frac{J_B}{r^2(m_B + \frac{J_B}{r^2})} \tag{33}$$

einen Faktor der ein Verhältnis zwischen Beschleunigung der Felge und Beschleunigung des Balles beschreibt.
Um nun von der Beschleunigung des Balles auf den Ort / die Auslenkung des Balles zu schließen muss die Beschleunigung zwei mal integriert werden. Denn :

$$\ddot{x}_B = a_B = \frac{m_B \cdot g}{(R+r)(m_B + \frac{J_B}{r^2})} + \frac{J_B}{r^2(m_B + \frac{J_B}{r^2})} \cdot a_F \tag{34}$$

4 Umsetzung in Simulink

4.1 Beschreibung Gesamtsystem

Im Gesamtsystem sind neben den beiden teilmodellen noch zusätzliche Gain Elemente vorhanden um die Übersetzung des MoTors zu berücksichtigen. Außerdem wird hier auch später das Neuronale Netz eingesetzt und die Ergebnisse in einem Scope dargestellt.

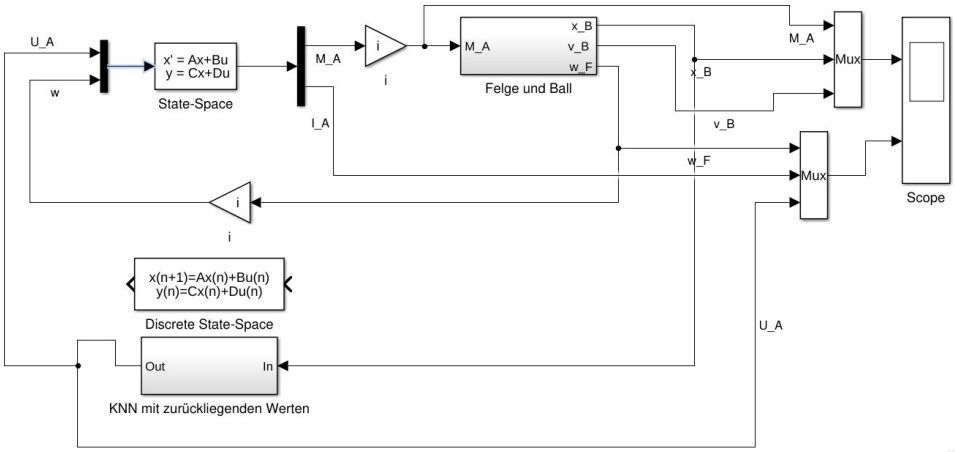


Abbildung 5: Die gesamt Ansicht in Simulink

4.2 Beschreibung Modell Ball & Felge

Die Modellierung des Balls und der Felge sind innerhalb Simulinks in einem Subsystem zusammengefasst. Es besteht aus den Subsystemen „Felge“ und „Ball“.

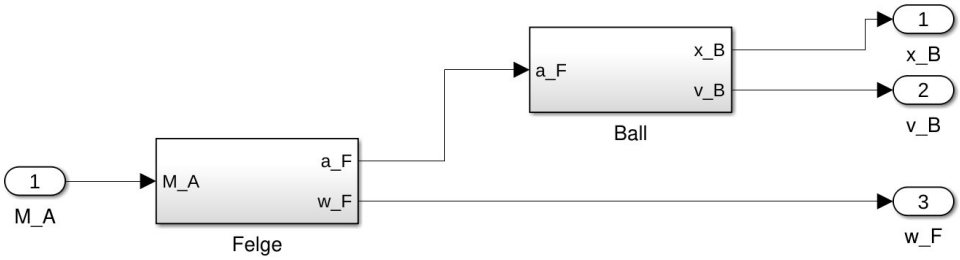


Abbildung 6: Die Ansicht von Ball und Felge

Das Subsystem Felge ist eine direkte Implementierung der obigen Formel aus dem Abschnitt 3.3. Es wird hierbei dafür genutzt für das Subsystem „Ball“ die Beschleunigung der Felge a_F und

für das Gesamtsystem die Kreisgeschwindigkeit der Felge ω_F (in Simulink mit w_F bezeichnet) zu berechnen.

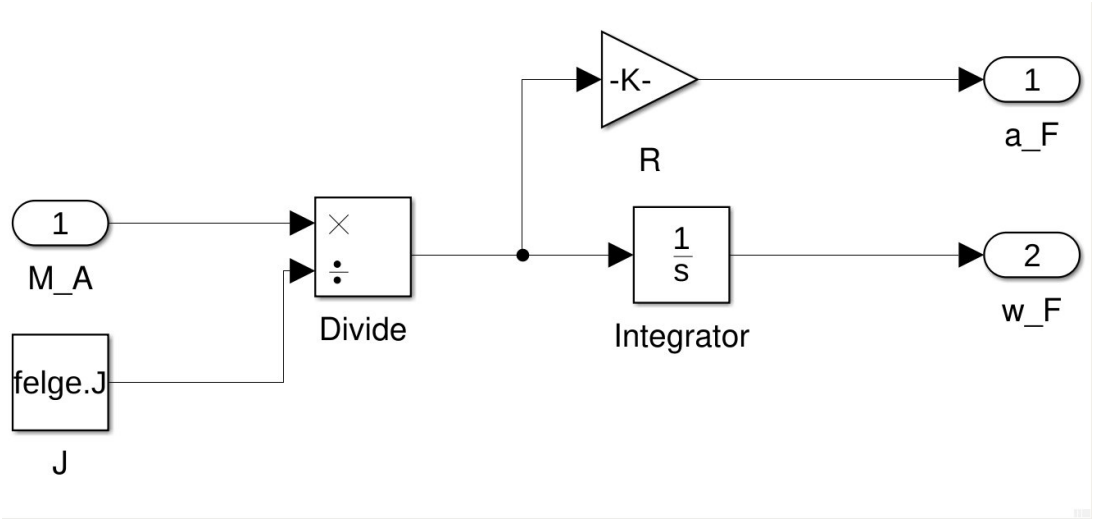


Abbildung 7: Felge

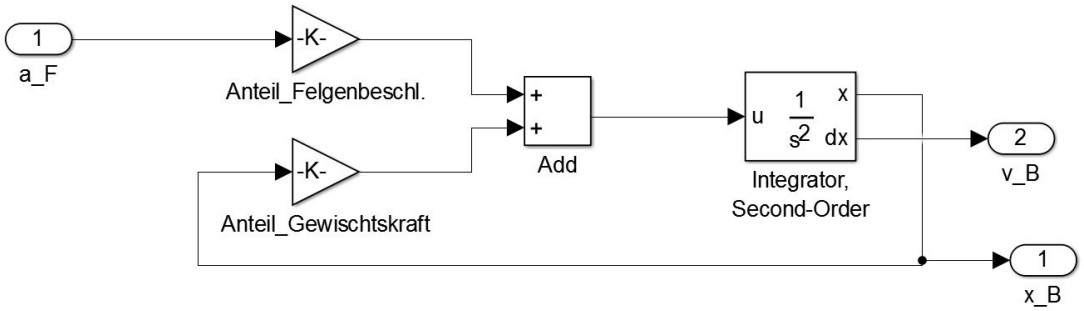
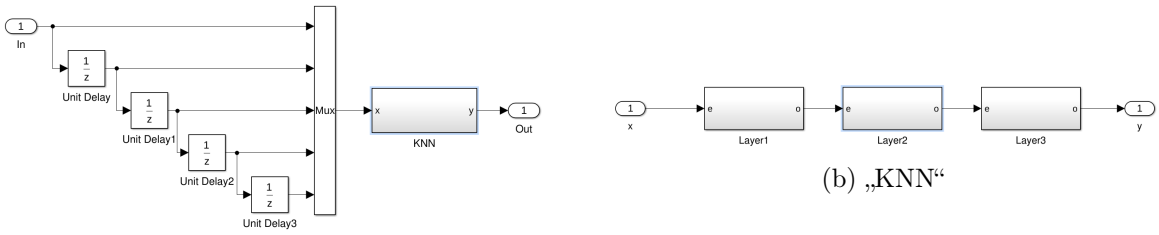


Abbildung 8: Ball

Die Abbildung 8 entspricht einer direkten Implementierung der Formel 34. Die einzelnen oben erwähnten Anteile für die Felgenbeschleunigung und die Gewichtskraft sind hier der übersichthalber in „Gains“ ausgelagert. Um von $\ddot{x}_B$ zu x_B und zu v_B zu gelangen wurde außerdem ein Second-Order-Integrator angewendet. Die Erste Integration („dx“) liefert hierbei die Geschwindigkeit des Balles v_B, die zweite Integration liefert den Ort / die Auslenkung des Balles x_B.

4.3 Beschreibung Neuronales Netz in Simulink

Das neuronale Netz wurde in Simulink umgesetzt und ist ein Teil der Abbildung 5



(a) „KNN mit zurückliegenden Werten“

(b) „KNN“

Abbildung 9: Das KNN in Simulink

Das neuronale Netz besteht aus drei Hidden-Layer, der Eingabe-Layer besitzt fünf Neuronen vom Schieberegister 4.4. Der Ausgabe-Layer besitzt nur ein Neuronen für die Spannung des Reglers. Ein Hidden Layer wie sie in Abbildung 10 entspricht der Formel 3 fürs MLP.

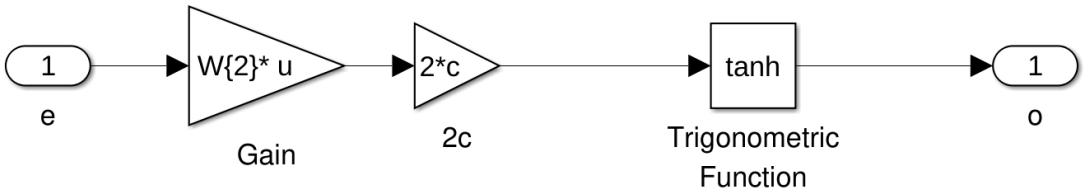


Abbildung 10: Ein Hidden-Layer im neuronalen Netz

4.4 Beschreibung Schieberegister

Das neuronale Netz soll den Ball auf der Felge halten, indem es die Felge dreht. Von der aktuellen Position des Balles auf der Felge kann höchstens die benötigte Beschleunigung der Felge vom neuronalen Netz berechnet werden. Für die Richtung und für einen genaueren Wert der Beschleunigung (Anpassung der Spannung am Regler) wird eine Historie an Positionen des Balls benötigt. Dies wir einfach durch ein Schieberegister realisiert, der jeden neuen Zeitabschnitt den Wert speichert beziehungsweise schiebt. Dadurch hat das neuronale Netz bei dieser Implementierung mit den Schieberegiester in Abbildung 11 ein Zugriff auf den aktuellen Wert und vier Werte zuvor.

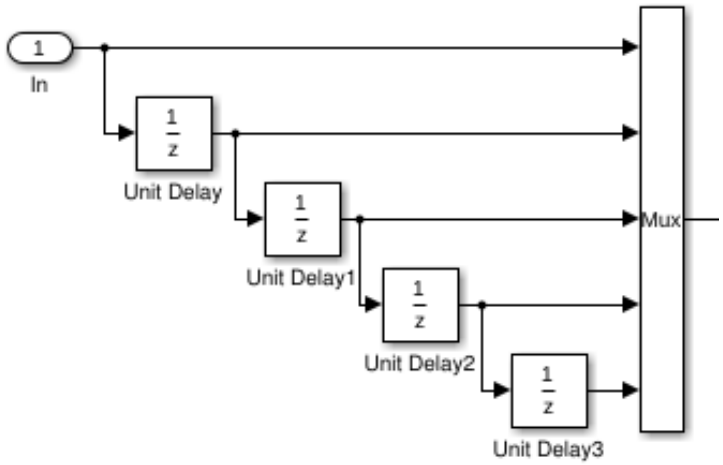


Abbildung 11: Das Schieberegister in „KNN mit zurückliegenden Werten“

5 Umsetzung des Trainingsalgorithmus

5.1 Auswahl der Lerndaten

Um das neuronale Netz einzulernen sind Eingabe-Daten für die Input-Neuronen und erwaretete Ausgabe-Werte für das Output-Neuron notwendig. Diese werden mit der Funktion „createTrainingDataAndExpectedData“ aus den Ausgaben des Scopes aus Simulink genereriert. Hierbei werden die ScopeDaten in Schritten der Größe „ScopeStepSize“ iteriert und jeweils für die Eingabedaten die Auslenkung des Balles zu dem jeweiligen Zeitpunkt und die Auslenkung zu den vier vorherigen Zeitpunkte ausgewählt. Die erwarteten Ausgabedaten werden aus den Spannungswerten des Reglers an den genau den selben Zeitpunkten wie die Eingabedaten ausgewählt.

```
%% function - createTrainingDataAndExpectedData
% function creates training data from Scope-Data
% Function iterates all scope-data in steps of "ScopeStepSize" and outputs
% the return values x_train and U_A.

% x_train is a matrix which contains training data for the neural net.
% x_train has one coloumn per timestep. Every column contains the current
% x-value of the timestep and some previous x-values.
% The parameter previous_x_count defines how many previous x-values are
% used. At the very beginning one does not have previous x-values. Here the
% value "zero" is used.
% Scope-Data contains very very much data, because of this the parameter
% ScopeStepSizes defines how many of the ScopeData-Values are used.
% For example if ScopeStepSize is 5000, ever 5000th value of the scope-data
% is used.

% U_A is a matrix, which contains the expected Output values of the neural
% net, for the same timesteps x_train refers to

% @param previous_x_count defines how many previous x-values are used
% @param ScopeStepSize defines how many of the ScopeData-Values are used
% @param ScopeData link to data create by simulink simulation

function [x_train, U_A] = createTrainingDataAndExpectedData(previous_x_count,
  ScopeStepSize, ScopeData)
  x_train = [];      % initialize as empty vector
  U_A     = [];      % initialize as empty vector
  previous_x = zeros(1, previous_x_count); % initialize with zeros

  % iterate through ScopeData.time
  for i=1:ScopeStepSize:length(ScopeData.time)
    % x_train is the training input data
    current_x = ScopeData.signals(1).values(i,2); % get current x-value
    % merge current_x with previous_x values and add them to x_train matrix
    x_train_temp = [x_train  [current_x; previous_x']];
    x_train = x_train_temp;
    % U_A are the expected results of the neural net
    U_A_temp = [U_A ScopeData.signals(2).values(i,3)];
    U_A = U_A_temp;
    % shift previous x-values through vector
    for i = length(previous_x):-1:2
      previous_x(i) = previous_x(i-1);
    end
    previous_x(1) = current_x; % put current x_value into first value of
  previous values
```

```
44      end
45    end
46
```

Listing 1: Das Listing zeigt einen Quellcode der Funktion createTrainingDataAndExpectedData

5.2 Implementierung von mlp_train

Zum Einlernen des neuronalen Netzes wird der Backpropagation-Algorithmus genutzt. Hierbei wird versucht die Gewichte des neuronalen Netzes so zu optimieren, dass der Fehler zwischen erwarteter Ausgabe und tatsächlicher Ausgabe des neuronalen Netzes minimiert wird. Dies ist im Backpropagation-Algorithmus möglich, in dem der Fehler an der Ausgangsschicht von hinten nach vorne durch das neuronale Netz zurück propagiert wird. Somit kann ein Zusammenhang zwischen dem Gesamtfehler und dem Fehler den die einzelnen Gewichte verursachen hergestellt werden.
In Matlab wurden zur Implementierung des Backpropagation-Algorithmus die Formeln aus der Vorlesung genutzt.

```
1   % iterate through all input values (x)
2   for ix = 1:length(x)
3     %% calc the deltas of the output layers
4     % calc difference between mlp and original function
5     I = y(:,ix)-ty(:,ix);
6     % create diagonal zero/value matrix and calc fault-vector (d)
7     % d = cf * I * diag(1-(y(:,ix).^2));
8     % changed diag() to .* due to performance optimization
9     d = cf * I .* (1-(y(:,ix).^2));
10    previous_o = o{n-1}(:,ix);
11    % calc gradient by fault-vector multiplied with previous output of the neurons
12    grad_w = d * previous_o';
13    % calc the delta of the weights
14    delta_w{n} = delta_w{n} + (grad_w * epsilon);
15
16    %%calculate the deltas of all other layers
17    d_j = d;
18    %iterate through all hidden layers and input layer
19    for j = 1:(n - 1)
20
21      % new d is previous d multiplied with previous weights, cf and
22      % (1-o^2), where o is o from the current layer
23
24      % d_j_minus1_trans = d_j' * weights{n-j+1} * cf * diag((1-(o{n-j}(:,ix).^2))
    ');
25      % changed diag() to .* due to performance optimization
26      d_j_minus1_trans = d_j' * weights{n-j+1} * cf .* (1-(o{n-j}(:,ix).^2))';
27      d_j_minus1 = d_j_minus1_trans';
28
29      % differ between hidden and input layer
30      if j < (n - 1)
31        previous_o_j_minus2 = o{n-j-1}(:,ix);
32      else
33        previous_o_j_minus2 = x(:,ix);
34      end
35
36      % the gradient is calculated by multiplying d with the previous o
37      grad_w_j_minus1 = d_j_minus1 * previous_o_j_minus2';
```

```
38
39      % the delta for the new weights is learn-factor epsilon multiplied
40      % with the gradient
41      delta_w{n-j} = delta_w{n-j} + epsilon * grad_w_j_minus1;
42
43      % from previous layer
44      d_j = d_j_minus1;
45    end
46  end
47
48  % subtract the cell array delta_w from the same structure cell-array
49  % weights --> new_weights = weights-delta_w
50  new_weights = cellfun(@minus,new_weights,delta_w,'UniformOutput',false);
51
```

Listing 2: Das Listing zeigt einen Auszug aus der Funktion mlp_train

5.2.1 Skalieren des Wertebereichs in den Wertebereich von tanh

Während der Arbeit mit dem neuronalen Netzes fiel immer wieder auf, dass die hohen Ausschläge der Regler-Kennlinie nicht vernünftig approximiert werden konnten. Dies wird dadurch verschuldet, dass als Aktivierungsfunktion Tangens-Hyperbolikus genutzt wird und diese Funktion nur Werte im Wertebereich von -1 bis 1 produzieren kann. Siehe hierzu auch die folgende Darstellung.

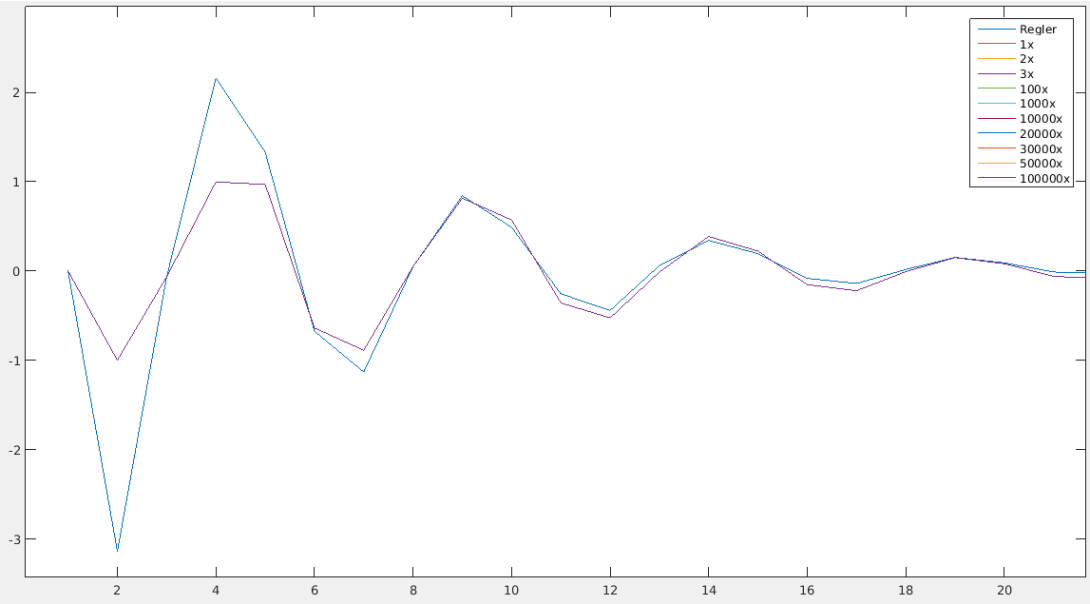


Abbildung 12: Ausgabe begrenzt auf Wertebereich von tanh

Um dieses Problem zu umgehen, wurden alle erwarteten Spannungswerte mit dem das Netz eingelernt werden soll, zuvor auf den Wertebereich von Tangens-Hyperbolikus skaliert. Das neuronale Netz wurde dann mit diesen skalierten Daten eingelernt. Die Ausgaben des neuronalen Netzes wurden dann dementsprechend mit dem gleichen Algorithmus wieder zurück auf den Wertebereich der Spannungswerte skaliert. In Matlab ist dies in der Funktion „scaleRangeOfValues“zu finden.

```
%% function - scaleRangeOfValues
% function scales all values within the vector 'values' from its range
% (max(values) - min(values)) into a range (maxAfter - minAfter)
% @param values    : a vector of values which have to be scaled
% @param minBefore : the smallest value of the old range
% @param maxBefore : the highest value of the old range
% @param minAfter  : the smallest value of the new range
% @param maxAfter  : the highest value of the new range
% @return scaledValues : a vector which contains the scaled values of 'values'
function scaledValues = scaleRangeOfValues(values, minBefore, maxBefore,
  minAfter, maxAfter)
rangeAfter = maxAfter - minAfter;          % target range
rangeBefore = maxBefore - minBefore; % range of values
valuesMinusOffset = values - minBefore;
ratioValues = valuesMinusOffset / rangeBefore; % ratio of values referring to
  range of values
scaledValues = (ratioValues * rangeAfter) + minAfter; % map values to target
  range
end

```

Listing 3: Das Listing zeigt einen Quellcode der Funktion scaleRangeOfValues

Das Skalieren kann zu Beginn der „main.m“ aktiviert werden.

```
% ...
% define use of extra-functions
useAutoAdjustEpsilon = false;
useScaling           = true;
%   ...

```

Listing 4: Das Listing zeigt eine Auszug aus main.m

Wird „useScaling“ an dieser Stelle auf „true“ gesetzt, werden die generierten Daten zum Einlernen entsprechend auf den Wertebereich von Tangens-Hyperbolikus skaliert.

```

% scale range of U_A into range of tanH
% ...
if (useScaling)
  minOfTanH = -1;
  maxOfTanH = 1;
  minOfU_A = min(U_A);
  maxOfU_A = max(U_A);
  U_A = scaleRangeOfValues(U_A,minOfU_A,maxOfU_A,minOfTanH,maxOfTanH);
end
% ...

```

Listing 5: Das Listing zeigt einen Auszug aus main.m

Nach dem Einlernen und Propagieren mit dem neuronalen Netz werden die Werte entsprechend wieder zurückskaliert.

```
% ...
% rescale U_A and results from neuro-net back
% from range of tanH to range of U_A
if (useScaling)
```

```
5      U_A       = scaleRangeOfValues(U_A,minOfTanH,maxOfTanH,minOfU_A,maxOfU_A);
6      y_1       = scaleRangeOfValues(y_1,minOfTanH,maxOfTanH,minOfU_A,maxOfU_A);
7      y_2       = scaleRangeOfValues(y_2,minOfTanH,maxOfTanH,minOfU_A,maxOfU_A);
8      y_3       = scaleRangeOfValues(y_3,minOfTanH,maxOfTanH,minOfU_A,maxOfU_A);
9      y_100     = scaleRangeOfValues(y_100    ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
10     y_1000    = scaleRangeOfValues(y_1000   ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
11     y_10000   = scaleRangeOfValues(y_10000  ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
12     y_20000   = scaleRangeOfValues(y_20000  ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
13     y_30000   = scaleRangeOfValues(y_30000  ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
14     y_50000   = scaleRangeOfValues(y_50000  ,minOfTanH,maxOfTanH, minOfU_A,maxOfU_A);
15     y_100000  = scaleRangeOfValues(y_100000 ,minOfTanH,maxOfTanH,minOfU_A,maxOfU_A);
16   end
17   % ...
18
```

Listing 6: Das Listing zeigt einen Auszug aus main.m

Wie erwartet führt das Skalieren, gerade bei der Approximation der Extremwerte, die -1 und 1 überschreiten, zu deutlich besseren Ergebnissen. Leider gibt es jedoch bei der Approximation der deutlich kleineren Werte, die um 0 herum liegen unerwareteter Weise deutlich stärkere Abweichungen als ohne Skalierung.

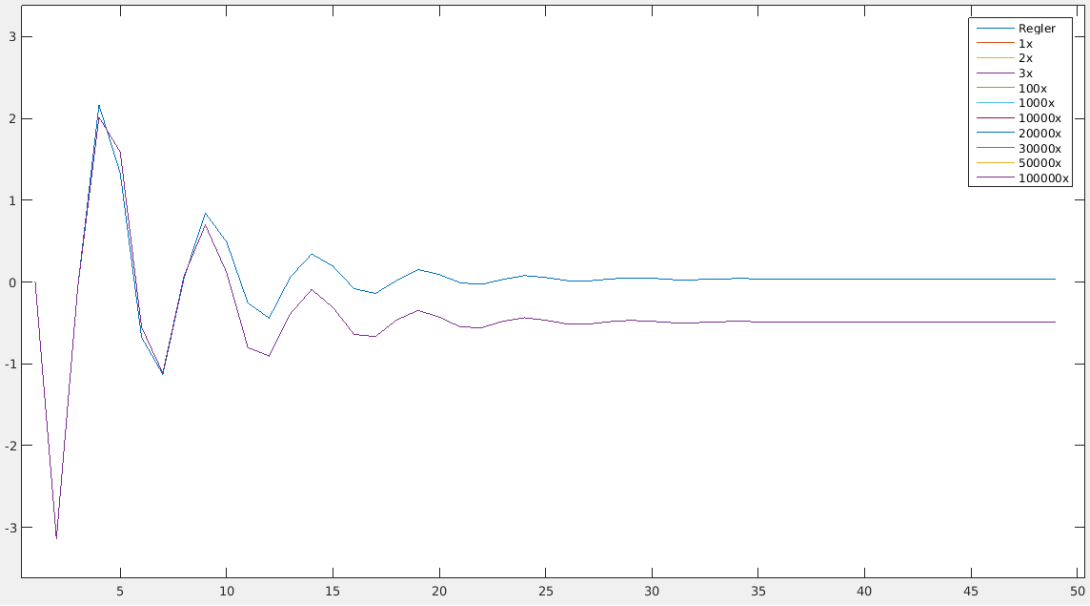


Abbildung 13: Ausgabe nach Begrenzung

Der Grund für diese Abweichungen konnte bis zur Fertigstellung dieser Arbeit leider nicht festgestellt werden. Falls diese jedoch beseitigt werden können, könnten mit Hilfe der Skalierung deutlich genauere Ergebnisse erreicht werden.

5.2.2 Wahl der Lernraten

Die Lernrate wurde zu Beginn manuell gesetzt. Hierbei wurde mit einer relativ kleinen Lernrate mit einem Wert von 0.001 begonnen und diese dann Stück für Stück gesteigert und die Ergebnisse verglichen. Insgesamt ergab sich bei einer festen Lernrate ein optimaler Wert von 0.01, da

hierbei nicht zu große Änderungen mit einem Iterationsschritt durchgeführt werden, die Änderungen jedoch groß genug sind um die Kennlinien ausreichend anzunähern.
Nach dem Testen mit festen Lernraten wurde ein automatisches Einstellen der Lernrate implementiert.
Die Implementierung ist in der Funktion „autoAdjustEpsilon“ zu finden.
Hierbei war der erste Ansatz ein Vergrößern der Norm beim Lernen zu verhindern. Hierzu wurde die Lernrate um einen bestimmten Faktor verkleinert, wenn die Norm und somit der Fehler ansteigt. Hierdurch lies sich der Trend der steigenden Norm in mehreren Tests jedoch nicht aufhalten. Die Lernrate wurde somit solange verkleinert bis sie letztlich so klein war, dass sie keine Veränderung mehr herbeiführt.

Daraufhin wurde versucht die Lernrate immer dann um einen bestimmten Faktor zu verkleinern, wenn die Norm einen neuen mininmal Wert erreicht. Da zu Beginn jedoch sehr oft das Minimum unterboten wird, werden nur Norm-Werte nach dem 100. Lernschritt betrachtet.

```
1   % decrease epsilon by factor 0.9
2   % if current norm is the smallest norm until now
3   % AND if at least 100 learning-cycles have been finished
4   lenNorm = length(normValues);
5   lastNorm = normValues(lenNorm);
6   if (lastNorm < minOfNormValues)
7     minOfNormValues = lastNorm;
8     if (lenNorm > 100)
9       epsilon = epsilon * 0.9
10      fprintf('decrease because of new min. epsilon : %d\n', normValues(lenNorm))
    ;
11    end;
12  end
13
```

Listing 7: Das Listing zeigt einen Auszug aus main.m

Da auch diese Änderung nur geringe Verbesserungen herbeiführte, wurde das Internet nach weiteren Verbesserungsansätzen durchsucht. Hierbei folgten wir einer Empfehlung das Einstellen der Lernphase nicht direkt vom Fehler oder von der Zeit abhängig zu machen, sondern schlicht nach einer gewissen Anzahl an Lernzyklen die Lernrate um einen bestimmten Faktor zu verkleinern [1].
Aus diesem Grund wurde die Funktion autoAdjustEpsilon soweit erweitert, dass sie alle 25000 Lernzyklen die Lernrate um einen bestimmten Faktor verringert.

```
1
2   % decrease epsilon by factor 0.5, always when 25.000 learn-cycles are
3   % finished
4   epsilon_regulation_steps = 25000;
5   if (mod(lenNorm,epsilon_regulation_steps) == 0)
6    epsilon = epsilon * 0.5
7    fprintf('decrease because of time. epsilon : %d\n', normValues(lenNorm));
8   end;
9
10
```

Listing 8: Das Listing zeigt einen Auszug aus autoAdjustEpsilon.m

[1] http://www.informatikseite.de/neuro/node24.php#SECTION00066600000000000000

5.3 Wahl der Dimensionl des neuronalen Netzes

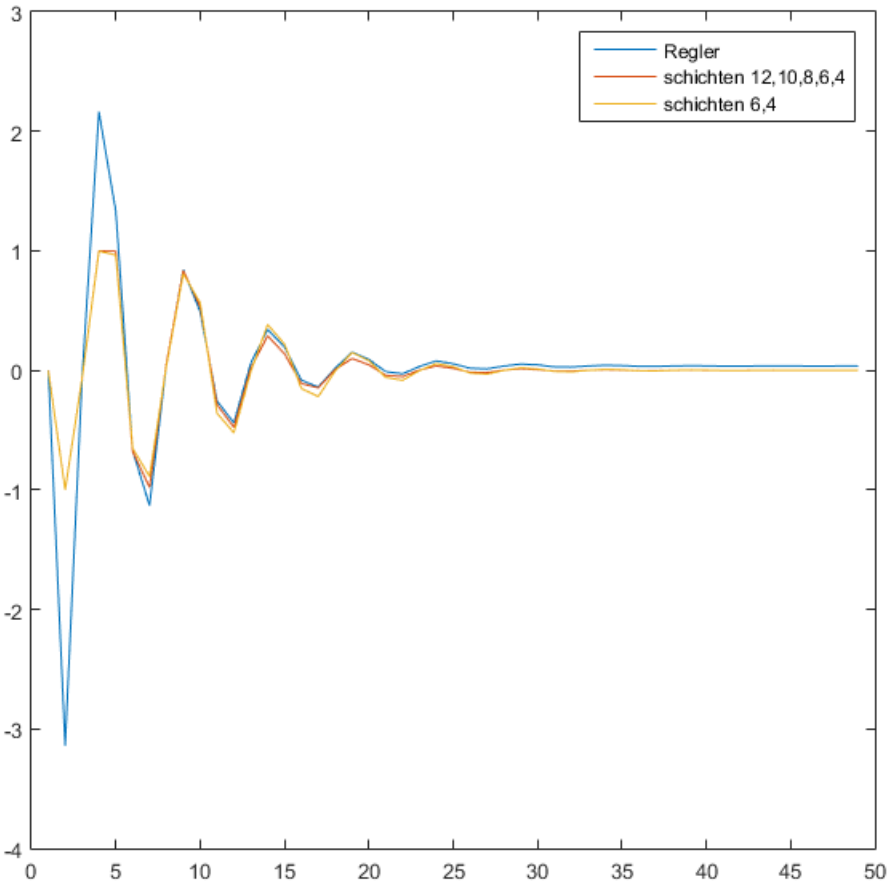


Abbildung 14: Vergleich 2-schichten / 5-schichten

Nach einigen Tests mit Neuronalen Netzen, die über mehr als zwei interne Schichten verfügten wurde festgestellt, dass die Verwendung von mehr als zwei Schichten nur kaum wahrnehmbare Verbesserungen, bei deutlich erhöhter Rechenzeit erbringt. Deshalb wurde im Endeffekt das am Anfang erstellte Netz mit zwei Schichten, welche sechs und vier Neuronen enthalten verwendet.

6 Ergebnisse der Simulation

6.1 Vergleich des neuronalen Netzes zur Reglerkennlinie

Beim Vergleich der Reglerkennlinie mit den Kennlinien des entwickelten neuronalen Netzes fällt auf, dass die vom neuronalen Netz erzeugten Kennlinien nicht vollständig mit denen des Reglers übereinstimmen, aber mit zunehmender Lerntiefe sich immer mehr an die Werte des Reglers annähern. Hier zeigt sich, dass die Güte des Netzes mit zunehmender Lerntiefe zwar steigt, jedoch bei hohen Lerntiefen die erzielte Verbesserung mit jedem neuen Lernzyklus kleiner wird. Hier muss ein möglichst guter Kompromiss zwischen der für das Lernen verwendeten Rechenzeit und den damit erzielten Ergebnissen gefunden werden. Bei den durchgeführten Tests zeigte sich, dass bei Lerntiefen von mehr als 500.000 Zyklen, keine sichtbaren Verbesserungen gegenüber kleineren Zyklenzahlen erreicht werden konnten. Die Verwendung von dynamischen Lernraten und einer Skalierung der vom Netz erzeugten Werte erbrachten nur geringe Verbesserung der erzeugten Kennlinien. Hauptsichlich sind Verbesserungen bei der Approximation der Regelerwerte über 1 und unter -1 unter Verwendung der Skalierung zu erkennen.
Da dies auf der anderen Seite jedoch zur Verschlechterung der Approximation von Regelerwerten um 0 herum führt, ist hier noch eine Verbesserung der Skalierung nötig um wirklich bessere Ergebnisse zu erzielen.
Das automatische Anpassen der Lernrate führte ebenso nur zu sehr geringen Verbesserungen. Auch hier ist eine weitere Optimierung des Algorithmus nötig um bessere Ergebnisse zu erzielen.
Insgesamt liefert das neuronale Netzt jedoch, auch wenn die Kennlinie sich sichtbar von der des Reglers unterscheidet, für die Simulation eine durchaus gut verwertbare Regelung.
Siehe hierzu auch den folgenden Abschnitt.

6.2 Ergebnisse in Simulink

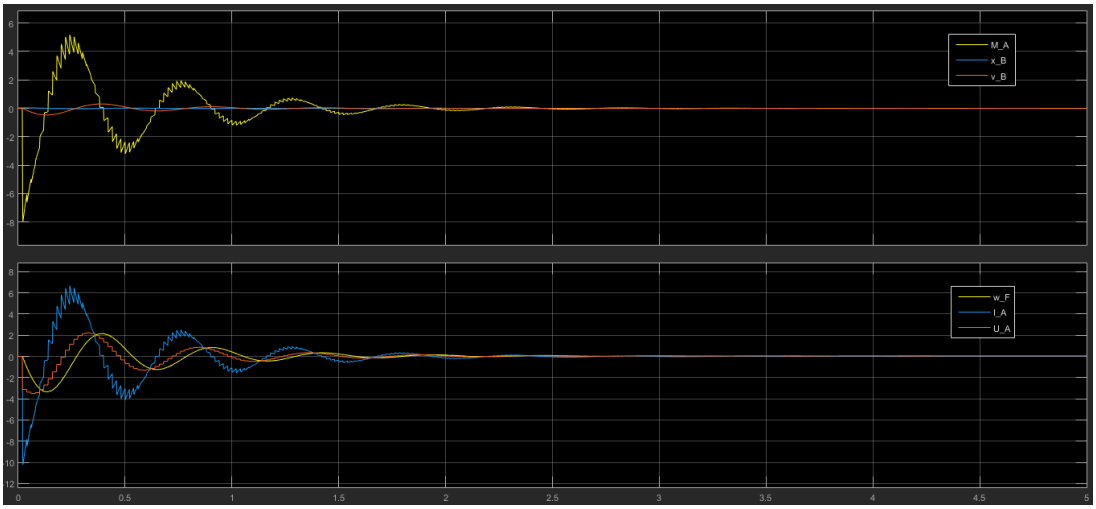


Abbildung 15: Kennlinien des Reglers

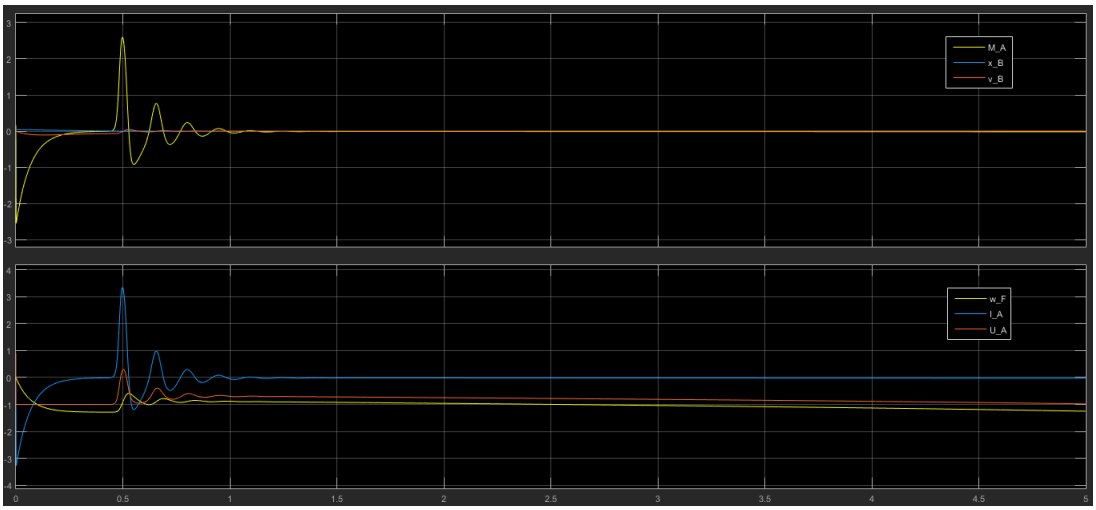


Abbildung 16: Kennlinien des KNN nach 500.000 Lernzyklen

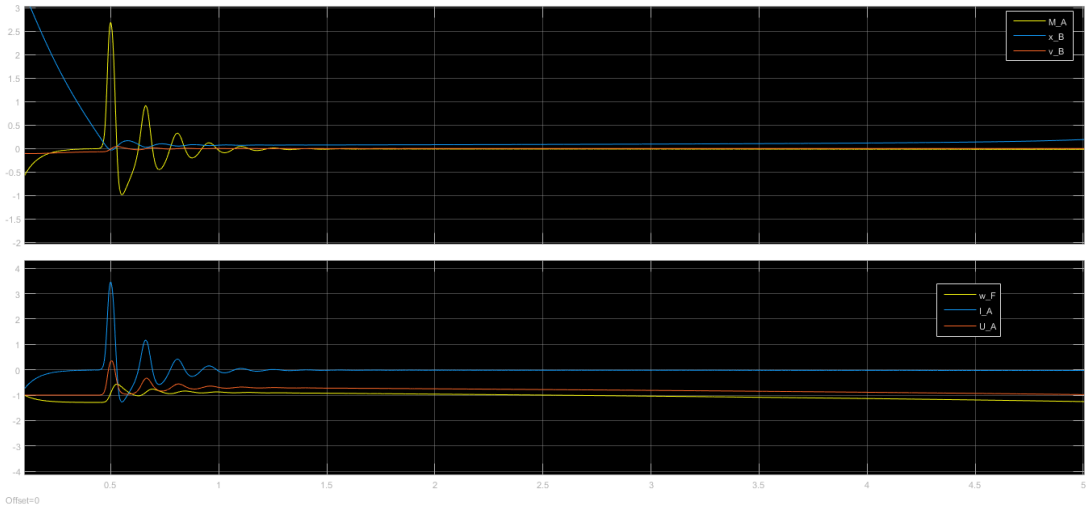


Abbildung 17: Kennlinien des KNN mit Position des Balls in cm

Trotz der erkennbaren Abweichungen der Kennlinien des KNN zu denen des Reglers ist in Abbildung 17 zu erkennen, dass auch das KNN die anfängliche Auslenkung des Balls von 4cm richtig korrigiert und ihn für die simulierte Dauer von fünf Sekunden auf der Felge hält.