Hochschule Karlsruhe
University of Applied Sciences
Fakultät für Elektro- und Informationstechnik

Labor Regelungstechnik

Laborversuch zur Drehzahlregelung

Laborbericht zum Versuch Nr. 3

Jan Hoegen* Maileen Schwenk†

3. Juni 2024

Abstract

Im dritten Regelungstechniklabor wird das Verhalten eines Elektromotors mithilfe von Matlab und SIMULINK betrachtet. Es wird ein stationär genauer Regelkreis als PI-Regler erarbeitet und sein Verhalten bei Überschwingern und zeitabhängigen Lastmomenten beobachtet.

1 Motor im Nennbetrieb

Die Gleichungen für den Gleichstrommotor wurden der Laboranleitung [1] entnommen.

$$\omega = 73.68 \frac{\mathrm{N\,m}}{\mathrm{A\,kg\,m^2}} \int i\,dt \tag{1}$$

$$i = \frac{1}{1 \cdot 10^8\,\mathrm{H}} \int u - 0.028 \frac{\mathrm{N\,m}}{\mathrm{A}} \omega - 5\Omega i\,dt \tag{2}$$

$$M = 0.028 \frac{\mathrm{N\,m}}{\mathrm{A}} i \tag{3}$$

In SIMULINK wird das zugehöirige Blockschaltbild als Subsystem entwickelt (siehe Abbildung 1) und anschließend an eine konstante Spannungsversorgung von 10 V angeschlossen. Der zeitliche Verlauf der Ausgangsgrößen ist in Abbildung 2 dargestellt. Die Zeitkonstante τ berechnet sich zu 2,3843 s.

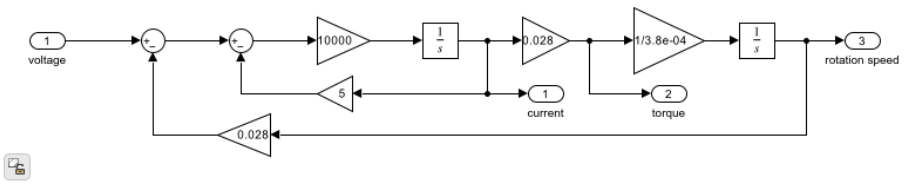


Abbildung 1: Blockschaltbild des Motor

*Matrikel-Nr. 82358. E-Mail hoja1028@h-ka.de
†Matrikel-Nr. 83802. E-Mail scma1315@h-ka-de

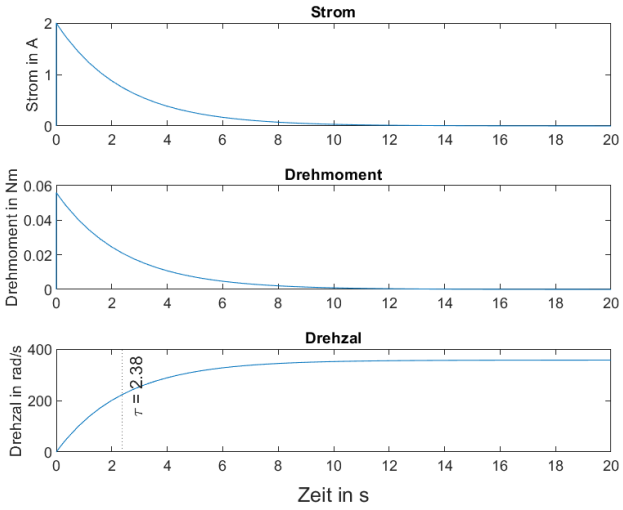


Abbildung 2: Motor im Nennbetrieb

2 Motorsteuerung mit P-Regler

Um die Drehzahl des Motors einzustellen wird ein P-Regler erstellt und an das Motorsubsystem angeschlossen (siehe Abbildung 3). Die Eingangsspannung am Motor wird mit einem Sättigungsblock in SIMULINK auf $U_{max} = \pm 10\,\mathrm{V}$ begrenzt. Die Verstärkung K_R des Reglers wird so eingestellt, dass der Sättigungsblock gerade nicht begrenzt bei einem Sollwert von $\omega_{soll} = 100\,\frac{\mathrm{rad}}{\mathrm{s}}$.

$$K_R = \frac{u_{max}}{\omega_{soll}} = \frac{10\,\mathrm{V}}{100\,\frac{\mathrm{rad}}{\mathrm{s}}} = 0{,}1\,\frac{\mathrm{V\,s}}{\mathrm{rad}} \tag{4}$$

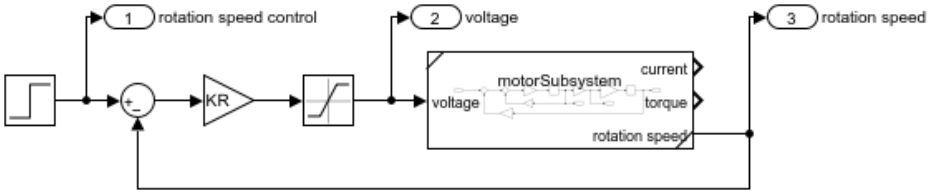


Abbildung 3: Blockschaltbild des P-Reglers

Der zeitliche Verlauf der Reglergrößen ist in Abbildung 4 gezeigt. Es ist zu erkennen, dass der Motor nur $\omega_{ist} = 78{,}1\,\frac{\mathrm{rad}}{\mathrm{s}}$ erreicht und damit nicht stationär genau ist. Das ist darin begründet, dass in diesem Zustand die Eingangsspannung am Motor

$$U = K_R \cdot (\omega_{soll} - \omega_{ist}) = 0{,}1\,\frac{\mathrm{V\,s}}{\mathrm{rad}} \cdot \left(100\,\frac{\mathrm{rad}}{\mathrm{s}} - 78{,}1\,\frac{\mathrm{rad}}{\mathrm{s}}\right) = 2{,}19\,\mathrm{V} \tag{5}$$

beträgt und diese Spannung wiederum zu einer Drehzahl von $\omega_{ist} = 78{,}1\,\frac{\mathrm{rad}}{\mathrm{s}}$ am Motor führt. Der Aussteuerbereich des Reglers ist also nicht groß genug.

3 Motorsteuerung mit PI-Regler

Der Regler aus dem vorherigen Abschnitt wird um einen Integrator erweitert (Abbildung 5). Die beste Nachstellzeit T_N, bei der gerade so kein Überschwinger für $\omega_{soll} = 100\,\frac{\mathrm{rad}}{\mathrm{s}}$ entsteht, wird mit einer ausgelagerten

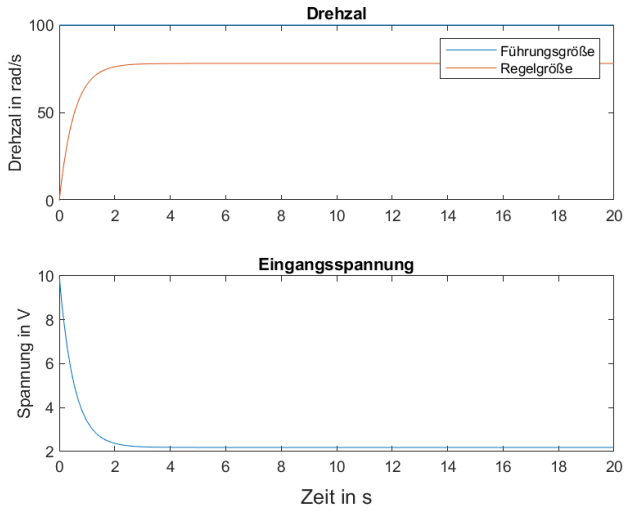


Abbildung 4: Reglergrößen des P-Reglers

Funktion ermittelt (siehe Anhang A). Dabei wird das Modell so lange mit veränderter Nachstellzeit aufgerufen, bis der Maximalwert der Reglergröße kleiner ist als der Sollwert. Es ergibt sich ein Wert von $T_N = 2{,}43\,\text{s}$

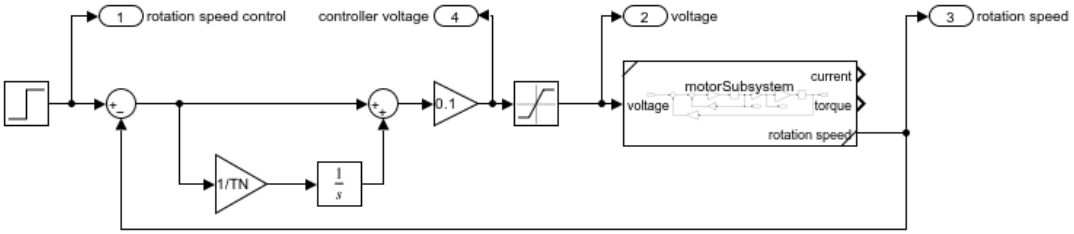


Abbildung 5: Blockschaltbild des PI-Reglers

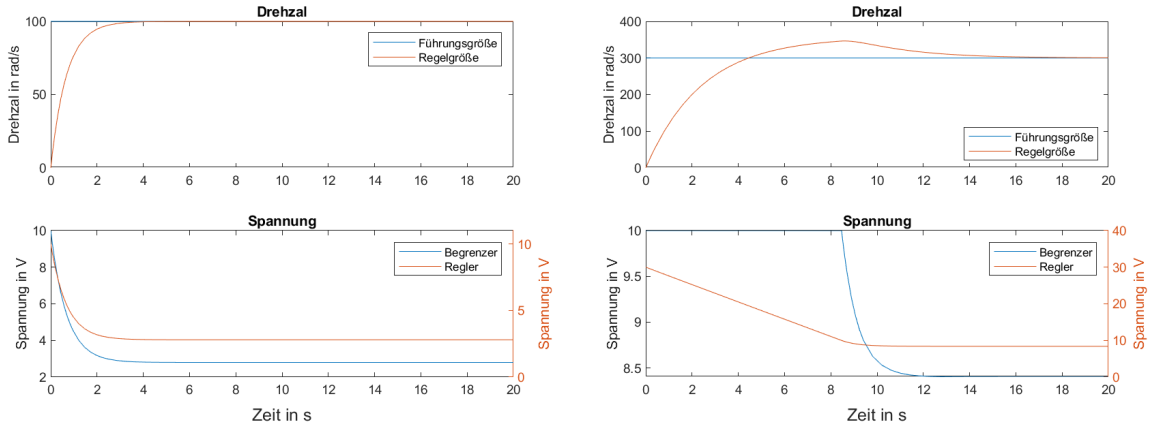
Der zeitliche Verlauf der Reglergrößen ist in Abbildung 6a dargestellt. Zusätzlich wird die Spannung vor und nach dem Sättigungsblock betrachtet. Wird die Führungsgröße auf $\omega_{soll} = 300\,\frac{\text{rad}}{\text{s}}$ angehoben (Abbildung 6b), kommt es zu Überschwingern. Das liegt daran, dass in der Zeit, in welcher der Begrenzer aktiv ist, der I-Anteil des Reglers weiter die Fehlerdifferenz aufsummiert. Wird der gewünschte Wert der Regelgröße erreicht, wird der hohe I-Anteil durch Überschwinger abgebaut.

4 Motorregelung mit veränderlichen Lastmoment

Nun wird ein Lastmoment, dass der Drehbewegung entgegenwirkt, nach 10 s hinzugeschaltet. Das erweiterte Blockschaltbild ist in Abbildung 7 zu finden, das Diagramm der Reglergrößen in Abbildung 8. Es wurde zusätzlich das Drehmoment des Motors und das Drehmoment am Ausgang aufgezeichnet.

5 Literatur

[1] F. Keller, *Labor Regelungstechnik, Laborversuch: Drehzahlregelung*, Karlsruhe: Hochschule Karlsruhe, 7. März 2024.



(a) Sollwert $\omega_{soll} = 100\ \frac{\text{rad}}{\text{s}}$

(b) Sollwert $\omega_{soll} = 300\ \frac{\text{rad}}{\text{s}}$

Abbildung 6: Reglergrößen des P-Reglers

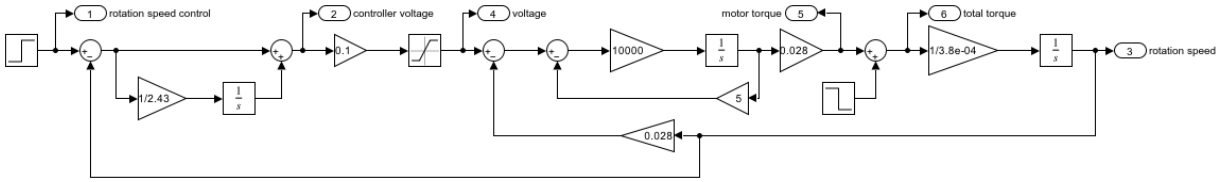


Abbildung 7: Blockschaltbild des PI-Reglers mit Lastmoment

6 Autorenbeiträge

Maileen Schwenk und Jan Hoegen erstellten die Vorbereitung und Messauswertung. Jan Hoegen schrieb den Bericht.

7 Verfügbarkeit des Codes

Der Code zum Auswerten der Daten und Erstellen der Diagramme findet sich unter https://github.com/JaxRaffnix/Regelungstechnik. Ebenfalls ist hier der Code zum Erstellen dieser Ausarbeitung hinterlegt.

A MATLAB-Code zur Modellauswertung

../versuch3/plotMotorModel.m

```
addpath('../functions/')

%_____________________________________________________________________
% run simulation
model = 'motorModel';
load_system(model);

output = sim(model);

%_____________________________________________________________________
% find the time constant value
```

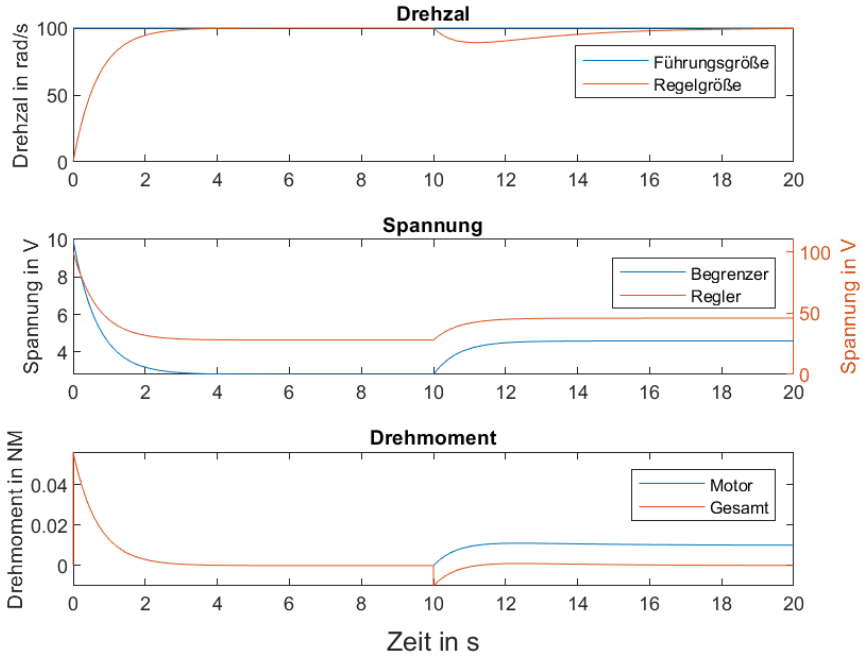


Abbildung 8: Reglergrößen des PI-Reglers mit Lastmoment

```
index = TimeConstantIndex(output.yout{3}.Values.Data);  % =  2.3843
tau = output.tout(index)

%_____________________________________________________________________
% plotting
figure
motor_plot = tiledlayout('vertical');

nexttile
plot(output.tout, get(output.yout, 'current').Values.Data);
title('Strom');
ylabel('Strom in A')

nexttile
plot(output.tout, get(output.yout, 'torque').Values.Data);
title('Drehmoment');
ylabel('Drehmoment in Nm')

nexttile
plot(output.tout, get(output.yout, 'rotation speed').Values.Data);
title('Drehzal');
ylabel('Drehzal in rad/s')
xlabel(motor_plot, 'Zeit in s')
%plot time constant
xline(tau, ':', ['\tau = ' sprintf('%.2f', tau)])

saveas(motor_plot, "graphMotorModel.png");
saveas(get_param(model, 'Handle'), 'blockMotorModel.png')

load_system('motorSubsystem')
saveas(get_param('motorSubsystem', 'Handle'), 'blockMotor.png')
close_system('motorSubsytem')
```

```
save_system(model)
close_system(model);
```

../versuch3/plotPController.m

```
%____________________________________________________________________
% run simulation
model = 'pController';
load_system(model);

output = sim(model);

%____________________________________________________________________
% plotting
figure
motor_plot = tiledlayout('vertical');

nexttile
plot(output.tout, get(output.yout, 'rotation speed control').Values.Data);
hold on
plot(output.tout, get(output.yout, 'rotation speed').Values.Data);
title('Drehzal');
ylabel('Drehzal in rad/s')
legend('Führungsgröße', 'Regelgröße');

nexttile
plot(output.tout, get(output.yout, 'voltage').Values.Data);
title('Eingangsspannung');
ylabel('Spannung in V')
xlabel(motor_plot, 'Zeit in s')

saveas(motor_plot, "graphPController.png");
saveas(get_param(model, 'Handle'), 'blockPController.png')

save_system(model)
close_system(model);
```

../versuch3/plotPiController.m

```
%____________________________________________________________________
% find optimum TN and run simulation

addpath('../functions/')

model = 'piController';
rotationSpeedControl = 100;

bestTN = BestResetTime(model, 'TN', 'rotation speed', 2.4, 0.01)
% =  2.43

output = sim(model);

%____________________________________________________________________
% plotting
figure
motor_plot = tiledlayout('vertical');

nexttile
plot(output.tout, get(output.yout, 'rotation speed control').Values.Data);
hold on
plot(output.tout, get(output.yout, 'rotation speed').Values.Data);
title('Drehzal');
ylabel('Drehzal in rad/s')
legend('Führungsgröße', 'Regelgröße');
```

```
nexttile
plot(output.tout, get(output.yout, 'voltage').Values.Data);
hold on
ylabel('Spannung in V')
yyaxis right
plot(output.tout, get(output.yout, 'controller voltage').Values.Data);
title('Spannung');
ylabel('Spannung in V')
xlabel(motor_plot, 'Zeit in s')
legend('Begrenzer', 'Regler');

saveas(motor_plot, "graphPiController.png")

%____________________________________________________________________
% plot with new control variable

rotationSpeedControl = 300;
output = sim(model);

figure
motor_plot_new = tiledlayout('vertical');

nexttile
plot(output.tout, get(output.yout, 'rotation speed control').Values.Data);
hold on
plot(output.tout, get(output.yout, 'rotation speed').Values.Data);
title('Drehzal');
ylabel('Drehzal in rad/s')
lgd = legend('Führungsgröße', 'Regelgröße');
lgd.Location = 'best';

nexttile
plot(output.tout, get(output.yout, 'voltage').Values.Data);
ylabel('Spannung in V')
hold on
yyaxis right
plot(output.tout, get(output.yout, 'controller voltage').Values.Data);
title('Spannung');
ylabel('Spannung in V')
legend('Begrenzer', 'Regler');

xlabel(motor_plot_new, 'Zeit in s')

saveas(motor_plot_new, "graphPiControllerNew.png");

saveas(get_param(model, 'Handle'), 'blockPiController.png')

% save_system(model);
close_system(model);
```

../versuch3/plotPiControllerLoad.m

```
%____________________________________________________________________
% find optimum TN and run simulation

% addpath('../functions/')

model = 'piControllerLoad';
load_system(model);
% rotationSpeedControl = 100;

output = sim(model);

%____________________________________________________________________
% plotting
```

```
figure
load_plot = tiledlayout('vertical');

nexttile
plot(output.tout, get(output.yout, 'rotation speed control').Values.Data);
hold on
plot(output.tout, get(output.yout, 'rotation speed').Values.Data);
title('Drehzal');
ylabel('Drehzal in rad/s')
legend('Führungsgröße', 'Regelgröße');

nexttile
plot(output.tout, get(output.yout, 'voltage').Values.Data);
hold on
ylabel('Spannung in V')
yyaxis right
plot(output.tout, get(output.yout, 'controller voltage').Values.Data);
title('Spannung');
ylabel('Spannung in V')
xlabel(load_plot, 'Zeit in s')
legend('Begrenzer', 'Regler');

nexttile
plot(output.tout, get(output.yout, 'motor torque').Values.Data);
hold on
% yyaxis right
plot(output.tout, get(output.yout, 'total torque').Values.Data);
title('Drehmoment');
ylabel('Drehmoment in NM')
xlabel(load_plot, 'Zeit in s')
legend('Motor','Gesamt');

saveas(load_plot, "graphPiControllerLoad.png");
saveas(get_param(model, 'Handle'), 'blockPiControllerLoad.png')

% save_system(model);
close_system(model);
```

../functions/BestResetTime.m

```
function bestResetTime = BestResetTime(model, resetTimeName, variableName, initialGuess,
    stepSize)
    % Goal: Find the best reset time for a given Simulink model to ensure there is no
        overshoot a variable.
    % CAVE: - initalGuess has to result in an overshoot.
    %       - variable has to be linked to an outport.
    %       - After the function execution, the model is closed.
    %       - steady state at the last data points is assumed.

    validateattributes(model, {'char', 'string'}, {'nonempty'}, mfilename, 'model', 1);
    validateattributes(resetTimeName, {'char', 'string'}, {'nonempty'}, mfilename, '
        resetTimeName', 2);
    validateattributes(variableName, {'char', 'string'}, {'nonempty'}, mfilename, '
        variableName', 3);
    validateattributes(initialGuess, {'numeric'}, {'scalar', 'positive'}, mfilename, '
        initialGuess', 4);
    validateattributes(stepSize, {'numeric'}, {'scalar', 'positive'}, mfilename, 'stepSize',
        5);

    % get current data
    load_system(model);
    assignin('base', resetTimeName, initialGuess)
    output = sim(model);
    data = get(output.yout, variableName).Values.Data;
    threshold = data(end);
```

```
    % Check if initial guess causes overshoot
    if max(data) < threshold
        error('No overshoot for initial guess %.4f. Decrease initialGuess.', initialGuess);
    end

    % Loop until no overshoot compared to threshold
    TN = initialGuess;
    maxValue = inf;
    while maxValue > threshold
        TN = TN + stepSize;
        if TN <= 0  % Check if TN is non-positive to avoid infinite loop
            error('TN %.4f has become negative or 0. Stopping the simulation.', TN);
        end
        assignin('base', resetTimeName, TN)

        output = sim(model);
        data = get(output.yout, variableName).Values.Data;
        maxValue = max(data);
        fprintf('Testing reset time: %.4f, Max value: %.4f\n', TN, maxValue);
    end

    close_system(model, 0);
    bestResetTime = TN;
    fprintf('Best reset time found: %.4f\n', bestResetTime);
end
```

../functions/TimeConstantIndex.m

```
function index = TimeConstantIndex(data)
    % GOAL: find the index number of the time constant value in a list

    % Check if the data is not empty
    if isempty(data)
        error('Input data is empty.');
    elseif ~isnumeric(data) || ~isvector(data)
        error('Input data must be a numeric vector.');
    end

    % check if the data is increasing or decreasing and set the offset
    diffs = diff(data);

    % Ensure the data is strictly monotonic
    if all(diffs >= 0)
        is_rising = true;
    elseif all(diffs <= 0)
        is_rising = false;
    else
        error('Input data must be monotonic increasing or decreasing.');
    end

    OFFSET = 0.632;
    if is_rising == false
        OFFSET = 1 - OFFSET;
    end

    % find the index of the time constant
    max_value = max(data);
    value_tau = max_value * OFFSET;
    [~, index] = min(abs(data - value_tau ));
end
```