Bachelorarbeit

Erweiterung des TDDS Temperaturreglers

ausgeführt am

Institut für Mikroelektronik

zum Zwecke der Erlangung des akademischen Grades eines

Bachelor of Science (BSc)

unter der Leitung von

A.o. Univ.-Prof. Dipl.-Ing. Dr.techn. Tibor Grasser

und

Dipl.-Ing. Dr.techn. Michael Waltl

ausgeführt durch

Christian Schleich

Matrikelnummer 1325958
Linzer Straße 160/8/15
1140 Wien

Wien, im Jänner 2017

Christian Schleich

Diese Arbeit wurde in Kooperation mit Hr. Andreas Bauer, mb-Technologies, durchgeführt, welcher den Temperaturkontroller (Hardware und Firmware) zur Verfügung gestellt hat.

Kurzfassung

Die Leistungsfähigkeit moderner Metall-Oxid-Halbleiter Transistoren wird von Störstellen in der Kristallstruktur, sogenannten Defekten, wesentlich beeinflusst. Um solche elektrisch aktiven Defekte charakterisieren zu können, wurde die zeitabhängige Defektspektroskopie (engl. Time-Dependent Defect Spectroscopy, TDDS) entwickelt. Dabei werden einzelne Defekte durch Anlegen einer Stressspannung gezielt geladen und ihr Entladeverhalten, nämlich die Entladedauer, sowie der Einfluss des Ladungszustandes des Defektes auf den Drain-Source Strom, genauer analysiert. Die Lade- und Entladezeiten der Defekte hängen sehr stark von der Temperatur ab, weshalb eine exakte Regelung der Transistortemperatur notwendig ist. Die Temperatur soll dabei möglichst schnell auf einen exakten Wert geregelt werden und anschließend für mehrere Monate stabil geregelt werden. Um dies zu gewährleisten, wurde in dieser Arbeit ein PID Regler in einem Mikrokontroller implementiert und auf verschiedene Heizelemente optimiert. Durch die Entwicklung eines mathematischen Modells für den Regler können nun die Regler sehr effizient auf die jeweilige Regelstrecke angepasst werden. Um die exakten Regelparameter zu bestimmen wurde dazu ein Tool in der Programmiersprache Python entwickelt. Dies hat den großen Vorteil, dass somit auch ein nichtlineares Verhalten der Regelstrecke bereits während der Simulation des Regelprozesses berücksichtigt werden kann. In weiterer Folge wurde der Regler an einfachen kompakten Heizelementen, an Wolframheizspulen in institutsinternen Messöfen, sowie an verschiedenen Peltierelementen getestet und ausgewertet. Da im Gegensatz zum Heizvorgang der Abkühlvorgang in den meisten Fällen nur sehr langsam durch freie Konvektion der Wärme an die Umgebung statt findet, ist der Einsatz von Peltierelementen besonders vorteilhaft. Diese ermöglichen auch einen gezielten Abkühlvorgang und werden im Rahmen der Arbeit abschließend evaluiert.

Abstract

The performance of modern metal-oxide-semiconductor transistors is seriously affected by imperfections in the atomic structure of the device, called defects. In order to characterize such electrically active defects, the Time-Dependent Defect Spectroscopy (TDDS) has been proposed. Using TDDS, single defects are deliberately charged by applying a stress bias at the device, and afterwards their recovery behavior, i.e. their emission times and their impact on the drain-source current, thoroughly analyzed. Furthermore, as the charge capture and charge emission times are strongly sensitive to the device temperature a precise control of the device temperature is inevitable. Therefore the temperature should on the one hand reach the target temperature as fast as possible, and on the other hand the temperature has to be controlled to be stable over several month. To fulfil these requirements, a PID controller has been implemented in a microcontroller and optimized for different heating elements. By developing a mathematical model for the controller its parameters can be adjusted to be suitable for different systems very efficiently. To further determine the exact parameters for the controller a simulation tool has been implemented in python. This tool offers the big advantage that a possible non-linear behaviour of the system can already be considered during simulation of the controlling process. In the following the

implementation was verified utilizing compact heating elements, tungsten based heating coils embedded in the Institute's furnaces, and various Peltier elements. In contrast to the fast heating process, the cooling down process is very slow as no active cooling is typically available. However, by using Peltier elements a defined cooling temperature profile can be applied, which is finally evaluated in this work.

Inhaltsverzeichnis

1 Einführung

Die vorliegende Arbeit ist in einen theoretischen und einen praktischen Teil unterteilt. Beginnend mit einer Vorstellung des Bauelementes MOSFET werden anschließend Degradiationsmechanismen, welche sich nachteilig auf die Leistungsfähigkeit der Transistoren auswirken, präsentiert. Dafür verantwortlich sind Defekte in der idealen Atomstruktur, welche sich im Oxid (sogenannte oxide traps) oder an der Si/SiO_2 Grenzschicht (sogenannte interface states) befinden. Anschließend wird die sogenannte Time-Dependent Defect Spectroscopy (TDDS) erläutert, welche speziell zur Charakterisierung einzelner Defekte am Institut für Mikroelektronik der TU Wien entwickelt wurde. Dem folgt eine kurze Einführung und Begriffsbildung der Regelungstechnik und Erklärung der Eigenschaften eines PID - Reglers.

Der praktische Teil der Arbeit befasst sich zuerst mit der Beschreibung der vorliegenden Hardware, welche den Temperaturkontroller und die Heizaufbauten umfasst. Danach wird die Implementierung der Software im Mikrokontroller vorgestellt. Zusätzlich wird noch auf die Implementierung der zusätzlich notwendigen Kommunikations- und Berechnungsmethoden für den Regler in Python eingegangen. Um den Anforderungen einer möglichst schnellen Regelung auf eine Zieltemperatur gerecht zu werden, wird ein optimierter Regelalgorithmus, beginnend mit der Modellierung und experimentellen Bestimmung der zu regelnden Strecke, präsentiert. Zusätzlich besteht noch die Möglichkeit gezielte Temperaturrampen vorzugeben.

Abschließend werden die gemessenen Regelungsergebnisse diskutiert und ein Ausblick auf mögliche Verbesserungen des Software Reglers, sowie der Hardware durch den Einsatz neuer Heizelemente, gegeben.

2 Grundlagen

In diesem Abschnitt werden die theoretischen Grundlagen, mit denen sich die vorliegende Arbeit befasst, erläutert. Beginnend mit den Eigenschaften und dem Aufbau des sogenannten Metall-Oxid-Halbleiter Feldeffekttransistor (MOSFET), über Defekte, welche während des Betriebes geladen und entladen werden können, sowie deren Auswirkung auf die Lebensdauer des Bauteils, wird anschließend die Time-Dependent Defect Spectroscopy (TDDS) erklärt. Abschließend wird auf Grundlagen der Regelungstechnik eingegangen, und der PID - Regler im Speziellen behandelt.

2.1 MOSFET

Feldeffekttransistoren sind Bauteile mit denen der Strom zwischen den Anschlüssen **Source** und **Drain** durch eine am **Gate** - Kontakt anliegende Spannung moduliert werden kann. Diese Transistoren können, neben vielen anderen Vorteilen, wie einstellbarer Schwellenspannung, vor allem sehr gut miniaturisiert und integriert sowie komplementär aufgebaut werden, wodurch diese Technologie den Grundstein für moderne komplexe Logikschaltungen bildet.

Der Transistor ist, wie in Abbildung 1 schematisch gezeigt, aufgebaut. Hierbei handelt es sich um einen sogenannten n-Kanal MOSFET, welcher aus einem großen p - dotierten Bereich, dem Substrat unter dem Gateanschluss, sowie zwei kleineren n - dotierten Wannen unter dem Source - bzw. Drainanschluss aufgebaut ist. Source und Drainelektrode sind direkt mit dem Halbleiter verbunden, wohingegen die Gateelektrode durch eine wenige Nanometer dünne Gateoxidschicht von dem p - dotierten Substrat getrennt ist. Wichtig für die Funktion ist, dass die n-dotierten Gebiete unter die Gateelektrode reichen.

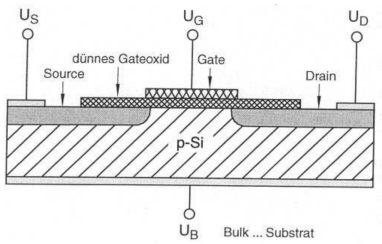


Abbildung 1: Schematischer Aufbei eines n-Kanal MOSFET aus [12]

Da nur eine Ladungsträgerart am Stromfluss beteiligt ist, werden solche Bauelemente auch als unipolare Transistoren bezeichnet. Je nach anliegender Gatespannung wird zwischen drei Betriebsbereichen unterschieden.

Akkumulation:
Hierbei werden durch Anlegen einer negativen Gatespannung Löcher hin zur SiO_2/Si Grenzfläche gezogen und dort angereichert. Es ist kein Stromfluss zwischen Source und Drain möglich, da die pn - Übergänge in Sperrrichtung gepolt sind.

Verarmung:
Umgekehrt werden die positiven Ladungsträger bei Anlegen einer positiven Gatespannung in den Substratkörper (Bulk) verdrängt. Die Majoritätsladungsträger verarmen an der Grenzfläche, wodurch sich schließlich eine Raumladungszone unter dem Gatekontakt ausbildet.

Inversion:
Bei einer weiteren Erhöhung der Gatespannung in Richtung positiver Werte, sammeln sich ab der sogenannten Schwellenspannung Elektronen an der SiO_2/Si Grenzfläche. Die Ansammlung von Elektronen an der Grenzschicht spiegelt sich im Verbiegen der Energiebandkanten an der Grenzfläche wieder (siehe Abbildung 2). Die Anzahl der Elektronen im sogenannten leitfähigen Kanal ändert sich proportional zur angelegten Gatespannung

$$| \triangle Q_{\mathrm{Ch}}| = C_{\mathrm{Ox}} | \triangle U_{\mathrm{GS}}| \tag{1}$$

mit C_{Ox}, der Kapazität der Oxidschicht.
Legt man nun eine Spannung U_{DS} zwischen Drain - und Source - Kontakt an, fließt zunächst ein Strom vom Drain - in das Source - Gebiet. Im ohmschen Betriebsbereich ist dieser Strom proportional zur angelegten Spannung U_{DS}. Jener Betriebsbereich wird als ohmscher Bereich bezeichnet. Wird U_{DS} weiter erhöht, verformt sich der bestehende Kanal unsymmetrisch bis er schließlich abgeschnürt wird und nicht mehr bis zum Draingebiet reicht. Die angelegte Spannung U_{DS} fällt nun fast vollständig im Abschnürbereich ab und es kommt zu keiner weiteren Stromerhöhung. Aus diesem Grund wird dieser Betriebsbereich auch als Stromquellenbereich bezeichnet [2].

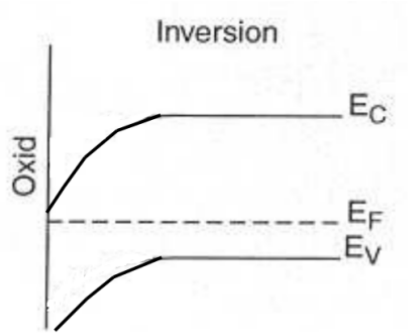


Abbildung 2: Bandschema des n-MOSFETs an der Oxidgrenzschicht [12]

2.2 Defekte und Bias Temperature Instabilities (BTI)

Bei der Transistorherstellung können beim Aufwachsen des Oxids strukturelle Defekte, sowohl direkt in der Oxidschicht, als auch im Interface zwischen Halbleiter und Isolator, entstehen. Letztere sind eine Ursache von unterschiedlichen Gitterkonstanten zwischen SiO_2 und dem Si - Substrat. Diese Defekte gelten als Ursache für das sogenannte Random Telegraph Noise (RTN) und 1/f - Rauschen. Weiters werden sie auch für unerwünschte Änderungen von Bauteilparametern des MOSFET, wie z.B. eine Verschiebung der Thresholdspannung oder eine Verringerung der Sub-Thresholdslope verantwortlich gemacht.

Einer der wichtigsten Degradiationsmechanismen, nämlich *Negative Bias Temperature Instabilities* (NBTI), wird bei Anlegen einer negativen Stress - Spannung am Gate - Kontakt des p-MOSFETs beobachtet.Besonders bemerkenswert jedoch ist, wenn man NBTI in kleinen Transistoren ($W \sim L \sim 160\,\mathrm{nm}$) studiert, so beobachtet man eine sprunghafte Änderung der Thresholdspannung, welche durch Lade- und Entladevorgänge von einzelnen Defekten verursacht werden. Die Zeitkonstanten solcher Lade- und Entladevorgänge können durch Erhöhen der Temperatur wesentlich verringert werden. Typische Werte für die Feldstärke im Oxid, bei denen NBTI untersucht wird, sind kleiner $8\,\mathrm{MV\,cm^{-1}}$ und die Temperatur wird typisch zwischen 100 - 300 °C variiert.
Eine wichtige Eigenschaft von NBTI ist, dass ab dem Abschalten der Gate - Spannung nach einer gewissen Zeit ein Teil der Defekte relaxiert und der andere Teil permanent geladen bleibt [8]. Zur Modellierung von solchen Einzeldefekten wird ein sogenanntes Vier - Zustands Defektmodell verwendet. Dabei werden die Lade- und Entladevorgänge von den einzelnen Defekten durch statistische Verteilungen beschrieben [5].

2.3 Time-Dependent Defect Spectroscopy (TDDS)

Zur Charakterisierung der in Abschnitt 2.2 behandelten Defekte wurde am Institut für Mikroelektronik an der TU Wien die sogenannte Time-Dependent Defect Spectroscopy entwickelt. Dazu wird eine Einzelmessung N = 100 mal wiederholt und anschließend statistisch ausgewertet. Solch eine Einzelmessung besteht aus zwei Messzyklen - dem **Stresszyklus** und dem **Relaxationszyklus**. Während des ersteren wird, wenn wir einen p-MOSFET betrachten, eine negative Gatespannung $V_{\mathrm{GS}} < V_{\mathrm{Th}}$ angelegt, wobei $V_{\mathrm{DS}} = 0\,\mathrm{V}$ erfüllt sein muss, damit ein homogenes Oberflächenpotential an der Si/SiO_2 Grenzfläche besteht. Es werden beim NBTI - Stress nur Defekte im Oxid geladen. Im Gegensatz dazu wird bei Hot Carrier - Stress eine Spannung $V_{\mathrm{DS}} > 0\,\mathrm{V}$ angelegt, um auch Defekte im Halbleiter zu aktivieren.
Um einen Defekt analysieren zu können, muss dieser nun während der Stressphase geladen werden. Die Zeitdauer des Stresszyklus muss dabei größer als die mittlere Einfangzeit τ_{c} eines Defektes sein [6]. In der darauf folgenden Relaxationsphase wird bei $V_{\mathrm{GS}} \sim V_{\mathrm{Th}}$ eine kleine Drain-Source Spannung angelegt ($V_{\mathrm{DS}} \sim -0.1\,\mathrm{V}$) und der Drain-Source Strom I_{DS} gemessen. Mit fortschreitender Zeitdauer sind sprunghafte Anstiege des Stromes zu beobachten, die dadurch Zustande kommen, dass die Defekte vom geladenen in den neutralen Zustand übergehen. Die dabei während des Relaxationszyklus (Abbildung 3) aufgezeichneten Sprunghöhen an den dazugehörigen Emmissionszeiten τ_{e} werden in einem 2D - Histogramm ausgewertet und geben Auskunft über die mittlere Emissionszeit eines einzelnen Defektes bei den aktuell vorherrschenden Stress- und Relaxationsbedingungen. In weiterer Folge wird die Einfangzeit über ein indirektes Verfahren bestimmt. Die Emissions- und Einfangzeiten werden auch noch bei verschiedenen Temperaturen, Stress- und Relaxationsspannungen extrahiert und anschließend mittels dem Vier-Zustands Defektmodell abgebildet. Daraus lassen sich nun Rückschlüsse auf die Position des Defektes sowie, in Kombination mit DFT Simulationen, auf die atomistische Struktur ziehen. Diese Daten werden anschließend zur Modellierung des Bauteils inklusive Defekten verwendet.

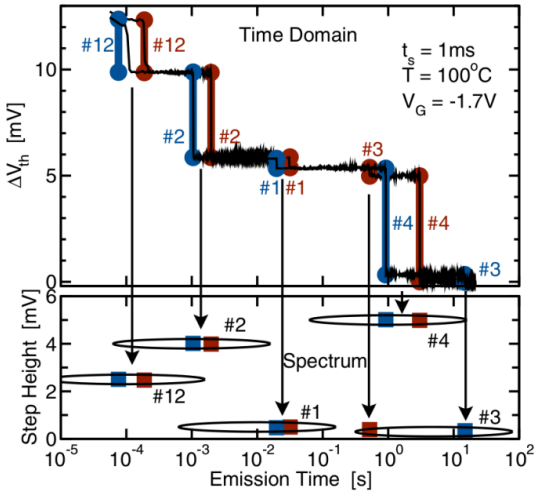


Abbildung 3: Relaxationsverläufe einer TDDS Messung aus [6] - Oben: Zwei Verläufe mit Emissionsevents von fünf verschiedenen Defekten - Unten: 2D - Histogramm, auch Spectral - Map genannt. Jeder Defekt bildet einen Cluster im Histogramm, welcher als sein Fingerabdruck gesehen werden kann.

2.4 Regelungstechnik - Regelkreis

Charakteristisch bei der Regelung einer *Ausgangsgröße y(t)* ist, dass diese zum Regler zurückgeführt wird, wohingegen dies bei Steuerungen nicht der Fall ist. Ein Regelkreis, wie in Abbildung 4 gezeigt, besteht aus dem Regler, auf den die *Führungsgröße r(t)* aufgeschalten wird. Dieser erzeugt gemeinsam mit der *gemessenen Ausgangsgröße* $\overline{y}(t)$ die *Stellgröße u(t)*, welche auf das Stellglied, welches wiederum auf das zu regelnde System - die Regelstrecke wirkt, aufgeschalten wird. Die Differenz zwischen Stellgröße und Ausgangsgröße wird dabei als *Regelabweichung e(t)* bezeichnet [9].

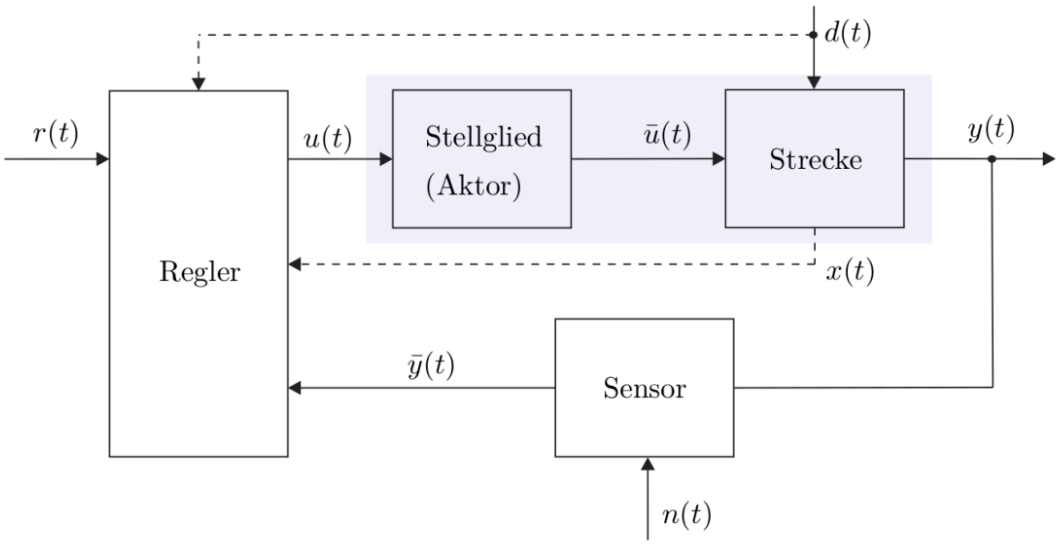


Abbildung 4: Regelkreis aus [9]

Im Falle linearer zeitinvarianter (LTI) Systeme lassen sich die Übertragungsfunktionen für den offenen bzw. geschlossenen Regelkreis bequem mithilfe der Laplace - Transformation berechnen. Bei Vernachlässigung der *Störgröße d(t)* sowie des *Sensorrauschens n(t)* ergibt sich die Übertragungsfunktion L(s) des offenen Regelkreises mit den Übertragungsfunktionen R(s) des Reglers und G(s) der Strecke zu

$$L(s) = R(s)G(s). \tag{2}$$

Daraus folgt die Führungsübertragungsfunktion zu

$$T_{\mathrm{r,y}}(s) = \frac{y(s)}{r(s)} = \frac{L(s)}{1 + L(s)} \tag{3}$$

welche durch inverse Laplacetransformation in den Zeitbereich rücktransformiert werden kann.

2.4.1 Die zeitdiskrete Regelung

Für die Implementierung eines Reglers in Mikroprozessoren müssen die analogen Eingangssignale abgetastet werden. Die so gewonnenen digitalen Signale werden im Prozessor verarbeitet und danach mittels eines DA - Umsetzers in ein analoges Ausgangssignal konvertiert, um als Eingangssignal an das Stellglied übertragen zu werden. Abbildung 5 zeigt das Blockschaltbild des geschlossenen Regelkreises einer solchen Regelstruktur, wobei die Führungsgröße hier als w_{k} bezeichnet ist.

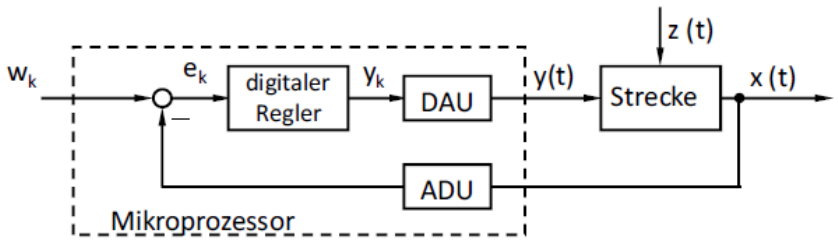


Abbildung 5: Blockschaltbild eines Digitalreglers aus [1]

Übertragungsfunktionen zeitdiskreter linearer Systeme werden mithilfe der Z-Transformation beschrieben [3]. Die Streckenübertragungsfunktion lässt sich auch wie in [9] gezeigt über die Beziehung

$$G(z) = \frac{z-1}{z}\mathcal{Z}\left(\frac{G(s)}{s}\right) \tag{4}$$

aus dem Laplacebereich in den Z-Bereich übertragen.

Wahl der Abtastzeit

Die Wahl der Abtastzeit ist durch die in der Strecke auftretenden Zeitkonstanten und den

damit verbundenen Anstiegszeiten der Sprungantworten der Strecke bestimmt. Beispielsweise ist die Anstiegszeit (siehe Abbildung 6) eines Verzögerungsgliedes erster Ordnung, eines sogenannten PT1-Gliedes, mit der Übertragungsfunktion

$$G(s) = K\frac{1}{1+T_1 s} \tag{5}$$

durch T_1 gegeben.

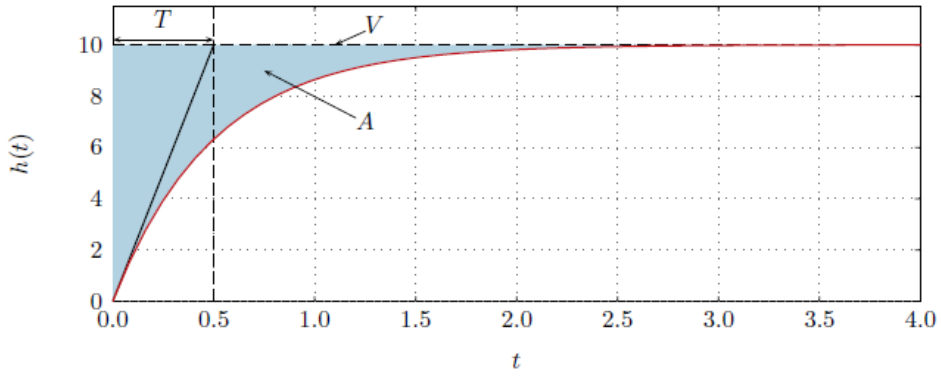


Abbildung 6: Sprungantwort eines PT1-Gliedes mit der Anstiegszeit T und der Verstärkung V (die blau hinterlegte Fläche ergibt sich zu A = VT) [9]

Bei mehreren Energiespeichern ergeben sich, entsprechend der Ordnung des Systems, mehrere Anstiegszeiten, wobei die kleinste auftretende Zeitkonstante für die Wahl der Abtastzeit maßgeblich ist. Die Abtastzeit T_a wird empirisch mittels der Faustformel

$$\frac{\min(t_\mathrm{r})}{T_\mathrm{a}} = 4 \ldots 10 \tag{6}$$

gewählt, wobei t_r für die Anstiegszeit des PT1- bzw. PT2-Gliedes steht. Diese Wahl hat sich für Systeme bestehend aus PT1- und PT2- Gliedern als praktikabel erwiesen [9]. Wie in Kapitel 4.1 noch beschrieben wird, bestehen die hier betrachten Heizvorrichtungen näherungsweise aus PT1-Gliedern bzw. einer Serienschaltung mehrerer dieser PT1-Glieder.

2.5 PID Regler

Der PID Regler kombiniert die Vorteile eines PD - Reglers, welcher eine sehr hohe Anfangsstellgröße liefert, mit jenen des PI - Reglers, der eine vollständige Kompensation der Regelabweichung ermöglicht. Dadurch wird, je nach Wahl der Reglerparameter, eine rasche Führungsregelung oder eine gute Störgrößenkompensation ermöglicht. Die Rechenvorschrift für den PID - Regler in Parallelstruktur, wie in Abbildung 7 dargestellt, ergibt sich mit dem Regelfehler

$$e(t) = r(t) - y(t) \tag{7}$$

zu

$$y(t) = K_\mathrm{p}\left(e(t) + \frac{1}{T_\mathrm{i}}\int e(t)dt + T_\mathrm{d}\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right) \tag{8}$$

wobei T_i und T_d die Integral - und Differiantialparamter durch

$$K_\mathrm{i} = \frac{K_\mathrm{p}}{T_\mathrm{i}}; K_\mathrm{d} = K_\mathrm{p} T_\mathrm{d} \tag{9}$$

bestimmen. Im zeitdiskreten Fall muss Gleichung 8 umgeformt werden zu

$$y_k = K_p \left[e_k + \frac{T_\mathrm{a}}{T_\mathrm{i}} \sum_{i=1}^{k} e_i + \frac{T_\mathrm{d}}{T_\mathrm{a}} (e_k - e_{k-1}) \right] \tag{10}$$

wobei T_a die verwendete Abtastzeit bezeichnet. Das Differential wird in diesem Fall durch einen Rückwärts-Differenzenquotienten ersetzt und das Integral geht in eine Summe über.

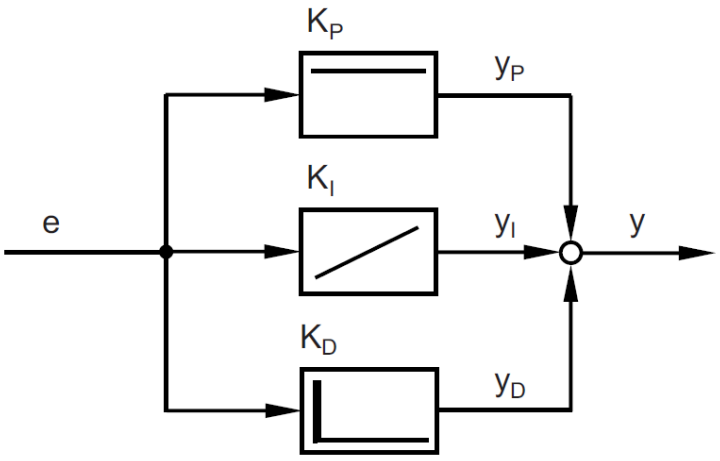


Abbildung 7: Blockschaltbild eines PID - Reglers (Parallelstruktur) aus [1]

3 Umsetzung

Dieses Kapital beschreibt sowohl die bestehenden Hardwarekomponeneten des Temperaturkontrollers, sowie die verwendeten Entwicklungswerkzeuge und die Softwareimplementierung eines PID Reglers.

3.1 Hardware und Entwicklungswerkzeuge

Der bestehende Temperaturkontroller wurde von der Firma mb - Technologies entwickelt und gefertigt. Auch bestehende Firmware für den verwendeten Mikrocontroller ATMega16 von Atmel wurde von mb - Technologies implementiert. Zum Programmieren des Mikrokontrollers wird ein JTAG-ICE3 Modul von Atmel verwendet.
Bei den verwendeten Heizelementen handelt es sich einerseits um die Compact Measurement Extension (CME), welche im Rahmen einer Bachelorarbeit am Institut für Mikroelektronik entwickelt wurde und ein PTC Heizelement verwendet. Als zweites wird ein Ofen von Infineon, welcher mit einer Heizspule aus Wolfram arbeitet, eingesetzt.

Zur Temperaturmessung kommt bei den Heizaufbauten ein PT100 bzw. ein PT1000 Temperaturmesswiderstand zum Einsatz. Der Schaltungsaufbau erlaubt neben der Messung der Temperatur auch eine Messung des Stromes durch das Heizelement, sowie der Spannung am Heizelement, woraus sich die aufgebrachte Heizleistung berechnen lässt. Weiters ist noch die Messung der Temperatur der Platine des Temperaturkontrollers, auf welcher der zur Steuerung verwendete Leistungs - MOSFET sitzt, implementiert (siehe Anhang).

Für die Temperaturmessung müssen zyklisch drei Spannungswerte ausgelesen werden, um die Spannung am Temperaturmesswiderstand zu bestimmen (zwei Potentiale am Widerstand und ein Referenzspannungswert) und diese in eine äquivalente Temperatur umzurechnen. Zusätzlich wird in jedem vierten Ausleseschritt, jeweils hintereinander Strom bzw. Spannung am Heizelement, sowie die Temperatur der Platine ausgelesen.
Die analogen Spannungswerte werden dazu über einen Multiplexer an den AD - Wandler angelegt welcher am Port D des Mikrokontrollers angeschlossen ist. Die minimale Wandlungsdauer des AD - Wandlers beträgt 160 ms pro Messung. Um zu garantieren, dass die Wandlung fehlerfrei abgeschlossen wird, werden noch zusätzliche 40 ms abgewartet, wodurch sich eine Zykluszeit von 200 ms pro Spannungsmessung ergibt. Daraus errechnet sich die verwendete Abtastzeit des Regelkreis bei vier Einzelmessungen pro Abtastschritt zu $\mathbf{T_a = 800ms}$. Für das zyklische Auslesen der Messwerte wird dafür das Compare-Register des Timer1 des Mikrokontrollers so gesetzt, dass die ISR des Timers alle 200 ms ausgelöst wird und der entsprechende Wert aus dem AD - Wandler entnommen wird. Wie bereits in Abschnitt 2.4.1 erklärt wird, ist dadurch auch die Dynamik des geschlossenen Regelkreises beschränkt. Die hier verwendeten Heizelemente sind allerdings wesentlich träger, weshalb die verwendete Abtastzeit ausreichend ist (siehe auch 4.1.1).

Die maximale Ausgangsleistung des Reglers ist durch den verwendeten Transformator gegeben. Dieser erzeugt sekundärseitig eine Spannung von 32 V. Bei einem maximalen Heizstrom von 3 A ergibt sich eine maximale Ausgangsleistung $P_{\max} = 96W$. Diese wird über ein PWM - Signal vom Kontroller durch Vorgabe des sogenannten Dutycycle (0 - 100 %) geregelt, welches anschließend geglättet wird und über einen Leistungs - MOSFET die Heizspannung vorgibt.

Portierung des bestehenden Code auf AVR-GCC

Zur Kompilierung des C-Codes im Mikrocontroller auf einer Linux-basierten Plattform wurde der AVR-GCC Compiler ausgewählt. Die Umwandlung der vom AVR-GCC erzeugten Objektdatei in den Maschinencode (.hex Datei), welchen in den Flash - Speicher des Mikrocontrollers geschrieben wird, erfolgt mithilfe des Programmes AVRDude. Um den Kompilierschritt und das Neubeschreiben des Speichers zu vereinfachen, wird ein Makefile verwendet, das alle notwendigen Befehle enthält (siehe Anhang).
Da eine vollständige Neuprogrammierung des Controllers inklusive Displaysteuerung, serielle Schnittstellenkommunikation, Temperaturmessung usw. den zeitlichen Rahmen dieser Arbeit überschritten hätte, ist der bestehende Firmware Code, welcher mit der Entwick-

lungsumgebung CodeVisionAVR und dem entsprechenden Compiler entwickelt wurde, für den AVR-GCC Compiler portiert worden. Nachfolgend werden die wichtigsten Unterschiede der beiden Compiler kurz zusammengefasst.

Speicherzugriff

Der Speicherzugriff auf den internen SRAM - Speicher oder das EEPROM des Mikrokontrollers erfolgt mit dem AVR-GCC Compiler über die Schlüsselwörter:

```
PROGMEM
EEMEM
```

Die auf die Schlüsselwörter folgenden Variablen werden im entsprechenden Speicherbereich deklariert.
Der Zugriff auf entsprechende Variablen im EEPROM erfolgt über die Funktionen:

```
//Update einer EPROM Variable
void eeprom_update(int var)
//Auslesen der Variablen aus EEPROM
void eeprom_read(int &var)
```

Die Funktionen `eeprom_update()` überschreiben eine Variable nur im Fall einer Änderung, um die Lebensdauer des EEPROM (ca. 100000 Schreibzyklen) zu verlängern.

Interrupts

Interruptserviceroutinen müssen für den AVR-GCC Compiler wie folgt implementiert werden:

```
ISR(vectorID)
{
//... function ...
}
```

Portadressierung

Mit dem AVR-GCC Compiler können nicht direkt einzelne PORT PINs adressiert werden, wie es beim CodeVisionAVR über zum Beispiel

```
PORTB.2 = 1;
PORTB.4 = 0;
```

möglich wäre. Dies wird für den AVR-GCC Compiler durch entsprechende Maskierung ersetzt:

```
        PORTB |= 0x04;
        PORTB &= 0xEF;
```

3.2 Implementierung in Software

Formel (10) für den zeitdiskreten PID - Regler muss zur Implementierung im Mikrokontroller dahingehend angepasst werden, dass der Integralanteil rekursiv berechnet wird, da ansonsten unbegrenzt viel Speicher notwendig wäre, um die vorausgegangenen Abtastwerte zu speichern. Für den integralen Anteil zum Zeitpunkt $k = nT_a$ gilt somit

$$y_{int,k} = y_{int,k-1} + K_i e_k \tag{11}$$

Die Regleranteile werden einzeln berechnet und anschließend addiert.

3.2.1 Regler im Mikrokontroller

Die Reglerimplementierung wird nach dem in Kapitel 2.4.1 erläuterten Formeln implementiert. Um die Stellgrößenbeschränkung für die unterschiedlichen Heizelemente zu realisieren, muss ein oberes Limit für die Heizspannung und damit auch der Heizleistung gesetzt werden.

```
case PID:
  cerr[1]    = cerr[0];
  cerr[0]    = oven_target - oven_temp;
  oven_ramp = cerr[0] - cerr[1];

  kp = eeprom_read_float(&cntr_pro);
  ki = eeprom_read_float(&cntr_int);
  kd = eeprom_read_float(&cntr_dif);

  power_pro = limit(kp * cerr[0],-oven_limit,oven_limit);
  //Anti - Wind up
  power_lim = limit (power_pro, -oven_limit, oven_limit);
  //recursive integral power
  power_int = limit(power_int + ki * cerr[0],
                    -power_lim,oven_limit - power_lim);
  if(fabs(cerr[0]) < TEMP_TARGET)
  {
        power_dif = kd * (oven_ramp);
  }
  else
  {
        power_dif = 0.0;
  }
  set_heater (power_pro + power_int + power_dif);
  break;
```

Es wird zuerst die Temperaturabweichung (=Regelabweichung) berechnet, um diese mit den PID-Koeffizienten zu multiplizieren. Mit der Funktion `limit()` lässt sich dabei jeder Regleranteil entsprechend der gewünschten Stellgrößenbeschränkung limitieren. Die entsprechende Beschränkung des I-Anteils wird dabei auch als Anti-Windup Maßnahme bezeichnet und lässt sich von der für die jeweils gewünschte Zieltemperatur benötigten Leistung abschätzen. Da die stationäre Regelabweichung nur vom I-Anteil des Reglers kompensiert wird, ist ein unbeschränktes Anwachsen desselben bei langsamen Regelungen unerwünscht. Der somit entstehende hohe I-Anteil würde erst bei Überschreiten der Zieltemperatur wieder abgebaut werden und sich negativ auf die Regeldauer auswirken.
Zusätzlich wird ab einem gewissen Toleranzbereich um den Zielwert, der (meist hohe) D-Anteil nicht mehr berücksichtigt, um bei kleinen Messschwankungen eine stabile und schnelle Zielpunktregelung zu garantieren. Ab Erreichen eines definierten Toleranzbereichs um die Zieltemperatur wird somit auf reine PI-Regelung umgeschalten.

3.2.2 Steuerung des Reglers über serielle Schnittstelle

Der Temperaturcontroller verfügt über eine Serielle Schnittstelle, über welche mittels USB - Seriell Konverter mit dem steuernden Rechner kommuniziert wird. Dabei wurde das vorhandene Modul *TDDSUnitFurnace_v1_9.py* mit dem bestehenden Befehlssatz (siehe Dokumentation TC1010 - Controller), um einige Befehle speziell zur Erweiterung des Reglers in Abschnitt 4 ergänzt.
Ein neuer Befehl muss dabei sowohl im Temperaturcontroller in der Funktion `rs232()` über einen den Befehl bezeichnenden String implementiert werden, sowie als Methode der Klasse `controlFurnace` hinzugefügt werden, welche den entsprechenden String auf die serielle Schnittstelle schreibt. Dieser wird in der entsprechenden Interrupt-Service-Routine des Mikrokontrollers in einem Buffer zwischengespeichert und in der Funktion `rs232()` verglichen und je nach Befehl ein Parameter gesetzt oder ausgelesen.
Beispielhaft wird hier das Auslesen der Temperatur angegeben:

```
def getTemperature(self):
 ret = self.writeCommand('GET TEMP')

 if ret.find('NO OVEN') != -1:
  common.printWarning('Furnace communication error -
                       getTemp - NO OVEN')
  return 0

 try:
  return float(ret)
 except:
  common.printWarning('Furnace communication error - getTemp')
```

```
void rs232 (void)
{
 float f;
 int   n;
```

```
 if (!receive_valid) return;
 receive_pointer = 0;
 transmit_count  = 0;
  ....
 else if (receive_keyword ("GETTEMP"))
 {
  if (!oven_fail)
  {
    transmit_float (oven_temp, 2);
  }
  else
  {
    transmit_error (ERROR_NOOVEN);
  }
 }
}
```

Eine Übersicht der neu hinzugefügten Befehle findet man im Anhang zusammengestellt.

3.3 Regelung einer Rampe als Führungsgröße

Beim Aufschalten einer Rampe als Führungsgröße wird ein weiterer Integrator im Regelkreis notwendig, um eine stationäre Regelabweichung zu eliminieren [9].
Dies wird speziell ersichtlich, wenn man als Ausgangsgröße die Temperaturänderungsrate wählt, wodurch die Übertragungsfunktion (5) zu

$$G(s) = K\frac{s}{1+T_1 s} \tag{12}$$

wird. Die Nullstelle im Nenner muss durch eine Polstelle im Zähler der Übertragungsfunktion des Reglers kompensiert werden. Die bestehende PID Reglerstruktur wurde deshalb um einen Integrationsschritt am Ausgang des Reglers erweitert:

```
case RAMP:

  oven_ramp_temp[1] = oven_ramp_temp[0];
  oven_ramp_temp[0] = oven_temp;
  oven_ramp = (oven_ramp_temp[0] - oven_ramp_temp[1]) / ADC_RATE;

  cerr[1] = cerr[0];
  cerr[0] = (oven_target_ramp - oven_ramp);

  power_pro = limit(kp * cerr[0],-30.0,30.0);
  power_int = limit(power_int + ki * cerr[0],0,30.0);
  power_dif = kd * (cerr[0] - cerr[1]);

  power = limit(power + ki2 * (power_pro +
                power_int + power_dif),0.0,oven_limit);
```

```
if((oven_target - oven_temp) < 10.0)
{
  cntr_mode = PID;
  power_int = 0;
  cerr[0]   = oven_target - oven_temp;
}

set_heater(power);
break;
```

Die `oven_target` Variable stellt im Rampen - Modus eine Temperaturänderungsrate dar und die Rampe eine weitere Änderungsrate dieser.
Nachteilig wirkt sich die Verwendung mehrerer Integratoren jedoch auf die Dynamik des Regelkreises aus, weil dadurch die Durchtrittsfrequenz des offenen Regelkreises, welche ein Maß für die Dynamik der Regelung darstellt, reduziert wird [9].

4 Optimierung

Der im vorigen Kapitel beschriebene PID - Regler lässt sich vor allem sehr flexibel für verschiedene nicht näher bestimmte Regelstrecken einsetzen, die sich in weiten Teilen als linear erweisen. Nichtlinearitäten können nur durch Änderung der Regelparameter berücksichtigt werden. Um einen weitere Optimierung des Reglers zu erreichen, ist allerdings eine genaue Kenntnis der Regelstrecke erforderlich, um zu jedem Zeitpunkt die Stellgröße entsprechend vorgeben zu können. In diesem Kapitel wird daher eine Umsetzung eines Reglers bei Kenntnis der Regelstrecke gezeigt.

4.1 Modellierung

Um eine mathematische Beschreibung und eine damit einhergehende Optimierung des Systems zu ermöglichen, muss zuerst die zu regelnde Strecke gemessen und modelliert werden. Eine solche Modellierung kann auch bei einfachen thermischen Systemen bereits sehr aufwändig werden und nur mehr durch numerische Computational Fluid Dynamics (CFD) - Simulationen durchgeführt werden. Dabei wird zum Beispiel auf die finite Differenzen Methode zurückgegriffen [10]. Im Rahmen dieser Arbeit wird auf die Messung der Strecke und die Synthetisierung einer Streckenfunktion anhand der Messdaten eingegangen. Dabei auftretende Nichtlinearitäten werden in diesem Zusammenhang allerdings im ersten Schritt noch nicht berücksichtigt, da die folgenden Berechnungen sich auf eine lineare Systemtheorie stützen.

Im einfachsten Fall lässt sich die vorliegende Strecke über konzentrierte thermische Bauelemente beschreiben. Die elektrische Leistung am Widerstand des Heizelementes wird dabei in thermische Leistung umgewandelt. Die umgesetzte Wärmeenergie überwindet einen thermischen Widerstand, und erwärmt gemäß dem zweiten Hauptsatz der Thermodynamik das umgebende kältere Material. Die dafür benötigte Zeit ist von der spezifischen thermischen Kapazität und somit von der Masse und dem Material des Körpers abhängig. Ein entsprechendes Ersatzschaltbild mit konzentrierten Bauelementen ist in Abbildung 8 gezeigt.

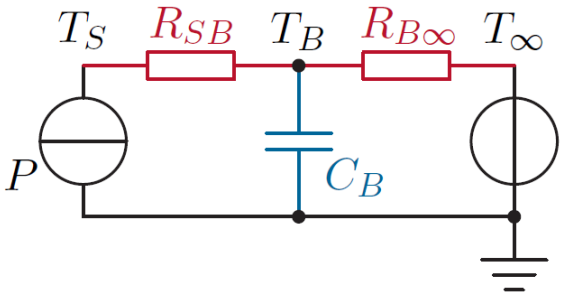


Abbildung 8: Darstellung der CME - Heizeinheit mit konzentrierten Bauelementen aus [10]

Der Wärmeübergangskoeffizient $\alpha_{\mathrm{SB}} = 1/R_{\mathrm{SB}}$ beschreibt dabei am Beispiel der CME den Übergang vom Heizelement auf den Aluminiumkörper und die Kapazität $C_{\mathrm{B}} = m_{\mathrm{B}} c_{\mathrm{B}}$ die Wärmekapazität eines Körpers mit der Masse m_{B} und der spezifischen Wärmekapazität c_{B}. Der Widerstand $R_{\mathrm{B}\infty}$ beschreibt schließlich den thermischen Übergangswiderstand zur

Umgebung.
Die Übertragungsfunktion dieses linearen Netzwerkes lässt sich nun mit

$$m_{\mathrm{B}} c_{\mathrm{B}} \frac{\mathrm{d}T}{\mathrm{d}t} = -Q + P, \tag{13}$$

wobei

$$Q = \alpha_{\mathrm{B}\infty}(T_{\mathrm{ist}} - T_{\infty}) \tag{14}$$

der an die Umgebung abgeführten Wärmemenge pro Zeiteinheit und P der zugeführten Wärmeleistung entspricht, berechnen. Nach Transformation in den Laplace - Bereich erhält man

$$G(s) = \frac{P(s)}{T_{\mathrm{B}}(s)} = \frac{k}{1 + s\tau_1} \tag{15}$$

mit $\tau_1 = \frac{C_{\mathrm{B}}}{\alpha_{\mathrm{B}\infty}}$ und $k = R_{\mathrm{B}\infty}$. Dies ist die Übertragungsfunktion eines PT1 - Gliedes und entspricht einem Tiefpassverhalten.

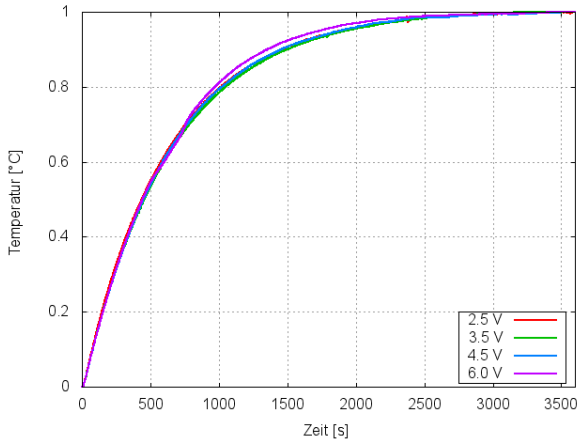
4.1.1 Bestimmung Sprungantwort der Regelstrecke

Bei der Bestimmung der Sprungantwort der Regelstrecke wird sprungartig eine Führungsgröße auf die Regelstrecke aufgeschalten und die Systemantwort am Ausgang der Strecke aufgezeichnet. Anhand der somit gewonnen Sprungantwort werden die Parameter der Strecke direkt mittels PT1 - Glied im Zeitbereich mit

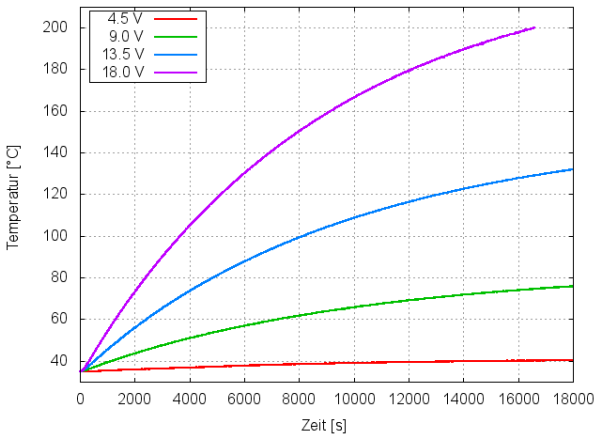
$$h(t) = k\left(1 - \mathrm{e}^{-\frac{t}{\tau_1}}\right) \tag{16}$$

modelliert.

Zur Bestimmung der Streckenparameter von PT1 und PT2 - Gliedern sind in der Praxis oft empirische Methoden, wie das Wendetangentenverfahren gebräuchlich. Auch MATLAB Simulink bietet mit dem *PID-Tuner* ein Tool, welches zur Bestimmung der Streckenparameter verwendet werden kann, wobei hier die aufgenommene Sprungantwort manuell an eine PT1-Sprungantwort angepasst wird. Um jedoch eine genauere Bestimmung der Parameter zu gewährleisten, wurden die aufgenommenen Sprungantworten, mittels *Least-Square - Methode* auf die Funktion (16) gefittet. Dabei werden die Parameter τ_1 und k der Funktion $h(t)$ derart variiert, dass der quadratische Fehler der Datenpunkte zur Funktion minimal wird. Das Tool *gnuplot* stellt dafür die Funktion *fit* zur Verfügung. Mithilfe dieses *Least-Square* Fits werden die Parameter für die CME in Tabelle 1 und den Ofen in Tabelle 2 ermittelt.
Um die Streckenparameter zu bestimmen, wird die Sprungantwort mit verschiedenen Sprunghöhen gemessen. Dazu werden verschiedene konstante Heizspannungen an das Heizelement angelegt und die Temperatur des Heizelementes aufgezeichnet. Die auf die erreichte Temperatur normierten und auf die Funktion (16) gefitteten Sprungantworten sind in Abbildung 9 gezeigt. Eine Aufnahme mit höheren Heizspannungen wäre nicht möglich, da die maximal zulässige Temperatur für den CME - Aufbau von 140 °C sonst überschritten würde.



(a) Sprungantworten der CME bei verschiedenen Heizspannungen (normiert)



(b) Sprungantworten des Ofens bei verschiedenen Heizspannungen

Abbildung 9: Sprungantworten CME und Ofen

Heizspannung [V]	k [K/W]	$\tau_1[s]$
2.5	21.99	628.26
3.5	19.33	652.58
4.5	16.68	652.43
6.0	14.39	613.58

Tabelle 1: Ermittelte Parameter aus Sprungantworten CME

Heizspannung [V]	k [K/W]	$\tau_1[s]$
4.5	11.938	10452.22
9.0	9.990	10440.95
13.5	9.009	9996.99
19.0	8.228	9241.93

Tabelle 2: Ermittelte Parameter aus Sprungantworten Ofen

Die Streckenanalyse bei verschiedenen Heizspannungen zeigt schon deutlich die Nichtlinearitäten der Strecke. Etwa den thermischen Übergangskoeffizienten durch freie Konvektion, welcher selbst von der Temperatur abhängt, ebenso wie die temperaturabhängige Wärmekapazität.

4.2 Reglerentwurf anhand des Modells

Die in Kapitel 3 beschriebene Regelung mittels eines PID - Reglers stellt zwar eine einfache Methode zur Regelung von Strecken mit nicht näher bekannten Streckenparametern dar, aber eine exakte Bestimmung der optimalen Regelparameter durch Berechnung und

numerische Simulation ist nur für lineare Strecken möglich. Empirische Bestimmung der Parameter ist für die vorliegenden Heizelemente aufgrund der großen Zeitkonstanten ebenso nicht wünschenswert. Daher wurde ausgehend von der Modellierung in Abschnitt 4.1 die vom Regler auszugebende Heizspannung für eine vorgegebene Heizrampe berechnet.
Die Heizspannung hängt dabei mit der umgesetzten Wärmeleistung über

$$P = \frac{U^2}{R(T)} \tag{17}$$

zusammen, wobei der elektrische Widerstand R(T) selbst von der Temperatur abhängig ist, was sich vor allem beim Heizelement der CME, wie in Abschnitt 4.2.1 beschrieben, sehr stark auswirkt.

4.2.1 Entwurf für CME und Ofen

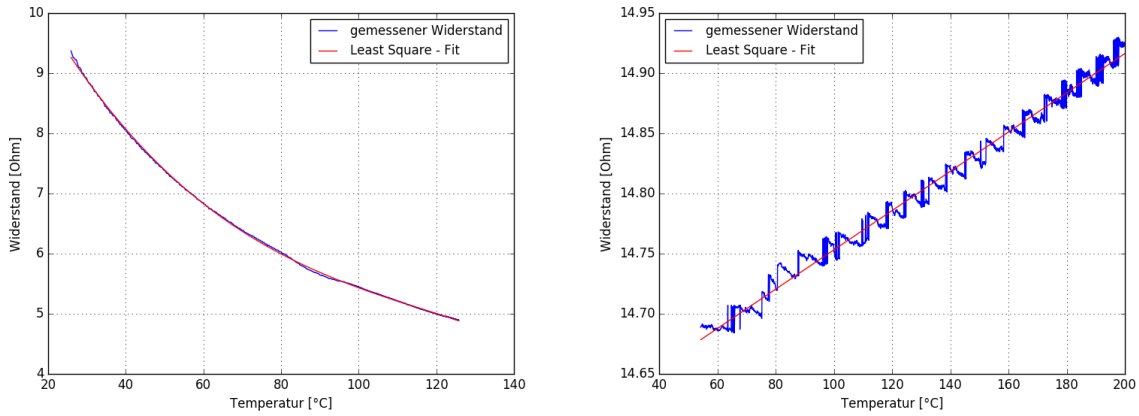
Um die Heizspannung korrekt vorgeben zu können, muss auch die Temperaturabhängigkeit des elektrischen Widerstandes bestimmt werden. Dazu wird das Heizelement der CME über den gesamten Arbeitsbereich (25 - 140 °C) geheizt und dabei der Strom durch das und die Spannung am Heizelement aufgezeichnet, um daraus den elektrischen Widerstand des Heizelementes zu berechnen.
Die gewonnen Daten werden in Python eingelesen und durch ein Polynom dritter Ordnung für die CME und ein Polynom erster Ordnung für den Ofen beschrieben. Hier zeigt sich deutlich, dass die CME ein NTC-Verhalten im Arbeitsbereich aufweist, wie es etwa bei Halbleitern auftritt und näherungsweise der Beziehung

$$R(T) = R_{T_0} e^{b\left(\frac{1}{T} - \frac{1}{T_0}\right)} \tag{18}$$

mit der Materialkonstante b genügt. Dieses Verhalten ist bedingt durch die Erhöhung der Zahl der freien Ladungsträger und somit der Eigenleitfähigkeit mit zunehmender Temperatur.
Der Ofen mit Wolfram-Heizdraht erfüllt die in weiten Temperaturbereichen gültige Näherung für Metalle (PTC - Verhalten)

$$R(T) = R_{T_0} \left[1 + \alpha \left(T - T_0\right)\right] \tag{19}$$

[11].



(a) CME (b) Ofen

Abbildung 10: Gemessener elektrischer Widerstand der Heizelemente CME und Ofen und darauf gefittete Funktion als Polynom dritter Ordnung bzw. erster Ordnung

Abbildung 10 zeigt, dass sich der Widerstand der CME bei der maximalen Temperatur fast halbiert. Der Widerstand des Heizelementes als Funktion der Temperatur wird mit den ermittelten Koeffizienten des Polynoms als Funktion im Mikrokontroller abgebildet.

```
float res_CME (float t)
{
  return(-2.68E-06 *t*t*t + 9.792E-04 *t*t -1.3913E-01 *t + 12.235);
}
```

```
float res_oven (float t)
{
  return (0.0012*t+14.6987);
}
```

Um eine stationäre Regelabweichung auszuregeln, wird ein Regelgesetz mit zwei Freiheitsgraden

$$\frac{\mathrm{d}e}{\mathrm{d}t} = -k_\mathrm{p} e + k_\mathrm{i} \int e dt \tag{20}$$

mit

$$e = T_\mathrm{Soll} - T_\mathrm{Ist} \tag{21}$$

angewandt. Dies entspricht einem PI - Regler. Einsetzen von (21) in (20) und Verwendung von (13), (14) und (17) ergibt

$$\frac{\mathrm{d}T_\mathrm{Soll}}{\mathrm{d}t} - \left(\frac{1}{m_\mathrm{B} c_\mathrm{B}} \left(\frac{U^2}{R(T)} - \alpha_\mathrm{SB}(T_\mathrm{Ist} - T_\infty\right)\right) = -k_\mathrm{p} \left(T_\mathrm{Soll} - T_\mathrm{Ist}\right) + k_\mathrm{i} \int \left(T_\mathrm{Soll} - T_\mathrm{Ist}\right) dt \tag{22}$$

Nach Umformung ergibt sich die Regelspannung zu

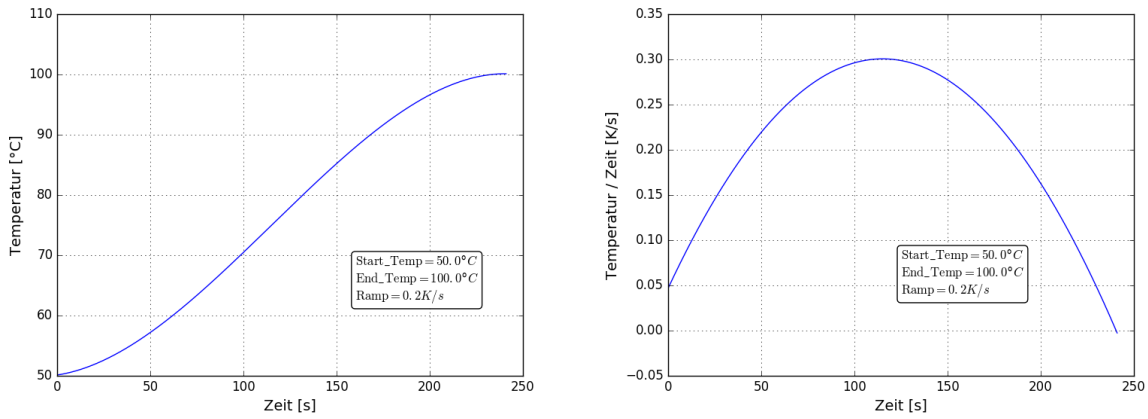
$$U = \sqrt{R(T)C_B \left[\underbrace{\frac{\mathrm{d}T_{Soll}}{\mathrm{d}t}}_{1} + \underbrace{\frac{\alpha_{SB}}{C_B}\left(T_{ist} - T_{\infty}\right)}_{2} + \underbrace{k_p\left(T_{Soll} - T_{Ist}\right)}_{3} + \underbrace{k_i \int \left(T_{Soll} - T_{Ist}\right) dt}_{4} \right]}. \tag{23}$$

Diese Gleichung wird, wie in Abschnitt 2.4.1 beschrieben, diskretisiert und im Mikrokontroller eingebunden.

Wie in Gleichung (23) ersichtlich, ist nun auch eine Vorgabe der Rampe durch den Term 1 möglich. Dies wird so realisiert, dass die gewünschte durchschnittliche Temperaturänderungsrate zusammen mit der Endtemperatur vorgegeben wird. Auf diese Fixpunkte wird ein Polynom dritter Ordnung gefittet und dessen Zeit-Ableitung gebildet, wie in Abbildung 11 gezeigt. Dies wird im Pythonmodul *TPatterngernerator.py* (siehe Anhang) umgesetzt und die generierten Parameter des resultierenden Polynoms zweiter Ordnung werden dem Mikrokontroller übergeben. Die Temperaturrampe kann somit durch die Funktion *target_ramp*() zu jedem Zeitpunkt berechnet werden.

Durch die Rampenvorgabe wird auch die Zeit errechnet, nach welcher das Heizelement die Zieltemperatur erreichen sollte. Ab diesem Zeitpunkt wird die Temperaturänderungsrate konstant auf 0 gehalten (Term 1 in (23)). Eine eventuell verbleibende Regelabweichung wird nun durch die Terme 3 und 4 in (23) korrigiert (PI - Regelung), welche erst nach der berechneten Endzeit der Rampenvorgabe wirksam werden.

```
float target_ramp(float t)
{
  return (ramppar[0] *t*t + ramppar[1]*t + ramppar[2]);
}
```



(a) Temperaturvorgabe für Regelung von 50 °C auf 100 °C bei Rampe von 0.2 K s^{-1}

(b) Temperaturänderungsrate für Regelung von 50 °C auf 100 °C bei Rampe von 0.2 K s^{-1}

Abbildung 11: Temperaturrampenvorgabe

Für die Implementierung im Mikrokontroller wird eine weitere Regeloption in der case - Anweisung hinzugefügt (siehe Anhang).

4.3 Simulation der Regelung in Python

Durch die bisher beschriebene Modellierung des Systems kann nun der geschlossene Regelkreis sehr einfach berechnet werden. Dadurch lassen sich, ohne teilweise zeitaufwändige empirische Methoden, optimale Parameter für die Regelung bestimmen. Die Berechnung des Regelkreises wurde dazu in Python implementiert. Hierfür wurde das Modul *Simulate.py* (siehe Anhang) entwickelt. Zur Berechnung des Reglerausgangs wird die in Abschnitt 4.2.1 beschriebene Formel (23) zeitdiskretisiert und die Spannungsvorgabe für das Heizelement berechnet. Zur Berechnung der Temperatur am Heizelement wird die Differentialgleichung (13) gelöst, wodurch sich die Temperatur zu

$$T(t) = T_\infty + \frac{U(t)^2}{R(T)\alpha} + C_1 e^{-\frac{\alpha}{C}t} \tag{24}$$

ergibt. Mit der Anfangsbedingung, dass zum Zeitpunkt t = 0 die Temperatur gleich der aktuellen Temperatur ist, also $T(0) = T_{\text{act}}$, wird die Konstante C_1 berechnet und (24) wird zu

$$T(t) = (T_{\text{act}} - T_\infty)\, e^{-\frac{\alpha}{C}t} + \left(1 - e^{-\frac{\alpha}{C}t}\right) \frac{U^2(t)}{R(T)\alpha} + T_\infty \tag{25}$$

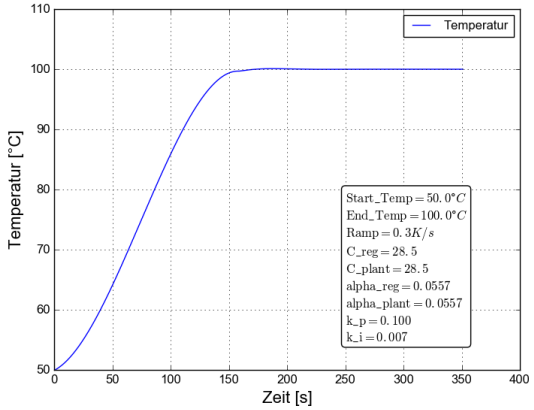
umgeformt. Die Gleichung wird ebenso zeitdiskretisiert und für die Zeitpunkte $t = kT_a$ mit $k \in \mathbb{N}$ berechnet. Dadurch ergibt sich für den k-ten Abtastzeitpunkt

$$T_k = (T_{k-1} - T_\infty)\, e^{-\frac{\alpha}{C}T_{\text{a}}} + \left(1 - e^{-\frac{\alpha}{C}T_{\text{a}}}\right) \frac{U_k^2}{R\left(T_{k-1}\right)\alpha} + T_\infty. \tag{26}$$

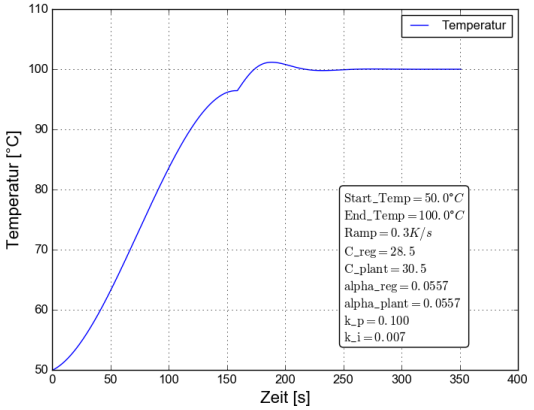
Der Simulator wurde als Klasse `Simul` implementiert, welche die Instanzattribute `cap_r, cap_s, alp_r, alp_s` und `t_amb` enthält, wobei damit die Wärmekapazitäten bzw. die Wärmeübergangskoeffizienten von Regler und Strecke, sowie die im Regler fixierte Umgebungstemperatur beschrieben wird. An die Methode `simulate` werden dann Start- und Endtemperatur, die Temperaturänderungsrate, die gewünschte Simulationszeit, die Regelparameter für Proportional- und Integralanteil und die aktuelle Umgebungstemperatur übergeben. Ruft man die Methode auf, so werden die Heizspannung unter Anwendung der Gleichung (26) und die Temperatur des Heizelements mit Gleichung (23) berechnet, wie in Abbildung 12 dargestellt.

Zusätzlich ist es auch noch möglich die Anti-Windup Limitierung für den Integralanteil zu berechnen, wodurch ein unerwünschtes Überschwingen der Temperatur bei nicht optimal gewählten Regelparametern vermieden wird. Da im stationären Fall zur Berechnung der Heizspannung nur die Terme 2 und 4 in Gleichung (23) wirksam sind, werden die Terme 1 und 3 zur Berechnung der Anti-Windup Limitierung auf null gesetzt. Nun wird die benötigte Heizspannung für die Zieltemperatur und jene für die Zieltemperatur plus der erlaubten Temperaturabweichung berechnet und die Differenz dieser beiden Heizspannungen gebildet. Diese Differenz wird als Beschränkung für den Integralanteil übernommen.

Die Berechnung wird durch die Methode `getInteMax(maxDiffTemp,T_Target)`, welcher die maximale erlaubte Temperaturabweichung und die Zieltemperatur übergeben werden, implementiert. Abbildung 13 zeigt die Wirkung der Windup - Beschränkung mit verschiedenen Maximalabweichungen.

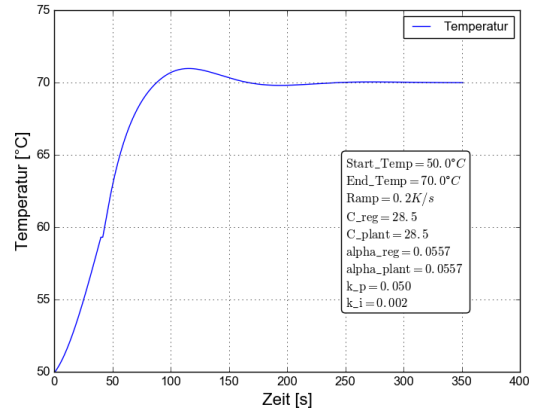


(a) Simulation der Temperaturrampe für Regelung von 50 °C auf 100 °C bei Rampe von 0.02 °C s^{-1} bei gleichen Parametern für Strecke und Regler

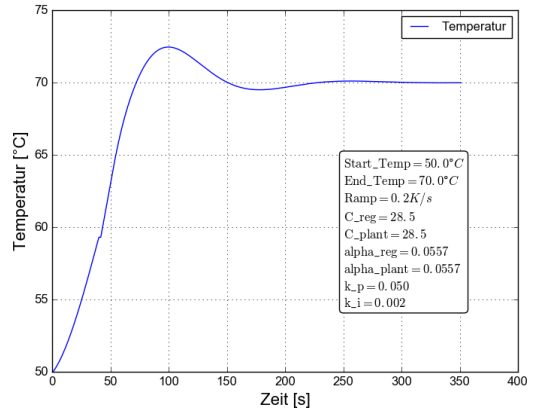


(b) Simulation der Temperaturrampe für Regelung von 50 °C auf 100 °C bei Rampe von 0.02 °C s^{-1} bei unterschiedlichen Parametern für Strecke und Regler

Abbildung 12: Simulation des Reglers



(a) Anti-Windup für maximale Abweichung von 1 °C



(b) Anti-Windup für maximale Abweichung von 2.5 °C

Abbildung 13: Simulation Antiwindup

5 Messungen

In diesem Abschnitt wird das Regelverhalten der unterschiedlichen Reglerimplementierungen für die verschiedenen Heizelemente verglichen. Außerdem wird auf die Abweichungen zwischen realen Heizelementen und Simulation eingegangen und die Ursachen der Differenzen erörtert.

5.1 CME

Die CME weist aufgrund ihrer geringeren Masse und somit auch kleineren Wärmekapazität ein schnelleres Regelverhalten auf. Für eine erste Abschätzung der Regelparameter wird die Wärmekapazität aufgrund der Masse des Heizelementes berechnet. Die Masse wird zu $m_{\mathrm{CME}} = 21\,\mathrm{g}$ vermessen, wodurch sich mit der spezifischen Wärmekapazität von Aluminium $c_{\mathrm{Alu}} = 897\,\mathrm{J\,kg^{-1}\,K^{-1}}$ die Wärmekapazität zu $C_{\mathrm{CME}} = 18.84\,\mathrm{J\,K^{-1}}$ errechnet. Der Wärmeübergangskoeffizient lässt sich aus der Sprungantwort abschätzen. Dazu wird für eine bestimmte Zieltemperatur die Sprungantwort, deren Endtemperatur nahe an der Sollgröße liegt, ausgewählt und aus dem ermittelten Verstärkungsfaktor $k = 1/\alpha$ der Koeffizient α berechnet(siehe Gleichungen (15) oder (16)).

5.1.1 PID Regler

Beim PID Regler (siehe 2.5) für die CME gilt es aufgrund der Stellgrößenbeschränkung des Reglers (nur positive Heizspannung möglich - keine aktive Kühlung) ein Überschwingen der Führungsübertragungsfunktion zu vermeiden. Abbildung 14 zeigt die Führungsübertragungsfunktion des Regelkreises für verschiedene Zieltemperaturen mit den PID Parametern aus Tabelle 3.

k_P	k_I	k_D
2.5	0.035	25.0

Tabelle 3: Parameterset für Regelungen in Abbildung 14

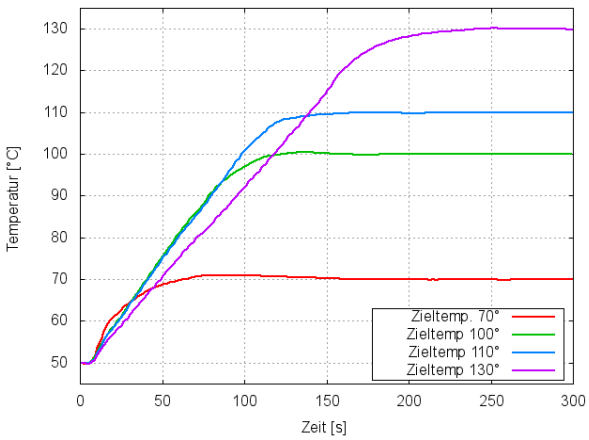


Abbildung 14: PID Regelung bei verschiedenen Zieltemperaturen

Wie in Abbildung 14 zu sehen ist, ergibt sich für eine Zieltemperatur von 100 °C ein nahezu perfektes Regelverhalten. Die Abweichungen der Regelkurven bei niedrigeren und höheren Temperaturen ist durch Nichtlinearitäten der Regelstrecke (Temperaturabhängigkeit der thermischen Kapazität und des Wärmeübergangskoeffizienten) zu erklären. Die unterschiedlichen maximalen Temperaturänderungsraten sind eine Konsequenz der Strombegrenzung des Reglers für das Heizelement.
Eine Möglichkeit zur Berechnung der Temperaturabhängigkeit des Wärmeübergangskoeffizienten eines Körpers wie auch Daten zur Temperaturabhängigkeit der Wärmekapazität findet man in [7].

5.1.2 PID Regler für Rampe

Wie in Abschnitt 3.3 beschrieben, wird beim Aufschalten einer Rampe als Führungsgröße ein zweiter Integrator notwendig. Abbildung 15 zeigt die Führungsübertragungsfunktion für unterschiedliche Zielrampen und die Abweichungen von den Sollgrößen mit den Parametern aus Tabelle 4.

k_P	k_{I1}	k_{I2}	k_D
2.0	0.07	1.0	0.0

Tabelle 4: Parameterset für Regelungen in Abbildung 15

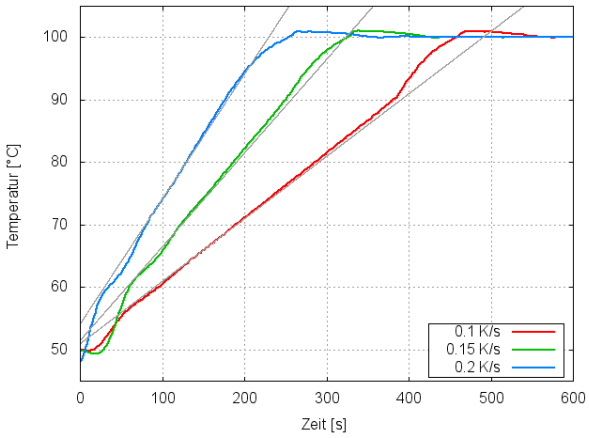


Abbildung 15: PID Regler mit zusätlichem Integrator zur Regelung einer Rampe als Führungsgröße

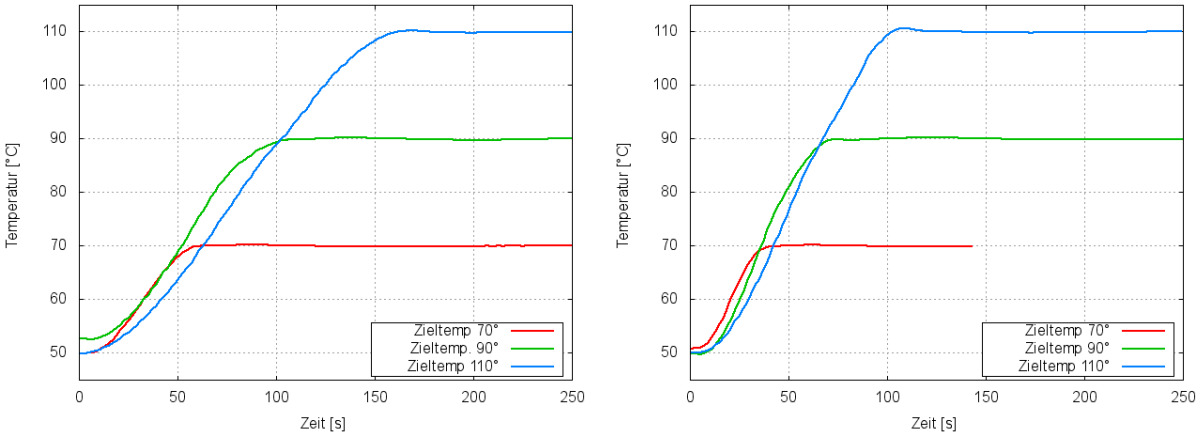
Abbildung 15 zeigt auch das durch den 2. Integrator bedingte langsame Ausregeln der Regelabweichung zu Beginn der Messkurven. Ab einer Temperaturdifferenz kleiner 10 °C wird die Regelung automatisch auf die in Abschnitt 5.1.1 gezeigte Regelung umgeschalten und auf die Zieltemperatur geregelt.

5.1.3 Regler mit zwei Freiheitsgraden

Die Regelung mit dem modellierten Regler wird in Abbildung 16 gezeigt, wobei die in Tabelle 5 gelisteten Parameter bestimmt und verwendet werden.

Zieltemp.°C	C_T	α	k_P	k_I
70	28.5	0.0557	0.04	0.001
90	29.5	0.0577	0.04	0.001
110	28.5	0.0657	0.04	0.001

Tabelle 5: Parameterset für Regelungen in Abbildung 16



(a) CME Regelung von Temperatursprüngen bei einer Rampe von $0.35\,\mathrm{K\,s^{-1}}$

(b) CME Regelung von Temperatursprüngen bei einer Rampe von $0.55\,\mathrm{K\,s^{-1}}$

Abbildung 16: Regelung der CME

Abbildung 16 zeigt einerseits die flexibel einstellbaren Änderungsrampen und ebenso die im Vergleich zu Abschnitt 5.1.1 deutlich kürzeren Regelzeiten. So gelingt es zum Beispiel die Regeldauer für eine Zieltemperatur von 110 °C deutlich zu reduzieren. Die Parameter müssen dabei allerdings so gewählt sein, dass nach der Rampenvorgabe im Regelzyklus, die Isttemperatur sehr nahe am Toleranzbereich für die Solltemperatur liegt und nur noch eine sehr kleine Regelabweichung vom Integrator kompensiert werden muss. Dies verdeutlicht noch einmal die limitierende Wirkung eines Integrators in Bezug auf die Dynamik des Regelkreises, neben der durch die Stellgrößenbeschränkung gegebenen Begrenzung. Abweichungen des Regelverhaltens zur Simulation können unter anderem durch Nichtberücksichtigung der zusätzlichen Verzögerung durch den Sensor im Rückkopplungszweig des Regelkreises, sowie jener die durch das Heizelement selbst bei der Erwärmung des Heizdrahtes entsteht, erklärt werden.

5.2 Ofen

Aufgrund der viel höheren Masse und somit auch thermischen Kapazität ergibt sich eine sehr viel höhere Zeitkonstante der Regelstrecke beim Ofen im Vergleich mit der CME.

5.2.1 PID Regler

Abbildung 17 zeigt die Regelkurven mit den in Tabelle 6 verwendeten Parametern. In allen Regelkurven ist ein zu groß gewählter Integralparameter ersichtlich. In Kombination mit

einem zu für den jeweiligen Integralparameter zu kleinen Differentialparameter führt dies zu einem deutlichen Überschwingen. Speziell beim zweiten Parameterset (grüne Charakteristik) ist auch ersichtlich, dass bei einem sehr hohen Integralanteil ein deutliches und nur sehr langsam abklingendes Schwingen beim Ausregeln der stationären Regelabweichung auftritt.

Parameterset	k_P	k_I	k_D
Set 1	3.0	0.01	0.0
Set 2	3.0	0.02	100.0
Set 3	3.0	0.005	300.0

Tabelle 6: Verwendete Parameter für Abbildung 17

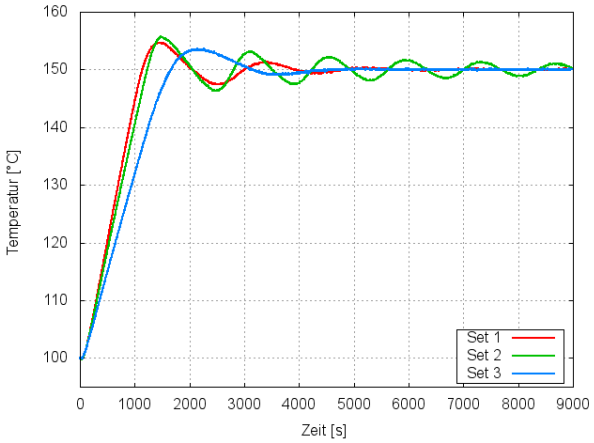


Abbildung 17: PID Regelung Ofen

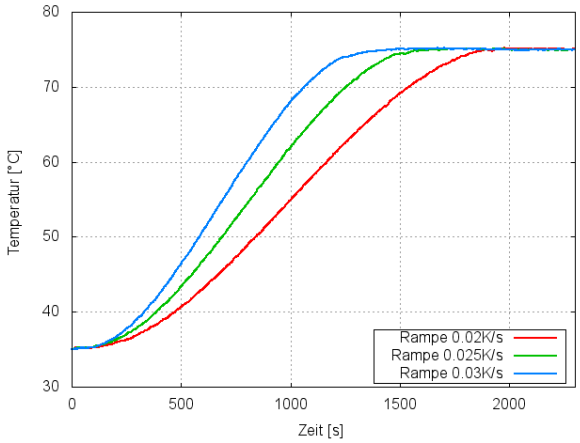
5.2.2 Regler mit zwei Freiheitsgraden

Die Regelung mit dem modellierten Regler wird in Abbildung 18 gezeigt, wobei die in Tabelle 7 gelisteten Parameter verwendet werden.

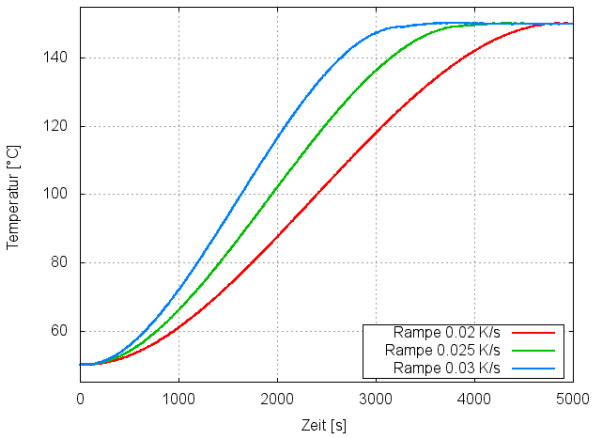
Zieltemp.°C	C_T	α	k_P	k_I
75	1185	0.100	0.04	0.001
150	1215	0.115	0.04	0.001
230	1240	0.122	0.04	0.001

Tabelle 7: Parameterset für Regelungen in Abbildung 18

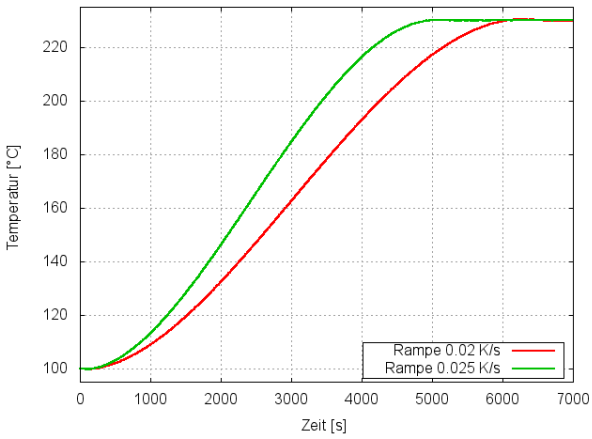
Zusätzlich wird bei der Regelung auf die Zieltemperatur von 230 °C in Abbildung 18(d) der Verlauf der Heizspannung gezeigt, wobei hier die Limitierung der maximalen Heizrampe durch die maximale Heizspannung deutlich wird (Stellgrößenbeschränkung).
Die Temperaturabhängigkeit der thermischen Kapazität C_T und des Wärmeübergangskoeffizienten α sind ebenfalls deutlich an den Parametern in Tabelle 7 zu sehen.



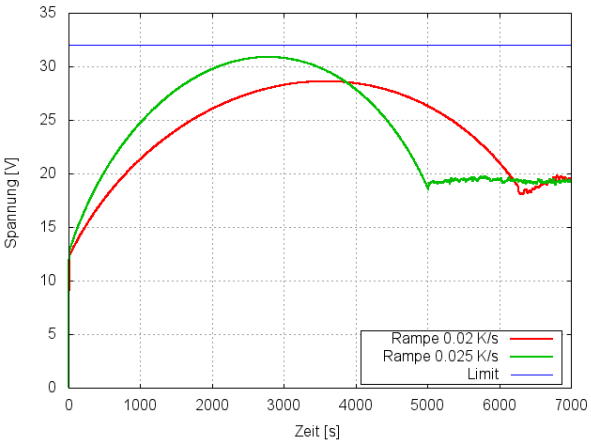
(a) Ofen Regelung auf Zieltemperatur 75 °C



(b) Ofen Regelung auf Zieltemperatur 150 °C



(c) Ofen Regelung auf Zieltemperatur 230 °C



(d) Heizspannung bei Ofen Regelung auf Zieltemperatur 230 °C

Abbildung 18: Regelung des Ofen

5.3 Vergleich der beiden Regler

Die in diesem Abschnitt gezeigten Messungen machen deutlich, dass der optimierte Regler mit 2 Freiheitsgraden, welcher eine genaue Kenntnis der Regelstrecke erfordert, Vorteile gegenüber dem PID - Regler hinsichtlich der Regelzeit aufweist. Zusätzlich erlaubt dieser Regler die Vorgabe einer Temperaturrampe, wodurch bei den Heizvorgängen jedem Zeitpunkt ein genauer Temperaturwert zugeordnet werden kann. Insbesondere lässt sich durch Vorgabe einer gezielten Temperaturrampe der Recoveryzyklus der TDDS beschleunigen, da Defekte mit langen Recoveryzeitkonstanten durch Erhöhen der Temperatur kürzere Emissionszeiten aufweisen. Die Messergebnisse können danach in der Simulation bei Vorgabe der gleichen Temperatur in jedem Zeitpunkt überprüft werden.
Der PID Regler hat den Vorteil für den Anwender, dass auch ohne Kenntnis der Regelstrecke durch empirische Einstellmethoden in kurzer Zeit ein gutes Regelverhalten erreicht werden kann und eignet sich dadurch für den Einsatz bei neuen Messaufbauten.

6 Ausblick

Die modulare Implementierung der Steuerung und Simulation des Temperaturcontrollers in Python erlaubt weitreichende Erweiterungen des Reglers. Als Beispiel sei hier eine grafische Eingabe und Ausgabe der Regelparameter und Messwerte erwähnt. Im Folgenden wird noch ein Ausblick auf mögliche Verbesserungen des bestehenden Reglers, sowie die Erhöhung der Regeldynamik und Erweiterung des Arbeitsbereiches auf niedrige Temperaturen gegeben.

6.1 Weiterentwicklung Regler für CME und Ofen

Die momentane Implementierung des Reglers erfordert die manuelle Angabe der Wärmekapazität und des Wärmeübergangskoeffizienten durch den Benutzer. Durch genaue Bestimmung der Temperaturabhängigkeit dieser Kenngrößen über den gesamten Arbeitsbereich können diese, wie im Fall des elektrischen Widerstandes (siehe Abbildung 10), auf eine Funktion gefittet und somit automatisch für das jeweilige Heizelement berechnet werden. Die ermittelten Parameter für verschieden Heizelemente könnten somit in einer Datenbank gespeichert werden, wodurch nur noch die Auswahl des Elementes vom Anwender durchgeführt werden muss.

6.2 Erweiterung des Messplatzes um aktive Kühlmethoden und Niedertemperaturmessungen

Da die bisherigen Heizelemente nur passiv (Wärmeenergie wird durch freie Konvektion an Umgebungsluft abgegeben) gekühlt werden können, was sich besonders im Fall des Ofens als sehr zeitintensiv erweist, erlaubt der Einsatz von Peltier Elementen durch Umkehr der Stromflussrichtung auch eine aktive Kühlung. Der thermoelektrische Peltier-Effekt beruht auf der Energieaufnahme bzw. -abgabe der Elektronen beim Durchfluss einer Kontaktstelle zweier unterschiedlicher Leiter an das Gitter, bedingt durch unterschiedliche mittlere Energien der Leiterwerkstoffe [4]. Herkömmliche Peltier Elemente bestehen aus sehr vielen solcher parallel angeordneten Kontaktstellen. Je nach Stromrichtung, in der diese Kontaktstellen durchflossen werden, wird eine Seite des Elementes gekühlt und die gegenüberliegende Seite erhitzt.
Abbildung 19 zeigt die unterschiedlichen Temperaturverläufe mit einer aktiven und passiven Kühlung eines typischen Peltier Elementes, wobei dieses zuvor gezielt mittels PID Regelung auf 70 °C erhitzt wurde und im Fall der passiven Kühlung die Heizspannung weggeschalten wird. Zur aktiven Kühlung des Elementes wird die Heizspannung umgepolt. Dies wird durch eine Modifikation des bestehenden Temperaturkontrollers mithilfe eines monostabilen Wechselrelais und einer entsprechenden Erweiterung der Software erreicht. Abbildung 19 zeigt auch deutlich die hohe Dynamik des Regelkreises bei Einsatz eines Peltier Elementes. Temperaturrampen mit Änderungsraten größer $10\,\mathrm{K\,s^{-1}}$ konnten bei einem ersten Testaufbau erreicht werden. Diese hohe Dynamik erfordert auch eine Anpassung der Hardware, da wie in Abschnitt 3.1 beschrieben die Abtastzeit durch den AD Wandler beschränkt ist, aber die Bedingung von Gleichung (6) erfüllt werden muss.

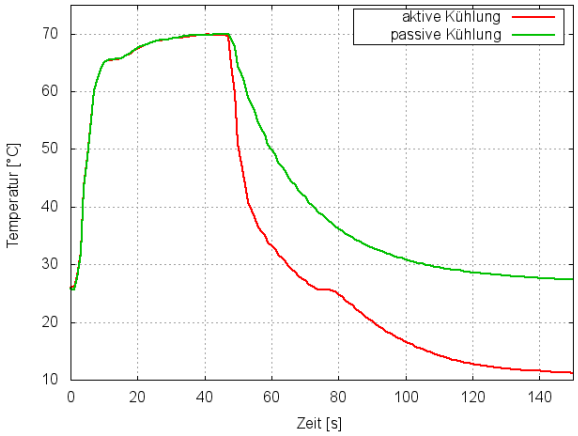


Abbildung 19: Vergleich aktive und passive Kühlung eines Peltier Elementes (der Knick der Messkurve bei der aktiven Kühlung zeigt ein unerwünschtes Schaltverhalten des Relais)

Der Temperaturkontroller kann in diesem Zusammenhang so erweitert werden, dass sowohl Hochtemperaturmessungen, als auch Niedertemperaturmessungen mit nur einem Heizelement durchgeführt werden können. Entsprechende Anpassungen der Software - Regelungen können dabei sehr einfach realisiert werden. Um eine effiziente Kühlwirkung des Elements zu erreichen, ist darauf zu achten, dass die an der warmen Seite des Peltier-Elementes erzeugte Wärmemenge effizient abtransportiert werden kann, da ansonsten die Wärmeenergie gemäß dem zweiten Hauptsatz der Thermodynamik zur kalten Seite diffundiert und es somit wiederum zu einer Erwärmung kommt. Entsprechende Maßnahmen zur Verhinderung dieses Effektes sind sowohl beim Hardware- als auch beim Softwareentwurf zu berücksichtigen.

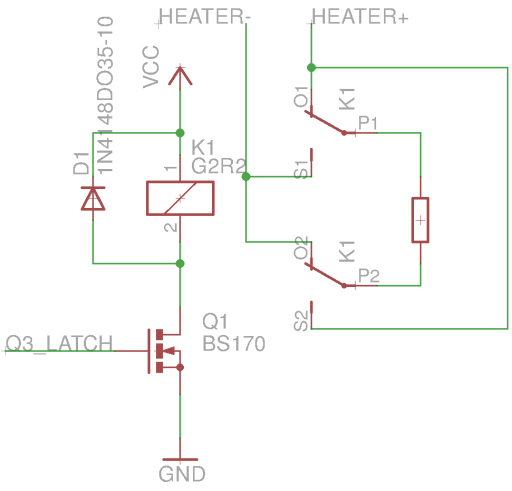


Abbildung 20: Adapterplatine mit monostabilem Wechselrelais für Umpolung der Heizspannung zum Ansteuern eines Peltier Elementes

A Anhang

Befehlsübersicht - neue Methoden der Klasse `controlFurnace` des Moduls `TDDSUnitFurnace_v2_0`

Methode	Parameter-Range	Beschreibung
`setTemperatureCME (self, T, Ramp)`	T: 20 - 400 Ramp: 0.001 - 0.5	Zieltemperatur wird auf T gesetzt. Gleichzeitig wird eine Rampe gemäß Ramp berechnet und Polynomkoeffizienten und Endzeit der Rampe im Controller gesetzt und eine neue Regelung startet (siehe Auszug).
`setAlpha (self, alph)`	alph: 0.03 - 0.2	Setzt den Wärmeübergangskoeffizienten auf alph
`setCap (self, cap)`	cap: 20 - 1500	Setzt die Wärmekapazität auf cap
`setIntemax (self, intmax)`	intmax: 0 - 100	Setzt das Anti-Windup Limit für den Integralteil des Reglers
`setOvenMode (self, i)`	i: 0 ... CME, 1 ... MiniOven, 2 ... Peltier	Setzt das aktuell verwendete Heizelement
`getOvenMode(self)`		Gibt das aktuell ausgewählte Heizelement zurück - Parameter wie bei setOvenMode
`setContMode(self, i)`	i: 0 ... PID, 1 ... Ramp, 2 ... CMER	Setzt den aktuellen Regler auf PID (Standard Regelung) auf Rampenregelung (Ramp) oder optimierte Regelung (CMER)

Codeauszug aus dem Modul `TDDSUnitFurnace_v2_0.py` für die Methode `setTemperatureCME`:

```
def setTemperatureCME(self, T, Ramp):
  self.setContMode(3)
  actT = self.getTemperature()
  g    = gen.Tpatt(50.,actT,T,Ramp,0.)
  par  = g.generate()
  if actT+5. < T:
        end = g.getEndTime()
  else:
        end = 0.
  # write parameters for ramp
  ret = self.writeCommand('SETRAMPPARAM0 %.12lf' %par[0])
  if ret.find('OK') >= 0:
    ret = self.writeCommand('SETRAMPPARAM1 %.12lf' %par[1])
    if ret.find('OK') >= 0:
      ret = self.writeCommand('SETRAMPPARAM2 %.12lf' %par[2])
      if ret.find('OK') >= 0:
        # write ramp end time
        ret = self.writeCommand('SETENDTIME %.2lf' %end)
        if ret.find('OK') >= 0:
          self.setTemperature(T)
          return 1
  else:
    common.printWarning('Furnace communication error - setTemp')
    return 0
```

Codeauszug aus dem Modul `TPatterngenerator.py`:

```
class Tpatt:
  def __init__(self, delay, startTemp, targetTemp, targetRamp,
               offset):
    self.t_temp   = targetTemp+offset
    self.s_temp   = startTemp
    self.t_ramp   = targetRamp
    self.delay    = delay
    self.end_time = (targetTemp+offset - startTemp)/targetRamp

  def generate(self):
    x = np.array([-self.delay, 0.0, -self.delay +
                  self.end_time/2.0, -self.delay +
                  self.end_time, self.end_time])
    y = np.array([self.s_temp, self.s_temp , self.s_temp +
                  (self.t_temp - self.s_temp)/2.0 ,
                   self.t_temp , self.t_temp])
    z = np.polyfit(x, y, 3)
    funcPoly = lambda params, x: params[0]*x**3 + params[1]*x**2 +
```

```
                                    params[2]*x + params[3]
    funcderPoly = lambda params, x: params[0]*x**2 + params[1]*x +
                                        params[2]
    return np.polyder(z)
```

Auszug - wichtigste Befehle Makefile zum Programmieren des Mikrokontrollers mit AVR - Dude und JTAG-ICE3:

```
# MCU name for programmer
MCU2 = m128

# Output format (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension)
TARGET = oven

# List C source files here (C dependencies are autom. generated)
SRC  = $(TARGET).c
SRC += display.c
SRC += utilities.c
SRC += keyboard.c

AVRDUDE_PROGRAMMER   = jtag3
AVRDUDE_WRITE_FLASH  = -U flash:w:$(TARGET).hex
AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep
AVRDUDE_FLAGS        = -p $(MCU2) -c $(AVRDUDE_PROGRAMMER)
```

Codeauszug optimierter Regler aus der Datei `oven.c`:

```
case CMER:
  cerr[1]    = cerr[0];
  cerr[0]    = oven_target - oven_temp;
  oven_ramp = (cerr[0] - cerr[1]) / ADC_RATE;

  kp = eeprom_read_float(&cntr_pro);
  ki = eeprom_read_float(&cntr_int);
  kd = eeprom_read_float(&cntr_dif);

  if(k > end) // limit the oven_target polynom to rise time (100s)
  {
    oven_target_diff = 0.0;
  }
  else
  {
    oven_target_diff = target_ramp(k);
    power_int = 0.0;
    power_pro = 0.0;
  }
  k = k + ADC_RATE;
  if(power < oven_limit && k > end) //anti wind up + start
                                    //PI Controll only after ramping
  {
    power_int = ki * ADC_RATE * cerr[0] + power_int;
    power_int = limit(power_int,0.,0.01);
    power_pro = kp * cerr[0];
  }
  u_k = res_CME(oven_temp) * c_heat * (oven_target_diff +
        alp_h / c_heat * (oven_temp - t_ambient) + power_pro +
        power_int);

  if(u_k > 0) //prevent negative - argument for square root
  {
    power = limit(sqrt(u_k),0.,100.);
  }
  else
  {
    power = 0;
  }
  set_heater(power/0.5+1.0); // measured a 1% offset for voltage
                             // on adc - 0.5 V / % of max vol-change

  break;
```

Codeauszug Simulator aus dem Modul `Simulate.py`:

```
class Simul:
  def __init__(self,cap_reg,cap_s,alph_reg,alph_s,t_amb):
    self.cap_r = cap_reg
    self.cap_s = cap_s
    self.alp_r = alph_reg
    self.alp_s = alph_s
    self.t_amb = t_amb
    self.T_a   = 0.8

  def simulate(self,t_start,t_target,t_ramp,k_end,k_p,k_i,t_ambact):

    T      = []
    u      = []
    targl  = []
    targu  = []
    pro    = 0.
    inte   = 0.
    intema = self.getInteMax(1.0,t_target)

    g    = gen.Tpatt(10.0,t_start,t_target,t_ramp,0.)
    para = g.generate()

    T.append(t_start)
    T.append(t_start)
    u.append(0.)

    for k in range(1,int(k_end/self.T_a)):
      if(k < g.getEndTime()/self.T_a):
        tcurve = (self.getTsoll(x[k],para))
      else:
        tcurve = 0.

      if (self.getResCME(T[k])* self.cap_r * ( tcurve + self.alp_r /
          self.cap_r * (T[k] - self.t_amb) + pro + inte)) > 0:
        u.append(self.getCont(T[k], tcurve, inte, pro,
                 self.getResCME(T[k])))
      else:
        u.append(0.)

      T.append((T[k] - t_ambact)*m.exp(-self.alp_s/self.cap_s*
               self.T_a)+(1.-m.exp(-self.alp_s/
               self.cap_s*self.T_a))*(u[k]**2)/
               (self.getResCME(T[k])*self.alp_s)+t_ambact)

      if(k > g.getEndTime()/self.T_a):
```

```
        if (inte > intema):
          inte = intema
        if (inte < -intema):
          inte = -intema
        else:
          inte = inte + self.getPIDVal(k_i,t_target,T[k+1])
        pro = self.getPIDVal(k_p,t_target,T[k+1])
```

Literatur

[1] Thomasl Beier and Petra Wurl. *Regelungstechnik*. Hanser, 2016.

[2] Emmerich Bertagnolli. *Elektronische Bauelemente*. 2015.

[3] G. Doblinger. *Zeitdiskrete Signale und Systeme: eine Einführung in die grundlegenden Methoden der digitalen Signalverarbeitung*. Schlembach, 2010.

[4] Gerhard Fasching. *Werkstoffe für die Elektrotechnik: Mikrophysik, Struktur, Eigenschaften*. Springer-Verlag, 2005.

[5] T. Grasser. Stochastic Charge Trapping in Oxides: From Random Telegraph Noise to Bias Temperature Instabilities. *Microelectronics Reliability*, 52(1), 2012.

[6] T. Grasser, H. Reisinger, P.-J. Wagner, F. Schanovsky, W. Goes, and B. Kaczer. The Time Dependent Defect Spectroscopy TDDS for the Characterization of the Bias Temperature Instability. *Proceedings of the International Reliability Physics Symposium (IRPS)*, pages 16–25, May 2010.

[7] Verein Deutscher Ingenieure. *VDI-Wärmeatlas*. Heidelberg:Springer, 2006.

[8] B. Kaczer. Recent Trends in Bias Temperature Instability. *Journal of Vacuum Science & Technology B*, 29(1), 2011.

[9] Andreas Kugi. *Automatisierung Vorlesung und Übung*. 2016.

[10] Andreas Kugi. *Modellbildung Vorlesung und Übung*. 2016.

[11] E. Schrüfer. *Elektrische Messtechnik: Messung elektrischer und nichtelektrischer Größen*. Hanser, 2014.

[12] Gottfried Strasser. *Halbleiterphysik*. 2014.