Technische Universität Braunschweig

Bachelorarbeit

Entwicklung und Aufbau einer Regelung zum Spannen von Werkstücken mittels Adhäsionskräften in der Mikrobearbeitung

von Lukas Bommes

Matrikelnr.: 4367361

Technische Universität Braunschweig
Fakultät für Maschinenbau

Institut für Werkzeugmaschinen und Fertigungstechnik

Prüfer: Dr.-Ing. H.-W. Hoffmeister
Betreuer: Dipl.-Ing. Tarek Tounsi

Eingereicht am 06. Juli 2016

Bachelorarbeit

18. April 2016

für

Herr cand. B. Sc. Lukas Bommes

Studiengang : Bachelor Maschinenbau

Matr. Nr.: 4 367 361

Thema: **„Entwicklung und Aufbau einer Regelung zum Spannen von Werkstücken mittels Adhäsionskräften in der Mikrobearbeitung“**

Heutige Werkzeugmaschinen für die Mikrozerspanung sind in ihren Abmessungen bezogen auf das zu fertigende Werkstück bzw. die herzustellenden Strukturen unverhältnismäßig groß. Die Reduzierung der Maschinenabmessungen erlaubt deutliche Einsparungen an Energie, Kosten und Platzbedarf. Darüber hinaus fehlt ein modularer Aufbau der Werkzeugmaschinen, um diese ad hoc an wechselnde Fertigungsaufgaben anpassen zu können. Die Vision dahinter ist ein Baukastenprinzip, das den schnellen und einfachen Aufbau einer flexiblen Fertigungslinie ermöglicht. Für die Erschließung dieser Potentiale wurde im Rahmen des DFG-Schwerpunktprogramms 1476 „Kleine Werkzeugmaschinen für kleine Werkstücke“ ein neuartiges Maschinenkonzept entwickelt und vorgestellt, das auf einer inversen und kooperativen Achsbewegung von Werkzeug und Werkstück basiert. Mit diesem Gestaltungsansatz konnte eine insgesamt kompaktere Maschinenbauweise erreicht werden.

Die Modularität der Werkzeugmaschine wird über ein würfelförmiges Rahmengestell in Form eines „Production Cube Module“ (PCM) erreicht. Mit Hilfe von Adapterplatten können verschiedene Maschinenkomponenten wie beispielsweise Messvorrichtungen, Werkzeuge oder Werkstückspanneinheiten rekonfigurierbar eingewechselt werden. Im Rahmen dieser studentischen Arbeit soll eine Regelung für ein bestehendes Werkstückspannsystem entwickelt und aufgebaut werden. Bei diesem System werden die Werkstücke mittels Adhäsionskräfte gespannt. Als Spannmedium kann entweder Wasser (Gefrierspannen) oder Wachs (Wachsspannen) eingesetzt werden. Je nach Medium muss die Regelung dafür sorgen, dass das Spannmedium für den Ein- bzw. Ausspannvorgang verflüssigt und für den eigentlichen Bearbeitungsprozess verfestigt wird.

Im Rahmen der Arbeit sind von Herrn Lukas Bommes folgende Teilaufgaben auszuführen:

- Entwicklung eines Gesamtmodells des zu regelnden Systems. Die zugrundeliegenden regelungstechnischen Grundlagen werden ausführlich erläutert.
- Entwurf und Optimierung eines geeigneten Reglers. Dieser soll anschließend auf dem vorhandenen Mikrocontroller in dem Werkstückspannsystem implementiert werden.
- Das Werkstückspannsystem soll hinsichtlich der Eigenschaften Ein- und Ausspannzeiten, minimale Spanntemperaturen und seiner Eignung für Minimalmengenschmierung experimentell charakterisiert werden.

- Entwicklung einer übergeordneten Steuerungssoftware für das Werkstückspannsystem.
  - Erstellung eines Protokolls zur Ansteuerung und Kommunikation über eine serielle Schnittstelle.
  - Das System soll die Möglichkeit bieten, die Reglerparameter über die Kommunikationsschnittstelle anzupassen. Dazu bietet es sich an, mehrere Profile anzulegen, die ausgewählt, editiert und wieder auf „Werkseinstellungen" zurückgesetzt werden können.
  - Das Werkstückspannsystem besitzt mehrere eingebaute Temperatursensoren. Die Steuerungssoftware soll es einem angeschlossenen PC ermöglichen, die Temperaturen live darstellen und abspeichern zu können.
  - Anfertigung einer ausführlichen Dokumentation der implementierten Steuerungsbefehle.

Das Institut für Werkzeugmaschinen und Fertigungstechnik ist bereit, zur Durchführung obiger Bachelorarbeit institutseigenes Know-how zur Verfügung zu stellen (Zeichnungen, Berechnungen, Software, Unterlagen etc.), an dessen weiterer Geheimhaltung ein berechtigtes Interesse besteht. Voraussetzung hierfür ist die durch die nachfolgende Unterschrift des Verfassers anerkannte Verpflichtung des Verfassers, eine Veröffentlichung und /oder Verwertung des Gegenstandes obiger Bachelorarbeit oder aber Teilen hiervon nur im Rahmen einer vorherigen schriftlichen Vereinbarung mit dem oben genannten Institut vorzunehmen. Weiter verpflichtet sich der Verfasser mit seiner Unterschrift die Arbeit vor der Abgabe für die Plagiatskontrolle zur Verfügung zu stellen.

1. Prüfer Dr.-Ing. Hans-Werner Hoffmeister

Student

☐ Diese Arbeit **mit Sperrvermerk** ist vertraulich zu behandeln.

☒ Kein **Sperrvermerk**

Art der Arbeit:	experimentell
Dauer der Arbeit:	300 Arbeitsstunden bzw. 3 Monate Bearbeitungszeit
Betreuer:	Dipl.-Ing. Tarek Tounsi
IWF-Kennung:	BA 12/16
Ausgabedatum:	19.04.2016
Abgabedatum:	

Technische Universität
Braunschweig
Institut für Werkzeugmaschinen und Fertigungstechnik

Leitung:
Prof. Dr.-Ing. Klaus Dröder
Prof. Dr.-Ing. Christoph Herrmann

Langer Kamp 19 b
38106 Braunschweig
Deutschland

Tel. +49 (0) 531 391-7600
Fax +49 (0) 531 391-5842
k.droeder@tu-bs.de
www.iwf.tu-bs.de
Tel. +49 (0) 531 391-7601(Sekretariat)

18. April 2016

Begleitbogen Geheimhaltung - zur Anmeldung einer Studentischen Arbeit

Titel der Arbeit:

„Entwicklung und Aufbau einer Regelung zum Spannen von Werkstücken mittels Adhäsionskräften in der Mikrobearbeitung“

Bitte vor Anmeldung einer studentischen Arbeit den Titel oben eintragen und die Art der Geheimhaltung ankreuzen und anschließend unterschreiben.

☒ Kein Sperrvermerk

☐ **Sperrvermerk**

- ☐ Schriftliche Vereinbarung zwischen dem IWF und dem beteiligtem Forschungspartner /Firma
  (Original mitbringen ⇨ Archivordner, Kopie ⇨ stud. Arbeit)
- ☐ Projekt mit Geheimhaltungsklausel wurde beantragt.
- ☐ Mündliche Absprache mit Forschungspartner.

☐ Student wurde über Sperrvermerk informiert.

19.04.2016

Datum, Unterschrift Betreuer der Studentischen Arbeit

Abstract

Ziel der Arbeit ist die Entwicklung eines Software-Systems zur Ansteuerung einer thermischen Spanneinheit für die Mikrobearbeitung. Die Software umfasst einen digitalen PI-Regler als Temperaturregler, die Steuerlogik, eine graphische Nutzeroberfläche und eine Schnittstelle zum bidirektionalen Datenaustausch zwischen Spannsystem und der auf einem externen PC ausgeführten GUI. Nach einer Einführung in die Grundlagen der Regelungstechnik wird ein Modell der Regelstrecke durch experimentelle Analyse des Ein- und Ausgangsverhaltens entwickelt. Auf dessen Basis werden ein Temperaturregler entworfen, die Reglerparameter eingestellt und das Verhalten des geschlossenen Regelkreises simuliert sowie der Regler auf einem Mikrocontroller implementiert. Anschließend werden die graphische Nutzeroberfläche und die übergeordnete Steuerungssoftware entwickelt. Methodisch wiederholen sich im Rahmen dieser Arbeit insbesondere die mathematische Modellierung, der Entwurf auf Basis des Modells und die Simulation sowie Validierung des Modells anhand von Messdaten. Die vorliegende Bachelorarbeit richtet sich vor allem an diejenigen, die in Zukunft an der Weiterentwicklung des flexiblen Mikrofertigungssystems, in dem die Spanneinheit zum Einsatz kommt, arbeiten werden sowie an alle, die Interesse an Regelungstechnik und an elektronischen und informationsverarbeitenden Systemen besitzen.

The aim of this paper is the development of a software system for controlling a thermal clamping unit for micromachining. The software includes a digital PI-controller as temperature controller, the control logic, a graphical user interface and an interface for bidirectional data transfer between the clamping system and the GUI, running on an external PC. After an introduction to the fundamentals of control systems engineering a model of the controlled system is developed through experimental analysis of the input and output behaviour. Based on this model a temperature controller is designed, the controller parameters are tuned, the behaviour of the closed loop is simulated and the controller is implemented on a microcontroller. Afterwards the graphical user interface and the superior control software are developed. Methodically, especially mathematical modelling, the design on the basis of that model and simulation as well as validation of the model with the aid of measured data recur within this paper. The present bachelor thesis adresses particularly those who will work on the further development of the mikromachining system in which the clamping system is integrated as well as everybody who is interested in control systems engineering and in electronic and information processing systems.

Schlüsselwörter: Regelungstechnik, Regler, Temperaturregler, Reglerentwurf, digitaler Regler, PID-Regler, Spannsystem, Modellierung, Simulation, Validierung, Softwareentwicklung, MATLAB, Simulink, Arduino, Mikrocontroller, Peltier-Element, Elektronik, Mikrotechnik, Fertigungstechnik

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ADC	Analog-Digital-Wandler
ASCII	American Standard Code for Information Interchange
CAD	Rechnerunterstütztes Design
COM-Port	Communication-Port, Serielle Schnittstelle
CPU	Zentrale Recheneinheit
D-	Differential-
DAC	Digital-Analog-Wandler
FET	Feldeffekt-Transistor
FTDI	Future Technology Devices International Ltd., Bauteilhersteller
GND	Massepotential
GUI	Graphische Nutzeroberfläche
H-Brücke	Transistor-Vollbrücke
I-	Integral-
IC	Integrierter Schaltkreis
ISR-Routine	Interrupt Service Routine
LC-Filter	Tiefpassfilter aus Kapazität und Induktivität
MATLAB	Matrix Laboratory, Software zur numerischen Berechnung
MIMO	Multiple Input Multiple Output, Mehrgrößensystem
n-dotiert	negativ-dotiert
N-Kanal	Negativ-Kanal
P-	Proportional-
p-dotiert	positiv-dotiert
P-Kanal	Positiv-Kanal
PC	Personalcomputer
PID-Regler	Propotional-Integral-Differential-Regler
PWM	Pulsweitenmodulation
RS-232	Serielle Schnittstelle
SISO	Single Input Single Output, Eingrößensystem
SMD	Surface Mounted Device
TEC	Thermoelektrisches Element, Peltier-Element
UART	Universal Asynchronous Receiver Transmitter, Schnittstelle
USB	Universal Serial Bus, Schnittstelle
µC	Mikrocontroller
VCC, VIN	Versorgungsspannung
VCP	Virtueller COM-Port Treiber
VReg	Spannungsregler
WOK	Wurzelortskurve

1 Einleitung

Der Einsatz dünner Filme zum Spannen von Werkstücken mittels Adhäsionskräften, ist ein vielversprechender Ansatz zur Miniaturisierung von Fertigungsmaschinen in der Mikrobearbeitung. Als Medien kommen, wie in [27] ausführlich betrachtet, beispielsweise Wachs oder Wasser in Frage. Um das Werkstück zuverlässig spannen und lösen zu können, ist eine Änderung des Aggregatzustandes des Spannmediums durch Erhitzen und Abkühlen der Spannfläche erforderlich. Demzufolge müssen je nach Zustand des Systems zum Spannen und Lösen vorgegebene Sollwertverläufe der Spannflächentemperatur angefahren werden. Da das System zusätzlich Störungen, beispielsweise durch den Einsatz von Kühlschmierstoffen oder im Prozess entstehender Wärme, ausgesetzt ist, ist eine Regelung der Temperaturen erforderlich. In Hinblick auf eine hohe Flexibilität ist es wünschenswert, diese Regelung in Form einer Software zu realisieren. Diese Software soll unter anderem auch zur komfortablen Ansteuerung und Diagnose des Spannsystems mithilfe eines PCs dienen. Der Benutzer soll über die Software ein Werkstück einspannen und lösen können und dabei stets über den aktuellen Zustand des Systems informiert werden. Dieser umfasst unter anderem die im System gemessenen Temperaturen, die innerhalb der Software graphisch dargestellt und in einer Datei abgespeichert werden können. Eine Anpassung wichtiger Systemparameter und Abspeichern von Parametersätzen in Profilen sowie Laden von Profilen soll ebenfalls über die Software möglich sein.

Gegenstand dieser Arbeit werden daher der Entwurf und die Implementierung eines Software-Systems bestehend aus einer Firmware, die auf dem Mikrocontroller der Spanneinheit ausgeführt wird, und einer graphischen Anwendungssoftware zur übergeordneten Ansteuerung des Spannsystems sein. Zusätzlich wird eine Schnittstelle zum Datenaustausch zwischen Spannsystem und PC entwickelt. Ein weiterer Schwerpunkt der Arbeit ist der Entwurf und die Implementierung eines geeigneten Temperaturreglers, der auf einem ebenfalls im Rahmen dieser Arbeit erstellten mathematischen Modell des Spannsystems basiert. Die Modellbildung erfolgt dabei größtenteils auf experimentellem Weg durch Messung des Ein- und Ausgangsverhaltens der Regelstrecke.

Modellbildung, Reglerentwurf und Simulation des Regelkreises erfolgen in MATLAB/Simulink mithilfe der in der Control System Toolbox [31] bereitgestellten Werkzeuge. Für die Implementierung der graphischen Anwendungssoftware kommt der ebenfalls in MATLAB integrierte GUI-Editor GUIDE zum Einsatz. Die Mikrocontroller-Firmware wird weitgehend in C programmiert und greift auf die umfangreichen Funktionen der Arduino-Bibliothek zurück.

Eine thematische Abgrenzung der Arbeit muss in Hinblick auf die im Spannsystem enthaltene Steuerelektronik vorgenommen werden. Diese sollte neben dem thermischen Teil des Spannsystems bereits im Rahmen einer vorangegangenen Arbeit [1] entwickelt und aufgebaut werden, was jedoch nicht vollständig gelungen ist. Um dennoch das Software-System entwickeln zu können, wird daher im Rahmen dieser Arbeit eine eigene Steuerschaltung realisiert. Deren Entwicklungsaufwand wird durch eine Vereinfachung des Schaltungsprinzips und Zusammensetzung der

Schaltung aus Zukaufteilen möglichst gering gehalten. Da im Rahmen einer anderen Arbeit parallel eine Steuerelektronik entwickelt wird, geht die vorliegende Arbeit nur kurz auf diesen Aspekt ein.

Die Arbeit selbst ist in vier Kapitel gegliedert. Zunächst wird im Grundlagenkapitel eine umfassende Einführung in die Regelungstechnik gegeben. Hieran schließt sich ein Kapitel über die Analyse und Modellbildung der Regelstrecke an. Nach der Modellierung werden im vierten Kapitel zunächst einige Grundlagen zur Regelung linearer Systeme erläutert und anschließend ein Temperaturregler entworfen, eingestellt und implementiert sowie das Verhalten des geschlossenen Regelkreises simuliert. Abschließend geht das fünfte Kapitel auf die Steuerungssoftware ein, wobei eine Untergliederung des Kapitels entsprechend der Teilmodule der Software vorgenommen wird.

2 Regelungstechnische Grundlagen

Das folgende Kapitel soll eine kurze Einführung in die Regelungstechnik bieten. Zunächst werden dazu ganz allgemein die Funktionsweise einer Regelung, die in einem Regelkreis anzutreffenden Bauteile sowie das Vorgehen beim Entwurf einer Regelung erläutert. Anschließend wird die Laplace-Transformation als wichtiges Werkzeug zur Behandlung regelungstechnischer Probleme und zur Darstellung von Systemen im sogenannten Bildbereich eingeführt. Weitere Inhalte dieses Kapitels sind unterschiedliche Darstellungsformen dynamischer Systeme im Zeit- und Bildbereich. Hier werden die Übertragungs-, Übergangs- und Gewichtsfunktion, die Pol-Nullstellen-Verteilung sowie schließlich der Frequenzgang und mit ihm die Ortskurve und das Bode-Diagramm eingeführt. Danach wird die Zustandsraumdarstellung nichtlinearer Systeme als wichtige Darstellungsform dynamischer Systeme behandelt und auf ihre Linearisierung eingegangen. Ein weiterer hier vorgestellter Aspekt ist die Stabilität von Übertragungssystemen. Nach einer Definition des Begriffs Stabilität werden die wichtigsten Kriterien zur Stabilitätsanalyse präsentiert. Den Abschluss dieser Einführung stellen zeitdiskrete Systeme dar. Es werden zunächst Grundlagen der Quantisierung und Abtastung sowie der Aufbau eines zeitdiskreten Regelkreises besprochen, bevor schließlich einige Methoden zur mathematischen Behandlung zeitdiskreter Systeme an die Hand gegeben werden. Zu guter Letzt wird auf den Entwurf zeitdiskreter Regler und die Stabilität zeitdiskreter Systeme eingegangen.

2.1 Funktionsweise einer Regelung

Im folgenden Abschnitt soll kurz auf die Funktionsweise eines Regelkreises und der darin enthaltenen Komponenten eingegangen werden.

Die DIN IEC 60050-351 definiert den Begriff der *Regelung* folgendermaßen:

> Die *Regelung* bzw. das *Regeln* ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst.

$y(t)$	Regelgröße (Istwert)	$z(t)$	Störgröße
$w(t)$	Führungsgröße (Sollwert)	$e(t)$	Regelabweichung
$u(t)$	Stellgröße	$y_m(t)$	gemessene Regelgröße

Abb. 2.1 zeigt die vereinfachte Darstellung eines Regelkreises und alle darin auftretenden Größen. Eine Regelung besteht dabei im Wesentlichen aus dem zu regelnden dynamischen System,

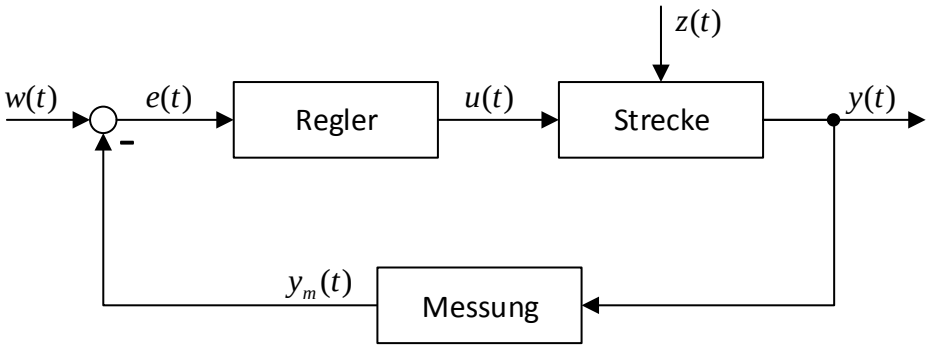


Abbildung 2.1: Blockschaltbild und Größen eines allgemeinen Regelkreises nach [8].

das hier als Strecke bezeichnet wird, einem weiteren dynamischen System, dem Regler, und einem Messglied zur Messung und Rückführung der Regelgröße $y(t)$. Eine Regelung kann zwei unterschiedliche Ziele verfolgen.

1) Einstellen der Regelgröße $y(t)$ auf die vorgegebene Führungsgröße $w(t)$, das heißt einen vorgegebenen Sollwert. Dies soll möglichst schnell und mit geringer Abweichung zwischen Regel- und Führungsgröße erfolgen. Eine solche Regelung wird als *Folgeregelung* bezeichnet.

2) Ausgleich von Störungen $z(t)$, die ungewollt auf die Regelstrecke und damit die Regelgröße $y(t)$ einwirken. Ziel der Regelung ist das möglichst schnelle Ausgleichen von Störungen und eine geringe Beeinflussung der Regelgröße durch Störungen. Eine derartige Regelung nennt sich *Festwertregelung*.

Der eigentliche Regelvorgang kann in drei Schritte zerlegt werden. So erfolgt zunächst eine Messung der Regelgröße $y(t)$ durch das Messglied, welches beispielsweise ein Temperatursensor sein kann. Dieses formt die physikalische Größe am Ausgang der Strecke, also zum Beispiel die Temperatur, in eine vom Regler verwertbare physikalische Größe, den Messwert $y_m(t)$, um. In der Regel ist die Messgröße ein elektrisches Signal. Im zweiten Schritt erfolgt ein Vergleich zwischen dem am Eingang der Regelung vorgegebenen Führungswert $w(t)$ und der gemessenen Regelgröße $y_m(t)$. Diese Größe wird als Regelabweichung $e(t) = w(t) - y(t)$ bezeichnet. Abschließend berechnet der Regler die Stellgröße $u(t)$ aus der an seinem Eingang anliegenden Regelabweichung. Die Stellgröße wirkt sich auf die Strecke aus und beeinflusst die Regelgröße am Ausgang der Strecke.

Der hier als Regler bezeichnete Block enthält, wie in Abb. 2.2 dargestellt, eine informationsverarbeitende Komponente in Form eines Algorithmus, der einen Zusammenhang zwischen Regelabweichung $e(t)$ und Stellgröße $u(t)$ herstellt, und ein Stellglied, das einen meist elektrischen Hilfsenergiestrom entsprechend der Stellgröße $u(t)$ stellt und in die für die Beeinflussung des Prozesses erforderliche Energieform umwandelt. Beispielhaft könnte der Energiesteller ein Leistungstransistor und der Energiewandler eine elektrische Heizung, die elektrische Energie in thermische Energie umwandelt, sein.

Der Regelalgorithmus ist nichts anderes als die mathematische Beschreibung des Zusammenhangs

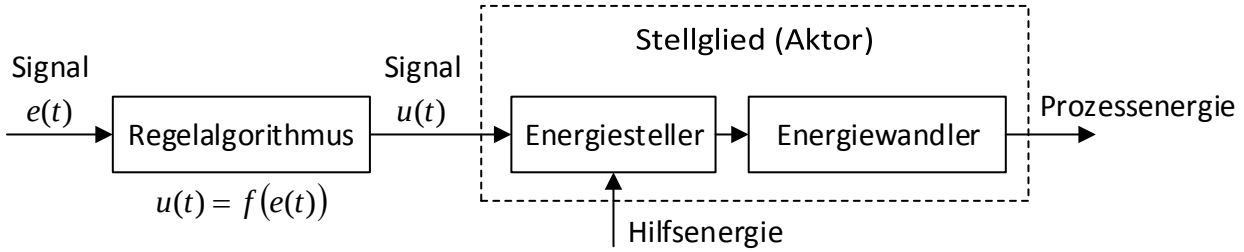


Abbildung 2.2: Detailaufbau eines Reglers. Dieser enthält eine informationsverarbeitende Komponente sowie ein Stellglied zur Beeinflussung des zu regelnden Prozesses.

zwischen Regelabweichung $e(t)$ und Stellgröße $u(t)$, das sogenannte Reglergesetz.

$$u(t) = f(e(t)) \tag{2.1}$$

Dieses kann eine beliebige Differentialgleichung sein, das heißt die Stellgröße kann sowohl von der Regelabweichung selbst (P-Regler) als auch von ihrem Differential nach der Zeit (D-Regler) sowie von ihrem zeitlichen Integral (I-Regler) abhängen. In der Praxis treten meist Kombinationen dieser drei elementaren Reglertypen auf, so beispielsweise beim PID-Regler, dessen Reglergesetz

$$u(t) = K_P\, e(t) + K_I \int_0^t e(\tau)\,\mathrm{d}\tau + K_D\, \frac{de(t)}{dt} \tag{2.2}$$

lautet. Er bezieht sowohl den aktuellen Wert der Regelabweichung als auch die vorangegangenen Abweichungen sowie die aktuelle Änderungsrate der Abweichung mit in die Berechnung der Stellgröße ein. Die Umsetzung des Reglergesetzes kann beispielsweise in Form einer Software auf einem Mikrocontroller erfolgen. Näheres zum Regler wird in Kapitel 4 erläutert. [8, 34]

2.2 Vorgehensweise bei der Lösung von Regelungsaufgaben

Für die Entwicklung einer Regelung, das heißt die Ermittlung und Implementierung eines Reglergesetzes sowie die Bewertung der korrekten Funktionsweise der Regelung existiert ein Vorgehensmodell, das in Abb. 2.3 schematisch dargestellt ist. Der weitere Aufbau dieser Arbeit orientiert sich an diesem Vorgehensmodell.

Im ersten Schritt werden die gewünschten Eigenschaften des geschlossenen Regelkreises, wie beispielsweise sein Führungs- und Störverhalten sowie die Stabilität und Reaktionszeit, festgelegt. Anschließend wird entschieden, welche physikalische Größe des Systems als Regelgröße und welche als Stellgröße verwendet werden soll. Kriterien für die Eignung der Größen sind dabei die einfache Messbarkeit und im Falle der Stellgröße die Empfindlichkeit der Regelgröße auf die jeweilige Stellgröße. Im vierten Schritt wird ein mathematisches Modell der Regelstrecke, also des zu regelnden Prozesses, aufgestellt. Dieses enthält Informationen über das dynamische Verhalten der Regelstrecke und unterstützt somit im weiteren Verlauf den Entwurf des Reglers, also die Bestimmung eines geeigneten Reglergesetzes. Der Reglerentwurf befasst sich darüber hinaus mit der Auswahl einer geeigneten Reglerstruktur, die den in der Regelungsaufgabe festgelegten

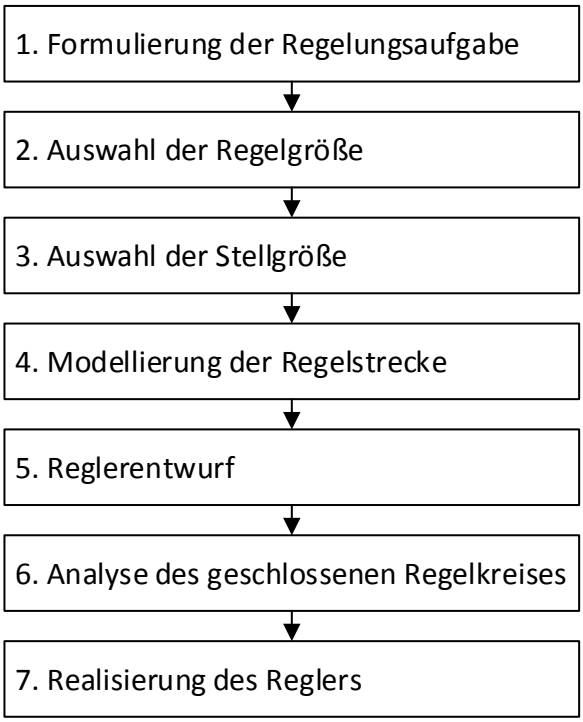


Abbildung 2.3: Vorgehensmodell zur Lösung regelungstechnischer Problemstellungen nach [8].

Güteanforderungen gerecht wird. Nach dem Reglerentwurf folgt die Analyse des geschlossenen Regelkreises. Hier wird dessen Verhalten numerisch simuliert und überprüft, ob das System die Güteanforderungen erfüllt. Im abschließenden Schritt erfolgt die praktische Realisierung des Reglergesetzes, beispielsweise durch Implementierung in Form eines Programms auf einem Mikrocontroller. [8]

2.3 Laplace-Transformation

(a) Dynamisches System im Zeitbereich. **(b)** Dynamisches System im Bildbereich.

Abbildung 2.4: Ein- und Ausgangsgrößen eines dynamischen Systems im Zeit- und Bildbereich. Die Überführung vom Zeit- in den Bildbereich erfolgt durch Laplace-Transformation.

Dynamische Systeme können auf verschiedenen Arten dargestellt werden. Die erste Möglichkeit ist die Darstellung im Zeitbereich in Form einer linearen Differentialgleichung n. Ordnung mit

konstanten Koeffizienten.

$$\begin{aligned} a_n \frac{dy^n(t)}{dt^n} + a_{n-1} \frac{dy^{n-1}(t)}{dt^{n-1}} + \ldots + a_1 \frac{dy(t)}{dt} + a_0\, y(t) = \\ b_m \frac{du^m(t)}{dt^m} + b_{m-1} \frac{du^{m-1}(t)}{dt^{m-1}} + \ldots + b_1 \frac{du(t)}{dt} + b_0\, u(t). \end{aligned} \tag{2.3}$$

Die Ausgangsgröße $y(t)$ des Systems sowie die zeitlichen Ableitungen dieser Größe hängen hierbei von der Eingangsgröße $u(t)$ und deren zeitlichen Ableitungen ab (Abb. 2.4a). In physikalischen Systemen gilt dabei immer $n > m$, was als Kausalität bezeichnet wird und bedeutet, dass sämtliche Signale nur für $t > 0$ definiert sind. Problematisch ist jedoch, dass die analytische Lösung derartiger Differentialgleichungen im Zeitbereich nicht immer trivial ist. Aus diesem Grund erfolgt oftmals eine Transformation in den Bildbereich (Abb. 2.4b) mithilfe der *Laplace-Transformation*. Dies ist eine Integraltransformation, die einer großen Menge von Originalfunktionen $y(t)$ umkehrbar eindeutig eine Bildfunktion $Y(s)$ zuordnet. Diese Zuordnung erfolgt durch Berechnung des Laplace-Integrals von $y(t)$

$$Y(s) = \int_0^\infty y(t)\, \mathrm{e}^{-st}\, \mathrm{d}t \tag{2.4}$$

mit der komplexen Variable $s = \sigma + i\omega$ im Argument der Laplace-Transformierten $Y(s)$. Anwendbar ist diese Transformation nur dann, wenn das Integral konvergiert und $y(t) = 0$ für $t < 0$ gilt, was jedoch in vielen Fällen gegeben ist. In der Praxis ist eine Berechnung des Laplace-Integrals meist nicht erforderlich. Stattdessen kann auf Korrespondenztafeln zurückgegriffen werden, in denen für diverse Funktionen $y(t)$ korrespondierende Laplace-Transformierte $Y(s)$ tabelliert sind. Zu finden sind solche Tafeln beispielsweise in [39] oder [41].

Die Laplace-Transformation verfügt dabei nach [39] über eine Reihe wichtiger Eigenschaften, die im Folgenden kurz dargestellt werden sollen.

Linearität: Die Laplace-Transformation erfüllt für beliebige Konstanten a_1 und a_2 das Superpositionsprinzip

$$\mathcal{L}\left\{a_1 y_1(t) + a_2 y_2(t)\right\} = a_1 Y_1(s) + a_2 Y_2(s) \tag{2.5}$$

und ist damit linear.

Differentiation: Eine n-fache Differentiation im Zeitbereich entspricht einer Multiplikation mit s^n im Bildbereich. Zudem müssen die Anfangswerte, also die Funktionswerte zum Zeitpunkt $t = 0$, entsprechend des Zusammenhangs

$$\mathcal{L}\left\{\frac{d^n y(t)}{dt^n}\right\} = s^n Y(s) - \sum_{i=1}^{n} s^{n-i} \left.\frac{d^{i-1} y(t)}{dt^{i-1}}\right|_{t=0} \tag{2.6}$$

berücksichtigt werden.

Integration: Eine Integration im Zeitbereich entspricht einer Multiplikation mit $\frac{1}{s}$ im Bildbereich

$$\mathcal{L}\left\{\int_0^t y(\tau)\, \mathrm{d}t\right\} = \frac{1}{s}\, Y(s). \tag{2.7}$$

Verschiebungssatz: Für eine beliebige Konstante $a > 0$ gilt

$$\mathcal{L}\{y(t-a)\} = \mathrm{e}^{-as}\, Y(s). \tag{2.8}$$

Ähnlichkeitssatz: Für eine beliebige Konstante $a > 0$ gilt

$$\mathcal{L}\{y(at)\} = \frac{1}{a}\, Y\left(\frac{s}{a}\right). \tag{2.9}$$

Faltungssatz: Die Faltung zweier Funktionen im Zeitbereich entspricht der Multiplikation der beiden zugehörigen Bildfunktionen im Bildbereich

$$\mathcal{L}\left\{\int_0^t y(\tau) g(t-\tau)\,\mathrm{d}\tau\right\} = Y(s)G(s). \tag{2.10}$$

Grenzwertsätze: Wenn $y(t)$ und $\dot{y}(t)$ jeweils eine Laplace-Transformierte besitzen, dann gilt der *Satz vom Anfangswert*

$$\lim_{t\to 0} y(t) = \lim_{s\to\infty} sY(s). \tag{2.11}$$

Der *Satz vom Endwert* setzt ebenfalls die Existenz der Laplace-Transformierten von $y(t)$ und $\dot{y}(t)$ sowie zusätzlich die Existenz des Grenzwertes $\lim\limits_{t\to\infty} y(t)$ voraus und besagt

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s). \tag{2.12}$$

Auf Basis der Regel für die Differentiation kann nun Gleichung (2.3) in den Bildbereich transformiert werden. Dazu werden alle Anfangswerte zu null gesetzt und die Laplace-Transformierten der linken und rechten Seite der Gleichung gebildet. Es ergibt sich der rein algebraische Ausdruck

$$\begin{aligned} a_n s^n\, Y(s) + a_{n-1} s^{n-1}\, Y(s) + \ldots + a_1 s\, Y(s) + a_0\, Y(s) = \\ b_m s^m\, U(s) + b_{m-1} s^{m-1}\, U(s) + \ldots + b_1 s\, U(s) + b_0\, U(s) \end{aligned} \tag{2.13}$$

und nach Umformung

$$Y(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0}\, U(s). \tag{2.14}$$

Dieser Ausdruck kann in Partialbrüche zerlegt und durch Bestimmung der inversen Laplace-Transformierten der jeweiligen Partialbrüche in den Zeitbereich zurücktransformiert werden. Auf diese Weise lässt sich eine Lösung $y(t)$ für die Differentialgleichung angeben. Dieses Vorgehen wird in [39] ausführlich beschrieben und soll hier nicht weiter thematisiert werden. [8]

2.4 Übertragungsfunktion

Aus Gleichung (2.14) lässt sich durch Division mit $U(s)$ der Ausdruck

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0} \tag{2.15}$$

berechnen. Er wird die *Übertragungsfunktion* des dynamischen Systems genannt und beschreibt den Einfluss der Eingangsgröße $U(s)$ auf die Ausgangsgröße $Y(s)$, also das dynamische Systemverhalten. Die Übertragungsfunktion enthält dieselben Informationen wie die Differentialgleichung im Zeitbereich, ist jedoch ein rein algebraischer Ausdruck und daher wesentlich einfacher zu handhaben als eine Differentialgleichung.

Überführt man sämtliche Größen des allgemeinen Regelkreises aus Abb. 2.1 in den Bildbereich, so ergibt sich das in Abb. 2.5 dargestellte System. Sämtliche Eingangs- und Ausgangsgrößen hängen nun von der komplexen Bildvariablen s ab und Strecke sowie Regler können durch Übertragungsfunktionen $G_S(s)$ und $G_R(s)$ beschrieben werden. Das Übertragungsverhalten des Messgliedes wird durch die Übertragungsfunktion $G_M(s)$ in der Rückführung berücksichtigt.

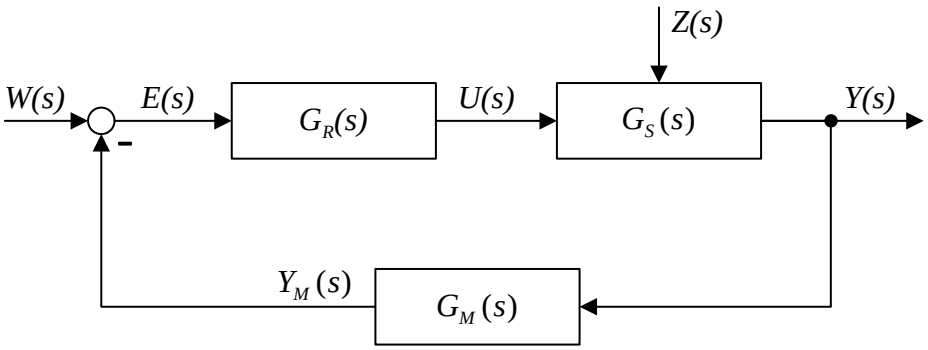


Abbildung 2.5: Blockschaltbild und Größen eines allgemeinen Regelkreises im Bildbereich.

Die Ausgangsgröße $Y(s)$ des dargestellten Regelkreises berechnet sich aus der Führungsgröße $W(s)$ und der Störgröße $Z(s)$ zu

$$Y(s) = \frac{G_R(s)G_S(s)W(s) + G_S(s)Z(s)}{1 + G_R(s)G_S(s)G_M(s)}. \tag{2.16}$$

Wie bereits in Abschnitt 2.1 erläutert, gibt es Folgeregelungen und Festwertregelungen, die wahlweise auf ein günstiges Führungs- oder Störverhalten hin ausgelegt werden.

Bei Folgeregelungen ist der Einfluss der Führungsgröße $W(s)$ auf die Ausgangsgröße $Y(s)$ von Interesse. Hierzu lässt sich unter Vernachlässigung von Störungen ($Z(s) = 0$) die *Führungsübertragungsfunktion*

$$G_W(s) = \frac{Y(s)}{W(s)} = \frac{G_R(s)G_S(s)}{1 + G_R(s)G_S(s)G_M(s)} \tag{2.17}$$

des geschlossenen Regelkreises angeben.

Im Gegensatz dazu ist bei Festwertregelungen die Kenntnis des Einflusses der Störgröße $Z(s)$ auf die Ausgangsgröße $Y(s)$ essentiell. Dieser Zusammenhang wird bei ausgeschalteter Führung ($W(s) = 0$) durch die *Störübertragungsfunktion*

$$G_Z(s) = \frac{Y(s)}{Z(s)} = \frac{G_S(s)}{1 + G_R(s)G_S(s)G_M(s)} \tag{2.18}$$

beschrieben. Dabei sei angemerkt, dass hier die Störung vor dem Eingang der Strecke angreift. Genausogut ist es möglich, dass die Störung sich erst auf den Ausgang der Strecke auswirkt. In diesem Fall lautet die Störungsübertragungsfunktion

$$G_Z(s) = \frac{Y(s)}{Z(s)} = \frac{1}{1 + G_R(s)G_S(s)G_M(s)}. \tag{2.19}$$

Wird die Rückführschleife geöffnet, wie es in Abb. 2.6 durch den geöffneten Schalter angedeutet ist, liegt also eine offene Wirkkette wie bei einer Steuerung vor, so lässt sich die Übertragungsfunktion des offenen Regelkreises zu $G_0(s) = G_R(s)G_S(s)G_M(s)$ bestimmen. [39]

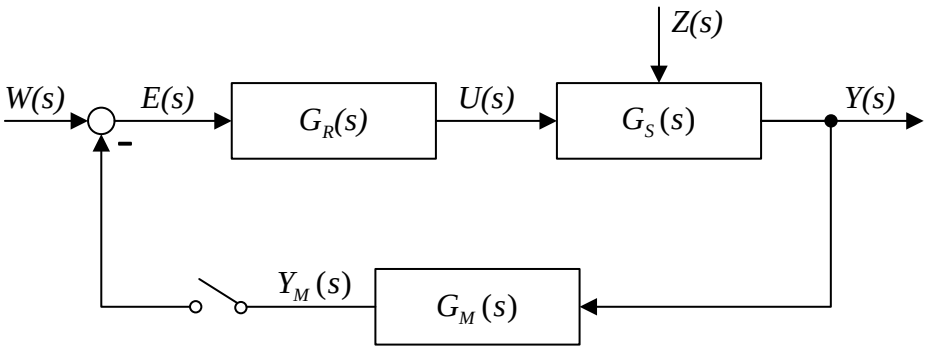


Abbildung 2.6: Blockschaltbild des offenen Regelkreises.

2.5 Darstellungsformen von dynamischen Systemen im Zeitbereich

Wie bereits in Abschnitt 2.3 beschrieben, können dynamische Systeme im Zeitbereich durch Differentialgleichungen beschrieben werden. Eine einfache Darstellung des Systemverhaltens, das heißt die Reaktion des Systems auf ein bestimmtes Eingangssignal, kann durch zwei Kennfunktionen, die *Übergangsfunktion* und die *Gewichtsfunktion*, erfolgen. Diese sollen im Folgenden kurz vorgestellt werden.

2.5.1 Übergangsfunktion

Die Übergangsfunktion beschreibt die Reaktion des Systems auf eine sprungförmige Anregung am Systemeingang. Die Anregung ergibt sich folglich durch die Sprungfunktion

$$\sigma(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases} \tag{2.20}$$

und die Sprunghöhe u_0 zu

$$u(t) = u_0\,\sigma(t). \tag{2.21}$$

Die *Übergangsfunktion* $h(t)$ wird durch den Quotienten der Systemantwort $y(t)$ auf die sprungförmige Anregung, der *Sprungantwort*, und der Sprunghöhe u_0 als

$$h(t) = \frac{y(t)}{u_0} \tag{2.22}$$

beschrieben. In Abb. 2.7a ist schematisch eine Übergangsfunktion für ein Verzögerungsglied zweiter Ordnung, zum Beispiel einen gedämpften Feder-Masse-Schwinger, dargestellt. [8]

2.5.2 Gewichtsfunktion

Die Gewichtsfunktion beschreibt die Reaktion des Systems auf eine impulsförmige Anregung am Systemeingang. Diese wird durch den Dirac-Impuls beschrieben, der sich aus dem Rechteckimpuls

$$r_\varepsilon(t) = \begin{cases} \frac{1}{\varepsilon}, & 0 \le t \le \varepsilon \\ 0, & \text{sonst} \end{cases} \tag{2.23}$$

durch den Grenzübergang $\varepsilon \to 0$ zu

$$\delta(t) = \lim_{\varepsilon \to 0} r_\varepsilon(t) \tag{2.24}$$

ergibt. Der Impuls, der keine Funktion im klassischen Sinne, sondern eine Distribution dargestellt, schließt eine Fläche der Größe 1 ein und ist unendlich kurz und hoch.

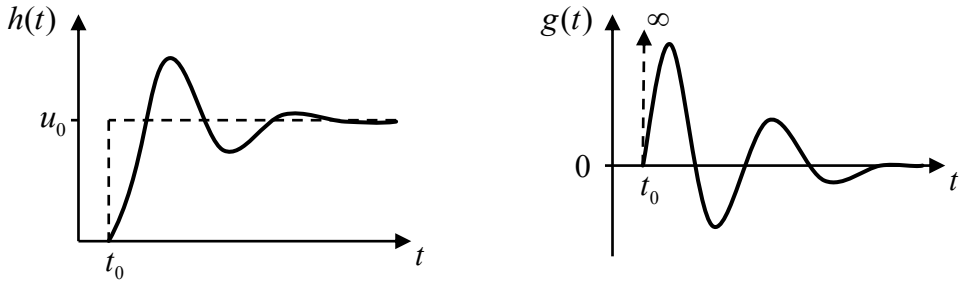
Aus der Systemantwort $y(t)$ auf die impulsförmige Anregung und der Fläche des Impulses kann nun die *Gewichtsfunktion*

$$g(t) = \frac{y(t)}{\int\limits_{-\infty}^{\infty} u(t)\mathrm{d}t} \tag{2.25}$$

berechnet werden. Bei Anregung mit einem Einheitsimpuls der Fläche 1 entspricht die Systemantwort $y(t)$ direkt der Gewichtsfunktion. Abb. 2.7b zeigt beispielhaft die Gewichtsfunktion eines Verzögerungsgliedes zweiter Ordnung. [8]

2.6 Darstellungsformen von dynamischen Systemen im Bildbereich

Eine weitere Möglichkeit der Darstellung dynamischer Systeme im Bildbereich bietet die in Abschnitt 2.4 vorgestellte Übertragungsfunktion $G(s)$. Die Tatsache, dass die Variable s komplex ist, macht jedoch eine direkte anschauliche Darstellung der Übertragungsfunktion unmöglich. Aus diesem Grund gibt es verschiedene Ansätze zur indirekten Darstellung der Übertragungsfunktion, von denen nun die Pol-Nullstellen-Verteilung, die Ortskurve und das Bode-Diagramm erläutert werden sollen.



(a) Übergangsfunktion. **(b)** Gewichtsfunktion.

Abbildung 2.7: Kennfunktionen eines dynamischen Systems im Zeitbereich.

2.6.1 Pol-Nullstellen-Verteilung

Die Übertragungsfunktion aus Gleichung (2.15) lässt sich durch Berechnung der Nullstellen und Faktorisierung der Zähler- und Nennerpolynome umschreiben in

$$G(s) = K \frac{(s - s_{Z,m})(s - s_{Z,m-1}) \dots (s - s_{Z,1})}{(s - s_{N,m})(s - s_{N,m-1}) \dots (s - s_{N,1})} \tag{2.26}$$

mit $K = \frac{b_m}{a_m}$. Die Nullstellen $s_{Z,m}$, $s_{Z,m-1} \dots s_{Z,1}$ und Pole $s_{N,m}$, $s_{N,m-1} \dots s_{N,1}$ können dabei reell oder komplex sein und lassen sich somit sehr einfach, wie in Abb. 2.8 dargestellt, in der komplexen Zahlenebene darstellen. Zusammen mit dem Vorfaktor K enthalten sie die gleichen Informationen wie die Übertragungsfunktion, beschreiben also eindeutig das dynamische Verhalten des Systems. Ist $s = \sigma + i\omega$ eine komplexe Pol- oder Nullstelle des Systems, so ist auch ihr komplex Konjugiertes $\bar{s} = \sigma - i\omega$ Pol- bzw. Nullstelle. [41]

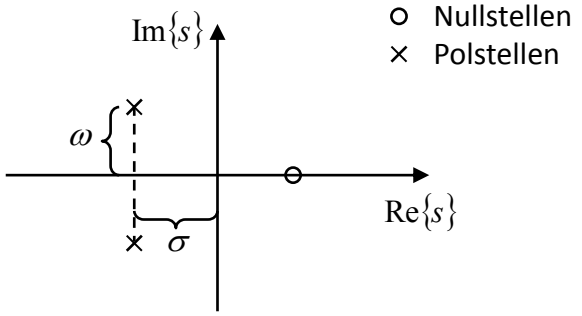


Abbildung 2.8: Pol-Nullstellen-Verteilung einer Übertragungsfunktion.

2.6.2 Frequenzgang

Wird innerhalb der Übertragungsfunktion $G(s)$ die Bildvariable $s = \sigma + i\omega$ mit verschwindendem Realteil, also $\sigma = 0$ gewählt, hängt die Übertragungsfunktion nur noch vom rein imaginären Argument $s = i\omega$ ab. Die daraus erhaltene Darstellungsform des Systemverhaltens wird *Frequenzgang*

$G(i\omega)$ genannt und kann entweder in der Form

$$G(i\omega) = \mathrm{Re}\left\{G(i\omega)\right\} + i\,\mathrm{Im}\left\{G(i\omega)\right\} \tag{2.27}$$

mit Realteil $\mathrm{Re}\left\{G(i\omega)\right\}$ und Imaginärteil $\mathrm{Im}\left\{G(i\omega)\right\}$ oder nach Betrag und Phase in der Form

$$G(i\omega) = |G(i\omega)|\,\mathrm{e}^{\varphi(\omega)} \tag{2.28}$$

dargestellt werden. Dabei wird $|G(i\omega)|$ der *Amplitudengang* und $\varphi(\omega)$ der *Phasengang* des Systems genannt.

Dieses Vorgehen hat insofern praktische Bedeutung, als dass der Frequenzgang eines Systems direkt gemessen werden kann. Hierzu wird der Eingang mit einer harmonischen Schwingung

$$u(t) = u_0 \sin(\omega t) \tag{2.29}$$

mit Frequenz ω und Amplitude u_0 beaufschlagt und das Ausgangssignal gemessen. Dieses ist im Allgemeinen ebenfalls eine harmonische Schwingung, weist jedoch wie in Abb. 2.9 gezeigt, frequenzabhängig eine veränderte Amplitude $y_0(\omega)$ und eine Phasenverschiebung $\varphi(\omega)$ relativ zum Eingangssignal auf, sodass für das Signal am Ausgang

$$y(t) = y_0(\omega) \sin(\omega t - \varphi(\omega)) \tag{2.30}$$

gilt. Wiederholt man die Messung für verschiedene Frequenzen ω, so kann der Amplitudengang des Systems zu

$$|G(i\omega)| = \frac{y_0(\omega)}{u_0} = \sqrt{\mathrm{Re}\left\{G(i\omega)\right\}^2 + \mathrm{Im}\left\{G(i\omega)\right\}^2} \tag{2.31}$$

bestimmt werden. Wird der Phasengang

$$\varphi(\omega) = \arg G(i\omega) = \arctan\left(\frac{\mathrm{Im}\left\{G(i\omega)\right\}}{\mathrm{Re}\left\{G(i\omega)\right\}}\right) \tag{2.32}$$

berechnet, liegt abschließend der experimentell ermittelte Frequenzgang $G(i\omega)$ des Systems vor. Wie die Frequenzen ω und die Amplitude u_0 des Eingangssignals zu wählen sind, um das Systemverhalten eindeutig zu erfassen, ist im Einzelnen abhängig vom zu untersuchenden System zu entscheiden. [39]

2.6.2.1 Ortskurve

Eine Möglichkeit der grafischen Darstellung des Frequenzgangs $G(i\omega)$ ist die *Ortskurve*. Wie in Abb. 2.10a gezeigt, werden zur Erstellung der Ortskurve die zu jeder Frequenz ω_i gehörige Phasenverschiebung $\varphi(\omega_i)$ sowie das Amplitudenverhältnis $|G(i\omega_i)|$ zwischen Eingangssignal $u(t)$ und Ausgangssignal $y(t)$ in der komplexen Ebene aufgetragen. Wiederholt man diese Auftragung für alle Frequenzen ω von 0 bis ∞, erhält man die Ortskurve.

In Abb. 2.10b ist beispielhaft die Ortskurve für ein PT_1-Glied mit der Übertragungsfunktion

$$G(s) = \frac{K_P}{sT_1 + 1} \tag{2.33}$$

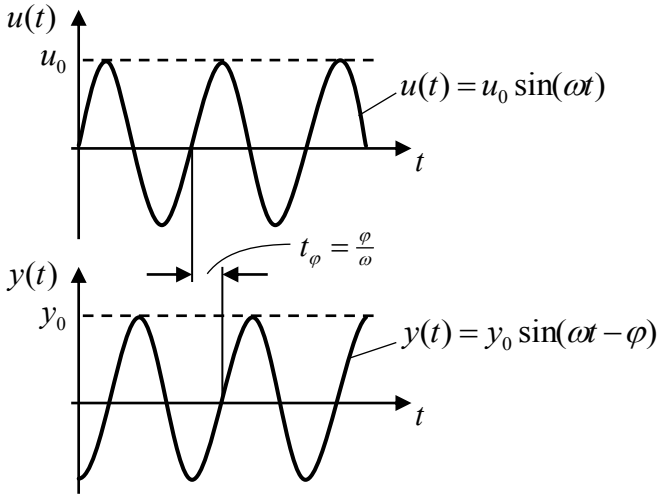
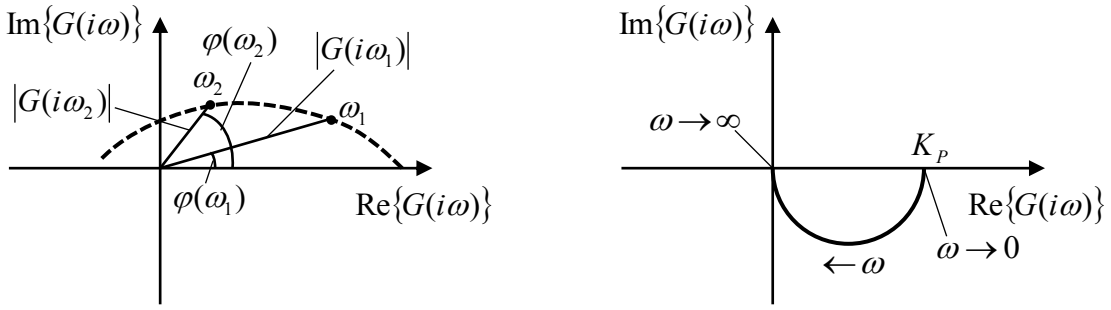


Abbildung 2.9: Ein- und Ausgangssignal bei sinusförmiger Anregung eines Systems.



(a) Erstellung der Ortskurve. **(b)** Ortskurve eines PT_1-Glieds.

Abbildung 2.10: Darstellung dynamischer Systeme mittels Ortskurve des Frequenzgangs.

dargestellt. Durch die Substitution $s = i\omega$ und Erweitern des Ausdrucks mit dem komplex Konjugierten des Nenners ergibt sich der Frequenzgang

$$G(i\omega) = K_P \left(\frac{1}{1 + \omega^2 T_1^2} - i \frac{\omega T_1}{1 + \omega^2 T_1^2} \right) \tag{2.34}$$

und daraus, entsprechend Gleichung (2.31), der Amplitudengang

$$|G(i\omega)| = \frac{K_P}{\sqrt{1 + \omega^2 T_1^2}} \tag{2.35}$$

sowie, entsprechend Gleichung (2.32), der Phasengang

$$G(i\omega) = -\arctan(\omega T_1). \tag{2.36}$$

Zum Zeichnen der Ortskurve setzt man einige ω in den Frequenzgang $G(i\omega)$ ein, berechnet den entsprechenden Real- und Imaginärteil und trägt diese anschließend in die komplexe Ebene ein. [39]

2.6.2.2 Bode-Diagramm

Eine andere Möglichkeit der grafischen Darstellung des Frequenzgangs $G(i\omega)$ ist das *Bode-Diagramm*. Wie in Abb. 2.11 dargestellt, werden hier der Amplitudengang $|G(i\omega)|$ logarithmisch und der Phasengang $\varphi(\omega)$ linear über die logarithmische Frequenz ω aufgetragen.

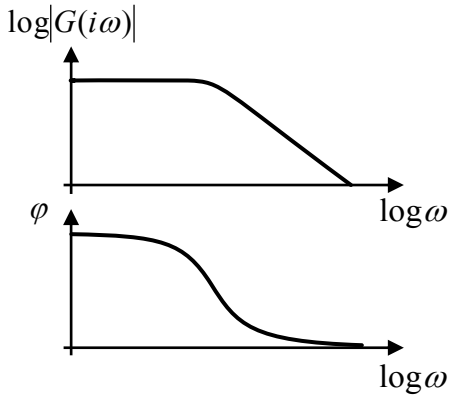


Abbildung 2.11: Bode-Diagramm eines PT_1-Glieds. Im Bode-Diagramm werden Amplituden und Phasengang doppelt- bzw. einfachlogarithmisch über die Frequenz aufgetragen.

Grund für die doppeltlogarithmische Darstellung des Amplitudengangs und die einfachlogarithmische Darstellung des Phasengangs ist die einfache Bestimmung des Bode-Diagramms zweier in Reihe geschalteter Übertragungsglieder. Hierzu werden einfach die beiden Bode-Diagramme der einzelnen Übertragungsglieder addiert. [39]

Neben der logarithmischen Skalierung des Amplitudengangs kann auch eine lineare Skalierung vorgenommen werden, wenn der Amplitudengang zuvor gemäß der Beziehung

$$|G(i\omega)|_{\mathrm{dB}} = 20 \cdot \log_{10} |G(i\omega)| \tag{2.37}$$

in eine Dezibel-Skala umgerechnet wird. Das Dezibel ist dabei eine Hilfsmaßeinheit, die die logarithmische Skala in eine lineare Skala abbildet. Eine Umrechnung der Dezibel-Skala in die logarithmische Skala ist mittels des Zusammenhangs

$$|G(i\omega)| = 10^{\frac{|G(i\omega)|_{\mathrm{dB}}}{20}} \tag{2.38}$$

möglich. [22]

2.7 Zustandsraumdarstellung

Die Zustandsraumdarstellung ist eine weitere in der Regelungstechnik oft anzutreffende Darstellungsform dynamischer Systeme im Zeitbereich. Hierbei wird der zu einem bestimmten Zeitpunkt t vorliegende Zustand sämtlicher Energiespeicher des Systems durch die Zustandsgrößen $\boldsymbol{x}(t) = (x_1, x_2, \ldots, x_{n_x})^\intercal$ beschrieben. Beispiele für Zustandsgrößen können die Temperatur

eines Wärmespeichers oder die kinetische Energie eines Fahrzeugs sein. Diese Zustände werden, wie in Abb. 2.12 dargestellt, durch die Eingangsgrößen $\boldsymbol{u}(t) = (u_1, u_2, \ldots, u_{n_u})^\intercal$ beeinflusst und wirken sich auf die Ausgangsgrößen $\boldsymbol{y}(t) = \left(y_1, y_2, \ldots, y_{n_y}\right)^\intercal$ aus. Ein solches System mit mehreren Ein- und Ausgangsgrößen wird als Mehrgrößensystem oder MIMO-System (Multiple Input Multiple Output) bezeichnet. Im Gegensatz dazu wurde in den voherigen Abschnitten mit Eingrößensystemen oder SISO-Systemen (Single Input Single Output) gearbeitet, bei denen Ein- und Ausgangsgröße skalar waren.

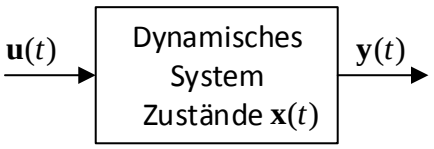


Abbildung 2.12: Dynamisches System mit Zustandsgrößen $\boldsymbol{x}(t)$.

Mathematisch erfolgt die Aufstellung des Zustandsraummodells durch Reduktion der linearen System-Differentialgleichung n. Ordnung in ein System von n Differentialgleichungen 1. Ordnung. Weiterhin kann das Zustandsraummodell direkt aus den Koeffizienten der Differentialgleichung oder der Übertragungsfunktion abgeleitet werden.

2.7.1 Nichtlineare zeitinvariante Zustandsraumdarstellung

Zunächst soll in diesem Abschnitt die allgemeinere Form eines nichtlinearen zeitinvarianten Zustandsraummodells angegeben werden. Anschließend wird in Abschnitt 2.7.2 auf die Linearisierung dieses Systems eingegangen. Zeitinvariant heißt in diesem Zusammenhang, dass die Vektorfunktionen $\boldsymbol{f}$ und $\boldsymbol{g}$ nicht explizit von der Zeit abhängen dürfen. Die Fälle $\boldsymbol{f} = \boldsymbol{f}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right)$ und $\boldsymbol{g} = \boldsymbol{g}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),t\right)$ sind also in der folgenden Darstellung des Zustandsraums

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t),\boldsymbol{u}(t)\right), \quad \boldsymbol{x}(t_0) = \boldsymbol{x_0} \tag{2.39}$$

$$\boldsymbol{y}(t) = \boldsymbol{g}\left(\boldsymbol{x}(t),\boldsymbol{u}(t)\right) \tag{2.40}$$

nicht inbegriffen. Dabei bezeichnet Gleichung (2.39) die Zustandsdifferentialgleichung 1. Ordnung für die Zustände $\boldsymbol{x}(t)$ und Gleichung (2.40) die Ausgangsgleichung des dynamischen Systems. Der Anfangszustand des Systems zum Zeitpunkt t_0 wird durch die Anfangswerte $\boldsymbol{x_0}$ der Zustandsgrößen berücksichtigt. Die Vektorfunktionen $\boldsymbol{f}$ und $\boldsymbol{g}$ können hier beliebige nichtlineare Zusammenhänge in $\boldsymbol{x}(t)$ und $\boldsymbol{u}(t)$ darstellen, müssen aber genügend oft stetig differenzierbar in $\boldsymbol{x}(t)$ und $\boldsymbol{u}(t)$ sein. [39]

2.7.2 Lineare zeitinvariante Zustandsraumdarstellung

In vielen Fällen ist eine lineare Näherung der nichtlinearen Vektorfunktionen $\boldsymbol{f}$ und $\boldsymbol{g}$ möglich. Eine Linearisierung kann durch Abbruch der zugehörigen Taylorreihe nach dem linearen Term erfolgen. Dies ist in Abb. 2.13 beispielhaft anhand einer skalaren Funktion $y = f(u)$, die durch

eine Tangente an die Funktion im Arbeitspunkt u_0 angenähert werden kann, gezeigt. Für die Gleichung der Tangente gilt

$$y = f(u_0) + \frac{\partial f(u_0)}{\partial u}(u - u_0). \tag{2.41}$$

Bei der Linearisierung des Systems gilt entsprechend für die i. Komponente der nichtlinearen

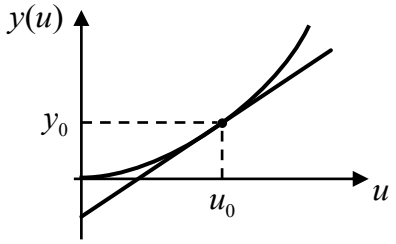


Abbildung 2.13: Annäherung einer nichtlinearen Funktion durch eine Tangente im Arbeitspunkt.

Vektorfunktion $\boldsymbol{f}\,(\boldsymbol{x}(t),\boldsymbol{u}(t))$ die lineare Näherung

$$f_i\,(\boldsymbol{x},\boldsymbol{u}) = f_i(\boldsymbol{x_0},\boldsymbol{u_0}) + \sum_{j=1}^{n_x} \frac{\partial f_i(\boldsymbol{x_0},\boldsymbol{u_0})}{\partial x_j}(x_j - x_{0,j}) + \sum_{k=1}^{n_u} \frac{\partial f_i(\boldsymbol{x_0},\boldsymbol{u_0})}{\partial u_k}(u_k - u_{0,k}) \tag{2.42}$$

um den Arbeitspunkt $\boldsymbol{x_0}, \boldsymbol{u_0}$ des Systems. Führt man diesen Vorgang für alle Komponenten von $\boldsymbol{f}$ und $\boldsymbol{g}$ unter Beachtung von $\boldsymbol{f}\,(\boldsymbol{x_0},\boldsymbol{u_0}) = \boldsymbol{0}$ und $\boldsymbol{g}\,(\boldsymbol{x_0},\boldsymbol{u_0}) = \boldsymbol{y_0}$ mit der stationären Ausgangsgröße $\boldsymbol{y_0}$ im Arbeitspunkt durch, erhält man die lineare zeitinvariante Zustandsraumdarstellung des dynamischen Systems

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \tag{2.43}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t) \tag{2.44}$$

mit der Systemmatrix $\boldsymbol{A} \in \mathcal{R}^{n_x \times n_x}$, der Eingangsmatrix $\boldsymbol{B} \in \mathcal{R}^{n_x \times n_u}$, der Ausgangsmatrix $\boldsymbol{C} \in \mathcal{R}^{n_y \times n_x}$ und der Durchgangsmatrix $\boldsymbol{D} \in \mathcal{R}^{n_y \times n_u}$. Zu beachten ist, dass die Vektoren nun anstelle absoluter Größen nur noch die Abweichungen vom Arbeitspunkt beschreiben. So gilt für den Zustandsvektor $\boldsymbol{x}(t) = (x_1 - x_{0,1}, x_2 - x_{0,2}, \ldots, x_{n_x} - x_{0,n_x})^\intercal$, für den Eingangsvektor $\boldsymbol{u}(t) = (u_1 - u_{0,1}, u_2 - u_{0,2}, \ldots, u_{n_u} - u_{0,n_u})^\intercal$ und für den Ausgangsvektor $\boldsymbol{y}(t) = \left(y_1 - y_{0,1}, y_2 - y_{0,2}, \ldots, y_{n_y} - y_{0,n_y}\right)^\intercal$. Details zur Linearisierung des Systems können in [6] nachgelesen werden.

Das linearisierte System kann, wie in Abb. 2.14 zu sehen, in Form eines Blockschaltbildes dargestellt werden. Doppelpfeile kennzeichnen hierbei vektorielle Signale. Der Block $\frac{1}{s}$ stellt einen Integrator dar, an dessen Ausgang der Zustandsvektor $\boldsymbol{x}(t)$ ausgegeben wird. Für physikalisch kausale Systeme, in deren Differentialgleichung immer $n > m$ gilt, ist die Durchgangsmatrix stets eine Nullmatrix, die Eingangsgrößen $\boldsymbol{u}(t)$ haben also keine direkte Wirkung auf die Ausgangsgrößen $\boldsymbol{y}(t)$.

Grund für die Einführung des Zustandsraummodells ist die einfache Handhabung komplexer Mehrgrößensysteme und die Tatsache, dass Ein- und Mehrgrößensysteme formal gleich behandelt werden können. Zudem eignet sich die Matrixschreibweise sehr gut für die computergestützte numerische Berechnung, beispielsweise in MATLAB. Auch viele Entwurfsverfahren für Regler benötigen ein Zustandsraummodell des Systems. [39]

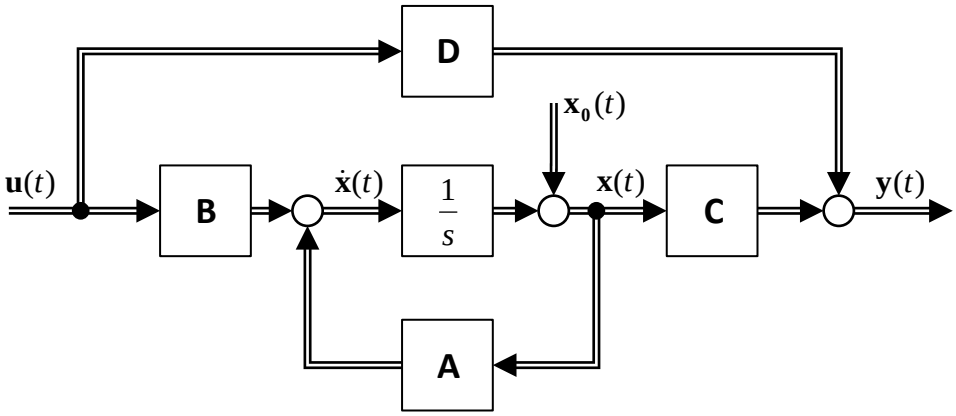


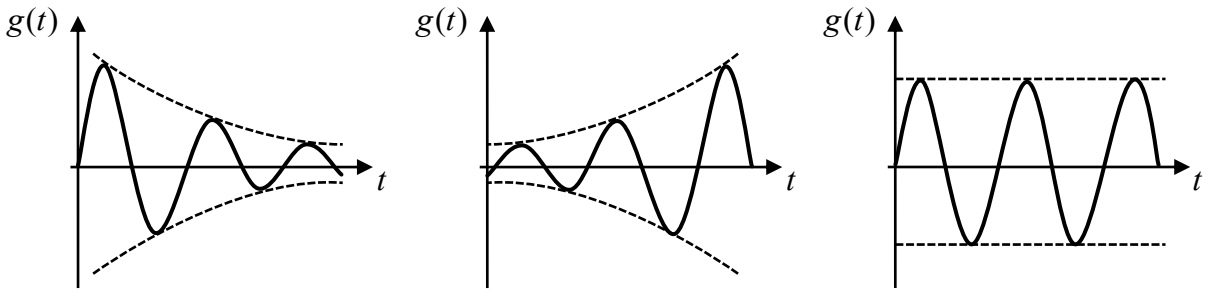
Abbildung 2.14: Blockschaltbild des linearen Zustandsraummodells eines Mehrgrößensystems nach [8].

2.8 Stabilität dynamischer Übertragungssysteme

Die Stabilität eines dynamischen System beschreibt dessen Verhalten bei Auslenkung aus der Ruhelage. Zur Beurteilung der Stabilität eines linearen Übertragungssystems wird daher die in Abschnitt 2.5.2 eingeführte Gewichtsfunktion $g(t)$ untersucht. Gilt

$$\lim_{t\to\infty} g(t) = 0, \tag{2.45}$$

wird das System als *asymptotisch stabil* bezeichnet. Ein solches System, dessen Gewichtsfunktion in Abb. 2.15a dargestellt ist, kehrt nach einer impulsförmigen Anregung wieder in seine ursprüngliche Ruhelage zurück. Im Gegensatz dazu wird ein System als *instabil* bezeichnet, wenn dessen Gewichtsfunktion für wachsendes t betragsmäßig gegen unendlich geht. Dies ist in Abb. 2.15b dargestellt. Ein dritter Fall, die *Grenzstabilität*, tritt auf, wenn die Gewichtsfunktion für wachsendes t einen endlichen Grenzwert nicht überschreitet. Das System führt in einem solchen Fall, wie in Abb. 2.15c gezeigt, Dauerschwingungen aus oder verharrt in einem stationären ausgelenkten Zustand. [39]



(a) Stabiles System. **(b)** Instabiles System. **(c)** Grenzstabiles System.

Abbildung 2.15: Schwingungsverhalten stabiler und instabiler Systeme bei Auslenkung aus der Ruhelage.

2.8.1 Lage der Pole und Nullstellen

Welches Stabilitätsverhalten ein System aufweist, kann anhand der Lage der Pole seiner Übertragungsfunktion $G(s)$ in der komplexen Ebene beurteilt werden. Dazu müssen lediglich die Pole der Übertragungsfunktion, wie in Abschnitt 2.6.1 beschrieben, ermittelt werden. Es ergeben sich folgende, in Abb. 2.16 dargestellte, Fälle.

Asymptotische Stabilität: Liegen alle Pole der Übertragungsfunktion in der linken Hälfte der komplexen Ebene, das heißt gilt

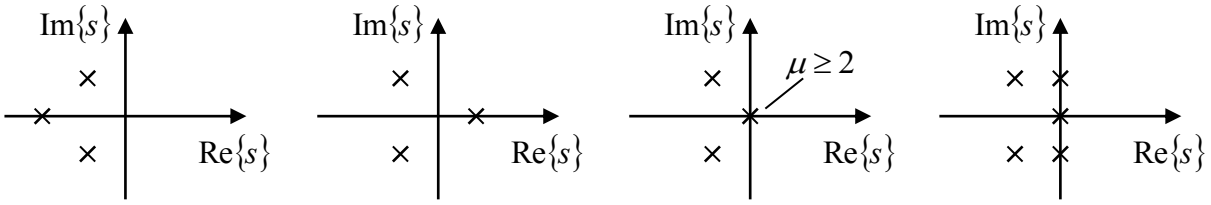
$$\operatorname{Re} s_{N,i} < 0 \quad \text{für alle} \quad i = 1,2,\ldots,m, \tag{2.46}$$

so ist das Übertragungssystem asymptotisch stabil (Abb. 2.16a).

Instabilität: Liegt mindestens ein Pol der Übertragungsfunktion auf der rechten Seite der komplexen Ebene, also existiert mindestens ein Pol, für den $\operatorname{Re} s_{N,i} > 0$ gilt, so ist das Übertragungssystem instabil (Abb. 2.16b). Ebenfalls instabil ist das System, wenn mindestens ein μ-facher Pol ($\mu \geq 2$) mit $\mathrm{Re} = 0$ existiert (Abb. 2.16c).

Grenzstabilität: Liegt kein Pol der Übertragungsfunktion in der rechten Hälfte der komplexen Ebene, kein mehrfacher Pol auf der imaginären Achse und existiert mindestens ein *einfacher* Pol mit $\mathrm{Re} = 0$, ist das Übertragungssystem grenzstabil (Abb. 2.16d).

Generell muss bei den Polen zwischen komplex konjugierten und reellen Polen unterschieden werden. Liegt mindestens ein komplex konjugiertes Polpaar vor, so ist das System schwingungsfähig. Liegen hingegen nur reelle Pole vor, führt das System keine Schwingungen aus, sondern die Signale verlaufen entweder monoton steigend, monoton fallend oder sind zeitlich konstant. [39]



(a) stabil. **(b)** instabil. **(c)** instabil. **(d)** grenzstabil.

Abbildung 2.16: Zusammenhang zwischen Lage der Pole und Stabilität des geschlossenen Regelkreises nach [39].

2.8.2 Hurwitz-Kriterium

Da die Bestimmung der Polstellen unter Umständen mit erheblichem Rechenaufwand verbunden sein kann, gibt es mit dem Hurwitz-Kriterium ein alternatives Stabilitätskriterium, das eine Aussage über die Stabilität des Übertragungssystem direkt anhand der Übertragungsfunktion macht.

Liegt die Übertragungsfunktion eines dynamischen Systems in der Form von Gleichung (2.15) vor, ist das Übertragungssystem genau dann asymptotisch stabil, wenn die Koeffizienten $a_1, a_2, \ldots, a_n$ folgende Bedingungen erfüllen:

- alle Koeffizienten sind von null verschieden, das heißt es gilt $a_i \neq 0$ für $i = 1{,}2, \ldots, n$,
- alle Koeffizienten haben das gleiche Vorzeichen, das heißt es gilt entweder $a_i > 0$ oder $a_i < 0$ für $i = 1{,}2, \ldots, n$,
- alle Hurwitz-Determinanten sind positiv, sprich es gilt $D_i > 0$ für $i = 1{,}2, \ldots, n$.

Die Hurwitz-Determinanten berechnen sich dabei als

$$D_1 = |a_1|, \quad D_2 = \begin{vmatrix} a_1 & a_3 \\ a_0 & a_2 \end{vmatrix}, \quad D_3 = \begin{vmatrix} a_1 & a_3 & 0 \\ a_0 & a_2 & 0 \\ 0 & a_1 & a_3 \end{vmatrix}, \quad D_4 = \ldots \tag{2.47}$$

Es werden also die Koeffizienten $a_1, a_2, \ldots, a_n$ auf die Hauptdiagonale eingetragen und anschießend jede Spalte so ergänzt, dass von unten nach oben Koeffizienten a_i mit aufsteigenden Indices i stehen. Wird der Index $i < 0$ oder $i > n$, so wird anstatt des Koeffizienten eine Null eingetragen. [6, 36]

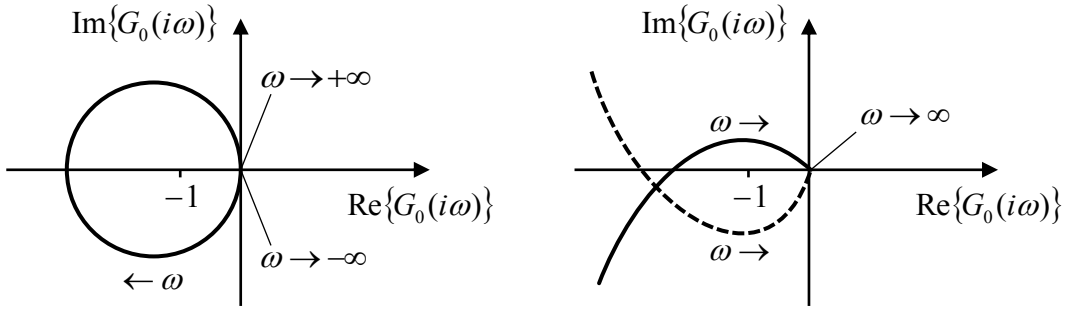
2.8.3 Nyquist-Kriterium

Das Nyquist-Kriterium erlaubt anders als die beiden vorangegangenen Stabilitätskriterien die Beurteilung der Stabilität eines geschlossenen Regelkreises durch Betrachtung des Frequenzgangs des in Abb. 2.6 dargestellten *offenen* Regelkreises $G_0(i\omega)$. Um das Kriterium anzuwenden, muss dementsprechend zunächst die Ortskurve für $0 \leq \omega \leq \infty$ gezeichnet werden. Liegt ein nicht integrierendes System, das heißt ein System ohne Pol bei $s = 0$, vor, wird die so erhaltene Ortskurve anschließend an der reellen Achse gespiegelt. Es ergibt sich der beispielhaft in Abb. 2.17a dargestellte *Nyquist-Plot* für Frequenzen $-\infty \leq \omega \leq \infty$. Anhand dieses Plots wird nun gezählt, wie viele Umläufe m die Ortskurve $G_0(i\omega)$ um den Punkt -1 macht. Dabei wird in Richtung wachsender Frequenzen ω im Uhrzeigersinn positiv und gegen den Uhrzeigersinn negativ gezählt. Für die in Abb. 2.17a dargestellte Ortskurve ergibt sich folglich $m = 1$. Um eine Aussage über die Stabilität des geschlossenen Regelkreises machen zu können, bedarf es weiterhin der Anzahl p der Pole der Übertragungsfunktion des offenen Regelkreises mit positivem Realteil. Anschließend kann über die Beziehung

$$n = m + p \tag{2.48}$$

die Anzahl n von Polen mit positivem Realteil der Übertragungsfunktion des geschlossenen Regelkreises bestimmt werden. Ist diese größer null, das heißt gilt $n > 0$, ist der geschlossene Regelkreis instabil. Andernfalls, also für $n = 0$, ist der geschlossene Regelkreis entweder asymptotisch stabil oder grenzstabil, da keine Pole in der rechten Hälfte der komplexen Ebene liegen.

Eine vereinfachte Variante des Nyquist-Kriteriums kann für integrierende Systeme, die durch einen Pol an der Stelle $s = 0$ zu erkennen sind, angewendet werden. Integrierende Systeme zeichnen sich durch nicht geschlossene Ortskurven, wie sie in Abb. 2.17b gezeigt sind, aus. Folglich ist es nicht möglich, die Anzahl der Umläufe um den Punkt -1 anzugeben. Stattdessen wird lediglich überprüft, ob der Punkt -1 links oder rechts der in Richtung wachsender Frequenzen ω durchlaufenen Ortskurve liegt. Liegt er links der Ortskurve des offenen Regelkreises, ist der geschlossene Regelkreis stabil (gestrichelte Ortskurve in Abb. 2.17b). Liegt der Punkt -1 hingegen rechts der in Richtung wachsender Frequenzen ω durchlaufenen Ortskurve des offenen Regelkreises, ist der geschlossene Regelkreis instabil (durchgezogene Ortskurve in Abb. 2.17b). Alternativ kann das vereinfachte Nyquist-Kriterium auch auf das Bode-Diagramm des offenen Regelkreises angewendet werden. Ist die Phasenverschiebung $\varphi(\omega_d)$ bei der Durchtrittsfrequenz ω_d (siehe Abschnitt 2.8.4) positiv, so ist der geschlossene Regelkreis stabil. Ist sie negativ, so ist der geschlossene Kreis instabil. Gilt $\varphi(\omega_d) = 0$, so liegt der Fall der Grenzstabilität vor. [6, 33, 39]



(a) Nyquist-Plot zur Bestimmung der Anzahl der Umläufe um den Punkt -1.

(b) Ortskurven global integrierender Übertragungssysteme.

Abbildung 2.17: Ortskurven des offenen Regelkreises $G_0(i\omega)$ zur Stabilitätsanalyse mittels Nyquist-Kriterium.

2.8.4 Amplituden- und Phasenreserve

Um eine quantitative Aussage über die Stabilität des geschlossenen Regelkreises anhand der Ortskurve des offenen Regelkreises mit dem Frequenzgang

$$G_0(i\omega) = |G_0(i\omega)|\, \mathrm{e}^{i\varphi_0(\omega)} \tag{2.49}$$

machen zu können, werden die *Amplitudenreserve* A_R und die *Phasenreserve* α_R eingeführt. Mithilfe dieser Größen kann ausgesagt werden, an welcher Stelle die Ortskurve die reelle Achse schneidet, das heißt wie groß der Abstand der Ortskurve zum kritischen Punkt -1 ist. Verläuft die Ortskurve nahe am Punkt -1, so ist zwar die Dynamik des geschlossenen Regelkreises hoch, die Robustheit jedoch gering. Schon kleine Störungen können dazu führen, dass der Regelkreis instabil wird. Ist der Abstand der Ortskurve vom Punkt -1 groß, ist die Robustheit des geschlossenen Regelkreises entsprechend hoch, die Dynamik jedoch gering.

Die Amplitudenreserve ist, wie in Abb. 2.18 gezeigt, als derjenige reelle Verstärkungsfaktor A_R definiert, mit dem der Betrag von $G_0(i\omega_\pi)$ multipliziert werden muss, damit das System am Stabilitätsrand -1 betrieben wird. Folglich gilt für die Amplitudenreserve

$$A_R = \frac{1}{|G_0(i\omega_\pi)|} \quad \text{mit} \quad \varphi_0(\omega_\pi) = -\pi. \tag{2.50}$$

Die Phasenreserve wiederum definiert sich, wie ebenfalls in Abb. 2.18 zu sehen, als Winkel zwischen der reellen Achse und der Verbindungslinie zwischen Ursprung und Schnittpunkt der Ortskurve $G_0(i\omega_d)$ mit dem Einheitskreis. Es gilt also für die Phasenreserve der Zusammenhang

$$\alpha_R = \varphi_0(\omega_d) + \pi \tag{2.51}$$

mit der Durchtrittsfrequenz ω_d.

Aufgrund der Möglichkeit, mit ihnen die Dynamik und Robustheit des geschlossenen Regelkreises beschreiben zu können, helfen Amplituden- und Phasenreserve beim Reglerentwurf. Zu diesem Zweck gibt es in der Literatur für Folge- und Festwertregelungen jeweils vorgegebene Intervalle, in denen Amplituden und Phasenreserve für ein optimales Verhalten des geschlossenen Regelkreises liegen sollten. [6]

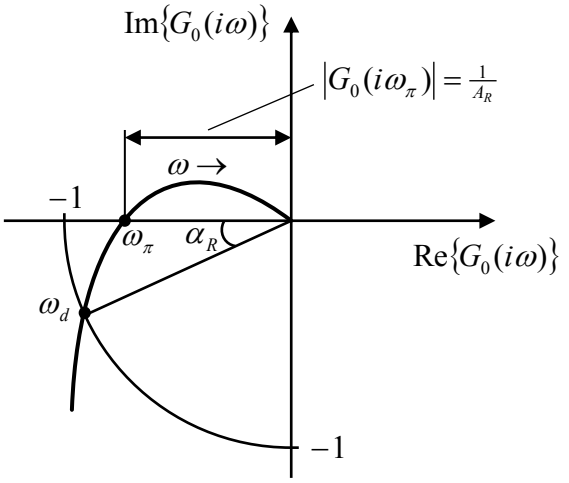


Abbildung 2.18: Definition von Amplituden- und Phasenreserve nach [6].

Weiterhin können Amplituden- und Phasenreserve auch aus dem Bode-Diagramm des dynamischen Systems abgelesen werden. Wie in Abb. 2.19 gezeigt, ist dabei die Phasenreserve α_R diejenige Phasenverschiebung bei der Durchtrittsfrequenz ω_d, bei der der Amplitudengang $|G_0(i\omega)|$ die 0 dB-Linie schneidet. Die Amplitudenreserve ist demgegenüber der Betrag des Amplitudengangs bei der Frequenz ω_π, bei der die Phasenverschiebung genau $-180°$ beträgt. [33]

2.9 Zeitdiskrete Systeme

Bei Systemen wird zwischen zeitkontinuierlichen und zeitdiskreten Systemen unterschieden. Zeitkontinuierliche Systeme können zu jedem beliebigen Zeitpunkt t ein Eingangssignal beliebiger

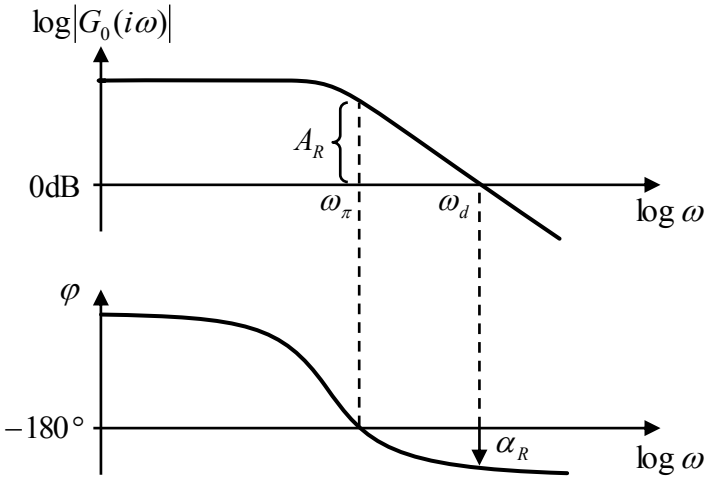


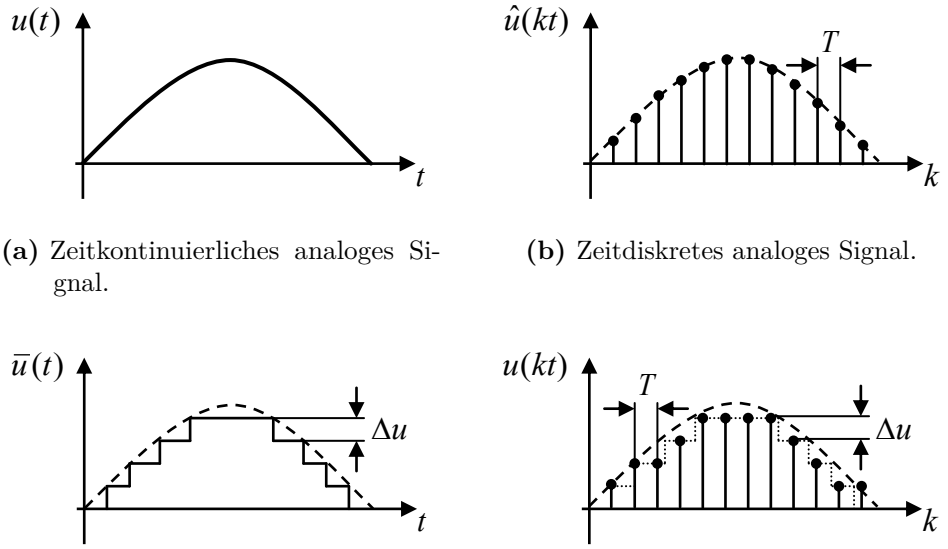
Abbildung 2.19: Definition von Amplituden- und Phasenreserve im Bode-Diagramm nach [33].

Amplitude $u(t)$ aufnehmen und in ein Ausgangssignal mit beliebiger Amplitude $y(t)$ umwandeln. Der Systemzustand $x(t)$ ist hier ebenfalls für jeden Zeitpunkt t definiert. Anders sieht es bei zeitdiskreten Systemen aus. Diese können nur zu bestimmten Zeitpunkten, die in Vielfachen k einer *Abtastzeit* T angegeben sind, ein Eingangssignal $u(kt)$ aufnehmen und ein entsprechendes Ausgangssignal $y(kT)$ ausgeben. Auch der Systemzustand $x(kT)$ ist nur zu diesen diskreten Zeitpunkten $t = kT$ definiert. Solche Signale werden zeitdiskrete Signale genannt. Abb. 2.20 macht diesen Unterschied deutlich.

2.9.1 Quantisierung und Abtastung

Die meisten physikalischen Systeme weisen kontinuierliches Zeitverhalten auf. Im Gegensatz dazu sind digitale Rechner, die oft zur Realisierung eines Regelalgorithmus eingesetzt werden, zeitdiskrete Systeme. Sie können aufgrund einer endlichen Rechengeschwindigkeit nur zu bestimmten diskreten Zeitpunkten ein Eingangssignal aufnehmen und das zugehörige Ausgangssignal berechnen. Weiterhin muss das wertkontinuierliche (analoge) Signal auf ein wertdiskretes (quantisiertes) Signal abgebildet werden, was mit einem *Analog-Digital-Wandler* geschieht. Der Vorgang der Quantisierung eines analogen zeitkontinuierlichen Signals ist in Abb. 2.20c gezeigt. Die zeitliche Diskretisierung des zeitkontinuierlichen Eingangssignals erfolgt mithilfe eines *Abtastglieds*, das die Signalamplitude zu bestimmten Zeitpunkten $t = kT$ erfasst. Der gesamte Umwandlungsvorgang vom analogen zeitkontinuierlichen Signal $u(t)$ (Abb. 2.20a) über ein analoges zeitdiskretes Signal $\hat{u}(kT)$ (Abb. 2.20b) in ein quantisiertes zeitdiskretes Signal $u(kT)$ (Abb. 2.20d) ist in Abb. 2.21a dargestellt.

Gleichermaßen muss das quantisierte zeitdiskrete Ausgangssignal $u(kT)$ (Abb. 2.20d) des Digtialrechners mithilfe eines *Digital-Analog-Wandlers* in ein analoges, also wertkontinuierliches, Signal $\hat{u}(kT)$ (Abb. 2.20b) zurückgewandelt werden. Ein *Halteglied* hält dazu die Signalamplitude für die Dauer T einer Abtastung auf einem konstanten Wert, sodass aus dem zeitdiskreten Signal



(a) Zeitkontinuierliches analoges Signal.

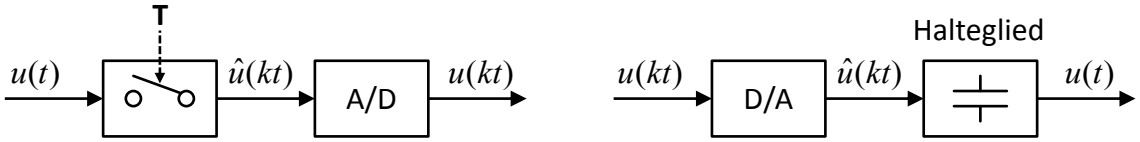
(b) Zeitdiskretes analoges Signal.

(c) Zeitkontinuierliches quantisiertes Signal.

(d) Zeitdiskretes quantisiertes Signal.

Abbildung 2.20: Diskretisierung und Quantisierung von Signalen nach [39].

ein zeitkontinuierliches Signal $u(t)$ mit treppenförmigem Verlauf entsteht. Dieser Sachverhalt ist in Abb. 2.21b gezeigt. [5]



(a) Analog-Digital-Wandlung mit Abtastung.

(b) Digital-Analog-Wandlung mit Halteglied.

Abbildung 2.21: Umwandlung von analogen zeitkontinuierlichen Signalen und quantisierten zeitdiskreten Signalen ineinander.

Die Abtastdauer T muss dabei nach dem Shannon'schen Abtasttheorem so gewählt werden, dass sie kleiner als die Hälfte der maximalen Periodendauer $T_{\max}$ des abzutastenden Signals ist. Es gilt also für die Abtastdauer

$$T < \frac{T_{\max}}{2}, \tag{2.52}$$

beziehungsweise mit der maximalen Frequenz $f_{\max} = 1/T_{\max}$

$$T < \frac{1}{2 \cdot f_{\max}}. \tag{2.53}$$

Hierdurch wird sichergestellt, dass das ursprüngliche zeitkontinuierliche Signal eindeutig aus der Folge von abgetasteten Signalamplituden rekonstruiert werden kann, sofern die Abtastung über einen genügend langen Zeitraum erfolgt. Wäre die Abtastdauer größer als obiger Grenzwert, so

wäre das rekonstruierte Signal niederfrequenter als das Ursprungssignal. Abb. 2.22 verdeutlicht diesen sogenannten *Aliasing-Fehler*. [35]

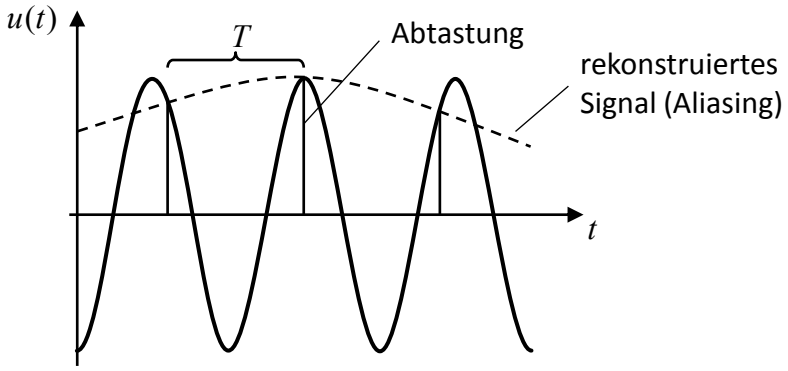


Abbildung 2.22: Aliasing-Fehler bei ungünstig gewählter Abtastdauer T. Das ursprüngliche Signal wird fehlerhaft rekonstruiert. Darstellung nach [35].

Ein mit der Quantisierung des Signals einhergehender Effekt ist der Quantisierungsfehler A_q, der durch die beschränkte Auflösung des Analog-Digital- bzw. Digital-Analog-Wandlers begründet ist. Die Auflösung bezeichnet dabei die Genauigkeit, mit der das analoge Signal auf den endlichen Wertevorrat des quantisierten Signals abgebildet wird. Sie wird in der Einheit bit angegebenen und bezeichnet die Anzahl der Werte im quantisierten Wertevorrat in Form der Stellenanzahl einer Binärzahl. Ein 10 bit-Analog-Digital-Wandler kann somit den analogen Wertebereich auf binäre Zahlen von 0000000000 bis 1111111111 oder in Dezimalschreibweise auf Zahlen von 0 bis 1023 abbilden. Entsprechend muss sich das Eingangssignal um den Betrag

$$u_q = \frac{u_{\max}}{2^{10}}, \tag{2.54}$$

das sogenannte Quantisierungsintervall, ändern, damit sich das quantisierte Signal um 1 bit ändert. Die Größe $u_{\max}$ stellt dabei den analogen Wertebereich der Eingangsgröße des Analog-Digital-Wandlers dar. Im schlechtesten Fall beträgt der Quantisierungsfehler, das heißt die Abweichung zwischen dem gemessenen analogen und dem zugehörigen digitalen Wert des Signals $u(t)$, genau die Hälfte des Quantisierungsintervalls u_q. Der Quantisierungsfehler des 10 bit-Analog-Digital-Wandlers beträgt also

$$A_q \leq \frac{u_{\max}}{2 \cdot 2^{10}}. \tag{2.55}$$

Dies entspricht etwa 0,0489 % des analogen Wertebereichs $u_{\max}$. [35]

2.9.2 Regelkreis mit zeitdiskretem Regler

Eine in der Praxis oft anzutreffende Regelkreisstruktur zeigt Abb. 2.23. Darin wird eine Regelstrecke, die zeitlich kontinuierliches Verhalten aufweist, von einem auf einem Digitalrechner implementierten zeitdiskreten Regler geregelt. Die Führungsgröße $W(z)$ wird hierbei auf dem

Digitalrechner vorgegeben, ist also zeitdiskret, was durch das Argument z der Größe verdeutlicht wird. Ebenfalls zeitdiskret ist das Reglergesetz $G_R(z)$. Entsprechend muss die vom Regler berechnete zeitdiskrete und quantisierte Stellgröße $U(z)$ mithilfe eines Digital-Analog-Wandlers in die analoge zeitkontinuierliche Stellgröße $U(s)$ gewandelt werden. Diese wirkt genau wie die Störgröße $Z(s)$ auf die zeitkontinuierliche Strecke ein. Die Ausgangsgröße $Y(s)$ des Systems wird über das Messglied $G_M(s)$, ein Abtastglied S_1 und einen Analog-Digital-Wandler in ein quantisiertes zeitdiskretes Signal $Y_M(z)$ umgewandelt, das innerhalb des Digitalrechners mit der Führungsgröße verrechnet und dem Regler als Eingangsgröße zugeführt wird. Zur Vereinfachung der Berechnung kann das zeitkontinuierliche Ausgangssignal $Y(s)$ als zeitdiskretes Signal $Y(z)$ betrachtet werden. Für diese Umwandlung ist ein gedachtes, in der realen Anordnung nicht vorhandenes Abtastglied, S_2 erforderlich.

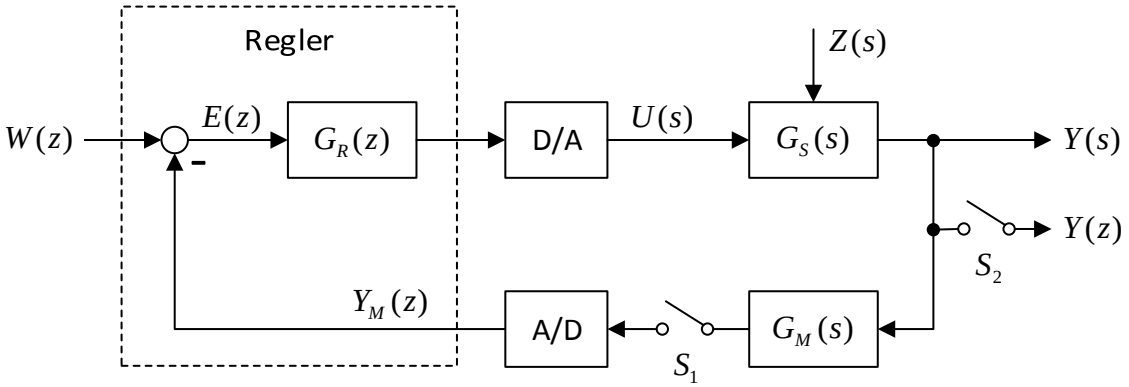


Abbildung 2.23: Blockschaltbild eines Regelkreises mit zeitdiskretem Regler nach [33].

Sämtliche Tastvorgänge und Umwandlungsvorgänge im Analog-Digital-Wandler und Digital-Analog-Wandler sowie die Berechnung der Stellgröße aus der Regelabweichung müssen theoretisch synchron ablaufen, sodass auf ein Eingangssignal $u(kT)$ zum Zeitpunkt $t = kT$ zum selben Zeitpunkt das Ausgangssignal $y(kT)$ folgt. Dies ist in der Praxis aufgrund der endlichen Rechenzeit des Digitalrechners und Verzögerungen in den Wandlern nicht realisierbar. Überschreitet die Verzögerungsdauer einen Wert von $0.2\,T$, sollte sie durch eine Totzeit in der Übertragungsfunktion des Reglers berücksichtigt werden. [33]

2.9.3 Mathematische Beschreibung zeitdiskreter Systeme

In den folgenden Abschnitten soll in aller Kürze auf die mathematische Behandlung zeitdiskreter Systeme eingegangen werden. Es werden analog zu den kontinuierlichen Systemen die Differenzengleichung zur Beschreibung im Zeitbereich und die z-Übertragungsfunktion zur Beschreibung im Bildbereich sowie die z-Transformation als zeitdiskrete Entsprechung zur Laplace-Transformation eingeführt. Ziel ist die Bereitstellung mathematischer Formalismen für den quasikontinuierlichen Reglerentwurf in Abschnitt 2.9.5. Umfassendere Beiträge zur mathematischen Theorie zeitdiskreter Systeme liefern beispielsweise [5, 7, 30].

2.9.3.1 Rekursive Differenzengleichung

Die mathematische Behandlung eines zeitdiskreten dynamischen Systems im Zeitbereich erfolgt analog zu der in Abschnitt 2.3 eingeführten linearen Differentialgleichung n. Ordnung in Form einer *rekursiven Differenzengleichung*. Um diese zu berechnen, wird die Zeitableitung einer Größe durch einen Differenzenquotienten angenähert

$$\dot{y}(t) = \frac{dy(t)}{dt} \approx \frac{\Delta y}{T} = \frac{y(kT) - y\left((k-1)T\right)}{T}. \tag{2.56}$$

Im Folgenden soll zur Verbesserung der Übersichtlichkeit die Abtastzeit T im Argument des Signals nicht mehr mitgeschrieben werden. Durch Ersetzen aller Differentiale durch Differenzenquotienten ergibt sich aus Gleichung (2.3) die Differenzengleichung n. Ordnung

$$\begin{aligned} y(kT) + a_1 y(k-1) + a_2 y(k-2) + \ldots + a_n y(k-n) = \\ b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) + \ldots + b_m u(k-m), \quad k = 1,2,3,\ldots \end{aligned} \tag{2.57}$$

oder kompakter notiert

$$y(k) = \sum_{j=0}^{m} b_j u(k-j) - \sum_{i=1}^{n} a_i y(k-i), \quad k = 1,2,3,\ldots. \tag{2.58}$$

Dabei gilt $u(k-j) = y(k-i) = 0$, wenn $k-j < 0$ bzw. $k-i < 0$, das heißt die Signale verschwinden, wenn ihr Argument kleiner null ist. Weiterhin gilt auch hier die Kausalitätsbedingung $n > m$. Andernfalls müssten zur Berechnung des Ausgangssignals $y(k)$ zukünftige Eingangswerte $u(k+j)$ mit $j > 0$ herangezogen werden. Der Anfangswert kann in der Form $y(0) = y_0$ berücksichtigt werden.

Diese Art der Notation des Systemverhaltens in einer rekursiven Gleichung eignet sich hervorragend für die Berechnung eines Ausgangssignals $y(k)$ für ein gegebenes Eingangssignal $u(k)$ mithilfe eines rekursiven Algorithmus auf einem Digitalrechner. [5, 37]

2.9.3.2 Die z-Transformation

Wie bereits in Abschnitt 2.3 beschrieben, bietet eine Transformation vom Zeit- in den Bildbereich einige Vorteile. Die bei zeitkontinuierlichen Systemen verwendete Laplace-Transformation ist jedoch für zeitdiskrete Systeme ungeeignet. Aus diesem Grund wird für zeitdiskrete Systeme die *z-Transformation* eingeführt, die ein Signal vom diskreten Zeitbereich in den z-Bereich abbildet.

Hierzu wird ein zeitkontinuierliches Signal $f(t)$ als Reihe von Dirac-Impulsen zu den Zeitpunkten kT, die mit dem Funktionswert zu diesen Zeitpunkten $f(kT)$ gewichtet werden, dargestellt

$$f(t) = \sum_{k=0}^{\infty} f(kT)\delta(t - kT). \tag{2.59}$$

Eine anschließende Laplace-Transformation führt auf

$$F(s) = \sum_{k=0}^{\infty} f(kT)\mathrm{e}^{-kTs}. \tag{2.60}$$

Wird hier der Ausdruck e^{Ts} durch die Variable z substituiert, erhält man die z-Transformierte $F(z)$ des zeitdiskreten Signals $f(k)$ in Form einer unendlichen Potenzreihe in der Variablen z^{-1}

$$F(z) = \mathcal{Z}\left\{f(k)\right\} = \sum_{k=0}^{\infty} f(k)z^{-k}. \tag{2.61}$$

Zur Berechnung der Summe der unendlichen geometrischen Reihe kann dabei die Beziehung

$$\sum_{k=0}^{\infty} p\left(z^{-1}\right)^k = \frac{p}{1-z^{-1}} \tag{2.62}$$

verwendet werden. In der Praxis kommen jedoch genau wie bei der Laplace-Transformation Korrespondenztafeln, wie sie in [37] zu finden sind, zum Einsatz. In diesen sind für zeitdiskrete Signale $f(k)$ korrespondierende z-Transformierte $F(z)$ angegeben.

Für die z-Transformation gelten ebenfalls einige grundlegende Rechengesetze, die in [37] nachgelesen werden können. Angegeben werden soll hier nur der Verschiebungssatz

$$f(k-n) = z^{-n}F(z), \tag{2.63}$$

der für $n \geq 0$ gilt. [5, 37]

2.9.3.3 Die z-Übertragungsfunktion

Wendet man auf die einzelnen Terme der rekursiven Differenzengleichung aus Gleichung (2.57) den Verschiebungssatz an und nutzt die Linearität der z-Transformation aus, ergibt sich folgender Ausdruck im Bildbereich

$$\begin{aligned} Y(z) + a_1 z^{-1}\,Y(z) + a_2 z^{-2}\,Y(z) + \ldots + a_n z^{-n}\,Y(z) = \\ b_0\,U(z) + b_1 z^{-1}\,U(z) + b_2 z^{-2}\,U(z) + \ldots + b_m z^{-m}\,U(z), \end{aligned} \tag{2.64}$$

der durch algebraische Umformung auf die *z-Übertragungsfunktion*

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n}} \tag{2.65}$$

führt. [5]

2.9.3.4 Berechnung der z-Übertragungsfunktion aus der kontinuierlichen Übertragungsfunktion

Für den im nächsten Abschnitt vorgestellten quasikontinuierlichen Reglerentwurf ist es wichtig, eine zuvor bestimmte zeitkontinuierliche Übertragungsfunktion $G(s)$ in eine zeitdiskrete z-Übertragungsfunktion $G(z)$ umzuwandeln.

Hierzu wird, wie in Abb. 2.24 dargestellt, ein System aus gemischt zeitkontinuierlichen und zeitdiskreten Elementen betrachtet. Ziel ist es, die gesamte Übertragungsfunktion des Systems in zeitdiskreter Form $G(z)$ anzugeben. Im s-Bereich kann zunächst die Übertragungsfunktion

$$G_H(s)G(s) = \frac{1 - \mathrm{e}^{-sT}}{s} G(s) = \frac{G(s)}{s} - \frac{G(s)}{s} \mathrm{e}^{-sT} \tag{2.66}$$

mit der Übertragungsfunktion $G_H(s)$ des Halteglieds angegeben werden. Zur Umrechnung in den z-Bereich muss nun $G_H(s)G(s)$ mit der inversen Laplace-Transformation in den Zeitbereich zurücktransformiert und anschließend zu den Zeitpunkten $t = kT$ abgetastet werden. Das so entstandene zeitdiskrete Signal kann abschließend in den z-Bereich transformiert werden. Es gilt somit für die z-Übertragungsfunktion

$$G(z) = \frac{Y(z)}{U(z)} = \mathcal{Z}\left\{ \mathcal{L}^{-1}\left\{ \frac{G(s)}{s} - \frac{G(s)}{s} \mathrm{e}^{-sT} \right\}\Bigg|_{t=kT} \right\}. \tag{2.67}$$

Die Transformation vom Laplace- in den z-Bereich kann auch mithilfe des $\mathscr{L}$-Operators als

$$F(z) = \mathcal{Z}\left\{ \mathcal{L}^{-1}\left\{F(s)\right\}\Big|_{t=kT} \right\} = \mathscr{L}\left\{F(s)\right\} \tag{2.68}$$

geschrieben werden, sodass gilt

$$G(z) = \mathscr{L}\left\{ \frac{G(s)}{s} - \frac{G(s)}{s} \mathrm{e}^{-sT} \right\}. \tag{2.69}$$

Durch Ausnutzung der Tatsache, dass

$$\mathcal{L}^{-1}\left\{ \frac{G(s)}{s} \right\}\Bigg|_{t=kT} = h(kT) \tag{2.70}$$

mit der zeitdiskreten Sprungantwort $h(kT)$ gilt, kann unter Verwendung des Verschiebungssatzes der z-Transformation für die z-Übertragungsfunktion des in Abb. 2.24 dargestellten Systems der Ausdruck

$$G(z) = \frac{Y(z)}{U(z)} = \mathcal{Z}\left\{h(kT) - h\left((k-1)T\right)\right\} = H(z) - z^{-1} H(z) = \frac{z-1}{z} H(z) \tag{2.71}$$

angegeben werden. Wegen

$$H(z) = \mathcal{Z}\left\{h(kT)\right\} = \mathscr{L}\left\{ \frac{G(s)}{s} \right\} \tag{2.72}$$

folgt schließlich ein direkter Zusammenhang zur Berechnung der z-Übertragungsfunktion $G(z)$ aus der Übertragungsfunktion $G(s)$ des kontinuierlichen Systems:

$$G(z) = \frac{z-1}{z} \mathscr{L}\left\{\frac{G(s)}{s}\right\}. \tag{2.73}$$

In [5] ist eine Korrespondenztabelle für den $\mathscr{L}$-Operator gegeben, die für verschiedene Funktionen im s-Bereich korrespondierende Funktionen im z-Bereich enthält. [5]

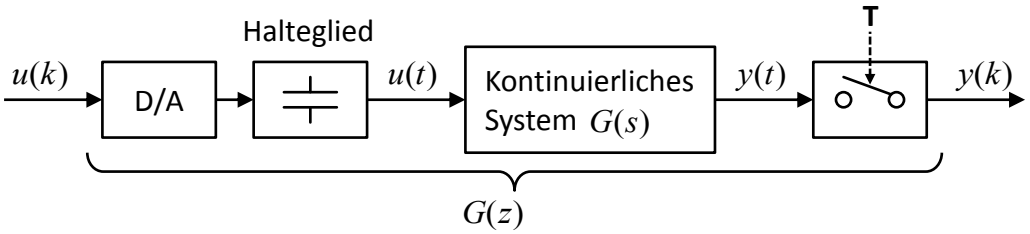


Abbildung 2.24: Übertragungssystem mit zeitlich kontinuierlicher Strecke und zeitdiskreten Ein- und Ausgangssignalen nach [5].

Eine zweite Möglichkeit zur Berechnung der z-Übertragungsfunktion aus der kontinuierlichen Übertragungsfunktion ist die Tustinsche Näherung

$$G(z) = G(s)\Big|_{s = \frac{2}{T}\frac{z-1}{z+1}}. \tag{2.74}$$

2.9.4 Umwandlung der z-Übertragungsfunktion in die Differenzengleichung

Zur Bestimmung der Differenzengleichung aus der z-Übertragungsfunktion muss genau umgekehrt zur in Abschnitt 2.9.3.3 durchgeführten Aufstellung vorgegangen werden. Dazu wird die z-Übertragungsfunktion aus Gleichung (2.65) zunächst durch kreuzweises Ausmultiplizieren in die Form aus Gleichung (2.64) gebracht und anschließend unter Ausnutzung der Linearität der z-Transformation und des inversen Verschiebungssatzes

$$z^{-n}F(z) = f(k-n) \tag{2.75}$$

in die rekursive Differenzengleichung aus Gleichung (2.57) überführt. [5]

2.9.5 Entwurf zeitdiskreter Regler

Zeitdiskrete Regler lassen sich wahlweise durch spezielle Entwurfsmethoden für zeitdiskrete Regler, wie beispielsweise Dead-Beat-Verfahren und spezielle zeitdiskrete Kompensationsverfahren, oder aber quasikontinuierlich entwerfen. Auf die Klasse der zeitdiskreten Entwurfsverfahren soll nicht näher eingegangen werden. Es sei an dieser Stelle lediglich auf die weiterführende Literatur [7] verwiesen. Stattdessen soll der quasikontinuierliche Reglerentwurf, der aufgrund seiner Einfachheit in der Praxis oft Anwendung findet, näher erläutert werden. Hierbei wird ein zeitdiskreter

Regler mit einer so kleinen Abtastzeit T betrieben, dass der Einfluss der Signaldiskretisierung vernachlässigt und näherungsweise von einem kontinuierlichen Verhalten des Reglers ausgegangen werden kann. Die zum Entwurf eines solchen quasikontinuierlichen Reglers erforderlichen Schritte sind in Abb. 2.25 dargestellt.

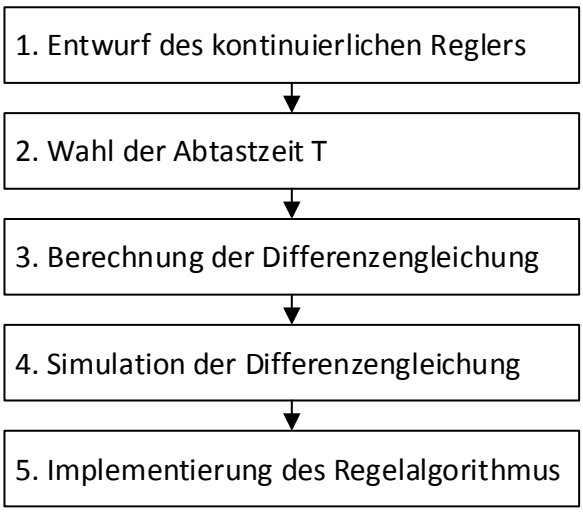


Abbildung 2.25: Vorgehen beim quasikontinuierlichen Reglerentwurf nach [33].

Dabei wird zunächst ein kontinuierlicher Regler durch eine beliebige kontinuierliche Entwurfsmethode entwickelt und anschließend eine geeignete Abtastzeit T entsprechend der Beziehung

$$T \leq 0{,}1 \frac{2\pi}{\omega_d} \tag{2.76}$$

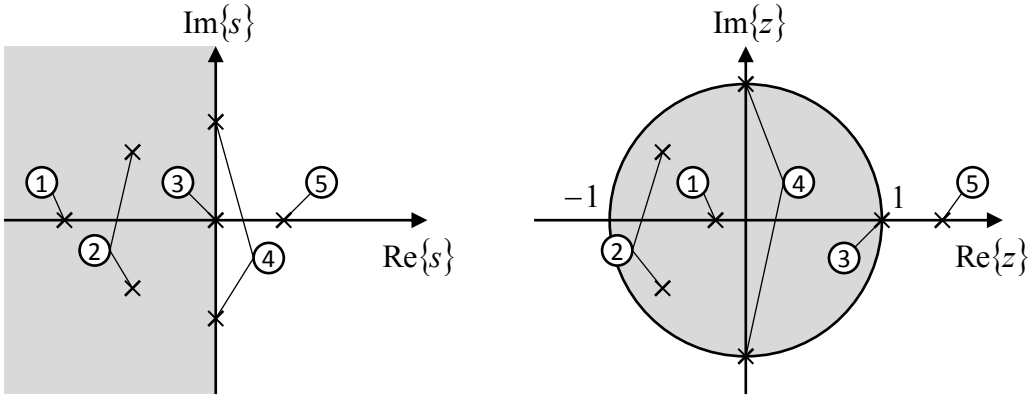
gewählt. Die Durchtrittsfrequenz ω_d ist dabei, wie in Abschnitt 2.8.4 ausführlich erläutert, diejenige Frequenz, bei der der Amplitudengang des offenen Regelkreises im Bode-Diagramm die 0 dB-Linie schneidet. Im nächsten Schritt wird nun die kontinuierliche Übertragungsfunktion $G_R(s)$ des Reglers unter Zuhilfenahme von Gleichung (2.73) oder durch Verwendung der Tustinschen Näherung in die z-Übertragungsfunktion $G_R(z)$ transformiert und hieraus, wie in Abschnitt 2.9.4 beschrieben, die rekursive Differenzengleichung bestimmt. Diese kann anschließend beispielsweise in Simulink implementiert und zwecks Validierung simuliert und schließlich in einem Digitalrechner implementiert werden. [33]

2.9.6 Stabilität zeitdiskreter Systeme

Äquivalent zur Stabilität kontinuierlicher Systeme ist ein zeitdiskretes System genau dann stabil, wenn zu jedem diskreten Eingangssignal $u(k)$ ein beschränktes diskretes Ausgangssignal $y(k)$ gehört.

Wie in Abschnitt 2.8.1 ausführlich beschrieben, kann die Stabilität eines dynamischen Systems anhand der Lage der Pole der Übertragungsfunktion beurteilt werden. Abb. 2.26a zeigt die Lage der Pole eines zeitkontinuierlichen Systems in der komplexen s-Ebene. Durch die Transformation in den zeitdiskreten z-Bereich mittels der z-Transformation wird die linke Hälfte der s-Ebene,

wie in Abb. 2.26b dargestellt, auf das Innere des Einheitskreises in der z-Ebene abgebildet. Die Imaginärachse entspricht in der z-Ebene dem Rand des Einheitskreises. Liegen alle Pole der z-Übertragungsfunktion $G(z)$ im Innern des Einheitskreises, ist das zeitdiskrete Übertragungssystem asymptotisch stabil. Liegt mindestens ein einfacher Pol auf dem Rand des Einheitskreises und alle anderen Pole in seinem Innern, ist das System grenzstabil. In allen anderen Fällen ist es instabil. [5, 7]



(a) Pole in der s-Ebene. **(b)** Pole in der z-Ebene.

Abbildung 2.26: Stabilitätsgebiete in der s- und z-Ebene nach [5].

3 Systemanalyse und Modellierung

In diesem Kapitel steht die Modellierung des zu regelnden Systems im Vordergrund. Dazu wird nach einer kurzen Vorstellung des zu regelnden Systems zunächst eine Einführung in die Modellierung dynamischer Systeme gegeben, indem auf verschiedene Modellarten und Eigenschaften von Modellen eingegangen wird. Es werden mögliche Eigenschaften dynamischer Systeme besprochen und das Vorgehen bei der Modellierung erläutert. Nach Klärung der Grundlagen werden das Ziel der Modellierung festgelegt und vereinfachende Annahmen getroffen. Anschließend folgt eine ausführliche Beschreibung der Regelstrecke in verbaler Form. Es wird zunächst die Gesamtfunktion beschrieben und hiernach näher auf die einzelnen Bestandteile der Strecke eingegangen. Im Rahmen dieser Beschreibung erfolgt zudem eine theoretische Modellierung der physikalischen Zusammenhänge in der thermischen Strecke. Nach der verbalen Beschreibung wird das System in Form von Blockschaltbildern abstrahiert. Erst danach erfolgen die Vermessung des Systemverhaltens im Zeitbereich und auf Basis der erhaltenen Messdaten eine Modellbildung des zu regelnden Systems mit dem Ziel, die erstellten Modelle für den Reglerentwurf nutzen zu können. Die Modelle werden anhand weiterer Messdaten validiert und ihre Eigenschaften im Detail erläutert. Abschließend wird ein Fazit zur Modellierung gezogen.

3.1 Vorstellung des Systems

Gegenstand der vorliegenden Arbeit ist die in Abb. 3.1 dargestellte Spannvorrichtung. Auf ihr können Mikrobauteile mithilfe dünner Filme aus gefrorenem Wasser oder aufgeschmolzenem und erstarrtem Wachs fixiert werden. Hierzu wird zunächst eine geringe Menge Wasser oder Wachs auf die Spannplatte aufgegeben, das Bauteil platziert und anschließend das Wasser gefroren beziehungsweise das Wachs aufgeschmolzen und anschließend erstarrt. Das Bauteil haftet nun aufgrund von Adhäsionskräften an der Spannplatte und kann beispielsweise spanend bearbeitet werden. Die Abkühlung der Spannplatte erfolgt dabei mithilfe eines Peltier-Elements, das einen Wärmestrom von der Spannplatte in einen angeschlossenen Wärmetauscher transportiert, sobald es von einem elektrischen Gleichstrom durchflossen wird. Durch Umpolung des elektrischen Stroms wird der Wärmestrom in die entgegengesetzte Richtung transportiert und die Spannplatte heizt sich auf. Zur exakten Einstellung der Temperaturen der Spannplatte bei diesen beiden Vorgängen ist der Einsatz einer Regelung erforderlich. Genaueres zur Funktionsweise des Systems wird in Abschnitt 3.5 erläutert.

Ein möglicher Einsatzzweck der Spannvorrichtung ist die in Abb. 3.2 gezeigte Werkzeugmaschine für die Fertigung von Mikrobauteilen. Mehrere dieser Würfel können miteinander in beliebiger Weise zu einem flexiblen Fertigungssystem, durch das sowohl Werkzeuge als auch Werkstücke mithilfe eines Fördersystems transportiert werden, verschaltet werden. Hierzu wird die gesamte Spanneinheit inklusive des darauf eingespannten Bauteils durch das Fertigungssystem befördert,

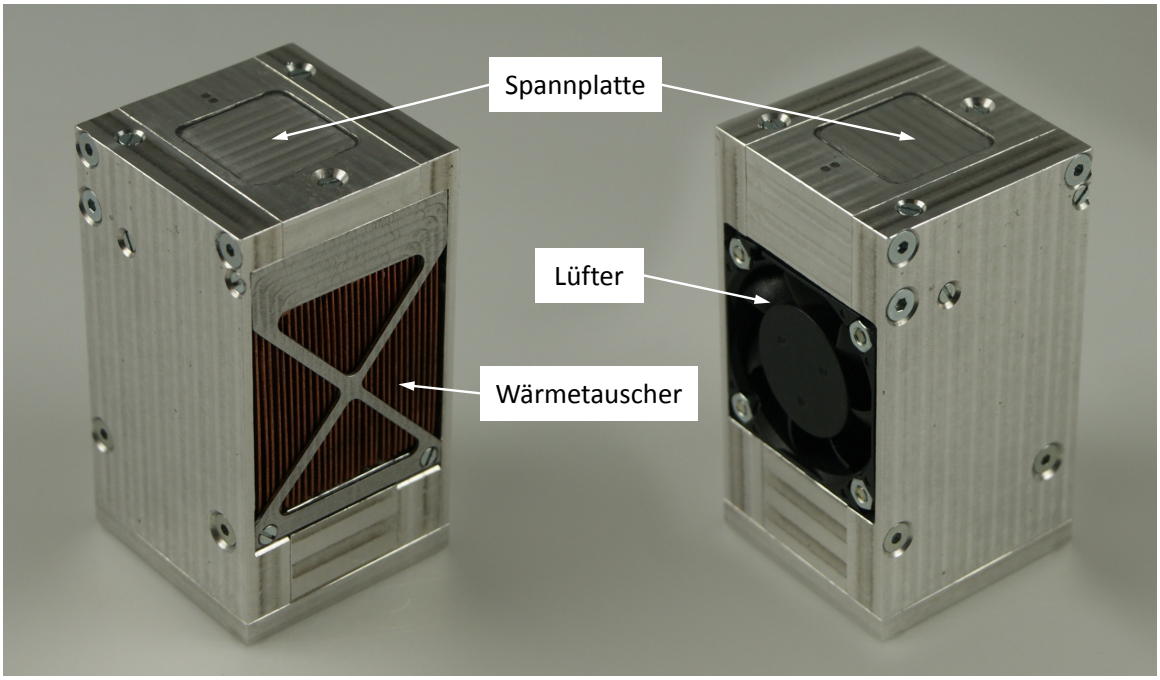


Abbildung 3.1: Front- und Rückansicht der Gefrierspanneinheit. [1]

sodass in jeder Station ein anderer Arbeitsschritt an einem Bauteil durchgeführt werden kann. Dazu ist die Spanneinheit an ein weiteres Modul, das über eine einheitliche Schnittstelle mit den einzelnen Fertigungswürfeln und dem Transportsystems verbunden werden kann, angekoppelt. Neben der mechanischen Kopplung dient dieses Modul zur Übertragung von elektrischer Energie und Informationen zur Ansteuerung der Spanneinheit und Identifikation des eingespannten Bauteils. Somit ist es möglich, jedes Bauteil auf einem individuellen Pfad durch das Fertigungssystem zu transportieren.

3.2 Grundlegendes zur Modellierung

Ziel der Modellierung eines technischen Systems ist die vollständige Beschreibung des Systemverhaltens in Form eines abstrakten mathematischen Modells, wie beispielsweise einer Differentialgleichung oder einer Übertragungsfunktion. Das System kann dabei als vollständig identifiziert betrachtet werden, wenn

- das statische Verhalten,
- das dynamische Verhalten und
- der Aussteuerbereich

des vorliegenden, zu identifizierenden Systems durch das Modell mit hinreichender Genauigkeit dargestellt werden. Ein solches Modell ermöglicht die Simulation des Systemverhaltens mithilfe rechnerbasierter Methoden und ersetzt somit die Notwendigkeit der Durchführung von Experimenten zur Bestimmung des Systemzustandes bei gegebenen Eingangssignalen. Dies ist

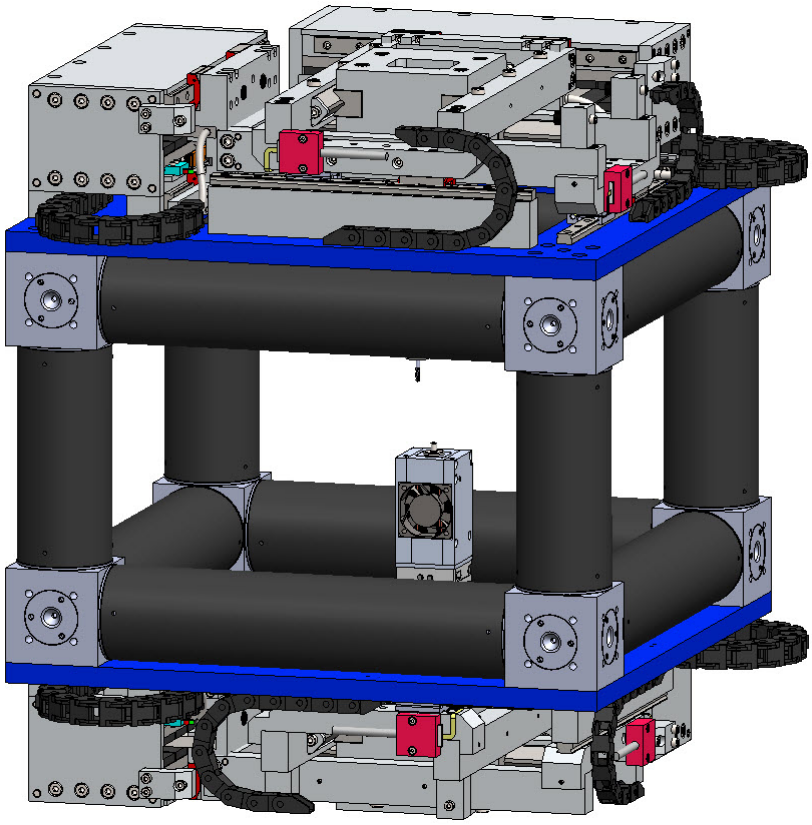


Abbildung 3.2: Fertigungswürfel für die Mikrozerspanung mit eingebauter Gefrierspanneinheit. [2]

einerseits sicherer, andererseits kann das System selbst bei Überschreitung von Grenzzuständen nicht zerstört werden. Weiterhin können Vorgänge im realen System bei einer Simulation zeitlich gedehnt oder verkürzt und somit einfach beobachtet werden. Zudem ist es je nach Modellform möglich, gezielt Modellparameter oder die Struktur des Modells zu verändern und somit einen Erkenntnisgewinn über die Funktionsweise des Systems zu erzielen. Der Hauptgrund für die Erstellung eines mathematischen Modells ist jedoch die Notwendigkeit eines Modells für die in Kapitel 4 angewendeten Verfahren zum Reglerentwurf. [3]

3.2.1 Arten und Eigenschaften von Modellen

Bei der Modellierung werden zunächst zwei grundsätzlich verschiedene Herangehensweisen unterschieden, die *experimentelle Modellierung* und die *theoretische Modellierung*. Bei der in Abb. 3.3 veranschaulichten experimentellen Modellierung wird das dynamische System mit bekannten, deterministischen Eingangssignalen beaufschlagt und die Systemantwort messtechnisch erfasst. Beispiele für die so erhaltenen *nichtparametrischen Modelle* sind die in Abschnitt 2.5 beschriebene Übergangs- und Gewichtsfunktion sowie der in Abschnitt 2.6.2 eingeführte Frequenzgang. Durch Anwendung bestimmter Algorithmen ist es zudem möglich, aus den so gewonnenen Daten *parametrische Modelle*, wie beispielsweise die Übertragungsfunktion oder die das System beschreibende Differentialgleichung, zu gewinnen. Die zweite Herangehensweise zur Modellierung, die *theoretische Modellierung*, führt grundsätzlich zu parametrischen Modellen. Hierbei wird

die System-Differentialgleichung durch Auswertung von Bilanzen oder Beachtung physikalischer Gesetze direkt aufgestellt. Voraussetzung hierfür ist tiefgehendes physikalisches Verständnis des Systems und der in ihm ablaufenden Vorgänge. Dementsprechend ist die theoretische Modellierung aufwändiger, das erhaltene Modell jedoch in der Regel deutlich hochwertiger, da es das Systemverhalten exakt darstellt. [3, 4]

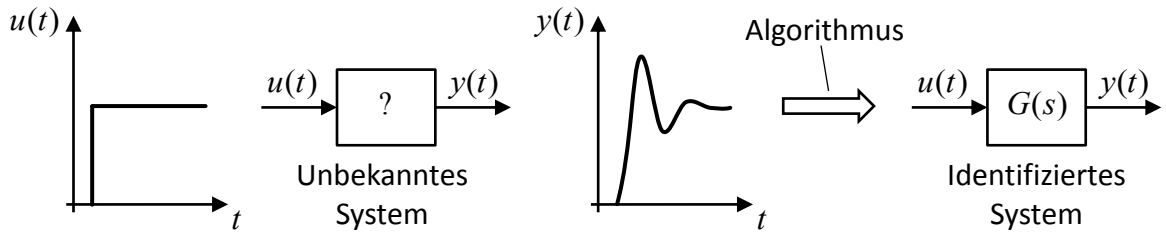


Abbildung 3.3: Vorgehen bei der experimentellen Systemidentifikation.

Modelle können durch bestimmte Eigenschaften klassifiziert werden. So wird zunächst einmal, wie bereits in Kapitel 2, nach dem Betrachtungsbereich unterschieden. Dabei können Modelle im Zeitbereich, im Bildbereich oder im Frequenzbereich vorliegen. Weiterhin unterschieden werden parametrische und nichtparametrische Modelle, Blackbox- und Glasbox-Modelle sowie kontinuierliche und zeitdiskrete Modelle. Auf diese Eigenschaften soll im Folgenden kurz eingegangen werden.

3.2.1.1 Parametrische und nichtparametrische Modelle

Nichtparametrische Modelle beschreiben Übertragungsglieder in Form von Graphen oder Wertetabellen, beispielsweise das Systemverhalten bei einem sprungförmigen Eingangssignal. Im Gegensatz dazu wird das Systemverhalten bei parametrischen Modelle analytisch in Form von Gleichungen beschrieben. Parametrische Modelle können beliebig zwischen Zeit- und Bildbereich transformiert werden und erlauben die Berechnung jedes nichtparametrischen Modells. Da parametrische Modelle durch theoretische Modellierung gewonnen werden, ist deren Erstellungsaufwand größer. [3, 5]

3.2.1.2 Blackbox- und Glasbox-Modelle

Die Unterscheidung zwischen Blackbox- und Glasbox-Modell erfolgt nur bei parametrischen Modellen. Bei Glasbox-Modellen bildet das Modell die Systemstruktur und relevante Wirkungselemente, das heißt die innere Struktur des realen Systems, möglichst originalgetreu ab. Entsprechend sind sowohl das innere als auch das äußere Modellverhalten an das Übertragungssystem angepasst. Im Gegensatz dazu bilden Blackbox-Modelle, die auch Eingangs-/Ausgangs-Modelle genannt werden, lediglich das äußere Verhalten des Übertragungssystems ab. Das Modell reagiert folglich auf Signale am Eingang mit demselben Ausgangssignal wie das reale System, die Vorgänge im Innern des realen Systems werden jedoch nicht weiter beachtet. Ein typisches Blackbox-Modell ist beispielsweise die Übertragungsfunktion des Systems. [3]

3.2.1.3 Kontinuierliche und zeitdiskrete Modelle

Wie bereits in Abschnitt 2.9 erläutert, kann bei Übertragungssystemen zwischen zeitdiskreten und zeitkontinuierlichen Systemen unterschieden werden. Entsprechend existieren zeitdiskrete und zeitkontinuierliche Modelle, wobei letztere öfter anzutreffen sind, da physikalische Regelstrecken, wie auch das hier vorliegende thermische System, in der Regel zeitkontinuierlich sind. Eine Umwandlung von zeitdiskreten in zeitkontinuierliche Modelle und umgekehrt ist möglich und in [5] näher beschrieben. [3]

3.2.2 Eigenschaften von Übertragungssystemen

Zur Modellierung dynamischer Systeme ist es entscheidend, sich darüber im Klaren zu sein, durch welche Eigenschaften ein solches System beschrieben wird und welche Ausprägungen dieser Eigenschaften auftreten. Einen Überblick hierüber gibt Tabelle 3.1.

Tabelle 3.1: Eigenschaften dynamischer Systeme nach [6].

Eigenschaft	Ausprägungen		
Ortsabhängigkeit	konzentriert	$\leftrightarrow$	verteilt
Zeitabhängigkeit	zeitinvariant	$\leftrightarrow$	zeitvariant
Kontinuität	zeitkontinuierlich	$\leftrightarrow$	zeitdiskret
Stabilität	stabil	$\leftrightarrow$	instabil
Linearität	linear	$\leftrightarrow$	nichtlinear
Ein-/Ausgangsgrößen	Eingrößensystem	$\leftrightarrow$	Mehrgrößensystem
Kausalität	kausal	$\leftrightarrow$	nichtkausal

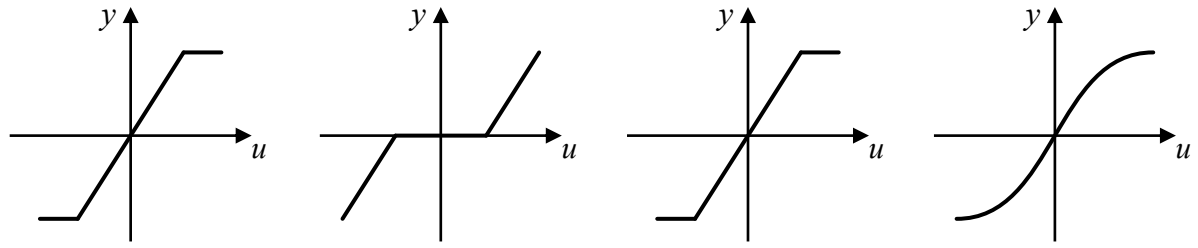
Bei konzentrierten Systemen ist dabei die Zeit die einzige differentielle Größe. Alle anderen Systemparameter sind räumlich gemittelte Werte. Bei verteilten Systemen wird im Gegensatz dazu auch die räumliche Verteilung der Parameter berücksichtigt, was eine wesentlich aufwändigere Systembeschreibung in Form von partiellen Differentialgleichungen nötig macht.

Das Zeitverhalten beschreibt, ob die Systemparameter explizit von der Zeit abhängen, wie es bei zeitvarianten Systemen der Fall ist, oder ob die Zeit nur als differentielle Größe in der System-Differentialgleichung auftritt. Letztere Systeme werden als zeitinvariant bezeichnet.

Wie auch schon in Abschnitt 2.9 detailliert erläutert, sind die das System beschreibenden Größen bei zeitkontinuierlichen Systemen zu allen Zeitpunkten t definiert. Bei zeitdiskreten Systemen hingegen sind sie nur zu bestimmten diskreten Zeitpunkten $t = kT$ festgelegt.

Eine sehr wichtige Eigenschaft dynamischer Systeme, insbesondere geschlossener Regelkreise, ist die bereits in Abschnitt 2.8 detailliert betrachtete Stabilität. Ist das Übertragungssystem stabil, kehrt die Ausgangsgröße des Systems nach einer Auslenkung aus der Ruhelage wieder in diese zurück (asymptotische Stabilität) oder überschreitet einen endlichen Wert nicht. Bei instabilen

Systemen führt eine Auslenkung aus der Ruhelage hingegen zu einer unendlichen Zunahme der Ausgangsgröße des Systems.



(a) Amplitudenbegrenzung. **(b)** Totzeit. **(c)** Hysterese. **(d)** Nichtlineare Kennlinie.

Abbildung 3.4: Statische Kennlinien von dynamischen Systemen mit verschiedenen Arten von Nichtlinearitäten nach [7].

Ein lineares System mit Eingangssignal $u(t)$ und Ausgangssignal $y = f\left(u(t)\right)$ erfüllt das Superpositionsprinzip

$$f\left(a_1 u_1(t) + a_2 u_2(t)\right) = a_1 f\left(u_1(t)\right) + a_2 f\left(u_2(t)\right). \tag{3.1}$$

Ein solches System hat als statische Kennlinie eine Gerade. Demgegenüber erfüllen nichtlineare Systeme, die in der Praxis sehr oft auftreten, das Superpositionsprinzip nicht. Unterschieden wird dabei zwischen Systemen mit unstetigem nichtlinearem Verhalten, wie es bei Systemen mit Amplitudenbegrenzung, Totzonen und Hysterese der Fall ist, und Systemen mit stetigem nichtlinearem Verhalten. Letztere weisen eine glatte, das heißt knickfreie, statische Kennlinie auf. Abb. 3.4 zeigt die statischen Kennlinien von Systemen mit verschiedenen Arten von Nichtlinearitäten. Eine Linearisierung stetiger nichtlinearer dynamischer Systeme um einen Arbeitspunkt ist möglich und wurde in Abschnitt 2.7.2 erläutert. Wird eine experimentelle Modellierung durch Messen des Ein- und Ausgangsverhaltens eines im Arbeitspunkt betriebenen nichtlinearen Systems durchgeführt, kann durch Verwendung kleiner Signalamplituden direkt ein lineares Modell gewonnen werden. Somit kann durch geschickte Wahl der Signalamplituden bei der Messung auf eine anschließende analytische Linearisierung verzichtet werden.

Ebenfalls charakterisierend für Systeme ist die Anzahl der Ein- und Ausgangsgrößen. Haben Systeme lediglich einen Ein- und einen Ausgang (SISO-Systeme), so kann ihr Übertragungsverhalten durch eine einzelne Übertragungsfunktion beschrieben werden. Besitzen Systeme, wie in Abb. 3.5 dargestellt, mehrere Ein- und Ausgänge, spricht man von Mehrgrößensystemen (MIMO-Systeme). Hier können die Ein- und Ausgangsgrößen beliebig miteinander verkoppelt sein. Die mathematische Beschreibung solcher Systeme im Bildbereich kann ebenfalls durch Übertragungsfunktionen erfolgen, jedoch muss nun das Übertragungsverhalten von jedem Eingang auf jeden Ausgang einzeln betrachtet werden. Dazu wird eine Übertragungsmatrix

$$\boldsymbol{G}(s) = \begin{pmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{pmatrix} \tag{3.2}$$

aufgestellt, deren Komponenten die Übertragungsfunktionen der einzelnen Ein- und Ausgangsgrößen untereinander sind:

$$G_{11}(s) = \frac{Y_1(s)}{U_1(s)}, \quad G_{12}(s) = \frac{Y_1(s)}{U_2(s)}, \quad G_{21}(s) = \frac{Y_2(s)}{U_1(s)}, \quad G_{22}(s) = \frac{Y_2(s)}{U_2(s)}. \tag{3.3}$$

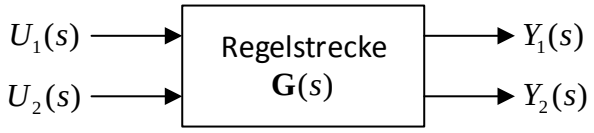


Abbildung 3.5: Blockschaltbild eines MIMO-Systems mit zwei Ein- und zwei Ausgängen.

Die letzte Eigenschaft dynamischer Systeme, die in Tabelle 3.1 angesprochen wird, ist die Kausalität des Übertragungssystems. In Abschnitt 2.3 wurde die Kausalität bereits in Zusammenhang mit der System-Differentialgleichung genannt, bei der die Ableitungen der Eingangsgrößen $u(t)$ immer eine niedrigere Ableitungsordnung m als die Ausgangsgrößen $y(t)$ mit Ordnung n besitzen müssen. Anschaulich gesprochen besagt die Kausalität, dass die Ausgangsgröße $y(t)$ eines kausalen Systems zu einem beliebigen Zeitpunkt t_0 nur vom Verlauf der Eingangsgröße $u(t)$ bis zu diesem Zeitpunkt abhängen kann. Nichtkausale Systeme erfüllen diese Forderung nicht. In der Realität existieren jedoch nur kausale Systeme. [3, 6]

3.2.3 Vorgehen bei der Modellierung

Die Modellierung der vorliegenden Regelstrecke erfolgt anhand der in Abb. 3.6 dargestellten Schritte. Entsprechend wird sich der weitere Aufbau dieses Kapitels stark am hier beschriebenen Vorgehen orientieren. Dabei wird zunächst das Modellierungsziel beschrieben, indem festgelegt wird, welcher Teil des vorliegenden technischen Systems die zu regelnde Strecke ist und über welche Ein- und Ausgangsgrößen diese verfügt. Die Regelungsaufgabe bestimmt ferner, welche Anforderungen an das Modell hinsichtlich Reglerentwurf und Modellgenauigkeit gestellt werden. Anschließend werden Modellannahmen ausgewählt. Dabei wird festgelegt, welche physikalischen Phänomene, Teilsysteme und Wechselwirkungen durch das Modell abgebildet werden müssen und welche bei der Modellierung vernachlässigt werden können. Im dritten Schritt wird die Regelstrecke zum Zwecke einer einfachen Erkennung von Teilsystemen, Funktionen und strukturellen Verknüpfungen verbal beschrieben. Hieraus wird anschließend das Blockschaltbild des Systems aufgestellt, indem Funktionslemente in Form von Blöcken und Verknüpfungen durch Signale dargestellt werden. Durch diese Zergliederung in Teilsysteme verringert sich die Komplexität der nachfolgenden Modellierungsschritte deutlich. So wird im fünften Schritt für jedes Teilelement des Systems ein quantitatives Modell durch wahlweise theoretische oder experimentelle Analyse bestimmt. Ist das Modell vollständig aufgestellt, empfiehlt sich eine Validierung des Modells. Hierzu können eine Simulation des Modells durchgeführt und die Ergebnisse mit dem realen Systemverhalten verglichen werden. [8]

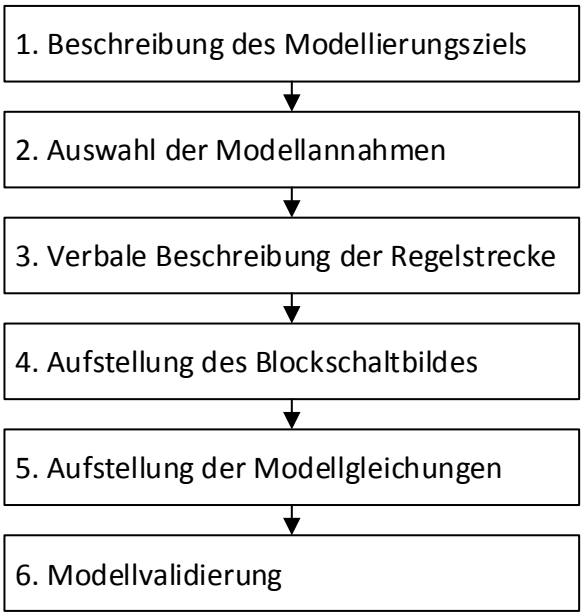


Abbildung 3.6: Vorgehensweise bei der Modellbildung.

3.3 Beschreibung des Modellierungsziels

Ziel der im Folgenden durchgeführten Modellierung der Regelstrecke wird es sein, das Ein- und Ausgangsverhalten der Regelstrecke experimentell zu bestimmen und basierend auf Messungen im Zeit- und Frequenzbereich zunächst nichtparametrische Modelle in Form von Übergangs- und Gewichtsfunktion, Bode-Diagramm, Ortskurve und Pol-Nullstellen-Verteilung aufzustellen. Hieraus soll anschließend ein parametrisches Blackbox-Modell, die Führungsübertragungsfunktion der Regelstrecke $G_S(s)$, ermittelt werden.

Die *Regelstrecke* umfasst dabei im Folgenden den Leistungstreiber für das Peltier-Element, die H-Brücke zur Umpolung des Peltier-Elements, das Peltier-Element selbst sowie alle thermisch angekoppelten Bauteile. Dies sind auf der Oberseite die Spannplatte und auf der Unterseite der Wärmetauscher. Auszugrenzen aus der Definition der Regelstrecke ist insbesondere der auf der Treiberplatine verbaute Mikrocontroller, der die Infrastruktur für die Implementierung des Regelalgorithmus bereitstellt. Weiterhin nicht Bestandteil der Regelstrecke sind die Messglieder in Form der Temperatursensoren auf der Unter- und Oberseite des Peltier-Elements sowie die zugehörigen Messverstärker und Analog-Digital-Wandler. Diese werden separat in der Übertragungsfunktion des Messglieds $G_M(s)$ modelliert.

Entsprechend der obigen Abrenzung der Regelstrecke vom restlichen System werden die Ein- und Ausgangsgrößen wie in Abb. 3.7 gewählt. Die Strecke verfügt somit neben der einwirkenden Störgröße $T_Z(t)$ über zwei weitere Eingangsgrößen und zwei Ausgangsgrößen, ist also ein Mehrgrößensystem. Die erste Eingangsgröße ist der Tastgrad $L(t)$, der das Verhältnis zwischen Impulsdauer und Periodendauer eines pulsweitenmodulierten (PWM) Signals angibt und der Steuerung der im Peltier-Element umgesetzten elektrischen Leistung dient. Das PWM-Signal wird vom Mikrocontroller ausgegeben und dessen Tastgrad kann einen dimensionslosen ganz-

zahligen Wert zwischen 0 und 255 annehmen, wobei bei einem Tastgrad von 0 kein Strom und bei einem Tastgrad von 255 der maximal mögliche Strom durch das Peltier-Element fließt. Die zweite Eingangsgröße $H(t)$ steuert die H-Brücke an und definiert damit die Stromrichtung durch das Peltier-Element. Diese Eingangsgröße ist ein binärer Wert, kann also entweder logisch 1, was einer Eingangsspannung von 5 V entspricht, oder logisch 0, entsprechend einer Eingangsspannung von 0 V, sein. Störungen treten hauptsächlich im thermischen Teil der Regelstrecke auf. Dies können Schwankungen der Lufttemperatur und ein erhöhter Wärmeeintrag bzw. Wärmeaustrag auf der Spannplatte, bedingt durch den Einsatz von Kühlschmierstoffen oder durch beim Zerspanungsprozess entstandene Wärme, sein. Ein eingespanntes Werkstück, das thermisch mit der Spannplatte gekoppelt ist, vergrößert dessen Oberfläche und führt somit ebenfalls zu einem erhöhten Wärmestrom in die kalte Seite des Systems. Alle Störungen auf das thermische System können durch eine an den beiden Ausgängen der Regelstrecke angreifende Temperaturstörung $T_Z(t)$ modelliert werden. Die Ausgangsgrößen der Strecke sind dabei die Temperatur der Spannplatte $T_1(t)$ auf der Oberseite des Peltier-Elements und die Temperatur des Wärmetauschers $T_2(t)$ in der Nähe der Unterseite des Peltier-Elements.

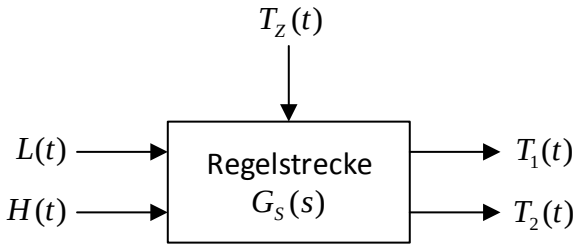


Abbildung 3.7: Relevante Ein- und Ausgangsgrößen der zu modellierenden Strecke.

Eine tabellarische Übersicht aller Ein- und Ausgangsgrößen der Regelstrecke sowie zugehöriger Wertebereiche ist in Tabelle 3.2 gegeben.

Tabelle 3.2: Ein- und Ausgangsgrößen der Regelstrecke und ihre Wertebereiche.

Größe	Wertebereich	Einheit	Bezeichnung
$L(t)$	0…255	-	PWM-Tastgrad zur Sollwertvorgabe für Stromregler
$H(t)$	0 oder 1	-	Umschalten zwischen Heiz- und Kühlbetrieb
$T_Z(t)$	−15…75	°C	Temperaturstörung am Streckenausgang
$T_1(t)$	−15…75	°C	Temperatur der Spannplatte
$T_2(t)$	−15…75	°C	Temperatur des Wärmetauschers

Da es sich bei der Regelstrecke um ein Mehrgrößensystem handelt, genügt eine einzelne Übertragungsfunktion zur vollständigen Beschreibung des Systems nicht aus. Stattdessen muss, wie bereits in Abschnitt 3.2.2 beschrieben, das Übertragungsverhalten jedes Eingangs auf jeden Ausgang durch eine eigene Übertragungsfunktion dargestellt werden. Diese Übertragungsfunktionen werden als Komponenten einer Matrix, der *Übertragungsmatrix* $\boldsymbol{G_S}(s)$, angeordnet. Da bei der Aufstellung der Führungsübertragungsmatrix die Störgröße $T_Z(t) = 0$ gesetzt wird, besitzt die Regelstrecke zwei Eingänge und zwei Ausgänge, sodass sich eine 2×2-Übertragungsmatrix

ergibt. Da die Regelstrecke in zwei grundsätzlich unterschiedlichen Betriebsmodi, nämlich zum einen im Heizbetrieb und zum anderen im Kühlbetrieb, betrieben werden kann, ist es sinnvoll, im weiteren Verlauf der Modellierung die Regelstrecke in Form zweier voneinander unabhängiger Regelstrecken zu behandeln. Heiz- und Kühlbetrieb unterscheiden sich dabei durch die Stromflussrichtung durch das Peltier-Element. Diese wird durch die Eingangsgröße $H(t)$ festgelegt. Im Kühlbetrieb kann die Regelstrecke somit durch die Übertragungsmatrix

$$\boldsymbol{G_{S,K}}(s) = \left.\begin{pmatrix} \frac{T_1(s)}{L(s)} \\ \frac{T_2(s)}{L(s)} \end{pmatrix}\right|_{H(t)=0} \tag{3.4}$$

modelliert werden, während die Strecke im Heizbetrieb durch die Übertragungsmatrix

$$\boldsymbol{G_{S,H}}(s) = \left.\begin{pmatrix} \frac{T_1(s)}{L(s)} \\ \frac{T_2(s)}{L(s)} \end{pmatrix}\right|_{H(t)=1} \tag{3.5}$$

beschrieben werden kann. Die Eingangsgröße $H(t)$ muss dabei nicht mehr berücksichtigt werden, da sie sobald der jeweilige Modus aktiviert ist, keine Auswirkung mehr auf die Regelgröße hat. Folglich kann die Strecke in diesem Fall wie ein System mit einer Eingangsgröße und zwei Ausgangsgrößen behandelt werden, was die Modellierung vereinfacht und die 2×2-Übertragungsmatrizen auf die oben genannten 2×1-Übertragungsvektoren reduziert. Eine weitere Vereinfachung der Modellbildung ergibt sich durch Vernachlässigen der Ausgangsgröße $T_2(t)$. Da für das Spannen eines Werkstückes nur die Temperatur $T_1(t)$ der Spannplatte von Interesse ist, genügt es, lediglich diese Größe zu regeln. Entsprechend ist es nicht erforderlich, das Übertragungsverhalten der Eingangsgröße $L(t)$ auf die Temperatur $T_2(t)$ zu modellieren. Hierdurch ergeben sich nun wieder Eingrößensysteme mit den skalaren Übertragungsfunktionen

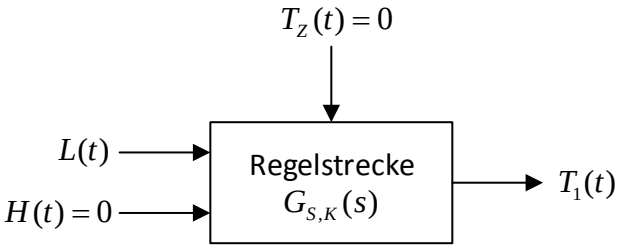
$$G_{S,K}(s) = \left.\frac{T_1(s)}{L(s)}\right|_{H(t)=0} \tag{3.6}$$

für den Kühlbetrieb und

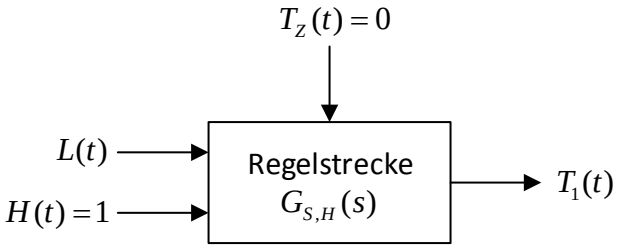
$$G_{S,H}(s) = \left.\frac{T_1(s)}{L(s)}\right|_{H(t)=1} \tag{3.7}$$

für den Heizbetrieb, die im Rahmen der folgenden Modellbildung ermittelt werden sollen. Abb. 3.8a zeigt abschließend das Blockschaltbild mit allen relevanten Ein- und Ausgangsgrößen der vereinfachten Regelstrecke im Kühlbetrieb und Abb. 3.8b ein entsprechendes Blockschaltbild der Strecke im Heizbetrieb.

Bezüglich der erforderlichen Güte des Modells lässt sich festhalten, dass bei der Regelung hauptsächlich der Ausgleich von Störungen im Vordergrund steht. Ein gutes Führungsverhalten mit hoher Dynamik ist zwar wünschenswert, aber nicht notwendigerweise erforderlich. Aufgrunddessen wird die Regelstrecke vorwiegend in einem festen Arbeitspunkt betrieben und nur geringe Schwankungen um diesen Punkt ausführen. Daher ist eine lineare Näherung des Systemverhaltens durch ein lineares Modell zulässig und entsprechend der Aufwand für die Erstellung eines nichtlinearen Modells nicht gerechtfertigt.



(a) Kühlbetrieb.



(b) Heizbetrieb.

Abbildung 3.8: Ein- und Ausgangsgrößen der vereinfachten Regelstrecke im Kühl- und Heizbetrieb.

3.4 Auswahl der Modellannahmen

Da die Modellierung der Regelstrecke in Form einer experimentellen Analyse durchgeführt wird, müssen fast keine Einschränkungen in Bezug auf die Modellannahmen gemacht werden. Alle inneren Effekte, die im Rahmen einer theoretischen Modellierung nur schwer zu erfassen wären, können durch Messung problemlos berücksichtigt und in das Modell einbezogen werden. Dies sind beispielsweise thermische Kontaktwiderstände, räumlich differentielle Temperaturverteilungen und Wärmeströme im Wärmetauscher sowie parasitäre Wärmeströme und Wärmequellen, wie die Leistungselektronik, im System.

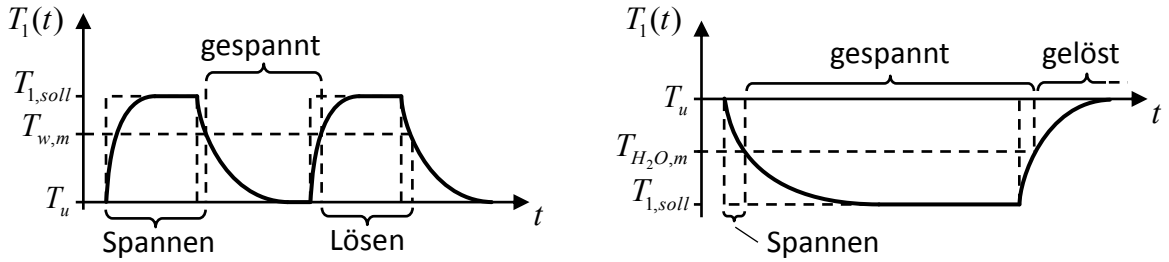
3.5 Verbale Beschreibung der Regelstrecke

Das vorliegende zu regelnde System stellt eine Einspannvorrichtung für die Zerspanung feinmechanischer und mikrotechnischer Werkstücke, meist kleine Quader aus diversen Werkstoffen, dar. Die Kraftübertragung zwischen den Werkstücken und der Einspannvorrichtung erfolgt durch Adhäsionskräfte. Diese werden wahlweise durch einen Film aus gefrorenem Wasser oder aufgeschmolzenem und anschließend erstarrtem Wachs erzeugt. Dementsprechend ergeben sich zwei sehr unterschiedliche Betriebsmodi des Systems, die in Abb. 3.9 schematisch veranschaulicht werden. Dabei zeigen die gestrichelten Linien den Verlauf des Sollwerts der Temperatur der Spannplatte $T_{1,soll}$ und die durchgezogenen Linien die gemessenen Temperaturverläufe von T_1.

1) *Wachsspannbetrieb:* Das Spannen des Werkstücks erfolgt mit Wachs (Abb. 3.9a). Hierzu muss zunächst das Wachs auf die Spannplatte aufgebracht und durch Erhitzen auf eine

Temperatur oberhalb der Schmelztemperatur $T_{w,m}$ verflüssigt werden. Nun kann das Werkstück in das geschmolzene Wachs eingelegt werden und das Peltier-Element abgeschaltet werden. Das Wachs erstarrt und bindet das Werkstück durch Adhäsion an die Spannplatte. Zum Lösen des Werkstücks muss das Wachs erneut aufgeschmolzen und das Werkstück entfernt werden.

2) *Gefrierspannbetrieb:* Das Spannen des Werkstücks erfolgt mittels gefrorenem Wasser (Abb. 3.9b). Hierzu wird flüssiges Wasser auf die Spannplatte aufgebracht, das Bauteil aufgelegt und anschließend durch Wärmeabfuhr mithilfe des Peltier-Elements das Wasser auf eine Temperatur unter den Gefrierpunkt $T_{H_2O,m}$ abgekühlt und gefroren. Hierdurch entsteht eine kraftschlüssige Verbindung zwischen Werkstück und Spannfläche. Im Gegensatz zur Befestigung mittels Wachs muss im gespannten Zustand dauerhaft Energie zugeführt werden, um ein Schmelzen des Eises zu verhindern. Das Lösen des Bauteils erfolgt entsprechend durch Abschalten des Peltier-Elements.



(a) Spannen mit Wachs. **(b)** Spannen mit Eis.

Abbildung 3.9: Verlauf der Temperatursoll- und -istwerte der Spannplatte im Wachs- und im Gefrierspannbetrieb.

Auf die genaue Funktionsweise und Implementierung dieser unterschiedlichen Betriebsmodi und ihre Auswirkung auf die Regelung des Systems wird in Abschnitt 5.4 noch einmal ausführlich eingegangen. Es sei lediglich an dieser Stelle darauf hingewiesen, da bei der experimentellen Modellierung ein gegebenenfalls unterschiedliches Verhalten der Regelstrecke beim Heizen und beim Kühlen berücksichtigt werden muss. So sollten, wie bereits in Abschnitt 3.3 ausführlich erläutert, der Heiz- und Kühlbetrieb durch jeweils voneinander unabhängige Übertragungsfunktionen $G_{S,H}(s)$ und $G_{S,K}(s)$ beschrieben werden. Entsprechend müssen sämtliche Messungen zur Bestimmung des Ein- und Ausgangsverhaltens für den Heiz- und Kühlbetrieb separat durchgeführt werden.

Der generelle Aufbau des Einspannsystems ist in Abb. 3.10 dargestellt. Dabei befindet sich das Peltier-Element im oberen Teil des Systems zwischen dem Wärmetauscher und der aus Aluminium gefertigten Spannplatte, auf der das Werkstück befestigt wird. Der Wärmetauscher besteht aus einzelnen dünnen Kupferblechen, die auf einem Kupfersockel aufgelötet sind. Zur Erhöhung des konvektiv abgeführten Wärmestroms wird Luft mithilfe eines Lüfters horizontal durch die Lamellen transportiert. Eingefasst werden die einzelnen Bestandteile der Spanneinheit durch ein Aluminium-Gehäuse, in dem sich ebenfalls die drei noch zu entwickelnden Platinen mit

dem Leistungstreiber für das Peltier-Element sowie dem Mikrocontroller inklusive der nötigen peripheren Beschaltung und Daten- sowie Energie-Schnittstellen befinden.

In den nachfolgenden Unterkapiteln soll jeweils eine detailliertere verbale Beschreibung des thermischen Systems inklusive des Peltier-Elements und der Treiberschaltung erfolgen.

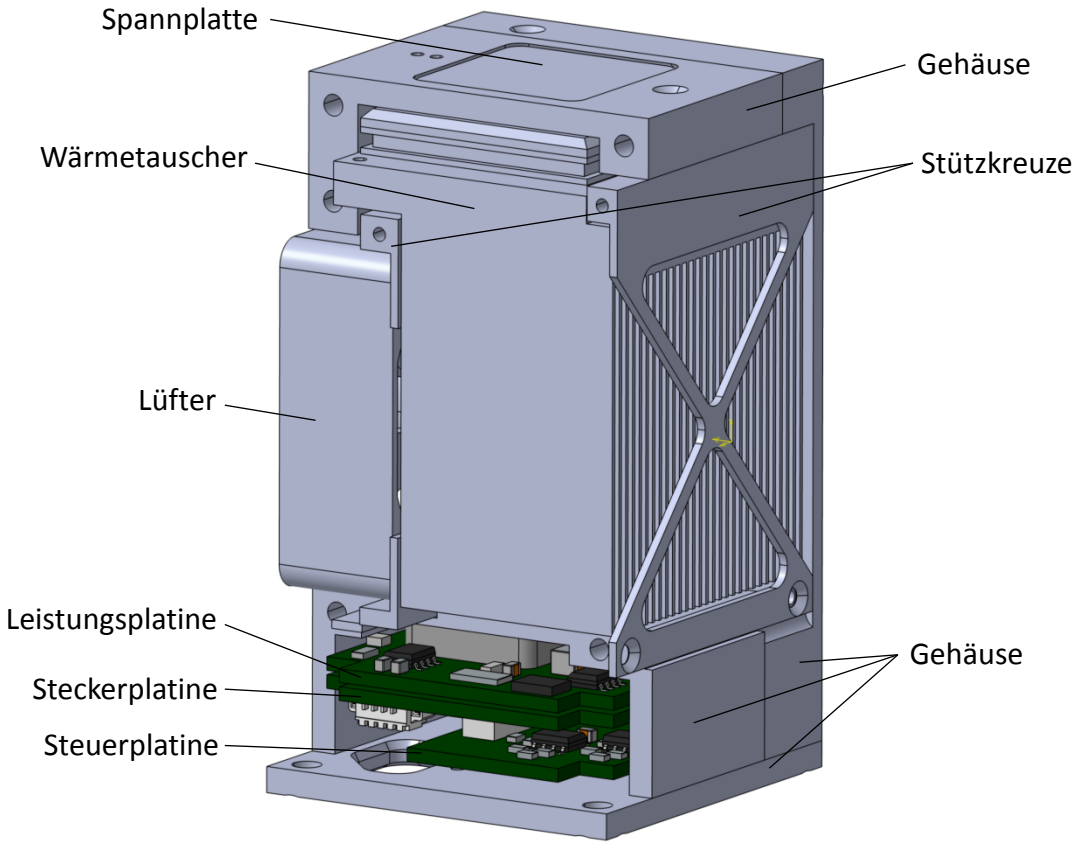


Abbildung 3.10: CAD-Modell des Gesamtsystems.

3.5.1 Thermisches System

Der folgende Abschnitt soll weit über die verbale Funktionsbeschreibung des thermischen Systems hinausgehen. Da das Peltier-Element als aktives Element des thermischen Systems ein zentrales Bauteil der Regelstrecke ist und maßgeblich ihre Eigenschaften mitbestimmt, wird an dieser Stelle eine vollständige theoretische Modellbildung inklusive Simulation und Diskussion der Simulationsergebnisse durchgeführt. Dabei wird zunächst lediglich auf den Gefrierbetrieb eingegangen und anschließend der Heizbetrieb betrachtet. Es sei angemerkt, dass diese Kenntnisse über den inneren Aufbau und die innere Funktionsweise der Strecke nicht notwendigerweise erforderlich sind, da im Rahmen der später durchgeführten experimentellen Modellierung lediglich das Ein- und Ausgangsverhalten der Strecke interessiert. Dennoch hilft ein umfassendes Wissen über die genaue Funktionsweise der Regelstrecke bei der späteren Validierung des Modells.

Wie in Abb. 3.11 dargestellt, umfasst das thermische System die Spannplatte mit Masse m_1 und Wärmekapazität c_1, das Peltier-Element als aktiven Teil des Systems und den Wärmetauscher mit Masse m_2 und Wärmekapazität c_2. Nicht dargestellt ist der Lüfter, der jedoch ebenfalls zur thermischen Strecke gehört, da er den Wärmeübergangskoeffizienten α_2 des Wärmetauschers maßgeblich beeinflusst. Wärme fließt im Kühlbetrieb von der Spannplatte durch das Peltier-Element in den Wärmetauscher und wird dort an die Umgebungsluft abgegeben. Die Temperatur der Spannplatte sinkt infolgedessen unterhalb des Gefrierpunktes von Wasser. Beim Heizbetrieb fließt ein Wärmestrom in genau umgekehrte Richtung vom Wärmetauscher in die Spannplatte und heizt diese entsprechend auf. Die Verhältnisse sind in Abb. 3.12 für den Kühlbetrieb dargestellt und werden später detailliert erläutert und modelliert.

Eine wichtige Eigenschaft des thermischen Systems sind seine Ein- und Ausgangsgrößen. Am Eingang liegt der konstante elektrische Strom I an, der infolge der transportierten Wärmeströme und gespeicherten inneren Energien zu den Temperaturen T_1 und T_2 der Ober- und Unterseite des Peltier-Elements führt. Diese werden näherungsweise durch Platin-Dünnschichtsensoren vom Typ Heraeus SMD 0603 PT1000, die mithilfe von Wärmeleitkleber an die Spannplatte und den Wärmetauscher geklebt sind, erfasst. Hierbei kommt es aufgrund der Kontaktwiderstände zu Messfehlern, die jedoch, wie in Abschnitt 3.4 bereits erläutert, durch die Messung mit berücksichtigt werden.

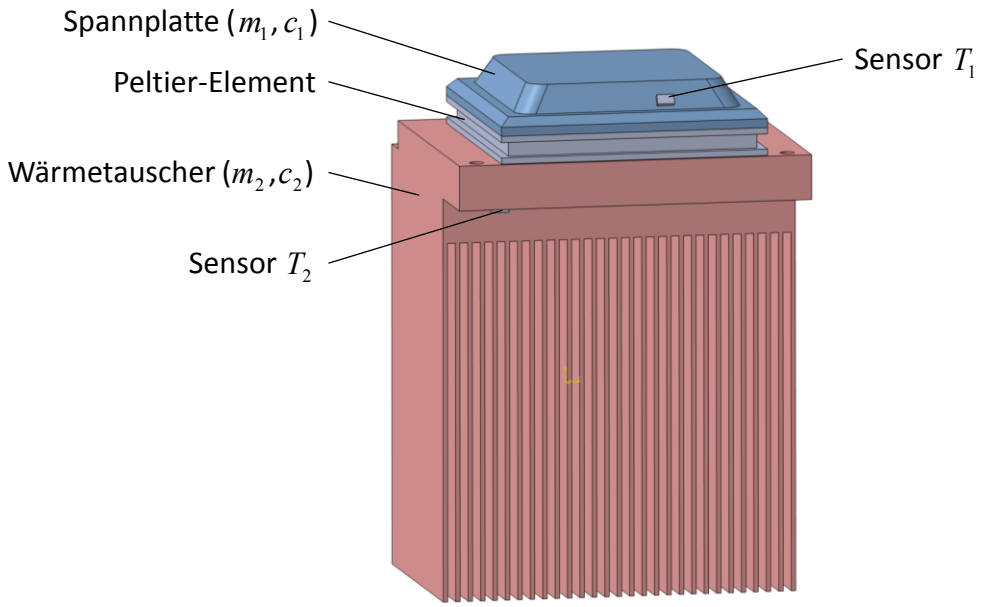


Abbildung 3.11: CAD-Modell der relevanten Komponenten des thermischen Systems.

3.5.1.1 Funktionsweise des Peltier-Elements

Ein wichtiger Bestandteil der thermischen Strecke ist das Peltier-Element, auch TEC (thermoelectric cooler) genannt. Abb. 3.12 zeigt schematisch den Aufbau eines solchen Elements. Zwischen zwei keramischen Platten sind Paare n- und p-dotierter Pellets aus Bismut-Tellurid

(Bi_2Te_3) elektrisch in Reihe und thermisch parallel verschaltet. Fließt ein konstanter elektrischer Strom durch das Peltier-Element, entsteht eine Temperaturdifferenz zwischen den beiden Seiten des TECs. Begründen lässt sich dies durch die unterschiedlichen Energieniveaus der Leitungsbänder in den n- und p-dotierten Pellets. Fließen Elektronen vom n- in den p-dotierten Halbleiter, nehmen sie thermische Energie auf, um in das energetisch höhere Leitungsband des p-Halbleiters zu gelangen. Entsprechend geben die Elektronen beim Übergang von einem p- auf einen n-Halbleiter wieder thermische Energie ab.

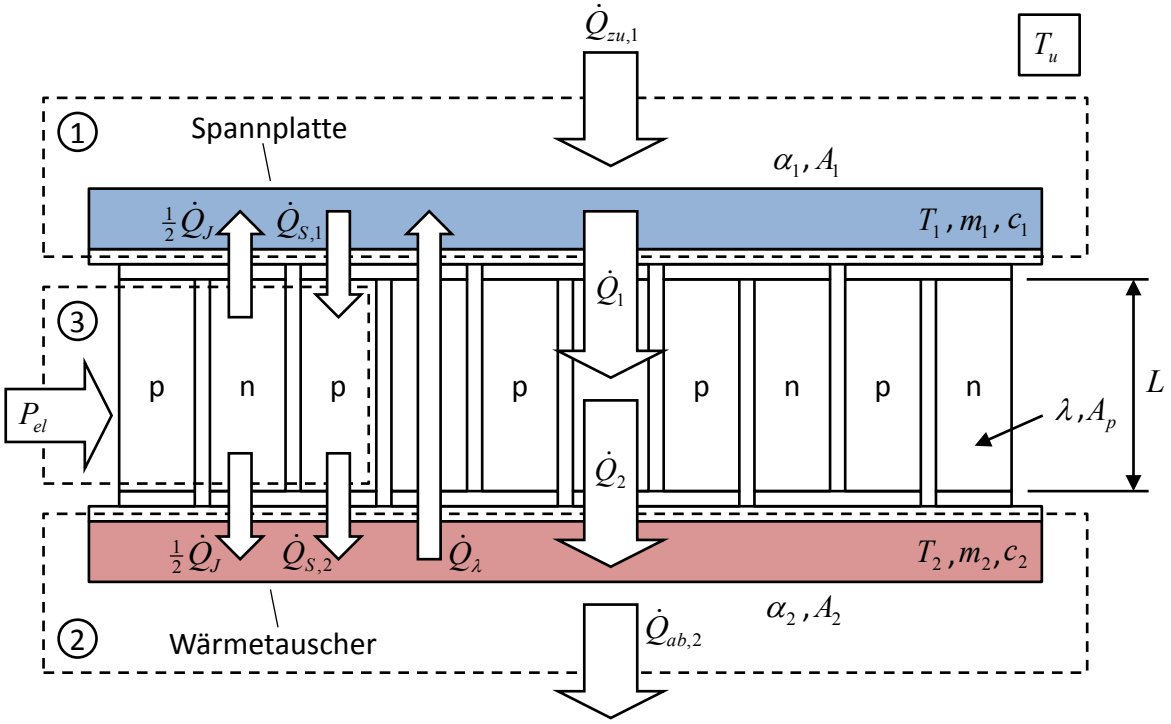


Abbildung 3.12: Schematischer Aufbau, Modellparameter und Energieströme des Peltier-Elements.

3.5.1.2 Theoretische Modellierung des thermischen Systems

Um ein Verständnis für das dynamische Verhalten und die genaue Funktionsweise des Peltier-Elements zu erhalten, soll im Folgenden ein Zustandsraummodell des thermischen Systems aufgestellt werden. Dabei wird zunächst der Kühlbetrieb, bei dem die Spannplatte mit der kalten Seite und der Wärmetauscher mit der heißen Seite des Peltier-Elements verbunden ist, betrachtet. Es gilt also $T_1 < T_u < T_2$. Die Eingangsgröße des Systems ist dabei der aufgeprägte konstante elektrische Strom I, die Zustandsgrößen, die hier gleichzeitig den Ausgangsgrößen entsprechen, sind die Temperaturen der kalten Seite T_1 sowie der heißen Seite T_2. Das Zustandsraummodell ergibt sich durch Bilanzierung der in Abb. 3.12 eingezeichneten Wärmeströme. Das Modell beinhaltet dabei sowohl die inneren Wärmeströme des Peltier-Elements, die später zu resultierenden Wärmeströmen zusammengefasst werden, als auch die mit der Umgebungsluft konvektiv ausgetauschten Wärmeströme $\dot{Q}_{zu,1}$ und $\dot{Q}_{ab,2}$. Es beschreibt neben dem Peltier-

Element die Spannplatte auf der kalten Seite und den Wärmetauscher auf der heißen Seite des Peltier-Elements durch die Massen m_1 und m_2 mit der jeweiligen Wärmekapazität c_1 und c_2.

Für den auf der kalten Seite aufgenommenen thermoelektrischen Wärmestrom gilt

$$\dot{Q}_{S,1} = N\alpha I T_1. \tag{3.8}$$

Dabei stellen α den Seebeck-Koeffizienten von Bismut-Tellurid und N die Anzahl thermoelektrischer Paare, das heißt Paare aus jeweils einem n- und einem p-dotierten Pellet dar. Entsprechend gilt für den thermoelektrischen Wärmestrom, der in die heiße Seite übergeht

$$\dot{Q}_{S,2} = N\alpha I T_2. \tag{3.9}$$

Aufgrund des elektrischen Widerstandes R aller in Reihe geschalteter Halbleiter-Pellets kommt es zu Energiedissipation, die durch die joulsche Wärme

$$\dot{Q}_J = RI^2 \tag{3.10}$$

beschrieben werden kann. Diese fließt zu jeweils gleichen Anteilen in die kalte und heiße Seite des Peltier-Elements. Zusätzlich zu diesen Wärmeströmen muss berücksichtigt werden, dass Wärme durch Wärmeleitung von der heißen zur kalten Seite zurückfließt. Dieser Wärmestrom kann mit der Hilfsgröße

$$\Theta = \frac{\lambda A_p}{L}, \tag{3.11}$$

die den Kehrwert des thermischen Widerstandes eines thermoelektrischen Paares in der Einheit W/K darstellt, als

$$\dot{Q}_\lambda = N\Theta(T_2 - T_1) \tag{3.12}$$

angegeben werden. Die Größe A_p stellt dabei die Querschnittsfläche und L die Länge eines Pellets dar. Die Wärmeleitfähigkeit von Bismut-Tellurid wird mit λ bezeichnet.

Betrachtet man diese vier Effekte, die innerhalb des Peltier-Elements ablaufen, so können die resultierenden Wärmeströme $\dot{Q}_1$, $\dot{Q}_2$ sowie die aufgenommene elektrische Energie P_{el} berechnet werden. Letztere ergibt sich durch Bilanzierung im Bilanzierungsraum 3 zu

$$P_{el} = \dot{Q}_J + \dot{Q}_{S,2} - \dot{Q}_{S,1} = RI^2 + N\alpha I(T_2 - T_1). \tag{3.13}$$

Berücksichtigt man alle in die kalte Seite des Peltier-Elements hinein- und hinausfließenden Wärmeströme, ergibt sich der resultierende auf der kalten Seite vom Peltier-Element aufgenommene Wärmestrom

$$\dot{Q}_1 = \dot{Q}_{S,1} - \dot{Q}_\lambda - \frac{1}{2}Q_J = N\alpha I T_1 - \frac{1}{2}RI^2 - N\Theta(T_2 - T_1). \tag{3.14}$$

Entsprechend kann man den auf der heißen Seite vom Peltier-Element in den Luftkühler fließenden Wärmestrom

$$\dot{Q}_2 = \dot{Q}_{S,2} - \dot{Q}_\lambda + \frac{1}{2}Q_J = \dot{Q}_1 + P_{el} = N\alpha I T_2 + \frac{1}{2}RI^2 - N\Theta(T_2 - T_1) \tag{3.15}$$

berechnen. [9–11]

Vernachlässigt man Wärmestrahlung, kann der von der Umgebung auf die kalte Seite des Peltier-Elements übergehende Wärmestrom als konvektiv übertragener Wärmestrom

$$\dot{Q}_{zu,1} = \alpha_1 A_1 (T_u - T_1) \tag{3.16}$$

angegeben werden. Entsprechend gilt für den von der heißen Seite konvektiv an die Luft abgegebenen Wärmestrom

$$\dot{Q}_{ab,2} = \alpha_2 A_2 (T_2 - T_u). \tag{3.17}$$

Auf Basis der Kenntnis aller in die kalte Seite hinein- und hinausfließenden Wärmeströme kann nun durch Bilanzierung im Bilanzierungsraum 1 die Änderung der inneren Energie der thermischen Masse m_1 der kalten Seite als

$$\frac{dU_1}{dt} = m_1 c_1 \frac{dT_1}{dt} = \dot{Q}_{zu,1} - \dot{Q}_1 = \alpha_1 A_1 (T_u - T_1) - N\alpha I T_1 + \frac{1}{2} R I^2 + N\Theta (T_2 - T_1) \tag{3.18}$$

beschrieben werden. Analog folgt durch Bilanzierung im Bilanzraum 2 die Änderung der inneren Energie der an der heißen Seite des Peltier-Elements angekoppelten Bauteile

$$\frac{dU_2}{dt} = m_2 c_2 \frac{dT_2}{dt} = \dot{Q}_2 - \dot{Q}_{ab,2} = -\alpha_2 A_2 (T_2 - T_u) + N\alpha I T_2 + \frac{1}{2} R I^2 - N\Theta (T_2 - T_1). \tag{3.19}$$

Durch Umstellen nach den ersten Ableitungen der Temperaturen T_1 und T_2 der kalten und der heißen Seite erhält man folgendes Differentialgleichungssystem erster Ordnung

$$\frac{dT_1}{dt} = \frac{1}{m_1 c_1} \left(\alpha_1 A_1 (T_u - T_1) - N\alpha I T_1 + \frac{1}{2} R I^2 + N\Theta (T_2 - T_1) \right) \tag{3.20}$$

$$\frac{dT_2}{dt} = \frac{1}{m_2 c_2} \left(\alpha_2 A_2 (T_u - T_2) + N\alpha I T_2 + \frac{1}{2} R I^2 - N\Theta (T_2 - T_1) \right). \tag{3.21}$$

Dies entspricht dem in Abschnitt 2.7.1 eingeführten nichtlinearen Zustandsraummodell. Nichtlinear deshalb, da die rechten Seiten der beiden Differentialgleichungen das Quadrat der Eingangsgröße, also des elektrischen Stroms I, enthalten. Sollte das Störungsverhalten des thermischen Systems, das heißt die Reaktion auf Temperaturschwankungen der heißen oder kalten Seite des Peltier-Elements, untersucht werden, so ließe sich eine Linearisierung entsprechend Abschnitt 2.7.2 durchführen, da hier die Abweichungen der Temperaturen von den Werten im Arbeitspunkt gering ausfallen. Es soll jedoch stattdessen die Sprungantwort (siehe Abschnitt 2.5.1) des Peltier-Elements bei einem aufgeprägten Stromsprung untersucht werden. Da hier die Temperaturänderungen sehr groß sind, sind die Angabe eines Arbeitspunktes und damit eine Linearisierung nicht möglich.

Eine Betrachtung des Heizbetriebs ist mit diesem Modell ebenfalls problemlos möglich. Dabei ist die Temperatur der Spannplatte größer als die des Wärmetauschers, es gilt also $T_1 > T_u > T_2$. Folglich kehren sich die Wärmeströme $\dot{Q}_{zu,1}$, $\dot{Q}_{ab,2}$, $\dot{Q}_{S,1}$, $\dot{Q}_{S,2}$, $\dot{Q}_\lambda$ und damit auch $\dot{Q}_1$ und $\dot{Q}_2$ um. Es ergibt sich ein Differentialgleichungssystem, das Gleichung (3.20) und Gleichung (3.21) entspricht, wenn man für den Strom I negative Werte einsetzt. Dies ist gleichbedeutend mit einer Umpolung der Stromrichtung.

3.5.1.3 Implementierung und Simulation des Modells

Das nichtlineare Gleichungssystem bestehend aus Gleichung (3.20) und Gleichung (3.21) kann in MATLAB numerisch gelöst werden. Hierzu werden zunächst die Modellparameter entsprechend Tabelle 3.3 basierend auf Messungen am realen System und Angaben aus dem Datenblatt des Peltier-Elements [12–14] gewählt und als Variablen im Workspace angelegt. Anschließend wird das Zustandsraummodell, wie in Listing A.1 gezeigt, als MATLAB-Funktion implementiert und kann mit Listing A.2 numerisch ausgewertet werden.

Tabelle 3.3: Angenommene Modellparameter des thermischen Systems für die Simulation.

Größe	Wert	Einheit	Bezeichnung
N	127	-	Anzahl der Thermopaare
A_1	$0{,}484 \cdot 10^{-3}$	m^2	Oberfläche der Spannplatte
A_2	0,071	m^2	Oberfläche des Wärmetauschers
m_1	0,019	kg	Masse der Spannplatte
m_2	0,256	kg	Masse des Wärmetauschers
c_1	900	$\mathrm{J/(kg\,K)}$	Wärmekapazität der Spannplatte (Aluminium)
c_2	380	$\mathrm{J/(kg\,K)}$	Wärmekapazität des Wärmetauschers (Kupfer)
α_1	1	$\mathrm{W/(m^2\,K)}$	Wärmeübergangskoeffizient zwischen Spannplatte und Luft
α_2	24	$\mathrm{W/(m^2\,K)}$	Wärmeübergangskoeffizient zwischen Wärmetauscher und Luft
R	1,85	Ω	elektrischer Widerstand des gesamten Peltier-Elements
α	$0{,}210 \cdot 10^{-3}$	V/K	Seebeck-Koeffizient von Bi_2Te_3
λ	1,5	$\mathrm{W/(m\,K)}$	Wärmeleitfähigkeit von Bi_2Te_3
$\frac{L}{A_p}$	720	1/m	Geometriefaktor der Pellets
T_u	293	K	Lufttemperatur

Die so erhaltenen Temperaturverläufe für die Temperaturen T_1 der kalten Seite und T_2 der heißen Seite sind in Abb. 3.13 für verschiedene aufgeprägte Ströme I dargestellt. Dabei betrugen die Anfangswerte beider Temperaturen 20 °C.

3.5.1.4 Diskussion der Ergebnisse

Werden die Temperaturverläufe nach obigem Modell mit denen der gemessenen Verläufe aus Abb. 3.19 verglichen, lässt sich eine gute Übereinstimmung der Temperaturverläufe $T_2(t)$ der heißen Seite mit den Messungen feststellen. Die Temperaturverläufe $T_1(t)$ der kalten Seite zeigen größere Abweichungen und teilweise stimmen gemessene und berechnete Verläufe auch qualitativ nicht überein. So verläuft T_1 bei $I = 5\,\mathrm{A}$ in der Realität stets unterhalb von T_1 bei $I = 1\,\mathrm{A}$. Im Modell hingegen liegt der stationäre Wert von T_1 bei 1 A unter demjenigen bei 5 A. Weiterhin liegen die modellierten Temperaturverläufe für T_1 alle um etwa 5 °C zu niedrig. Liegt die niedrigste erreichbare Temperatur in der Realität bei −8 °C, sind es im Modell −15 °C.

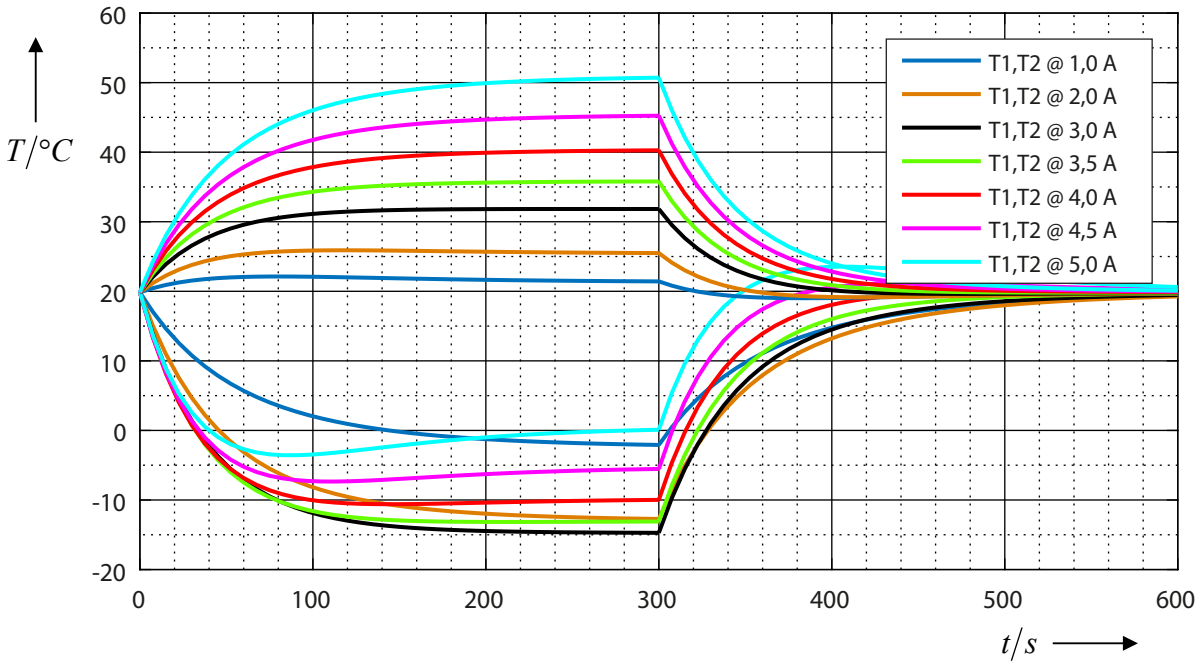


Abbildung 3.13: Modellierte Sprungantworten des thermischen Systems für verschiedene aufgeprägte konstante Ströme im Kühlbetrieb ($T_u = 20\,°\mathrm{C}$).

Weiterhin zeigt das Modell Abweichungen hinsichtlich des optimalen Stroms. Die Messergebnisse aus Abb. 3.19 machen deutlich, dass die minimale Temperatur der kalten Seite bei einem Strom von 3,5 A erreicht wird. Im Modell wird bei 3 A die niedrigste Temperatur erreicht. Ursache für diese Abweichungen zwischen Modell und Realität sind die großen Unsicherheiten in den Modellparametern. Die Massen, Oberflächen, Wärmekapazitäten und alle weiteren Stoffkonstanten sind zwar relativ genau, die genauen Werte der Wärmeübergangskoeffizienten zwischen Luft und Spannplatte bzw. Wärmetauscher sind jedoch unbekannt. Es können lediglich sinnvolle Annahmen für diese Werte basierend auf Angaben in der Literatur getroffen werden. Zudem ist mangels Zugänglichkeit des Peltier-Elements nicht sichergestellt, dass der Geometriefaktor Θ korrekt ist. Sein Wert wurde aus [12] entnommen. Weitere Abweichungen kommen dadurch zustande, dass im Modell die tatsächlichen Temperaturen T_1 und T_2 berechnet werden, die Messung jedoch die Messwerte $T_{1,M}$ und $T_{2,M}$ dieser Temperaturen erfasst. Das Modell berücksichtigt also nicht die Messübertragungsfunktion.

Trotz der genannten Abweichungen zeigt das Modell global eine gute Übereinstimmung mit den Messergebnissen. So entsprechen beispielsweise die Zeitkonstanten im Modell in etwa denjenigen des realen Systems. Weiterhin weist das Modell global proportional zeitverzögertes Verhalten auf, sodass sich nach einem aperiodischen Einschwingvorgang stationäre Temperaturwerte einstellen. Das Allpass-Verhalten des realen Systems wird durch das Modell ebenfalls abgebildet. So durchlaufen die Temperaturen T_1 bei Strömen über 4 A zunächst ein Minimum, um dann wieder leicht anzusteigen. Dies liegt daran, dass bei steigender Temperatur T_2 der heißen Seite der in die kalte Seite zurückfließende Wärmestrom $\dot{Q}_\lambda$ ebenfalls ansteigt.

Abschließend lässt sich also festhalten, dass mit Gleichung (3.20) und Gleichung (3.21) ein

Zustandsraummodell des thermischen Systems aufgestellt werden konnte, das das Verhalten des realen Systems gut annähert. Da das Modell wie eingangs erwähnt lediglich das physikalische Grundverständnis für das System fördern sollte und nicht für den Reglerentwurf benötigt wird, spielen die oben genannten Ungenauigkeiten des Modells im weiteren Verlauf der Arbeit keine Rolle.

3.5.2 Treiberschaltung

Wie bereits in Abb. 3.10 dargestellt, verfügt das Einspannsystem über eine elektrische Schaltung, deren Aufgaben die Ansteuerung des Peltier-Elements, die Erfassung der Temperaturen von Spannplatte und Wärmetauscher und die Realisierung der Regelung sind. Zum jetzigen Zeitpunkt liegt die finale Treiberschaltung noch nicht vor, weshalb mit einer vereinfachten Steuerschaltung gearbeitet wird. Eine stark abstrahierte Darstellung dieser Schaltung ist in Abb. 3.15 zu sehen. Eine ähnliche Darstellung der momentan in Entwicklung befindlichen erweiterten Treiberschaltung ist in Anhang A.2 zu finden.

Die vereinfachte Treiberschaltung, die im Folgenden zur Ansteuerung des Peltier-Elements dienen soll, besteht aus eingekauften und bereits fertig aufgebauten Modulen. Zum einen ist dies ein Arduino Pro Mini, der einen Mikrocontroller vom Typ *AtMega 328P* beherbergt, und zum anderen das Motortreiber-Board *Pololu G2 High-Power Motor Driver 18v17*. Letzteres beinhaltet die H-Brücke zur Umpolung des Peltier-Elements und ist in Abb. 3.14 dargestellt. Das Board kann mit Eingangsspannungen von 6,5 V bis 30 V betrieben werden und dabei einen maximalen konstanten Ausgangsstrom von 7 A schalten, was für die vorliegende Anwendung ausreicht. Um Spannungsspitzen in der Versorgungsspannung abzufangen und eine Rückkopplung des PWM-Signals in die Spannungsquelle zu verhindern, wird an die mit + und − gekennzeichneten Lötpads ein Elektrolyt-Kondensator mit einer Kapazität von 470 µF angelötet.

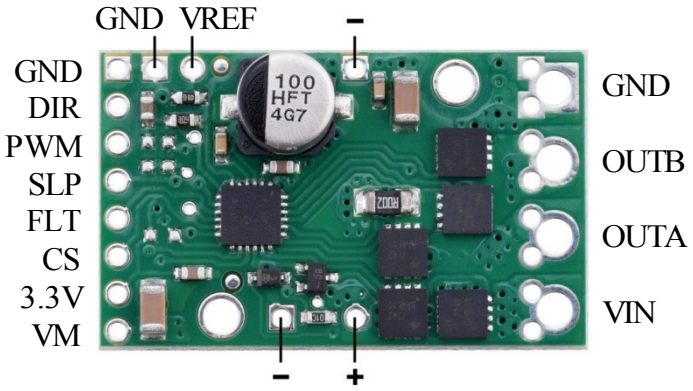


Abbildung 3.14: Verwendetes Treibermodul zur Ansteuerung des Peltier-Elements (Quelle: pololu.com).

An die Ausgänge *OUTA* und *OUTB* des Treibermoduls wird das Peltier-Element angeschlossen und zwischen den Eingängen *VIN* und *GND* die konstante Versorgungsspannung des Systems angelegt. Wie sich im späteren Verlauf der Arbeit noch herausstellen wird, liegt die optimale Eingangsspannung bei 11,3 V, da in diesem Fall bei einer Spannplattentemperatur von −7 °C

ein Strom von 3,5 A durch das Peltier-Element fließt. Bei diesem Strom wird die niedrigste Spannplattentemperatur erreicht und es kommt nicht zur Überhitzung des Wärmetauschers. Weiterhin verfügt das Treibermodul über den Eingang *DIR*, mit dem die Richtung des Stromflusses durch das Peltier-Element festgelegt werden kann. Wird an *DIR* auf logisch 1 gesetzt, fließt der Strom von *OUTA* zu *OUTB*. Ist *DIR* logisch 0, wird die Stromrichtung entsprechend umgekehrt. Ebenfalls sehr wichtig ist der Eingangspin *PWM*. An diesen wird das PWM-Signal angelegt, das die gesamte H-Brücke aktiviert oder deaktiviert und damit eine Pulsweitenmodulation des Stroms durch das Peltier-Element ermöglicht. Auf diese Weise kann die aufgenommene elektrische Leistung und damit die Kühlleistung des Peltier-Element eingestellt werden. Die maximal mögliche Schaltfrequenz, die das Board unterstützt, beträgt dabei 100 kHz. Tabelle 3.4 zeigt noch einmal in übersichtlicher Form mögliche Ein- und Ausgangszustände des Treibermoduls und den resultierenden Laststrom durch das Peltier-Element. Die Ströme I_K und I_H sind dabei betragsmäßig gleich, unterscheiden sich jedoch im Vorzeichen.

Tabelle 3.4: Ein- und Ausgangszustände des Treibermoduls nach [15].

PWM	DIR	$OUTA$	$OUTB$	Laststrom I
1	1	1	0	I_H
1	0	0	1	I_K
0	1/0	0	0	0

Um das Treibermodul zu aktivieren, muss der Eingang *SLP* auf logisch 1 gesetzt werden. Wird dieser Eingang auf logisch 0 gesetzt, wird das Modul deaktiviert und in einen Energiesparmodus versetzt. Das Board verfügt darüber hinaus über weitere Ein- und Ausgänge, die jedoch für die vorliegende Anwendung nicht von Interesse sind und daher an dieser Stelle nicht weiter erläutert werden sollen. [15]

Wie Abb. 3.15 zu entnehmen ist, enthält das Treibermodul einen eigenen Mikrocontroller, der die Ansteuerung der auf dem Board verbauten H-Brücke übernimmt. Diese besteht aus je zwei N-Kanal- und P-Kanal-FETs, die je nach Zustand der zuvor beschriebenen Eingangsgrößen leitend geschaltet oder gesperrt werden. Hierdurch wird die Umpolung der Stromrichtung durch das an die H-Brücke angeschlossene Peltier-Element ermöglicht.

Neben dem Treibermodul ist der AtMega 328P zentrales Element der Steuerschaltung. Auf ihm werden später der Regelalgorithmus sowie die Steuerlogik des gesamten Spannsystems ausgeführt. Zudem enthält der Mikrocontroller die Schnittstelle zur Kommunikation mit einem externen PC. An den nativen UART (Universal Asynchronous Receiver Transmitter) des Mikrocontrollers ist hierzu eine Wandlerplatine mit einem *FTDI FT232R*-Konverter-IC angeschlossen, das zur Kommunikation mit der USB-Schnittstelle des PCs dient. Über den PC kann dabei einerseits die Programmierung des Mikrocontrollers erfolgen und andererseits das Spannsystem mit der in Kapitel 5 entwickelten graphischen Oberfläche gesteuert und überwacht werden. Die Energieversorgung des Mikrocontrollers erfolgt über einen Festspannungsregler, der aus den zur Verfügung stehenden 20 V Gleichspannung eine Spannung von 5 V erzeugt. An den Mikrocontroller angeschlossen sind darüber hinaus zwei INA155-Instrumentenverstärker, die den Messbereich der beiden Temperatursensoren zur Erfassung der Spannplattentemperatur T_1 und

Wärmetauscher-Temperatur T_2 auf eine analoge Spannung von 0 V bis 5 V abbilden, was dem Eingangsspannungsbereich des Analog-Digital-Wandlers des Mikrocontrollers entspricht. Dies ermöglicht eine Digitalisierung des Signals im Mikrocontroller mit höchstmöglicher Auflösung von 10 bit. Das heißt der Messbereich der Sensoren, der Temperaturen von −15 °C bis 75 °C umfasst, wird auf diskrete Werte von 0 bis 1023 abgebildet. Ohne Instrumentenverstärker würde nicht der gesamte Eingangsspannungsbereich des Analog-Digital-Wandlers ausgenutzt, was in einer entsprechend niedrigeren Auflösung des digitalisierten Signals resultierte.

Abschließend sei angemerkt, dass die hier verwendete Schaltung keine Filterung des Laststroms durch das Peltier-Element vornimmt. Das heißt, dass pulsweitenmodulierte Signal wird auf den Laststrom übertragen, sodass das Peltier-Element mit einem gepulsten Strom betrieben wird. Nach [16] wirkt sich dies negativ auf die maximal erreichbare Temperaturdifferenz ΔT zwischen kalter und heißer Seite des Peltier-Elements aus. Diese berechnet sich mit der maximalen Temperaturdifferenz $\Delta T_{\max}$ bei Betrieb mit Gleichstrom und dem Ripple-Koeffizienten K zu

$$\frac{\Delta T}{\Delta T_{\max}} = \frac{1}{1 + K^2}. \tag{3.22}$$

Für den Ripple-Koeffizienten gilt dabei

$$K = \frac{I_{\text{Impuls}}}{I_{\text{DC}}} \cdot \frac{T_{\text{Impuls}}}{T} \tag{3.23}$$

mit der Amplitude I_{Impuls} und Dauer T_{Impuls} eines Strompulses sowie dem Betrag I_{DC} des Gleichstroms und der Periodendauer T des PWM-Signals. Bei einem Duty-Cycle des PWM-Signals von 50%, das heißt $T_{\text{Impuls}} = 0{,}5T$, ergibt sich mit $I_{\text{Impuls}} = I_{\text{DC}}$ der Ripple-Koeffizient $K = 0{,}5$ und damit eine erreichbare Temperaturdifferenz $\Delta T = 0{,}8\Delta T_{\max}$. Im schlechtesten Fall, das heißt für $K \to 1$, reduziert sich die erreichbare Temperaturdifferenz auf die Hälfte der maximal möglichen Temperaturdifferenz. Ist die Frequenz des PWM-Signals größer als 1 kHz, so wird die Lebensdauer des Peltier-Elements durch den Betrieb mit einem gepulsten Strom laut [16] nicht negativ beeinträchtigt.

Die in Anhang A.2 schematisch dargestellte und momentan in der Entwicklung befindliche Treiberschaltung verfügt über ein LC-Filter, das zur Glättung des gepulsten Stroms dient. Weiterhin besitzt diese Schaltung den Vorteil, dass der Strom durch das Peltier-Element unabhängig von der Eingangsspannung konstant gehalten wird. Hierzu wird der akutelle Laststrom über Messwiderstände erfasst und die FETs der H-Brücke mithilfe von Stromregler-ICs angesteuert. Die Stromregler erhalten dabei einen Sollwert für den Strom vom Mikrocontroller, der je nach Regler-IC entweder durch ein PWM-Signal oder eine analoge Spannung repräsentiert wird. Entsprechend zeichnet sich diese Treiberschaltung durch eine höhere Robustheit gegenüber Schwankungen der Versorgungsspannung sowie eine höhere Effizienz des Peltier-Elements aufgrund des Betriebs mit einem geglätteten Strom aus.

3.6 Aufstellung des Blockschaltbildes

Nachdem die Funktionsweise der Regelstrecke im vorherigen Abschnitt ausführlich erläutert wurde, kann nun ein detailliertes Blockschaltbild der inneren Struktur der Regelstrecke mit

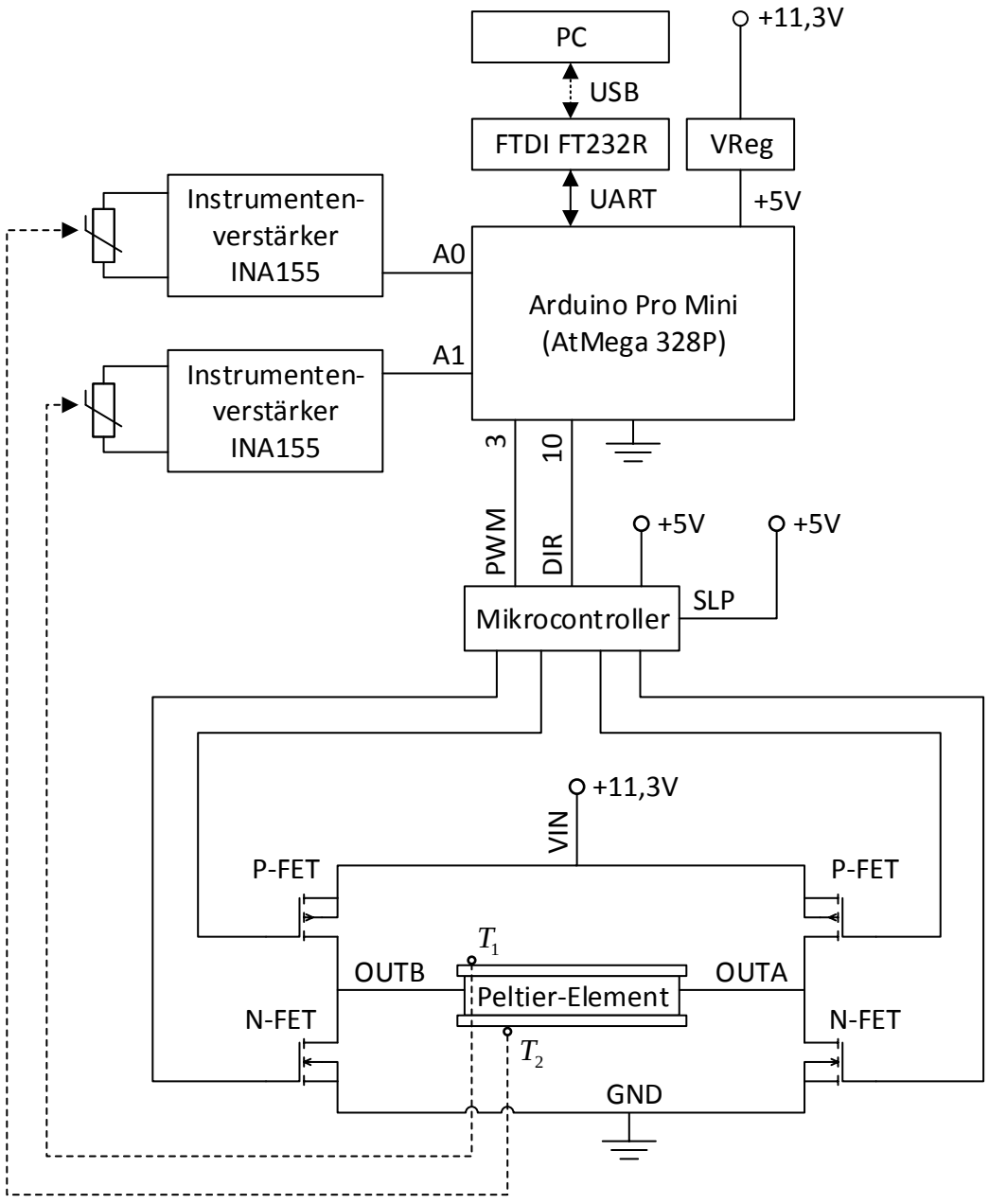


Abbildung 3.15: Schematische Darstellung der Steuerelektronik bestehend aus H-Brücke, Messeinrichtungen und Mikrocontroller.

allen relevanten Ein- und Ausgangsgrößen aufgestellt werden. Basis hierfür sind die in Abb. 3.8 dargestellten Blockschaltbilder, die das Übertragungsverhalten der gesamten Regelstrecke im Heiz- und Kühlbetrieb beschreiben.

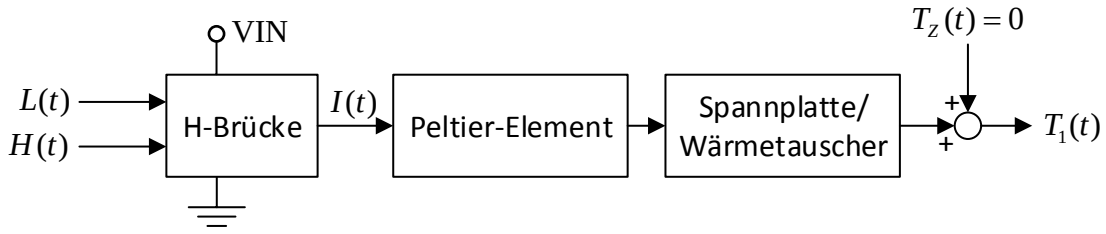


Abbildung 3.16: Blockschaltbild der inneren Struktur der Regelstrecke.

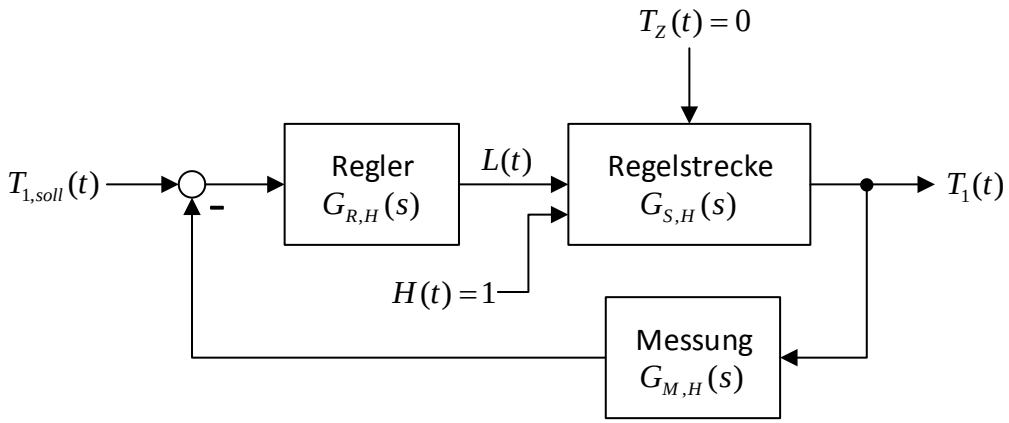
Im Heizbetrieb gilt $H(t) = 1$, im Kühlbetrieb $H(t) = 0$, wobei $H(t)$ der am Eingangspin *DIR* des Treibermoduls anliegenden Größe entspricht. Die Größe $L(t)$ stellt den Tastgrad des PWM-Signals am Eingang *PWM* des Treibers dar. Störungen werden nicht modelliert ($T_Z(t) = 0$), da lediglich das Führungsübertragungsverhalten der Regelstrecke interessiert.

Mithilfe der Blockschaltbilder aus Abb. 3.8 können zudem Blockschaltbilder für den Gesamtregelkreis aufgestellt werden. Das Blockschaltbild für den Regelkreis im Heizbetrieb ist in Abb. 3.17a dargestellt und dasjenige für die Regelung im Kühlbetrieb in Abb. 3.17b.

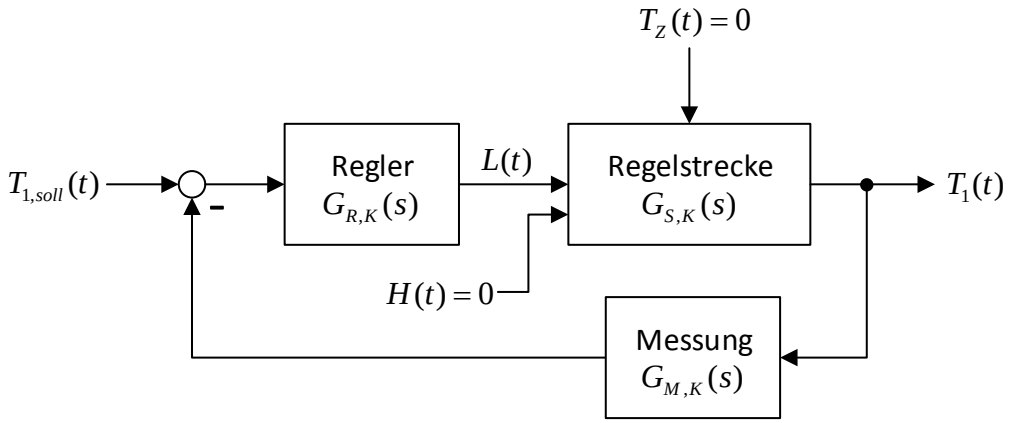
Die Messübertragungsfunktionen $G_{M,H}(s)$ und $G_{M,K}(s)$ berücksichtigen dabei das Übertragungsverhalten der Temperatursensoren von der Temperatur auf einen vom Analog-Digital-Wandler des Mikrocontrollers eingelesenen und quantisierten 10 bit-Zahlenwert. Die Regler sind auf dem Mikrocontroller implementierte Regelalgorithmen mit den Übertragungsfunktionen $G_{R,H}(s)$ und $G_{R,K}(s)$, die in Kapitel 4 entworfen werden. Abb. 3.18 zeigt ein Blockschaltbild der wesentlichen Komponenten der Regler-Hardware und der Kommunikationsschnittstelle. Der vom Temperatursensor ausgegebene Wert $T_1(t)$ für die Spannplattentemperatur wird vom Analog-Digital-Wandler quantisiert und diskretisiert (siehe Abschnitt 2.9.1) und steht der Software als Messgröße $T_{1,M}(t)$ zur Verfügung. Im Speicher des Mikrocontrollers befindet sich eine Variable, die die Solltemperatur $T_{1,soll}(t)$ enthält. Nach einem Vergleich von Sollwert und gemessenem Istwert wird die erhaltene Regelabweichung dem auf der CPU ausgeführten Regelalgorithmus zugeführt und der Wert der Stellgröße berechnet. Dieser Wert wird über den Ein- und Ausgabe-Port des Mikrocontrollers in den Tastgrad $L(t)$ des PWM-Signals, das später zur Ansteuerung der Regelstrecke dient, übersetzt.

3.7 Aufstellung der Modellgleichungen

Im folgenden Unterabschnitt soll das Übertragungsverhalten der Regelstrecke in Form von Übertragungsfunktionen modelliert werden. Hierzu werden eine experimentelle Systemidentifkation, wie sie in Abschnitt 3.2.1 beschrieben wird, durchgeführt und eine Modellschätzung mithilfe der MATLAB System Identification Toolbox vorgenommen. Zunächst jedoch werden einige Vorüberlegungen zur Energieversorgung des Stellglieds angestellt, der Aufbau des Messsystems und Rahmenbedingungen bei der Versuchsdurchführung erläutert sowie eine statische Kennlinie



(a) Geschlossener Regelkreis im Heizbetrieb.



(b) Geschlossener Regelkreis im Kühlbetrieb.

Abbildung 3.17: Blockschaltbilder der geschlossenen Regelkreise im Heiz- und Kühlbetrieb.

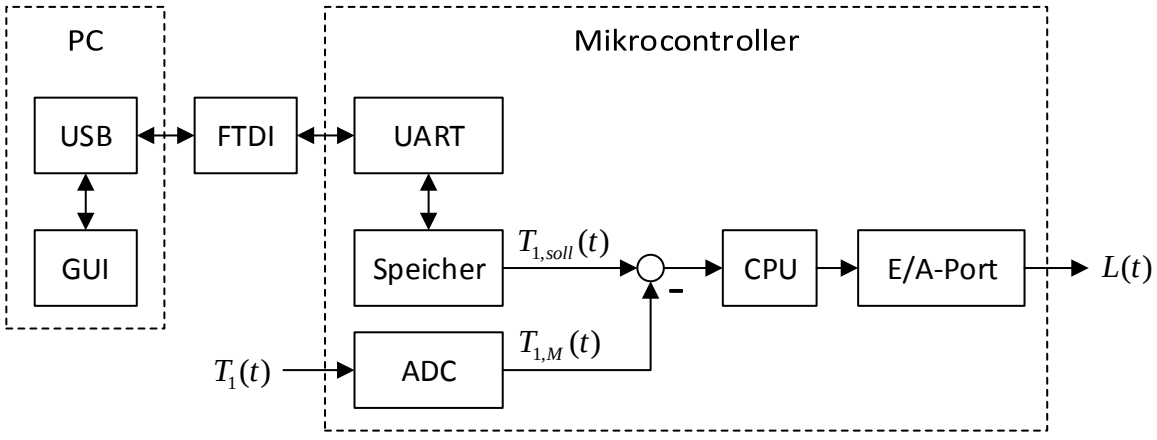


Abbildung 3.18: Hardware-Struktur des Temperaturreglers.

der Strecke ermittelt. In den nachfolgenden Kapiteln werden die Modelle validiert und die Ergebnisse präsentiert.

3.7.1 Festlegung der Versorgungsspannung des Stellglieds

Da die Treiberschaltung des Peltier-Elements, wie bereits in Abschnitt 3.5.2 erwähnt, mit einer konstanten Spannung betrieben wird, muss zunächst eine geeignete Spannung festgelegt werden. Eine konstante Betriebsspannung führt zu einem variablen Strom durch das Peltier-Element, da dessen elektrischer Widerstand temperaturabhängig ist. Die Spannung wird daher mit 11,3 V so gewählt, dass sich im kalten Zustand der Spannplatte (T_1 =−7 °C) ein Strom von 3,5 A durch das Peltier-Element einstellt. Bei Umgebungstemperatur (T_u =19,5 °C) liegt die Stromstärke in diesem Fall bei etwa 4 A. Die Wahl der Stromstärke von 3,5 A wird aus Abb. 3.19 ersichtlich. Das Diagramm zeigt die Sprungantworten der Spannplatten- und Wärmetauschertemperaturen im Kühlbetrieb für verschiedene auf das Peltier-Element aufgeprägte Ströme. Geringe Stromstärken bis 2 A führen dabei nur zu einer geringen Temperaturdifferenz zwischen den beiden Seiten des Peltier-Elements. Ein Optimum der minimalen Spannplattentemperatur stellt sich bei 3,5 A ein. Diese beträgt −8 °C bei T_u =20 °C. Die Wärmetauscher-Temperatur steigt in diesem Fall auf 35 °C an. Werden Ströme im oberen Grenzbereich, das heißt oberhalb von 4,5 A eingestellt, kommt es zur Überhitzung des Systems, was am Abknicken der Temperaturverläufe nach etwa 50 s zu erkennen ist. Begründen lässt sich dieses Verhalten durch die hohe Temperatur des Wärmetauschers. Da das Peltier-Element nur eine bestimmte Temperaturdifferenz zwischen heißer und kalter Seite aufrecht halten kann, steigt mit der Wärmetauscher-Temperatur auch die Spannplattentemperatur. Problematisch ist dieses sogenannte *Allpass-Verhalten* auch für die Regelung des Systems, da ein einfacher PID-Regler zur Regelung nicht ausreicht. Stattdessen sind in [17] verschiedene Reglernetzwerke zur Regelung eines Systems mit Allpass-Anteil angegebenen. Im Gegensatz dazu zeigt das System bei 3,5 A zeitverzögertes proportionales Verhalten. Eine solche Strecke kann vergleichsweise einfach mit einem PID-Regler geregelt werden. Ein Vergleich der Sprungantworten in Abb. 3.19 macht weiterhin deutlich, dass die Anstiegszeit der Strecke bei hohen Stromstärken nicht kleiner als bei 3,5 A ist. Folglich bietet die Einstellung einer höheren Stromstärke hinsichtlich der Dynamik des Systems keinen Vorteil. Aus den genannten Gründen wird daher für alle nachfolgenden Versuche eine konstante Versorgungsspannung von 11,3 V eingestellt, die je nach Temperatur der Spannplatte in einer Stromstärke von 3,5 A bis 4 A im Peltier-Element resultiert.

3.7.2 Aufbau des Messsystems

Der Aufbau des Messsystems ist in Abb. 3.20 dargestellt. Vermessen wird das Übertragungsverhalten der Reihenschaltung aus der Regelstrecke, wie sie in Abb. 3.16 zu sehen ist, und dem Messglied. Auf Basis dieser Messung kann die Übertragungsfunktion $G_0^*(s) = G_S(s)G_M(s)$ des offenen Regelkreises ohne Regler, die für den Reglerentwurf benötigt wird, bestimmt werden. Eine weitere Auflösung der ermittelten Übertragungsfunktion in Strecken- und Messglied-Übertragungsfunktionen ist nicht erforderlich. Die Ansteuerung der Treiberschaltung erfolgt über den Mikrocontroller, auf dem später der Regelalgorithmus ausgeführt wird. Zur Vermessung wird

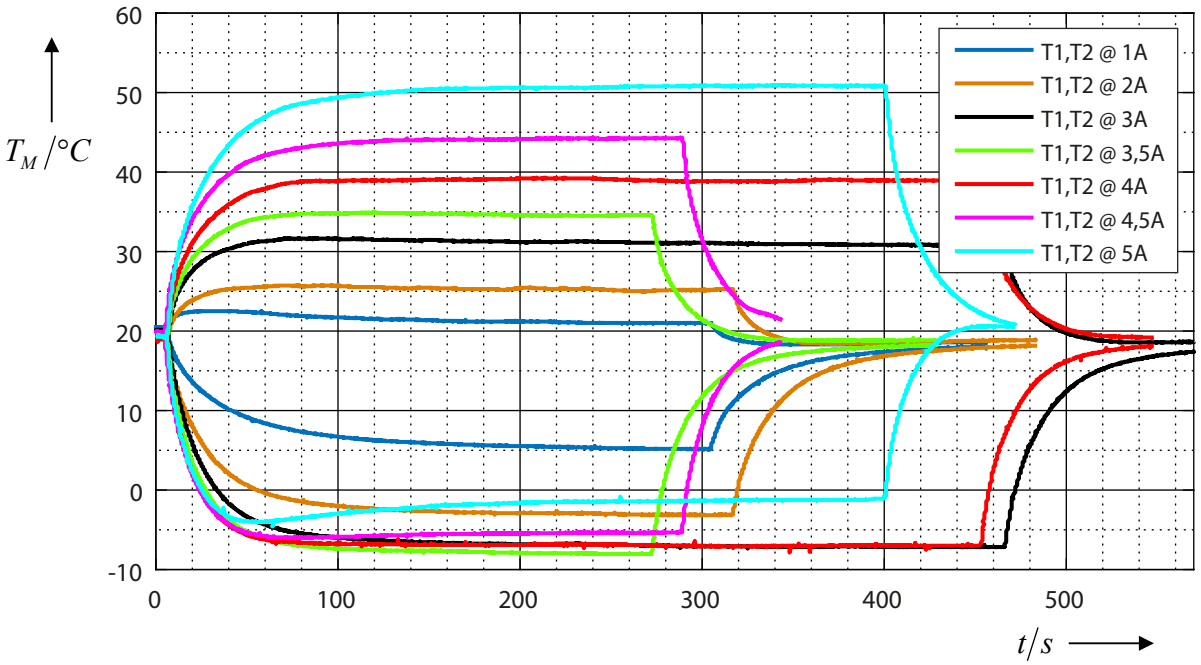


Abbildung 3.19: Sprungantworten des Systems für verschiedene auf das Peltier-Element aufgeprägte konstante Ströme im Kühlbetrieb ($T_u = 20\,°C$). [1]

auf diesem eine Firmware zur Ansteuerung der Regelstrecke mit definierten Eingangssignalen und Erfassung der Spannplattentemperatur ausgeführt. Da beim vorliegenden Messsystem nur Komponenten zum Einsatz kommen, die sich später auch im Regelkreis befinden werden, ist eine sehr genaue und unproblematische Erfassung des Systemverhaltens möglich.

3.7.3 Rahmenbedingungen bei der Versuchsdurchführung

Da die Modellierung der Strecke inklusive Messglied auf die Bestimmung des Übertragungsverhaltens der Stellgröße $L(t)$ auf die gemessene Regelgröße $T_{1,M}(t)$ abzielt, müssen Störungen $T_Z(t)$ während der Messung so klein wie möglich gehalten werden. Andernfalls ist keine genaue Abbildung des Systemverhaltens möglich, da Störungen zu Änderungen der Regelgröße führen, die nicht auf Änderungen der Stellgröße zurückzuführen sind. Aufgrund der langen Versuchsdauer von bis zu 54 min sind hierbei insbesondere niederfrequente Änderungen der Umgebungstemperatur problematisch. Daher muss die Umgebungstemperatur während der Versuchsdurchführung möglichst konstant gehalten werden. Die Auswirkung hochfrequenter Störungen auf das Modellierungsergebnis fällt aufgrund der großen Zeitkonstante der Strecke sehr viel kleiner aus. Zudem können diese vor der Modellierung durch ein Tiefpassfilter gedämpft werden. Eine weitere Rahmenbedingung betrifft die Wahl der Abtastperiode der Messgrößen. Diese wird auf einen Wert von 100 ms festgelegt. Ein Aliasing-Fehler (siehe Abschnitt 2.9.1) ist bei dieser hohen Abtastrate auszuschließen. [3]

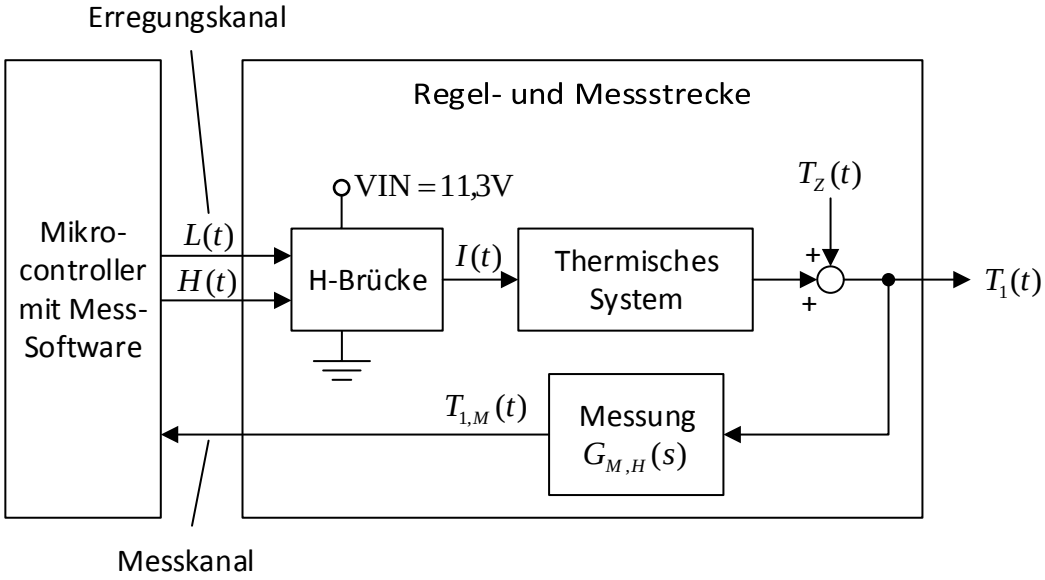


Abbildung 3.20: Blockschaltbild des Messaufbaus zur Ermittlung des Ein- und Ausgangsverhaltens der Strecke inklusive Messglied.

3.7.4 Ermittlung der statischen Kennlinie

Wie in Abschnitt 3.2.2 erläutert, können Systeme linear oder nichtlinear sein. Zur Untersuchung des hier vorliegenden Systems in Hinblick auf diese Eigenschaft wird eine statische Kennlinie aufgezeichnet. Hierzu werden Sprungantworten der gemessenen Spannplattentemperatur $T_{1,M}(t)$ für verschiedene Werte der Eingangsgröße, hier der Tastgrad $L(t)$ eines PWM-Signals, aufgezeichnet und der stationäre Endwert der Temperatur $T_{1,M}(\infty)$ gegen die Eingangsgröße aufgetragen. Das Ergebnis ist in Abb. 3.21 dargestellt. Dabei wurde nur der Kühlbetrieb betrachtet, da die Spannplattentemperatur im Heizbetrieb erst bei sehr hohen Temperaturen, bei denen es bereits zu einer Beschädigung des Systems käme, einen stationären Wert annimmt. Aus der vorliegenden statischen Kennlinie wird deutlich, dass das System nichtlinear ist. Wäre es linear, so wäre die Kennlinie eine Gerade. Da jedoch die Strecke meist in einem festen Arbeitspunkt L_A, $T_{1,M,A}$ betrieben und die Regelung hauptsächlich Störungen ausgleichen soll, kann die Kennlinie, wie ebenfalls in Abb. 3.21 zu sehen, durch eine Tangente im Arbeitspunkt angenähert werden. Das System wird also um den Arbeitspunkt herum linearisiert. [3]

3.7.5 Ermittlung der Modellgleichungen

Zur Modellierung der Übertragungsfunktionen der Strecke muss das Ein- und Ausgangsverhalten des Systems messtechnisch erfasst werden. Hierzu wird auf dem Mikrocontroller die Firmware aus Listing A.3 zur Modellierung des Kühlbetriebs und die Firmware aus Listing A.4 für den Heizbetrieb ausgeführt. Im Kühlbetrieb wird der Tastgrad des PWM-Signals 13-mal zwischen 0 und 255 umgeschaltet und nach jedem Umschalten eine Dauer von 120 s abgewartet. Somit erhält man mehrere Sprungantworten der gemessenen Spannplattentemperatur $T_{1,M}(t)$, die sowohl

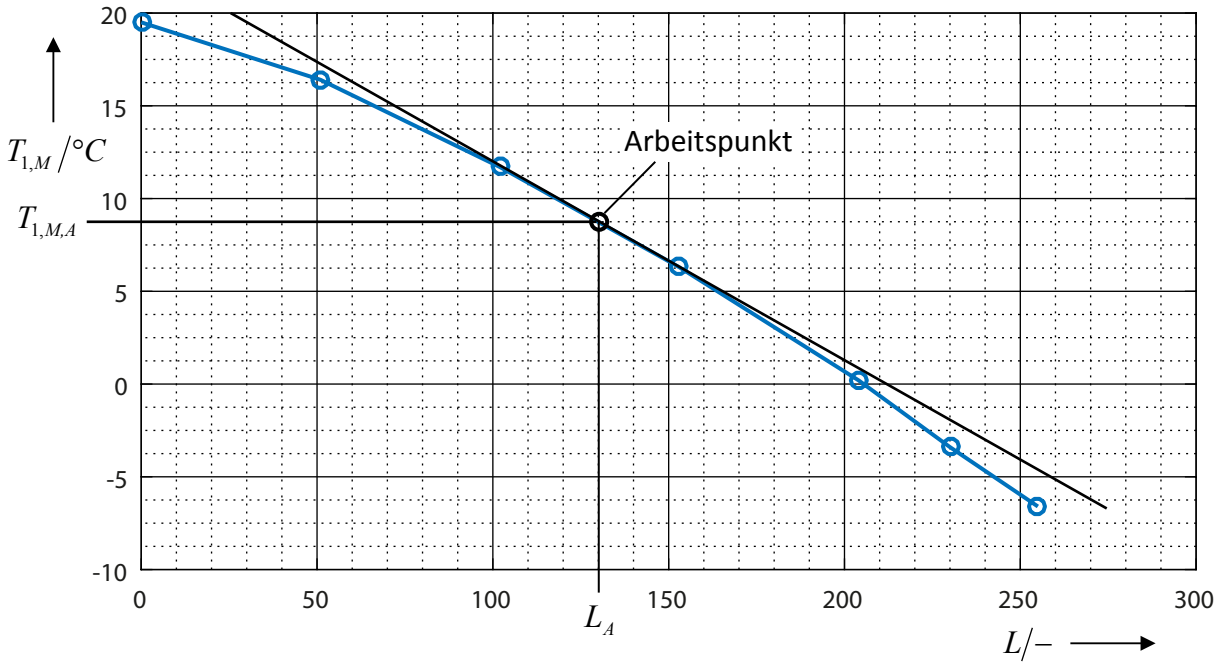


Abbildung 3.21: Statische Kennlinie der Regelstrecke im Kühlbetrieb ($T_u = 19{,}5\,°C$).

das Anstiegs- als auch das Ausklingverhalten der Temperatur wiederspiegeln. Der gesamte, in Abb. 3.22 dargestellte Datensatz, der für die Modellierung herangezogen wird, enthält dabei 32 400 Messpunkte im Abstand von 100 ms. Eine einzelne Sprungantwort zeigt Abb. 3.23. Der Strom im Peltier-Element beträgt dabei, wie zuvor erwähnt, zwischen 3,5 A und 4 A und hängt von der Temperatur sowie dem Tastgrad $L(t)$ des PWM-Signals ab.

Im Heizbetrieb wird der Tastgrad des PWM-Signals nicht automatisch festgelegt, da die Strecke hier für den zulässigen Temperaturbereich von $-15\,°C$ bis $75\,°C$ integrierendes Verhalten aufweist und somit die Temperatur $T_1(t)$ der Spannplatte bei konstantem Tastgrad $L(t)$ des PWM-Signals so weit anstiege, dass es zur Zerstörung des Systems käme. Daher wird im Heizbetrieb der Tastgrad durch ein am Mikrocontroller angeschlossenes Potentiometer manuell vorgegeben. Auf diese Weise ist es einfach möglich, den Tastgrad wieder auf 0 zurückzusetzen, wenn die Spannplatte eine kritische Temperatur von $70\,°C$ erreicht hat. Die Sprungantwort der Strecke im Heizbetrieb zeigt Abb. 3.25. Der für die Modellierung des Heizbetriebs zugrundeliegende und in Abb. 3.24 dargestellte Datensatz enthält 17 solcher Sprungantworten und umfasst 20 034 Messpunkte im Abstand von 100 ms.

Wie man Abb. 3.22 und Abb. 3.24 entnehmen kann, werden die Daten vor der Modell-Schätzung bearbeitet. Schon vor dem Import wurden sämtliche Daten durch ein Tiefpassfilter geglättet. Zusätzlich werden die Daten innerhalb der MATLAB System Identification Toolbox (Abb. 3.26) von überlagerten Offsets bereinigt, da in die Modellbildung nur die Änderungen der Messgrößen, nicht jedoch deren Absolutwerte, einbezogen werden. Weiterhin werden die Datensätze in jeweils zwei gleich große Teil-Datensätze aufgeteilt. Die erste Hälfte dient jeweils als Grundlage für die Parameterschätzung des Modells, die zweite Hälfte wird in Abschnitt 3.8 für die Validierung der Modelle benötigt.

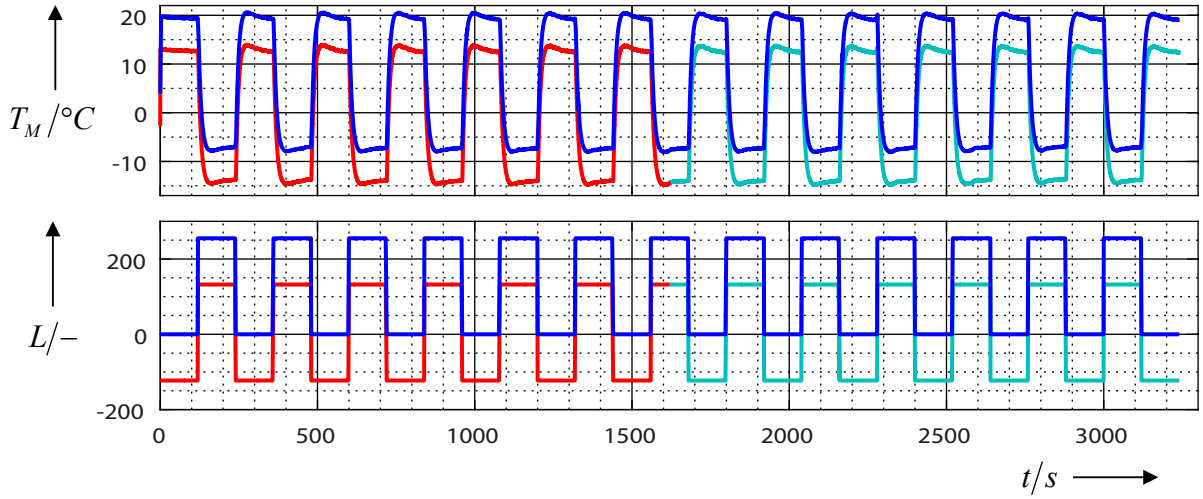


Abbildung 3.22: Datensatz für die Modellierung des Kühlbetriebs und Validierung des Modells. Blau: Originaldaten, Rot: Modellierungsdaten, Cyan: Validierungsdaten ($T_u = 19{,}5\,°C$).

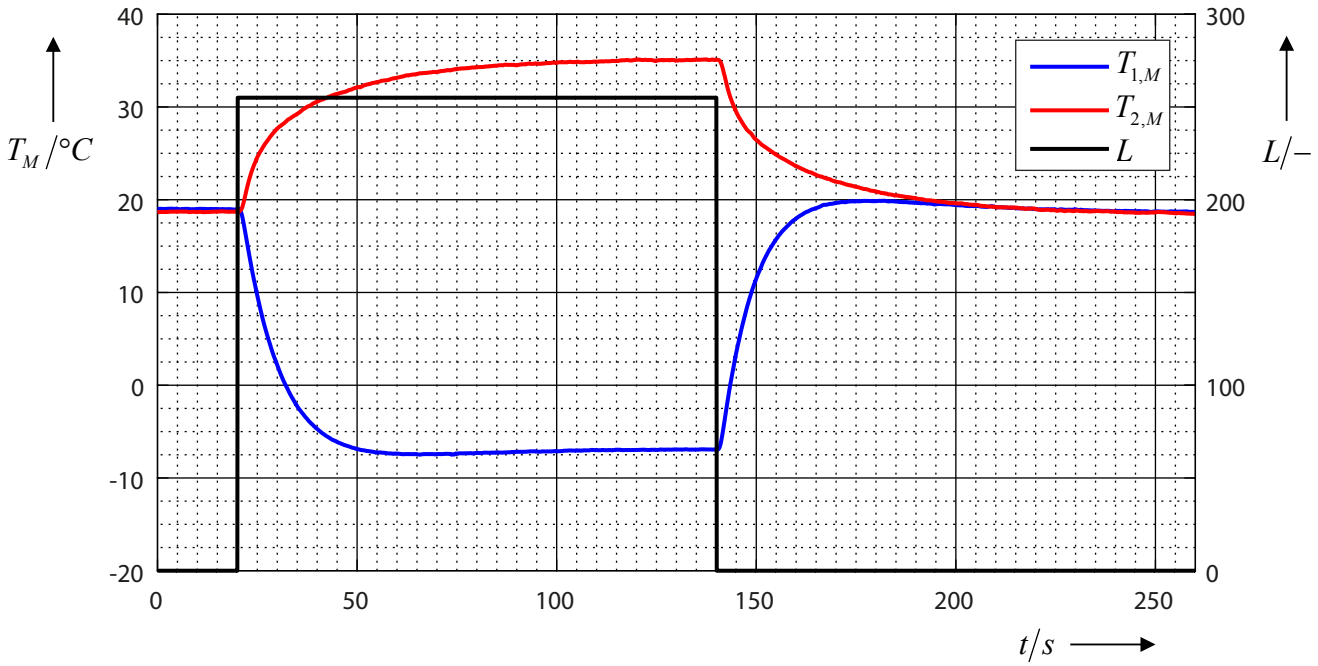


Abbildung 3.23: Sprungantwort der Regelstrecke im Kühlbetrieb ($T_u = 19{,}5\,°C$).

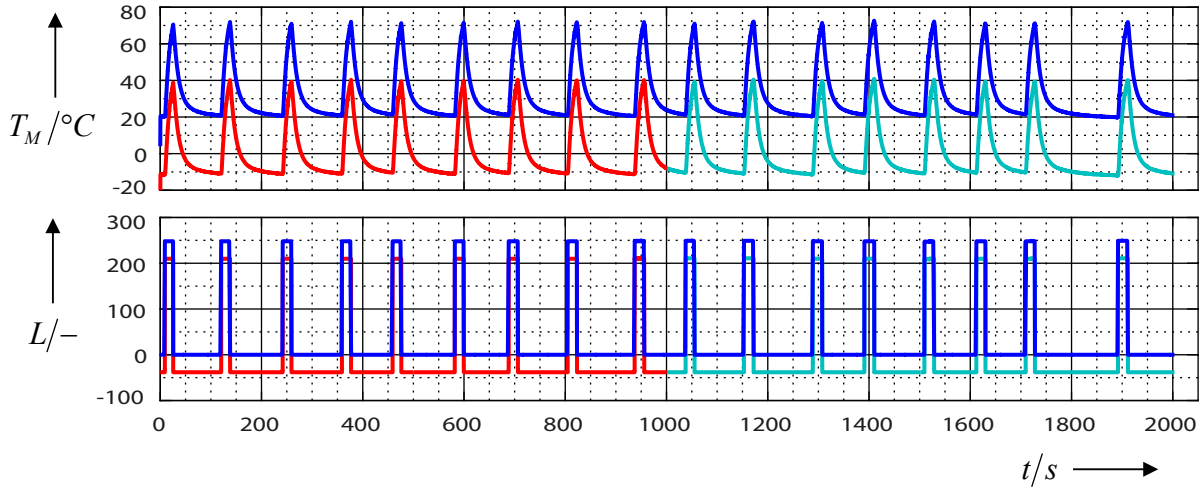


Abbildung 3.24: Datensatz für die Modellierung des Heizbetriebs und Validierung des Modells. Blau: Originaldaten, Rot: Modellierungsdaten, Cyan: Validierungsdaten ($T_u = 19{,}5\,°C$).

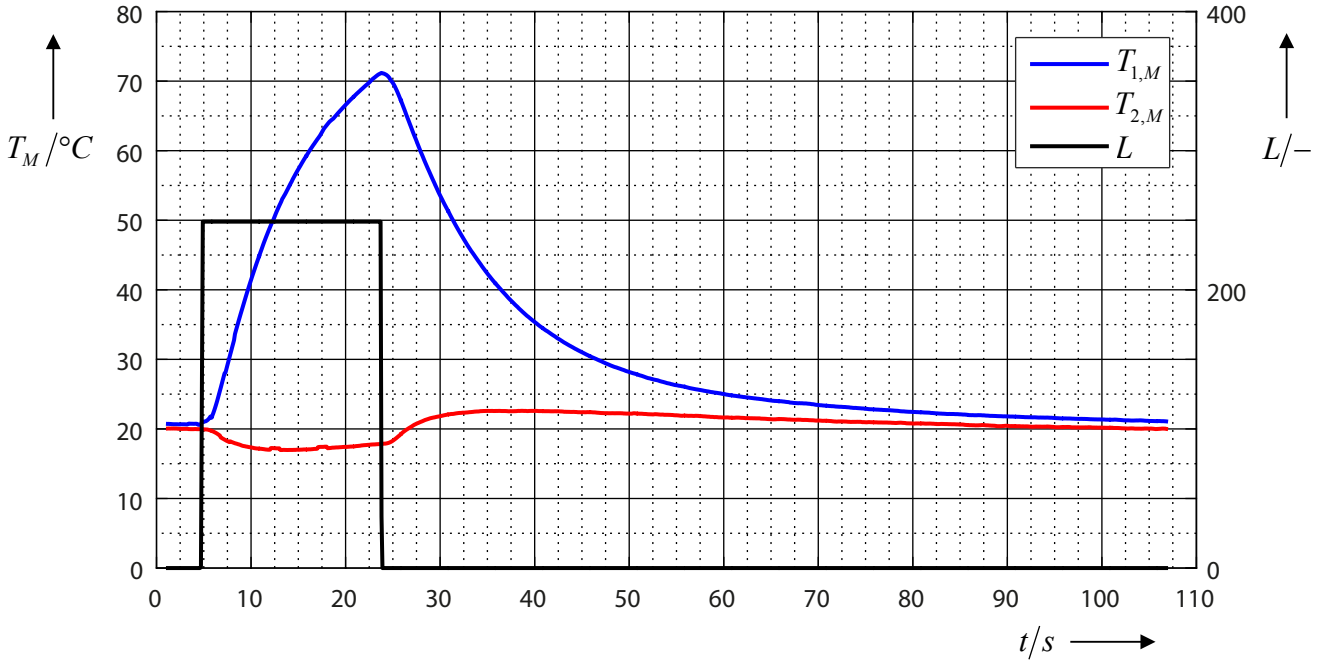


Abbildung 3.25: Sprungantwort der Regelstrecke im Heizbetrieb ($T_u = 19{,}5\,°C$).

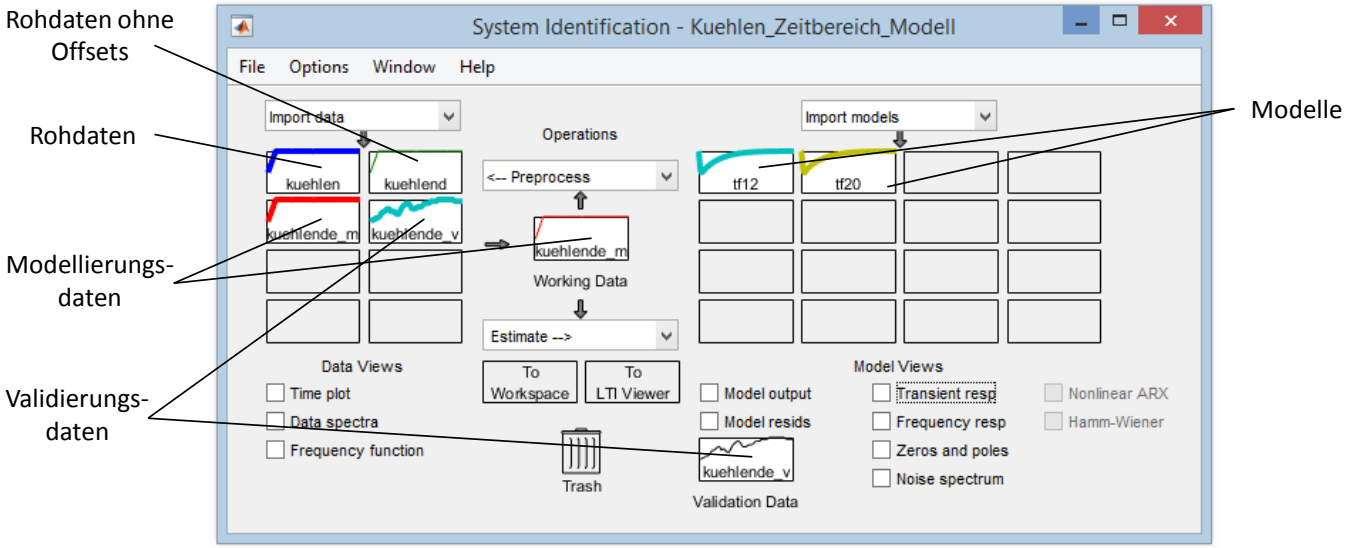


Abbildung 3.26: Hauptansicht der System Identification Toolbox mit geladenen und bearbeiteten Modellierungsdaten sowie geschätzten Modellen.

Auf Basis der aufbereiteten Datensätze können nun Modelle erstellt werden. Die System Identification Toolbox bietet hierfür eine große Bandbreite möglicher Modelltypen, deren Parameter automatisch geschätzt werden können. Im vorliegenden Fall werden zeitkontinuierliche Übertragungsfunktions-Modelle erstellt. Für diese muss jeweils die Anzahl der Pole und Nullstellen festgelegt werden, bevor die Parameterschätzung die optimalen Systemkonstanten bestimmt. Durch systematisches Erhöhen der Anzahl an Polen und Nullstellen kann man verschiedene Modelle erstellen und deren Güte anhand der Übereinstimmung zwischen Modell und Messung beurteilen. Auf diese Weise erhält man für den Kühlbetrieb das Modell

$$G^*_{0,K,1}(s) = \frac{-0{,}1252s - 0{,}003991}{s^4 + 5{,}902s^3 + 10{,}69s^2 + 1{,}43s + 0{,}03873} \tag{3.24}$$

mit vier Polen und einer Nullstelle sowie das Modell

$$G^*_{0,K,2}(s) = \frac{-1{,}308s^2 - 0{,}1164s - 0{,}002262}{s^6 + 20{,}89s^5 + 71{,}85s^4 + 115{,}3s^3 + 21{,}4s^2 + 1{,}251s + 0{,}022} \tag{3.25}$$

mit sechs Polen und zwei Nullstellen für die Übertragungsfunktion $G^*_{0,K}(s) = G_{S,K}(s)G_M(s)$ des offenen Regelkreises ohne Regler. Entsprechend ergeben sich für den Heizbetrieb das Modell

$$G^*_{0,H,1}(s) = \frac{0{,}0007727s^3 + 0{,}04042s^2 + 0{,}005635s + 0{,}0001067}{s^4 + 1{,}873s^3 + 0{,}4104s^2 + 0{,}02579s + 0{,}0003251} \tag{3.26}$$

mit vier Polen und drei Nullstellen und das Modell

$$G^*_{0,H,2}(s) = \frac{0{,}007519s^4 + 0{,}05967s^3 + 0{,}09061s^2 + 0{,}03587s + 0{,}0007898}{s^5 + 5{,}028s^4 + 4{,}856s^3 + 1{,}95s^2 + 0{,}1729s + 0{,}002388} \tag{3.27}$$

mit fünf Polen und vier Nullstellen für die Übertragungsfunktion $G^*_{0,H}(s) = G_{S,H}(s)G_M(s)$ des offenen Regelkreises ohne Regler. Details zur Modellerstellung mithilfe der System Identification Toolbox können der Dokumentation in [18] entnommen werden.

3.8 Modellvalidierung

Im nächsten Schritt erfolgt die Validierung der zuvor erstellten Modelle. Dazu wird der anhand der Modell-Übertragungsfunktion berechnete Verlauf der Systemausgangsgröße hinter dem Messglied $T_{1,M}(t)$ mit dem zuvor angelegten Validierungsdatensatz, der den gemessenen Verlauf der Spannplattentemperatur $T_{1,M}(t)$ enthält, verglichen und die Übereinstimmung berechnet. Die Ergebnisse sind Tabelle 3.5 zu entnehmen. Weiterhin ist dort die Übereinstimmung zwischen Modell und Modellierungsdatensatz aufgeführt.

Die hohe Übereinstimmung zwischen den Modellen und den Messungen zeigen auch die Temperaturverläufe in Abb. 3.27 für den Kühlbetrieb und in Abb. 3.28 für den Heizbetrieb. Die jeweils schwarze Kurve zeigt dabei einen Ausschnitt aus dem Validierungsdatensatz und die farbigen Kurven die gemessene Spannplattentemperatur $T_{1,M}(t)$ als Ergebnis der Berechnung mittels der zuvor bestimmten Modellübertragungsfunktionen von Strecke und Messglied.

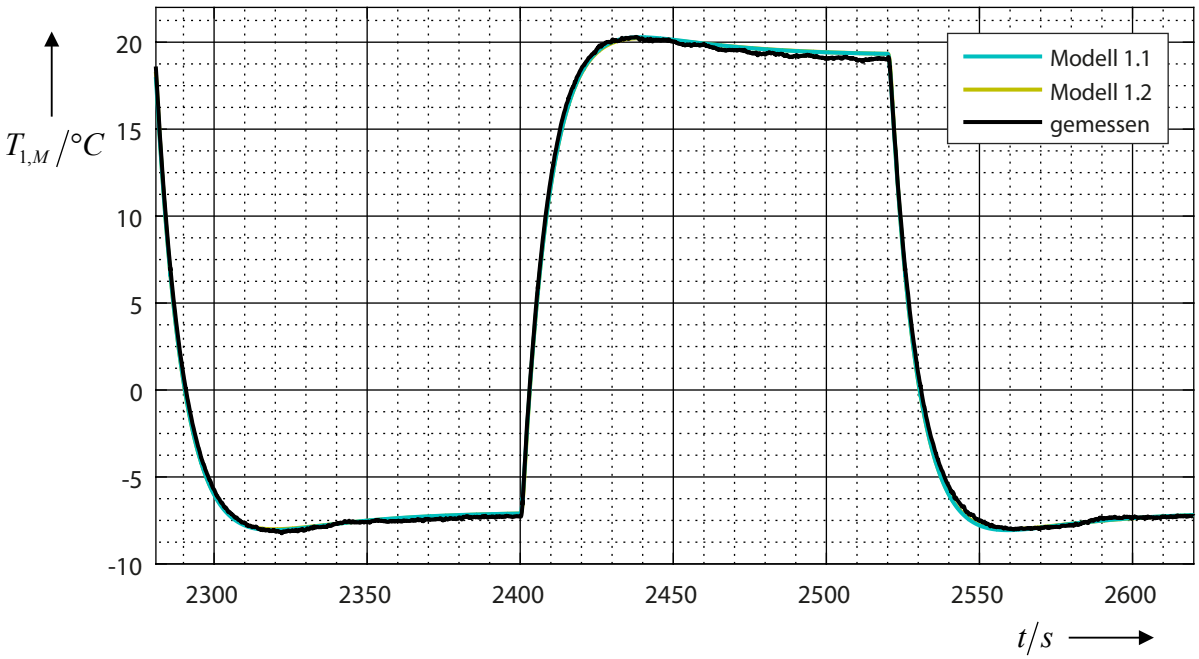


Abbildung 3.27: Gemessene und modellierte Spannplattentemperatur im Kühlbetrieb ($T_u = 19{,}5\,°\mathrm{C}$).

3.9 Ergebnisse der Modellierung

Im Folgenden sollen die Eigenschaften der in Abschnitt 3.7.5 ermittelten Modelle genauer untersucht werden, bevor sie in Abschnitt 4.2.4 als Grundlage für die Einstellung der Reglerparameter verwendet werden. Entsprechend Abschnitt 3.2.1 handelt es sich bei den erstellten Modellen um parametrische Blackbox-Modelle, da das Systemverhalten in Form analytischer Gleichungen beschrieben wird. Die innere Struktur des Systems wird nicht weiter beachtet, sondern lediglich

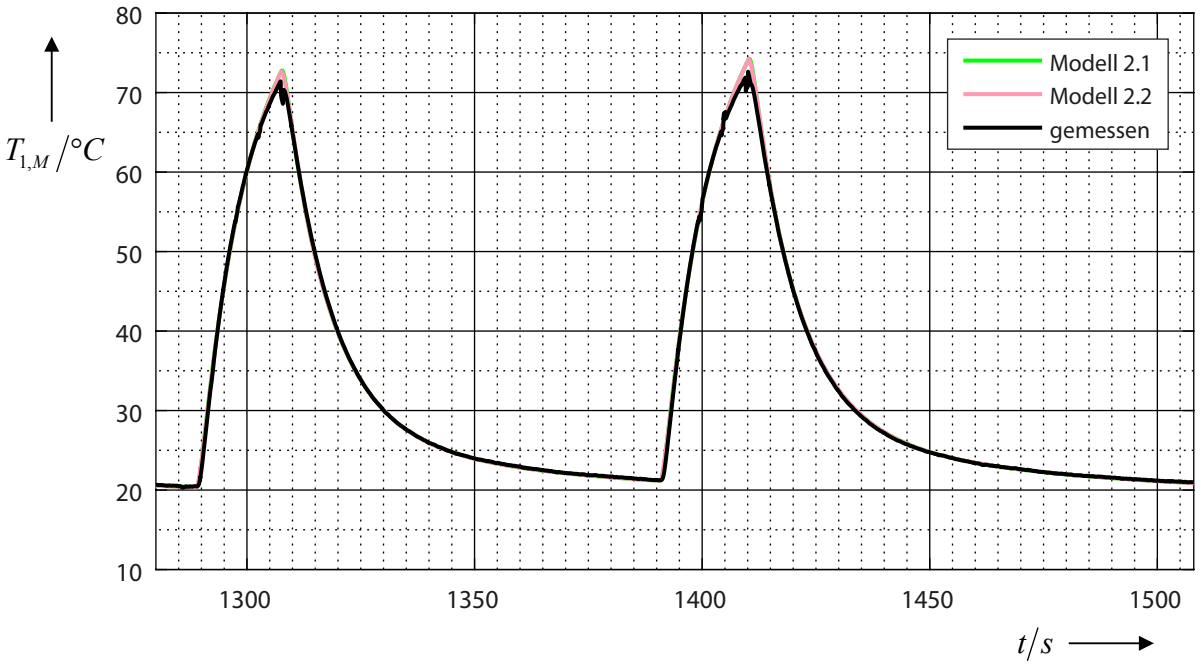


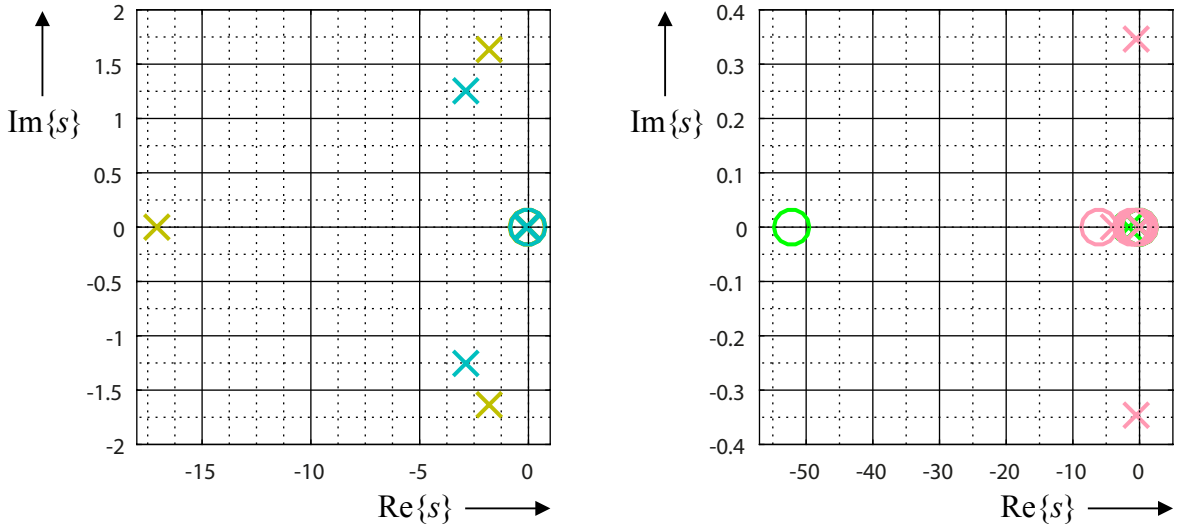
Abbildung 3.28: Gemessene und modellierte Spannplattentemperatur im Heizbetrieb ($T_u = 19{,}5\,°\mathrm{C}$).

das Ein- und Ausgangsverhalten abgebildet. Weiterhin ist das Modell zeitkontinuierlich und beschreibt die zeitkontinuierliche Regelstrecke. Zeitdiskret wird das System erst durch den digitalen Regler. Anhand der Modelle lassen sich auch Aussagen über die in Abschnitt 3.2.2 besprochenen Eigenschaften der tatsächlichen Regelstrecke inklusive Messglied machen. So ist das System konzentriert, sämtliche Systemgrößen werden also nur zeitlich, nicht jedoch räumlich aufgelöst. Weiterhin ist es zeitinvariant, da keine explizite Zeitabhängigkeit vorliegt, und kausal, da die Anzahl der Pole in allen Modell-Übertragungsfunktionen größer als die Anzahl der Nullstellen ist. Eine Aussage über die Linearität wurde bereits durch Betrachtung der statischen Kennlinie in Abschnitt 3.7.4 gemacht. Die letzte wichtige Eigenschaft der Strecke stellt die Stabilität dar. Um diese zu beurteilen, können die Pol-Nullstellen-Verteilungen aus Abb. 3.29 oder die Gewichtsfunktionen aus Abb. 3.30, die jeweils aus den Übertragungsfunktionen berechnet wurden, herangezogen werden. Aus den Pol-Nullstellen-Verteilungen wird ersichtlich, dass sämtliche

Tabelle 3.5: Übereinstimmung zwischen Modellen und den zugehörigen Modellierungs- und Validierungsdatensätzen.

Nr.	Modell	Übereinstimmung mit	
		Modellierungsdaten	Validierungsdaten
1.1	$G^*_{0,K,1}(s)$	98,10 %	98,01 %
1.2	$G^*_{0,K,2}(s)$	98,06 %	98,01 %
2.1	$G^*_{0,H,1}(s)$	97,47 %	97,43 %
2.2	$G^*_{0,H,2}(s)$	97,68 %	97,49 %

Pole einen negativen Realteil besitzen. Folglich ist das System sowohl im Kühl- als auch im Heizbetrieb bereits ohne Regler asymptotisch stabil. Dies wird auch aus den Gewichtsfunktionen des Systems deutlich, die jeweils für $t \to \infty$ gegen null gehen.



(a) Pol-Nullstellen-Verteilung im Kühlbetrieb (cyan: Modell 1.1, khaki: Modell 1.2).

(b) Pol-Nullstellen-Verteilung im Heizbetrieb (grün: Modell 2.1, pink: Modell 2.2).

Abbildung 3.29: Pol-Nullstellen-Verteilungen der geschätzten Modelle im Kühl- und Heizbetrieb.

Betrachtet man die in Abb. 3.31 dargestellten Übergangsfunktionen der Modelle, erhält man weitere Aussagen über das Systemverhalten. Es wird deutlich, dass das System neben dem Kühlbetrieb auch im Heizbetrieb, anders als zuvor angenommen, kein integrierendes, sondern proportional zeitverzögertes Verhalten aufweist und sich entsprechend auch hier eine stationäre Temperatur einstellt. Dies lässt sich dadurch erklären, dass der von der Spannplatte an die Luft abgegebene Wärmestrom mit steigender Spannplattentemperatur ebenfalls ansteigt und sich nach etwa 200 Sekunden ein Gleichgewicht zwischen abgegebenem Wärmestrom und im Peltier-Element dissipierter Energie einstellt. Die Spannplattentemperatur ist in diesem Fall jedoch so groß, dass das System zerstört würde. Daher wird im realen Betrieb nur die ansteigende Flanke der Übergangsfunktion ausgenutzt. Denn wie aus Abb. 3.25 ersichtlich wird, erreicht die Spannplatte die maximal erlaubte Temperatur von 70 °C bereits nach 17,5 s. Aus den Übergangsfunktionen lassen sich die Anstiegszeiten $T_{a,H} = 72{,}189\,\text{s}$ im Heizbetrieb und $T_{a,K} = 16{,}109\,\text{s}$ im Kühlbetrieb sowie die korrespondierenden Verzugszeiten $T_{u,K} = 1{,}427\,\text{s}$ und $T_{u,H} = 2{,}048\,\text{s}$ ermitteln. Die Anstiegszeit ist dabei die Zeit, die die Temperatur benötigt, um von 10 % auf 90 % des stationären Werts anzusteigen, während die Verzugszeit die zum Anstieg auf 10 % des stationären Werts benötigte Zeit beschreibt. Zudem erhält man aus den Übergangsfunktionen die Verstärkungsfaktoren der Regelstrecke inklusive Messglied von $K_{S,H} = 0{,}331$ im Heizbetrieb und $K_{S,K} = -0{,}103$ im Kühlbetrieb.

Die Bode-Diagramme aus Abb. 3.32 stellen den anhand des Modells berechneten Frequenzgang des Systems im Kühl- und im Heizbetrieb dar. Zusätzlich sind für die einzelnen Modelle 95 %-Konfidenzintervalle angegeben. Es zeigt sich, dass der Frequenzgang des Modells 2.1 für Frequenzen oberhalb von 5 Hz mit einer erheblichen Unsicherheit behaftet ist. Die anderen Modelle

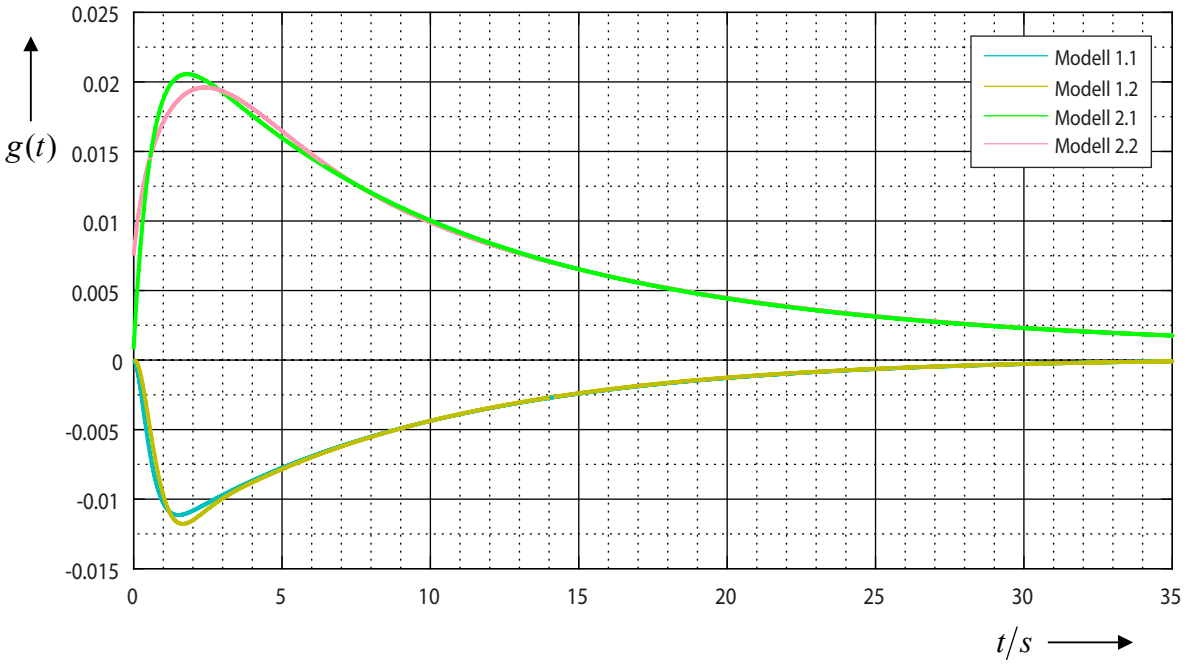


Abbildung 3.30: Gewichtsfunktionen der geschätzten Modelle im Kühl- und Heizbetrieb.

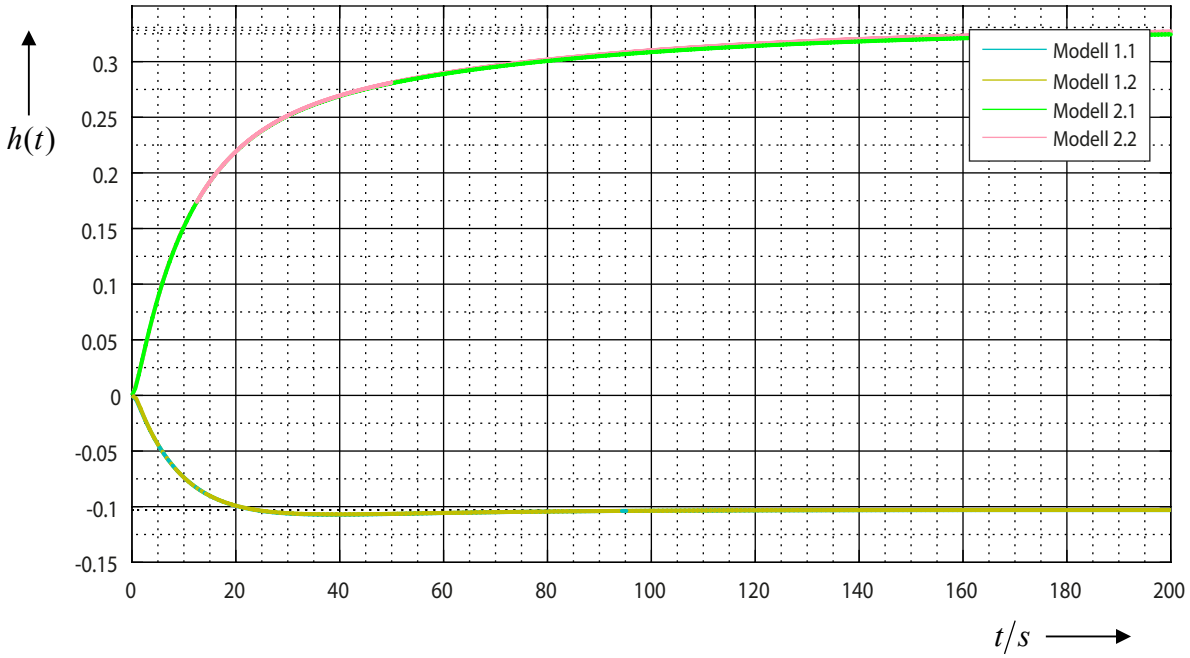
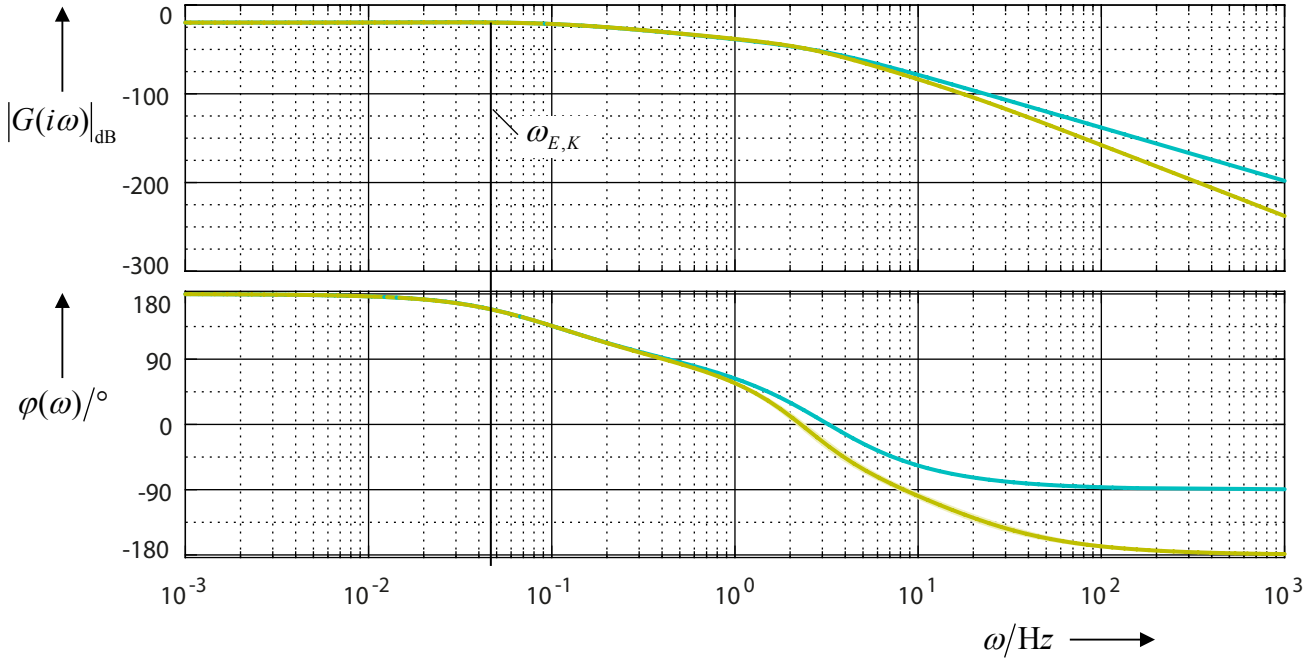


Abbildung 3.31: Übergangsfunktionen der geschätzten Modelle im Kühl- und Heizbetrieb.

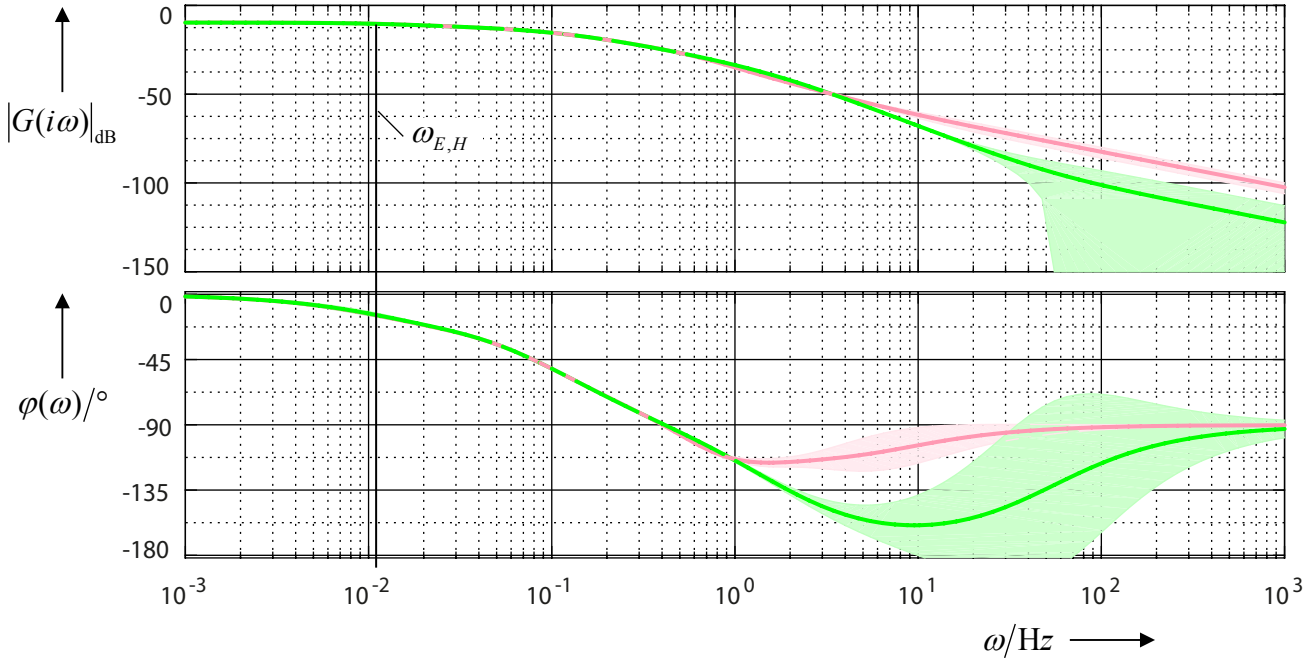
sind im Gegensatz dazu auch bei hohen Frequenzen genau. Beide Bode-Diagramme spiegeln das proportionale, zeitverzögerte Verhalten des Systems wieder. Da die Amplitudengänge unterhalb der Eckfrequenz ω_E des Systems horizontal verlaufen, nimmt die Spannplattentemperatur für $t \to \infty$ einen stationären Endwert an, sodass das System global proportionales Verhalten hat. Das zeitverzögerte Verhalten zeigt sich im Abknicken der Amplitudengänge oberhalb der Eckfrequenz. Diese liegt im Heizbetrieb bei $\omega_{E,H} = 1{,}425 \cdot 10^{-2}\,\text{Hz}$ und im Kühlbetrieb bei $\omega_{E,K} = 6{,}208 \cdot 10^{-2}\,\text{Hz}$. Es gilt der Zusammenhang $\omega_E = \frac{1}{T_a}$. Eine Betrachtung des Phasengangs des Systems im Kühlbetrieb macht deutlich, dass die Spannplattentemperatur bei aufgeprägter positiver Eingangsgröße absinkt, denn in diesem Bereich beträgt die Phasenverschiebung zwischen Ein- und Ausgangssignal genau 180°. Im Gegensatz dazu führt eine positive Eingangsgröße im Heizbetrieb auch zu einer steigenden Spannplattentemperatur, da hier die Phasenverschiebung 0° beträgt.

3.10 Fazit zur Modellbildung

Im vorangegangenen Kapitel wurden zunächst das Modellierungsziel und die Modellannahmen festgelegt. Anschließend wurde die Gesamtfunktion der Regelstrecke erläutert und die Funktionsweise ihrer einzelnen Bestandteile genauer erörtert. Hieran schloss sich eine theoretische Modellierung des thermischen Systems, die Aufschluss über die physikalischen Zusammenhänge im thermischen Teil des Systems gab, an. Nachdem das System in Form von Blockschaltbildern abgebildet wurde, konnten auf Basis von Messungen des Ein- und Ausgangsverhaltens des Systems parametrische Blackbox-Modelle in Form von Übertragungsfunktionen der gesamten Regelstrecke einschließlich des Messglieds ermittelt werden. Wie in den Modellannahmen erläutert, war dabei ein lineares Modell trotz des nichtlinearen Verhaltens des Systems ausreichend genau. Dies spiegelte sich auch im Vergleich zwischen den Validierungsdaten und den anhand der Modelle berechneten Daten wieder. Auf Basis der erstellten Modelle kann daher in Kapitel 4 der Entwurf des Reglers erfolgen.



(a) Bode-Diagramm im Kühlbetrieb (cyan: Modell 1.1, khaki: Modell 1.2).



(b) Bode-Diagramm im Heizbetrieb (grün: Modell 2.1, pink: Modell 2.2).

Abbildung 3.32: Bode-Diagramme der geschätzten Modelle im Kühl- und Heizbetrieb.

4 Entwurf und Implementierung des Reglers

Ziel dieses Kapitels ist die Entwicklung zweier Temperaturregler für die Spannplattentemperatur im Kühl- und im Heizbetrieb. Hierzu werden zunächst ausführlich die Grundlagen des Reglerentwurfes und einige typische Regler betrachtet. Daran anschließend erfolgt basierend auf den Ergebnissen der Modellierung im vorangegangenen Kapitel der schrittweise Entwurf der Regler durch Erfassung der Regelungsaufgabe, Definition der Güte-Anforderungen, Festlegung der Reglerstruktur und Bestimmung der Reglerparameter. Der so entwickelte Regler wird schließlich als digitaler Algorithmus auf einem Mikrocontroller implementiert. Den Abschluss dieses Kapitels bilden die Simulation und Analyse der geschlossenen Regelkreise, in deren Rahmen unter anderem die Einhaltung der zuvor definierten Güte-Anforderungen überprüft wird.

4.1 Grundlagen des Reglerentwurfs

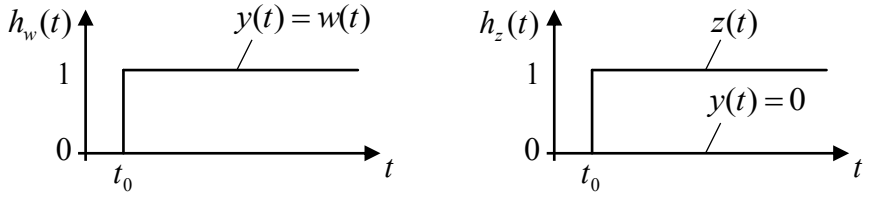
In diesem Grundlagenkapitel soll eine kurze Einführung in die Thematik des Reglerentwurfs gegeben werden. Dazu wird zunächst die Zielsetzung des Reglerentwurfes näher beschrieben. Anschließend wird mit dem PID-Regler ein universell einsetzbarer und einfach zu realisierender sowie einzustellender Regler vorgestellt und auf ein Verfahren zu dessen Diskretisierung eingegangen. Den Hauptteil dieses Einführungskapitels stellt eine umfangreiche Vorstellung verschiedener Verfahren für den Reglerentwurf auf Basis eines Modells der Regelstrecke dar.

4.1.1 Zielsetzung des Reglerentwurfs

Wie bereits in Abschnitt 2.1 erläutert, hat der Regler die Aufgabe, die Regelgröße $y(t)$ der Führungsgröße $w(t)$ nachzuführen und Störungen $z(t)$ auf die Regelstrecke auszugleichen. Entsprechend muss der geschlossene Regelkreis aus Abb. 2.1 folgende Anforderungen erfüllen:

- Der geschlossene Regelkreis muss stabil sein.
- Die Störgröße $z(t)$ sollte die Regelgröße $y(t)$ möglichst wenig beeinflussen.
- Die Regelgröße $y(t)$ soll der Führungsgröße $w(t)$ möglichst schnell und mit geringer Regelabweichung folgen.
- Parameteränderungen der Regelstrecke sollen sich möglichst gering auf das Verhalten des geschlossenen Regelkreises auswirken.

Ein idealer Regler würde folglich die in Abb. 4.1 gezeigte Führ- und Störübergangsfunktion besitzen. Bei aufgegebenem Einheitssprung der Führungsgröße $w(t) = \sigma(t)$ würde die Regelgröße $y(t)$ sprunghaft auf den neuen Wert der Führungsgröße ansteigen. Eine sprunghafte Störung $z(t) = \sigma(t)$ führt im Idealfall zu keiner Änderung der Regelgröße. Dass diese Forderungen in der Realität nicht einhaltbar sind, ist anschaulich klar, denn die meisten realen Systeme sind nicht sprungfähig, sondern besitzen eine Trägheit, die zu einer zeitlichen Verzögerung des Anstiegs der Regelgröße $y(t)$ führt. Weiterhin kommt es bei realen Prozessen unter Umständen zunächst zu Schwingungen der Regelgröße um den Sollwert, bevor sich der stationäre Wert einstellt. Auch eine bleibende Abweichung der Regelgröße vom Sollwert für $t \to \infty$ ist bei realen Systemen möglich und wird als bleibende Regelabweichung $e(\infty)$ bezeichnet. [39]



(a) Führübergangsfunktion eines idealen Reglers.

(b) Störübergangsfunktion eines idealen Reglers.

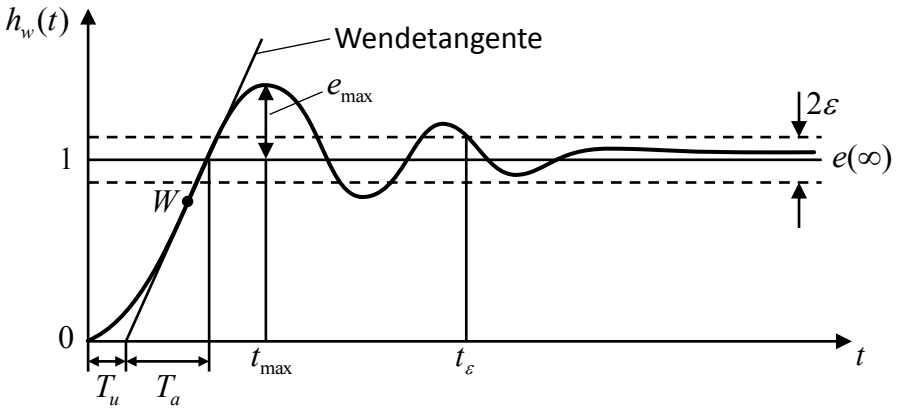
Abbildung 4.1: Führ- und Störübergangsfunktion eines idealen Reglers.

Abb. 4.2 zeigt beispielhaft die Verläufe der Führ- und Störübergangsfunktionen eines realen Systems. Die Abweichungen vom Idealverhalten können durch sogenannte Gütemaße quantitativ beschrieben werden. Dies sind im Einzelnen:

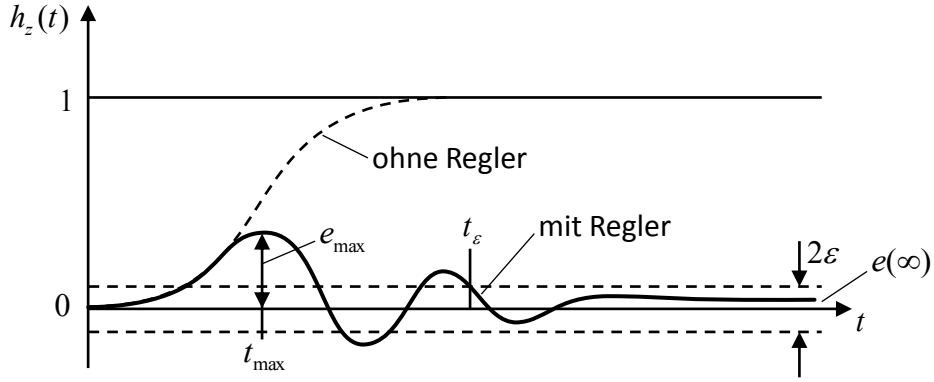
- Die Verzugszeit T_u als Schnittpunkt der Wendetangente an die Führübergangsfunktion mit der t-Achse.
- Die Anstiegszeit T_a als Schnittpunkt der Wendetangente mit der Linie $h_w(t) = 1$.
- Die Ausregelzeit t_ε als Zeitpunkt, ab dem die Übergangsfunktion einen vorgegebenen Korridor der Breite 2ε um den Sollwert nicht mehr verlässt.
- Die maximale Überschwingweite $e_{\max}$, die die maximale Abweichung zwischen Sollwert und Regelgröße beschreibt.
- Die $t_{\max}$-Zeit, bei der die maximale Überschwingweite erreicht wird.
- Die bleibende Regelabweichung $e(\infty)$ zwischen Sollwert und Regelabweichung für $t \to \infty$.

Die Größen $e_{\max}$ und t_ε machen dabei eine Aussage über die Dämpfung des Regelkreises, während T_a und $t_{\max}$ die Geschwindigkeit des Regelvorganges beschreiben. Die bleibende Regelabweichung $e(\infty)$ schließlich beschreibt das statische Verhalten des geschlossenen Regelkreises. [22, 39]

Die Zielsetzung des Reglerentwurfes besteht nun darin, einen für eine gegebene, zuvor identifizierte, Regelstrecke passenden Regler zu finden, der dafür sorgt, dass der geschlossene Regelkreis die oben beschriebenen Anforderungen an Stabilität, Robustheit und Dynamik erfüllt. Da die



(a) Gütemaße der Führübergangsfunktion.



(b) Gütemaße der Störübergangsfunktion.

Abbildung 4.2: Gütemaße der Stör- und Führübergangsfunktionen des geschlossenen Regelkreises nach [39].

genannten Gütemaße Abweichungen vom Idealverhalten des Regelkreises beschreiben, steht beim Reglerentwurf deren Minimierung im Vordergrund. Dass dies keine einfache Aufgabe ist, liegt unter anderem daran, dass die Güteanforderungen untereinander konkurrieren. Eine hohe Dynamik der Regelung wird in der Regel zu einer Vergrößerung der Überschwingweite und damit zu einer Verringerung der Stabilität führen. Eine Minimierung der Güteparameter führt zudem zu großen Werten der Stellgröße, die das Stellglied unter Umständen gar nicht mehr umsetzen kann. In diesem Fall müssen die Güteanforderungen an den Regelkreis weniger streng angesetzt werden. [39]

Für den Reglerentwurf existieren in der Literatur zahlreiche Entwurfsverfahren. Eine kleine Auswahl hiervon soll in Abschnitt 4.1.4 präsentiert werden. Zunächst jedoch werden im nächsten Abschnitt Übertragungsglieder beschrieben, die zur Realisierung des Reglers verwendet werden können.

4.1.2 PID-Regler und aus ihm ableitbare Reglertypen

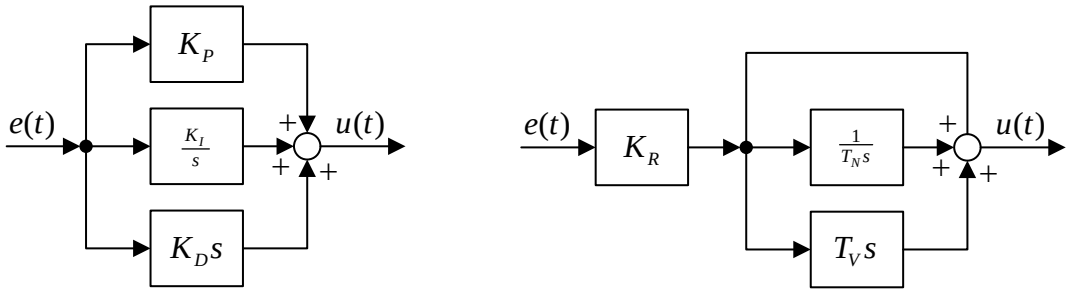
Die in der Praxis am weitesten verbreiteten Regler sind der PID-Regler und die aus ihm ableitbaren P-, I-, PI- und PD-Regler. Der PID-Regler setzt sich als Parallelschaltung der drei elementaren Glieder, einem P-, einem I- und einem D-Glied (siehe Anhang A.1) zusammen. Entsprechend ergibt sich für die in Abb. 4.3a gezeigte, parallele Struktur des PID-Reglers die Übertragungsfunktion

$$G_R(s) = \frac{U(s)}{E(s)} = K_P + \frac{K_I}{s} + K_D s. \tag{4.1}$$

Es ist ebenfalls möglich, die Konstante K_R auszuklammern. Dadurch ergibt sich die in Abb. 4.3b dargestellte serielle Struktur des PID-Reglers, für deren Übertragungsfunktion

$$G_R(s) = K_R \left(1 + \frac{1}{T_N s} + T_V s\right) \tag{4.2}$$

gilt. Dabei ist $K_R = K_P$ der Verstärkungsfaktor, $T_N = \frac{K_P}{K_I}$ die Nachstellzeit und $T_V = \frac{K_D}{K_P}$ die Vorhaltezeit. Die drei Konstanten sind dabei jeweils beliebig wählbar und müssen zur Optimierung des Reglerverhaltens angepasst werden. Verfahren hierfür werden später erläutert.



(a) Parallele Darstellung. **(b)** Serielle Darstellung.

Abbildung 4.3: Blockschaltbild des PID-Reglers nach [39].

Durch inverse Laplace-Transformation kann die Übertragungsfunktion des PID-Reglers in die Differentialgleichung im Zeitbereich

$$u(t) = K_R\, e(t) + \frac{K_R}{T_N} \int_0^t e(\tau)\, \mathrm{d}\tau + K_R T_V \frac{de(t)}{dt} \tag{4.3}$$

transformiert werden. Hieraus kann anschließend die Führübergangsfunktion berechnet werden, die in Abb. 4.4a schematisch dargestellt ist. Die charakteristischen Eigenschaften aller drei elementaren Glieder sind dort wiederzufinden. Der Anfangswert der Regelgröße zum Zeitpunkt $t = 0$ wird durch den Verstärkungsfaktor K_R, das heißt den P-Anteil, festgelegt. Die Höhe des Impulses wird maßgeblich durch den D-Anteil des Reglers beeinflusst und die Steigung der Übergangsfunktion für $t \to \infty$ wird durch den I-Anteil bestimmt.

Da der reine D-Anteil das Kausalitätsprinzip verletzt, ist es nicht möglich, diesen in die Realität umzusetzen. Stattdessen wird der PID-Regler durch einen PIDT_1-Regler angenähert, bei dem das

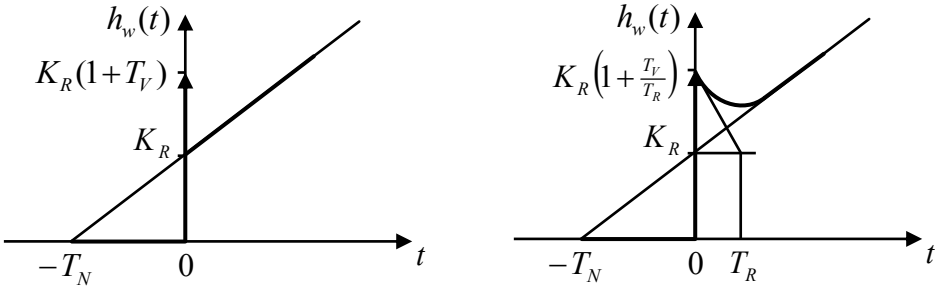
D-Glied durch ein zeitverzögertes DT_1-Glied ersetzt wird. Entsprechend gilt für den $PIDT_1$-Regler die Übertragungsfunktion

$$G_R(s) = K_P + \frac{K_I}{s} + K_D \frac{T_R s}{1 + T_R s} \tag{4.4}$$

mit der Realisierungszeitkonstante $T_R \ll 1$. In der Schreibweise mit Zeitkonstanten ergibt sich alternativ

$$G_R(s) = K_R \left(1 + \frac{1}{T_N s} + T_V \frac{s}{1 + T_R s}\right). \tag{4.5}$$

Hierbei gelten $K_R = K_P$, $T_N = \frac{K_R}{KI}$ sowie $T_V = \frac{K_D T_R}{K_R}$. Die Zeitverzögerung macht sich dabei in der Führübergangsfunktion des realen PID-Glieds, wie in Abb. 4.4b gezeigt, durch ein langsames Abklingen des Impulses bei $t = 0$ bemerkbar. [22, 39]



(a) Führübergangsfunktion des idealen PID-Glieds.

(b) Führübergangsfunktion des realen PID-Glieds.

Abbildung 4.4: Führübergangsfunktion des idealen und des realen PID-Glieds nach [39].

Durch Vernachlässigen der jeweiligen Terme in der Übertragungsfunktion des PID-Glieds können P-, I-, D-, PI- sowie PD-Regler abgeleitet werden. Deren Eigenschaften sind im Anhang A.1 ausführlich dargestellt, daher soll an dieser Stelle nicht näher darauf eingegangen werden. Stattdessen werden die Unterschiede zwischen den einzelnen Reglertypen herausgearbeitet. Hierzu wird die Regelstrecke mit der Übertragungsfunktion

$$G_S(s) = \frac{1}{(1 + sT)^3} \tag{4.6}$$

in dem einschleifigen Regelkreis aus Abb. 2.5 mit Messglied-Übertragungsfunktion $G_M(s) = 1$ und verschiedenen Reglertypen in Simulink simuliert. Die Führübergangsfunktion dieses Regelkreises ist in Abb. 4.5 für die unterschiedlichen Regler dargestellt. Es ist zu erkennen, dass nur die Regler mit einem I-Anteil statisch genau arbeiten, das heißt eine verschwindende bleibende Regelabweichung $e(\infty)$ besitzen. Der P- und PD-Regler hingegen führen zu einer starken Regelabweichung, wobei der PD-Regler ein etwas besseres Verhalten zeigt, da durch den D-Anteil der Verstärkungsfaktor größer gewählt werden kann und somit die bleibende Regelabweichung kleiner ausfällt. Der reine I-Regler hat das Problem, dass er zwar statisch genau arbeitet, aber eine sehr schlechte Dynamik besitzt. Das bedeutet, der Ausregelvorgang dauert sehr lange. Durch die Kombination des I-Glieds mit einem P-Glied wird dieses Problem vermieden. Der P-Anteil sorgt für eine schnelle Reaktionszeit, während der I-Anteil die statische Führungsgenauigkeit

sicherstellt. Den Optimalfall stellt für die gegebene Strecke der PID-Regler dar, da er ebenfalls statisch genau arbeitet, aber durch den zusätzlichen D-Anteil eine schnellere Reaktionszeit als der PI-Regler besitzt. [39]

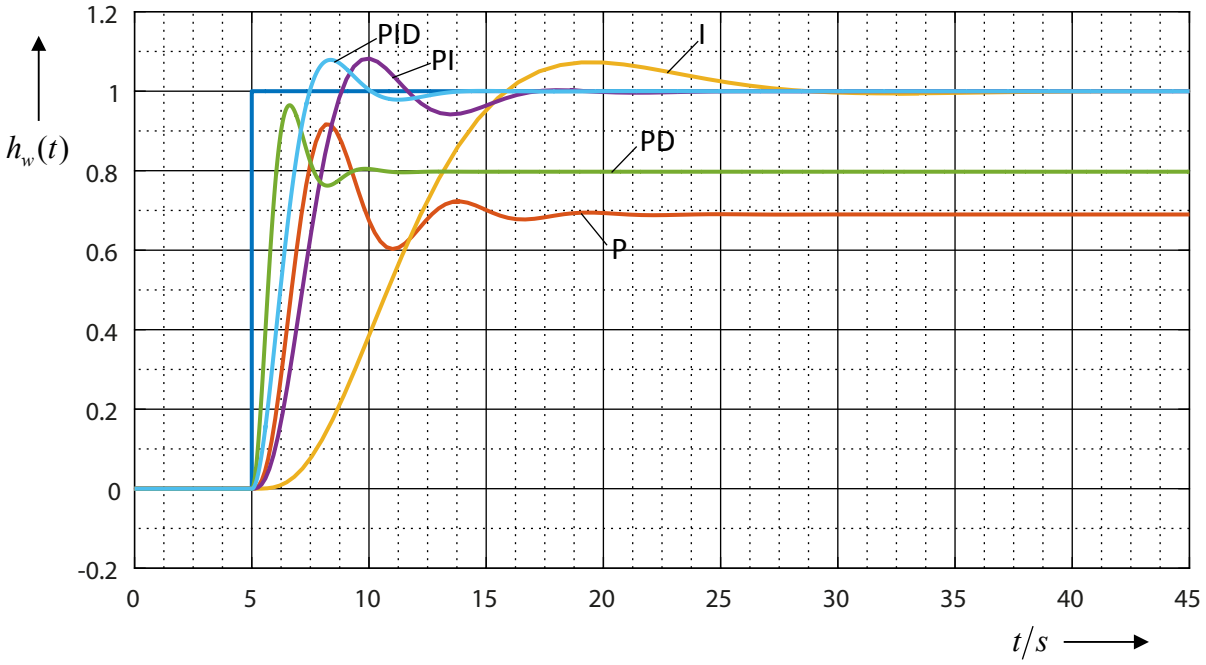


Abbildung 4.5: Übergangsfunktion einer mit verschiedenen Reglertypen geregelten PT_3-Strecke.

4.1.3 Diskretisierung des PID-Reglers

Neben der in Abschnitt 2.9.5 beschriebenen Vorgehensweise zum Entwurf eines diskreten Regelalgorithmus besteht die Möglichkeit, die in Abschnitt 4.1.2 vorgestellten Standard-Regelglieder in diskreter Form anzugeben und darauf aufbauend einen Algorithmus zur Berechnung der zeitdiskreten Stellgröße $u(k)$ aus einer gegebenen zeitdiskreten Regelabweichung $e(k)$ aufzustellen.

Das P-Glied kann durch einfaches Ersetzen der Regelabweichung $e(t)$ durch die zeitdiskrete Regelabweichung $e(k)$ diskretisiert werden. Es gilt folglich

$$u(k) = K_R e(k) = K_R \left(w(k) - y(k)\right) \tag{4.7}$$

als Differenzengleichung für den P-Regler.

Das D-Glied des Reglers, das durch die Differentialgleichung

$$u(t) = K_D \frac{de(t)}{dt} = K_R T_V \frac{de(t)}{dt} \tag{4.8}$$

beschrieben wird, kann durch Ersetzen des Differentialquotienten durch einen Differenzenquotienten in eine zeitdiskrete Form überführt werden. Es ergibt sich so die Differenzengleichung

$$u(k) = K_D \frac{e(k) - e(k-1)}{T} = K_R T_V \frac{e(k) - e(k-1)}{T}. \tag{4.9}$$

Beim I-Glied, das in seiner zeitkontinuierlichen Form bekanntermaßen durch die Gleichung

$$u(t) = K_I \int_0^t e(\tau)\,\mathrm{d}\tau = \frac{K_R}{T_N} \int_0^t e(\tau)\,\mathrm{d}\tau \tag{4.10}$$

beschrieben wird, muss zur Diskretisierung das Integral über die Zeit durch eine Summe über alle vergangenen Regelabweichungen $e(i)$ mit $i \leq k$ ersetzt werden. Auf diese Weise ergibt sich die Differenzengleichung des zeitdiskreten I-Glieds

$$u(k) = K_I \sum_{i=0}^{k} e(i)T = \frac{K_R}{T_N} \sum_{i=0}^{k} e(i)T. \tag{4.11}$$

Eine genauere Näherung des Integrals liefert die Trapezregel. Hierbei wird das Integral nicht wie oben durch eine Treppenfunktion angenähert, sondern durch Trapeze. Diese approximieren die Fläche unter der Regelabweichung mit geringerem Fehler. Abb. 4.6 macht diesen Sachverhalt deutlich, wobei die gestrichelte Kurve die Trapeznäherung darstellt. Das Reglergesetz lautet in diesem Fall

$$u(k) = K_I \left(\frac{e(0)}{2} + \sum_{i=1}^{k-1} e(i)T + \frac{e(k)}{2} \right). \tag{4.12}$$

Wird die Abtastzeit T jedoch klein genug gewählt, unterscheiden sich die beiden Approximationen kaum voneinander, sodass im Folgenden der Einfachheit halber die Approximation durch die Treppenfunktion aus Gleichung (4.11) erfolgen soll.

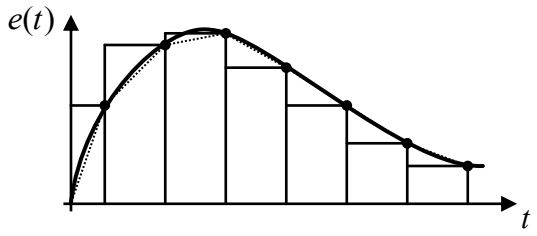


Abbildung 4.6: Annäherung der Fläche unterhalb der Regelabweichung durch eine Treppenkurve und durch Trapeze.

Auf Basis der Reglergesetze für die einzelnen Regelglieder lässt sich nun das Reglergesetz des zeitdiskreten PID-Reglers durch Summation von P-, I- und D-Glied angeben, sodass in der parallelen Form (Abb. 4.3a) das Reglergesetz

$$u(k) = K_R e(k) + K_I \sum_{i=0}^{k} e(i)T + K_D \frac{e(k) - e(k-1)}{T} \tag{4.13}$$

lautet. In der seriellen Form (Abb. 4.3b) gilt entsprechend

$$u(k) = K_R \left(e(k) + \frac{1}{T_N} \sum_{i=0}^{k} e(i)T + T_V \frac{e(k) - e(k-1)}{T} \right). \tag{4.14}$$

Ein Problem dieses Reglergesetzes ist die Tatsache, dass die Stellgröße $u(t)$ bei sprungartigen Änderungen der Führungsgröße $w(t)$, wie in Abb. 4.7 dargestellt, impulsartig sehr große Werte annimmt. Dies ist in der Theorie kein Problem, praktisch kann dies jedoch zur Übersteuerung des Stellgliedes führen. Als Lösung für dieses Problem wird in [19] vorgeschlagen, den D-Term nicht auf die Regelabweichung $e(t)$, sondern stattdessen auf die gemessene Regelgröße $y(t)$ zu beziehen. Mit der Definition der Regelabweichung lässt sich das Differential umschreiben zu

$$\frac{de(t)}{dt} = \frac{d\left(w(t) - y(t)\right)}{dt}. \tag{4.15}$$

Nimmt man eine zeitlich konstante Führungsgröße $w(t)$ an, folgt daraus

$$\frac{de(t)}{dt} = -\frac{dy(t)}{dt}. \tag{4.16}$$

Es muss also in Gleichung (4.13) der Differenzenquotient der Regelabweichung durch den negativen Differenzenquotienten der gemessenen Regelgröße ersetzt werden. Es folgt somit das angepasste Reglergesetz

$$u(k) = K_R e(k) + K_I \sum_{i=0}^{k} e(i)T - K_D \frac{y(k) - y(k-1)}{T}. \tag{4.17}$$

Bei einem auf diese Weise umgesetzten Regler führen sprunghafte Änderungen der Führungsgröße nicht mehr zu impulsartigen Anstiegen der Stellgröße.

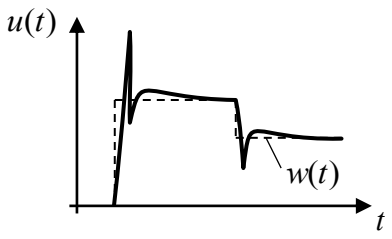


Abbildung 4.7: Impulsartiges Ansteigen der Stellgröße $u(t)$ bei Sprüngen der Führungsgröße $w(t)$ bedingt durch D-Anteil.

Wird die Abtastzeit T kleiner als 1/10 der dominierenden Zeitkonstante des Systems gewählt, verhält sich der Regelkreis annähernd wie ein kontinuierliches System. Entsprechend ist es, wie schon in Abschnitt 2.9.5 erwähnt, möglich, zunächst einen kontinuierlichen Regler mit den Methoden aus Abschnitt 4.1.4 zu entwerfen und die erhaltenen Parameter ohne Anpassung in Gleichung (4.13) einzusetzen. Der zeitdiskrete Regler wird sich dann ähnlich wie der entworfene kontinuierliche Regler verhalten. [19, 24, 35, 37]

Im späteren Verlauf dieser Arbeit soll ein Algorithmus zur Realisierung des Reglergesetzes aus Gleichung (4.17) erarbeitet und der Regler auf einem Mikrocontroller implementiert werden. Dazu sind weitere Anpassungen des Reglergesetzes erforderlich, auf die im Einzelnen später eingegangen werden soll.

4.1.4 Überblick über Verfahren zum Reglerentwurf

Im folgenden Unterkapitel sollen die wichtigsten Verfahren zur Einstellung und Optimierung von Reglern präsentiert werden. Dabei werden zunächst die empirischen Einstellregeln nach Ziegler und Nichols sowie Chien, Hrones und Reswick vorgestellt. Sie eignen sich für eine schnelle und unkomplizierte Inbetriebnahme der Regelung, führen aber in der Regel nicht zu einem optimal eingestellten Regler. Eine Optimierung des Reglers kann daher anhand der Regelfläche erfolgen. Weitere im nachfolgenden Abschnitt vorgestellte Verfahren sind das Frequenzkennlinien-Verfahren, das auf der Verformung des Frequenzgangs des offenen Regelkreises im Bode-Diagramm beruht, und das Verfahren der Polkompensation, bei dem durch geeignete Wahl der Regler-Übertragungsfunktion dominierende Zeitkonstanten aus der Streckenübertragungsfunktion herausgekürzt werden und somit das Verhalten des Regelkreises verbessert wird. Danach wird kurz auf das Wurzelortskurven-Verfahren eingegangen, bei dem der Regler durch gezielte Platzierung der dominierenden Polstellen in der komplexen s-Ebene optimiert wird. Abschließend wird mit dem Verfahren nach Truxal-Guillemin ein analytisches Entwurfsverfahren für Regler beschrieben. In der Literatur existieren zahlreiche weitere Verfahren, auf die jedoch im Rahmen dieser kurzen Einführung nicht eingegangen werden kann.

4.1.4.1 Einstellregeln nach Ziegler und Nichols

Das empirische Einstellverfahren nach Ziegler und Nichols dient dazu, für bestimmte Regelstrecken die erforderlichen Reglerparamter eines PID-Reglers festzulegen. Das Verfahren eignet sich dabei für Regelstrecken, die durch ein $\mathrm{PT_1T_t}$-Glied (siehe Tabelle A.6) mit der Übertragungsfunktion

$$G_S(s) = \frac{K_S}{1 + T_1 s}\,\mathrm{e}^{-T_t s} \tag{4.18}$$

mit Zeitkonstante T_1, Totzeit T_t und Verstärkungsfaktor K_S approximiert werden können. Dies ist für alle Regelstrecken, die $\mathrm{PT_n}$- oder $\mathrm{IT_n}$-Verhalten aufweisen, möglich.

Reglereinstellwerte können nun durch zwei Verfahren ermittelt werden:

1) *Methode des Stabilitätsrandes:* Die Regelstrecke wird mithilfe eines P-Reglers mit Verstärkungsfaktor K_R geregelt. Der Verstärkungsfaktor wird nun solange erhöht, bis das geregelte System Dauerschwingungen vollführt. Der dabei verwendete Verstärkungsfaktor wird als $K_{R,krit}$ bezeichnet und die Periodendauer der Dauerschwingung als T_{krit}. Nun können mithilfe dieser beiden Werte aus Tabelle 4.1 für einen P-, PI- oder PID-Regler die erforderlichen Reglerparameter ermittelt werden.

2) *Methode der Übergangsfunktion:* Ist es nicht möglich, das System am Stabilitätsrand zu betreiben, können die Reglerparameter auch aus der Führübergangsfunktion des Systems bestimmt werden. Hierzu wird, wie in Abb. 4.8 gezeigt, die Wendetangete an die Führübergangsfunktion der Regelstrecke gelegt und die Verzugszeit T_u und Anstiegszeit T_a

Tabelle 4.1: Reglereinstellwerte nach Ziegler und Nichols für die Methode des Stabilitätsrandes.

Reglertyp	Reglereinstellwerte		
	K_R	T_N	T_V
P	$0{,}5K_{R,krit}$	-	-
PI	$0{,}45K_{R,krit}$	$0{,}85T_{krit}$	-
PID	$0{,}6K_{R,krit}$	$0{,}5T_{krit}$	$0{,}12T_{krit}$

abgelesen. Außerdem kann aus der Übergangsfunktion der Verstärkungsfaktor der Strecke K_S bestimmt werden. Mithilfe dieser drei Werte wird die Hilfsgröße

$$\lambda = \frac{T_a}{T_u K_S} \tag{4.19}$$

berechnet, mit der nun aus Tabelle 4.2 Reglerparameter für die verschiedenen Reglertypen abgelesen werden können. [39]

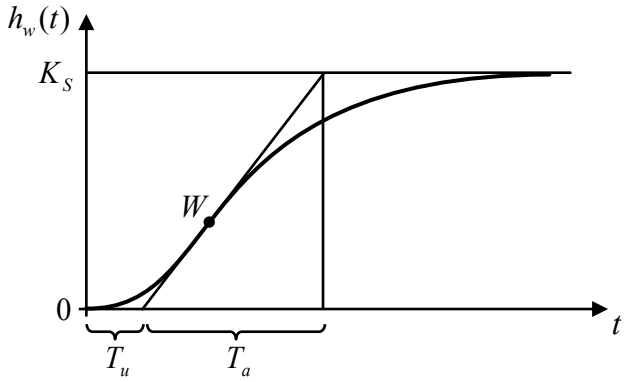


Abbildung 4.8: Führübergangsfunktion eines PT_nT_t-Glieds zur Bestimmung des Verstärkungsfaktors K_S, der Verzugszeit T_u und der Anstiegszeit T_a nach [39].

Tabelle 4.2: Reglereinstellwerte nach Ziegler und Nichols für die Methode der Übergangsfunktion.

Reglertyp	Reglereinstellwerte		
	K_R	T_N	T_V
P	λ	-	-
PI	$0{,}9\lambda$	$3{,}33T_u$	-
PID	$1{,}2\lambda$	$2T_u$	$0{,}5T_u$

4.1.4.2 Einstellregeln nach Chien, Hrones und Reswick

Ein weiteres empirisches Einstellverfahren ist das nach Chien, Hrones und Reswick. Diesem liegt ebenfalls eine durch ein PT_1T_t-Glied approximierte Regelstrecke zugrunde, aus deren Führübergangsfunktion mithilfe des Wendetangetenverfahrens, wie in Abb. 4.8 dargestellt, der Verstärkungsfaktor K_S, die Verzugszeit T_u und die Anstiegszeit T_a ermittelt werden. Aus diesen drei Werten wird die Hilfsgröße λ entsprechend Gleichung (4.19) berechnet und die für den gewählten Reglertyp erforderlichen Einstellwerte aus Tabelle 4.3 entnommen. Dabei unterscheidet dieses Verfahren, anders als das nach Ziegler und Nichols, ob eine Festwert- oder eine Folgeregelung realisiert werden soll und ob ein Überschwingen zulässig ist oder nicht.

Tabelle 4.3: Reglereinstellwerte nach Chien, Hrones und Reswick.

Reglertyp		aperiodischer Einschwingvorgang		20% Überschwingen	
		Führung	Störung	Führung	Störung
P	K_R	$0{,}3\lambda$	$0{,}3\lambda$	$0{,}7\lambda$	$0{,}7\lambda$
PI	K_R	$0{,}35\lambda$	$0{,}6\lambda$	$0{,}6\lambda$	$0{,}7\lambda$
	T_N	$1{,}17T_a$	$4T_u$	T_a	$2{,}33T_u$
PID	K_R	$0{,}6\lambda$	$0{,}95\lambda$	$0{,}95\lambda$	$1{,}2\lambda$
	T_N	T_a	$2{,}38T_u$	$1{,}36T_a$	$2T_u$
	T_V	$0{,}5T_u$	$0{,}42T_u$	$0{,}47T_u$	$0{,}42T_u$

Der Qoutient T_a/T_u wird als Regelbarkeit der Strecke bezeichnet. Je größer dieser Wert ist, desto besser lässt sich die Strecke regeln. Gilt $T_a/T_u < 3$, so ist die Strecke nur noch schlecht regelbar und es sollte ein anderes Entwurfsverfahren für den Regler verwendet werden. [21, 41]

4.1.4.3 Optimierung der Regelfläche mittels Integralkriterien

Eine weitere Möglichkeit der Einstellung eines Reglers besteht in der Optimierung der in Abschnitt 4.1.1 beschriebenen Gütemaße der Übergangsfunktion des geschlossenen Regelkreises. Wie dort bereits erwähnt, ist es schwierig, sämtliche Gütemaße zeitgleich zu optimieren, da diese teils untereinander konkurrieren. Daher gibt es mit den Integralkriterien ein Werkzeug, das die Güte einer Regelung durch eine einzige Größe, die Regelfläche I_k, beschreibt. Wie Abb. 4.9 verdeutlich ist die Regelfläche ein Maß für die Abweichung der Regelgröße $y(t)$ von der Führungsgröße $w(t)$. Folglich gilt ganz allgemein für die Regelfläche

$$I_k = \int_0^\infty f_k \left\{ w(t) - y(t) \right\} \, \mathrm{d}t = \int_0^\infty f_k \left\{ e(t) \right\} \, \mathrm{d}t, \tag{4.20}$$

wobei die Funktion f_k entsprechend Tabelle 4.4 gewählt werden kann, um unterschiedliche Aspekte der Regelabweichung zu gewichten. Je besser der Regler eingestellt ist, desto kleiner

wird die Regelfläche. Umgekehrt ist es möglich, bei bekannter Struktur des Reglers dessen freie Parameter so zu wählen, dass die durch das Integralkriterium beschriebene Regelfläche minimal wird. Die Reglereinstellung kann also in ein mathematisches Optimierungsproblem überführt werden. [24, 39]

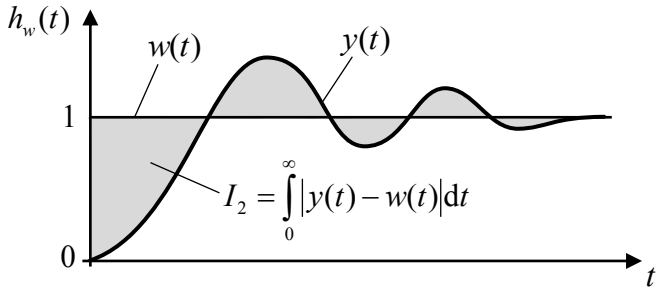


Abbildung 4.9: Die Regelfläche beschreibt die Abweichung der Regelgröße $y(t)$ von der Führungsgröße $w(t)$ und ist ein Maß für die Güte einer Regelung. Beispielhaft dargestellt ist die betragslineare Regelfläche.

Tabelle 4.4: Die wichtigsten Integralkriterien nach [39].

Kriterium	Eigenschaften
$I_1 = \int_0^\infty e(t)\,\mathrm{d}t$	*Lineare Regelfläche:* Geeignet für stark gedämpft oder monoton verlaufende Übergangsfunktionen.
$I_2 = \int_0^\infty \lvert e(t)\rvert\,\mathrm{d}t$	*Betragslineare Regelfläche:* Geeignet für Übergangsfunktion mit Überschwingern.
$I_3 = \int_0^\infty e^2(t)\,\mathrm{d}t$	*Quadratische Regelfläche:* Gewichtet große Abweichungen vom Sollwert stärker als kleine Abweichungen.
$I_4 = \int_0^\infty \lvert e(t)\rvert\, t\,\mathrm{d}t$	*Zeitbeschwerte betragslineare Regelfläche:* Gewichtet Abweichungen vom Sollwert umso stärker, je später sie auftreten.
$I_5 = \int_0^\infty e^2(t) t\,\mathrm{d}t$	*Zeitbeschwerte quadratische Regelfläche:* Gewichtet später auftretende und größere Abweichungen gleichermaßen stärker als früh auftretende, geringe Abweichungen.

4.1.4.4 Frequenzkennlinien-Verfahren

Ein sehr mächtiges Werkzeug zum Reglerentwurf basiert auf der Darstellung und Verformung des Bode-Diagramms des offenen Regelkreises $G_0(i\omega)$. Hierbei wird zunächst die erforderliche Güte der Regelung durch Vorgabe der Überschwingweite $e_{\max}$ und der Anstiegszeit T_a spezifiziert. Diese beiden Werte müssen nun in Anforderungen an die Amplitudenreserve α_R und die Durchtrittsfrequenz ω_d (siehe Abschnitt 2.8.4) umgeformt werden. Nach [33] gelten dabei folgende Näherungsbeziehungen:

$$\alpha_R \approx \begin{cases} 69° - 106° \cdot e_{\max} & \text{für} \quad 0{,}05 \leq e_{\max} < 0{,}25 \\ 62° - 76° \cdot e_{\max} & \text{für} \quad 0{,}25 \leq e_{\max} < 0{,}45 \end{cases} \tag{4.21}$$

und

$$\omega_d \approx \frac{1{,}44}{T_a}. \tag{4.22}$$

Voraussetzung für die Anwendbarkeit dieser Umrechnungsbeziehungen ist ein dominantes PT_2-Verhalten des geschlossenen Regelkreises. Dieses liegt vor, wenn der Amplitudengang des offenen Regelkreises in der Nähe der Durchtrittsfrequenz ω_d mit etwa 20 dB bis 40 dB pro Dekade abfällt und die Phasenreserve an dieser Stelle zwischen 30° und 70° liegt.

Nachdem die Anforderungen an die Regelgüte des geschlossenen Regelkreises in Anforderungen an ω_d und α_R übersetzt wurden, wird das Bode-Diagramm des offenen Regelkreises ohne Regler gezeichnet und dort die berechnete Durchtrittsfrequenz ω_d sowie Phasenreserve α_R eingetragen. Ziel des Reglerentwurfes wird es sein, das Bode-Diagramm des offenen Kreises mit Regler so zu verformen, dass die Durchtrittsfrequenz und die Phasenreserve den zuvor ermittelten und eingetragenen Sollwerten entsprechen. Hierzu müssen ein zur Regelstrecke passender Regler ausgewählt und dessen Parameter geeignet variiert werden. Je nach Reglerwahl ist es nicht möglich, die Forderungen an ω_d und α_R gleichzeitig umzusetzen. In diesem Fall kann nur eine der beiden Größen optimiert werden.

Als Beispiel soll ein P-Regler mit $G_R(s) = K_P$ zur Regelung des Systems mit dem in Abb. 4.10 dargestellten Amplitudengang des offenen Regelkreises ausgelegt werden. Die maximale Überschwingweite $e_{\max}$ soll dabei 20 % betragen, woraus sich entsprechend Gleichung (4.21) eine erforderliche Phasenreserve von $\alpha_R = 48°$ ergibt. Zur Optimierung des Reglers muss der Verstärkungsfaktor K_P so eingestellt werden, dass bei derjenigen Frequenz ω_d, bei der sich die Phasenreserve $\alpha_R = 48°$ einstellt, der Amplitudengang genau die 0 dB-Linie schneidet. Das heißt durch Anpassung der Reglerverstärkung wird der Amplitudengang so verschoben, dass sich die gewünschte Phasenreserve einstellt. In diesem Fall müsste der Amplitudengang also um 14 dB abgesenkt werden, da sich eine Phasenreserve von 48° bei der Durchtrittsfrequenz $\omega_d = 0{,}19°$ einstellt. Das entspricht einer Reglerverstärkung von

$$K_P = 10^{\frac{-14\text{dB}}{20\text{dB}}} = 0{,}2. \tag{4.23}$$

Ein gleichzeitiges Einstellen der gewünschten Phasenreserve und Durchtrittsfrequenz ist bei einem P-Regler nicht möglich. Folglich wird sich in der Führübergangsfunktion des geschlossenen Regelkreises auch nur entweder die gewünschte Überschwingweite $e_{\max}$ oder die spezifizierte Anstiegszeit T_a einstellen. [33]

Zur Optimierung weiterer Reglertypen nach diesem Verfahren sei auf die weiterführende Literatur [33] verwiesen, da eine Erläuterung an dieser Stelle zu weit führt.

4.1.4.5 Verfahren der Polkompensation

Liegt die Übertragungsfunktion der Regelstrecke vor, kann diese in die sogenannte *V-Normalform*

$$G_S(s) = \frac{K_S}{(1 + T_1 s)(1 + T_2 s) \dots (1 + T_n s)} \tag{4.24}$$

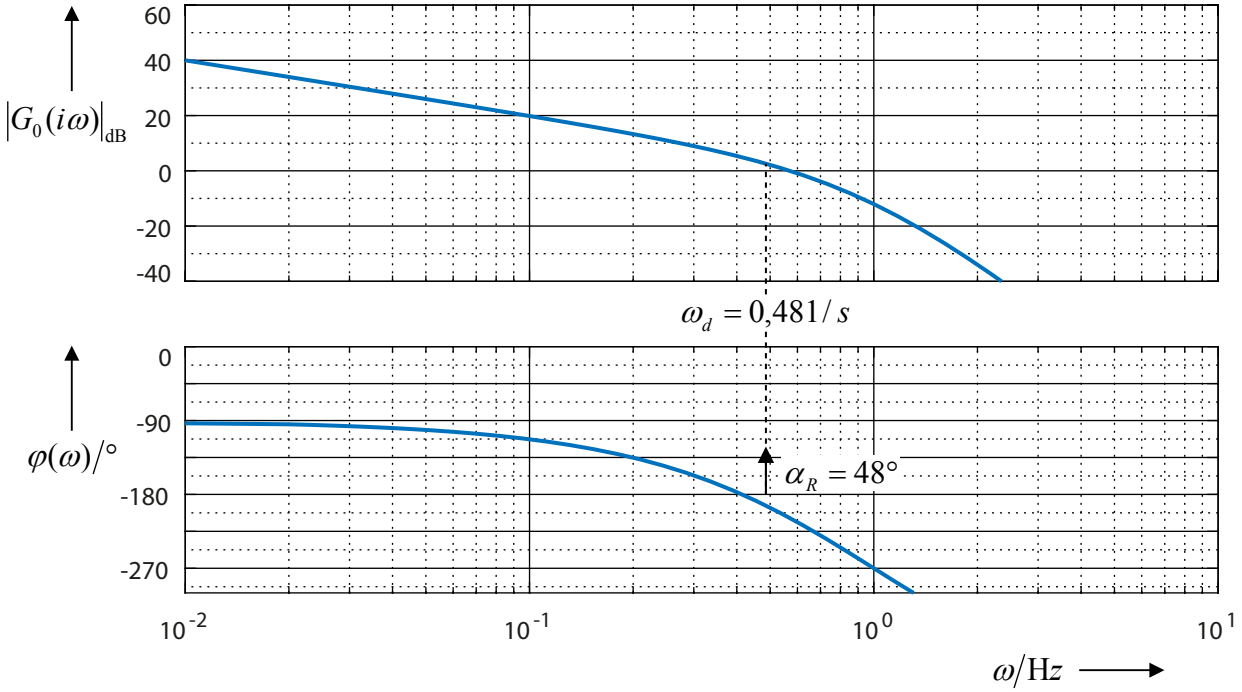


Abbildung 4.10: Bode-Diagramm des offenen Regelkreises ohne Regler mit Sollwerten für die Durchtrittsfrequenz ω_d und Phasenreserve α_R. [33]

mit den Zeitkonstanten $T_1 \geq T_2 \geq \ldots \geq T_n$ und der Streckenverstärkung K_S umgeformt werden. Je größer die Zeitkonstanten sind, desto eher knickt das Bode-Diagramm des offenen Regelkreises nach unten ab, was in einer kleinen Durchtrittsfrequenz ω_d und damit gemäß Gleichung (4.22) in einer hohen Anstiegszeit der Übergangsfunktion, das heißt einer geringen Dynamik des geschlossenen Regelkreises, resultiert. Im Sinne einer hohen Dynamik der Regelung ist folglich eine hohe Durchtrittsfrequenz wünschenswert. Diese kann durch eine Kompensation der dominierenden Zeitkonstanten eingestellt werden. Zur Kompensation wählt man die Vorhaltezeitkonstanten T_1 und T_2 in den Reglerübertragungsfunktionen von PI- Regler

$$G_R(s) = \frac{V(1+sT_1)}{s}, \tag{4.25}$$

PD-Regler

$$G_R(s) = \frac{V(1+sT_1)}{1+sT_R}, \tag{4.26}$$

oder PID-Regler

$$G_R(s) = \frac{V(1+sT_1)(1+sT_2)}{1+sT_R} \tag{4.27}$$

so, dass sich die Terme mit den dominierenden Zeitkonstanten in der Übertragungsfunktion des offenen Regelkreises $G_0(s) = G_R(s)G_S(s)$ herauskürzen.

Hat die Regelstrecke beispielsweise die Übertragungsfunktion

$$G_S(s) = \frac{5}{(1+10s)(1+5s)(1+0{,}5s)} \tag{4.28}$$

und soll mit einem PID-Regler geregelt werden, so müssen dessen Zeitkonstanten zu $T_1 = 10$, $T_2 = 5$ und $T_R = \min\{T_1, T_2\} = 5$ gewählt werden, um die Durchtrittsfrequenz ω_d zu höheren Frequenzen hin zu verschieben und damit die Dynamik des geschlossenen Regelkreises zu verbessern.

Anschließend muss noch der Verstärkungsfaktor V des Reglers bestimmt werden. Hierzu wird, wie bereits in Abschnitt 4.1.4.4 erläutert, zunächst das Bode-Diagramm des offenen Regelkreises ohne Regler gezeichnet. In dieses Diagramm wird nun die Phasenreserve α_R, die aus der gewünschten Überschwingweite mithilfe von Gleichung (4.21) ermittelt wurde, eingetragen. Jetzt kann, wie im vorherigen Abschnitt erläutert, durch Wahl des Verstärkungsfaktors V_{dB} der Amplitudengang so verschoben werden, dass die Duchtrittsfrequenz ω_d an derjenigen Stelle entsteht, an der die Phasenreserve α_R den eingetragenen Wert annimmt. Gemäß der Beziehung

$$V = 10^{\frac{V_{\mathrm{dB}}}{20\mathrm{dB}}} \tag{4.29}$$

kann schließlich der Verstärkungsfaktor V des Reglers bestimmt werden. Damit ist der Regler vollständig optimiert.

Enthält die Regelstrecke anstelle einiger dominierender Zeitkonstanten viele kleine oder annähernd gleiche Zeitkonstanten, wie z.B.

$$G_S(s) = \frac{5}{(1 + 0{,}5s)(1 + 0{,}8s)(1 + 0{,}3s)} \tag{4.30}$$

kann eine *Summenzeitkonstante*

$$(1 + 0{,}5s)(1 + 0{,}8s)(1 + 0{,}3s) \approx (1 + 1{,}6s) \tag{4.31}$$

gebildet werden, die anschließend mittels Polkompensation eliminiert wird. [33]

4.1.4.6 Wurzelortskurven-Verfahren

Das Wurzelortskurven-Verfahren basiert auf der Übersetzung von Forderungen an die Güte der Regelung in Form von Überschwingweite $e_{\max}$ und Anstiegszeit T_a der Führübergangsfunktion in Forderungen an die Lage der Pole des geschlossenen Regelkreises in der komplexen s-Ebene. Da in der Regel ein dominierendes PT_2-Verhalten des geschlossenen Regelkreises gewünscht ist, wird das Aussehen der Übergangsfunktion im Wesentlichen durch ein konjugiert komplexes Polpaar in der linken s-Halbebene, das sogenannte *dominierende Polpaar*, bestimmt. Wie in [33] genauer erläutert, hängt dabei die Überschwingweite der Übergangsfunktion vom Abstand des dominierenden Polpaars zur Imaginärachse und die Anstiegszeit vom Abstand der Pole zur reellen Achse ab. Liegt das Polpaar nahe an der Imaginärachse, wird die Überschwingweite groß. Eine große Nähe zur reellen Achse führt entsprechend zu einer größeren Anstiegszeit.

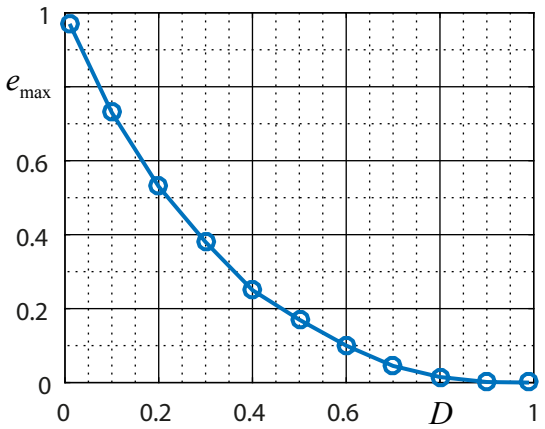
Zur Übersetzung der Anforderungen an die Überschwingweite $e_{\max}$ und Anstiegszeit T_a in die Lage des dominierenden Polpaares kann man die grafischen Beziehungen aus Abb. 4.11a und

Abb. 4.11b verwenden. Aus diesen können das Dämpfungsmaß D und die Zeitkonstante T_1 der Übertragungsfunktion des PT_2-Glieds

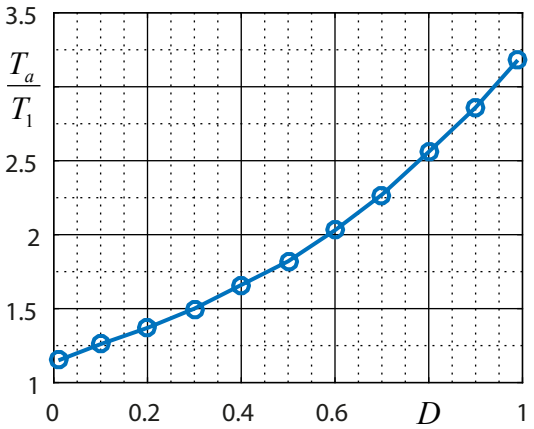
$$G_W(s) = \frac{1}{1 + 2DT_1 s + T_1^2 s^2}, \tag{4.32}$$

dem die spätere Führungsübertragungsfunktion des geschlossenen Regelkreises entsprechen soll, abgelesen werden. Die beiden Werte beschreiben dabei die Lage der Pole des geschlossenen Regelkreises in der komplexen s-Ebene

$$s_{1,2} = -\frac{D}{T_1} \pm i\frac{1}{T_1}\sqrt{1 - D^2}. \tag{4.33}$$



(a) Zusammenhang zwischen Überschwingweite $e_{\max}$ und Dämpfungskonstante D.



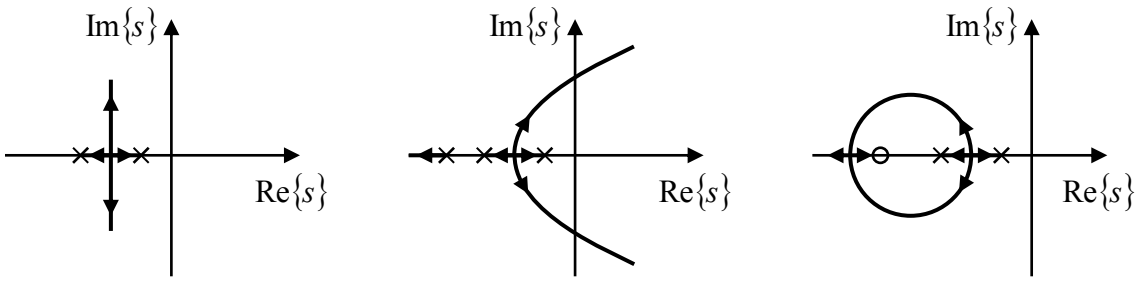
(b) Zusammenhang zwischen T_a/T_1 und Dämpfungskonstante D.

Abbildung 4.11: Zusammenhang zwischen den Gütemaßen $e_{\max}$ und T_a und der die Pollage des geschlossenen Regelkreises beschreibenden Parameter D und T_1 eines PT_2-Glieds nach [33].

Im nächsten Schritt muss nun durch geeignete Wahl des Reglers die gewünschte Lage des dominierenden Polpaars eingestellt werden. Hierzu bedient man sich der *Wurzelortskurve*, die die Lage der Pole des geschlossenen Regelkreises in Abhängigkeit eines freien Parameters K angibt. Diese Lage berechnet sich mit dem Nenner $N_0(s)$ und dem Zähler $Z_0(s)$ der Übertragungsfunktion $G_0(s) = G_R(s)G_S(s)G_M(s)$ des offenen Regelkreises mithilfe der charakteristischen Gleichung

$$N_0(s) + KZ_0(s) = 0 \tag{4.34}$$

durch Einsetzen von $0 \leq K \leq \infty$. Für den Fall, dass obige Beziehung nicht mehr analytisch berechnet werden kann, sind in [39] Regeln für die manuelle Konstruktion einer Wurzelortskurve angegeben. Durch Wahl einer geeigneten Reglerübertragungsfunktion $G_R(s)$ kann nun die Wurzelortskurve in der komplexen s-Ebene beliebig verformt und verschoben werden. Wie in Abb. 4.12 dargestellt, führt das Einfügen eines zusätzlichen Pols in die Übertragungsfunktion des offenen Regelkreises zu einer Verbiegung der Wurzelortskurve nach rechts und das Einfügen einer zusätzlichen Nullstelle zu einer Verbiegung nach links. Auf diese Weise kann die gewünschte Lage des dominierenden Polpaars eingestellt werden, sodass der geschlossene Regelkreis die spezifizierten Güte-Anforderungen erfüllt. [33, 39]



(a) Wurzelortskurve des geschlossenen Regelkreises.

(b) Verformung nach rechts durch zusätzlichen Pol.

(c) Verformung nach links durch zusätzliche Nullstelle.

Abbildung 4.12: Verformung der Wurzelortskurve des geschlossenen Regelkreises durch Einfügen von Pol- und Nullstellen nach [39].

4.1.4.7 Analytische Entwurfsverfahren

Beim analytischen Reglerentwurf wird mithilfe von tabellierten Übergangsfunktionen das Nennerpolynom der Führungsübertragungsfunktion des Regelkreises in der Form

$$G_W(s) = \frac{\beta_0}{\beta_0 + \beta_1 s + \beta_2 s^2 + \ldots + \beta_u s^u} = \frac{Z_W(s)}{N_W(s)} \tag{4.35}$$

bestimmt. Hierzu existieren normierte Übergangsfunktionen für verschiedene Ordnungen u des Nennerpolynoms, aus denen mit der zuvor spezifizierten 50%-Anstiegszeit $t_{a,50\%}$ die Bezugsfrequenz ω_W für eine gewählte Ordnung des Polynoms, wie in Abb. 4.13 gezeigt, auf der Abszisse abgelesen werden kann. In der Literatur [21] sind verschiedene Standard-Formen für die Übergangsfunktion und die zugehörigen Nennerpolynome von $G_W(s)$ tabelliert. So gibt es neben der *Butterworth-Form*, der *Integral-Form* und der *Abklingzeit-Form* die im Folgenden beispielhaft aufgeführte *Binomial-Form*. Zu dieser gehören die normierte Übergangsfunktion in Abb. 4.13 und die nachfolgenden Nennerpolynome $N_W(s)$ der Führungsübertragungsfunktion

$$\begin{aligned} u = 1\text{:}\ & s + \omega_W, \\ u = 2\text{:}\ & s^2 + 2\omega_W s + \omega_W^2, \\ u = 3\text{:}\ & s^3 + 3\omega_W s^2 + 3\omega_W^2 s + \omega_W^3, \\ u = 4\text{:}\ & s^4 + 4\omega_W s^3 + 6\omega_W^2 s^2 + 4\omega_W^3 s + \omega_W^4. \end{aligned} \tag{4.36}$$

Liegt eine Standard-Regelkreisstruktur mit negativer Rückkopplung wie in Abb. 2.5 mit $G_M(s) = 1$ vor, so können die Übertragungsfunktion der Strecke

$$G_S(s) = \frac{d_0 + d_1 s + d_2 s^2 + \ldots + d_m s^m}{c_0 + c_1 s + c_2 s^2 + \ldots + c_n s^n} = \frac{Z_S(s)}{N_S(s)} \tag{4.37}$$

und die des Reglers

$$G_R(s) = \frac{b_0 + b_1 s + b_2 s^2 + \ldots + b_w s^w}{a_0 + a_1 s + a_2 s^2 + \ldots + a_z s^z} = \frac{Z_R(s)}{N_R(s)} \tag{4.38}$$

angegeben werden. Die Führungsübertragungsfunktion des geschlossenen Regelkreises berechnet sich damit zu

$$G_W(s) = \frac{G_R(s)G_S(s)}{1 + G_R(s)G_S(s)}. \tag{4.39}$$

Formt man diesen Ausdruck nun nach $G_R(s)$ um, ergibt sich die Übertragungsfunktion des Reglers

$$G_R(s) = \frac{1}{G_S(s)} \cdot \frac{G_W(s)}{1 - G_W(s)} \tag{4.40}$$

oder in der Schreibweise mit den zuvor eingeführten Zähler- und Nennerpolynomen

$$G_R(s) = \frac{N_S(s)Z_W(s)}{Z_S(s)\left(N_W(s) - Z_W(s)\right)}. \tag{4.41}$$

Damit die Kausalitätsbedingung $w \leq z$ für den Regler erfüllt ist, muss wegen $w = n$ und $z = u + m$ gelten

$$u \geq n - m. \tag{4.42}$$

Um nun einen Regler nach dem sogenannten *Verfahren nach Truxal-Guillemin* auszulegen, betrachtet man die Übertragungsfunktion $G_S(s)$ der Strecke und berechnet anhand von Gleichung (4.42) die erforderliche Ordnung u des zu bestimmenden Nennerpolynoms von $G_W(s)$ aus Gleichung (4.35). Nun ermittelt man für die gewünschte 50%-Anstiegszeit $t_{a,50\%}$, wie in Abb. 4.13 beispielhaft für $u = 2$ gezeigt, die Bezugsfrequenz ω_W und setzt diese in das Nennerpolynom der entsprechenden Ordnung aus Gleichung (4.36) ein. Damit sind alle Polynome bekannt, um gemäß Gleichung (4.41) die Reglerübertragungsfunktion $G_R(s)$ zu bestimmen. [21, 39]

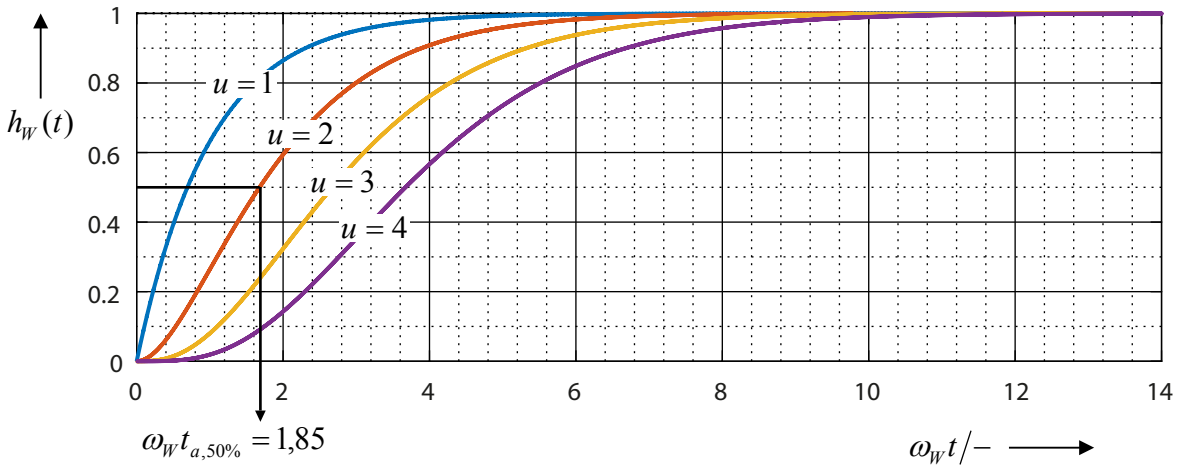


Abbildung 4.13: Normierte Führübergangsfunktion in der Binomial-Form nach [21].

4.2 Entwurf des Reglers

Beim Entwurf eines Reglers wird grundsätzlich nach dem in Abb. 4.14 dargestellten Schema vorgegangen. Zunächst muss die Regelungsaufgabe definiert werden. Hiernach können Gütean-

forderungen an den geschlossenen Regelkreis wahlweise im Zeit- oder Bildbereich formuliert werden. Anschließend wird ein für das vorliegende Problem geeigneter Reglertyp ausgewählt beziehungsweise eine eigene Reglerstruktur entwickelt. Im nächsten Schritt gilt es, den Regler mit Hilfe eines der in Abschnitt 4.1.4 vorgestellten Verfahren einzustellen und möglichst optimale Reglerparameter zu finden. Nun kann entweder durch Vermessung des aufgebauten Regelkreises oder durch Simulation des geschlossenen Regelkreises überprüft werden, ob dieser die Güteanforderungen einhält. Ist das der Fall, so ist das Regelungsproblem erfolgreich gelöst. Andernfalls kann zunächst versucht werden, durch Wahl anderer Reglerparameter die Güteanforderungen zu erreichen. Gelingt dies nicht, muss die Reglerstruktur überarbeitet werden.

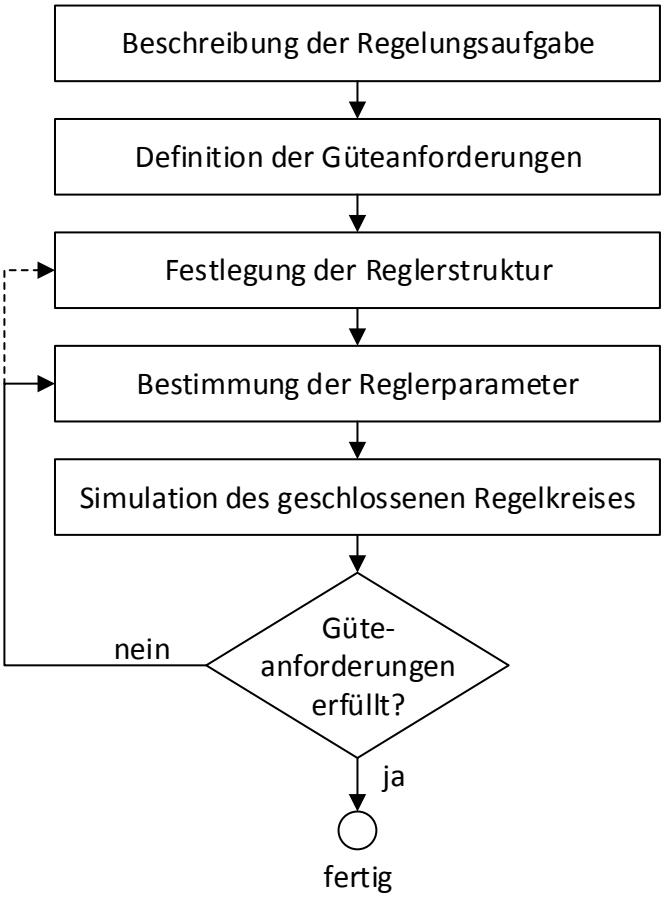


Abbildung 4.14: Vorgehensweise beim Entwurf eines Reglers.

Im folgenden Kapitel wird der Reglerentwurf für das Spannsystem anhand dieses Vorgehensmodells durchgeführt. Entsprechend gliedern sich die Unterkapitel nach den einzelnen Vorgehensschritten des Schemas.

4.2.1 Beschreibung der Regelungsaufgabe

Aufgabe des im Folgenden zu entwerfenden digitalen Reglers wird es sein, die Temperatur T_1 der Spannplatte, die Werte zwischen $-15\,°C$ und $75\,°C$ annehmen kann, möglichst konstant auf

einem vorgegebenen Sollwert, der im selben Temperaturbereich wie T_1 liegen kann, zu halten. Das bedeutet, Störungen der Spannplattentemperatur müssen so schnell wie möglich ausgeregelt werden und es darf zu keiner großen Auslenkung der Spannplattentemperatur infolge einer Störung kommen. Folglich ist die Realisierung einer Festwertregelung erforderlich. Da jedoch im Betrieb des Spannsystems verschiedene Temperatursollwerte angefahren werden sollen, ist ebenfalls ein gutes Führungsverhalten der Regelung wünschenswert. Dies schließt insbesondere ein, dass der Temperatursollwert statisch führungsgenau, das heißt ohne bleibende Regelabweichung, angefahren wird. Weiterhin ist ein möglichst schnelles Erreichen der Solltemperatur wünschenswert. Ein Überschwingen der Regelgröße ist zulässig, sofern es dabei nicht zur Überschreitung einer kritischen Maximaltemperatur von 75 °C kommt. Andernfalls könnte das Peltier-Element beschädigt werden.

Der Regler selbst ist als digitaler Regler, der gerätetechnisch auf einem Mikrocontroller ausgeführt werden soll, zu realisieren. Dementsprechend ist das Reglergesetz zeitdiskret zu formulieren. Auf Basis der Modellierungsergebnisse wird die Abtastzeit T dabei auf einen Wert von 100 ms festgelegt. Ausgangsgröße des Reglers ist der Tastgrad eines PWM-Signals, der ganzzahlige Werte zwischen 0 und 255 annehmen kann.

4.2.2 Definition der Güte-Anforderungen

Die Güte einer Regelung kann durch die in Abschnitt 4.1.1 eingeführten Größen quantitativ beschrieben werden. Sie werden an dieser Stelle als Zielvorgaben des Reglerentwurfes festgelegt und dienen als Bewertungskriterien für die im weiteren Verlauf ausgelegten Regler. Die Überprüfung der Güte-Anforderungen findet in Abschnitt 4.4 statt.

Da die Spannplattentemperatur sowohl im Heiz- als auch im Kühlbetrieb vorgegebene Werte möglichst genau annehmen soll, muss die bleibende Regelabweichung $e(\infty)$ des geschlossenen Regelkreises möglichst klein ausfallen oder idealerweise verschwinden. Damit sich die gewünschten Temperaturwerte möglichst schnell einstellen, die Dynamik der Regelung also möglichst groß ausfällt, sind eine geringe Verzugszeit T_u, Anstiegszeit T_a und Ausregelzeit t_ε anzustreben. Dabei ist ein Überschwingen im Kühlbetrieb zulässig, muss jedoch im Heizbetrieb sehr klein gehalten oder vermieden werden, um eine Zerstörung des Systems zu verhindern. Im Kühlbetrieb wird daher die maximal erlaubte Überschwingweite $e_{\max}$ auf 25 % festgelegt, während sie im Heizbetrieb einen Wert von 5 % nicht überschreiten sollte. Treten Überschwinger auf, so sollte dies zu einem möglichst frühen Zeitpunkt $t_{\max}$ geschehen. Grundvoraussetzung für den Betrieb der Regelung ist darüber hinaus die Stabilität des geschlossenen Regelkreises, da eine instabile Regelung zu einer Zerstörung des Systems führen könnte.

4.2.3 Festlegung der Reglerstruktur

Als Regler für die Temperaturregelung eignet sich aufgrund der relativ hohen Zeitkonstante der Regelstrecke ein PI-Regler. Dieser arbeitet einerseits aufgrund des integrierenden Verhaltens statisch führungsgenau und ist andererseits bedingt durch den enthaltenen Proportionalterm schnell genug, um Störungen innerhalb kurzer Zeit zuverlässig auszuregeln sowie Änderungen

der Führungsgröße ausreichend schnell zu folgen. Im Sinne einer hohen Flexibilität der Regelung soll jedoch zusätzlich ein D-Anteil mit in den Regler aufgenommen werden, sodass sich ein PID-Regler ergibt. Durch entsprechende Parameterwahl ($K_D = 0$) kann ein solcher Regler problemlos zunächst als reiner PI-Regler betrieben werden, ermöglicht jedoch im Rahmen einer etwaigen späteren Modifikation der Regelstrecke, einen D-Anteil einstellen zu können. Daher wird als Reglerstruktur ein PID-Regler entsprechend Abschnitt 4.1.2 und Abschnitt 4.1.3 zugrundegelegt. Dieser ist einfach zu implementieren und optimieren und daher Quasi-Standard in der Regelungstechnik. Wie in Abschnitt 4.1.3 bereits angedeutet, müssen am Reglergesetz aus Gleichung (4.17) noch einige Anpassungen vorgenommen werden. Darunter insbesondere die Filterung des D-Anteils, was der Einführung einer Verzögerungszeitkonstante entspricht. Aus dem reinen D-Anteil wird somit ein zeitverzögerter DT_1-Anteil mit der Übertragungsfunktion

$$G_{R,D}(s) = \frac{U(s)}{E_y(s)} = K_D \frac{N}{1 + \frac{N}{s}}. \tag{4.43}$$

Die Filterkonstante N bestimmt dabei das Verhalten des D-Anteils. Für $N \to \infty$ geht der Filter-Term gegen s und es ergibt sich ein rein differentielles Glied. Die so modifizierte Regler-Struktur ist in Abb. 4.15 in Form eines Blockschaltbilds dargestellt. Hierin ist die Eingangsgröße des D-Anteils die Differenz zweier aufeinanderfolgender Messwerte der Regelgröße $e_y(k) = y(k-1) - y(k)$. Das zugehörige Reglergesetz lautet

$$u(k) = K_R e(k) + K_I \sum_{i=0}^{k} e(i)T + \mathrm{FK}(k). \tag{4.44}$$

Dabei stellt die Größe $\mathrm{FK}(k)$ den D-Anteil des Reglers dar, der als

$$\mathrm{FK}(k) = N\left(K_D e_y(k) - \mathrm{FS}(k)\right) \tag{4.45}$$

berechnet werden kann. $\mathrm{FS}(k)$ ist dabei die Summe über alle vorangegangenen Werte des D-Anteils, berechnet sich also zu

$$\mathrm{FS}(k) = \sum_{i=0}^{k-1} \mathrm{FK}(i)T. \tag{4.46}$$

Die rekursive Berechnung des gefilterten D-Anteils ist in der Rückkopplung des Ausgangssignals durch den Integrator auf das Eingangssignal begründet. Weitere Details zur Filterung können in [32] nachgelesen werden.

Weiterhin ist die Realisierung eines Anti-Windup-Mechanismus für den Integralterm erforderlich. Ein Windup bezeichnet dabei das Anwachsen des Integralterms der Regelung auf sehr große Werte, die über den vom Stellglied realisierbaren Grenzwerten der Stellgröße (hier Werte zwischen 0 und 255) liegen. Dies tritt immer dann auf, wenn die Regelgröße den Sollwert nicht erreicht, da der Regler in diesem Fall die Stellgröße in Erwartung einer Beeinflussung der Regelgröße immer weiter vergrößert. Wird der Sollwert anschließend auf einen kleineren Wert zurückgesetzt, dauert es eine gewisse Zeit, bis der Integralterm und damit die Ausgangsgröße des Reglers kleine Werte erreicht haben. Dies resultiert in einem zeitlich verzögerten Führungsverhalten der Regelgröße. Wird der Integralterm jedoch auf den Wertebereich begrenzt, den das Stellglied verarbeiten kann,

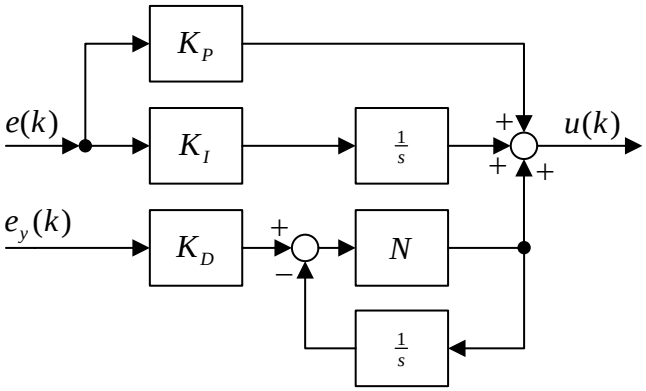


Abbildung 4.15: Blockschaltbild des angepassten PID-Reglers mit Filterung des D-Anteils und Abweichung der Regelgröße $e_y(k)$ als Eingang des D-Anteils.

tritt dieses Problem nicht mehr auf. Ein solcher Mechanismus wird als Anti-Integrator-Windup bezeichnet. [21]

Eine weitere Anpassung des Reglergesetzes umfasst die Beschränkung der Ausgangsgröße des Reglers auf vorgegebene Grenzwerte. Da der Regler später den Eingangswert einer Funktion berechnen soll, der sich nur im Wertebereich zwischen 0 und 255 bewegen darf, sollte auch die Ausgangsgröße des Reglers auf entsprechende Werte beschränkt werden. Andernfalls könnte die Stellgröße trotz Anti-Integrator-Windup diese Grenzwerte überschreiten, da neben dem Integralterm auch der Proportional- und Differentialterm in die Stellgröße mit einfließen. [20, 25, 40]

Die um den Anti-Integrator-Windup und die Stellgrößenbeschränkung ergänzte Reglerstruktur ist in Abb. 4.16 zu sehen. Sie stellt die Grundlage für die Implementierung des Reglers in Abschnitt 4.3 dar.

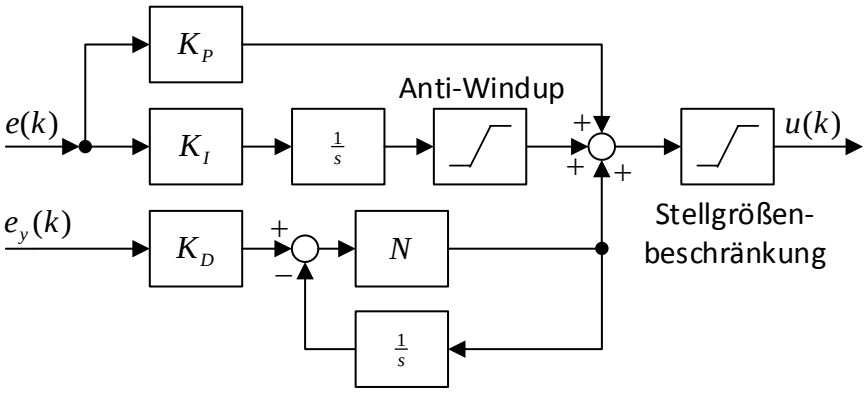


Abbildung 4.16: Blockschaltbild des um einen Anti-Integrator-Windup sowie eine Stellgrößenbeschränkung erweiterten PID-Reglers.

4.2.4 Bestimmung der Reglerparameter

Die Reglerparameter des PI-Reglers werden auf Basis der in Abschnitt 3.7 aufgestellten Modelle 1.1 für den Kühlbetrieb und 2.2 für den Heizbetrieb ermittelt. Hierzu werden zunächst Parameter mithilfe des in Abschnitt 4.1.4.2 vorgestellten empirischen Einstellverfahrens nach Chien, Hrones und Reswick ermittelt und anschließend Reglereinstellwerte mithilfe der PID-Tuner Toolbox in MATLAB bestimmt. Da die Anforderungen an die Regelung gering sind, kann auf eine Optimierung der Reglerparameter verzichtet werden.

Zur empirischen Bestimmung der Reglerparameter nach Chien, Hrones und Reswick werden die in Abschnitt 3.9 aus den Übergangsfunktionen der Modelle abgelesenen Werte für Verstärkungsfaktoren, Verzugs- und Anstiegszeiten benötigt. Diese sind in Tabelle 4.5 erneut aufgeführt. Anders als in Abschnitt 4.1.1 definiert, ist die Anstiegszeit T_a hier die Zeit, die die Temperatur benötigt, um von 10 % auf 90 % ihres stationären Wertes anzusteigen. Entsprechend ist die Verzugszeit diejenige Zeit, die vergeht, bis die Temperatur 10 % ihres stationären Wertes erreicht hat.

Tabelle 4.5: Aus den Übergangsfunktionen abgelesene Verstärkungsfaktoren, Anstiegs- und Verzugszeiten der Regelstrecke zur Parameterbestimmung nach Chien, Hrones und Reswick.

Größe	Einheit	Betriebsmodus	
		Kühlbetrieb	Heizbetrieb
Anstiegszeit T_a	s	16,109	72,189
Verzugszeit T_u	s	1,427	2,048
Verstärkungsfaktor K_S	-	−0,103	0,331
Hilfsgröße λ	-	−109,600	106,491

Im Kühlbetrieb beträgt die Regelbarkeit $T_{a,K}/T_{u,K} = 11{,}289$. Das System lässt sich folglich gut durch ein PT_1T_t-Glied approximieren und somit ist das Verfahren nach Chien, Hrones und Reswick anwendbar. Es ergibt sich im Kühlbetrieb für die Hilfsgröße λ entsprechend Gleichung (4.19) ein Wert von −109,6. Mittels Tabelle 4.3 werden nun die Reglereinstellwerte $K_R = -76{,}72$ und $T_N = 3{,}325\,\mathrm{s}$ für den PI-Regler ermittelt. Für die Parallelform des Reglers wird der Wert $K_I = K_R/T_N = -23{,}074$ errechnet. Dabei wurde der Fall der Festwertregelung mit einer erlaubten Überschwingweite von 20 % ausgewählt.

Analog lassen sich für den Heizbetrieb die Hilfsgröße $\lambda = 106{,}491$ und damit die Reglereinstellwerte $K_R = 63{,}894$ sowie $T_N = 8{,}192\,\mathrm{s}$, also $K_I = 7{,}8$, bestimmen. Dabei wurde der Fall einer Festwertregelung mit aperiodischem Führungsverhalten gewählt, um ein Überschwingen der Temperatur zu verhindern. Die Regelbarkeit beträgt in diesem Fall $T_{a,H}/T_{u,H} = 35{,}249$, womit das Verfahren nach Chien, Hrones und Reswick ebenfalls anwendbar ist.

Ein weiterer Satz Reglerparameter wird mithilfe des in MATLAB enthaltenen PID-Tuners generiert. In diesen werden die in Abschnitt 3.7.5 erstellten Modelle geladen und anschließend ein automatisches Tuning der Reglerparameter durchgeführt. Anschließend erfolgt ein manuelles Justieren der Parameter, indem Antwortzeit und Robustheit der Regelung vorgegeben werden.

Es ergeben sich für den Kühlbetrieb die Reglereinstellwerte $K_R = -13{,}542$ und $K_I = -3{,}487$ und für den Heizbetrieb $K_R = 8{,}347$ und $K_I = 1{,}119$. Dabei wurden die Werte im Heizbetrieb so gewählt, dass die maximale Überschwingweite 5 % beträgt.

In Abschnitt 4.4 wird das Verhalten des geschlossenen Regelkreises mit den soeben ermittelten Reglerparametern simuliert und die Eignung der Parameter in Hinblick auf die Einhaltung der Anforderungen an die Regelgüte überprüft.

4.3 Implementierung des PID-Reglers

Basierend auf den in Abschnitt 4.2 getätigten Überlegungen zum Reglerentwurf soll im Folgenden ein digitaler PID-Regler auf dem in der Treiberschaltung verbauten Mikrocontroller implementiert werden. Das zugrundeliegende Regelgesetz wurde dabei in Abschnitt 4.2.3 erarbeitet und stellt eine Verbesserung der Arduino-PID-Bibliothek nach [20] dar. Um eine möglichst hohe Flexibilität des Reglers zu gewährleisten, ist dieser als Klasse mit entsprechenden Attributen und Methoden implementiert. Der vollständige Quellcode ist in Listing A.8 zu finden. Die Implementierung in Form einer Klasse bietet den Vorteil, dass innerhalb der übergeordneten Software, in der ein Regler eingebunden wird, beliebig viele Objekte vom Typ der Regler-Klasse instanziiert werden können. Diese sind voneinander unabhängig, sodass ohne zusätzlichen Aufwand mehrere Regler mit unterschiedlichen Ein- und Ausgangsgrößen sowie Reglerparametern gleichzeitig angelegt und betrieben werden können.

Das UML-Klassendiagramm der Reglerklasse in Abb. 4.17 enthält sämtliche private Attribute und öffentliche Methoden der Klasse. Die zugehörige Implementierung ist in Listing 4.1 zu finden. Das Attribut *Regelwert* repräsentiert die zeitdiskrete Regelgröße $y(k)$ zum Zeitpunkt $t = kT$, *Regelabweichung* beschreibt die Größe $e(k)$, *Stellwert* die Reglerausgangsgröße $u(k)$ und das Attribut *Letzter_Regelwert* ist die Regelgröße zum Zeitpunkt $t = (k-1)T$, also $y(k-1)$. Die Attribute *Regelabweichung_Summe* und *Filter_Summe* sind Hilfsgrößen zur Berechnung der im Reglergesetz auftretenden Integral-Anteile. Weitere wichtige Attribute sind die Reglerparameter *Kp*, *Ki*, *Kd* und *N*, wobei der letzte Parameter den Filter-Koeffizienten zur Zeitverzögerung des D-Anteils des Reglers darstellt. Ebenfalls in der Klassendefinition enthalten ist die Abtastzeit T, einmal als Attribut *Sample_Zeit* in Mikrosekunden und als Attribut *Sample_Zeit_in_Sekunden* in der Einheit Sekunden. Die Attribute *Input* und *Output* sind Zeiger auf die im übergeordneten Kontext mit dem Regler gekoppelte Ein- beziehungsweise Ausgangsgröße. Das Attribut *Modus* schließlich legt fest, ob der Regler ein- oder ausgeschaltet ist. Die Methoden der Reglerklasse umfasssen den Konstruktor, mit *PID_SetzeModus()* eine Methode zum Aktivieren und Deaktivieren des Reglers und mit *PID_SetzeParameter()* eine Methode zum Anpassen der Reglerparameter während der Laufzeit des Reglers. Weiterhin gibt es eine Methode zur Initialisierung des Reglers und die eigentliche Kalkulationsroutine, in der aus dem aktuellen Wert der Eingangsgröße *Input* entsprechend des Reglergesetzes die Ausgangsgröße *Output* berechnet wird.

Listing 4.1: Klassendefinition der PID-Regler-Klasse.

```
1 class PID_Regler {
```

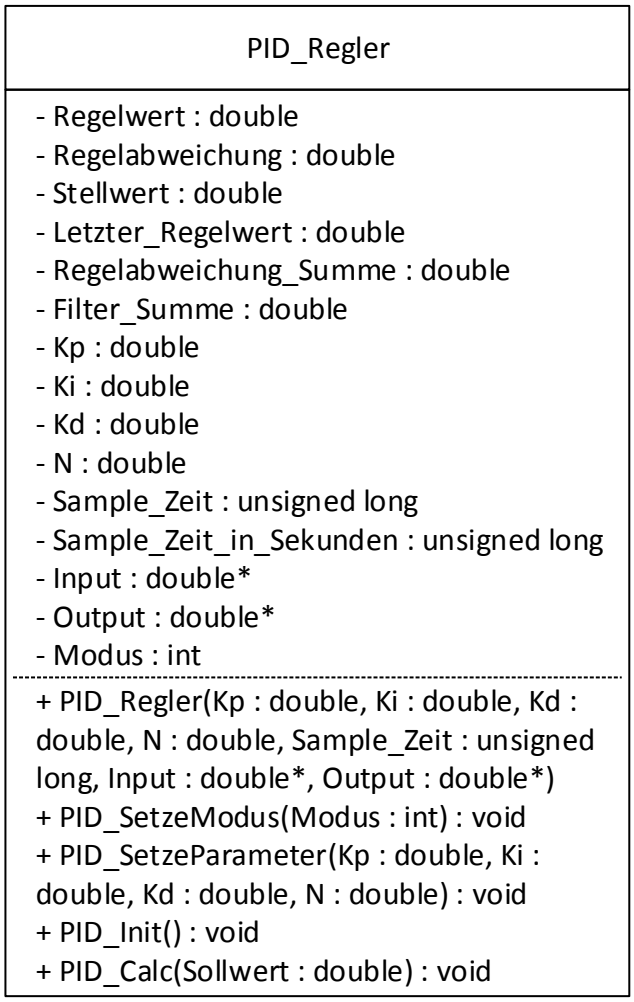


Abbildung 4.17: Das UML-Klassendiagramm des PID-Reglers enthält alle Attribute und Methoden der Klasse.

```
2
3   private:
4     double Regelwert;
5     double Regelabweichung;
6     double Stellwert;
7     double Letzter_Regelwert;
8     double Regelabweichung_Summe;
9     double Filter_Summe;
10    double Kp, Ki, Kd, N;
11    unsigned long Sample_Zeit;
12    double Sample_Zeit_in_Sekunden;
13    double* Input;
14    double* Output;
15    int Modus;
16
17  public:
18    PID_Regler(double Kp, double Ki, double Kd, double N, unsigned long Sample_Zeit, double*
```

```
    Input, double* Output);
19    void PID_SetzeModus(int Modus);
20    void PID_SetzeParameter(double Kp, double Ki, double Kd, double N);
21    void PID_Init();
22    void PID_Calc(double Sollwert);
23 };
```

Der in Listing 4.2 dargestellte Konstruktor der Klasse dient zur Initialisierung eines PID-Regler-Objekts im übergeordneten Kontext. Dem zu erstellenden Objekt müssen dabei konkrete Werte für alle vier Reglerparameter, die Sample-Zeit sowie die Speicheradressen von Ein- und Ausgangsgröße des Reglers übergeben werden. Der Konstruktor weist diese Werte dann den Attributen der Klasse zu. Der Modus wird zunächst auf den Zustand *AUS*, was einer logischen 0 entspricht, gesetzt.

Listing 4.2: Konstruktor der PID-Regler-Klasse.

```
1 PID_Regler::PID_Regler(double Kp, double Ki, double Kd, double N, unsigned long Sample_Zeit,
  double* Input, double* Output) {
2   this->Kp = Kp;
3   this->Ki = Ki;
4   this->Kd = Kd;
5   this->N = N;
6   this->Sample_Zeit = Sample_Zeit;
7   this->Sample_Zeit_in_Sekunden = ((double)Sample_Zeit)/1000000;
8   this->Input = Input;
9   this->Output = Output;
10  this->Modus = AUS;
11 }
```

Nach Erstellung eines PID-Regler-Objekts muss der Regler aktiviert, das heißt der Zustand auf *AN*, also logisch 1, gesetzt werden. Dies geschieht mithilfe der Methode *PID_SetzeModus()* durch Übergabe des Werts *AN*. Die Implementierung dieser Methode ist in Listing 4.3 dargestellt. Beim Aufruf der Methode wird immer geprüft, ob sich der Modus von *AUS* in *AN* geändert hat. Ist das der Fall, wird die Initialisierungsmethode aufgerufen. Hierdurch wird sichergestellt, dass der Regler beim erstmaligen Einschalten oder Wiedereinschalten nach Deaktivierung ein sinnvolles Verhalten zeigt.

Listing 4.3: Methode zur Aktivierung und Deaktivierung des PID-Reglers.

```
1 void PID_Regler::PID_SetzeModus(int Modus) {
2   bool neuerModus = (Modus == AN);
3   if(neuerModus == !(this->Modus)) {
4     PID_Regler::PID_Init();
5   }
6   this->Modus = neuerModus;
7 }
```

Die in Listing 4.4 gezeigte Initialisierungsmethode hat die Aufgabe, den Attributen der Reglerklasse geeignete Werte zuzuweisen. Erwähnenswert ist an dieser Stelle das Attribut *Regelabweichung_Summe*, das den Integralterm des Reglers darstellt. Dieser wird bei der Initialisierung

deshalb auf den Wert der Reglerausgangsgröße gesetzt, damit beim Wiedereinschalten aus dem deaktivierten Zustand der Integralterm genau die benötigte Ausgangsgröße einstellt. Dies ist erforderlich, weil P- und D-Anteil zu diesem Zeitpunkt den Wert null haben. Einhergehend muss eine Begrenzung des Integralterms erfolgen, damit die Reglerausgangsgröße innerhalb der in *OUTPUT_MIN_VALUE* und *OUTPUT_MAX_VALUE* spezifizierten Grenzen bleibt.

Listing 4.4: Initialisierungsmethode der PID-Regler-Klasse.

```
1 void PID_Regler::PID_Init() {
2   // Werte initialisieren
3   this->Letzter_Regelwert = *(this->Input);
4   this->Regelabweichung_Summe = *(this->Output);
5   this->Filter_Summe = 0.0;
6   if(this->Regelabweichung_Summe > OUTPUT_MAX_VALUE) this->Regelabweichung_Summe =
    OUTPUT_MAX_VALUE;
7   else if(this->Regelabweichung_Summe > OUTPUT_MIN_VALUE) this->Regelabweichung_Summe =
    OUTPUT_MIN_VALUE;
8 }
```

Die Implementierung des in Abschnitt 4.2.3 entwickelten Reglers innerhalb der Kalkulationsmethode ist in Listing 4.5 zu sehen. Ist der Regler aktiviert, wird zunächst die Eingangsgröße erfasst, diese mit dem an die Methode übergebenen Sollwert für die Regelgröße verglichen und entsprechend die Regelabweichung berechnet. Anschließend wird der aktuelle Wert des I-Anteils auf den Integralterm aufaddiert und auf einen Wertebereich von 0 bis 255 beschränkt (Anti-Integrator-Windup). Anschließend wird die Stellgröße des Reglers entsprechend des Reglergesetzes berechnet und ebenfalls auf einen Wertebereich von 0 bis 255 begrenzt. Zum Schluss wird die berechnete Stellgröße auf den Reglerausgang geschrieben und der Größe $y(k-1)$ der aktuelle Wert der Regelgröße zugewiesen, sodass bei erneutem Aufruf der Kalkulationsmethode wieder ein Wert der Regelgröße zum früheren Zeitpunkt $t = (k-1)T$ vorliegt.

Listing 4.5: Kalkulationsmethode der PID-Regler-Klasse.

```
1  void PID_Regler::PID_Calc(double Sollwert) {
2
3    // Berechnung nur durchführen, wenn der Regler akitiviert ist
4      if(Modus == AN) {
5
6        // Regelgröße einlesen und Regelabweichung berechnen
7        this->Regelwert = *(this->Input);
8        this->Regelabweichung = Sollwert - this->Regelwert;
9
10       // Integralterm berechnen
11       this->Regelabweichung_Summe += Ki * this->Regelabweichung * this->Sample_Zeit_in_Sekunden
         ;
12
13       // Integralterm begrenzen
14       if(this->Regelabweichung_Summe > OUTPUT_MAX_VALUE) this->Regelabweichung_Summe =
         OUTPUT_MAX_VALUE;
15       else if(this->Regelabweichung_Summe < OUTPUT_MIN_VALUE) this->Regelabweichung_Summe =
         OUTPUT_MIN_VALUE;
16
```

```
17      // Filterkoeffizient und Filterintegral berechnen
18      double FilterKoeffizient = (Kd * (this->Letzter_Regelwert - this->Regelwert) - this->
        Filter_Summe) * this->N;
19      this->Filter_Summe += this->Sample_Zeit_in_Sekunden * FilterKoeffizient;
20
21      // Stellgröße berechnen
22      this->Stellwert = Kp * this->Regelabweichung + this->Regelabweichung_Summe +
        FilterKoeffizient;
23
24      // Stellwert begrenzen
25      if(this->Stellwert > OUTPUT_MAX_VALUE) this->Stellwert = OUTPUT_MAX_VALUE;
26      else if(this->Stellwert < OUTPUT_MIN_VALUE) this->Stellwert = OUTPUT_MIN_VALUE;
27
28      // Stellgröße auf Ausgang geben
29      *(this->Output) = this->Stellwert;
30
31      // Regelwert für Berechnung im nächsten Funktionsaufruf speichern
32      this->Letzter_Regelwert = this->Regelwert;
33  }
34 }
```

Die PID-Reglerklasse verfügt weiterhin über eine Methode zum Setzen der Reglerparameter, die in Listing 4.6 dargestellt ist. Dieser werden beim Aufruf Werte für die Reglerparameter K_P, K_I, K_D und N übergeben, die den entsprechenden Attributen zugewiesen werden. Eine solche Methode ist sinnvoll, wenn zur Laufzeit des Reglers die Reglerparameter verändert werden sollen.

Listing 4.6: Methode zum Setzen der Reglerparameter eines PID-Regler-Objektes.

```
1 void PID_Regler::PID_SetzeParameter(double Kp, double Ki, double Kd, double N) {
2    this->Kp = Kp;
3    this->Ki = Ki;
4    this->Kd = Kd;
5    this->N = N;
6 }
```

Die Verwendung der implementierten Reglerklasse im übergeordneten Kontext zeigt das Beispiel in Listing 4.7. Dort wird zuerst die Header-Datei mit der obigen Implementierung der Reglerklasse eingebunden. Anschließend werden die Variablen *Eingangswert* als Regel- und Eingangsgröße, *Ausgangswert* als Stell- und Ausgangsgröße sowie *Sollwert* als Sollwert des Reglers gewählt. Weiterhin wird die Abtastdauer in der Variable *Sample_Zeit* festgelegt. Die genannten Variablen sind dabei Fließkommazahlen vom Datentyp *double*. Anschließend wird das Regler-Objekt durch den Aufruf des Konstruktors und Übergabe der Reglerparameter $K_P = 1$, $K_I = 1$, $K_D = 1$ und $N = 1$ sowie der Abtastdauer und Adressen der Ein- und Ausgangsvariablen erstellt. In der einmalig beim Start des Mikrocontrollers ausgeführten *setup()*-Funktion muss der Regler initialisiert werden. Hierzu werden der aktuelle Wert der Regelgröße erfasst, indem mithilfe des Befehls *analogRead(0)* ein Messwert des Sensors am Analog-Digital-Wandler 0 eingelesen wird, und die Initialisierungsroutine durch Änderung des Reglermodus von *AUS* in *AN* aufgerufen. Zudem wird der Sollwert auf einen konstanten Wert von 50.0 festgelegt. Die eigentliche Regelung

findet in der *loop()*-Funktion, die in einer Endlosschleife immer wieder aufgerufen wird, statt. Hier werden in jedem Schleifendurchlauf zunächst die Regelgröße gemessen, die Stellgröße für den aktuellen Sollwert durch Aufruf der Kalkulationsroutine *PID_Calc(Sollwert)* berechnet und mittels der Funktion *analogWrite()* die berechnete Stellgröße auf den Ausgangspin 3 des Mikrocontrollers geschrieben. Anschließend wird das Programm durch Aufruf der Funktion *delay(100)* für eine Dauer von 100 ms pausiert, bevor die Schleife erneut durchlaufen wird. Hierdurch wird die konstante Abtastzeit des Reglers eingestellt.

Listing 4.7: Verwendung der PID-Regler-Klasse im übergeordneten Kontext.

```
1  #include Regler.h // Regler-Klasse einbinden
2
3  double Eingangswert = 0.0; // Eingangsvariable
4  double Ausgangswert = 0.0; // Ausgangsvariable
5  double Sollwert; // globale Variable für den Sollwert der Temperatur
6
7  double Sample_Zeit = 100000; // Mikrosekunden
8  PID_Regler Regler(1.0,1.0,1.0,1.0,Sample_Zeit,&Eingangswert,&Ausgangswert); // Regler-Objekt
   erstellen
9
10 void setup()
11 {
12   Eingangswert = analogRead(0); // Eingangsgröße einlesen
13   Regler.PID_SetzeModus(AN); // Regler einschalten
14
15   Sollwert = 50.0; // Sollwert für die Regelgröße setzen
16 }
17
18 void loop()
19 {
20   Eingangswert = analogRead(0); // Eingangsgröße einlesen
21   Regler.PID_Calc(Sollwert); // Stellgröße für gegebenen Sollwert berechnen
22   analogWrite(3,Ausgangswert); // Stellgröße auf Pin 3 ausgeben
23
24   delay(100); // 100 ms warten
25 }
```

4.4 Simulation des geschlossenen Regelkreises

Nachdem der Regler in den vorangegangenen Abschnitten entworfen und eingestellt wurde, wird nun das Verhalten des geschlossenen Regelkreises simuliert und die Einhaltung der in Abschnitt 4.2.2 aufgestellten Güteanforderungen überprüft. Die Simulation erfolgt in Simulink auf Basis des in Abb. 4.18 dargestellten Modells des geschlossenen Regelkreises. Das Integrationsintervall wird dabei auf 0 bis 60 s festgelegt und ein Solver mit fester Schrittweite von 0,01 s ausgewählt. Die Regelstrecken inklusive Messglied werden im Modell durch die Übertragungsfunktionen der in Abschnitt 3.7 aufgestellten Modelle 1.1 und 2.2 repräsentiert. Der Regler wird durch einen kontinuierlichen PI-Regler-Block realisiert, dessen Ausgangsgröße auf

einen Wertebereich zwischen 0 und 255 beschränkt ist. Als Anti-Windup-Methode wird *clamping* ausgewählt. Dies entspricht der oben beschriebenen Beschränkung des Integrator-Terms auf den Ausgangswertebereich des Reglers. Die Reglerparameter werden entsprechend der in Abschnitt 4.2.4 bestimmten Werte eingestellt. Als Führungsgröße dient eine Sprungfunktion, die zum Zeitpunkt $t = 1\,\mathrm{s}$ im Kühlbetrieb von $20\,^{\circ}\mathrm{C}$ auf $-5\,^{\circ}\mathrm{C}$ abfällt und im Heizbetrieb von $20\,^{\circ}\mathrm{C}$ auf $70\,^{\circ}\mathrm{C}$ ansteigt. Da das Modell der Regelstrecke durch Linearisierung erstellt wurde und folglich nur Änderungen um einen Arbeitspunkt berücksichtigt, muss auf sämtliche Ausgangswerte der Strecke ein der Umgebungstemperatur entsprechender Offset von $20\,^{\circ}\mathrm{C}$ addiert werden. Die simulierten zeitlichen Verläufe von Führungs-, Stell- und Regelgröße werden mithilfe eines Scope-Blocks aufgezeichnet. Insgesamt wurden dabei zwei Simulink-Modelle, eines für den Kühl- und eines für den Heizbetrieb, erstellt und simuliert. Einzelheiten zur Simulation mit Simulink können in [23] nachgelesen werden.

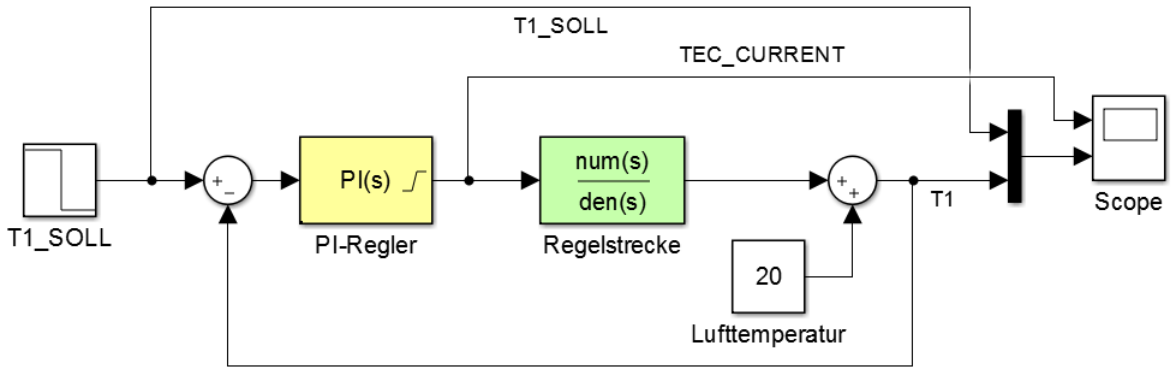
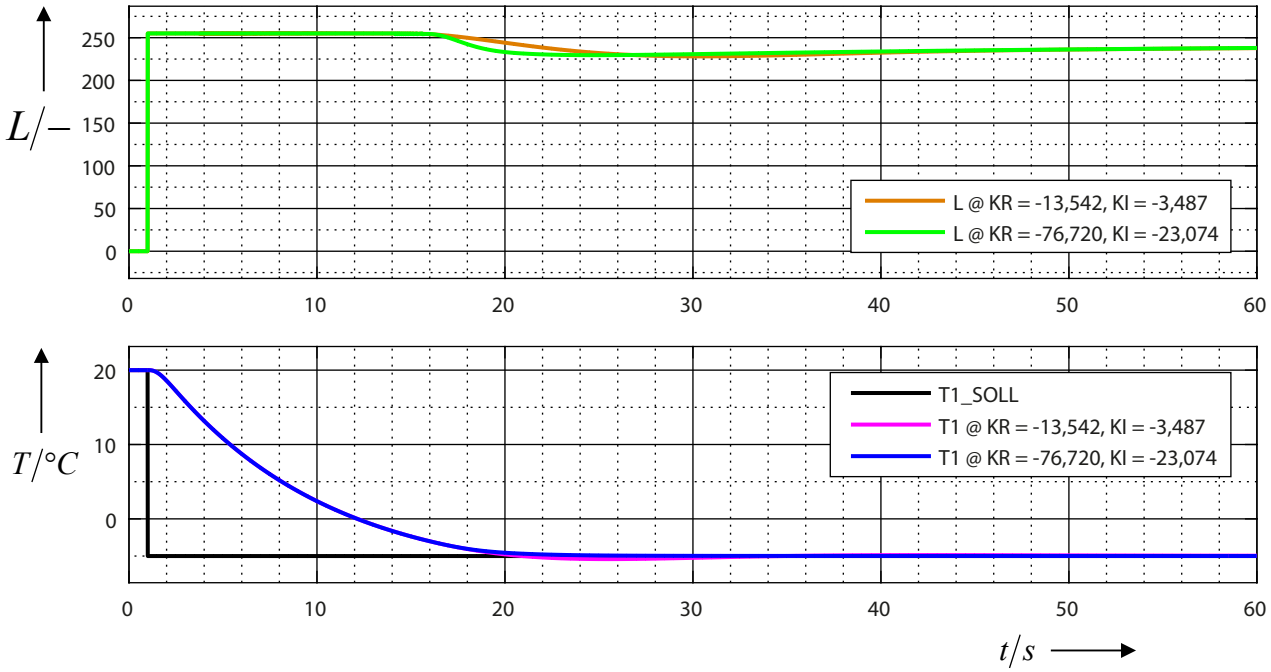


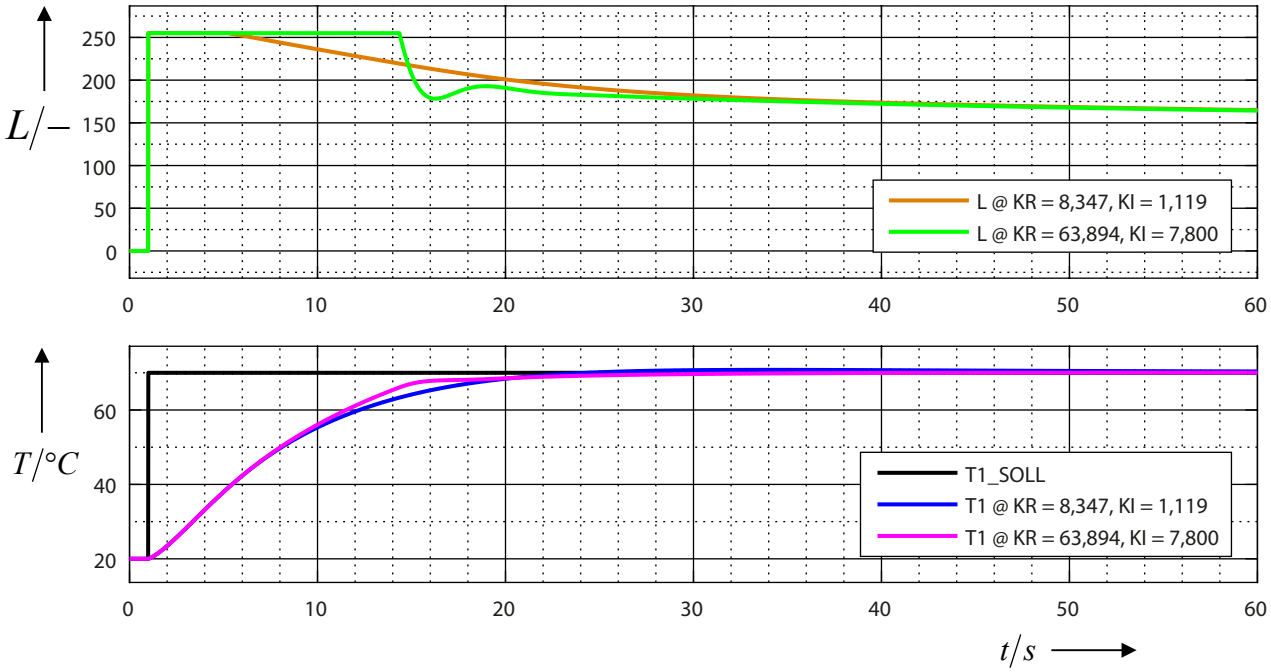
Abbildung 4.18: Simulink-Modell des geschlossenen Regelkreises mit Modellstrecke und PI-Regler.

Die Ergebnisse der Simulation sind in Abb. 4.19a für den Regelkreis im Kühlbetrieb und in Abb. 4.19b für den Regelkreis im Heizbetrieb dargestellt. Dabei wurden jeweils die Stell-, Regel- und Führungsgröße sowohl für den mittels MATLAB PID-Tuner als auch für den empirisch nach Chien, Hrones und Reswick eingestellten Regler berechnet. Trotz der unterschiedlichen Reglerparameter zeigen sich im Verhalten des geschlossenen Regelkreises nur geringfügige Unterschiede zwischen den beiden Varianten. Dies liegt insbesondere daran, dass das Stellglied bei beiden Reglern zu Beginn des Regelungsvorganges in die Sättigung geht und sich somit ein identisches Anstiegsverhalten der Regelgröße ergibt. Erst nachdem die Regelgröße unter den Grenzwert von 255 fällt, unterscheidet sich das Verhalten der Regelkreise. Aufgrund dessen konnte zuvor auf eine Optimierung der Reglerparameter verzichtet werden, da eine große Bandbreite verschiedener Reglerparameter zum gleichen Verhalten des geschlossenen Regelkreises führt. Ein leistungsfähigeres Stellglied wäre die einzige Möglichkeit, die Anstiegszeit der Spannplattentemperatur zu verkürzen.

Quantitative Aussagen über die Anstiegszeit T_a, die Ausregelzeit t_ε, die maximale Überschwingweite $e_{\max}$ und das Amplitudenmaximum können Tabelle 4.6 entnommen werden. Sie wurden aus den Sprungantworten in Abb. 4.19 abgelesen. Die Anstiegszeit T_a bezeichnet hierbei die Zeit, die die Temperatur T_1 der Spannplatte benötigt, um von $10\,\%$ auf $90\,\%$ ihres stationären Wertes anzusteigen. Für die Ausregelzeit t_ε wurde ein Toleranzband von $\pm 2\,\%$ um den stationären Wert



(a) Simulation der Regelung im Kühlbetrieb.



(b) Simulation der Regelung im Heizbetrieb.

Abbildung 4.19: Simulationsergebnisse des geschlossenen Regelkreises im Kühl- sowie Heizbetrieb. Dargestellt sind die Verläufe der Führungsgröße $T_{1,soll}(t)$, der Stellgröße $L(t)$ und der Regelgröße $T_1(t)$ für jeweils zwei verschiedene Sätze von Reglerparametern.

festgelegt. Auch an dieser Stelle werden die nur geringfügigen Unterschiede zwischen den jeweils zwei verschiedenen Parametersätzen für den Heiz- und Kühlregler deutlich.

Tabelle 4.6: Eigenschaften des nichtlinearen Modells des geschlossenen Regelkreises.

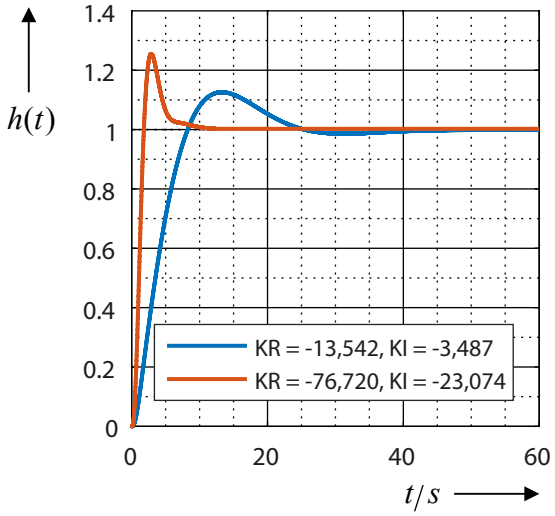
Größe	Einheit	Kühlbetrieb		Heizbetrieb	
		$K_R = -13{,}542$, $K_I = -3{,}487$	$K_R = -76{,}720$, $K_I = -23{,}074$	$K_R = 8{,}347$, $K_I = 1{,}119$	$K_R = 63{,}894$, $K_I = 7{,}800$
Anstiegszeit T_a	s	12,85	12,64	13,82	11,34
Ausregelzeit t_ε	s	19,21	19,16	22,73	21,26
Überschwingweite $e_{\max}$	%	7,55	0,00	0,37	0,00
Amplitudenmaximum	°C	−5,41	−4,99	70,79	70,04

Um weitere Aussagen über das Verhalten des Regelkreises treffen zu können, bietet sich eine Linearisierung an, da der ursprüngliche Regelkreis aufgrund der Beschränkung der Stellgröße des Reglers ein stark nichtlineares Verhalten aufweist. Die Eigenschaften des linearisierten Modells sind in Tabelle 4.7 dargestellt. Zu diesen gehören die in Abb. 4.20 aufgetragenen Übergangsfunktionen. Es zeigen sich nun deutlichere Unterschiede zwischen den verschiedenen Konfigurationen der Regler. Dies liegt daran, dass im linearisierten Modell die Beschränkung der Stellgröße nicht berücksichtigt wird. Aufgrund der Abweichungen zum nichtlinearen Modell aus Tabelle 4.6 sind die folgenden Ergebnisse allerdings unter Vorbehalt zu betrachten. Dennoch können auf Basis des linearisierten Modells Aussagen über das grundsätzliche Verhalten des Regelkreises, zum Beispiel über dessen Stabilität, gemacht werden.

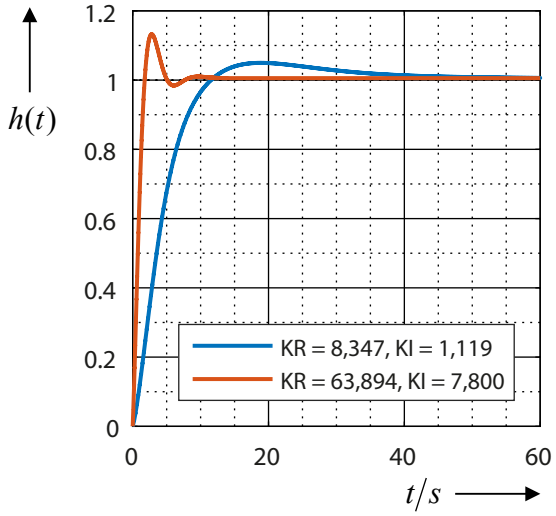
Tabelle 4.7: Eigenschaften des linearisierten geschlossenen Regelkreises.

Größe	Einheit	Kühlbetrieb		Heizbetrieb	
		$K_R = -13{,}542$, $K_I = -3{,}487$	$K_R = -76{,}720$, $K_I = -23{,}074$	$K_R = 8{,}347$, $K_I = 1{,}119$	$K_R = 63{,}894$, $K_I = 7{,}800$
Anstiegszeit T_a	s	5,74	1,12	7,26	1,30
Ausregelzeit t_ε	s	22,80	7,74	34,40	4,64
Überschwingweite $e_{\max}$	%	12,60	25,50	4,98	13,2
Amplitudenmaximum	-	1,13	1,25	1,05	1,13
Amplitudenreserve A_R	dB	29,7	14,4	∞	∞
Phasenreserve α_R	°	60,0	47,6	75,6	58,9
Stabilität	-	stabil	stabil	stabil	stabil

Die Stabilität des geschlossenen Regelkreises kann anhand der in Abb. 4.21 dargestellten Nyquist-Diagramme beurteilt werden. Sie zeigen die Ortskurven des offenen Regelkreises im Kühl- und im Heizbetrieb. Eine nähere Betrachtung des Frequenzganges in der Umgebung des Punktes -1 auf der reellen Achse zeigt, dass sämtliche in Richtung wachsender Frequenzen durchlaufenen



(a) Übergangsfunktion des geschlossenen Regelkreises im Kühlbetrieb.



(b) Übergangsfunktion des geschlossenen Regelkreises im Heizbetrieb.

Abbildung 4.20: Übergangsfunktionen der geschlossenen Regelkreise im Kühl- und im Heizbetrieb.

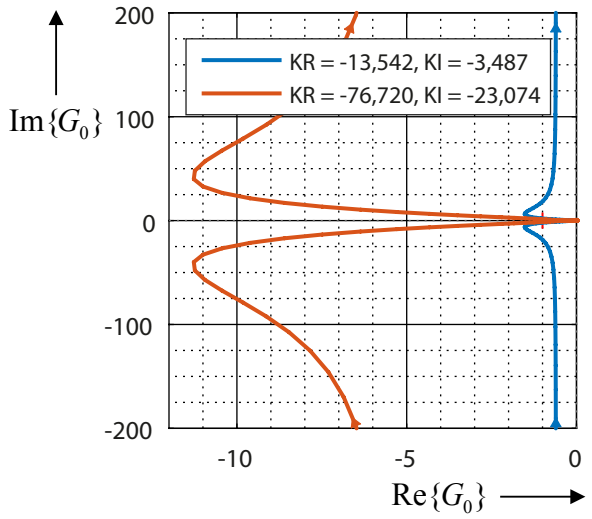
Ortskurven rechts vom Punkt -1 liegen. Aufgrund des integrierenden Verhaltens des offenen Regelkreises kann damit nach dem vereinfachten Nyquist-Kriterium auf die Stabilität der geschlossenen Regelkreise geschlossen werden.

Dies wird auch durch Betrachtung der Pol-Nullstellen-Verteilungen der geschlossenen Regelkreise in Abb. 4.22 deutlich. Da sämtliche Pole in der linken Hälfte der komplexen Ebene liegen, sind die geschlossenen Regelkreise asymptotisch stabil.

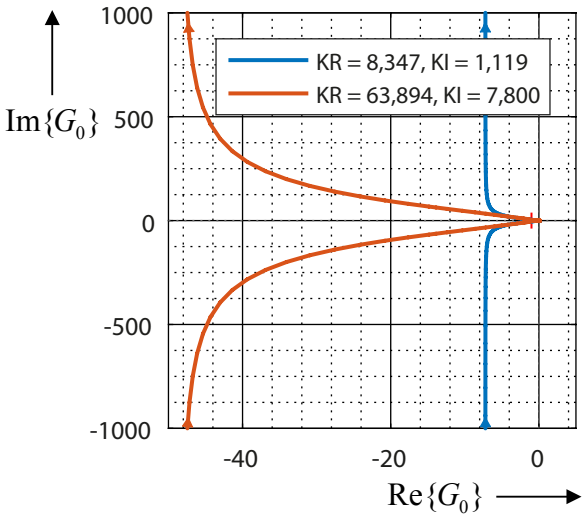
Eine Betrachtung der Bode-Diagramme der offenen Regelkreise erlaubt darüber hinaus eine quantitative Beschreibung des Stabilitätsverhaltens des geschlossenen Regelkreises durch Angabe der Amplitudenreserve A_R und der Phasenreserve α_R. Diese sind neben den zugehörigen Durchtrittsfrequenzen ω_d und ω_π in Tabelle 4.8 aufgelistet. Im Kühlbetrieb verlaufen Amplituden- und Phasengang wie in Abb. 4.23a dargestellt. Bemerkenswert ist an dieser Stelle, dass die Amplitudenreserven negative Werte annehmen. Das liegt daran, dass aufgrund der negativen Reglerparameter eine Phasenumkehr um 180° im Regler stattfindet. Der geschlossene Regelkreis ist also für beide Sätze von Reglerparametern stabil. Gleiches kann über den Regler im Heizbetrieb, dessen Bode-Diagramm in Abb. 4.23b dargestellt ist, gesagt werden. Hier schneidet der Phasengang die $-180°$-Linie nie, sodass der geschlossene Regelkreis nicht instabil werden kann.

Das integrierende Verhalten des offenen Regelkreises wird bei Betrachtung der zugehörigen Übergangsfunktionen deutlich. Diese sind in Abb. 4.24 dargestellt. Auf eine sprungförmige Anregung mit Amplitude 1 am Eingang reagiert die offene Kette mit näherungsweise linear ansteigenden Ausgangswerten.

Abschließend soll anhand der präsentierten Ergebnisse überprüft werden, ob der geschlossene Regelkreis die in Abschnitt 4.2.2 spezifizierten Güte-Anforderungen erfüllt. Betrachtet man die

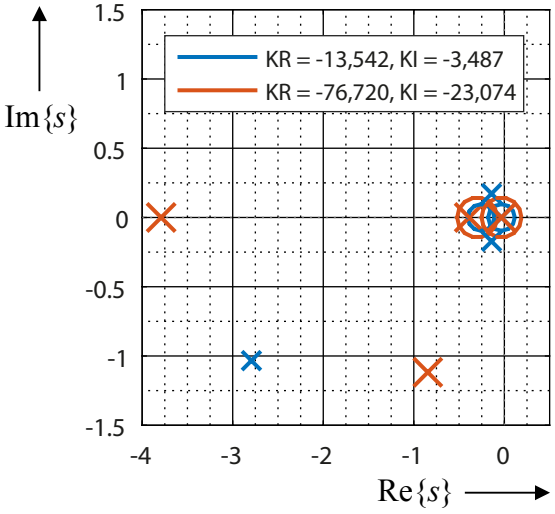


(a) Nyquist-Diagramm des offenen Regelkreises im Kühlbetrieb.

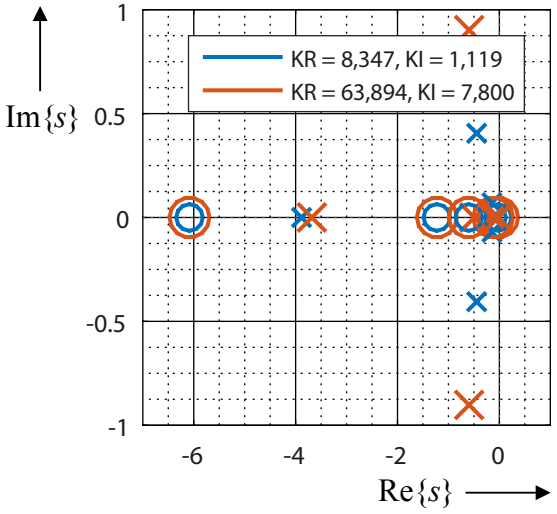


(b) Nyquist-Diagramm des offenen Regelkreises im Heizbetrieb.

Abbildung 4.21: Nyquist-Diagramme der offenen Regelkreise im Kühl- und im Heizbetrieb.

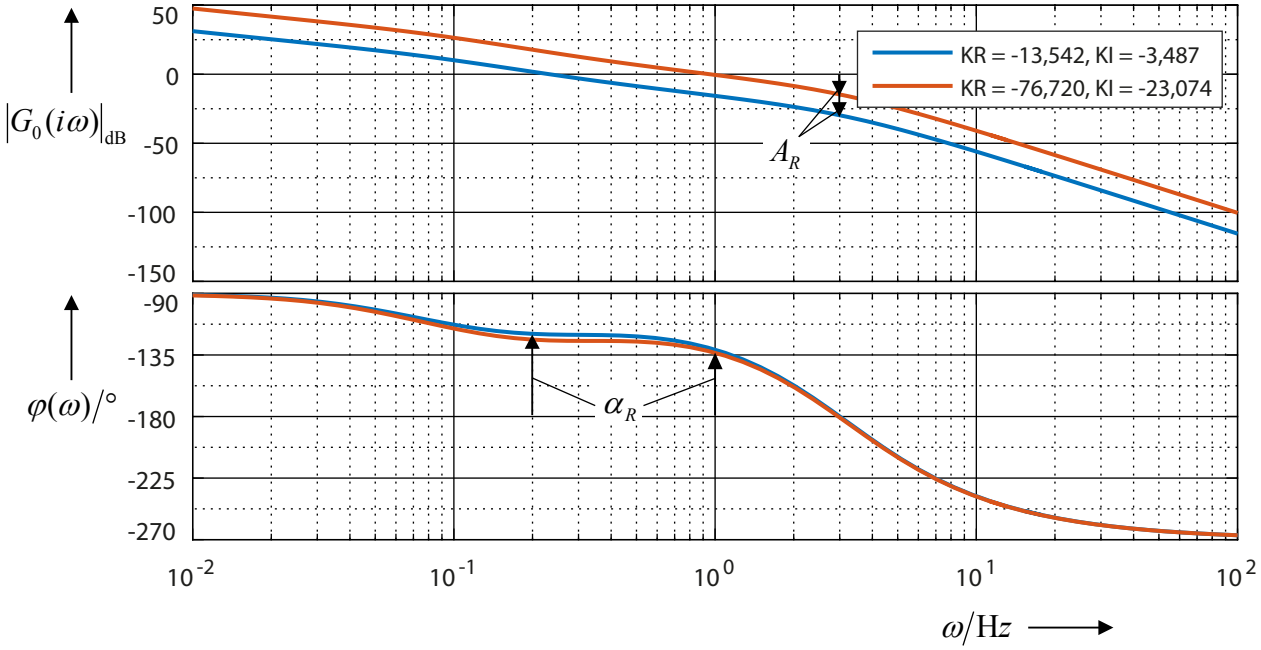


(a) Pol-Nullstellen-Verteilung des geschlossenen Regelkreises im Kühlbetrieb.

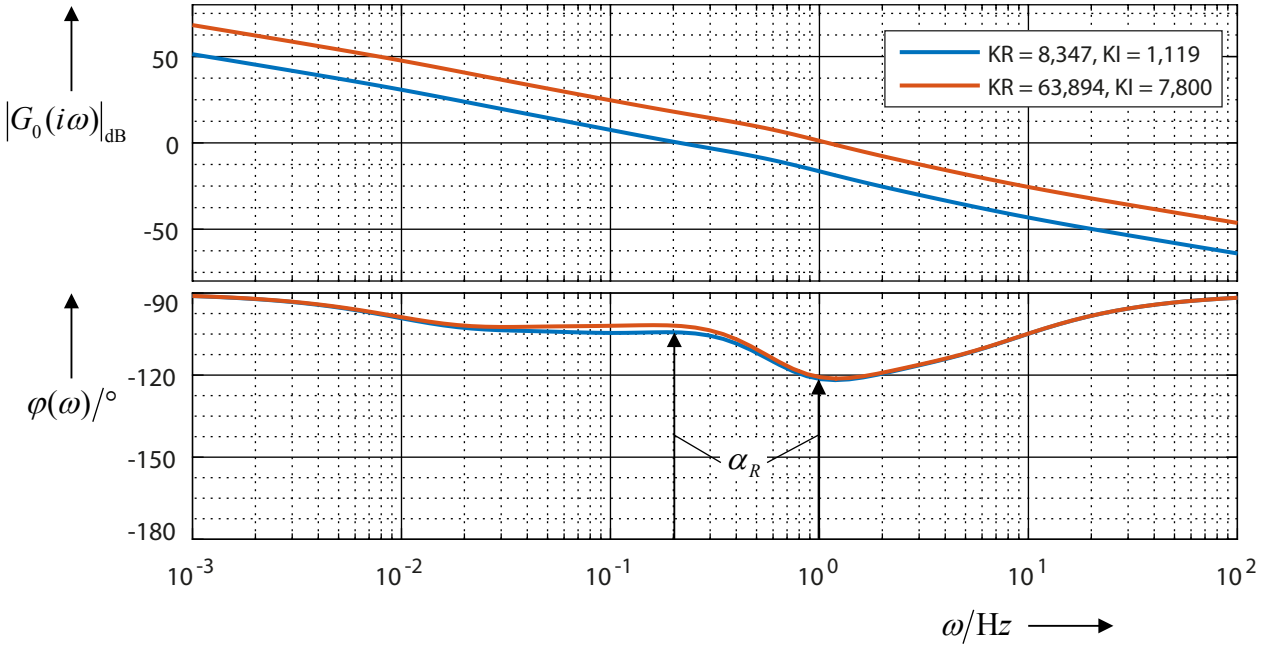


(b) Pol-Nullstellen-Verteilung des geschlossenen Regelkreises im Heizbetrieb.

Abbildung 4.22: Pol-Nullstellen-Verteilungen der geschlossenen Regelkreise im Kühl- und im Heizbetrieb.



(a) Bode-Diagramm des offenen Regelkreises im Kühlbetrieb.

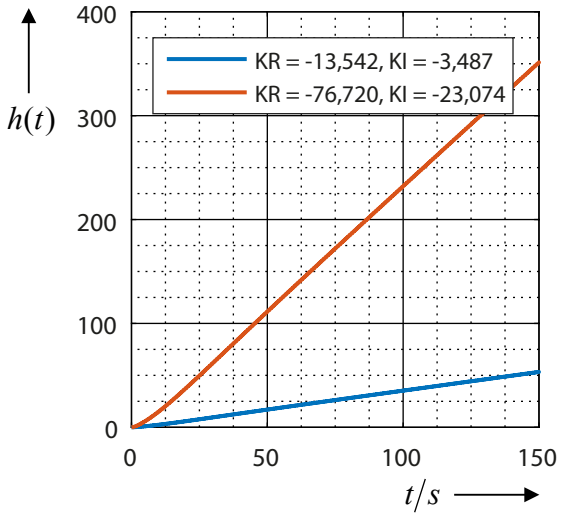


(b) Bode-Diagramm des offenen Regelkreises im Heizbetrieb.

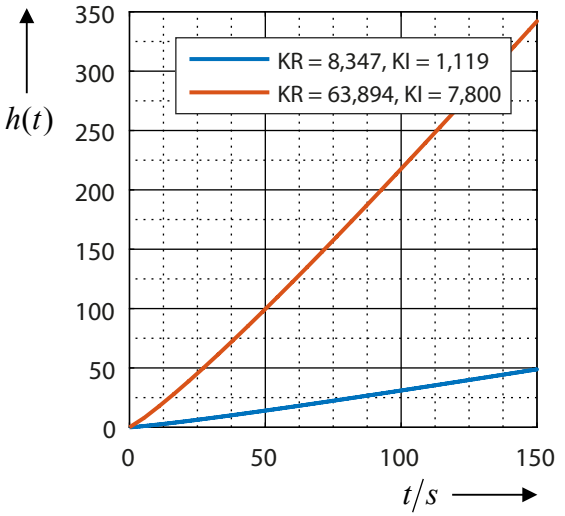
Abbildung 4.23: Bode-Diagramme der offenen Regelkreise im Kühl- und im Heizbetrieb.

Tabelle 4.8: Amplituden- und Phasenreserve des offenen Regelkreises.

Größe	Einheit	Kühlbetrieb		Heizbetrieb	
		$K_R = -13{,}542$, $K_I = -3{,}487$	$K_R = -76{,}720$, $K_I = -23{,}074$	$K_R = 8{,}347$, $K_I = 63{,}894$	$K_R = 1{,}119$, $K_I = 7{,}800$
Amplitudenreserve A_R	dB	30,589	5,248	∞	∞
Phasenreserve α_R	°	60,000	47,630	75,600	58,881
Durchtrittsfrequenz ω_d	Hz	3,003	2,962	-	-
Durchtrittsfrequenz ω_π	Hz	0,232	0,953	0,214	1,095



(a) Übergangsfunktion des offenen Regelkreises im Kühlbetrieb.



(b) Übergangsfunktion des offenen Regelkreises im Heizbetrieb.

Abbildung 4.24: Übergangsfunktionen der offenen Regelkreise im Kühl- und im Heizbetrieb.

Sprungantworten in Abb. 4.19 und die zugehörigen charakteristischen Werte in Tabelle 4.6, wird deutlich, dass alle Regelkreise statisch führungsgenau arbeiten, die bleibende Regelabweichung $e(\infty)$ also verschwindet. Die erlaubten Überschwingweiten von 25 % im Kühlbetrieb und 5 % im Heizbetrieb werden eingehalten. Bei den nach Chien, Hrones und Reswick eingestellten Reglern ist das Übergangsverhalten sogar aperiodisch. Die Unterschiede bei den Anstiegszeiten T_a und den Ausregelzeiten t_ε fallen nur gering aus, jedoch zeigen hier die nach Chien, Hrones und Reswick eingestellten Regler eine minimal höhere Dynamik. Die Voraussetzung der Stabilität wird von allen vier Regelkreisen, wie zuvor erläutert, erfüllt. Werden zur Beurteilung hingegen die linearisierten Modelle mit den Eigenschaften aus Tabelle 4.7 und den Übergangsfunktionen aus Abb. 4.20, herangezogen, lassen sich deutlichere Unterschiede in Hinblick auf Dynamik und Robustheit feststellen. Die beiden nach Chien, Hrones und Reswick eingestellten Regelkreise besitzen mit 1,12 s im Kühlbetrieb und 1,30 s im Heizbetrieb weitaus kürzere Anstiegszeiten als die mittels MATLAB PID-Tuner eingestellten Regelkreise. Gleiches gilt auch für die Ausregelzeiten.

Erkauft wird dieser Vorteil in der Dynamik durch eine etwa doppelt so große Überschwingweite, die im Kühlbetrieb mit 25,5 % bereits den zulässigen Maximalwert überschreitet. Vorteilhaft ist jedoch wiederum, dass die Überschwinger zu einem früheren Zeitpunkt auftreten als bei den mittels PID-Tuner eingestellten Regelkreisen. Es lässt sich also festhalten, dass die mittels des Verfahrens nach Chien, Hrones und Reswick eingestellten Regler eine hohe Dynamik und geringere Robustheit besitzen, während die mittels PID-Tuner eingestellten Regler sich durch eine größere Robustheit, dafür aber geringere Dynamik auszeichnen. Diese Unterschiede werden jedoch durch die Beschränkung der Stellgröße im Stellglied und damit die Nichtlinearität des Regelkreises weitgehend aufgehoben, sodass es letztlich unerheblich ist, welcher der beiden Parametersätze für den jeweiligen Regler verwendet wird. Da die Güte-Anforderungen erfüllt werden, kann die in Abschnitt 4.2.1 gestellte Regelungsaufgabe hiermit als erfolgreich gelöst betrachtet werden.

5 Entwicklung der Steuerungssoftware

Nachdem in den vorangegangenen Kapiteln die Regelstrecke modelliert und ein Temperaturregler entworfen und implementiert wurde, soll im folgenden Kapitel die Steuerungssoftware, bestehend aus einer graphischen Nutzeroberfläche und einer Firmware für den Mikrocontroller, entwickelt werden. Zu diesem Zweck werden zunächst die vom fertigen Software-System zu erfüllenden Anforderungen spezifiziert. Anschließend erfolgt die Entwicklung und Implementierung einer Schnittstelle, die den Datenaustausch zwischen GUI und Mikrocontroller ermöglichen soll. Auf Basis dieser Schnittstelle wird eine graphische Nutzeroberfläche realisiert, die es dem Nutzer ermöglicht, auf komfortable und intuitive Weise mit der Spanneinheit zu interagieren. Hierbei wird im Detail auf die Implementierung sämtlicher in der Spezifikation geforderter Funktionen eingegangen. Weiterhin befasst sich dieses Kapitel mit der Entwicklung der Mikrocontroller-Firmware, die neben dem zuvor implementierten Temperaturregler die Programmlogik des Spannsystems enthält. Den Abschluss bildet ein kurzer Praxistest der Spanneinheit.

5.1 Softwarespezifikation

Die zweiteilige Steuerungssoftware dient der Steuerung und Überwachung des Spannsystems. Sie besteht einerseits aus der graphischen Nutzeroberfläche (GUI), die auf einem PC ausgeführt wird, und andererseits einer auf dem Mikrocontroller ausgeführten Firmware. Diese hat die Aufgabe, das Spannsystem auf hardwarenaher Ebene anzusteuern, indem dort sämtliche Sensorwerte eingelesen und Ausgangsports, unter anderem zur Ansteuerung des Leistungstreibers für das Peltier-Element, entsprechend gesetzt werden. Auf dem Mikrocontroller wird zudem der Regelalgorithmus für die Temperaturregelung ausgeführt und die eigentliche Programmlogik des Spannsystems hinterlegt. Die GUI dient lediglich dem Empfangen und graphischen Darstellen der vom Mikrocontroller gesendeten Messdaten und dem Senden von Befehlen an den Mikrocontroller zur Auslösung von Aktionen, wie beispielsweise dem Einspannen oder Lösen eines Werkstücks. Der Datenaustausch zwischen Mikrocontroller und GUI erfolgt dabei über einen virtuellen COM-Port über die USB-Schnittstelle. Abschließend festzuhalten sind folgende wesentliche Anforderungen an die GUI.

- Die GUI soll primär dem Senden von Steuerbefehlen an den Mikrocontroller und dem Empfangen und graphischen Darstellen von Messdaten dienen.
- Dazu wird eine Verbindung über einen virtuellen COM-Port, der innerhalb der GUI aus einer Liste verfügbarer Ports ausgewählt werden soll, hergestellt, sobald ein Button angeklickt wurde. Der gleiche Button soll auch die Verbindung zum Mikrocontroller trennen, wobei sämtliche Parameter wieder auf Standardwerte zurückgesetzt werden sollen.

- Die Temperaturen der Spannplatte und des Wärmetauschers, die Stellgröße des Reglers sowie der Temperatursollwert der Spannplatte sollen in einem Plot über die Zeit aufgetragen werden.
- Die im Plot dargestellten Messwerte sollen in einer Textdatei geloggt werden können. Dabei wird bei jedem Systemstart eine neue Log-Datei angelegt. Benannt werden die Log-Dateien entsprechend des aktuellen Datums und der aktuellen Uhrzeit. Weiterhin soll die Logging-Funktion vom Benutzer aktiviert oder deaktiviert werden können.
- Parameter, die das Betriebsverhalten des Systems beeinflussen – darunter fallen beispielsweise die Reglerparameter des Temperaturreglers sowie Solltemperaturen in den unterschiedlichen Betriebsmodi – sollen in der GUI verändert werden können und sich möglichst direkt auf das Systemverhalten auswirken.
- Diese Parameter sollen bestimmte Standardwerte besitzen, die bei jedem Systemstart eingestellt werden. Veränderte Parametersätze sollen in Form von Profilen abgespeichert und geladen werden können. Profile sollen dabei frei benannt und bei Bedarf auch über die GUI gelöscht werden können. Beim Start der GUI sollen verfügbare Profile angezeigt werden.
- Über entsprechende Bedienelemente (Radio-Buttons) soll zunächst festgelegt werden, ob das Spannsystem mit Wachs oder mit Wassereis betrieben wird. Anschließend sollen mittels dreier Buttons die Aktionen *Spannen*, *Lösen* und *Abbrechen* auf dem Mikrocontroller ausgelöst werden.

Entsprechend sollte die Mikrocontroller-Firmware folgende Eigenschaften aufweisen.

- Die Mikrocontroller-Firmware enthält in erster Linie die Programmlogik, die das Betriebsverhalten des Spannsystems beeinflusst.
- Sämtliche in der GUI editierbare Parameter sollen vom Mikrocontroller an die GUI zurückgesendet werden. Somit ist eine hohe Datenkonsistenz zwischen GUI und Mikrocontroller-Firmware sichergestellt.
- Von der GUI gesendete Aktionen zur Festlegung der Betriebsart oder Auslösung eines Prozesses (Spannen, Lösen, Abbrechen) führen dazu, dass die Mikrocontroller-Firmware in einen bestimmten Zustand (bereit, lösen, spannen, gespannt) übergeht. In diesem werden Sollwerte für die Temperaturen festgelegt und Übergänge zwischen den einzelnen Zuständen erfolgen je nach aktuellen Temperaturen oder zeitbasiert.
- Der aktuelle Systemzustand des Spannsystems soll dabei an die GUI übertragen werden.
- Die Temperaturen des Systems sollen mithilfe des zuvor implementierten Reglers geregelt werden. Auch die Ansteuerung der H-Brücke zur Auswahl zwischen Heizen und Kühlen erfolgt in der Mikrocontroller-Firmware entsprechend des aktuellen Systemzustandes.

Eine weitere Aufbereitung der Anforderungen sowie Ausführungen zur Umsetzung erfolgen in den nachfolgenden Kapiteln.

5.2 Datenschnittstelle

Elementarer Bestandteil des Softwaresystems ist die Schnittstelle zwischen GUI und Mikrocontroller-Firmware. Im Folgenden wird zuerst auf die Schnittstellen-Hardware eingegangen, bevor eine Spezifikation der Schnittstelle ausgearbeitet wird. Anschließend werden die Schnittstelle auf dem Mikrocontroller und in der GUI implementiert und Tipps für die Entwicklung einer eigenen Steuerungssoftware als Ersatz für die GUI gegeben.

5.2.1 Schnittstellen-Hardware

Vor der Entwicklung und Implementierung eines Übertragungsprotokolls sollen zunächst die verwendete Schnittstellenhardware sowie Systemsoftware näher betrachtet werden. Diese ist in Abb. 5.1 schematisch dargestellt. Das Steuerprogramm in Form der graphischen Nutzeroberfläche wird als Anwendungsprogramm auf einem Windows PC ausgeführt, während das Spannsystem von einem Mikrocontroller (AtMega 328P) angesteuert wird. Dieser besitzt einen UART (Universal Asynchronous Receiver Transmitter), welcher der Realisierung einer seriellen Schnittstelle dient. Über den UART können Daten asynchron, also zu beliebigen Zeiten, gesendet und empfangen werden. Um den UART des Mikrocontrollers mit der USB-Schnittstelle des PCs verbinden zu können, ist ein Wandler-IC erforderlich. Hierfür kommt ein FTDI FT232R (Datenblatt siehe [26]) zum Einsatz, der mit dem Mikrocontroller über das serielle Übertragungsprotokoll kommuniziert und den Datenaustausch mit der USB-Schnittstelle des PCs organisiert. Am USB Root Hub eingehende Daten werden vom USB Host Controller empfangen und an den USB-Treiber weitergeleitet. Diesem ist ein virtueller COM-Port Treiber (VCP) vorgelagert, der das an den USB-Anschluss angeschlossene Gerät als virtuelle EIA RS-232C-Schnittstelle emuliert. Das Anwenderprogramm, in diesem Fall die GUI, behandelt das USB-Gerät folglich wie ein über RS-232 an den PC angeschlossenes Gerät. Diese Schichtenstruktur hat also die Funktion, spezifische Gegebenheiten der USB-Schnittstelle vor dem Anwendungsprogramm und der Mikrocontroller-Firmware zu verbergen. Dadurch vereinfacht sich die Software-Entwicklung, da GUI und Mikrocontroller aus Sicht des Programmierers direkt über eine RS-232 Schnittstelle verbunden sind. [26, 38]

5.2.2 Spezifikation der Schnittstelle

Der bidirektionale Datenaustausch zwischen GUI und Mikrocontroller erfolgt nach dem in Abb. 5.2 schematisch dargestellten Muster. Daten werden von der GUI immer dann an den Mikrocontroller gesendet, wenn der Nutzer eine Interaktion mit der grafischen Oberfläche, wie zum Beispiel einen Button-Klick, durchführt. Der Mikrocontroller durchläuft eine Programmschleife, innerhalb derer er diese Daten empfängt. Unabhängig hiervon erfolgt der Datenaustausch vom Mikrocontroller zur GUI in einem festen Zeitintervall, dass der Abtastzeit T des Reglers entspricht. Daten werden vom Mikrocontroller in diesen Zeitabständen gesendet und von der GUI eingelesen, sobald sie dort ankommen. Im Folgenden muss also zwischen zwei Richtungen des Datenstroms unterschieden werden, zum einen von der GUI zum Mikrocontroller und zum

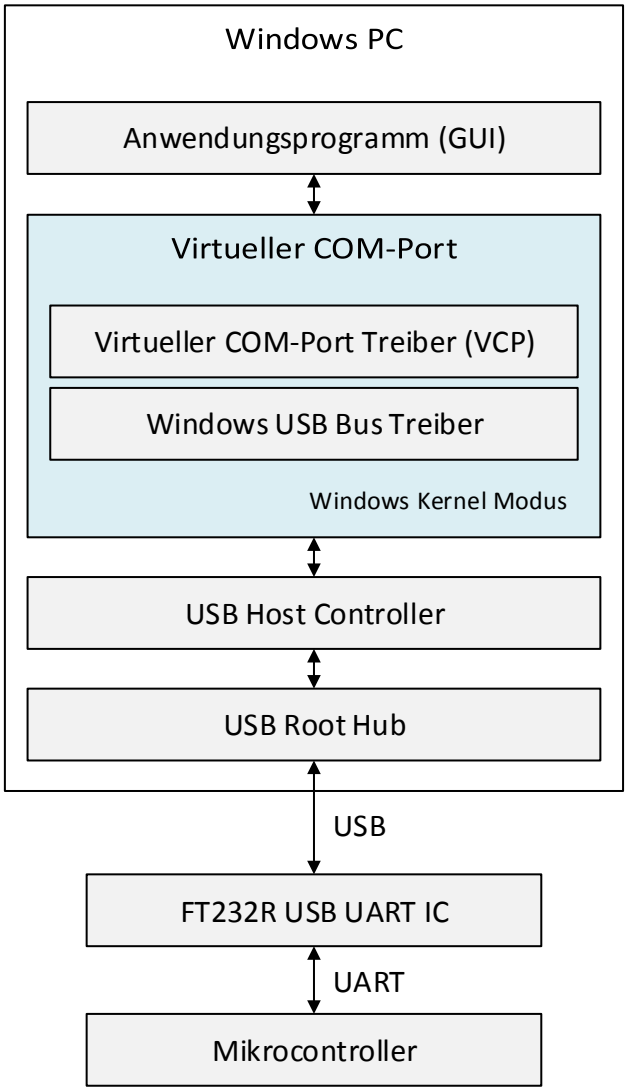


Abbildung 5.1: Schichtenstruktur der Schnittstellenhard- und -software.

anderen vom Mikrocontroller zur GUI. Eine gegenseitige Beeinflussung der beiden Datenströme tritt jedoch nicht auf.

Vor der Implementierung der Schnittstelle sollen zunächst die übertragenen Daten und ihre praktische Bedeutung näher betrachtet werden. Tabelle 5.1 enthält hierzu in übersichtlicher Form alle Variablennamen, Datentypen, Werte(bereiche) sowie die Bedeutung der einzelnen Daten. Die Übertragungsrichtung kann Abb. 5.3 entnommen werden. Grün dargestellte Daten werden dabei nur von der GUI zum Mikrocontroller, rot dargestellte nur vom Mikrocontroller zur GUI und gelb dargestellte in beide Richtungen übertragen.

Die Anordnung der Daten erfolgt zweckmäßigerweise in Form einer ASCII-codierten Zeichenkette, innerhalb derer die einzelnen Variablen durch Kommata getrennt sind. Das Ende der Zeichenkette wird durch das *Newline*-Steuerzeichen „\n“, den sogenannten Terminator, markiert. Zeichenkette und Terminator bilden zusammen ein Datenpaket. Durch Verwendung des Terminators kann

Tabelle 5.1: Beschreibung der zwischen GUI und Mikrocontroller übertragenen Daten.

Name	Datentyp	Einheit	Wertebereich	Beschreibung
Betriebsmodus	int	-	0	Betrieb mit Wachs
			1	Betrieb mit Wasser
Zustand	int	-	0	System bereit
			1	Werkstück wird gespannt
			2	Werkstück wird gelöst
			3	Werkstück eingespannt
Aktion	int	-	0	keine Aktion
			1	Werkstück spannen
			2	Werkstück lösen
			3	Spann-/Lösevorgang abbrechen
Zeit	ulong int	ms	$0 \dots \infty$	Zeit seit Systemstart
T1_SOLL	double	°C	$-15.0 \dots 75.0$	Solltemperatur der Spannplatte
T1	double	°C	$-15.0 \dots 75.0$	Temperatur der Spannplatte
T2	double	°C	$-15.0 \dots 75.0$	Temperatur des Wärmetauschers
TEC_CURRENT	double	%	$0 \dots 100$	Stellgröße des Reglers
KP_HEIZEN	double	-	$0 \dots \infty$	Proportionalterm des Heiz-Reglers
KI_HEIZEN	double	-	$0 \dots \infty$	Integralterm des Heiz-Reglers
KP_KUEHLEN	double	-	$-\infty \dots 0$	Proportionalterm des Kühl-Reglers
KI_KUEHLEN	double	-	$-\infty \dots 0$	Integralterm des Kühl-Reglers
T_WACHS_SCHMELZ	double	°C	$-15.0 \dots 75.0$	Obere Grenztemperatur im Heizbetrieb
T_WACHS_ERSTARR	double	°C	$-15.0 \dots$ T_WACHS_SCHMELZ	Erstarrungstemperatur des Wachses
T_WACHS_ABKUEHL	double	°C	$-15.0 \dots$ T_WACHS_ERSTARR	Solltemperatur beim Abkühlen nach dem Aufschmelzen
T_WASSER_FRIER	double	°C	$-15.0 \dots 75.0$	Untere Grenztemperatur im Kühlbetrieb
T_WASSER_ERSTARR	double	°C	T_WASSER_FRIER $\dots 75.0$	Erstarrungstemperatur von Wasser
DAUER_SCHMELZEN	ulong int	ms	$0 \dots \infty$	Zeitdauer der Wachsverflüssigung beim Spannen/Lösen im Betrieb mit Wachs
DAUER_ABKUEHLEN	ulong int	ms	$0 \dots \infty$	Zeitdauer des Abkühlens nach dem Spannen/Lösen im Betrieb mit Wachs

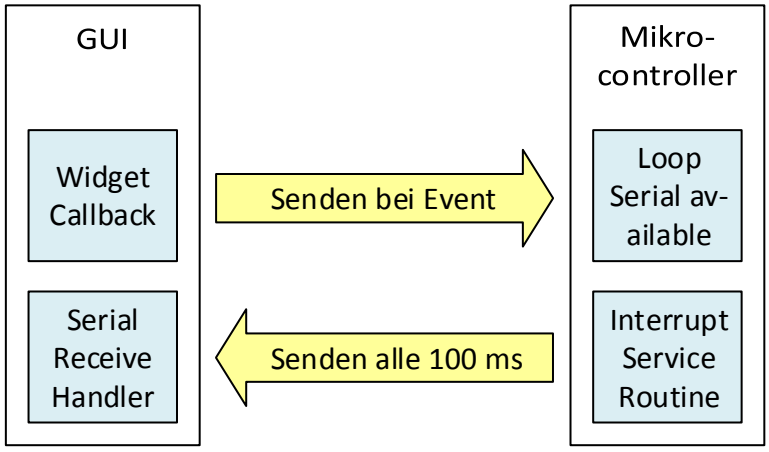


Abbildung 5.2: Schema des Datenaustausches zwischen GUI und Mikrocontroller.

der Empfänger auf einfache Art und Weise erkennen, wann ein Datenpaket endet und ein neues beginnt.

Je nach Richtung des Datenstroms unterscheiden sich die gesendeten Pakete, da sie nur die in Abb. 5.3 gezeigten Daten enthalten müssen. Somit versendet der Mikrocontroller das in Listing 5.1 gezeigte Datenpaket an die GUI, während die GUI das in Listing 5.2 dargestellte Paket an den Mikrocontroller zurückschickt.

Listing 5.1: Vom Mikrocontroller an die GUI gesendetes Datenpaket.

```
Betriebsmodus,Zustand,Zeit,T1_SOLL,T1,T2,TEC_CURRENT,KP_HEIZEN,KI_HEIZEN,KP_KUEHLEN,KI_KUEHLEN,
T_WACHS_SCHMELZ,T_WACHS_ERSTARR,T_WACHS_ABKUEHL,T_WASSER_FRIER,T_WASSER_ERSTARR,DAUER_SCHMELZEN
,DAUER_ABKUEHLEN\n
```

Listing 5.2: Von der GUI an den Mikrocontroller gesendetes Datenpaket.

```
Betriebsmodus,Aktion,KP_HEIZEN,KI_HEIZEN,KP_KUEHLEN,KI_KUEHLEN,T_WACHS_SCHMELZ,T_WACHS_ERSTARR,
T_WACHS_ABKUEHL,T_WASSER_FRIER,T_WASSER_ERSTARR,DAUER_SCHMELZEN,DAUER_ABKUEHLEN\n
```

5.2.3 Implementierung der Schnittstelle im Mikrocontroller

Listing 5.3 zeigt die Implementierung der Schnittstelle im Mikrocontroller. Dabei wird auf die Methoden der Arduino-Klasse *Serial* zurückgegriffen. Zunächst werden in der Setup-Funktion die serielle Schnittstelle mit einer Baud-Rate von 115 200 Bd initialisiert und der Timer 1 des Mikrocontrollers so konfiguriert, dass er alle 100 ms einen Interrupt auslöst. Dieser Interrupt ruft eine Interrupt-Routine auf, innerhalb derer ein Status-Flag gesetzt wird. Das gesetzte Flag führt dazu, dass in der Hauptprogrammschleife *loop()* die Sende-Funktion *serialSendData()* aufgerufen und somit das in Listing 5.1 spezifizierte Datenpaket an den COM-Port gesendet wird. Hierzu werden zunächst die einzelnen Variablen in der String-Variable *send_string* gespeichert und anschließend mit der Klassenmethode *println()* die Zeichenkette inklusive Terminator an die serielle Schnittstelle gesendet.

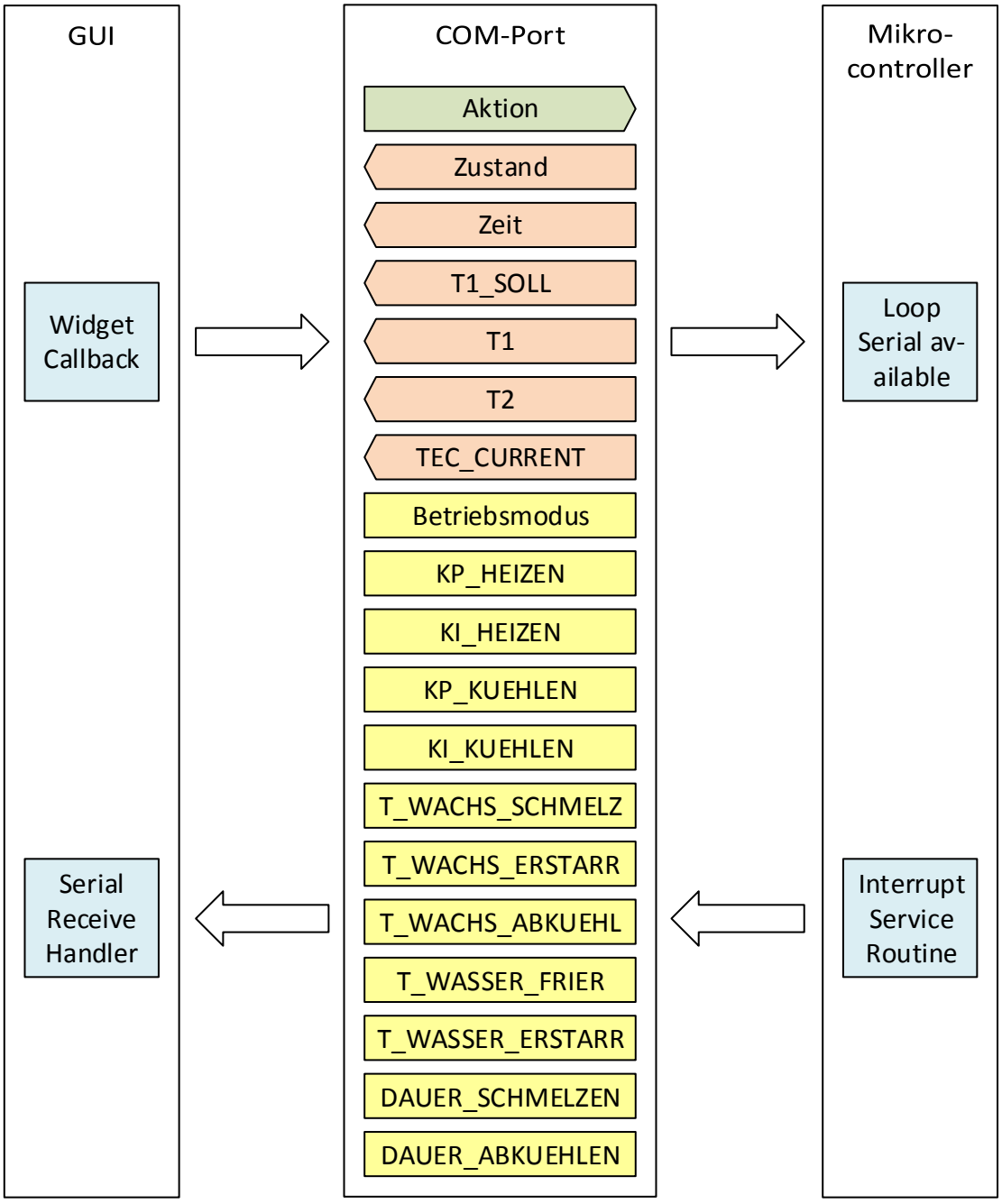


Abbildung 5.3: Schema der Schnittstelle zwischen GUI und Mikrocontroller. Grün: von der GUI gesendete Daten, Rot: von der GUI empfangene Daten, Gelb: bidirektional ausgetauschte Daten.

Der Datenempfang erfolgt ebenfalls in der Hauptprogrammschleife, jedoch nicht zeitgesteuert, sondern immer dann, wenn Daten an der seriellen Schnittstelle ankommen. Dies kann durch die Klassenmethode *available()* geprüft werden, die die Größe der im Empfangs-Buffer liegenden und lesebereiten Daten angibt. Liegen keine Daten vor, wird die gesamte while-Schleife übersprungen. Andernfalls werden solange Daten eingelesen, bis der Empfangs-Buffer leer ist. Das Einlesen in die einzelnen Variablen erfolgt dabei mithilfe der Klassenmethoden *parseInt()* und *parseFloat()*, die einzelne Zahlenwerte zwischen den Kommata aus der Zeichenkette extrahieren.

Listing 5.3: Implementierung der Schnittstelle im Mikrocontroller.

```
String send_string = String();
static volatile uint8_t isr_active_flag = 0;

void setup()
{
  // serielle Schnittstelle initialisieren
  Serial.begin(115200);
  Serial.setTimeout(10);

  // Einstellung von Timer1, Aufruf der ISR alle 100 ms
  cli();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = ((16.0/256.0)*100000)-1.0;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12);
  TIMSK1 |= (1 << OCIE1A);
  sei();
}

// ISR-Routine
ISR(TIMER1_COMPA_vect)
{
  isr_active_flag = 1;
}

void serialSendData()
{
  // Variablen zu einem String zusammenführen
  send_string += Betriebsmodus;
  send_string += ,;
  send_string += Zustand;
  send_string += ,;
  send_string += millis(); // Zeit
  send_string += ,;
  send_string += T1_SOLL;
  send_string += ,;
  send_string += T1;
  send_string += ,;
  send_string += T2;
  send_string += ,;
  send_string += adcToPercent(TEC_CURRENT); // Ausgabe in Prozent
```

```
  send_string += ,;
  send_string += KP_HEIZEN;
  send_string += ,;
  send_string += KI_HEIZEN;
  send_string += ,;
  send_string += KP_KUEHLEN;
  send_string += ,;
  send_string += KI_KUEHLEN;
  send_string += ,;
  send_string += T_WACHS_SCHMELZ;
  send_string += ,;
  send_string += T_WACHS_ERSTARR;
  send_string += ,;
  send_string += T_WACHS_ABKUEHL;
  send_string += ,;
  send_string += T_WASSER_FRIER;
  send_string += ,;
  send_string += T_WASSER_ERSTARR;
  send_string += ,;
  send_string += DAUER_SCHMELZEN;
  send_string += ,;
  send_string += DAUER_ABKUEHLEN;

  // Datenpaket an die serielle Schnittstelle senden
  Serial.println(send_string);

  send_string = ;
}

void loop()
{
  // Einlesen von Datenpaketen, sobald sie am Port ankommen
  while (Serial.available() > 0)
  {
    Betriebsmodus = Serial.parseInt();
    Aktion = Serial.parseInt();
    KP_HEIZEN = Serial.parseFloat();
    KI_HEIZEN = Serial.parseFloat();
    KP_KUEHLEN = Serial.parseFloat();
    KI_KUEHLEN = Serial.parseFloat();
    T_WACHS_SCHMELZ = Serial.parseFloat();
    T_WACHS_ERSTARR = Serial.parseFloat();
    T_WACHS_ABKUEHL = Serial.parseFloat();
    T_WASSER_FRIER = Serial.parseFloat();
    T_WASSER_ERSTARR = Serial.parseFloat();
    DAUER_SCHMELZEN = Serial.parseInt();
    DAUER_ABKUEHLEN = Serial.parseInt();

    if (Serial.read() == '\n'); // Warten, bis gesamte Zeile gelesen wurde
  }

  // Senden von Datenpaketen alle 100 ms
  if(isr_active_flag)
  {
```

```
98      isr_active_flag = 0;
99      serialSendData();
100   }
101
102   // restliche Programmlogik folgt hier...
103 }
```

5.2.4 Implementierung der Schnittstelle in der GUI

Die Schnittstelle auf Seiten der GUI ist entsprechend Listing 5.4 implementiert und basiert auf den Methoden der MATLAB-Klasse *serial*. Beim Start der GUI wird eine Variable *s* angelegt, der beim Klick auf den *Verbinden*-Button ein serielles Port-Objekt zugewiesen wird. Über das Attribut *BytesAvailableFcnMode* wird festgelegt, dass der in *BytesAvailableFcn* spezifizierte Event-Handler *serial_read_data()* immer dann aufgerufen wird, wenn an der seriellen Schnittstelle das Terminatorzeichen eingelesen wurde, also ein neues Datenpaket angekommen ist. Innerhalb des Event-Handlers wird das gesamte Datenpaket als ASCII-Zeichenkette eingelesen und mithilfe des Befehls *eval()* in ein Array aus Zahlenwerten umgewandelt. Die einzelnen Werte können nun den Textboxen zugewiesen oder geplottet werden. Da die Befehle innerhalb eines *try*-Blocks stehen, führt ein fehlerhaft eingelesenes Datenpaket nicht direkt zum Absturz des Programms, sondern stattdessen wird eine Fehlermeldung im *catch*-Block ausgegeben und das Einlesen beim nächsten ankommenden Datenpaket fortgesetzt.

Das Senden von Daten erfolgt bei Auslösung eines Widget-Callbacks, hier beispielweise ein Klick auf die Radio-Box *Betrieb mit Wachs*. Zunächst wird dort der Wert der entsprechenden Variable geändert und anschließend die Funktion *serial_send_commands()* ausgeführt. Innerhalb dieser wird ein String entsprechend Listing 5.2 zusammengestellt und mittels *fprintf()* auf den seriellen Port geschrieben.

Listing 5.4: Implementierung der Schnittstelle in der GUI.

```
1  % --- Executes just before Controller_GUI is made visible.
2  function Controller_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
3  handles.s = 0; % initialise serial port object
4  % Update handles structure
5  handles.output = hObject;
6  guidata(hObject, handles);
7
8
9  % --- Executes on button press in Connect_togglebutton.
10 function Connect_togglebutton_Callback(hObject, eventdata, handles)
11 button_state = get(hObject,'Value');
12
13 % on click of 'Verbinden'
14 if button_state == get(hObject,'Max')
15  % delete any existing port objects
16     delete(instrfindall);
17     % create serial port object
18     handles.s = serial(handles.portSelected,'BaudRate',115200);
19     fclose(handles.s);
```

```
20      % specify callbackfunction, that is executed everytime the terminator '\n' is read on
        serial port
21      handles.s.BytesAvailableFcnMode = 'terminator';
22      handles.s.BytesAvailableFcn = {@serial_read_data, hObject, handles};
23      fopen(handles.s);
24
25  % on click of 'Trennen'
26  elseif button_state == get(hObject,'Min')
27      % Clean up serial port if connected
28      if handles.s ~= 0
29          fclose(handles.s);
30          delete(handles.s);
31          clear handles.s;
32          delete(instrfindall);
33      end
34      handles.s = 0;
35  end
36
37  % Update handles structure
38  guidata(hObject, handles);
39
40
41  % ---- Executes on button press in Heizen_radiobutton.
42  function Heizen_radiobutton_Callback(hObject, eventdata, handles)
43  handles.Betriebsmodus = 0; % Heizen
44  % send command to serial
45  serial_send_commands(handles);
46  % Update handles structure
47  guidata(hObject, handles);
48
49
50  % ---- Writes commands to serial port
51  function serial_send_commands(handles)
52  if handles.s ~= 0
53      string = [num2str(handles.Betriebsmodus),',',num2str(handles.Aktion)...
54          ,',',num2str(handles.KP_HEIZEN),',',num2str(handles.KI_HEIZEN)...
55          ,',',num2str(handles.KP_KUEHLEN),',',num2str(handles.KI_KUEHLEN)...
56          ,',',num2str(handles.T_WACHS_SCHMELZ),',',num2str(handles.T_WACHS_ERSTARR)...
57          ,',',num2str(handles.T_WACHS_ABKUEHL),',',num2str(handles.T_WASSER_FRIER)...
58          ,',',num2str(handles.T_WASSER_ERSTARR),',',num2str(handles.DAUER_SCHMELZEN)...
59          ,',',num2str(handles.DAUER_ABKUEHLEN)];
60      fprintf(handles.s,string);
61  end
62
63
64  % ---- Event Handler for Reading data from serial port
65  function serial_read_data(s, event, hObject, handles)
66  % read data from serial port
67  serial_incoming_data = fscanf(s,'%s');
68  try
69      % parse incoming data
70      converted_data = eval(['[',serial_incoming_data,']']);
71      % update variables based on data
72      handles.Betriebsmodus = converted_data(1);
```

```
73      handles.Zustand = converted_data(2);
74      converted_data(3) = converted_data(3)/1000; % change time format into seconds
75      handles.Zeit = converted_data(3);
76      handles.T1_SOLL = converted_data(4);
77      handles.T1 = converted_data(5);
78      handles.T2 = converted_data(6);
79      handles.TEC_CURRENT = converted_data(7);
80      handles.KP_HEIZEN = converted_data(8);
81      handles.KI_HEIZEN = converted_data(9);
82      handles.KP_KUEHLEN = converted_data(10);
83      handles.KI_KUEHLEN = converted_data(11);
84      handles.T_WACHS_SCHMELZ = converted_data(12);
85      handles.T_WACHS_ERSTARR = converted_data(13);
86      handles.T_WACHS_ABKUEHL = converted_data(14);
87      handles.T_WASSER_FRIER = converted_data(15);
88      handles.T_WASSER_ERSTARR = converted_data(16);
89      handles.DAUER_SCHMELZEN = converted_data(17);
90      handles.DAUER_ABKUEHLEN = converted_data(18);
91      % if no execption occurs change status to 'Okay'
92      set(handles.Status_text,'String','Status: Daten wurden erfolgreich empfangen');
93  catch
94      % if execption occurs change status to 'Fehler'
95      set(handles.Status_text,'String','Status: Fehler beim Empfangen der Daten');
96  end
97
98  % Update handles structure
99  guidata(hObject, handles);
```

5.2.5 Entwicklung einer eigenen Steuerungssoftware

Prinzipiell kann die GUI durch jede beliebige Software ersetzt werden, sofern die Schnittstelle entsprechend Abschnitt 5.2.4 angesprochen wird. Die Software muss folglich in der Lage sein, ankommende Daten am seriellen Port zu erkennen, diese einzulesen und weiterzuverarbeiten. Die Erkennung eines abgeschlossenen Datenpakets kann dabei beispielsweise anhand des Terminatorzeichens „\n“ erfolgen. Das Senden von Daten an den Mikrocontroller kann zu beliebigen Zeitpunkten vorgenommen werden, wobei die Struktur der versendeten Daten der Zeichenkette aus Listing 5.2 entsprechen muss. Die Daten selbst müssen hinsichtlich Datentyp und Wertebereich mit den Angaben in Tabelle 5.1 übereinstimmen. Eine Limitierung ergibt sich durch die verhältnismäßig geringe Rechenleistung des Mikrocontrollers. Befehle dürfen daher maximal so schnell hintereinander an den Mikrocontroller gesendet werden, dass diesem genügend Zeit zur Verarbeitung und auch zum Versenden eigener Datenpakete alle 100 ms bleibt. Andererseits ist sicherzustellen, dass die Software ausreichend schnell genug ist, um ankommende Datenpakete in diesem Zeitintervall zu empfangen und weiterzuverarbeiten. Zudem sollte ein unerwünschtes Verhalten der Software bei fehlerhaft eingelesenen Datenpaketen vermieden werden. Denn insbesondere direkt nach dem Aufbau der Verbindung über die serielle Schnittstelle stehen oft beliebige Zeichen im Sende-Buffer, die zunächst verschickt werden und ohne entsprechende Maßnahmen zum Absturz der Software führen.

5.3 Graphische Benutzeroberfläche

Im Folgenden sollen kurz der Aufbau und die verschiedenen Funktionen der graphischen Nutzeroberfläche beschrieben werden. An wichtigen Stellen wird dabei auf die Implementierung der Funktionen eingegangen. Der gesamte Quellcode der GUI ist in Anhang A.5 zu finden.

Wie in der Gesamtansicht in Abb. 5.4 zu erkennen, gliedert sich die GUI in mehrere Panels auf. Diese dienen zur Einstellung des Ports (1), zur Durchführung von Aktionen (2), zur Einstellung von Systemparametern (3) und Speicherung von Parameter-Profilen (4) sowie schließlich zur Ausgabe von Messdaten (5 und 6). Die Messdaten werden weiterhin in einem dynamischen Plot (7) dargestellt. Auch das Loggen von Messdaten ist möglich (8). In der Statusleiste (9) wird der aktuelle Verbindungszustand angezeigt.

5.3.1 Initialisierung der GUI

Sämtliche in der GUI angezeigte Parameter und Variablen benötigen eine Initialisierung, das heißt, es muss eine Zuweisung von Standardwerten vor der Ausführung der GUI erfolgen. Hierzu wird, wie in Listing 5.5 zu sehen, die Funktion *Controller_GUI_OpeningFcn()* aufgerufen, innerhalb derer sämtliche Variablen in einer Struktur mit dem Namen *handles* angelegt und mit Standardwerten beschrieben werden. Der Aufruf der Funktion erfolgt dabei vor dem Anzeigen des GUI-Fensters. Zudem wird innerhalb dieser Funktion der dynamische Plot in Form eines Grafikobjektes vom Typ *axes* erstellt. Da zwei Hochachsen mit verschiedenen Skalierungen, einmal für die Temperaturen in Grad Celsius und einmal für die Stellgröße in Prozent, benötigt werden, werden hier zwei Plots mit entsprechend positionierten und skalierten Achsen übereinander gelegt. Die Zeitachsen der beiden Plots sind dabei mittels des Befehls *linkaxes()* synchronisiert. Bei der Erstellung des Plots werden weiterhin die anzeigten Kurven als *animatedline*-Objekte sowie eine Legende zur Beschreibung definiert. Ebenfalls Teil der Initialisierungsfunktion ist die Erstellung von Verzeichnissen für die Log-Dateien und Profil-Dateien, sofern diese noch nicht vorhanden sind. Abschließend wird die innerhalb der Funktion veränderte handles-Struktur mit dem Befehl *guidata()* aktualisiert und damit werden die Änderungen global verfügbar gemacht.

Listing 5.5: Initialisierungsfunktion der GUI.

```
1  % --- Executes just before Controller_GUI is made visible.
2  function Controller_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
3  % Setup default values for parameters
4  handles = declareDefaults(handles);
5
6  % miscellaneous
7  handles.s = 0; % serial port
8  handles.logFileName = ['logging\','Messung_',datestr(now,'dd-mmm-yyyy_HH-MM-SS'),'.txt']; %
   initial filename for logfile
9
10 % axes setup
11 xlabel(handles.Ausgabeplot,'Zeit t in s');
12 ylim([handles.T_MIN-5.0 handles.T_MAX+5.0]);
13 set(handles.Ausgabeplot,'Box','off');
```

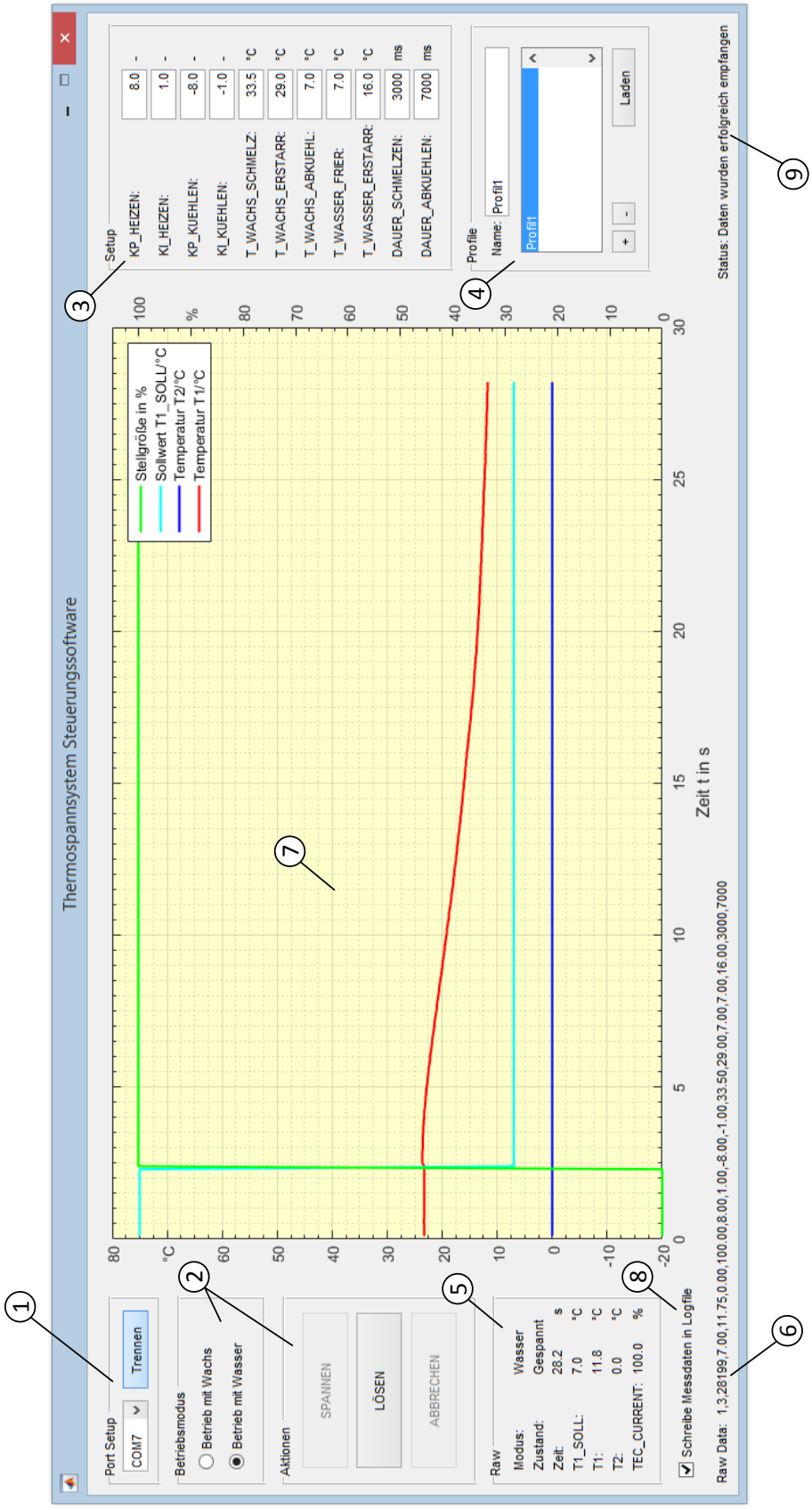


Abbildung 5.4: Gesamtansicht der graphischen Oberfläche der Steuerungssoftware.

```
14 % create new axes on top with y axis on the right
15 handles.Ausgabeplot2 = axes('Units','character');
16 set(handles.Ausgabeplot2,'Position',get(handles.Ausgabeplot,'Position'),'Color','none','YLim'
   ,[0 105],'YAxisLocation','right','XAxisLocation','top','XTickLabel',[],'Box','off','XMinorTick'
   ,'on','YMinorTick','on');
17 linkaxes([handles.Ausgabeplot,handles.Ausgabeplot2],'x'); % link x axis of both axes
18 % define plot lines for usage in callback-function
19 handles.h1 = animatedline('Parent',handles.Ausgabeplot,'Color','r','LineWidth',1.5); % T1
20 handles.h2 = animatedline('Parent',handles.Ausgabeplot,'Color','b','LineWidth',1.5); % T2
21 handles.h3 = animatedline('Parent',handles.Ausgabeplot,'Color','c','LineWidth',1.5); % T1_SOLL
22 handles.h4 = animatedline('Parent',handles.Ausgabeplot2,'Color','g','LineWidth',1.5); %
   TEC_CURRENT
23 % legend setup
24 legend_handle = legend([handles.h4;handles.h3;handles.h2;handles.h1],'Stellgröße in %','
   Sollwert T1\_SOLL/°C','Temperatur T2/°C','Temperatur T1/°C');
25 set(legend_handle,'color','w');
26 % change second last y axis tick to unit symbol
27 Myl = get(handles.Ausgabeplot,'YTickLabel'); % put left y-axis ticks into matrix
28 Myl{length(Myl)-1} = '°C';
29 set(handles.Ausgabeplot,'YTickLabel',Myl); % set new axis ticks with units inserted
30 Myr = get(handles.Ausgabeplot2,'YTickLabel');
31 Myr{length(Myr)-1} = '%';
32 set(handles.Ausgabeplot2,'YTickLabel',Myr);
33
34 % create profiles folder if not already there
35 if ~exist('profiles','dir')
36     mkdir('profiles');
37 end
38
39 % create logging folder if not already there
40 if ~exist('logging','dir')
41     mkdir('logging');
42 end
43
44 % Choose default command line output for Controller_GUI
45 handles.output = hObject;
46
47 % Update handles structure
48 guidata(hObject, handles);
49
50
51 % --- Creates handles structure with default values, can be used for resetting to defaults
52 function handles = declareDefaults(handles)
53 % create user data in handle object
54 handles.Betriebsmodus = 1;
55 handles.Zustand = 0;
56 handles.Aktion = 0;
57 handles.Zeit = 0;
58 handles.T1 = 0;
59 handles.T2 = 0;
60 handles.TEC_CURRENT = 0;
61 handles.T1_SOLL = 0;
62 % constants for system setup
63 handles.KP_HEIZEN = 63.89;
```

```
64 handles.KI_HEIZEN = 7.80;
65 handles.KP_KUEHLEN = -13.54;
66 handles.KI_KUEHLEN = -3.49;
67 handles.T_WACHS_SCHMELZ = 70.0;
68 handles.T_WACHS_ERSTARR = 60.0;
69 handles.T_WACHS_ABKUEHL = -15.0;
70 handles.T_WASSER_FRIER = -15.0;
71 handles.T_WASSER_ERSTARR = 0.0;
72 handles.DAUER_SCHMELZEN = 8000;
73 handles.DAUER_ABKUEHLEN = 15000;
74 % constant temperature limits (just for checking input values)
75 handles.T_MIN = -15.0;
76 handles.T_MAX = 75.0;
```

5.3.2 Verbindungsaufbau zum Mikrocontroller

Wie zuvor beschrieben, kommunizieren GUI und Mikrocontroller-Firmware über einen virtuellen COM-Port miteinander. Dieser wird neben allen anderen im System verfügbaren COM-Ports im Popup-Menü links neben dem Verbinden/Trennen-Button aufgeführt. Existieren mehrere Ports, muss zunächst der entsprechende COM-Port ausgewählt und mit einem Klick auf *Verbinden*, wie in Abb. 5.5a gezeigt, eine Verbindung aufgebaut werden. Ist die Verbindung erfolgreich etabliert, so ändert sich die Beschriftung des Buttons, wie in Abb. 5.5b zu sehen, in *Trennen*. Der erfolgreiche Aufbau der Verbindung ist außerdem an einer entsprechenden Ausgabe innerhalb der Statusleiste sowie an der Darstellung von Messdaten im *Raw*-Panel und im Plot zu erkennen. Die anfänglich ausgegrauten Bedienelemente zur Steuerung des Systems und die Edit-Boxen zur Änderung der Systemparameter sind ebenfalls nur bei bestehender Verbindung mit dem COM-Port aktiviert. Kommt es beim Empfangen oder Interpretieren eines Datenpakets zu einem Fehler, wird in der Statusleiste eine entsprechende Fehlermeldung ausgegeben.

(a) Verbinden. **(b)** Trennen.

Abbildung 5.5: Verbinden und Trennen des virtuellen COM-Ports in der GUI.

Der für den Verbindungsaufbau relevante Teil des Quellcodes ist in Listing 5.6 aufgeführt. Wichtig ist dabei zunächst die Funktion *ports_popupmenu_CreateFcn()*, die beim Start der GUI ausgeführt wird. Hier werden mithilfe des in Listing A.6 aufgeführten Skriptes sämtliche im Betriebssystem verfügbare COM-Ports ermittelt und in das Popup-Menü eingetragen. Die Bezeichnung des standardmäßig ausgewählten COM-Ports wird anschließend in der handle-Variable *portSelected* gespeichert. Die zweite wichtige Funktion *ports_popupmenu_Callback()* wird immer dann aufgerufen, wenn im Popup-Menü ein Port ausgewählt wird. Hierbei wird die Bezeichnung des ausgewählten COM-Ports in der handle-Variable *portSelected* gespeichert. Der eigentliche Verbindungsaufbau geschieht beim Klick auf den Button *Verbinden*. Dabei werden

die Befehle im ersten Teil der if-Verzweigung in der Funktion *Connect_togglebutton_Callback()* ausgeführt. Das heißt, zunächst werden alle alten COM-Ports gelöscht, der im Popup-Menü ausgewählte COM-Port mit der entsprechenden Baud-Rate definiert und der Event-Handler zugewiesen (siehe Abschnitt 5.2.4). Anschließend werden die Beschriftung des Buttons aktualisiert und die ausgegrauten Bedienfelder aktiviert sowie die Log-Datei angelegt, sofern die Logging-Funktion aktiviert ist. Ein erneuter Klick auf den Toggle-Button trennt die Verbindung zum COM-Port. Das Port-Objekt wird zerstört, die Bedienfelder ausgegraut, der Plot zurückgesetzt und die Standardwerte für alle Variablen wiederhergestellt.

Listing 5.6: Implementierung des Verbindungsaufbaus zum COM-Port in der GUI.

```
% ---- Executes on selection change in ports_popupmenu.
function ports_popupmenu_Callback(hObject, eventdata, handles)
% retireve selected COM-Port from popupmenu
handles.portSelected = handles.portsAvailable{get(hObject,'Value')};

% Update handles structure
guidata(hObject, handles);


% ---- Executes during object creation, after setting all properties.
function ports_popupmenu_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% search for available com ports
handles.portsAvailable = getAvailableComPort();
set(hObject,'String',handles.portsAvailable);
handles.portSelected = handles.portsAvailable{get(hObject,'Value')};

% Update handles structure
guidata(hObject, handles);


% ---- Executes on button press in Connect_togglebutton.
function Connect_togglebutton_Callback(hObject, eventdata, handles)
button_state = get(hObject,'Value');
% Beim Klick auf 'Verbinden'
if button_state == get(hObject,'Max')
    % delete any existing port objects
    delete(instrfindall);
    % create serial port object
    handles.s = serial(handles.portSelected,'BaudRate',115200);
    fclose(handles.s);
    % specify callbackfunction, that is executed everytime the terminator '\n' is read on
    serial port
    handles.s.BytesAvailableFcnMode = 'terminator';
    handles.s.BytesAvailableFcn = {@serial_read_data, hObject, handles};
    fopen(handles.s);
    % change button string
    set(hObject,'String','Trennen');
    % enable buttons and radio boxes and edits
    set(handles.Heizen_radiobutton,'Enable','on');
```

```
42      set(handles.Kuehlen_radiobutton,'Enable','on');
43      set(handles.Spannen_pushbutton,'Enable','on');
44      set(handles.Loesen_pushbutton,'Enable','on');
45      set(handles.Abbrechen_pushbutton,'Enable','on');
46      set(handles.KP_HEIZEN_edit,'Enable','on');
47      set(handles.KI_HEIZEN_edit,'Enable','on');
48      set(handles.KP_KUEHLEN_edit,'Enable','on');
49      set(handles.KI_KUEHLEN_edit,'Enable','on');
50      set(handles.T_WACHS_SCHMELZ_edit,'Enable','on');
51      set(handles.T_WACHS_ERSTARR_edit,'Enable','on');
52      set(handles.T_WACHS_ABKUEHL_edit,'Enable','on');
53      set(handles.T_WASSER_FRIER_edit,'Enable','on');
54      set(handles.T_WASSER_ERSTARR_edit,'Enable','on');
55      set(handles.DAUER_SCHMELZEN_edit,'Enable','on');
56      set(handles.DAUER_ABKUEHLEN_edit,'Enable','on');
57      % create new logfile on connect if, checkbox checked
58      createLogFile(hObject, handles);
59  % Beim Klick auf 'Trennen'
60  elseif button_state == get(hObject,'Min')
61      % Clean up serial port if connected
62      if handles.s ~= 0
63          fclose(handles.s);
64          delete(handles.s);
65          clear handles.s;
66          delete(instrfindall);
67      end
68      handles.s = 0;
69      % change button string
70      set(hObject,'String','Verbinden');
71      % disable buttons and radio boxes and edits
72      set(handles.Heizen_radiobutton,'Enable','off');
73      set(handles.Kuehlen_radiobutton,'Enable','off');
74      set(handles.Spannen_pushbutton,'Enable','off');
75      set(handles.Loesen_pushbutton,'Enable','off');
76      set(handles.Abbrechen_pushbutton,'Enable','off');
77      set(handles.KP_HEIZEN_edit,'Enable','off');
78      set(handles.KI_HEIZEN_edit,'Enable','off');
79      set(handles.KP_KUEHLEN_edit,'Enable','off');
80      set(handles.KI_KUEHLEN_edit,'Enable','off');
81      set(handles.T_WACHS_SCHMELZ_edit,'Enable','off');
82      set(handles.T_WACHS_ERSTARR_edit,'Enable','off');
83      set(handles.T_WACHS_ABKUEHL_edit,'Enable','off');
84      set(handles.T_WASSER_FRIER_edit,'Enable','off');
85      set(handles.T_WASSER_ERSTARR_edit,'Enable','off');
86      set(handles.DAUER_SCHMELZEN_edit,'Enable','off');
87      set(handles.DAUER_ABKUEHLEN_edit,'Enable','off');
88      % reset plot
89      clearpoints(handles.h1);
90      clearpoints(handles.h2);
91      clearpoints(handles.h3);
92      clearpoints(handles.h4);
93      xlim(handles.Ausgabeplot,[0 30]);
94      % Reset to default parameters and update widgest
95      handles = declareDefaults(handles);
```

```
96      outputDataToWidgets(handles); % output data to widgets
97      % Reset status
98      set(handles.Status_text,'String','Status: System getrennt, keine Daten emfpangen');
99  end
100
101 % Update handles structure
102 guidata(hObject, handles);
```

5.3.3 Empfangen und Darstellen der Messdaten

Wie bereits in Abb. 5.4 zu sehen, erfolgt die Ausgabe der Daten, die vom Mikrocontroller an die GUI gesendet werden, auf verschiedene Arten. Zum einen werden die Rohdaten im Panel *Raw* (5) dargestellt, zum anderen wird das aktuell eingehende Datenpaket in der Fußzeile (6) der GUI angezeigt. Am wichtigsten ist jedoch die Darstellung der Messdaten im dynamischen Plot (7). Dieser wird immer dann aktualisiert, wenn ein neues Datenpaket eingelesen wurde, also alle 100 ms. Dargestellt werden die Temperatur T_1 der Spannplatte, die Temperatur T_2 des Wärmetauschers, der Sollwert $T_{1,soll}$ der Spannplattentemperatur sowie die Stellgröße des Reglers in Prozent. Zunächst werden dabei nur Daten in einem Zeitraum von 0 bis 30 Sekunden dargestellt. Nach Ablauf dieser Zeit wird die Zeitachse um einen Offset von 30 Sekunden verschoben und die Werte im nächsten 30-sekündigen Intervall dargestellt.

Listing 5.7 enthält die Implementierung der für das Anzeigen der Messdaten relevanten Funktionen. Hierbei erfolgt zunächst, wie bereits in Abschnitt 5.2.4 ausführlich erläutert, das Einlesen der vom Mikrocontroller gesendeten Messwerte innerhalb des Event-Handlers *serial_read_data()*. Die Messwerte stehen anschließend in den jeweiligen handles-Variablen zur Verfügung und werden über die Funktion *outputDataToWidgets()* als Strings in die Textboxen des *Raw*-Panels geschrieben. Weiterhin werden die Radio-Buttons zur Auswahl des Betriebsmodus entsprechend des vom Mikrocontroller zurückgesendeten Wertes eingestellt. Anschließend werden im Event-Handler die Aktions-Buttons (Spannen, Lösen und Abbrechen) je nach aktuellem Systemzustand ausgegraut, sodass der Nutzer einfach erkennen kann, welche Aktionen im aktuellen Betriebszustand zur Verfügung stehen. Die Edit-Boxen für die Einstellung von Systemparametern werden ebenfalls je nach Systemzustand ausgegraut, sodass eine Veränderung der Parameter nur im Zustand *bereit* erfolgen kann. Hiernach werden die Messdaten in die zuvor in Abschnitt 5.3.1 definierten Kurven innerhalb des Plots eingetragen und somit die Kurven mit jedem eingehenden Datenpaket um jeweils einen Datenpunkt vergrößert. Auch das Hinzufügen des Offsets zur Zeitachse des Plots wird im Event-Handler realisiert.

Listing 5.7: Implementierung der Messdatenanzeige in der GUI.

```
1 % ---- Event Handler for Reading data from serial port
2 function serial_read_data(s, event, hObject, handles)
3 % read data from serial port
4 serial_incoming_data = fscanf(s,'%s');
5 set(handles.serialOutput,'String',serial_incoming_data); % Raw Data Label
6 % if an error occurs within try block, following commands in try are skipped
7 % and error label will be set in catch block. Prevents system from crashing
8 % due to faulty serial reads.
```

```
try
    % parse incoming data
    converted_data = eval(['[',serial_incoming_data,']']);
    % update variables based on data
    handles.Betriebsmodus = converted_data(1);
    handles.Zustand = converted_data(2);
    converted_data(3) = converted_data(3)/1000; % change time format into seconds
    handles.Zeit = converted_data(3);
    handles.T1_SOLL = converted_data(4);
    handles.T1 = converted_data(5);
    handles.T2 = converted_data(6);
    handles.TEC_CURRENT = converted_data(7);
    handles.KP_HEIZEN = converted_data(8);
    handles.KI_HEIZEN = converted_data(9);
    handles.KP_KUEHLEN = converted_data(10);
    handles.KI_KUEHLEN = converted_data(11);
    handles.T_WACHS_SCHMELZ = converted_data(12);
    handles.T_WACHS_ERSTARR = converted_data(13);
    handles.T_WACHS_ABKUEHL = converted_data(14);
    handles.T_WASSER_FRIER = converted_data(15);
    handles.T_WASSER_ERSTARR = converted_data(16);
    handles.DAUER_SCHMELZEN = converted_data(17);
    handles.DAUER_ABKUEHLEN = converted_data(18);

    % write data to logfile, if checkbox is checked
    if get(handles.logfile_checkbox,'Value') == 1
        dlmwrite(handles.logFileName,converted_data(1:7),'-append','newline','pc');
    end

    % output data to widgest (text boxes and edit boxes)
    outputDataToWidgets(handles);

    % deactive buttons corresponding to 'Zustand'
    switch handles.Zustand
        case 0 % BEREIT
            set(handles.Spannen_pushbutton,'Enable','on');
            set(handles.Loesen_pushbutton,'Enable','off');
            set(handles.Abbrechen_pushbutton,'Enable','off');
        case 1 % SPANNEN
            set(handles.Spannen_pushbutton,'Enable','off');
            set(handles.Loesen_pushbutton,'Enable','off');
            set(handles.Abbrechen_pushbutton,'Enable','on');
        case 2 % LOESEN
            set(handles.Spannen_pushbutton,'Enable','off');
            set(handles.Loesen_pushbutton,'Enable','off');
            set(handles.Abbrechen_pushbutton,'Enable','on');
        case 3 % GESPANNT
            set(handles.Spannen_pushbutton,'Enable','off');
            set(handles.Loesen_pushbutton,'Enable','on');
            set(handles.Abbrechen_pushbutton,'Enable','off');
    end

    % deactive edit-boxes corresponding to 'Zustand'
    if handles.Zustand == 0; % BEREIT
```

```
63          set(handles.KP_HEIZEN_edit,'Enable','on');
64          set(handles.KI_HEIZEN_edit,'Enable','on');
65          set(handles.KP_KUEHLEN_edit,'Enable','on');
66          set(handles.KI_KUEHLEN_edit,'Enable','on');
67          set(handles.T_WACHS_SCHMELZ_edit,'Enable','on');
68          set(handles.T_WACHS_ERSTARR_edit,'Enable','on');
69          set(handles.T_WACHS_ABKUEHL_edit,'Enable','on');
70          set(handles.T_WASSER_FRIER_edit,'Enable','on');
71          set(handles.T_WASSER_ERSTARR_edit,'Enable','on');
72          set(handles.DAUER_SCHMELZEN_edit,'Enable','on');
73          set(handles.DAUER_ABKUEHLEN_edit,'Enable','on');
74      else % SPANNEN, LOESEN, GESPANNT
75          set(handles.KP_HEIZEN_edit,'Enable','off');
76          set(handles.KI_HEIZEN_edit,'Enable','off');
77          set(handles.KP_KUEHLEN_edit,'Enable','off');
78          set(handles.KI_KUEHLEN_edit,'Enable','off');
79          set(handles.T_WACHS_SCHMELZ_edit,'Enable','off');
80          set(handles.T_WACHS_ERSTARR_edit,'Enable','off');
81          set(handles.T_WACHS_ABKUEHL_edit,'Enable','off');
82          set(handles.T_WASSER_FRIER_edit,'Enable','off');
83          set(handles.T_WASSER_ERSTARR_edit,'Enable','off');
84          set(handles.DAUER_SCHMELZEN_edit,'Enable','off');
85          set(handles.DAUER_ABKUEHLEN_edit,'Enable','off');
86      end
87
88      % plot data points
89      addpoints(handles.h1,handles.Zeit,handles.T1);
90      addpoints(handles.h2,handles.Zeit,handles.T2);
91      addpoints(handles.h3,handles.Zeit,handles.T1_SOLL);
92      addpoints(handles.h4,handles.Zeit,handles.TEC_CURRENT);
93      % wrap x-axis every 30 seconds
94      XL = xlim(handles.Ausgabeplot);
95      if handles.Zeit > XL(2)
96          xlim(handles.Ausgabeplot,[XL(1)+30 XL(2)+30])
97      end
98      drawnow limitrate
99
100     % if no execption occurs change status to Okay
101     set(handles.Status_text,'String','Status: Daten wurden erfolgreich empfangen');
102 catch
103     % if execption occurs change status to Fehler
104     set(handles.Status_text,'String','Status: Fehler beim Empfangen der Daten');
105     % because errors mainly occur shortly after establishing the connection,
106     % it makes sense, to clear the plot after a faulty reading. Otherwise
107     % wrong values will be dsiplayed.
108     clearpoints(handles.h1);
109     clearpoints(handles.h2);
110     clearpoints(handles.h3);
111     clearpoints(handles.h4);
112 end
113
114 % Update handles structure
115 guidata(hObject, handles);
116
```

```
117
118 % ---- Updates static text boxes and edit boxes with current handle values
119 function outputDataToWidgets(handles)
120 switch handles.Betriebsmodus
121     case 0
122         set(handles.Modus_text,'String','Wachs');
123         set(handles.Heizen_radiobutton,'Value',1);
124         set(handles.Kuehlen_radiobutton,'Value',0);
125     case 1
126         set(handles.Modus_text,'String','Wasser');
127         set(handles.Heizen_radiobutton,'Value',0);
128         set(handles.Kuehlen_radiobutton,'Value',1);
129 end
130 switch handles.Zustand
131     case 0
132         set(handles.Zustand_text,'String','Bereit');
133     case 1
134         set(handles.Zustand_text,'String','Spannen');
135     case 2
136         set(handles.Zustand_text,'String','Lösen');
137     case 3
138         set(handles.Zustand_text,'String','Gespannt');
139 end
140 set(handles.Zeit_text,'String',sprintf('%0.1f',handles.Zeit));
141 set(handles.T1_text,'String',sprintf('%0.1f',handles.T1));
142 set(handles.T2_text,'String',sprintf('%0.1f',handles.T2));
143 set(handles.TEC_CURRENT_text,'String',sprintf('%0.1f',handles.TEC_CURRENT));
144 set(handles.T1_SOLL_text,'String',sprintf('%0.1f',handles.T1_SOLL));
145 set(handles.KP_HEIZEN_edit,'String',sprintf('%0.2f',handles.KP_HEIZEN));
146 set(handles.KI_HEIZEN_edit,'String',sprintf('%0.2f',handles.KI_HEIZEN));
147 set(handles.KP_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KP_KUEHLEN));
148 set(handles.KI_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KI_KUEHLEN));
149 set(handles.T_WACHS_SCHMELZ_edit,'String',sprintf('%0.1f',handles.T_WACHS_SCHMELZ));
150 set(handles.T_WACHS_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WACHS_ERSTARR));
151 set(handles.T_WACHS_ABKUEHL_edit,'String',sprintf('%0.1f',handles.T_WACHS_ABKUEHL));
152 set(handles.T_WASSER_FRIER_edit,'String',sprintf('%0.1f',handles.T_WASSER_FRIER));
153 set(handles.T_WASSER_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WASSER_ERSTARR));
154 set(handles.DAUER_SCHMELZEN_edit,'String',handles.DAUER_SCHMELZEN);
155 set(handles.DAUER_ABKUEHLEN_edit,'String',handles.DAUER_ABKUEHLEN);
```

5.3.4 Loggen der Messdaten

Eine wichtige Funktion der GUI ist das Loggen von Messdaten. Innerhalb einer Log-Datei im Unterverzeichnis *logging* werden die Variablen *Betriebszustand*, *Zustand*, *Zeit*, *T1_SOLL*, *T1*, *T2* und *TEC_CURRENT* im CSV-Format, das heißt als durch Kommata getrennte Zeichenketten, abgespeichert. Dabei wird jedes eingehende Datenpaket geloggt, sodass in der Log-Datei Daten in Zeitabständen von 100 ms zur Verfügung stehen. Jede Log-Datei enthält darüber hinaus in der ersten Zeile einen Header, in dem beschrieben ist, um welche Daten es sich in der Log-Datei handelt. Die Log-Dateien werden von der GUI automatisch angelegt, sobald die Checkbox *Schreibe Messdaten in Logfile* aktiviert und das System erfolgreich verbunden wurde. Wird das

System getrennt und erneut verbunden, wird eine neue Log-Datei mit dem aktuellen Datum und der aktuellen Uhrzeit im Namen angelegt. Ein Deaktivieren und erneutes Aktivieren der Logging-Checkbox führt ebenfalls dazu, dass eine neue Log-Datei angelegt wird.

Die Implementierung des Loggings erstreckt sich über mehrere Funktionen. Zunächst wird in der Initialisierungsfunktion in Listing 5.5 der Standard-Dateiname der Log-Datei in der handles-Variable *logFileName* definiert und mittels des Befehls *mkdir* das Untervezeichnis *logging* erstellt. Beim Klick auf den *Verbinden*-Button wird die in Listing 5.8 aufgeführte Funktion *createLogFile()* aufgerufen. Diese erstellt für den Fall, dass die Logging-Checkbox aktiviert ist, eine Datei mit dem aktuellen Datum und aktueller Uhrzeit im Namen und schreibt in diese mithilfe der Funktion *fprintf()* den File-Header. Hierzu muss mithilfe der Funktion *fopen()* ein Filestream in die Log-Datei geöffnet und nach dem Schreiben mittels *fclose()* geschlossen werden.

Listing 5.8: Implementierung des Messdaten-Loggings in der GUI.

```
% ---- Executes on button press in logfile_checkbox.
function logfile_checkbox_Callback(hObject, eventdata, handles)
% if already connected and then logfile button checked
if get(handles.Connect_togglebutton,'Value') == get(handles.Connect_togglebutton,'Max')
    createLogFile(hObject, handles);
end

% Update handles structure
guidata(hObject, handles);


% ---- Creates logfile and writes file header
function createLogFile(hObject, handles)
if get(handles.logfile_checkbox,'Value') == 1
    % change filename according to current date and time
    handles.logFileName = ['logging\','Messung_', datestr(now,'dd-mmm-yyyy_HH-MM-SS'),'.txt'];
    % update handles in serial read data callback
    handles.s.BytesAvailableFcn = {@serial_read_data, hObject, handles};
    % write fileheader
    fid = fopen(handles.logFileName, 'wt');
    fprintf(fid,'Betriebsmodus, Zustand, Zeit, T1_SOLL, T1, T2, TEC_CURRENT');
    fprintf(fid,'\n');
    fclose(fid);
end
```

Das Schreiben der eigentlichen Messdaten in die Log-Datei geschieht im Event-Handler aus Listing 5.7. In diesem werden die Messdaten, wie in Listing 5.9 dargestellt, mittels der Funktion *dlmwrite()* in die zuvor erstellte Log-Datei geschrieben, sofern die Logging-Checkbox aktiviert ist. Dabei muss das zusätzliche Attribut *append* gesetzt werden, damit der bereits in die Datei eingetragene Header nicht überschrieben wird.

Listing 5.9: Schreiben der Messdaten in die Log-Datei innerhalb des Event-Handlers.

```
% write data to logfile, if checkbox is checked
if get(handles.logfile_checkbox,'Value') == 1
```

```
3     dlmwrite(handles.logFileName,converted_data(1:7),'-append','newline','pc');
4 end
```

5.3.5 Steuerung des Spannsystems

Wie später noch genauer erläutert wird, können im Spannsystem verschiedene Aktionen getriggert werden, die das System in einen bestimmten Zustand versetzen. Dieser Zustand kann sich durch zeitliche Bedingungen oder Bedingungen an die Temperaturen ändern. Entsprechend ist es in der GUI möglich, je nach aktuellem Zustand des Systems bestimmte Aktionen durchzuführen. Befindet sich das System beispielsweise im Zustand *bereit*, ist es möglich einen Spannvorgang durch Klick auf den Button *Spannen* auszulösen. Dies führt dazu, dass das System zuunächst in den Zustand *spannen* übergeht und bei Erreichen einer bestimmten Temperatur den Zustand *gespannt* annimmt. In diesem Zustand ist anschließend nur noch die Aktion *Lösen* verfügbar, was durch Ausgrauen der unzulässigen Buttons verdeutlicht wird. Abb. 5.6 zeigt die für die Steuerung der System-Aktionen relevanten Bedienelemente der GUI.

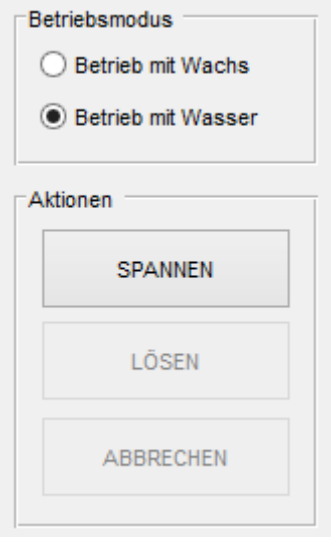


Abbildung 5.6: Buttons und Radio Boxes zur Steuerung des Spannsystems innerhalb der GUI.

Die Implementierung innerhalb der GUI erfolgt in den Callback-Funktionen der Buttons und Radio-Buttons. Listing 5.10 zeigt beispielhaft die Callback-Funktion, die beim Klick auf den *Spannen*-Button ausgeführt wird. Hierbei wird zunächst die Variable *Aktion* auf den Wert 1 gesetzt, was der Aktion *Spannen* entspricht. Durch Aufruf der Funktion *serial_send_commands()* wird ein ausgehendes Datenpaket mit dem aktualisierten Wert der Variable *Aktion* an den Mikrocontroller geschickt. Dieser reagiert daraufhin durch entsprechende Änderung seines Zustands und setzt die Aktions-Variable auf den Wert 0 zurück, sodass nach Empfang des Datenpakets eine weitere Aktion ausgeführt werden kann. Entsprechend ist es nicht möglich, Aktionen in kürzeren Zeitintervallen als die durch die Abtastzeit vorgegebenen 100 ms auszuführen. Dies stellt jedoch keine Limitierung des praktischen Nutzens dar. Bei Betätigung der Radio-Buttons wird entsprechend die Variable *Betriebsmodus* manipuliert.

Listing 5.10: Setzen der Aktions-Variable und Senden an den Mikrocontroller bei Klick auf den Button *Spannen*.

```
% ---- Executes on button press in Spannen_pushbutton.
function Spannen_pushbutton_Callback(hObject, eventdata, handles)
handles.Aktion = 1; % Spannen
% send command to serial
serial_send_commands(handles);

% Update handles structure
guidata(hObject, handles);
```

5.3.6 Setup des Spannsystems

Ist die GUI erfolgreich mit dem Port verbunden und empfängt Daten vom Mikrocontroller, können die Systemparameter des Spannsystems innerhalb des in Abb. 5.7 dargestellten *Setup*-Panels angepasst werden. Voraussetzung hierfür ist weiterhin, dass sich das Spannsystem im Zustand *bereit* befindet. Das heißt während der Durchführung einer Aktion ist keine Änderung der Parameter möglich und die Edit-Boxen sind ausgegraut. Die Bedeutungen der einzelnen Parameter können Tabelle 5.1 entnommen werden. Standardmäßig sind sie auf die in Tabelle 5.2 aufgelisteten Werte, die beim Start der GUI eingestellt werden, festgelegt. Durch Klicken in eine der Edit-Boxen und Eingabe eines gültigen Wertes wird die Größe beim Verlassen der Edit-Box (Klicken auf anderes GUI-Element) aktualisiert und der aktualisierte Wert an den Mikrocontroller geschickt. Genau wie bei den Aktions-Buttons aus Abschnitt 5.3.5 werden die Inhalte der Edit-Boxen auf Basis der eingehenden Datenpakete gesetzt. Daher ist stets eine maximale Integrität der Daten innnerhalb der GUI und auf dem Mikrocontroller sichergestellt. Bei der Eingabe von Werten ist der zulässige Wertebereich für die jeweilige Größe aus Tabelle 5.1 zu beachten. Liegt ein Wert außerhalb des angegebenen Wertebereiches oder wird ein nichtnumerischer Wert, beispielsweise ein Buchstabe, eingegeben, wird keine Aktualisierung des Wertes vorgenommen.

Listing 5.11 zeigt die Implementierung der Parameteranpassung am Beispiel der Edit-Box für den Parameter *KP_HEIZEN*. Innerhalb der Callback-Funktion der Edit-Box wird der eingegebene Zeichenstring ermittelt und mithilfe des Befehls *str2num()* in einen numerischen Wert umgewandelt sowie gespeichert. Anschließend wird geprüft, ob der eingegebene Wert ein gültiger numerischer Wert ist und im erlaubten Wertebereich liegt. Ist dies der Fall, wird die handles-Variable des Parameters aktualisiert und ein Datenpaket mit dem aktualisierten Wert an den Mikrocontroller geschickt. Innerhalb der *serial_read_data()*-Funktion (siehe Listing 5.7) wird schließlich der vom Mikrocontroller zurückgesendete Wert in die Edit-Box eingetragen.

Listing 5.11: Implementierung der Parameteranpassung innerhalb einer Edit-Box.

```
% ---- Update parameters based upon text in edit box
function KP_HEIZEN_edit_Callback(hObject, eventdata, handles)
val = str2num(get(hObject,'String'));
if ~isempty(val) % check if value is numeric
    if val >= 0 % check if value is valid
```

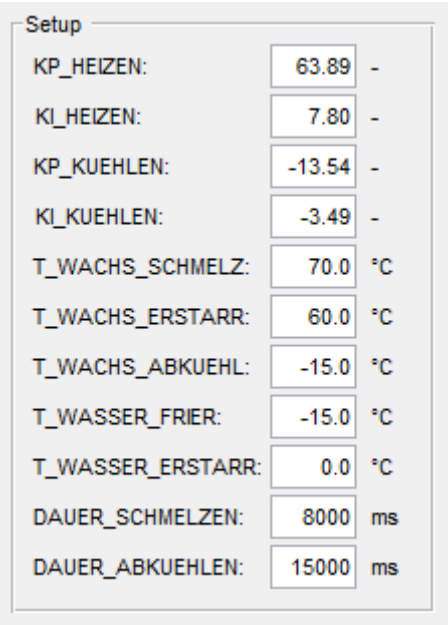


Abbildung 5.7: In der GUI einstellbare Parameter der Steuerung.

```
6         handles.KP_HEIZEN = val;
7         % send command to serial
8         serial_send_commands(handles);
9     end
10 end
11 % Update handles structure
12 guidata(hObject, handles);
```

5.3.7 Verwalten von Setup-Profilen

Da die in Abb. 5.7 gezeigten Systemparameter bei jedem neuen Verbindungsaufbau auf die Standardwerte aus Tabelle 5.2 zurückgesetzt, unter Umständen jedoch andere Werte benötigt werden, bietet die GUI die Möglichkeit, verschiedene Profile anlegen zu können. Hierzu muss, wie in Abb. 5.8 zu sehen, ein eindeutiger Name in die Edit-Box (1) eingegeben werden. Mit einem Klick auf den *Hinzufügen*-Button (2) werden sämtliche Systemparameter in einer Profildatei gespeichert und das Profil in der Liste (5) verfügbarer Profile angezeigt. Zum Laden eines Profils, muss dieses innerhalb der Liste ausgewählt und anschließend der *Laden*-Button geklickt werden. Profile können ebenfalls durch Auswählen und Klicken auf den *Löschen*-Button (3) wieder entfernt werden. Das entsprechende Profil verschwindet in diesem Fall aus der Liste und die zugehörige Profildatei wird gelöscht. Profile können dabei in jedem Zustand des Systems, selbst wenn die GUI nicht mit dem Mikrocontroller verbunden ist, angelegt werden. Das Laden eines Profils ist jedoch nur dann möglich, wenn eine Verbindung zum Spannsystem besteht und sich dieses im Zustand *bereit* befindet. Die Profildateien selbst werden mit der Endung **.prf* im Verzeichnis *profiles* hinterlegt und enthalten die Systemparameter als eine durch Kommata getrennte Zeichenkette. Beim Start der GUI wird das *profiles*-Verzeichnis nach bestehenden

Tabelle 5.2: Standardwerte der Systemparameter.

Parameter	Einheit	Wert
KP_HEIZEN	-	63,89
KI_HEIZEN	-	7,00
KP_KUEHLEN	-	−13,54
KI_KUEHLEN	-	−3,49
T_WACHS_SCHMELZ	°C	70,0
T_WACHS_ERSTARR	°C	60,0
T_WACHS_ABKUEHL	°C	−15,0
T_WASSER_FRIER	°C	−15,0
T_WASSER_ERSTARR	°C	0,0
DAUER_SCHMELZEN	ms	8000
DAUER_ABKUEHLEN	ms	15 000

Profilen durchsucht und diese gegebenenfalls in der Profilliste (5) aufgeführt. Existiert ein Profil bereits in der Liste, ist es nicht möglich, ein weiteres Profil mit demselben Namen anzulegen.

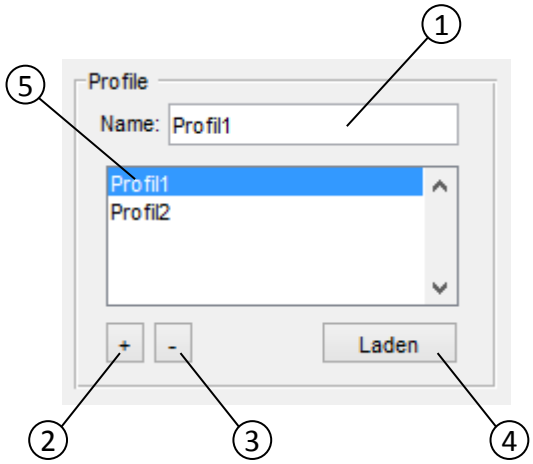


Abbildung 5.8: Speichern, Löschen und Laden von Parameter-Profilen in der GUI.

Die Implementierung der Profilverwaltung ist Listing 5.12 zu entnehmen. Zunächst wird dort bei Erstellung der Profil-Listbox die Funktion *getAvailableProfileNames()* aufgerufen. Diese Funktion durchsucht das *profiles*-Verzeichnis und gibt die Namen aller dort gespeicherten Dateien ohne Dateiendung in Form eines Cell-Arrays zurück. Anschließend werden diese Namen in die Profil-Listbox eingetragen. Hierdurch werden beim Start der GUI bereits vorhandene Profile in die Liste geladen. Beim Hinzufügen eines neuen Profils wird die Funktion *add_pushbutton_Callback()* ausgeführt. Hier wird zunächst der in die Edit-Box eingegebene Name des zu speichernden Profils eingelesen. Anschließend wird wieder die Funktion *getAvailableProfileNames()* aufgerufen, um die Namen bereits vorhandener Profile zu erhalten. Anschließend wird der eingegebene Name

mit jedem vorhandenen Namen verglichen. Liegt bereits ein Profil mit gleichem Namen vor, wird das *exist_flag* auf den Wert 1 gesetzt und alle nachfolgenden Befehle übersprungen, das heißt, das neue Profil wird nicht angelegt. Existiert kein Profil mit dem eingegebenen Namen, hat das *exist_flag* den Wert 0 und es wird eine Datei im Unterverzeichnis *profiles* mit dem Namen des Profils und der Endung **.prf* angelegt. In diese wird mittels *fprintf()* eine Zeichenkette mit allen durch Kommata getrennten Systemparametern geschrieben. Danach muss die Liste aktualisiert und das neue Profil in die Liste eingefügt werden. Hierzu werden zunächst die alten Listeneinträge in die Variable *prev_Content* gespeichert. War die Liste zuvor leer, wird nun einfach der Name des neuen Profils eingefügt. Lagen hingegen bereits Listeneinträge vor, wird der neue Listeninhalt aus dem alten Listeninhalt und dem Namen des neuen Profils zusammengesetzt und der neue Inhalt in die Liste geschrieben. Beim Klick auf den *Löschen*-Button wird die Funktion *delete_pushbutton_Callback()* ausgeführt. Hierin wird zunächst abgefragt, welches der Profile innerhalb der Liste gerade ausgewählt ist. Dessen Name wird anschließend abgefragt und in der Variable *current_Items* abgespeichert. Existieren Profil-Dateien im *profiles*-Verzeichnis und insbesondere die Datei des ausgewählten zu löschenden Profils, so wird diese Datei mit dem Befehl *delete()* gelöscht. Weiterhin wird das entsprechende Profil aus der Profil-Liste gelöscht. Zu guter Letzt führt ein Klick auf den *Laden*-Button zur Ausführung der Funktion *Laden_pushbutton_Callback()*. Innerhalb dieser wird zunächst der Name des aktuell in der Profil-Liste ausgewählten Profils abgefragt. Dieser Profilname wird an die Funktion *csvread()* übergeben, die die durch Kommata getrennten Systemparameter aus der Profil-Datei ausliest und ein Array mit den entsprechenden Werten zurückgibt. Aus diesem Array werden anschließend die einzelnen Werte extrahiert und den zugehörigen handles-Variablen zugewiesen sowie in die Edit-Boxen als formatierte Zeichenketten eingefügt. Abschließend werden die aktualisierten Systemparameter mittels der Funktion *serial_send_commands()* an den Mikrocontroller geschickt.

Listing 5.12: Implementierung der Profilverwaltung in der GUI.

```
% ---- Executes during object creation, after setting all properties.
function Profile_listbox_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% load existing profiles into listbox
profiles_available = getAvailableProfileNames();
% put items into listbox
set(hObject,'String',profiles_available);


% ---- Executes on button press in add_pushbutton.
function add_pushbutton_Callback(hObject, eventdata, handles)
profil_name = get(handles.Profil_name_edit,'String');
% check, if entered profile name already exist in the directory
profiles_available = getAvailableProfileNames();
exist_flag = 0;
for i = 1:length(profiles_available)
    if strcmp(profiles_available{i},profil_name)
        exist_flag = 1;
    end
end
```

```
% only run this code, if profile name doesn't exist
if ~exist_flag
    fid = fopen(['profiles\',profil_name,'.prf'], 'wt');
    fprintf(fid,[num2str(handles.KP_HEIZEN),',',num2str(handles.KI_HEIZEN)...
        ,',',num2str(handles.KP_KUEHLEN),',',num2str(handles.KI_KUEHLEN)...
        ,',',num2str(handles.T_WACHS_SCHMELZ),',',num2str(handles.T_WACHS_ERSTARR)...
        ,',',num2str(handles.T_WACHS_ABKUEHL),',',num2str(handles.T_WASSER_FRIER)...
        ,',',num2str(handles.T_WASSER_ERSTARR),',',num2str(handles.DAUER_SCHMELZEN)...
        ,',',num2str(handles.DAUER_ABKUEHLEN)]);
    fclose(fid);

    % add item to listbox with name entered in textedit
    set(handles.Profile_listbox,'Value',1);
    prev_Content = get(handles.Profile_listbox,'String');
    % don't copy old content on empty ist
    if isempty(prev_Content)
        new_content = {profil_name};
    else
        new_content = [cellstr(prev_Content);{profil_name}];
    end
    set(handles.Profile_listbox,'String',new_content);
end
% Update handles structure
guidata(hObject, handles);


% --- Returns names (without extension) of profiles available in profile directory
function profiles_available_names_without_ext = getAvailableProfileNames()
% list all *.prf-files
profiles_available = dir('profiles\*.prf');
% retrieve names of all *.prf-files
profiles_available_names = {profiles_available(:).name};
% get the name of available profiles without file extension
profiles_available_names_without_ext = [];
for i = 1:length(profiles_available_names)
    [f_pathstr,f_name,f_ext] = fileparts(profiles_available(i).name);
    profiles_available_names_without_ext{end+1,1} = f_name;
end


% --- Executes on button press in delete_pushbutton.
function delete_pushbutton_Callback(hObject, eventdata, handles)
% delete selected profile
current_Selection = get(handles.Profile_listbox,'Value');
current_Items = get(handles.Profile_listbox,'String');
% check if profiles available
if ~isempty(current_Items)
    % delete profile-file if file exists
    if exist(['profiles\',current_Items{current_Selection},'.prf'],'file') > 0
        delete(['profiles\',current_Items{current_Selection},'.prf']);
    end
    % delete selected item from list
    current_Items(current_Selection) = [];
    set(handles.Profile_listbox,'String',current_Items,'Value',1);
```

```
77 end
78
79
80 % ---- Executes on button press in Laden_pushtbutton.
81 function Laden_pushtbutton_Callback(hObject, eventdata, handles)
82 if handles.Zustand == 0 && handles.s ~= 0 % loading only possible if Zustand = 'BEREIT' and
   connected
83     % get selected item
84     current_Selection = get(handles.Profile_listbox,'Value');
85     current_Items = get(handles.Profile_listbox,'String');
86     % read values from file and set parameters accordingly
87     imported_data = csvread(['profiles\',current_Items{current_Selection},'.prf']);
88     handles.KP_HEIZEN = imported_data(1);
89     handles.KI_HEIZEN = imported_data(2);
90     handles.KP_KUEHLEN = imported_data(3);
91     handles.KI_KUEHLEN = imported_data(4);
92     handles.T_WACHS_SCHMELZ = imported_data(5);
93     handles.T_WACHS_ERSTARR = imported_data(6);
94     handles.T_WACHS_ABKUEHL = imported_data(7);
95     handles.T_WASSER_FRIER = imported_data(8);
96     handles.T_WASSER_ERSTARR = imported_data(9);
97     handles.DAUER_SCHMELZEN = imported_data(10);
98     handles.DAUER_ABKUEHLEN = imported_data(11);
99
100    % set contents of edit boxes
101    set(handles.KP_HEIZEN_edit,'String',sprintf('%0.2f',handles.KP_HEIZEN));
102    set(handles.KI_HEIZEN_edit,'String',sprintf('%0.2f',handles.KI_HEIZEN));
103    set(handles.KP_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KP_KUEHLEN));
104    set(handles.KI_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KI_KUEHLEN));
105    set(handles.T_WACHS_SCHMELZ_edit,'String',sprintf('%0.1f',handles.T_WACHS_SCHMELZ));
106    set(handles.T_WACHS_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WACHS_ERSTARR));
107    set(handles.T_WACHS_ABKUEHL_edit,'String',sprintf('%0.1f',handles.T_WACHS_ABKUEHL));
108    set(handles.T_WASSER_FRIER_edit,'String',sprintf('%0.1f',handles.T_WASSER_FRIER));
109    set(handles.T_WASSER_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WASSER_ERSTARR));
110    set(handles.DAUER_SCHMELZEN_edit,'String',handles.DAUER_SCHMELZEN);
111    set(handles.DAUER_ABKUEHLEN_edit,'String',handles.DAUER_ABKUEHLEN);
112
113    % send command to serial
114    serial_send_commands(handles);
115 end
116
117 % Update handles structure
118 guidata(hObject, handles);
```

5.3.8 Schließen der GUI

Wird der *Schließen*-Button des GUI-Fensters angeklickt, so versucht das Betriebssystem, die Anwendung zu terminieren. Zuvor wird jedoch die in Listing 5.13 gezeigte Funktion *gui_window_CloseRequestFcn()* ausgeführt. Hierin werden der serielle Port *s* mittels *fclose()* geschlossen und das Port-Objekt mit dem Befehl *delete()* gelöscht. Dadurch wird sichergestellt,

dass der Port freigegeben wird und sich die GUI bei einem erneuten Start problemlos mit dem Port verbinden kann.

Listing 5.13: Funktion zur Bereinigung vor dem Schließen der GUI.

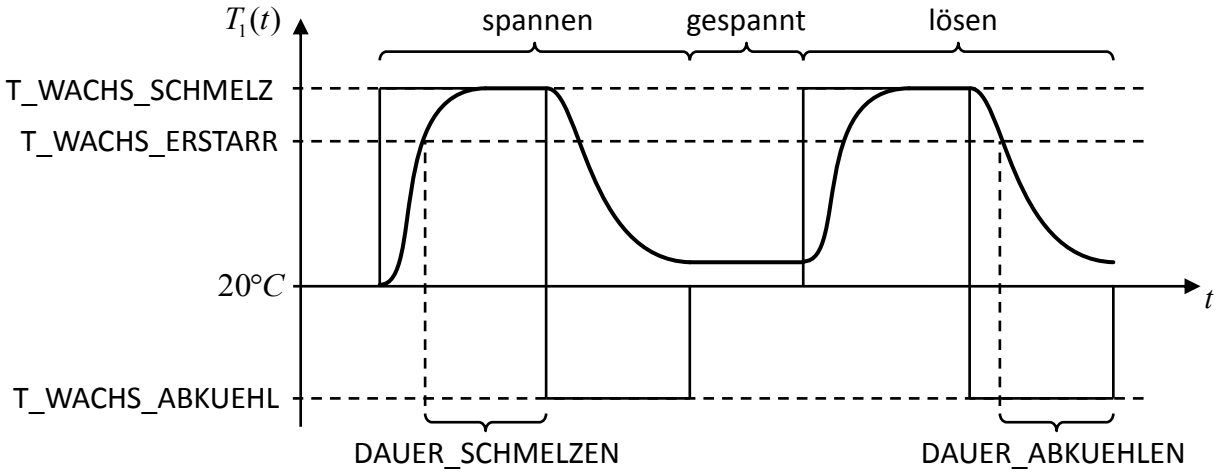
```
% ——— Executes when user attempts to close gui_window.
function gui_window_CloseRequestFcn(hObject, eventdata, handles)
% Clean up serial port if connected
if handles.s ~= 0
    fclose(handles.s);
    delete(handles.s);
    clear handles.s;
    delete(instrfindall);
end

% close the figure
delete(hObject);
```

5.4 Mikrocontroller-Firmware

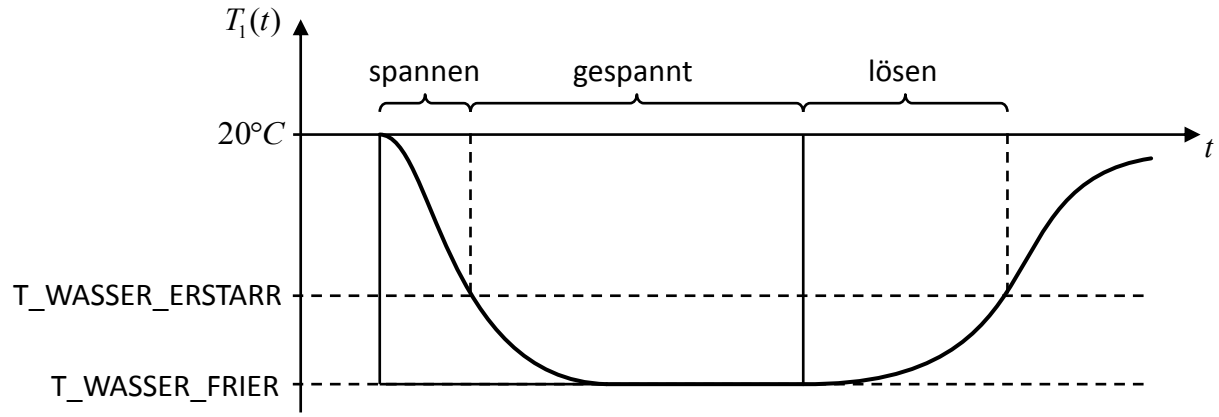
Ein wichtiger Bestandteil des Software-Systems ist die Firmware des Mikrocontrollers, der in der Steuerelektronik der Spanneinheit verbaut ist. Diese umfasst neben dem in Abschnitt 4.3 implementierten Regler und einem Teil der in Abschnitt 5.2.3 entwickelten Schnittstelle die Programmlogik der Spanneinheit. Während die GUI lediglich die Funktion einer Schnittstelle zwischen Anwender und Steuerungssoftware darstellt, übernimmt die Firmware die konkrete Ansteuerung des thermischen Systems. Je nach Betriebsmodus, aktuellem Zustand des Spannsystems sowie von der GUI eingehenden Befehlen werden die Sollwerte der Spannplattentemperatur angepasst und damit der gewünschte Vorgang im Spannsystem ausgelöst.

Eine detaillierte Darstellung des in der Firmware implementierten Systemverhaltens zeigen die Sollwertverläufe der Spannplattentemperatur in Abb. 5.9a für den Betrieb mit Wachs und in Abb. 5.9b für den Betrieb mit Wasser. Sie sind an die bereits in Abschnitt 3.5 erfolgte Funktionsbeschreibung des Spannsystems angelehnt.

Im Betrieb mit Wachs erfolgt ein Einspannvorgang grundsätzlich so, dass in der GUI zunächst der Button *Spannen* gedrückt wird. Daraufhin wird der Sollwert der Spannplattentemperatur auf den in *T_WACHS_SCHMELZ* gespeicherten Wert gesetzt und damit der Heizvorgang gestartet. Das System ändert seinen Zustand von *bereit* in *spannen*. Sobald die Erstarrungstemperatur des Wachses *T_WACHS_ERSTARR* überschritten wird, wird die Zeit *DAUER_SCHMELZEN* abgewartet, bevor der Sollwert der Temperatur auf den Wert *T_WACHS_ABKUEHL* gesetzt und das System in den Kühlmodus versetzt wird. Auf diese Weise wird die beim Schmelzen des Wachses entstandene Wärme abgeführt und der Erstarrungsvorgang beschleunigt. Ist die Spannplattentemperatur unter die Erstarrungstemperatur abgefallen, wird die Zeit *DAUER_ABKUEHLEN* abgewartet, bevor die Kühlung abgeschaltet und das System vom Zustand *spannen* in den Zustand *gespannt* übergeht. Das Lösen des Werkstücks erfolgt gleichermaßen. Das Abbrechen



(a) Sollwertverlauf der Spannplattentemperatur im Betrieb mit Wachs.



(b) Sollwertverlauf der Spannplattentemperatur im Betrieb mit Wasser.

Abbildung 5.9: Zu implementierende Sollwertverläufe der Spannplattentemperatur im Wachs- und Gefrierspannbetrieb.

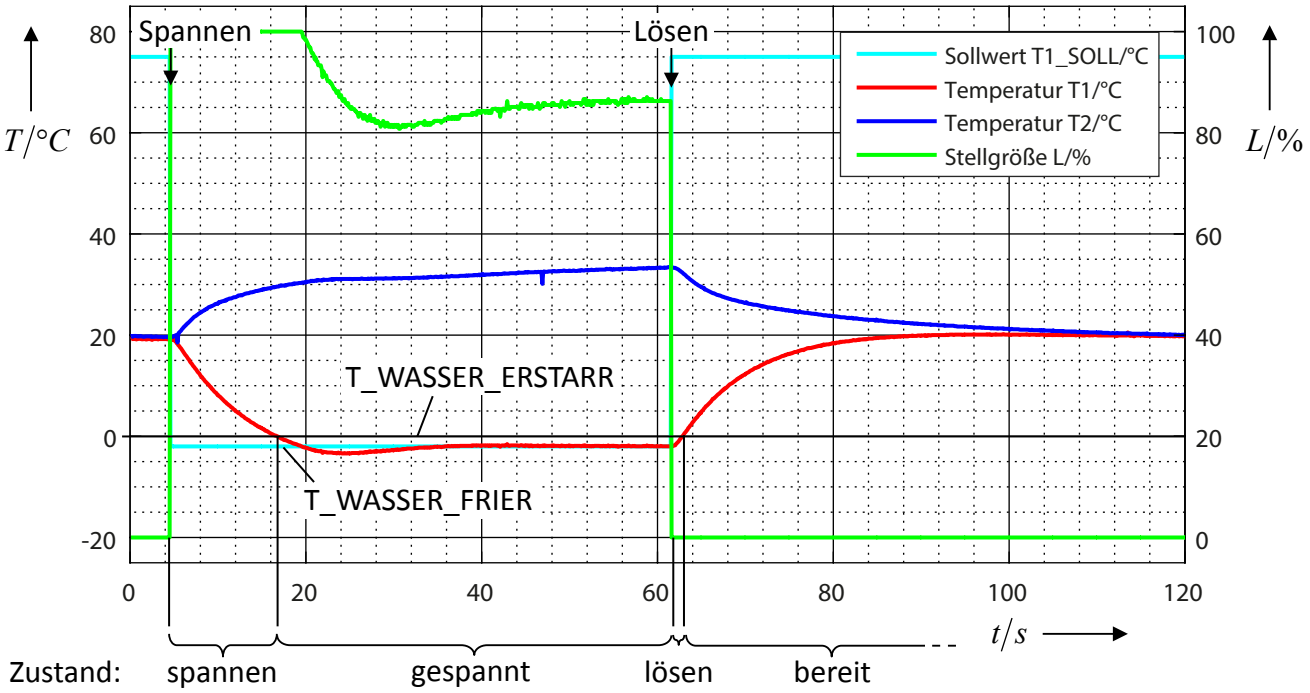
einer Aktion ist nur in den Zuständen *spannen* und *lösen* möglich. In diesen Fällen wird die Heizung sofort abgeschaltet und das System geht in den Zustand *bereit* über.

Befindet sich das Spannsystem im Gefrierspannbetrieb, verhält es sich wie in Abb. 5.9b dargestellt. Wird in der GUI der Button *Spannen* gedrückt, wird der Sollwert der Spannplattentemperatur auf den in *T_WASSER_FRIER* gespeicherten Wert gesetzt und damit die Spannplatte abgekühlt. Der Zustand des Systems ändert sich von *bereit* in *spannen*. Unterschreitet die Spannplattentemperatur den Wert *T_WASSER_ERSTARR*, ändert sich der Zustand in *gespannt*, die Kühlung läuft jedoch weiter, da andernfalls das Eis schmelzen würde. Wird in diesem Zustand in der GUI der Button *Lösen* gedrückt, wird der Zustand in *lösen* geändert und die Kühlung abgeschaltet. Überschreitet die Temperatur anschließend den Wert *T_WASSER_ERSTARR*, geht das System vom Zustand *lösen* in den Zustand *bereit* über und ein erneutes Spannen ist möglich. Ein Klick auf den Button *Abbrechen* in der GUI ist nur möglich, wenn sich das System im Zustand *spannen* befindet. In diesem Fall wird die Kühlung sofort abgeschaltet und der

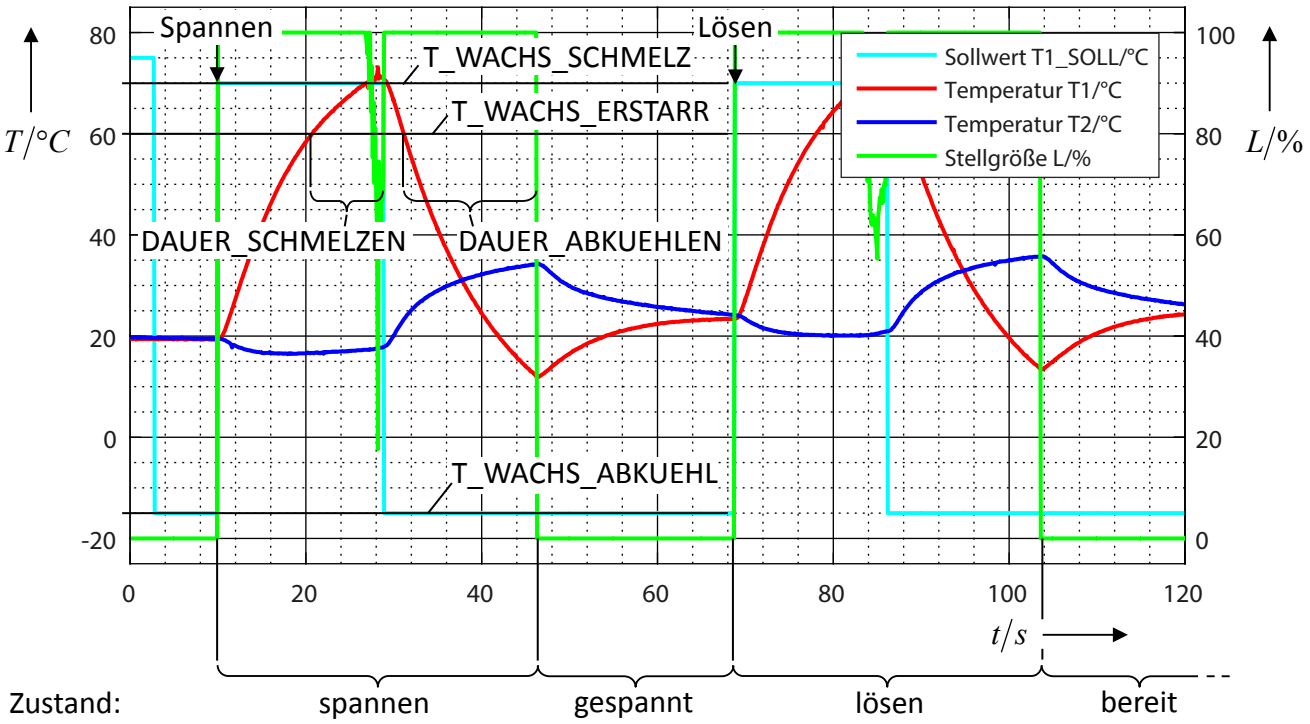
Zustand in *bereit* geändert.

Die Implementierung dieses Verhaltens auf dem Mikrocontroller ist in Listing A.7 dargestellt und soll an dieser Stelle nicht im Detail ausgeführt werden. Erwähnenswert ist lediglich, dass innerhalb der Firmware die Variablen *Betriebsmodus*, *Zustand* und *Aktion* gespeichert werden, die entsprechend des aktuellen Systemzustandes und entsprechend der von der GUI gesendeten Befehle bestimmte Werte annehmen. Anhand der aktuellen Werte dieser Variablen wird bei jedem Durchlauf der Hauptprogrammschleife ein bestimmter Anweisungsblock innerhalb einer Switch-Verzweigung ausgeführt, wobei es für jede Kombination von Werten genau einen Block, das heißt insgesamt 27 verschiedene Blöcke, gibt. In diesen Anweisungsblöcken werden unter anderem die Solltemperatur und der Systemzustand geändert. Wird beispielsweise die Aktion *Spannen* im Betriebsmodus mit Wasser ausgelöst und befindet sich das System im Zustand *bereit*, wird im nächsten Schleifendurchlauf im entsprechenden Anweisungsblock zunächst die Variable *Aktion* zurückgesetzt, der Zustand in *spannen* geändert und der Sollwert der Temperatur auf *T_WASSER_FRIER* gesetzt.

Zur Überprüfung der korrekten Funktion der Mikrocontroller-Firmware wird ein Werkstück sowohl im Gefrier- als auch im Wachsspannbetrieb eingespannt und anschließend gelöst. Die Verläufe der Spannplattentemperatur $T_1(t)$, der Wärmetauschertemperatur $T_2(t)$, des Temperatursollwerts $T_{1,soll}(t)$ und der Stellgröße $L(t)$ des Reglers werden dabei mithilfe der Logging-Funktion innerhalb der GUI aufgezeichnet. Sie sind in Abb. 5.10 dargestellt. Die Systemparameter wurden bei dem Versuch auf den in Tabelle 5.2 dargestellten Standardwerten belassen. Eine Modifikation der Parameter wird das Betriebsverhalten des Spannsystems entsprechend ändern.



(a) Spannen und Lösen eines Werkstücks im Gefrierspannbetrieb.



(b) Spannen und Lösen eines Werkstücks im Wachsspannbetrieb.

Abbildung 5.10: In der GUI aufgezeichnete Verläufe der Systemgrößen beim Spannen und Lösen eines Werkstücks im Gefrier- und Wachsspannbetrieb.

6 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein modulares Software-System zur Ansteuerung eines thermischen Spannsystems für die Mikrobearbeitung, bestehend aus einem Temperaturregler, einer Steuerlogik und einem graphischen Nutzerprogramm, das auf einem externen PC ausgeführt wird, entwickelt. Sämtliche in der Aufgabenstellung genannten Ziele der Arbeit konnten erreicht werden. So wurde nach einer umfassenden Erläuterung der regelungstechnischen Grundlagen zunächst ein Modell der zu regelnden Strecke in Form einer verbalen Beschreibung des Systems sowie Blockschaltbildern und in Form von Übertragungsfunktionen aufgestellt. Letztere wurden experimentell durch Messung des Ein- und Ausgangsverhalten der Regelstrecke im Zeitbereich bestimmt und boten eine sehr gute Übereinstimmung mit den Validierungsdatensätzen. Zusätzlich wurde ein detailliertes theoretisches Modell des thermischen Teilsystems aufgestellt, simuliert und validiert. Ebenfalls wurde eine statische Kennlinie des Systems aufgezeichnet, um die Nichtlinearität des Systems zu beurteilen. Nach einer detaillierten Präsentation der Modellierungsergebnisse folgte der Entwurf eines Temperaturreglers basierend auf den erstellten Modellen. Zuvor jedoch wurden in einer umfassenden Literaturrecherche verschiedene Verfahren zum Entwurf von Reglern zusammengetragen und in knapper Form beschrieben. Zusätzlich wurden Grundlagen zu Reglern, insbesondere zum PID-Regler und zu dessen Diskretisierung erläutert. Auf Basis dieser Kenntnisse wurde anschließend ein PI-Regler als geeignete Reglerstruktur ausgewählt, die Güteanforderungen an den geschlossenen Regelkreis definiert und dementsprechend die Reglerparameter nach dem Verfahren von Chien, Hrones und Reswick sowie durch automatisches Tuning mithilfe des MATLAB PID-Tuners bestimmt. Anschließend wurde der Regler in Form einer C++-Klasse auf dem Mikrocontroller implementiert. Das Verhalten des geschlossenen Regelkreises konnte abschließend auf Basis der zuvor erstellten Modelle der Regelstrecke und mit dem entworfenen Regler in Simulink simuliert und die Einhaltung der Güteanforderungen an die Regelung überprüft werden. Im anschließenden Kapitel wurde die übergeordnete Steuerungssoftware entwickelt. Hierzu wurden zunächst die wichtigsten Anforderungen an das Software-System festgehalten und eine Schnittstelle zum Datenaustausch zwischen Mikrocontroller-Firmware und graphischer Nutzeroberfläche entwickelt. Nachdem deren Funktionsweise und Implementierung im Detail erläutert wurde, folgte die Entwicklung der graphischen Nutzeroberfläche. Es wurde die Implementierung der einzelnen, in der Spezifikation geforderten Funktionen der Software erläutert. Anschließend konnte mit der Entwicklung der Mikrocontroller-Firmware, die die eigentliche Programmlogik des Spannsystems sowie den zuvor erstellten digitalen Regler enthält, fortgefahren werden. Hierzu wurde zunächst genau spezifiziert, wie das Spannsystem in den unterschiedlichen Betriebszuständen reagieren sollte. Abschließend wurden die Funktion des Spannsystems praktisch erprobt und die Ergebnisse präsentiert.

Auch wenn alle Ziele erreicht werden konnten, stellten sich im Rahmen der Entwicklung des vorliegenden Systems einige kleinere Probleme heraus, die in Zukunft besser gelöst werden könnten. So verfolgte der Reglerentwurf hauptsächlich das Ziel einer einfach zu implementierenden

und schnell zu entwickelnden Temperaturregelung. Daher wurde zunächst mit dem PI-Regler eine gängige und bewährte Reglerstruktur ausgewählt und bei deren Einstellung lediglich auf empirische Einstellregeln beziehungsweise eine einfache manuelle Justierung der Reglerparamter zurückgegriffen. Besser wäre es gewesen, hier mit einem der vorgestellten Entwurfsverfahren, beispielsweise einem Integralkriterium, optimale Parameter für Regler zu finden. Unter Umständen könnte auch ein gänzlich anderes Konzept zur Regelung verwendet werden, wie zum Beispiel eine Fuzzy-Regelung oder ein lernfähiges neuronales Netz, das künstliche Intelligenz abzubilden versucht. Bei Verwendung eines klassischen PID-Reglers könnte zudem ein automatisches Tuning der Parameter, wie in [40] beschrieben, erfolgen, sodass der Regler unabhängig vom Streckenzustand immer optimal eingestellt ist. Neben dem nicht optimal eingestellten Regler hätte die Umsetzung der Schnittstelle anders gelöst werden können. Aufgrund der einfachen Realisierbarkeit wurde hier ein Verfahren gewählt, bei dem beide Kommunikationspartner unabhängig voneinander Daten in Form von ASCII-Zeichenstrings senden und empfangen können. Durch entsprechende Maßnahmen bei der Implementierung konnte zwar eine ausreichende Zuverlässigkeit dieses Übertragungssystems erzielt werden, in Hinblick auf eine höhere Flexibilität, bessere Erweiterbarkeit, höhere Geschwindigkeit und leichtere Verständlichkeit wäre aber die Entwicklung eines bidirektionalen Schnittstellen-Protokolls, das entweder binär oder auch über ASCII-Zeichen arbeitet, sinnvoller gewesen. Es hätten Lese- und Schreib-Buffer implementiert und verschiedene Befehle für einzelne Aktionen und die Datenübertragung realisiert werden können. Ein weiteres Problem zeigt sich in der GUI. Diese wurde aufgrund der verhältnismäßig einfachen Skriptsprache und der umfangreichen zur Verfügung stehenden Werkzeuge in MATLAB implementiert. Dies stellt kein Problem dar, solange die Software direkt in MATLAB ausgeführt wird. Die Ausführung als Standalone-Anwendung jedoch ist aufgrund der geringen Performance der Skriptsprache und der Notwendigkeit eines Interpreters verhältnismäßig langsam. Abhilfe würde nur die Implementierung in einer anderen Programmiersprache, beispielsweise C/C++ schaffen, die jedoch mit einem nicht in Kauf zu nehmenden Mehraufwand bei der Realisierung verbunden wäre. Trotz dieser Probleme muss jedoch gesagt werden, dass die Software sämtliche in der Spezifikation festgelegten Anforderungen erfüllt und sich in der Praxis als zuverlässige, intuitive und komfortable Möglichkeit zur Steuerung und Überwachung des Spannsystems aus der Ferne herausgestellt hat.

Sollten sich weitere Arbeiten an die hier vorliegende anschließen, so ist insbesondere die Neuentwicklung der Steuerungselektronik anzustreben. Wie eingangs erwähnt, wurde hier eine vereinfachte Schaltung verwendet. Um die Regelung des Spannsystems noch schneller zu machen, ist es erforderlich, einen konstanten Strom im Peltier-Element einstellen zu können. Dies war mit der vorhandenen Treiberschaltung nicht möglich, sodass die Leistungsfähigkeit des Stellgliedes eingeschränkt wurde. Nach der Entwicklung einer neuen Treiberschaltung sind die Schritte der Modellbildung und des Reglerentwurfes erneut durchzuführen. Hierbei stellt die vorliegende Arbeit einen geeigneten Leitfaden für diese Arbeitsschritte dar und stellt umfangreiches Grundlagenwissen für eine weitere Optimierung der Regelung zur Verfügung. Eine denkbare Aufgabe für zukünftige Arbeiten wäre auch die Weiterentwicklung der GUI. Diese könnte um zusätzliche Funktionen ergänzt und hinsichtlich ihrer Performance verbessert werden. Sinnvoll wäre auch die Einbindung von Netzwerk-Funktionen in die GUI oder die Möglichkeit, mehrere Spannsysteme gleichzeitig ansteuern zu können. So könnten die GUI in Zukunft auf einem zentralen Rechner, an dem die Spanneinheit angeschlossen ist, ausgeführt werden, und weitere

Instanzen der Software könnten auf anderen PCs über das Internet auf diese zugreifen. Dies würde auch die Entwicklung einer Applikation für mobile Endgeräte begünstigen. Alternativ könnte das Spannsystem über drahtlose Schnittstellenhardware, wie beispielsweise Bluetooth oder andere Funksysteme, mit einem Mobilgerät, auf dem eine Steuerungsapplikation ausgeführt wird, kommunizieren. Zusätzlich sollte bei zukünftigen Weiterentwicklungen eine Erweiterung der Diagnosefunktionen in Betracht gezogen werden. Hierbei sollten unzulässige Betriebszustände des Systems erkannt und gegebenenfalls automatisch beseitigt werden.

Abschließend lässt sich festhalten, dass das in der vorliegenden Arbeit behandelte Thema eine bedeutende Stellung in der zukünftigen Entwicklung flexibler Fertigungssysteme einnehmen wird. Die mit der Miniaturisierung von Werkstücken einhergehende Miniaturisierung der Fertigungsmaschinen ermöglicht aufgrund der Verringerung des Energiebedarfs sowie des erforderlichen Bauraums der Anlagen eine wirtschaftlichere Fertigung. Das Einspannen von Werkstücken mittels Adhäsionskräften stellt dabei eine gute Möglichkeit zur Verkleinerung des erforderlichen Bauraums der Spannvorrichtung dar. Durch die Integration von dezentralen Steuerungen in die thermischen Spanneinheiten wird eine dynamische Zuordnung von Fertigungsaufträgen zu einzelnen Werkstücken innerhalb eines flexiblen Fertigungssystems ermöglicht, wodurch die Variantenvielfalt von Produkten bei gleichbleibender Wirtschaftlichkeit der Fertigung vergrößert werden kann. Die entwickelte Software stellt eine erste Stufe der Ansteuerung eines solchen Systems dar. Aufgrund ihrer modularen Struktur bietet sie dabei optimale Voraussetzungen für die Integration der beschriebenen Funktionen in das Spannsystem.

Literaturverzeichnis

[1] Glindemann, T. »Konzeptionierung und Aufbau eines Gefrierspannsystems für die Mikrobearbeitung«. 2016.

[2] IWF - TU Braunschweig. ISW - Uni Stuttgart. *DFG-Schwerpunktprogramm SPP 1476*. 2016.

[3] Ottens, M. »Praktische Verfahren zur experimentellen Systemidentifikation: Skript zur Vorlesung«. Berlin, 2008.

[4] Imboden, D. M. und Koch, S. *Systemanalyse: Einführung in die mathematische Modellierung natürlicher Systeme*. 2. korrigierter Nachdr. der 1. Aufl. Berlin: Springer, 2005. isbn: 3-540-43935-8.

[5] Ottens, M. »Grundlagen der Systemtheorie: Skript zur Vorlesung«. Berlin, 2008.

[6] Krewer, U. »Einführung in die Regelungstechnik: Skript zur Vorlesung«. Braunschweig, 2015.

[7] Unbehauen, H. *Regelungstechnik II: Zustandsregelungen, digitale und nichtlineare Regelsysteme*. 9., durchgesehene und korrigierte Auflage. Studium Technik. Wiesbaden: Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH Wiesbaden, 2007. isbn: 978-3-528-83348-0. url: http://dx.doi.org/10.1007/978-3-8348-9139-6.

[8] Lunze, J. *Regelungstechnik 1*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. isbn: 978-3-642-53908-4. doi: 10.1007/978-3-642-53909-1.

[9] Derebasi, N., Eltez, M., Guldiken, F. u. a. »Performance of Novel Thermoelectric Cooling Module Depending on Geometrical Factors«. In: *Journal of Electronic Materials* 44.6 (2015), S. 1566–1572. doi: 10.1007/s11664-014-3482-x.

[10] Lineykin, S. und Ben-Yaakov, S. »Modeling and Analysis of Thermoelectric Modules«. In: *IEEE Transactions on Industry Applications* 43.2 (2007), S. 505–512. issn: 0093-9994. doi: 10.1109/TIA.2006.889813.

[11] Manikandan, S. und Kaushik, S. C. »Energy and exergy analysis of an annular thermoelectric cooler«. In: *Energy Conversion and Management* 106 (2015), S. 804–814. issn: 01968904. doi: 10.1016/j.enconman.2015.10.029.

[12] Custom Thermoelectric. *Analysis of Thermoelectric Modules*. 2015. url: http://www.customthermoelectric.com/Peltier_analysis.html (besucht am 26. 03. 2016).

[13] Multicomp. *Datenblatt des Peltier-Elements MHCPE-127-10-08*. 2016. url: http://www.farnell.com/datasheets/2012335.pdf.

[14] Thermonamic. *Datenblatt des Peltier-Elements TEC1-12706*. 2015. url: http://www.produktinfo.conrad.com/datenblaetter/175000-199999/189115-da-01-en-PELTIER_ELEMENT_TEC1_12706.pdf.

[15] Pololu Corporation. *Produktinformation Pololu G2 High-Power Motor Driver 18v17*. 2015. URL: https://www.pololu.com/product/2991 (besucht am 20. 04. 2016).

[16] Quick Ohm Küpper & Co. GmbH. *Herstellerinformation zur Stromversorgung von Peltier-Elementen*. 2014. URL: http://www.quick-cool.de/peltierelemente/stromversorgung-peltierelemente.htm (besucht am 20. 04. 2016).

[17] Plessmann, K. »Regelung von Strecken mit Allpass-Anteil«. In: *Regelungstechnik* 15.2 (1967), S. 60–66. (Besucht am 24. 04. 2016).

[18] Ljung, L. *MATLAB System Identification Toolbox Benutzerhandbuch*. 2015. URL: http://de.mathworks.com/help/pdf_doc/ident/ident.pdf (besucht am 24. 04. 2016).

[19] Atmel Corporation. *AVR221: Discrete PID controller: Application Note*. 2006. URL: http://www.atmel.com/images/doc2558.pdf (besucht am 02. 04. 2016).

[20] Beauredard, B. *Improving the Beginner's PID*. 2011. URL: http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/ (besucht am 21. 04. 2016).

[21] Bechtloff, J. *Regelungstechnik*. 1. Aufl. Vogel-Studienmodule. Würzburg: Vogel, 2012. ISBN: 9783834332028.

[22] Beier, T. und Wurl, P. *Regelungstechnik: Basiswissen, Grundlagen, Beispiele*. 2., neu bearb. Aufl. Hanser eLibrary. München: Hanser, 2015. ISBN: 9783446442108. URL: http://dx.doi.org/10.3139/9783446443938.

[23] Bode, H. *MATLAB-SIMULINK: Analyse und Simulation dynamischer Systeme*. 2., vollst. überarb. Aufl. Lehrbuch Elektrotechnik. Wiesbaden: Teubner, 2006. ISBN: 3-8351-0050-5. URL: http://deposit.ddb.de/cgi-bin/dokserv?id=2788528&prov=M&dok_var=1&dok_ext=htm.

[24] Brouër, B. *Regelungstechnik für Maschinenbauer*. 2., überarb. und erw. Aufl. Stuttgart: Teubner, 1998. ISBN: 3519163284.

[25] Datta, A., Ho, M.-T. und Bhattacharyya, S. P. *Structure and synthesis of PID controllers*. Advances in Industrial Control. London: Springer, 2000. ISBN: 1-85233-614-5. URL: http://www.loc.gov/catdir/enhancements/fy0816/99043761-d.html.

[26] Future Technology Devices International Limited. *Datenblatt FT232R USB UART IC*. 2015. URL: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf (besucht am 15. 04. 2016).

[27] Gerdes, A. »Einsatz einer kompakten Gefrierspanneinheit für Mikrozerspanmaschinen«. Diss. Essen und Braunschweig: Techn. Univ., 2014.

[28] Lavoie, D. *Matlab Skript zur Auflistung verfügbarer COM-Ports*. 2005. URL: http://www.mathworks.com/matlabcentral/fileexchange/9251-get-available-com-port (besucht am 14. 04. 2016).

[29] *Übersicht über Lineare Regelkreisglieder*. 2015. URL: http://200.126.14.82/web/Documentos/Gr%C3%83%C2%A1ficosSistemas.pdf (besucht am 01. 04. 2016).

[30] Lunze, J. *Regelungstechnik 2*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. ISBN: 978-3-642-53943-5. DOI: 10.1007/978-3-642-53944-2.

[31] MathWorks Inc. *Control System Toolbox: Entwurf und Analyse von Steuerungs- und Regelungssystemen*. 2016. URL: http://de.mathworks.com/products/control/ (besucht am 16. 05. 2016).

[32] MathWorks Inc. *MATLAB Dokumentation zum PID-Regler*. 2009. URL: http://de.mathworks.com/help/simulink/slref/pidcontroller.html (besucht am 21. 04. 2016).

[33] Ottens, M. »Einführung in die Regelungstechnik: Skript zur Vorlesung«. Berlin, 2008.

[34] Roth, G. *Regelungstechnik: Wirkungsweisen, Anwendungen und systemorientierte Grundlagen in anschaulicher Darstellung*. 2., völlig neu bearb. und erw. Aufl. Heidelberg: Hüthig, 2001. ISBN: 3-7785-2820-3.

[35] Schneider, W. *Praktische Regelungstechnik: Ein Lehr- und Übungsbuch für Nicht-Elektrotechniker*. 3., vollständig überarbeitete und erweiterte Auflage. Wiesbaden: Vieweg+Teubner / GWV Fachverlage GmbH Wiesbaden, 2008. ISBN: 9783528246624. URL: http://dx.doi.org/10.1007/978-3-8348-9512-7.

[36] Schulz, G. *Lineare und nichtlineare Regelung, rechnergestützter Reglerentwurf*. 4., überarb. Aufl. Regelungstechnik. München: Oldenbourg, 2010. ISBN: 3486591940.

[37] Svaricek, F. »Digitale Regelung: Skript zur Vorlesung«. München, 2012.

[38] Thesycon Software Solutions GmbH. *USB CDC/ACM Class Driver for Windows*. 2016. URL: http://www.thesycon.de/eng/usb_cdcacm.shtml (besucht am 15. 04. 2016).

[39] Unbehauen, H. *Regelungstechnik I: Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. 15., überarbeitete und erweiterte Auflage. Wiesbaden: Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2008. ISBN: 978-3-8348-0497-6. URL: http://dx.doi.org/10.1007/978-3-8348-9491-5.

[40] Visioli, A. *Practical PID Control*. Advances in Industrial Control. London: Springer-Verlag London Limited, 2006. ISBN: 1-84628-585-2. URL: http://dx.doi.org/10.1007/1-84628-586-0.

[41] Zacher, S. und Reuter, M. *Regelungstechnik für Ingenieure: Analyse, Simulation und Entwurf von Regelkreisen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014. ISBN: 978-3-8348-1786-0. DOI: 10.1007/978-3-8348-2216-1.

A Anhang

Der vorliegende Anhang enthält Inhalte, die für das Verständnis der Arbeit von untergeordneter Bedeutung sind, jedoch vom Autor als hilfreich angesehen werden. Es werden zunächst typische Übertragungsglieder der Regelungstechnik in tabellarischer Form dargestellt, anschließend eine erweiterte Treiberschaltung präsentiert und die zur experimentellen Modellbildung verwendeten Testskripte dargestellt. Abschließend werden sämtliche Quellcodes der Steuerungssoftware in zusammenhängender Form aufgelistet.

A.1 Übersicht über lineare Übertragungsglieder

Um ein Verständnis für das Verhalten der Regelstrecke sowie anderer im Regelkreis enthaltener Übertragungsglieder zu entwickeln, können die in der vorliegenden Tabelle enthaltenen Standard-Glieder der Regelungstechnik betrachtet werden. Es werden jeweils die Übergangsfunktion, die Übertragungsfunktion, die Pol- und Nullstellenverteilung, das Bode-Diagramm und die Ortskurve der Übertragungsglieder dargestellt.

Tabelle A.1: Übersicht über Übertragungsglieder 1. [29]

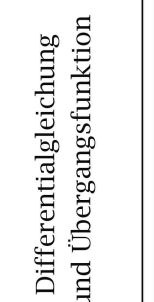
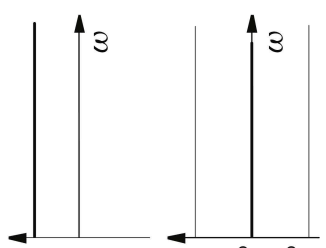
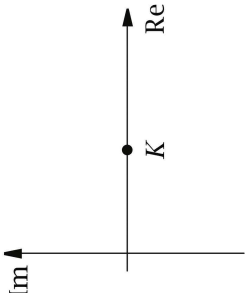
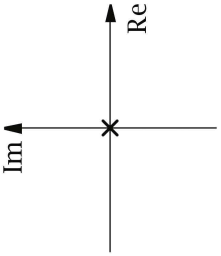
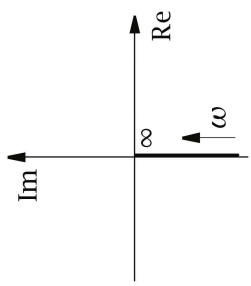
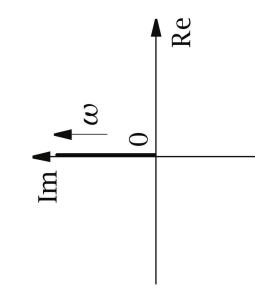
Bez.	Differentialgleichung und Übergangsfunktion	Übertragungsfunktion und Pol- und Nullstellen-Diagramm	Bode-Diagramm Amplituden- und Phasengang	Ortskurve des Frequenzgangs
P	$y=Ku$ 	$G(s)=K$		
I	$y=K_I \int u dt$	$G(s)=\frac{K_I}{s}$ 		
D	$y=K_D \dot{u}$	$G(s)=K_D s$		

Tabelle A.2: Übersicht über Übertragungsglieder 2. [29]

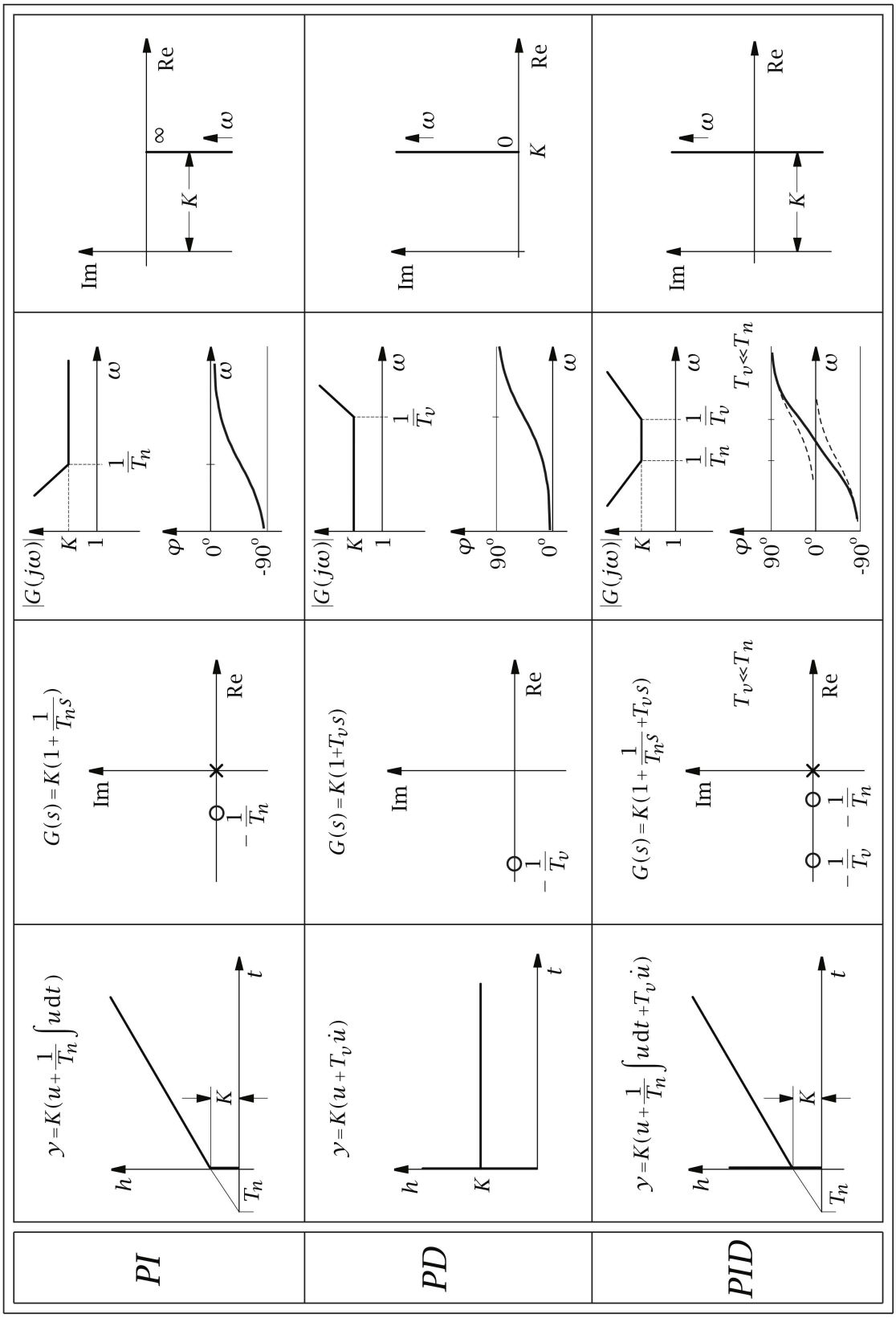


Tabelle A.3: Übersicht über Übertragungsglieder 3. [29]

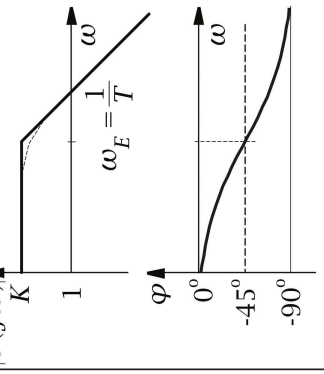
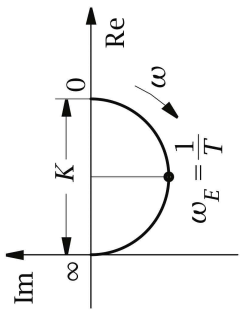
Bez.	Differentialgleichung und Übergangsfunktion	Übertragungsfunktion und Pol- und Nullstellen Diagramm	Bode-Diagramm Amplituden- und Phasengang	Ortskurve des Frequenzgangs
PT_1	$T\dot{y} + y = Ku$ 	$G(s) = \frac{K}{1+Ts}$ 		
PT_2 $(D \geq 1)$	$T_1 T_2 \ddot{y} + (T_1 + T_2)\dot{y} + y = Ku$	$G(s) = \frac{K}{T_1 T_2 s^2 + (T_1+T_2)s+1}$		
PT_2 $(D < 1)$	$\ddot{y} + 2D\omega_0\dot{y} + \omega_0^2 y = \omega_0^2 Ku$	$G(s) = \frac{\omega_0^2 K}{s^2 + 2D\omega_0 s + \omega_0^2}$		

Tabelle A.4: Übersicht über Übertragungsglieder 4. [29]

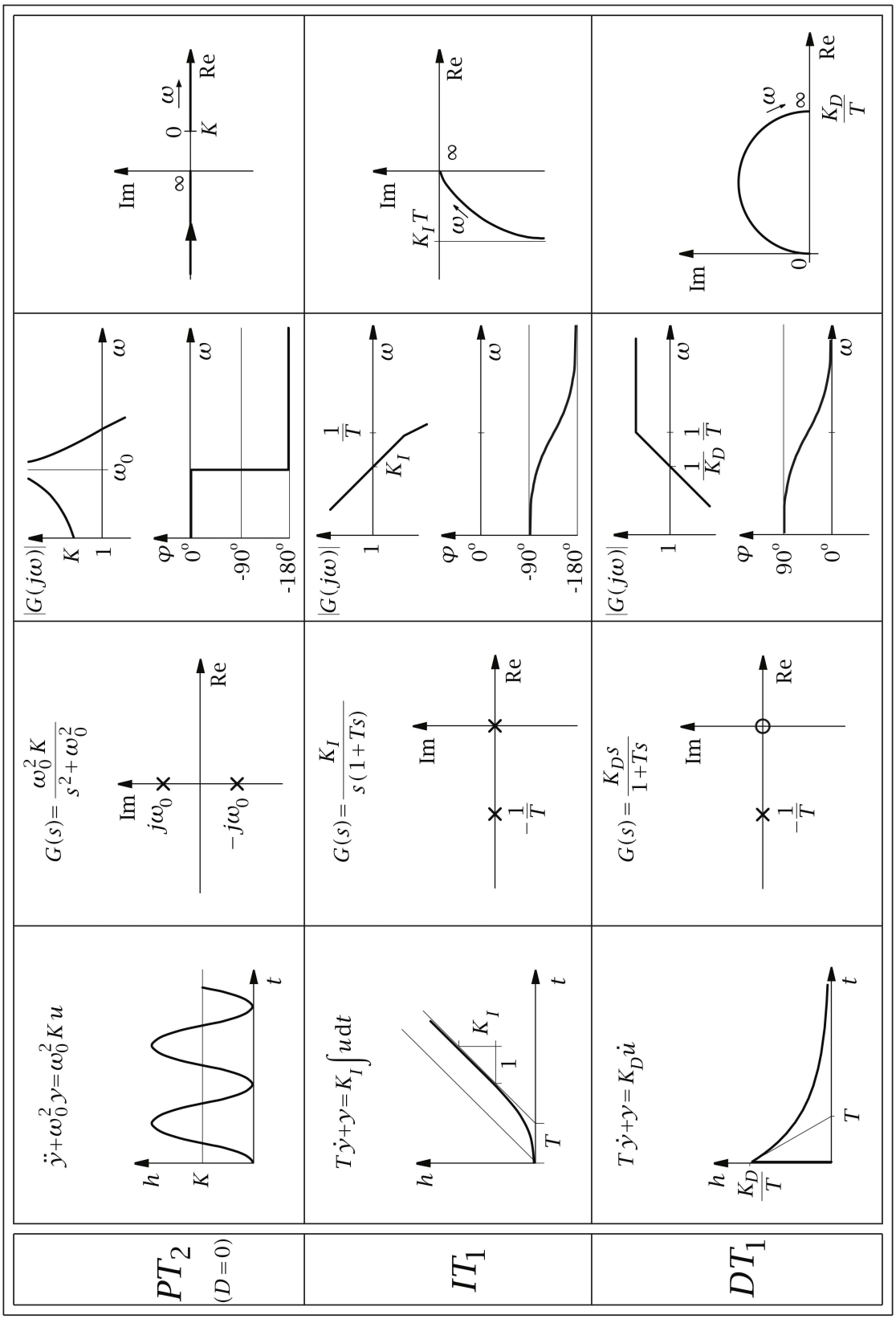
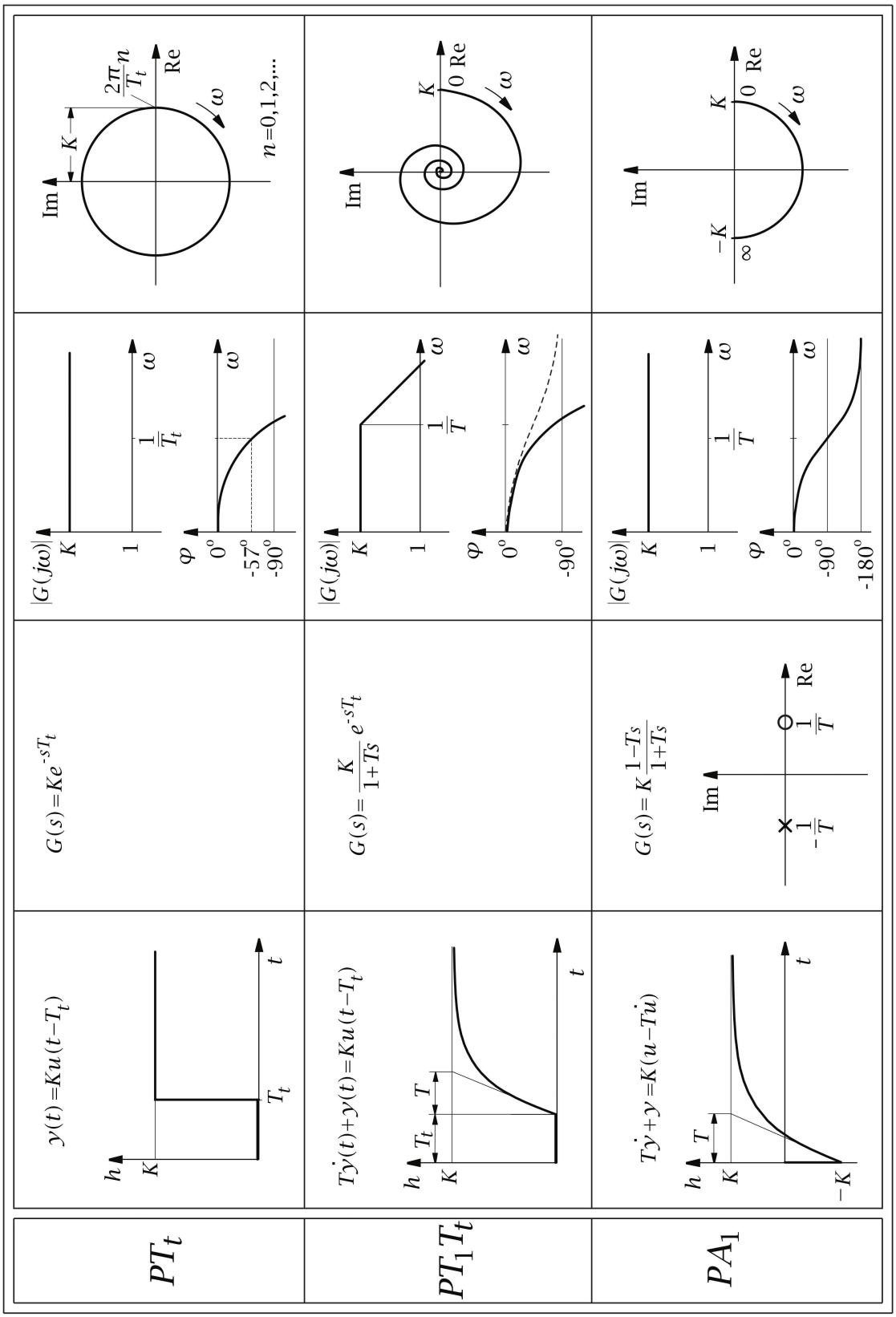


Tabelle A.5: Übersicht über Übertragungsglieder 5. [29]

Bez.	Differentialgleichung und Übergangsfunktion	Übertragungsfunktion und Pol- und Nullstellen Diagramm	Bode-Diagramm Amplituden- und Phasengang	Ortskurve des Frequenzgangs
PIT_1	$T\dot{y}+y=K(u+\frac{1}{T_n}\int u\,dt)$ Steigung $\frac{K}{T_n}$ h, K, T, t	$G(s)=\frac{K(1+\frac{1}{T_n s})}{1+Ts}$ Im, Re $-\frac{1}{T}$, $-\frac{1}{T_n}$	$\lvert G(j\omega)\rvert$: 1, $\frac{1}{T_n}$, $\frac{1}{T}$, ω φ: 0^o, -90^o, ω	Im, Re $K(1-\frac{T}{T_n})$, ∞, ω
PDT_1	$T\dot{y}+y=K(u+T_v\dot{u})$ h, $K\frac{T_v}{T}$, K, T, t	$G(s)=K\frac{1+T_v s}{1+Ts}$ Im, Re $-\frac{1}{T}$, $-\frac{1}{T_v}$	$\lvert G(j\omega)\rvert$: K, 1, $\frac{1}{T_v}$, $\frac{1}{T}$, ω φ: 90^o, 0^o, ω	Im, Re ω, 0, K, ∞, $K\frac{T_v}{T}$
PPT_1	$T\dot{y}+y=K(u+T_v\dot{u})$ h, K, T, $K\frac{T_v}{T}$, t	$G(s)=K\frac{1+T_v s}{1+Ts}$ Im, Re $-\frac{1}{T_v}$, $-\frac{1}{T}$	$\lvert G(j\omega)\rvert$: K, 1, $\frac{1}{T}$, $\frac{1}{T_v}$, ω φ: 0^o, -90^o, ω	Im, Re K, 0, ω, $K\frac{T_v}{T}$, ∞

Tabelle A.6: Übersicht über Übertragungsglieder 6. [29]



A.2 Erweiterte Steuerschaltung

Das Schema zeigt den prinzipiellen Aufbau der ursprünglichen Steuerelektronik des Spannsystems. Im Gegensatz zur tatsächlich umgesetzten Schaltung, die in Abschnitt 3.5.2 detailliert beschrieben wird, verfügt diese Schaltung über eine Stromregelung, die den Strom durch das Peltier-Element auf einem konstanten Wert hält.

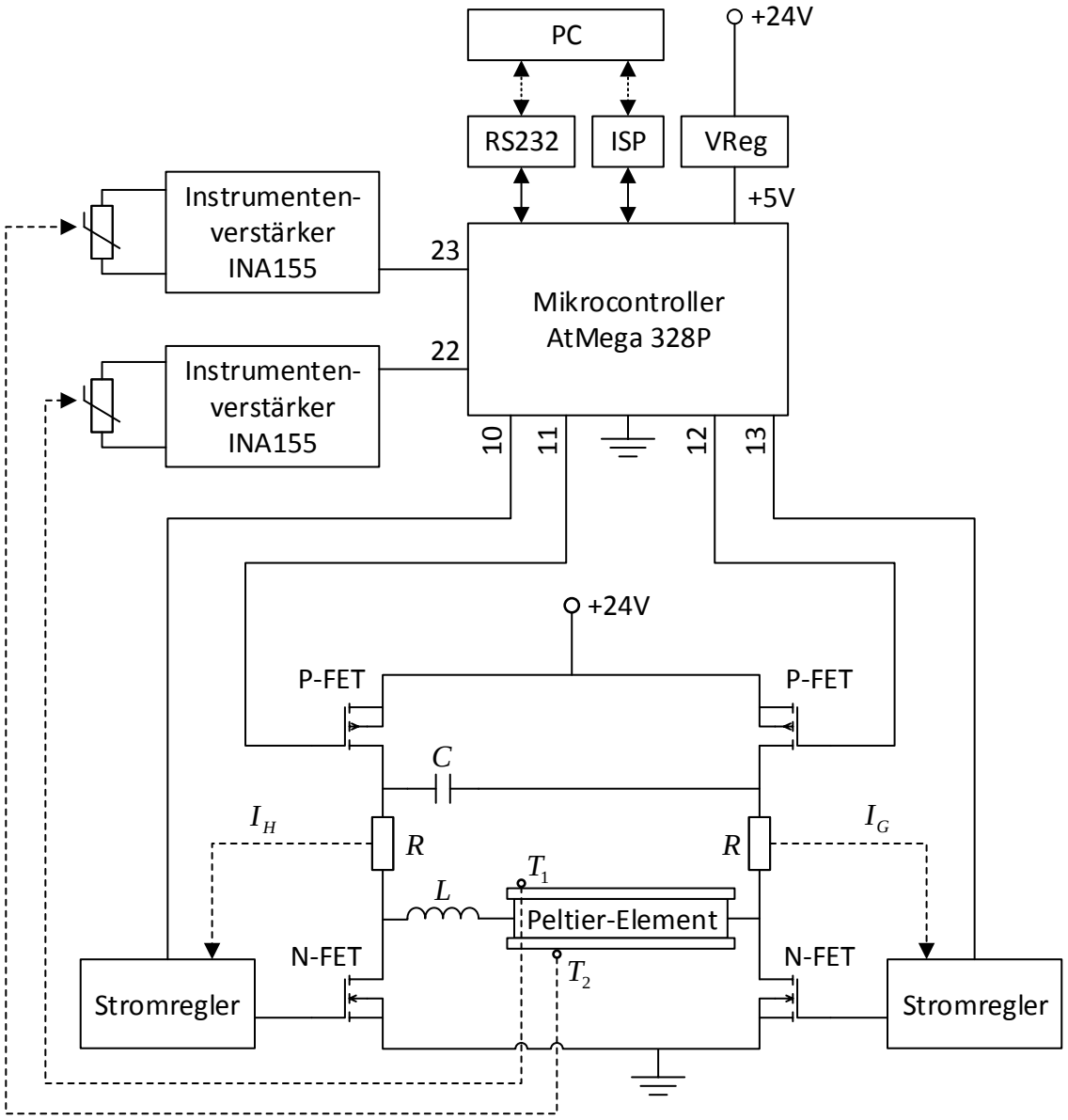


Abbildung A.1: Schematische Darstellung der um eine Stromregelung erweiterten Treiberschaltung inklusive Messeinrichtungen und Mikrocontroller.

A.3 Skripte für die Auswertung des theoretischen Modells der Strecke

Die beiden vorliegenden Skripte werden im Rahmen der theoretischen Modellbildung des Peltier-Elements in Abschnitt 3.5.1.3 zur Simulation des aufgestellten Differentialgleichungssystems verwendet.

Listing A.1: Skript zur Beschreibung des Differentialgleichungssystems des Peltier-Elements in MATLAB.

```
function dT = DGL_System(t,T,I)
%Modellkonstanten
N = 127; %Anzahl der Thermopaare
A_1 = 0.484E-3; %[m^2] Oberfläche der Spannplatte
A_2 = 0.071; %[m^2] Oberfläche des Wärmetauschers
m_1 = 0.019; %[kg] Masse der Spannplatte
m_2 = 0.256; %[kg] Masse des Wärmetauschers
c_1 = 900; %[J/(kg*K)] Wärmekapazität der Spannplatte (Aluminium)
c_2 = 380; %[J/(kg*K)] Wärmekapazität des Wärmetauschers (Kupfer)
alpha_1 = 1; %[W/(m^2*K)] Wärmeübergangskoeffizient zwischen Spannplatte und Luft
alpha_2 = 24; %[W/(m^2*K)] Wärmeübergangskoeffizient zwischen Wärmetauscher und Luft
alpha = 0.210E-3; %[V/K] Seebeck-Koeffizient von Bi2Te3
R = 1.85; %[Ohm] elektrischer Widerstand des gesamten Peltier-Elements
Theta = 0.2084E-2; %[W/K] thermische Leitfähigkeit des Peltier-Materials
T_u = 293; %[K] Lufttemperatur

%DGlL-System
dT = zeros(2,1);
dT(1) = 1/(m_1*c_1)*(alpha_1*A_1*(T_u-T(1))-N*alpha*I*T(1)+0.5*R*I^2+N*Theta*(T(2)-T(1)));
dT(2) = 1/(m_2*c_2)*(alpha_2*A_2*(T_u-T(2))+N*alpha*I*T(2)+0.5*R*I^2-N*Theta*(T(2)-T(1)));
end
```

Listing A.2: Skript zur numerischen Lösung des Differentialgleichungssystems des Peltier-Elements.

```
%%löst das nichtlineare Gleichungssystem für die Temperaturverläufe
clear all;
dt = [0 300]; %[s] Zeitintervall für das Aufklingen
dt2 = [300 600]; %[s] Zeitintervall für das Abklingen
dT_0 = [293 293]; %[K] Anfangswerte für T_1 und T_2

%numerische Lösung Aufklingen
[tsim1,Tsim1] = ode45(@DGL_System,dt,dT_0,[],1);
[tsim2,Tsim2] = ode45(@DGL_System,dt,dT_0,[],2);
[tsim3,Tsim3] = ode45(@DGL_System,dt,dT_0,[],3);
[tsim4,Tsim4] = ode45(@DGL_System,dt,dT_0,[],3.5);
[tsim5,Tsim5] = ode45(@DGL_System,dt,dT_0,[],4);
[tsim6,Tsim6] = ode45(@DGL_System,dt,dT_0,[],4.5);
[tsim7,Tsim7] = ode45(@DGL_System,dt,dT_0,[],5);

%numerische Lösung Abklingen
[tsim1a,Tsim1a] = ode45(@DGL_System,dt2,[Tsim1(end,1) Tsim1(end,2)],[],0);
[tsim2a,Tsim2a] = ode45(@DGL_System,dt2,[Tsim2(end,1) Tsim2(end,2)],[],0);
[tsim3a,Tsim3a] = ode45(@DGL_System,dt2,[Tsim3(end,1) Tsim3(end,2)],[],0);
```

```
[tsim4a,Tsim4a] = ode45(@DGL_System,dt2,[Tsim4(end,1) Tsim4(end,2)],[],0);
[tsim5a,Tsim5a] = ode45(@DGL_System,dt2,[Tsim5(end,1) Tsim5(end,2)],[],0);
[tsim6a,Tsim6a] = ode45(@DGL_System,dt2,[Tsim6(end,1) Tsim6(end,2)],[],0);
[tsim7a,Tsim7a] = ode45(@DGL_System,dt2,[Tsim7(end,1) Tsim7(end,2)],[],0);

%plotten der Daten
evaluation
```

A.4 Skripte für die Modellierung

Listing A.3 und Listing A.4 enthalten Skripte, die zur Ansteuerung der Regelstrecke im Rahmen der Untersuchung des Ein- und Ausgangsverhaltens dienen. Hierauf wird in Abschnitt 3.7.5 näher eingegangen.

Listing A.3: Skript zur Ansteuerung der Strecke für die Modellierung des Kühlbetriebs.

```
/*********************************************************************
*  Testskript zur Messung der Kühlübertragungsfunktion
*  Lukas Bommes 2016
*********************************************************************/

// Hilsvariablen
double Sample_Zeit = 100000;
String send_string = String(); // String für serielle Ausgabe
static volatile uint8_t isr_active_flag = 0; // ISR_Flag für das Senden der Daten

// Betriebsvariablen
double T1 = 0.0; // Temperatur der Spannplatte
double T1_ADC = T1; // entspricht ADC-Wert von T1
double T2 = 0.0; // Temperatur des Wärmetauschers
double T2_ADC = T2; // entspricht ADC-Wert von T2

// Testdatenvektor (augegebener Sprung des PWM-Duty Cycles)
double TEC_CURRENT[27] =
{0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0,255,0};
int i = 0, k = 0;

void setup()
{
  Serial.begin(115200);
  Serial.setTimeout(10);

  digitalWrite(10,LOW); // H-Brücke auf Kühlbetrieb setzen

  // Timer1 entsprechend der Sample-Zeit einstellen
  cli(); // Interrupts anhalten
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0; // Counter mit 0 initialisieren
  OCR1A = ((16.0/256.0)*Sample_Zeit)-1.0; // (muss < 65536 sein) // Compare Match Register
  setzen
```

```
34   TCCR1B |= (1 << WGM12); // CTC-Modus
35   TCCR1B |= (1 << CS12); // Prescaler = 256
36   TIMSK1 |= (1 << OCIE1A); // Timer Compare Interrupt aktivieren
37   sei(); // Interrupts erlauben
38 }
39
40 // ISR-Routine wird von Timer1 entsprechend Sample_Zeit aufgerufen
41 ISR(TIMER1_COMPA_vect)
42 {
43   // Flag setzen führt zur Ausführung des entsprechenden Codes in der main-loop
44   isr_active_flag = 1;
45 }
46
47 // Daten an serielle Schnittstelle senden
48 void serialSendData(int k)
49 {
50   // Variablen zu einem String zusammenführen
51   send_string += millis(); // Zeit
52   send_string += ,;
53   send_string += T1_ADC; // Temperatur der Spannplatte
54   send_string += ,;
55   send_string += T2_ADC; // Temperatur des Wärmetauschers
56   send_string += ,;
57   send_string += T1; // Temperatur der Spannplatte
58   send_string += ,;
59   send_string += T2; // Temperatur des Wärmetauschers
60   send_string += ,;
61   send_string += TEC_CURRENT[k]; // sprungförmig aufgeprägte Eingangsgröße in Prozent
62
63   // String an die serielle Schnittstelle senden
64   Serial.println(send_string);
65
66   // String löschen
67   send_string = ;
68 }
69
70 // Funktion rechnet ADC-Werte (0..1023) in Grad Celsius um
71 double adcToDegreeCelsius(int adc_value)
72 {
73   // Ausgleichsgerade mit -15 Grad Celsius..75 Grad Celsius und 0..1023
74   return 0.087977 * adc_value - 15.0;
75 }
76
77 void loop() {
78   // erfüllt, wenn die ISR aufgerufen wurde
79   if(isr_active_flag)
80   {
81     isr_active_flag = 0;
82
83     // erhöhe k alle 120 Sekunden
84     if(i == 1200) {
85       i = 0;
86       k++;
87     }
```

```
88      i++;
89
90      // so lange durchführen, bis gesamter Testdaten-Vektor durchlaufen wurde
91      if(k < 27)
92      {
93        // Ein- und Ausgangsgrößen einlesen bzw. setzen
94        T1_ADC = analogRead(1); // ADC-Wert der Spannplattentemperatur
95        T2_ADC = analogRead(0); // ADC-Wert der Wärmetauschertemperatur
96        T1 = adcToDegreeCelsius(T1_ADC); // Temperatur der Spannplatte einlesen
97        T2 = adcToDegreeCelsius(T2_ADC); // Temperatur des Wärmetauschers einlesen
98        analogWrite(3,TEC_CURRENT[k]); // Stellgröße auf Pin 3 ausgeben
99
100       // Daten auf serieller Schnittstelle ausgeben
101       serialSendData(k);
102     }
103   }
104
105   // Notabschaltung bei zu hohen Temperaturen
106   if(T1 >= 50.0) analogWrite(3,0);
107   if(T2 >= 60.0) analogWrite(3,0);
108 }
```

Listing A.4: Skript zur Ansteuerung der Strecke für die Modellierung des Heizbetriebs.

```
1  /*********************************************************************
2  *  Testskript zur Messung der Heizübertragungsfunktion
3  *  Lukas Bommes 2016
4  *********************************************************************/
5
6  // Hilsvariablen
7  double Sample_Zeit = 100000;
8  String send_string = String(); // String für serielle Ausgabe
9  static volatile uint8_t isr_active_flag = 0; // ISR_Flag für das Senden der Daten
10
11 // Betriebsvariablen
12 double T1 = 0.0; // Temperatur der Spannplatte
13 double T1_ADC = T1; // entspricht ADC-Wert von T1
14 double T2 = 0.0; // Temperatur des Wärmetauschers
15 double T2_ADC = T2; // entspricht ADC-Wert von T2
16
17 // Sollwert für Spannplattentemperatur
18 double TEC_CURRENT = 0;
19
20 void setup()
21 {
22   Serial.begin(115200);
23   Serial.setTimeout(10);
24
25   digitalWrite(10,HIGH); // H-Brücke auf Heizbetrieb setzen
26
27   // Timer1 entsprechend der Sample-Zeit einstellen
28   cli(); // Interrupts anhalten
29   TCCR1A = 0;
```

```
30   TCCR1B = 0;
31   TCNT1  = 0; // Counter mit 0 initialisieren
32   OCR1A = ((16.0/256.0)*Sample_Zeit)-1.0; // (muss < 65536 sein) // Compare Match Register
     setzen
33   TCCR1B |= (1 << WGM12); // CTC-Modus
34   TCCR1B |= (1 << CS12); // Prescaler = 256
35   TIMSK1 |= (1 << OCIE1A); // Timer Compare Interrupt aktivieren
36   sei(); // Interrupts erlauben
37 }
38
39 // ISR-Routine wird von Timer1 entsprechend Sample_Zeit aufgerufen
40 ISR(TIMER1_COMPA_vect)
41 {
42   // Flag setzen führt zur Ausführung des entsprechenden Codes in der main-loop
43   isr_active_flag = 1;
44 }
45
46 // Daten an serielle Schnittstelle senden
47 void serialSendData()
48 {
49   // Variablen zu einem String zusammenführen
50   send_string += millis(); // Zeit
51   send_string += ,;
52   send_string += T1_ADC; // Temperatur der Spannplatte
53   send_string += ,;
54   send_string += T2_ADC; // Temperatur des Wärmetauschers
55   send_string += ,;
56   send_string += T1; // Temperatur der Spannplatte
57   send_string += ,;
58   send_string += T2; // Temperatur des Wärmetauschers
59   send_string += ,;
60   send_string += TEC_CURRENT; // sprungförmig aufgeprägte Eingangsgröße in Prozent
61
62   // String an die serielle Schnittstelle senden
63   Serial.println(send_string);
64
65   // String löschen
66   send_string = ;
67 }
68
69 // Funktion rechnet ADC-Werte (0..1023) in Grad Celsius um
70 double adcToDegreeCelsius(int adc_value)
71 {
72   // Ausgleichsgerade mit -15 Grad Celsius..75 Grad Celsius und 0..1023
73   return 0.087977 * adc_value - 15.0;
74 }
75
76 void loop() {
77
78   // Sollwert vom Potentiometer einlesen
79   TEC_CURRENT = map(analogRead(3),0,1023,0,255);
80
81   // erfüllt, wenn die ISR aufgerufen wurde
82   if(isr_active_flag)
```

```
83   {
84     isr_active_flag = 0;
85
86     // Ein- und Ausgangsgrößen einlesen bzw. setzen
87     T1_ADC = analogRead(1); // ADC-Wert der Spannplattentemperatur
88     T2_ADC = analogRead(0); // ADC-Wert der Wärmetauschertemperatur
89     T1 = adcToDegreeCelsius(T1_ADC); // Temperatur der Spannplatte einlesen
90     T2 = adcToDegreeCelsius(T2_ADC); // Temperatur des Wärmetauschers einlesen
91     analogWrite(3,TEC_CURRENT); // Stellgröße auf Pin 3 ausgeben
92
93     // Daten auf serieller Schnittstelle ausgeben
94     serialSendData();
95   }
96
97   // Notabschaltung bei zu hoher Temperatur der Spannplatte
98   if(T1 >= 74.0) analogWrite(3,0);
99   if(T2 >= 74.0) analogWrite(3,0);
100 }
```

A.5 Sourcecode der GUI

In Listing A.5 ist der bereits in Abschnitt 5.3 im Detail vorgestellte Quellcode des graphischen Anwenderprogramms in zusammenhängender Form dargestellt. Listing A.6 enthält ein Skript zur Auflistung verfügbarer COM-Ports. Es wird in Abschnitt 5.3.2 näher beschrieben.

Listing A.5: Hauptprogramm der graphischen Nutzeroberfläche der Steuerungssoftware.

```
1  function varargout = Controller_GUI(varargin)
2  % CONTROLLER_GUI MATLAB code for Controller_GUI.fig
3  % See documentation for more information.
4
5  % Last Modified by GUIDE v2.5 17-Apr-2016 23:52:43
6
7  % Begin initialization code - DO NOT EDIT
8  gui_Singleton = 1;
9  gui_State = struct('gui_Name',       mfilename, ...
10                    'gui_Singleton',  gui_Singleton, ...
11                    'gui_OpeningFcn', @Controller_GUI_OpeningFcn, ...
12                    'gui_OutputFcn',  @Controller_GUI_OutputFcn, ...
13                    'gui_LayoutFcn',  [] , ...
14                    'gui_Callback',   []);
15 if nargin && ischar(varargin{1})
16     gui_State.gui_Callback = str2func(varargin{1});
17 end
18
19 if nargout
20     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
21 else
22     gui_mainfcn(gui_State, varargin{:});
23 end
```

```
24 % End initialization code - DO NOT EDIT
25
26
27 % --- Executes just before Controller_GUI is made visible.
28 function Controller_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
29 % Setup default values for parameters
30 handles = declareDefaults(handles);
31
32 % miscellaneous
33 handles.s = 0; % initialise serial port object
34 handles.logFileName = ['logging\','Messung_',datestr(now,'dd-mmm-yyyy_HH-MM-SS'),'.txt']; %
   initial filename for logfile
35
36 % axes setup
37 xlabel(handles.Ausgabeplot,'Zeit t in s');
38 ylim([handles.T_MIN-5.0 handles.T_MAX+5.0]);
39 set(handles.Ausgabeplot,'Box','off');
40 % create new axes on top with y axis on the right
41 handles.Ausgabeplot2 = axes('Units','character');
42 set(handles.Ausgabeplot2,'Position',get(handles.Ausgabeplot,'Position'),'Color','none','YLim'
   ,[0 105],'YAxisLocation','right','XAxisLocation','top','XTickLabel',[],'Box','off','XMinorTick'
   ,'on','YMinorTick','on');
43 linkaxes([handles.Ausgabeplot,handles.Ausgabeplot2],'x'); % link x axis of both axes
44 % define plot lines for usage in callback-function
45 handles.h1 = animatedline('Parent',handles.Ausgabeplot,'Color','r','LineWidth',1.5); % T1
46 handles.h2 = animatedline('Parent',handles.Ausgabeplot,'Color','b','LineWidth',1.5); % T2
47 handles.h3 = animatedline('Parent',handles.Ausgabeplot,'Color','c','LineWidth',1.5); % T1_SOLL
48 handles.h4 = animatedline('Parent',handles.Ausgabeplot2,'Color','g','LineWidth',1.5); %
   TEC_CURRENT
49 % legend setup
50 legend_handle = legend([handles.h4;handles.h3;handles.h2;handles.h1],'Stellgröße in %','
   Sollwert T1\_SOLL/°C','Temperatur T2/°C','Temperatur T1/°C');
51 set(legend_handle,'color','w');
52 % change second last y axis tick to unit symbol
53 Myl = get(handles.Ausgabeplot,'YTickLabel'); % put left y-axis ticks into matrix
54 Myl{length(Myl)-1} = '°C';
55 set(handles.Ausgabeplot,'YTickLabel',Myl); % set new axis ticks with units inserted
56 Myr = get(handles.Ausgabeplot2,'YTickLabel');
57 Myr{length(Myr)-1} = '%';
58 set(handles.Ausgabeplot2,'YTickLabel',Myr);
59
60 % create profiles folder if not already there
61 if ~exist('profiles','dir')
62     mkdir('profiles');
63 end
64
65 % create logging folder if not already there
66 if ~exist('logging','dir')
67     mkdir('logging');
68 end
69
70 % Choose default command line output for Controller_GUI
71 handles.output = hObject;
72
```

```
% Update handles structure
guidata(hObject, handles);


% --- Creates handles structure with default values, can be used for resetting to defaults
function handles = declareDefaults(handles)
% create user data in handle object
handles.Betriebsmodus = 1;
handles.Zustand = 0;
handles.Aktion = 0;
handles.Zeit = 0;
handles.T1 = 0;
handles.T2 = 0;
handles.TEC_CURRENT = 0;
handles.T1_SOLL = 0;
% constants for system setup
handles.KP_HEIZEN = 63.89;
handles.KI_HEIZEN = 7.80;
handles.KP_KUEHLEN = -13.54;
handles.KI_KUEHLEN = -3.49;
handles.T_WACHS_SCHMELZ = 70.0;
handles.T_WACHS_ERSTARR = 60.0;
handles.T_WACHS_ABKUEHL = -15.0;
handles.T_WASSER_FRIER = -15.0;
handles.T_WASSER_ERSTARR = 0.0;
handles.DAUER_SCHMELZEN = 8000;
handles.DAUER_ABKUEHLEN = 15000;
% constant temperature limits (just for checking input values)
handles.T_MIN = -15.0;
handles.T_MAX = 75.0;


% --- Outputs from this function are returned to the command line.
function varargout = Controller_GUI_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in ports_popupmenu.
function ports_popupmenu_Callback(hObject, eventdata, handles)
% retireve selected COM-Port from popupmenu
handles.portSelected = handles.portsAvailable{get(hObject,'Value')};

% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function ports_popupmenu_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% search for available com ports
handles.portsAvailable = getAvailableComPort();
```

```
127 set(hObject,'String',handles.portsAvailable);
128 handles.portSelected = handles.portsAvailable{get(hObject,'Value')};
129
130 % Update handles structure
131 guidata(hObject, handles);
132
133
134 % --- Executes on button press in logfile_checkbox.
135 function logfile_checkbox_Callback(hObject, eventdata, handles)
136 % if already connected and then logfile button checked
137 if get(handles.Connect_togglebutton,'Value') == get(handles.Connect_togglebutton,'Max')
138     createLogFile(hObject, handles);
139 end
140
141 % Update handles structure
142 guidata(hObject, handles);
143
144
145 % --- Executes on button press in Connect_togglebutton.
146 function Connect_togglebutton_Callback(hObject, eventdata, handles)
147 button_state = get(hObject,'Value');
148 % on click of 'Verbinden'
149 if button_state == get(hObject,'Max')
150     % delete any existing port objects
151     delete(instrfindall);
152     % create serial port object
153     handles.s = serial(handles.portSelected,'BaudRate',115200);
154     fclose(handles.s);
155     % specify callbackfunction, that is executed everytime the terminator '\n' is read on
    serial port
156     handles.s.BytesAvailableFcnMode = 'terminator';
157     handles.s.BytesAvailableFcn = {@serial_read_data, hObject, handles};
158     fopen(handles.s);
159     % change button string
160     set(hObject,'String','Trennen');
161     % enable buttons and radio boxes and edits
162     set(handles.Heizen_radiobutton,'Enable','on');
163     set(handles.Kuehlen_radiobutton,'Enable','on');
164     set(handles.Spannen_pushbutton,'Enable','on');
165     set(handles.Loesen_pushbutton,'Enable','on');
166     set(handles.Abbrechen_pushbutton,'Enable','on');
167     set(handles.KP_HEIZEN_edit,'Enable','on');
168     set(handles.KI_HEIZEN_edit,'Enable','on');
169     set(handles.KP_KUEHLEN_edit,'Enable','on');
170     set(handles.KI_KUEHLEN_edit,'Enable','on');
171     set(handles.T_WACHS_SCHMELZ_edit,'Enable','on');
172     set(handles.T_WACHS_ERSTARR_edit,'Enable','on');
173     set(handles.T_WACHS_ABKUEHL_edit,'Enable','on');
174     set(handles.T_WASSER_FRIER_edit,'Enable','on');
175     set(handles.T_WASSER_ERSTARR_edit,'Enable','on');
176     set(handles.DAUER_SCHMELZEN_edit,'Enable','on');
177     set(handles.DAUER_ABKUEHLEN_edit,'Enable','on');
178     % create new logfile on connect if, checkbox checked
179     createLogFile(hObject, handles);
```

```
% on click of 'Trennen'
elseif button_state == get(hObject,'Min')
    % Clean up serial port if connected
    if handles.s ~= 0
        fclose(handles.s);
        delete(handles.s);
        clear handles.s;
        delete(instrfindall);
    end
    handles.s = 0;
    % change button string
    set(hObject,'String','Verbinden');
    % disable buttons and radio boxes and edits
    set(handles.Heizen_radiobutton,'Enable','off');
    set(handles.Kuehlen_radiobutton,'Enable','off');
    set(handles.Spannen_pushbutton,'Enable','off');
    set(handles.Loesen_pushbutton,'Enable','off');
    set(handles.Abbrechen_pushbutton,'Enable','off');
    set(handles.KP_HEIZEN_edit,'Enable','off');
    set(handles.KI_HEIZEN_edit,'Enable','off');
    set(handles.KP_KUEHLEN_edit,'Enable','off');
    set(handles.KI_KUEHLEN_edit,'Enable','off');
    set(handles.T_WACHS_SCHMELZ_edit,'Enable','off');
    set(handles.T_WACHS_ERSTARR_edit,'Enable','off');
    set(handles.T_WACHS_ABKUEHL_edit,'Enable','off');
    set(handles.T_WASSER_FRIER_edit,'Enable','off');
    set(handles.T_WASSER_ERSTARR_edit,'Enable','off');
    set(handles.DAUER_SCHMELZEN_edit,'Enable','off');
    set(handles.DAUER_ABKUEHLEN_edit,'Enable','off');
    % reset plot
    clearpoints(handles.h1);
    clearpoints(handles.h2);
    clearpoints(handles.h3);
    clearpoints(handles.h4);
    xlim(handles.Ausgabeplot,[0 30]);
    % Reset to default parameters and update widgest
    handles = declareDefaults(handles);
    outputDataToWidgets(handles); % output data to widgets
    % Reset status
    set(handles.Status_text,'String','Status: System getrennt, keine Daten emfpangen');
end

% Update handles structure
guidata(hObject, handles);


% ---- Creates logfile and writes file header
function createLogFile(hObject, handles)
if get(handles.logfile_checkbox,'Value') == 1
    % change filename according to current date and time
    handles.logFileName = ['logging\','Messung_', datestr(now,'dd-mmm-yyyy_HH-MM-SS'),'.txt'];
    % update handles in serial read data callback
    handles.s.BytesAvailableFcn = {@serial_read_data, hObject, handles};
    % write fileheader
```

```
234     fid = fopen(handles.logFileName, 'wt');
235     fprintf(fid,'Betriebsmodus, Zustand, Zeit, T1_SOLL, T1, T2, TEC_CURRENT');
236     fprintf(fid,'\n');
237     fclose(fid);
238 end
239 
240 
241 % --- Executes on button press in Kuehlen_radiobutton.
242 function Kuehlen_radiobutton_Callback(hObject, eventdata, handles)
243 handles.Betriebsmodus = 1; % Kühlen
244 % send command to serial
245 serial_send_commands(handles);
246 
247 % Update handles structure
248 guidata(hObject, handles);
249 
250 
251 % --- Executes on button press in Heizen_radiobutton.
252 function Heizen_radiobutton_Callback(hObject, eventdata, handles)
253 handles.Betriebsmodus = 0; % Heizen
254 % send command to serial
255 serial_send_commands(handles);
256 
257 % Update handles structure
258 guidata(hObject, handles);
259 
260 
261 % --- Executes on button press in Spannen_pushbutton.
262 function Spannen_pushbutton_Callback(hObject, eventdata, handles)
263 handles.Aktion = 1; % Spannen
264 % send command to serial
265 serial_send_commands(handles);
266 
267 % Update handles structure
268 guidata(hObject, handles);
269 
270 
271 % --- Executes on button press in Loesen_pushbutton.
272 function Loesen_pushbutton_Callback(hObject, eventdata, handles)
273 handles.Aktion = 2; % Loesen
274 % send command to serial
275 serial_send_commands(handles);
276 
277 % Update handles structure
278 guidata(hObject, handles);
279 
280 
281 % --- Executes on button press in Abbrechen_pushbutton.
282 function Abbrechen_pushbutton_Callback(hObject, eventdata, handles)
283 handles.Aktion = 3; % Abbrechen
284 % send command to serial
285 serial_send_commands(handles);
286 
287 % Update handles structure
```

```
guidata(hObject, handles);


% --- Update parameters based upon text in edit box
function KP_HEIZEN_edit_Callback(hObject, eventdata, handles)
val = str2num(get(hObject,'String'));
if ~isempty(val) % check if value is numeric
    if val >= 0 % check if value is valid
        handles.KP_HEIZEN = val;
        % send command to serial
        serial_send_commands(handles);
    end
end
% Update handles structure
guidata(hObject, handles);


% --- Update parameters based upon text in edit box
function KI_HEIZEN_edit_Callback(hObject, eventdata, handles)
val = str2num(get(hObject,'String'));
if ~isempty(val) % check if value is numeric
    if val >= 0 % check if value is valid
        handles.KI_HEIZEN = val;
        % send command to serial
        serial_send_commands(handles);
    end
end
% Update handles structure
guidata(hObject, handles);


% --- Update parameters based upon text in edit box
function KP_KUEHLEN_edit_Callback(hObject, eventdata, handles)
val = str2num(get(hObject,'String'));
if ~isempty(val) % check if value is numeric
    if val <= 0 % check if value is valid
        handles.KP_KUEHLEN = val;
        % send command to serial
        serial_send_commands(handles);
    end
end
% Update handles structure
guidata(hObject, handles);


% --- Update parameters based upon text in edit box
function KI_KUEHLEN_edit_Callback(hObject, eventdata, handles)
val = str2num(get(hObject,'String'));
if ~isempty(val) % check if value is numeric
    if val <= 0 % check if value is valid
        handles.KI_KUEHLEN = val;
        % send command to serial
        serial_send_commands(handles);
    end
```

```
342 end
343 % Update handles structure
344 guidata(hObject, handles);
345
346
347 % --- Update parameters based upon text in edit box
348 function T_WACHS_SCHMELZ_edit_Callback(hObject, eventdata, handles)
349 val = str2num(get(hObject,'String'));
350 if ~isempty(val) % check if value is numeric
351     if val >= handles.T_MIN && val <= handles.T_MAX % check if value is valid
352         handles.T_WACHS_SCHMELZ = val;
353         % send command to serial
354         serial_send_commands(handles);
355     end
356 end
357 % Update handles structure
358 guidata(hObject, handles);
359
360
361 % --- Update parameters based upon text in edit box
362 function T_WACHS_ERSTARR_edit_Callback(hObject, eventdata, handles)
363 val = str2num(get(hObject,'String'));
364 if ~isempty(val) % check if value is numeric
365     if val >= handles.T_MIN && val <= handles.T_WACHS_SCHMELZ % check if value is valid
366         handles.T_WACHS_ERSTARR = val;
367         % send command to serial
368         serial_send_commands(handles);
369     end
370 end
371 % Update handles structure
372 guidata(hObject, handles);
373
374
375 % --- Update parameters based upon text in edit box
376 function T_WACHS_ABKUEHL_edit_Callback(hObject, eventdata, handles)
377 val = str2num(get(hObject,'String'));
378 if ~isempty(val) % check if value is numeric
379     if val >= handles.T_MIN && val <= handles.T_WACHS_ERSTARR % check if value is valid
380         handles.T_WACHS_ABKUEHL = val;
381         % send command to serial
382         serial_send_commands(handles);
383     end
384 end
385 % Update handles structure
386 guidata(hObject, handles);
387
388
389 % --- Update parameters based upon text in edit box
390 function T_WASSER_FRIER_edit_Callback(hObject, eventdata, handles)
391 val = str2num(get(hObject,'String'));
392 if ~isempty(val) % check if value is numeric
393     if val >= handles.T_MIN && val <= handles.T_MAX % check if value is valid
394         handles.T_WASSER_FRIER = val;
395         % send command to serial
```

```
396         serial_send_commands(handles);
397     end
398 end
399 % Update handles structure
400 guidata(hObject, handles);
401
402
403 % --- Update parameters based upon text in edit box
404 function T_WASSER_ERSTARR_edit_Callback(hObject, eventdata, handles)
405 val = str2num(get(hObject,'String'));
406 if ~isempty(val) % check if value is numeric
407     if val >= handles.T_WASSER_FRIER && val <= handles.T_MAX % check if value is valid
408         handles.T_WASSER_ERSTARR = val;
409         % send command to serial
410         serial_send_commands(handles);
411     end
412 end
413 % Update handles structure
414 guidata(hObject, handles);
415
416
417 % --- Update parameters based upon text in edit box
418 function DAUER_SCHMELZEN_edit_Callback(hObject, eventdata, handles)
419 val = str2num(get(hObject,'String'));
420 if ~isempty(val) % check if value is numeric
421     if val >= 0 % check if value is valid
422         handles.DAUER_SCHMELZEN = val;
423         % send command to serial
424         serial_send_commands(handles);
425     end
426 end
427 % Update handles structure
428 guidata(hObject, handles);
429
430
431 % --- Update parameters based upon text in edit box
432 function DAUER_ABKUEHLEN_edit_Callback(hObject, eventdata, handles)
433 val = str2num(get(hObject,'String'));
434 if ~isempty(val) % check if value is numeric
435     if val >= 0 % check if value is valid
436         handles.DAUER_ABKUEHLEN = val;
437         % send command to serial
438         serial_send_commands(handles);
439     end
440 end
441 % Update handles structure
442 guidata(hObject, handles);
443
444
445 % --- Writes commands to serial port
446 function serial_send_commands(handles)
447 if handles.s ~= 0
448     string = [num2str(handles.Betriebsmodus),',',num2str(handles.Aktion)...
449         ,',',num2str(handles.KP_HEIZEN),',',num2str(handles.KI_HEIZEN)...
```

```
450         ,',',num2str(handles.KP_KUEHLEN),',',num2str(handles.KI_KUEHLEN)...
451         ,',',num2str(handles.T_WACHS_SCHMELZ),',',num2str(handles.T_WACHS_ERSTARR)...
452         ,',',num2str(handles.T_WACHS_ABKUEHL),',',num2str(handles.T_WASSER_FRIER)...
453         ,',',num2str(handles.T_WASSER_ERSTARR),',',num2str(handles.DAUER_SCHMELZEN)...
454         ,',',num2str(handles.DAUER_ABKUEHLEN)];
455     fprintf(handles.s,string);
456 end
457
458
459 % --- Executes during object creation, after setting all properties.
460 function Profile_listbox_CreateFcn(hObject, eventdata, handles)
461 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
462     set(hObject,'BackgroundColor','white');
463 end
464 % load existing profiles into listbox
465 profiles_available = getAvailableProfileNames();
466 % put items into listbox
467 set(hObject,'String',profiles_available);
468
469
470 % --- Executes on button press in add_pushbutton.
471 function add_pushbutton_Callback(hObject, eventdata, handles)
472 profil_name = get(handles.Profil_name_edit,'String');
473 % check, if entered profile name already exist in the directory
474 profiles_available = getAvailableProfileNames();
475 exist_flag = 0;
476 for i = 1:length(profiles_available)
477     if strcmp(profiles_available{i},profil_name)
478         exist_flag = 1;
479     end
480 end
481 % only run this code, if profile name doesn't exist
482 if ~exist_flag
483     fid = fopen(['profiles\',profil_name,'.prf'], 'wt');
484     fprintf(fid,[num2str(handles.KP_HEIZEN),',',num2str(handles.KI_HEIZEN)...
485         ,',',num2str(handles.KP_KUEHLEN),',',num2str(handles.KI_KUEHLEN)...
486         ,',',num2str(handles.T_WACHS_SCHMELZ),',',num2str(handles.T_WACHS_ERSTARR)...
487         ,',',num2str(handles.T_WACHS_ABKUEHL),',',num2str(handles.T_WASSER_FRIER)...
488         ,',',num2str(handles.T_WASSER_ERSTARR),',',num2str(handles.DAUER_SCHMELZEN)...
489         ,',',num2str(handles.DAUER_ABKUEHLEN)]);
490     fclose(fid);
491
492     % add item to listbox with name entered in textedit
493     set(handles.Profile_listbox,'Value',1);
494     prev_Content = get(handles.Profile_listbox,'String');
495     % don't copy old content on empty ist
496     if isempty(prev_Content)
497         new_content = {profil_name};
498     else
499         new_content = [cellstr(prev_Content);{profil_name}];
500     end
501     set(handles.Profile_listbox,'String',new_content);
502 end
503 % Update handles structure
```

```
guidata(hObject, handles);


% ---- Returns names (without extension) of profiles available in profile directory
function profiles_available_names_without_ext = getAvailableProfileNames()
% list all *.prf-files
profiles_available = dir('profiles\*.prf');
% retrieve names of all *.prf-files
profiles_available_names = {profiles_available(:).name};
% get the name of available profiles without file extension
profiles_available_names_without_ext = [];
for i = 1:length(profiles_available_names)
    [f_pathstr,f_name,f_ext] = fileparts(profiles_available(i).name);
    profiles_available_names_without_ext{end+1,1} = f_name;
end


% ---- Executes on button press in delete_pushbutton.
function delete_pushbutton_Callback(hObject, eventdata, handles)
% delete selected profile
current_Selection = get(handles.Profile_listbox,'Value');
current_Items = get(handles.Profile_listbox,'String');
% check if profiles available
if ~isempty(current_Items)
    % delete profile-file if file exists
    if exist(['profiles\',current_Items{current_Selection},'.prf'],'file') > 0
        delete(['profiles\',current_Items{current_Selection},'.prf']);
    end
    % delete selected item from list
    current_Items(current_Selection) = [];
    set(handles.Profile_listbox,'String',current_Items,'Value',1);
end


% ---- Executes on button press in Laden_pushtbutton.
function Laden_pushtbutton_Callback(hObject, eventdata, handles)
if handles.Zustand == 0 && handles.s ~= 0 % loading only possible if Zustand = 'BEREIT' and
connected
    % get selected item
    current_Selection = get(handles.Profile_listbox,'Value');
    current_Items = get(handles.Profile_listbox,'String');
    % read values from file and set parameters accordingly
    imported_data = csvread(['profiles\',current_Items{current_Selection},'.prf']);
    handles.KP_HEIZEN = imported_data(1);
    handles.KI_HEIZEN = imported_data(2);
    handles.KP_KUEHLEN = imported_data(3);
    handles.KI_KUEHLEN = imported_data(4);
    handles.T_WACHS_SCHMELZ = imported_data(5);
    handles.T_WACHS_ERSTARR = imported_data(6);
    handles.T_WACHS_ABKUEHL = imported_data(7);
    handles.T_WASSER_FRIER = imported_data(8);
    handles.T_WASSER_ERSTARR = imported_data(9);
    handles.DAUER_SCHMELZEN = imported_data(10);
    handles.DAUER_ABKUEHLEN = imported_data(11);
```

```
557
558     % set contents of edit boxes
559     set(handles.KP_HEIZEN_edit,'String',sprintf('%0.2f',handles.KP_HEIZEN));
560     set(handles.KI_HEIZEN_edit,'String',sprintf('%0.2f',handles.KI_HEIZEN));
561     set(handles.KP_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KP_KUEHLEN));
562     set(handles.KI_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KI_KUEHLEN));
563     set(handles.T_WACHS_SCHMELZ_edit,'String',sprintf('%0.1f',handles.T_WACHS_SCHMELZ));
564     set(handles.T_WACHS_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WACHS_ERSTARR));
565     set(handles.T_WACHS_ABKUEHL_edit,'String',sprintf('%0.1f',handles.T_WACHS_ABKUEHL));
566     set(handles.T_WASSER_FRIER_edit,'String',sprintf('%0.1f',handles.T_WASSER_FRIER));
567     set(handles.T_WASSER_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WASSER_ERSTARR));
568     set(handles.DAUER_SCHMELZEN_edit,'String',handles.DAUER_SCHMELZEN);
569     set(handles.DAUER_ABKUEHLEN_edit,'String',handles.DAUER_ABKUEHLEN);
570
571     % send command to serial
572     serial_send_commands(handles);
573 end
574
575 % Update handles structure
576 guidata(hObject, handles);
577
578
579 % --- Executes when user attempts to close gui_window.
580 function gui_window_CloseRequestFcn(hObject, eventdata, handles)
581 % Clean up serial port if connected
582 if handles.s ~= 0
583     fclose(handles.s);
584     delete(handles.s);
585     clear handles.s;
586     delete(instrfindall);
587 end
588
589 % close the figure
590 delete(hObject);
591
592
593 % --- Event Handler for Reading data from serial port
594 function serial_read_data(s, event, hObject, handles)
595 % read data from serial port
596 serial_incoming_data = fscanf(s,'%s');
597 set(handles.serialOutput,'String',serial_incoming_data); % Raw Data Label
598 % if an error occurs within try block, following commands in try are skipped
599 % and error label will be set in catch block. Prevents system from crashing
600 % due to faulty serial reads.
601 try
602     % parse incoming data
603     converted_data = eval(['[',serial_incoming_data,']']);
604     % update variables based on data
605     handles.Betriebsmodus = converted_data(1);
606     handles.Zustand = converted_data(2);
607     converted_data(3) = converted_data(3)/1000; % change time format into seconds
608     handles.Zeit = converted_data(3);
609     handles.T1_SOLL = converted_data(4);
610     handles.T1 = converted_data(5);
```

```
611     handles.T2 = converted_data(6);
612     handles.TEC_CURRENT = converted_data(7);
613     handles.KP_HEIZEN = converted_data(8);
614     handles.KI_HEIZEN = converted_data(9);
615     handles.KP_KUEHLEN = converted_data(10);
616     handles.KI_KUEHLEN = converted_data(11);
617     handles.T_WACHS_SCHMELZ = converted_data(12);
618     handles.T_WACHS_ERSTARR = converted_data(13);
619     handles.T_WACHS_ABKUEHL = converted_data(14);
620     handles.T_WASSER_FRIER = converted_data(15);
621     handles.T_WASSER_ERSTARR = converted_data(16);
622     handles.DAUER_SCHMELZEN = converted_data(17);
623     handles.DAUER_ABKUEHLEN = converted_data(18);
624 
625     % write data to logfile, if checkbox is checked
626     if get(handles.logfile_checkbox,'Value') == 1
627         dlmwrite(handles.logFileName,converted_data(1:7),'-append','newline','pc');
628     end
629 
630     % output data to widgest (text boxes and edit boxes)
631     outputDataToWidgets(handles);
632 
633     % deactive buttons corresponding to 'Zustand'
634     switch handles.Zustand
635         case 0 % BEREIT
636             set(handles.Spannen_pushbutton,'Enable','on');
637             set(handles.Loesen_pushbutton,'Enable','off');
638             set(handles.Abbrechen_pushbutton,'Enable','off');
639         case 1 % SPANNEN
640             set(handles.Spannen_pushbutton,'Enable','off');
641             set(handles.Loesen_pushbutton,'Enable','off');
642             set(handles.Abbrechen_pushbutton,'Enable','on');
643         case 2 % LOESEN
644             set(handles.Spannen_pushbutton,'Enable','off');
645             set(handles.Loesen_pushbutton,'Enable','off');
646             set(handles.Abbrechen_pushbutton,'Enable','on');
647         case 3 % GESPANNT
648             set(handles.Spannen_pushbutton,'Enable','off');
649             set(handles.Loesen_pushbutton,'Enable','on');
650             set(handles.Abbrechen_pushbutton,'Enable','off');
651     end
652 
653     % deactive edit-boxes corresponding to 'Zustand'
654     if handles.Zustand == 0; % BEREIT
655         set(handles.KP_HEIZEN_edit,'Enable','on');
656         set(handles.KI_HEIZEN_edit,'Enable','on');
657         set(handles.KP_KUEHLEN_edit,'Enable','on');
658         set(handles.KI_KUEHLEN_edit,'Enable','on');
659         set(handles.T_WACHS_SCHMELZ_edit,'Enable','on');
660         set(handles.T_WACHS_ERSTARR_edit,'Enable','on');
661         set(handles.T_WACHS_ABKUEHL_edit,'Enable','on');
662         set(handles.T_WASSER_FRIER_edit,'Enable','on');
663         set(handles.T_WASSER_ERSTARR_edit,'Enable','on');
664         set(handles.DAUER_SCHMELZEN_edit,'Enable','on');
```

```
665         set(handles.DAUER_ABKUEHLEN_edit,'Enable','on');
666     else % SPANNEN, LOESEN, GESPANNT
667         set(handles.KP_HEIZEN_edit,'Enable','off');
668         set(handles.KI_HEIZEN_edit,'Enable','off');
669         set(handles.KP_KUEHLEN_edit,'Enable','off');
670         set(handles.KI_KUEHLEN_edit,'Enable','off');
671         set(handles.T_WACHS_SCHMELZ_edit,'Enable','off');
672         set(handles.T_WACHS_ERSTARR_edit,'Enable','off');
673         set(handles.T_WACHS_ABKUEHL_edit,'Enable','off');
674         set(handles.T_WASSER_FRIER_edit,'Enable','off');
675         set(handles.T_WASSER_ERSTARR_edit,'Enable','off');
676         set(handles.DAUER_SCHMELZEN_edit,'Enable','off');
677         set(handles.DAUER_ABKUEHLEN_edit,'Enable','off');
678     end
679 
680     % plot data points
681     addpoints(handles.h1,handles.Zeit,handles.T1);
682     addpoints(handles.h2,handles.Zeit,handles.T2);
683     addpoints(handles.h3,handles.Zeit,handles.T1_SOLL);
684     addpoints(handles.h4,handles.Zeit,handles.TEC_CURRENT);
685     % wrap x-axis every 30 seconds
686     XL = xlim(handles.Ausgabeplot);
687     if handles.Zeit > XL(2)
688         xlim(handles.Ausgabeplot,[XL(1)+30 XL(2)+30])
689     end
690     drawnow limitrate
691 
692     % if no execption occurs change status to Okay
693     set(handles.Status_text,'String','Status: Daten wurden erfolgreich empfangen');
694 catch
695     % if execption occurs change status to Fehler
696     set(handles.Status_text,'String','Status: Fehler beim Empfangen der Daten');
697     % because errors mainly occur shortly after establishing the connection,
698     % it makes sense, to clear the plot after a faulty reading. Otherwise
699     % wrong values will be dsiplayed.
700     clearpoints(handles.h1);
701     clearpoints(handles.h2);
702     clearpoints(handles.h3);
703     clearpoints(handles.h4);
704 end
705 
706 % Update handles structure
707 guidata(hObject, handles);
708 
709 
710 % --- Updates static text boxes and edit boxes with current handle values
711 function outputDataToWidgets(handles)
712 switch handles.Betriebsmodus
713     case 0
714         set(handles.Modus_text,'String','Wachs');
715         set(handles.Heizen_radiobutton,'Value',1);
716         set(handles.Kuehlen_radiobutton,'Value',0);
717     case 1
718         set(handles.Modus_text,'String','Wasser');
```

```
719         set(handles.Heizen_radiobutton,'Value',0);
720         set(handles.Kuehlen_radiobutton,'Value',1);
721 end
722 switch handles.Zustand
723     case 0
724         set(handles.Zustand_text,'String','Bereit');
725     case 1
726         set(handles.Zustand_text,'String','Spannen');
727     case 2
728         set(handles.Zustand_text,'String','Lösen');
729     case 3
730         set(handles.Zustand_text,'String','Gespannt');
731 end
732 set(handles.Zeit_text,'String',sprintf('%0.1f',handles.Zeit));
733 set(handles.T1_text,'String',sprintf('%0.1f',handles.T1));
734 set(handles.T2_text,'String',sprintf('%0.1f',handles.T2));
735 set(handles.TEC_CURRENT_text,'String',sprintf('%0.1f',handles.TEC_CURRENT));
736 set(handles.T1_SOLL_text,'String',sprintf('%0.1f',handles.T1_SOLL));
737 set(handles.KP_HEIZEN_edit,'String',sprintf('%0.2f',handles.KP_HEIZEN));
738 set(handles.KI_HEIZEN_edit,'String',sprintf('%0.2f',handles.KI_HEIZEN));
739 set(handles.KP_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KP_KUEHLEN));
740 set(handles.KI_KUEHLEN_edit,'String',sprintf('%0.2f',handles.KI_KUEHLEN));
741 set(handles.T_WACHS_SCHMELZ_edit,'String',sprintf('%0.1f',handles.T_WACHS_SCHMELZ));
742 set(handles.T_WACHS_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WACHS_ERSTARR));
743 set(handles.T_WACHS_ABKUEHL_edit,'String',sprintf('%0.1f',handles.T_WACHS_ABKUEHL));
744 set(handles.T_WASSER_FRIER_edit,'String',sprintf('%0.1f',handles.T_WASSER_FRIER));
745 set(handles.T_WASSER_ERSTARR_edit,'String',sprintf('%0.1f',handles.T_WASSER_ERSTARR));
746 set(handles.DAUER_SCHMELZEN_edit,'String',handles.DAUER_SCHMELZEN);
747 set(handles.DAUER_ABKUEHLEN_edit,'String',handles.DAUER_ABKUEHLEN);
```

Listing A.6: Skript zur Auflistung aller verfügbaren virtuellen COM-Ports. [28]

```
1  function lCOM_Port = getAvailableComPort()
2  % function lCOM_Port = getAvailableComPort()
3  % Return a Cell Array of COM port names available on your computer
4
5  try
6      s=serial('IMPOSSIBLE_NAME_ON_PORT');fopen(s);
7  catch
8      lErrMsg = lasterr;
9  end
10
11 %Start of the COM available port
12 lIndex1 = findstr(lErrMsg,'COM');
13 %End of COM available port
14 lIndex2 = findstr(lErrMsg,'Use')-3;
15
16 lComStr = lErrMsg(lIndex1:lIndex2);
17
18 %Parse the resulting string
19 lIndexDot = findstr(lComStr,',');
20
21 % If no Port are available
```

```
22 if isempty(lIndex1)
23     lCOM_Port{1}='';
24     return;
25 end
26 
27 % If only one Port is available
28 if isempty(lIndexDot)
29     lCOM_Port{1}=lComStr;
30     return;
31 end
32 
33 lCOM_Port{1} = lComStr(1:lIndexDot(1)-1);
34 
35 for i=1:numel(lIndexDot)+1
36     % First One
37     if (i==1)
38         lCOM_Port{1,1} = lComStr(1:lIndexDot(i)-1);
39     % Last One
40     elseif (i==numel(lIndexDot)+1)
41         lCOM_Port{i,1} = lComStr(lIndexDot(i-1)+2:end);
42     % Others
43     else
44         lCOM_Port{i,1} = lComStr(lIndexDot(i-1)+2:lIndexDot(i)-1);
45     end
46 end
```

A.6 Sourcecode der Mikrocontroller-Firmware

In den beiden nachfolgenden Listings ist die Mikrocontroller-Firmware, die die Steuerlogik des Spannsystems sowie die Temperaturregelung enthält, dargestellt. Auf sie wird in Abschnitt 5.4 umfassend eingegangen. Listing A.8 enthält die Implementierung des digitalen Temperaturreglers. Details hierzu werden in Abschnitt 4.3 beschrieben.

Listing A.7: Hauptfunktionen der Firmware des Mikrocontrollers.

```
1  /*********************************************************************
2  *  Steuerungssoftware für thermisches Einspann-System
3  *  Lukas Bommes 2016
4  *********************************************************************/
5  
6  #include Regler.h
7  
8  // Konstanten
9  double T_WACHS_NULL = -15; //°C
10 double T_WASSER_NULL = 75; //°C
11 
12 // Einstellwerte
13 double T_WACHS_SCHMELZ = 70.0; //°C
14 double T_WACHS_ERSTARR = 60.0; //°C
15 double T_WACHS_ABKUEHL = -15.0; //°C
```

```
16 double T_WASSER_FRIER = -15.0; //°C
17 double T_WASSER_ERSTARR = 0.0; //°C
18 unsigned long DAUER_SCHMELZEN = 8000; //ms
19 unsigned long DAUER_ABKUEHLEN = 15000; //ms
20 double KP_HEIZEN = 63.89;
21 double KI_HEIZEN = 7.80;
22 double KP_KUEHLEN = -13.54;
23 double KI_KUEHLEN = -3.49;
24
25 // Hilsvariablen
26 unsigned long timestamp = 0; // Hilfszeit für Zeitmessung
27 int zeit_gemerkt = 0; // Hilfsvariable für Zeitmessung
28 String send_string = String(); // String für serielle Ausgabe
29 static volatile uint8_t isr_active_flag = 0; // ISR_Flag für das Senden der Daten
30
31 // Betriebsvariablen
32 int Betriebsmodus = 1; // 0 = 'WACHS'; 1 = 'WASSER'
33 int Zustand = 0; // 0 = 'BEREIT'; 1 = 'SPANNEN'; 2 = 'LOESEN'; 3 = 'GESPANNT'
34 int Aktion = 0; // 0 = ''; 1 = 'AKTION_SPANNEN', 2 = 'AKTION_LOESEN', 3 = 'AKTION_ABBRECHEN'
35 double T1 = 0.0; // Temperatur der Spannplatte (-15.0 °C bis 75°C)
36 double T2 = 0.0; // Temperatur des Wärmetauschers (-15.0 °C bis 75°C)
37 double TEC_CURRENT; // Tastgrad des PWM-Signals der Treiberschaltung (0 bis 255)
38 double T1_SOLL; // globale Variable für den Sollwert der Temperatur (-15.0 °C bis 75°C)
39
40 // Regler
41 double Sample_Zeit = 100000; // Sample-Zeit in us (16 us <= Sample_Zeit < 1048592 us, ansonsten
    Prescaler von Timer1 aendern)
42 PID_Regler Regler(KP_HEIZEN,KI_HEIZEN,0,0,Sample_Zeit,&T1,&TEC_CURRENT); // Regler mit Kp, Ki,
   Kd, N, 100000 us Sample-Zeit und Eingang T1 sowie Ausgang TEC_CURRENT
43
44 void setup()
45 {
46   Serial.begin(115200);
47   Serial.setTimeout(10);
48
49    // H-Brücke auf Kühlbetrieb setzen
50   pinMode(10,OUTPUT);
51   digitalWrite(10,LOW);
52
53   // Regler starten
54   T1 = adcToDegreeCelsius(analogRead(1));
55   Regler.PID_SetzeModus(AN);
56
57   // Timer1 entsprechend der Sample-Zeit einstellen
58   cli(); // Interrupts anhalten
59   TCCR1A = 0;
60   TCCR1B = 0;
61   TCNT1  = 0; // Counter mit 0 initialisieren
62   OCR1A = ((16.0/256.0)*Sample_Zeit)-1.0; // (muss < 65536 sein) // Compare Match Register
     setzen
63   TCCR1B |= (1 << WGM12); // CTC-Modus
64   TCCR1B |= (1 << CS12); // Prescaler = 256
65   TIMSK1 |= (1 << OCIE1A); // Timer Compare Interrupt aktivieren
66   sei(); // Interrupts erlauben
```

```
67 }
68
69 // ISR-Routine
70 ISR(TIMER1_COMPA_vect)
71 {
72   // Flag setzen führt zur Ausführung des entsprechenden Codes in der main-loop
73   isr_active_flag = 1;
74 }
75
76 // Daten an serielle Schnittstelle senden
77 void serialSendData()
78 {
79   // Variablen zu einem String zusammenführen
80   send_string += Betriebsmodus;
81   send_string += ,;
82   send_string += Zustand;
83   send_string += ,;
84   send_string += millis(); // Zeit
85   send_string += ,;
86   send_string += T1_SOLL;
87   send_string += ,;
88   send_string += T1;
89   send_string += ,;
90   send_string += T2;
91   send_string += ,;
92   send_string += adcToPercent(TEC_CURRENT); // Ausgabe in Prozent
93   send_string += ,;
94   send_string += KP_HEIZEN;
95   send_string += ,;
96   send_string += KI_HEIZEN;
97   send_string += ,;
98   send_string += KP_KUEHLEN;
99   send_string += ,;
100  send_string += KI_KUEHLEN;
101  send_string += ,;
102  send_string += T_WACHS_SCHMELZ;
103  send_string += ,;
104  send_string += T_WACHS_ERSTARR;
105  send_string += ,;
106  send_string += T_WACHS_ABKUEHL;
107  send_string += ,;
108  send_string += T_WASSER_FRIER;
109  send_string += ,;
110  send_string += T_WASSER_ERSTARR;
111  send_string += ,;
112  send_string += DAUER_SCHMELZEN;
113  send_string += ,;
114  send_string += DAUER_ABKUEHLEN;
115
116  // String an die serielle Schnittstelle senden
117  Serial.println(send_string);
118
119  // String löschen
120  send_string = ;
```

```
121 }
122
123 // Funktion rechnet ADC-Werte (0..1023) in Grad Celsius um
124 double adcToDegreeCelsius(int adc_value)
125 {
126   // Ausgleichsgerade mit -15 °C..75 °C und 0..1023
127   return 0.087977 * adc_value - 15.0;
128 }
129
130 // Funktion rechnet Grad Celsius in ADC-Werte (0..1023) um
131 int degreeCelsiusToAdc(double degree_celsius_value)
132 {
133   // Ausgleichsgerade umgeformt, Ausgabe ganze Zahlen
134   return round((degree_celsius_value + 15.0)/0.087977);
135 }
136
137 // Funktion rechnet diskrete Werte (0..255) in Prozent (1..100) um
138 double adcToPercent(int adc_value)
139 {
140   return adc_value/255.0*100.0;
141 }
142
143 void loop() {
144
145   // zur Erkennung, ob Betriebsmodus zwischen den
146   // Schleifendurchläufen geändert wurde
147   int Betriebsmodus_alt = Betriebsmodus;
148
149   // Daten als CSV-String vom seriellen Port lesen, Integer-Werte zwischen den Kommata
150   // entnehmen und in die Variablen schreiben
151   while (Serial.available() > 0)
152   {
153     // Werte vom Port empfangen und in Variablen schreiben
154     Betriebsmodus = Serial.parseInt();
155     Aktion = Serial.parseInt();
156     KP_HEIZEN = Serial.parseFloat();
157     KI_HEIZEN = Serial.parseFloat();
158     KP_KUEHLEN = Serial.parseFloat();
159     KI_KUEHLEN = Serial.parseFloat();
160     T_WACHS_SCHMELZ = Serial.parseFloat();
161     T_WACHS_ERSTARR = Serial.parseFloat();
162     T_WACHS_ABKUEHL = Serial.parseFloat();
163     T_WASSER_FRIER = Serial.parseFloat();
164     T_WASSER_ERSTARR = Serial.parseFloat();
165     DAUER_SCHMELZEN = Serial.parseInt();
166     DAUER_ABKUEHLEN = Serial.parseInt();
167
168     if (Serial.read() == '\n'); // Warten, bis gesamte Zeile gelesen wurde
169   }
170
171   // erfüllt, wenn die ISR aufgerufen wurde
172   if(isr_active_flag)
173   {
174     isr_active_flag = 0;
```

```
175
176   // Temperaturen einlesen
177   T1 = adcToDegreeCelsius(analogRead(1)); // Spannplattentemperatur
178   T2 = adcToDegreeCelsius(analogRead(0)); // Wärmetauschertemperatur
179
180   // Reglerfunktionen
181   Regler.PID_Calc(T1_SOLL); // Sollwert des Reglers als ADC-Wert (0..1023) vorgeben
182   analogWrite(3,TEC_CURRENT); // Stellgröße auf Pin 3 ausgeben
183
184   // Senden der Daten an die serielle Schnittstelle
185   serialSendData();
186 }
187
188 // Wenn zwischen den beiden Betriebsmodi umgeschaltet wird, dafür sorgen,
189 // dass System in Zustand = 'BEREIT' geht
190 if(Betriebsmodus != Betriebsmodus_alt)
191 {
192   Aktion = 0;
193   Zustand = 0;
194 }
195
196 // Diese Verzweigung schaltet entsprechend der drei Variablen (Betriebsmodus,
197 // Zustand und Aktion) durch die einzelnen Unterfunktionen
198 if(Betriebsmodus == 0) // Betriebsmodus = 'WACHS'
199 {
200   switch(Zustand)
201   {
202     case 0: // Zustand = 'BEREIT'
203     {
204       digitalWrite(10,HIGH); // H-Brücke Richtung setzen
205       Regler.PID_SetzeParameter(KP_HEIZEN,KI_HEIZEN,0.0,0.0); // Reglerparameter festlegen
206       T1_SOLL = T_WACHS_NULL;
207       switch(Aktion)
208       {
209         case 0: // Aktion = ''
210         {
211           // keine Aktion
212           break;
213         }
214         case 1: // Aktion = 'AKTION_SPANNEN'
215         {
216           // Spannfunktion auslösen
217           Aktion = 0; // Aktion = ''
218           Zustand = 1; // Zustand = 'Spannen'
219           T1_SOLL = T_WACHS_SCHMELZ;
220           break;
221         }
222         case 2: // Aktion = 'AKTION_LOESEN'
223         {
224           Aktion = 0; // Aktion = ''
225           break;
226         }
227         case 3: // Aktion = 'AKTION_ABBRECHEN'
228         {
```

```
229             Aktion = 0; // Aktion = ''
230             break;
231           }
232         }
233         break;
234       }
235       case 1: // Zustand = 'SPANNEN'
236       {
237         // Wachs aufschmelzen und erstarren, Funktionsdefinition s. unten
238         FahreWachszyklus();
239         switch(Aktion)
240         {
241           case 0: // Aktion = ''
242           {
243             // keine Aktion
244             break;
245           }
246           case 1: // Aktion = 'AKTION_SPANNEN'
247           {
248             Aktion = 0; // Aktion = ''
249             break;
250           }
251           case 2: // Aktion = 'AKTION_LOESEN'
252           {
253             Aktion = 0; // Aktion = ''
254             break;
255           }
256           case 3: // Aktion = 'AKTION_ABBRECHEN'
257           {
258             // Abbrechen und dann Zustand = 'BEREIT' setzen
259             Aktion = 0; // Aktion = ''
260             T1_SOLL = T_WACHS_NULL; // Sollwert auf Nullwert setzen und damit TEC ausschalten
261             Zustand = 0; // Zustand = 'BEREIT'
262             zeit_gemerkt = 0; // Zeitmessung muss nach Klick auf Abbrechen wieder möglich sein
263             break;
264           }
265         }
266         break;
267       }
268       case 2: // Zustand = 'LOESEN'
269       {
270         // Wachs aufschmelzen und erstarren, Funktionsdefinition s. unten
271         FahreWachszyklus();
272         switch(Aktion)
273         {
274           case 0: // Aktion = ''
275           {
276             // keine Aktion
277             break;
278           }
279           case 1: // Aktion = 'AKTION_SPANNEN'
280           {
281             // keine Aktion
282             break;
```

```
283           }
284           case 2: // Aktion = 'AKTION_LOESEN'
285           {
286             // keine Aktion
287             break;
288           }
289           case 3: // Aktion = 'AKTION_ABBRECHEN'
290           {
291             // Abbrechen und dann Zustand = 'BEREIT' setzen
292             Aktion = 0; // Aktion = ''
293             T1_SOLL = T_WACHS_NULL; // Sollwert auf Nullwert setzen und damit TEC ausschalten
294             Zustand = 0; // Zustand = 'BEREIT'
295             zeit_gemerkt = 0; // Zeitmessung muss nach Klick auf Abbrechen wieder möglich sein
296             break;
297           }
298         }
299         break;
300       }
301       case 3: // Zustand = 'GESPANNT'
302       {
303         digitalWrite(10,HIGH); // H-Brücke Richtung setzen
304         Regler.PID_SetzeParameter(KP_HEIZEN,KI_HEIZEN,0.0,0.0); // Reglerparameter festlegen
305         switch(Aktion)
306         {
307           case 0: // Aktion = ''
308           {
309             // keine Aktion
310             break;
311           }
312           case 1: // Aktion = 'AKTION_SPANNEN'
313           {
314             Aktion = 0; // Aktion = ''
315             break;
316           }
317           case 2: // Aktion = 'AKTION_LOESEN'
318           {
319             // Lösefunktion auslösen
320             Aktion = 0; // Aktion = ''
321             Zustand = 2; // Zustand = 'LOESEN'
322             T1_SOLL = T_WACHS_SCHMELZ;
323             break;
324           }
325           case 3: // Aktion = 'AKTION_ABBRECHEN'
326           {
327             Aktion = 0; // Aktion = ''
328             break;
329           }
330         }
331         break;
332       }
333     }
334   }
335
336   else // Betriebsmodus = 'WASSER'
```

```
337  {
338    digitalWrite(10,LOW); // H-Brücke Richtung setzen
339    Regler.PID_SetzeParameter(KP_KUEHLEN,KI_KUEHLEN,0.0,0.0); // Reglerparameter festlegen
340    switch(Zustand)
341    {
342      case 0: // Zustand = 'BEREIT'
343      {
344        T1_SOLL = T_WASSER_NULL;
345        switch(Aktion)
346        {
347          case 0: // Aktion = ''
348          {
349            // keine Aktion
350            break;
351          }
352          case 1: // Aktion = 'AKTION_SPANNEN'
353          {
354            // Spannfunktion auslösen
355            Aktion = 0; // Aktion = ''
356            Zustand = 1; // Zustand = 'SPANNEN'
357            T1_SOLL = T_WASSER_FRIER;
358            break;
359          }
360          case 2: // Aktion = 'AKTION_LOESEN'
361          {
362            Aktion = 0; // Aktion = ''
363            break;
364          }
365          case 3: // Aktion = 'AKTION_ABBRECHEN'
366          {
367            Aktion = 0; // Aktion = ''
368            break;
369          }
370        }
371        break;
372      }
373      case 1: // Zustand = 'SPANNEN'
374      {
375        // prüfen, ob Erstarrungstemperatur erreicht wurde und ggf. Zustand in gespannt ändern
376        if(T1 < T_WASSER_ERSTARR)
377        {
378          Zustand = 3; // Zustand = 'GESPANNT'
379        }
380        switch(Aktion)
381        {
382          case 0: // Aktion = ''
383          {
384            // keine Aktion
385            break;
386          }
387          case 1: // Aktion = 'AKTION_SPANNEN'
388          {
389            Aktion = 0; // Aktion = ''
390            break;
```

```
391           }
392           case 2: // Aktion = 'AKTION_LOESEN'
393           {
394             Aktion = 0; // Aktion = ''
395             break;
396           }
397           case 3: // Aktion = 'AKTION_ABBRECHEN'
398           {
399             // Abbrechen und wenn T1 > T_WASSER_ERSTARR dann Zustand = 'BEREIT'
400             Aktion = 0; // Aktion = ''
401             T1_SOLL = T_WASSER_NULL; // Sollwert auf Nullwert setzen und damit TEC ausschalten
402             if(T1 > T_WASSER_ERSTARR)
403             {
404               Zustand = 0; // Zustand = 'BEREIT'
405             }
406             break;
407           }
408         }
409         break;
410       }
411       case 2: // Zustand = 'LOESEN'
412       {
413         // prüfen, ob Erstarrungtemperatur überschritten wurde und ggf. Zustand ändern
414         if(T1 > T_WASSER_ERSTARR)
415         {
416           Zustand = 0; // Zustand = 'BEREIT'
417         }
418         switch(Aktion)
419         {
420           case 0: // Aktion = ''
421           {
422             // keine Aktion
423             break;
424           }
425           case 1: // Aktion = 'AKTION_SPANNEN'
426           {
427             Aktion = 0; // Aktion = ''
428             break;
429           }
430           case 2: // Aktion = 'AKTION_LOESEN'
431           {
432             Aktion = 0; // Aktion = ''
433             break;
434           }
435           case 3: // Aktion = 'AKTION_ABBRECHEN'
436           {
437             Aktion = 0; // Aktion = ''
438             break;
439           }
440         }
441         break;
442       }
443       case 3: // Zustand = 'GESPANNT'
444       {
```

```
445         switch(Aktion)
446         {
447           case 0: // Aktion = ''
448           {
449             // keine Aktion
450             break;
451           }
452           case 1: // Aktion = 'AKTION_SPANNEN'
453           {
454             Aktion = 0; // Aktion = ''
455             break;
456           }
457           case 2: // Aktion = 'AKTION_LOESEN'
458           {
459             // Lösefunktion auslösen
460             Aktion = 0; // Aktion = ''
461             Zustand = 2;// Zustand = 'LOESEN'
462             T1_SOLL = T_WASSER_NULL;
463             break;
464           }
465           case 3: // Aktion = 'AKTION_ABBRECHEN'
466           {
467             Aktion = 0; // Aktion = ''
468             break;
469           }
470         }
471         break;
472       }
473     }
474   }
475 }
476
477 // Funktion für das Schmelzen und Erstarren des Wachses
478 void FahreWachszyklus()
479 {
480   // nachdem T_WACHS_ERSTARR überschritten wurde und gerade geheizt wird
481   if(T1_SOLL == T_WACHS_SCHMELZ && T1 >= T_WACHS_ERSTARR)
482   {
483     // aktuelle Zeit genau einmal merken
484     if(zeit_gemerkt == 0)
485     {
486       timestamp = millis();
487       zeit_gemerkt = 1; // blockt Zeitmessung für alle weiteren Durchläufe der Schleife
488     }
489     // bestimmte Zeit lang weiter heizen, anschließend kühlen starten
490     if(millis() - timestamp >= DAUER_SCHMELZEN) // nach DAUER_SCHMELZEN Sekunden
491     {
492       // Regler in Kühlbetrieb
493       digitalWrite(10,LOW);
494       Regler.PID_SetzeParameter(KP_KUEHLEN,KI_KUEHLEN,0.0,0.0);
495       T1_SOLL = T_WACHS_ABKUEHL;
496       zeit_gemerkt = 0; // Zeitmessung im nächsten Schgleifendurchlauf ermöglichen
497     }
498   }
```

```
499   // nachdem T_WACHS_ERSTARR wieder unterschritten wurde, nachdem zuvor geheizt wurde
500   if(T1_SOLL == T_WACHS_ABKUEHL && T1 < T_WACHS_ERSTARR)
501   {
502     // aktuelle Zeit genau einmal merken
503     if(zeit_gemerkt == 0)
504     {
505       timestamp = millis();
506       zeit_gemerkt = 1; // blockt Zeitmessung für alle weiteren Durchläufe der Schleife
507     }
508     // aktiv abkühlen für bestimmte zeit und Zustand in gespannt ändern
509     if(millis() - timestamp >= DAUER_ABKUEHLEN) // DAUER_ABKUEHLEN Sekunden später
510     {
511       // Regler in Heizbetrieb
512       digitalWrite(10,HIGH);
513       Regler.PID_SetzeParameter(KP_HEIZEN,KI_HEIZEN,0.0,0.0);
514       if(Zustand == 1) // Zustand == 'spannen'
515       {
516         Zustand = 3; // Zustand = 'GESPANNT'
517       }
518       else // Zustand == 'lösen'
519       {
520         Zustand = 0; // Zustand = 'BEREIT'
521       }
522       zeit_gemerkt = 0; // erneute Zeitmessung im nächsten Durchlauf ermöglichen
523     }
524   }
525 }
```

Listing A.8: Implementierung eines digitalen PID-Reglers als Klasse in C++.

```
1  /*********************************************************************
2  *  Klasse für PID-Regler
3  *  Lukas Bommes 2016
4  *********************************************************************/
5
6  #include Arduino.h
7
8  #define AN 1
9  #define AUS 0
10
11 #define OUTPUT_MAX_VALUE 255 // maximaler Ausgangswert des Reglers
12 #define OUTPUT_MIN_VALUE 0 // minimaler Ausgangswert des Reglers
13
14 class PID_Regler {
15
16   private:
17     double Regelwert;
18     double Regelabweichung;
19     double Stellwert;
20     double Letzter_Regelwert;
21     double Regelabweichung_Summe;
22     double Filter_Summe;
23     double Kp, Ki, Kd, N;
```

```
24      unsigned long Sample_Zeit;
25      double Sample_Zeit_in_Sekunden;
26      double* Input;
27      double* Output;
28      int Modus;
29
30    public:
31      PID_Regler(double Kp, double Ki, double Kd, double N, unsigned long Sample_Zeit, double*
        Input, double* Output);
32      void PID_SetzeModus(int Modus);
33      void PID_SetzeParameter(double Kp, double Ki, double Kd, double N);
34      void PID_Init();
35      void PID_Calc(double Sollwert);
36  };
37
38  /*************************************************************************
39  *  Der Konstruktor erstellt ein PID_Regler-Objekt. An ihn müssen die
40  *  Reglerparameter Kp, Ki, Kd und N sowie die Sample-Zeit in Mikrosekunden
41  *  und zwei Zeiger auf double-Variablen, die als Eingang und Ausgang
42  *  des Reglers dienen, übergeben werden. Der Wertebereich von Output
43  *  liegt dabei zwischen OUTPUT_MIN_VALUE und OUTPUT_MAX_VALUE und eignet
44  *  sich daher zur direkten Ansteuerung eines PWM-Signals.
45  *************************************************************************/
46
47  PID_Regler::PID_Regler(double Kp, double Ki, double Kd, double N, unsigned long Sample_Zeit,
    double* Input, double* Output) {
48    this->Kp = Kp;
49    this->Ki = Ki;
50    this->Kd = Kd;
51    this->N = N;
52    this->Sample_Zeit = Sample_Zeit;
53    this->Sample_Zeit_in_Sekunden = ((double)Sample_Zeit)/1000000;
54    this->Input = Input;
55    this->Output = Output;
56    this->Modus = AUS;
57  }
58
59  void PID_Regler::PID_SetzeModus(int Modus) {
60    bool neuerModus = (Modus == AN);
61    if(neuerModus == !(this->Modus)) {
62      PID_Regler::PID_Init();
63    }
64    this->Modus = neuerModus;
65  }
66
67  /*************************************************************************
68  *  Die Initialisierungsmethode wird i.d.R. im Setup() aufgerufen und
69  *  dient dem erstmaligen Setzten des Integralterms und des letzten
70  *  Regelwerts.
71  *************************************************************************/
72
73  void PID_Regler::PID_Init() {
74    // Werte initialisieren
75    this->Letzter_Regelwert = *(this->Input);
```

```
76   this->Regelabweichung_Summe = *(this->Output);
77   this->Filter_Summe = 0.0;
78   if(this->Regelabweichung_Summe > OUTPUT_MAX_VALUE) this->Regelabweichung_Summe =
     OUTPUT_MAX_VALUE;
79   else if(this->Regelabweichung_Summe > OUTPUT_MIN_VALUE) this->Regelabweichung_Summe =
     OUTPUT_MIN_VALUE;
80 }
81
82 /*************************************************************************
83 *  Die Kalkulationsroutine muss in festen Zeitintervallen (Sample_Zeit)
84 *  aufgerufen werden. Vor Ausführung der Kalkulationsmethde muss die
85 *  Input-Variable neu eingelesen und anschließend der Ouput, z.B.
86 *  mittels analogWrite(), geschrieben werden. Als Argument wird an die
87 *  Kalkulationsmethode der einzuhaltende Sollwert für die Regelgröße
88 *  übergeben.
89 *************************************************************************/
90
91 void PID_Regler::PID_Calc(double Sollwert) {
92
93   // Berechnung nur durchführen, wenn der Regler akitiviert ist
94     if(Modus == AN) {
95
96       // Regelgröße einlesen und Regelabweichung berechnen
97       this->Regelwert = *(this->Input);
98       this->Regelabweichung = Sollwert - this->Regelwert;
99
100      // Integralterm berechnen
101      this->Regelabweichung_Summe += Ki * this->Regelabweichung * this->Sample_Zeit_in_Sekunden
         ;
102
103      // Integralterm begrenzen
104      if(this->Regelabweichung_Summe > OUTPUT_MAX_VALUE) this->Regelabweichung_Summe =
         OUTPUT_MAX_VALUE;
105      else if(this->Regelabweichung_Summe < OUTPUT_MIN_VALUE) this->Regelabweichung_Summe =
         OUTPUT_MIN_VALUE;
106
107      // Filterkoeffizient und Filterintegral berechnen
108      double FilterKoeffizient = (Kd * (this->Letzter_Regelwert - this->Regelwert) - this->
         Filter_Summe) * this->N;
109      this->Filter_Summe += this->Sample_Zeit_in_Sekunden * FilterKoeffizient;
110
111      // Stellgröße berechnen
112      this->Stellwert = Kp * this->Regelabweichung + this->Regelabweichung_Summe +
         FilterKoeffizient;
113
114      // Stellgröße begrenzen
115      if(this->Stellwert > OUTPUT_MAX_VALUE) this->Stellwert = OUTPUT_MAX_VALUE;
116      else if(this->Stellwert < OUTPUT_MIN_VALUE) this->Stellwert = OUTPUT_MIN_VALUE;
117
118      // Stellgröße auf Ausgang geben
119      *(this->Output) = this->Stellwert;
120
121      // Regelgröße für Berechnung im nächsten Funktionsaufruf speichern
122      this->Letzter_Regelwert = this->Regelwert;
```

```
123   }
124 }
125
126 /*************************************************************************
127 *  Methode zum Ändern der PID-Parameter
128 *************************************************************************/
129
130 void PID_Regler::PID_SetzeParameter(double Kp, double Ki, double Kd, double N) {
131    this->Kp = Kp;
132    this->Ki = Ki;
133    this->Kd = Kd;
134    this->N = N;
135 }
```

Eidesstattliche Erklärung

Hiermit erkläre ich, Lukas Bommes, die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der angegebenen Mittel und Quellen erstellt zu haben.

Braunschweig, 06. Juli 2016

(Lukas Bommes)