Facharbeit in 12/2

Regelungstechnik

Versuche zum Einsatz eines digitalen Reglers für den Positionsregelkreis einer Kraftkompensationswaage

Verfasst von Lars Gottfriedsen

Verfasser:	Lars Gottfriedsen
Schule:	Hohenstaufen-Gymnasium Kaiserslautern
betreuende Lehrkraft:	Dr. Michael Savorić
Unterrichtsfach:	Informatik
Kursbezeichnung:	Inf_LK_1
Bearbeitungszeitraum:	04.02. - 30.05.2022

Inhaltsverzeichnis

1 Einleitung

Da ich schon länger ein gesteigertes Interesse für die MINT-Fächer, insbesondere für Informatik pflege, soll diese Facharbeit die Möglichkeit eines praktischen Projekts geben.
Konkret soll ein digitaler Regler für Wägezellen der Firma Wipotec entwickelt werden. Dabei liegt das Ziel weniger darin, den Regler in der Form, in der er später produziert wird, zu entwickeln, sondern viel mehr darin, als „Proof of Concept“ zu erörtern, ob für diese extrem schnelle Anwendung mithilfe von Digitaltechnik eine der bereits bestehenden analogen Regelung ebenbürtige Regelung zu realisieren ist.
Mit dem Unternehmen Wipotec sowie deren Wägetechnik bin ich bereits durch mein verpflichtendes Berufspraktikum in der 9. Klasse sowie durch meinen dort angestellten Vater Jan Gottfriedsen, der auch diese Facharbeit vermittelt hat, in Kontakt gekommen.

2 Grundlagen der Regeltechnik

Die Aufgabe eines Reglers ist es, die zu regelnde Größe x auf einen vorgeschriebenen Führungsgröße w (Sollwert) zu bringen und dort zu halten. Dazu wird die Regelgröße fortlaufend gemessen, als Rückführgröße r weitergegeben, die Regeldifferenz $e = r - w$ ermittelt und basierend darauf die Stellgröße y so verstellt, dass sie durch das Stellglied auf die Regelstrecke, also den zu regelnden Vorgang so einwirkt, dass sich die Regelgröße der Führungsgröße annähert. Neben der vom Regler kontrollierbaren Stellgröße wirken auch nicht kontrollierbare und meist auch nicht bekannte Störgrößen auf die Regelstrecke und damit auf die Regelgröße ein.[1]

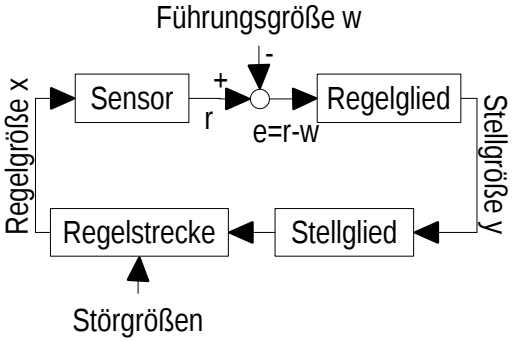

Abb. 1: Blockschaltbild eines Regelkreises

Ein Beispiel für einen Regler ist ein Raumthermostat. Die Regelstrecke wäre hier das Gebäude, die Regelgröße die Raumtemperatur, auf die einerseits Störgrößen, wie offene Fenster und die Außentemperatur aber auch die Heizung, die hier das Stellglied ist, einwirken. Die Regeleinrichtung kann zum Beispiel ein mechanischer Thermostat sein. Sie erfasst die Raumtemperatur über einen Sensor, dessen Ausgangssignal die Rückführgröße bildet, vergleicht sie mit der vom Nutzer vorgegebenen Führungsgröße und bildet abhängig von der Differenz die Stellgröße, die als Steuersignal der Heizung dient.
Da die beeinflusste Regelgröße erneut gemessen und als Grundlage der Regelung verwendet wird, bildet sich ein Kreislauf, der als geschlossener Wirkungsablauf bezeichnet wird. Das unter-

1 Stiny 2013, 9ff.

scheidet die Regelung von der Steuerung, die die zu regelnde Größe nicht erfasst, sondern die Stellgröße nur auf Basis einer Steuerungsvorschrift und ggf. auf Basis der Störgrößen einstellt. Das nennt man einen offenen Wirkungsablauf.[2]

2.1 Digitale Regler

Digitale Regler arbeiten zeit- und wertdiskret. Das bedeutet, dass Signale nicht kontinuierlich (stetig) sondern nur in diskreten Intervallen erfasst, verarbeitet und ausgegeben werden, wobei die Zykluszeit als Abtastzeit T_A bezeichnet wird. Dabei sind digitale Signale stets Zahlenwörter, die nur endlich viele, bestimmte Werte annehmen können.[3]

Damit analoge Signale in einen Mikrocontroller, der dann die eigentliche Regelung vornimmt, gelangen können, werden sie durch einen Analog-Digital-Umsetzer (ADC) in digitale Signale umgewandelt. Diesen Prozess nennt man auch Quantisierung. Um das Reglergebnis wieder in ein analoges Signal umwandeln zu können, wird ein Digital-Anaolg-Wandler (DAC) benötigt.[4]

Im folgenden sollen Vor- und Nachteile digitaler Regelungstechnik gegenüber analoger Regelungstechnik dargestellt werden:

- Obwohl die Preise von Mikrocontrollern in den letzten Jahren drastisch gesunken sind, bieten analoge Regler insbesondere bei einfachen Reglern die deutlich preiswertere Lösung.
- Durch die Zeitverzögerung für Berechnungen und Abtastzeiten des digitalen Reglers haben analoge Regler außerdem bei extrem schnellen Regelstrecken, wie der hier behandelten, klare Vorteile.
- Vorteile digitaler Regeltechnik liegen dabei unter anderem in geringeren Bauteiltoleranzen und mathematisch eindeutigen Regelvorschriften.
- Reglerstruktur und -eigenschaften sind nicht konstruktionsbedingt vorgegeben, was höhere Flexibilität sowie komplexere Regler ermöglicht.
- Das digitale Vorliegen der Daten vereinfacht dabei die im Rahmen von Industrie 4.0 geforderte Kommunikation mit anderen Teilen des Prozesses oder zur Fernwartung. Eine komplette Integration des Reglers in einen anderen Mikroprozessor ist außerdem möglich.

2 vgl. Stiny 2013, 7f.

3 vgl. Stiny 2013, 85

4 vgl. Stiny 2013, 45

- Schließlich hat ein digitaler Regler unter bestimmten Voraussetzungen die Möglichkeit, seine Regelparameter selbst einzustellen oder zu optimieren, bzw. sich selbst zu reparieren oder zu diagnostizieren.[5]

2.2 PID-Regler

Ein PID-Regler ist eine typische Regelarchitektur, bei der sich der Stellwert aus einem proportionalem, einem integralen und einem differentialen Anteil zusammensetzt. Durch eine Kombination dieser Anteile mit unterschiedlichem Zeitverhalten und ihrer Gewichtung über Regelparameter kann ein weites Feld an Regelverhalten erzielt werden. Anschaulich gesprochen bekämpft der P-Anteil dabei die aktuelle Regelabweichung, der I-Anteil gleicht die verbleibende Regeldifferenz auf Basis vergangener Abweichungen aus und der D-Anteil versucht zukünftige Regelabweichungen vorherzusagen und zu bekämpfen, wobei die Änderungsrate der Regeldifferenz genutzt wird.
In der Literatur findet man für zeitkontinuierliche Systeme folgende Regelvorschrift mit Übertragungsbeiwert K_S, Nachstellzeit T_N und Vorhaltezeit T_V als Regelparametern[6]:

$$y(t)=K_S \cdot [e(t) + \frac{1}{T_N}\cdot\int e(t)dt + T_V\cdot e'(t)]$$

Implementiert wurde die Vorschrift folgendermaßen:

$$y(n)=Offset+Faktor_P\cdot e(n) + Faktor_I\cdot\sum_{i=0}^{n}[e(i)] + Faktor_D\cdot[z(n)-z(n-1)]$$

Dabei wurden aus der kontinuierlichen Zeit t diskrete Abtastzyklen n . Die verschachtelten Faktoren und Divisoren wurden als einzelne Faktoren dargestellt um die Berechnung zu vereinfachen. Es wurde mit $z(n)=k\cdot e(n)+[1-k]\cdot z(n-1)$ das Signal für den Differenzialanteil durch einen Tiefpass gefiltert, wobei k als Filterparameter die Grenzfrequenz des Filters bestimmt. Dabei liegt k zwischen 0 und 1, wobei kleine Werte einer starken Filterung entsprechen. Ziel war es, die Einflüsse von Umgebungsrauschen zu verringern. Bei der Optimierung der Parameter hatte der Tiefpass aber keine signifikant positiven Einflüsse.[7]
Der Offset soll neben der Möglichkeit Störungen zu Testzwecken einzubringen im Regelbetrieb den I-Anteil kleiner halten, um dort einen Zahlenüberlauf zu verhindern. Er kann im Regelbetrieb beim Vorlastabgleich festgelegt werden.

5 vgl. Stiny 2013, 86f.

6 vgl. Stiny 2013, 123ff.

7 vgl. Anhang, 21

3 Funktion einer Kraftkompensationswaage

Die hier thematisierten Wägezellen arbeiten nach dem Prinzip der elektrodynamischen Kraftkompensation (EDK). Dabei führt die aufgelegte Masse über die Gewichtskraft F_G zu einer Bewegung eines Hebels. Diese Kraft wird durch die Lorentzkraft F_L einer in das Feld eines Permanentmagneten eingeführten Spule kompensiert, sodass sich der Hebel an einer festen Position in der Schwebe befindet. Der durch die Spule fließende Strom kann als Maß für die gewogene Masse erfasst werden.[8]

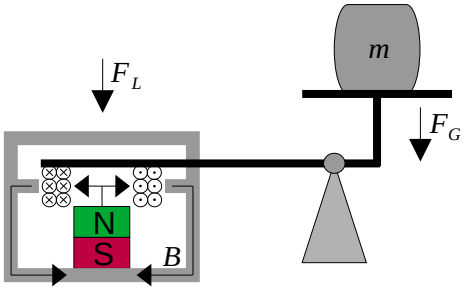

Abb 2: Skizze einer Kraftkompensationswaage

3.1 Positionsregelkreis

In dem hier behandelten Positionsregelkreis ist die Regelgröße also die Position des Hebels, die durch einen Sensor erfasst und als dessen Ausgangsspannung U_{pos} als Rückführgröße weiterverarbeitet wird. Störungsgrößen sind hier primär auf die Wägezelle aufgelegte Gewichte, aber auch Umgebungsvibrationen oder Wind. Die Aufgabe des Reglers ist es nun, mit U_{pos} als Eingangsgröße, diese der fest vorgegebenen Führungsgröße 0V anzunähern und sie dort zu halten. Dazu steht ihm als Stellgröße U_{Stell} zur Beeinflussung des Spulenstroms zur Verfügung.

4 Realisierung der Reglers

4.1 Elektrischer Aufbau

Kern des Aufbaus bildet ein Teensy 3.6, ein Entwicklerboard, dessen Mikrocontroller 16-bit ADCs und 12-bit DACs bereitstellt, über eine 32-bit Architektur verfügt und in die Arduino IDE auch mit den komplexeren Funktionen der ADCs integriert ist.

Die ADCs können bei voller Auflösung theoretisch mit Abtastfrequenzen bis zu 1,1MHz betrieben werden und erfüllen ihre Spezifikationen bis 278kHz, werden hier aber nur mit 37,5kHz betrieben, um Umwandlungsfehler geringer zu halten. Die DACs können mit 1,8MHz betrieben werden, stellen also keine Limitierung dar.

Zur Anbindung an die Wägezelle bzw. deren Hauptplatine waren für AGND, das Signal des Positionssensors U_{Pos} , sowie für die Manipulation der Spulenspannung U_{Stell} bereits Stecker zu Testzwecken vorhanden. In der Wägezelle war bereits eine bipolare Spannungsversorgung (±12V) vorhanden, an die der Regler angeschlossen wurde. Zum Deaktivieren des analogen Reglers

8 vgl. Wipotec 2022, Wägeprinzip

nach Bedarf wurde die Verbindung von analogem Regler und Verstärker getrennt.[9]

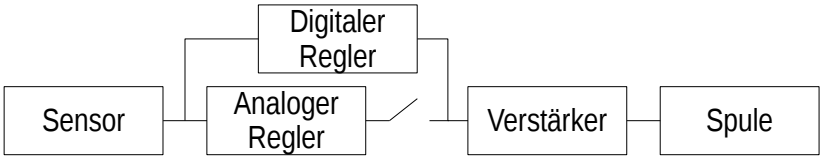

Abb 3: Blockschaltbild Anbindung an Hauptplatine

Außerdem wurden Operationsverstärkerschaltungen aufgebaut, die die Spannungsbereiche an den Ein- und Ausgängen der Mikrocontrollers in die für die Wägezelle benötigten umwandelt. Am Eingang wurden hier drei Schaltungen mit unterschiedlichem Messbereich aufgebaut, zwischen denen gewechselt werden kann.[10] Mehr dazu in Quantisierungsfehler.

Diese Schaltung wurde zunächst auf einem Steckbrett aufgebaut und später auf eine dafür hergestellte Platine gelötet.

Für unabhängige Messungen stand darüber hinaus ein Oszilloskop sowie betriebseigene Software zur Aufzeichnung des Wägergebnisses zur Verfügung.

Abb 4: Versuchsaufbau

4.2 Softwareseitige Realisierung

Die Software auf dem Mikrocontroller wurde in der Arduino-Programmiersprache mit Hilfe der ADC-Library geschrieben, während Software, die auf dem PC lief und mit dem Teensy über USB kommunizierte um komplexere Messungen anzuregen und grafisch darzustellen, in Python geschrieben wurde.

Das Programm ist in zwei grundlegende Ebenen aufgeteilt: Die Interrupt-Routine wird immer genau dann ausgeführt, wenn der dauerhaft arbeitende ADC eine Umwandlung beendet hat. Da-

9 Jan Gottfriedsen, persönliche Kommunikation, Februar 2022

10 vgl. Anhang, 23

durch wird sie in exakt gleichbleibenden Zeitintervallen ausgeführt, sodass zeitabhängige Funktionen (z.B. I- und D-Anteil der Regelung) nicht verzerrt werden. Damit das nicht nur für die Abtastung sondern auch für die Ausgabe gilt, findet diese immer zu Beginn der Interrupt-Routine statt. Außerdem ist sicherzustellen, dass eine Interrupt-Routine nie länger dauert, als die Dauer zwischen zwei Abtastungen. In ihr werden zeitkritische Berechnungen zur Regelung durchgeführt. Die zweite Ebene bildet die Hauptroutine, die stets läuft, wenn die Interrupt-Routine nicht aktiv ist. Sie übernimmt zeitlich unkritische Aufgaben wie die Kommunikation mit dem Nutzer sowie das Aktivieren von Sonderfunktionen. Hier ist zu beachten, dass, da die Hauptroutine stets von einem Interrupt unterbrochen werden kann, sobald der ADC fertig ist, keine Schreibvorgänge auf gemeinsam genutzten Variablen stattfinden, die unterbrochen werden könnten.[11]
Vorsicht ist außerdem geboten bei Datentypen, so werden Variablen, in denen Abweichungen aufaddiert werden (z.B. I-Anteil) schnell sehr groß, sodass es zu einem Zahlenüberlauf kommen kann, der durch Fallunterscheidungen zu unterbinden ist. An anderen Stellen können Zwischenergebnisse, die als Ganzzahlen definiert sind, viel zu klein werden und es treten starke Rundungsfehler auf, weshalb Werte teilweise mit festen Vorfaktoren gespeichert werden.

4.3 Bewertung des Regelergebnisses

Zur Bewertung der Regelgüte stehen mehrere Kriterien zur Verfügung:

- Eine bleibende Regeldifferenz, wie sie bei Proportionalreglern vorkommt, sollten bei einem PID-Regler eigentlich nicht vorkommen. Da in der Praxis aber sowohl der Stellwert, als auch der I-Anteil (Datentypen) begrenzt wird, ist er nicht gänzlich auszuschließen. Sie ist, sollte sie nicht unabhängig vom Gewicht fest sein, ein Ausschlusskriterium für den Regler.
- Ein dauerhaftes Rauschen ist speziell bei dieser Anwendung aufgrund von Quantisierungsfehlern möglich und kann als Standardabweichung erfasst werden. Auch ein dauerhaftes Schwingen aufgrund instabiler Regelparameter wird so erkannt.
- Das Folgeverhalten auf eine Änderung der Führungsgröße ist zwar grundsätzlich gut zu messen und wird daher gerne verwendet. Insofern bei dieser Anwendung die Führungsgröße aber konstant bleiben soll und auch ein Vergleich mit dem analogen Regler nicht ohne weiteres möglich ist, kommt es hier nicht zur Anwendung.
- Das Folgeverhalten auf eine (sprunghafte) Änderung einer Störgröße dagegen bietet sich als Kriterium an. Eine leicht steuerbare Störgröße bietet insbesondere eine von der Regeleinrichtung unabhängige Änderung des Spulenstroms.

11 vgl. Schumacher 2022, 228f.

- Die Überschwingweite $max(|e(t)|)$ beschreibt die maximale Regelabweichung und kommt bevorzugt bei Regelstrecken zum Einsatz, bei denen eine hohe Abweichung Beschädigungen am System hervorrufen kann. In dem hier betrachten System ist dies jedoch nicht der Fall.
- Die Einregelzeit beschreibt die Zeit, bis zu der Störung vollständig ausgeregelt ist. Problematisch ist aber die Festlegung eines Schwellenwerts, ab dem die Regelabweichung als vernachlässigbar gilt.
- Die Betragslineare Regelfläche $\int |e(t)|dt$ bietet ein Kriterium, dass die Dauer und das Ausmaß der Regelabweichung ohne das Festlegen einer Schwelle berücksichtigt, obwohl natürlich festgelegt werden muss, über welches Intervall gemessen wird.
- Einen stärkeren Anreiz, die Regelabweichung frühzeitig zu verringern bietet die Zeitbeschwerte betragslineare Regelfläche $\int [t \cdot |e(t)|]dt$, die den Zeitpunkt der Regelabweichung mit berücksichtigt und deshalb hier verwendet wurde.

4.4 Selbsteinstellung

Die Selbsteinstellung wurde aufgeteilt in eine Phase, in der zunächst Eigenschaften der Wägezelle bzw. der Regelstrecke ermittelt werden und aus diesen dann mithilfe von Faustformeln Regelparameter hergeleitet werden, sowie einer zweiten Phase, in der die Regelparameter ausgehend davon verändert werden und die aus der Veränderung folgende Regelgüte beurteilt wird, um eine optimale Einstellung zu finden.

Ein klassischer Streckenparameter ist die statische Kennlinie. Diese wird in einem offenen Regelkreis ermittelt, die Stellgröße wird also nicht von der Regelvorschrift bestimmt, sondern von Hand (bzw. dem Skript für die Kennlinie). Dazu wird jeweils eine feste Stellgröße eingestellt, gewartet, bis sich eine fester Wert bei der Regelgröße einstellt und dieser dann abgelesen. Hier erkennt man, dass diese in dem Bereich, in dem sie sich normalerweise bewegt, also der Schwebeposition zwischen den mechanischen Anschlägen, einer Gerade ähnelt. Man bezeichnet eine solche Regelstrecke als linear. Das bedeutet, dass einer Verdopplung der Eingangsgröße stets eine Verdopplung der Ausgangsgröße folgt. Man sieht aber auch, dass der Stellwertbereich, der den Hebel zwischen den mechanischen Anschlägen schweben lässt, weniger als 50 DAC-Punkte umfasst. Für den Messbereich von Eingangs-

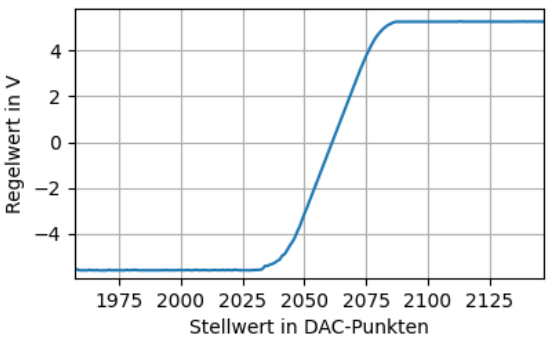

Abb 5: Statische Kennlinie

schaltung 3 sind es sogar nur ca. 3 DAC-Punkten. Darüber hinaus wird die Steigung der Kennlinie als Übertragungsbeiwert $K_S = \frac{\Delta x}{\Delta y}$ für manche Faustformeln benötigt. Außerdem liefert diese Kennlinie Punkte, in denen sich der Hebel in der Schwebe befindet, die für weitere Messungen benötigt werden, sowie Punkte, die dafür genutzt werden können, zwei Messbereiche zu justieren.

Um in dem relevanten, linearen Teil der Kennlinie für eine optimale Genauigkeit Einschwingzeiten von 2s pro DAC-Schritt zu realisieren, wären beim Abmessen des gesamten Stellwertbereichs $2\,s \cdot 2^{12} \approx 2{,}3\,h$ nötig, weshalb die Messung in mehreren Durchläufen vorgenommen wird, wobei anfängliche Durchläufe mit niedriger Genauigkeit genutzt werden, um den betrachteten Stellwertbereich einzugrenzen, was Zeiten von ca. 50s möglich macht.

Die zweite klassische Kennlinie ist die dynamische Kennlinie. Dabei wird im offenen Regelkreis die Regelgröße im Zeitverlauf nach einer sprunghaften Änderung der Stellgröße aufgezeichnet. Da beim Stellgrößensprung direkt eine Reaktion erkennbar ist, handelt es sich um eine Regelstrecke ohne Totzeit, der Verlauf der Sprungantwort deutet außerdem auf eine Regelstrecke erster Ordnung, also eine Regelstrecke mit einem Energiespeicher, hin. Das ist insofern verwunderlich, da mit Hebelmasse und der Federstärke der Gelenke des Hebels zwei Energiespeicher vorliegen sollten, aber vermutlich ist einer der Einflüsse vernachlässigbar. Eine solche Regelstrecke mit geringer Verzögerung gilt zwar als leicht zu regeln, die meisten Einstellregeln lassen sich aber hier nicht anwenden.

Um die Form der dynamischen Kennlinie weiter zu charakterisieren, stehen einerseits diverse Verfahren zur Verfügung, die auf der Bestimmung von (Wende-) Tangenten beruhen, was aber größeren Aufwand bei der Filterung der Daten benötigt hätte, weshalb beim automatisierten Einstellen zunächst auf die Auswertung der Sprungantwort nach Strejc und die Reglereinstellung nach Oppelt zurückgegriffen wurde, die als Parameter für ihre Faustformel die Zeit, die die Regelstrecke benötigt, um die Regelgröße bei einem Stellgrößensprung zu 10, 20, 80 und 90% anzupassen, nutzen.[12] Es stellte sich jedoch heraus, dass diese Einstellregeln keine zuverlässig stabilen Regelparameter liefern, was neben möglichen Fehlern bei der Implementierung daran liegen kann, dass diese Faustformel nicht für Regelstrecken erster Ordnung geeignet ist, was zwar nicht

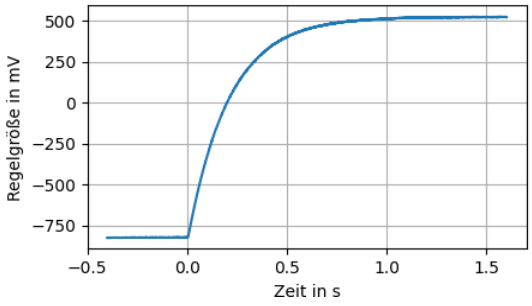

Abb 6: Dynamische Kennlinie

12 vgl. Stiny 2013, 150ff.

explizit genannt wird, da aber die damit ermittelten Streckenparameter T_u und T_g oft auch über Wendepunkte, die hier nicht existieren, bestimmt werden, nahe liegt.

Vereinfacht man die Regelstrecke zu einer Strecke erster Ordnung, wäre auch eine Simulation insbesondere bei der Selbsteinstellung möglich gewesen, es wurde jedoch darauf verzichtet, da diese Quantisierungsfehler unter Berücksichtigung des Rauschens des AD-Wandlers und Umgebungsvibrationen nicht wiedergegeben hätte, sodass der Aufbau von Messungen an der realen Wägezelle nötig war.

Das letztendlich verwendete Einstellverfahren ist die Schwingungsmethode nach Ziegler und Nichols, die darauf basiert, bei einem reinem Proportionalregler den P-Faktor so lange zu erhöhen, bis der Regelkreis instabil wird, der Hebel also ungedämpft schwingt und dann P-Faktor und Frequenz in eine Faustformel einzusetzen.[13]

Nun stellte sich die Frage, wie man automatisiert feststellen kann, dass eine ungedämpfte Schwingung vorliegt oder anders formuliert: wie man das normale Zappeln und Rauschen von einer sauberen Schwingung unterscheidet. Untersucht wurden hierfür drei Kriterien: die Standardabweichung der Regelgröße als Maß für die Amplitude, die Schwingungsdauer, gemessen über die Abstände zweier Nullstellen, wobei Abstände unter $0{,}27\,ms = 10 \cdot T_A$ ignoriert werden und als drittes Kriterium die Regelmäßigkeit der Schwingungsdauer, gemessen als Standardabweichung der Abstände zweier Nullstellen. Dabei wurde als Bezugswert jeweils nicht der Sollwert sondern ein Mittelwert der Regelgröße gewählt, da der hier verwendete P-Regler eine bleibende Regelabweichung besitzt. Durch das vorherige Einstellen des Offset wurde die Abweichung außerdem gering gehalten. Als am geeignetsten hat sich die Standardabweichung erwiesen, wobei auch die Schwingungsdauer als Kriterium in Frage gekommen wäre.

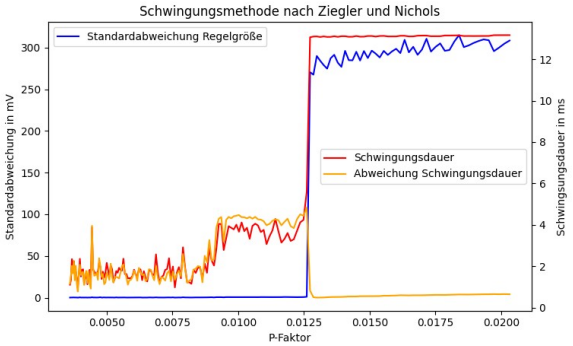

Abb 7: Auswertung von Kriterien zur Schwingungserkennung

Nachdem jetzt erste stabile Regelparameter zustande gekommen sind, die jedoch noch keine hohe Regelgüte aufweisen, werden diese davon ausgehend weiter optimiert. Dazu sind einerseits Kriterien festzulegen, welches das optimale Regelergebnis ist und andererseits ein Verfahren festzulegen, nach dem die Regelparameter verändert werden.

Die theoretisch beste Variante wäre, alle vier Regelparameter (Faktoren von P-, I- und D-Anteil und Parameter für Tiefpass bei D-Anteil) unabhängig voneinander in möglichst vielen Schritten zu verändern. Das führt aber bereits bei je 10 Schritten pro Parameter und 1,5s pro Durchgang zu

13 vgl. Stiny 2013, 146f.

$1{,}5\,s \cdot 10^4 \approx 4{,}2\,h$ Dauer, ist also also Optimierungsverfahren eher ungeeignet, da in einem Großteil der Fälle instabile Regelparameter getestet werden würden und das davon hervorgerufene Schwingen neben einer Lärmbelästigung in diesem Umfang auch eine Belastung der Wägezelle darstellt. Dennoch wäre es sinnvoll, dies einmalig durchzuführen, um festzustellen, ob es lokale Optima bei den Regelparametern gibt, bei denen andere Optimierungsalgorithmen fälschlicherweise stoppen könnten.

Die aus Zeitgründen gewählte Variante passt in jedem Schritt nacheinander die Regelparameter in jede Richtung solange an, wie sich die Regelgüte verbessert. Es zeigt sich, dass nach ca. 30 Schritten das Optimum erreicht wurde und sich feste Regelparameter einstellen, was inklusive der Schritte davor ca. 20 Minuten entspricht.

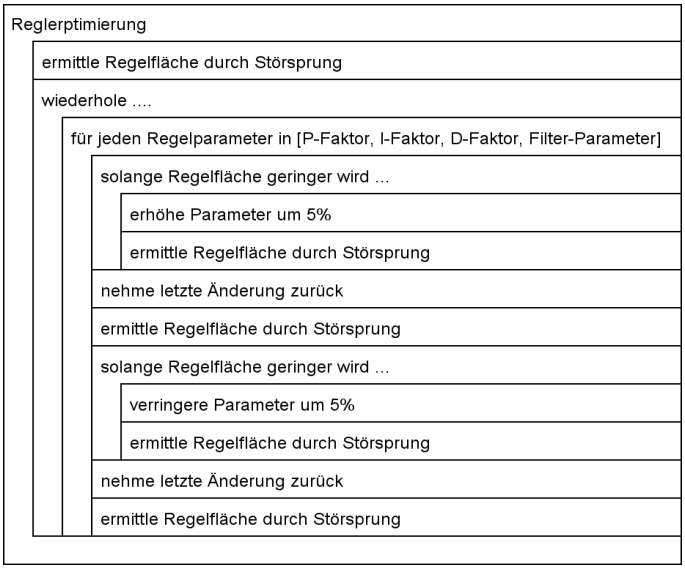

Abb 8: Struktogramm Optimierungsalgorithmus

Dabei ist insbesondere hervorzuheben, dass hierfür keine Arbeitskräfte gebunden werden, sondern die Wägezelle dies alleine und grundsätzlich auch in einer Produktionspause an ihrem Einsatzort machen kann, sodass auch längere Zeiten unkritisch wären.

Bei mehreren Selbsteinstellungsdurchgängen fiel auf, dass die meisten Durchgänge ähnliche Parameter und Regelgüte liefern, einzelne Durchgänge jedoch schlechtere Ergebnisse liefern, trotzdem aber zu bestimmten Regelparametern konvergieren.[14] Dass diese nicht die Parameter der anderen Durchgänge mit besserer Regelgüte annehmen, lässt sich nur durch die Existenz lokaler Optima erklären. Also Parameter, die zwar nicht global optimal sind, bei denen aber jede einzelne Veränderung eine Verschlechterung darstellt. Daher sind andere Optimierungsalgorithmen, die mehr Veränderungen mit Ergebnissen zwischenspeichern, bevor sie sich für eine entscheiden, in Betracht zu ziehen.

Sollten auch bei anderen Wägezellen die optimalen Regelparameter oberhalb der durch die Faustformel generierten Parameter liegen, wäre eine Anpassung der Faustformel nahezulegen.

Zu klären ist außerdem, ob von einem guten Regelergebnis aus eine sprunghafte Störung automatisch auf ein gutes Ergebnis bei anderen Störmustern zu schließen ist.

14 vgl. Anhang, 20f.

4.5 Quantisierungsfehler

Um das analoge Signal, das zunächst als Höhe des Hebels und danach als Ausgangsspannung des Sensors vorliegt, in ein digitales Signal zu verwandeln, wird es quantisiert. Dabei wird es wert- und zeitdiskret.

Eine Stellschraube zum Verringern von Quantisierungsfehlern bietet die Geschwindigkeit des ADCs. Da aber bei höheren Geschwindigkeiten die Qualität der Umwandlung zurückgeht, können hier durch Zeitdiskretisierung hervorgerufene Quantisierungsfehler gegen durch Wertdiskretisierung hervorgerufene getauscht werden. Die Abtastrate von 37,5kHz, die sich aus einer Frequenz von 4MHz für die ADC-Clock und hardwareseitiger Mittlung über 4 Messungen ergibt, wurde jedoch ohne weitergehende Messungen gewählt.

Die Beschränkung des DAC auf 12 Bit ist insofern problematisch, dass so mit dem verwendeten Messbereich ein DAC-Schritt 4,3g entspricht und das bei einer Wägezelle, bei der eine Auflösung von 0,01g erreicht werden soll, unzureichend erscheint. Wird hier lediglich gerundet oder gar stets abgerundet, kommt es offensichtlich zu bleibenden Abweichungen, Ungenauigkeiten und anderen ungewollten Effekten. Dies kann jedoch umgangen werden, indem die Rundungsfehler eines Abtastzyklus im nächsten Zyklus aufaddiert werden, sodass bei ausreichend hoher Frequenz die Zwischenwerte mit Pulsweitenmodulation (PWM) nachgestellt werden können.

Stattdessen scheint der ADC Ursache des verrauschten Wägeergebnisses zu sein, obwohl seine Auflösung von umgerechnet 0,8mg bei Eingangsschaltung 2, der Schaltung, die als Ausgangspunkt der Versuche diente, deutlich besser ist, als beim DAC. Es driften jedoch Genauigkeit und Auflösung hier auseinander, da Umwandlungen stets mit Fehlern verbunden sind, sodass die effektiv nutzbaren Bits im Datenblatt mit 12,75 angegeben werden[15], was 7,3mg Genauigkeit entspricht.[16]

Experimentell wurde gezeigt, dass eine Verbesserung der Auflösung der Eingangsschaltung auch das Wägeergebnisses verbessert. Dabei wurde zunächst das durch Umwandlungsfehler ermittelte Rauschen in Ruhe als Standardabweichung des mit Fastmode gemessenen Wägeergebnisses gemessen. Dabei war von Schaltung 1 zu Schaltung 2 eine Verbesserung zu erkennen, zu Schaltung 3 jedoch nicht.[17]

Deutlicher wird der Effekt, wenn die Regelgüte bei Störungen und in Ruhe zusammen betrachtet wird: So findet die Optimierung, die mit der Regelfläche als Kriterium beide Fälle betrachtet, konservative Regelparameter, die sich von dem höheren Quantisierungsrauschen nicht zum Schwingen anregen lassen. Es können jedoch durch die Optimierung keine Parameter gefunden wer-

15 vgl. NXP 2017, Fig. 19

16 vgl Anhang, 17

17 vgl. Anhang, 18

den, die auf Störgrößensprünge vergleichbar mit dem mit Eingangschaltung 3 realisierten Regler reagieren. Nutzt man die für Eingangsschaltung 3 gefundenen Parameter mit Umrechnung der Verstärkung für Schaltung 2, sieht zwar die Sprungantwort ähnlich ebenfalls gut aus, es existieren aber dauerhafte Schwingungen, die auch die Optimierung nicht beseitigen kann.[18] Neben Quantisierungsfehlern ließe sich dieser Unterschied der Schaltungen auch über nicht ideales Verhalten der Operationsverstärkerschaltungen erklären.

Dies kann wahlweise durch eine höhere Auflösung des ADC oder durch eine Verringerung des Messbereichs umgesetzt werden. Um mit dem internen ADC des Teensy arbeiten zu können, wurde letzteres Verfahren gewählt, was grundsätzlich erfolgreich war. Da der Bereich, in dem die Regelstrecke linear ist, aber so kleiner wurde, gibt es Regelparameter, die für „normale“ Lasten sehr gut funktionieren, verlässt der Hebel den Messbereich aber signifikant instabil werden. Der Versuch eine Messbereichsumschaltung mit dem zweiten ADC zu realisieren und diese durch die Ermittlung gemeinsamer Punkte auf der statischen Kennlinie zu synchronisieren, konnte das Problem nicht beheben, weshalb davon auszugehen ist, dass bei einer realen Umsetzung eines digitalen Reglers ein ADC mit höherer Auflösung gewählt werden würde.

Da dieser Effekt aber nur bei Belastungen von einem Vielfachen des Messbereichs, für den die Waage ausgelegt ist, zu beobachten war und, da digitale Regler komplexere Regelvorschriften erlauben, wäre auch der Wechsel zu konservativeren Regelparametern beim Erkennen einer instabilen Situation möglich.

Konkret müsste, da die Verstärkung von Eingangsschaltung 3 etwa 4,5 mal so groß ist, wie die von Eingangsschaltung 2,[19] die effektiven Bits des ADC $\log_2(4{,}5) \approx 2{,}2$ höher, also bei ca. 15 liegen, um die Auflösung von Eingangsschaltung 3 bei einem Messbereich von Eingangsschaltung 2 zu erreichen. Selbstverständlich wäre aber eine höhere Auflösung stets hilfreich.

5 Fazit

Es wurde eine Schaltung aufgebaut und an eine Standard-Wägezelle von Wipotec angeschlossen, um dort als digitaler Regler zu arbeiten. Mit diesem Aufbau konnten experimentell verschiedene Regler mit verschiedenen Einstellverfahren und verschiedenen Vorverstärkungen getestet werden.

Damit wurden grundsätzlich die mit dieser Facharbeit zu erörternden Fragen beantwortet: Es ist mit digitalen Reglern sowohl möglich, das Quantisierungsrauschen auf ein Niveau zu bringen, das das Wägeergebnis nicht signifikant beeinflusst als auch Regelparameter automatisiert zu fin-

18 vgl. Anhang, 19

19 vgl. Anhang, 24

den, die bei Störungen besser regeln, als die aktuellen Werkseinstellungen des analogen Reglers.
Dennoch ist nach dieser Machbarkeitsstudie noch einige Entwicklung nötig, bis ein solcher digitaler Regler real eingesetzt werden kann. Einerseits wird die Auswahl der Hardware wohl anders aussehen, da der Faktor der unkomplizierten Programmierung weniger gewichtet würde, was einen besseren ADC, als den hier intern verbauten ermöglicht.
Wenngleich hier ein funktionsfähiger Optimierungsalgorithmus gefunden wurde, reichte der Umfang dieser Arbeit nicht aus, um weitere Algorithmen zu testen und vergleichen zu können oder möglichst viele Regelparameter zu testen, um einen Überblick über die Verteilung geeigneter Parameter zu erhalten. Ebenfalls müssten sämtliche Tests auf mehrere Wägezellen ausgeweitet werden.
Für eine fertige Umsetzung sind natürlich weiter Fragen der Kommunikationsschnittstellen zu klären, Funktionen wie die Speicherung von Parametern im stromlosen Zustand (z.B. im EEPROM) zu implementieren sowie Fallback-Ebenen einzubauen, die, sollte die Einstellung in einer Sackgasse mit instabilen Parametern festsitzen, diese dann (evtl. mit anderen Ausgangswerten) neu startet.

6 Anhang

6.1 Erklärung zur Selbstständigkeit

Hiermit erkläre ich, dass ich, Lars Gottfriedsen, die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken im Sinne wissenschaftlicher Nachprüfbarkeit als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift

6.2 Literaturverzeichnis

- Arduino SA (Hg.): Sprach-Referenz, https://www.arduino.cc/reference/de, Stand 19.03.2022
- Band, Ricardo: Teensy symbol library for KiCAD, https://github.com/XenGi/teensy_library, Stand: 18.03.2022
- NXP Semiconductors (Hg.): Kinetis K66 Sub-Family, https://www.nxp.com/docs/en/data-sheet/K66P144M180SF5V2.pdf, Stand 28.05.2022
- Prof. Dr.-Ing. W. Schumacher: Grundlagen der Regelungstechnik, https://srv.ifr.ing.tu-bs.de/static/files/lehre/vorlesungen/gdr/Skript_GdR.pdf, Stand 22.02.2022, Kapitel 8 u. 12
- Stiny, Leonhard: Regelungstechnik – Crashkurs. Konstanz: Verlag Dr.-Ing. Paul Christiani GmbH & Co. KG, 2013
- Villanueva, Pedro: ADC_Module Class Reference, http://pedvide.github.io/ADC/docs/Teensy_3_6_html/class_a_d_c___module.html, Stand 19.03.2022
- Wipotec GmbH (Hg.): Wägeprinzip (EDK/EMFR), https://www.wipotec-wt.com/de/technologie/waegeprinzip, Stand 25.05.2022

6.3 Abbildungsverzeichnis

6.4 Messdaten

Ausschnitt aus statischer Kennlinie: Im restlichen DAC-Bereich von 0 bis 4095 bleibt der Regelwert konstant.

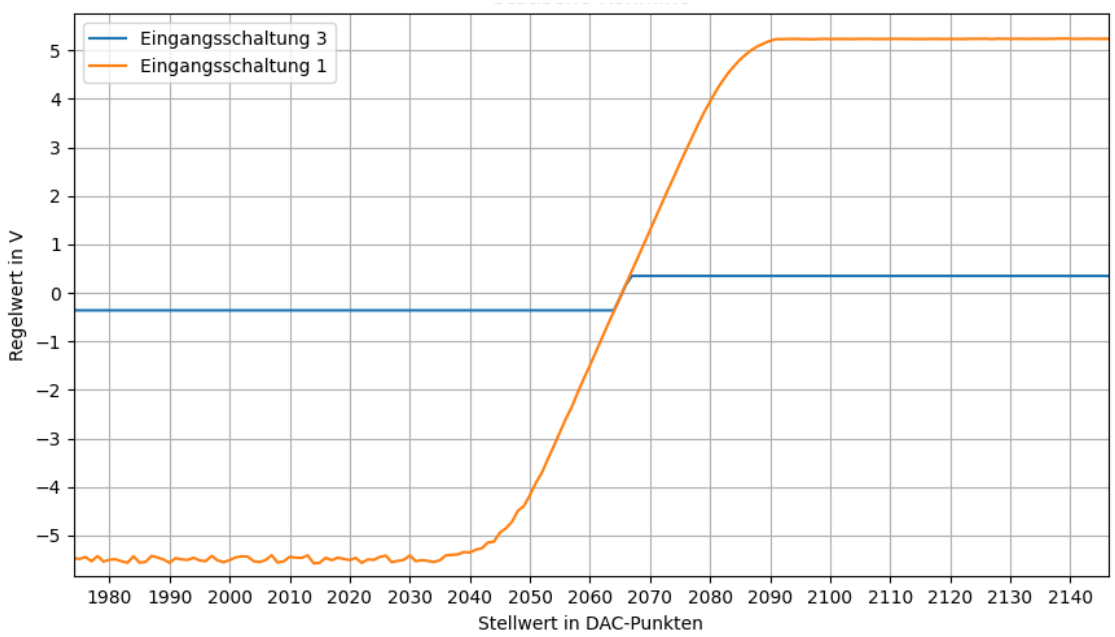

Abb 9: Statische Kennlinie mit mehreren Eingangsschaltungen

Dynamische Kennlinie gemessen am Oszilloskop mit Last von ca. 775g bei Stellwertsprung von 2060 auf 2065 DAC-Schritte bei t=0:

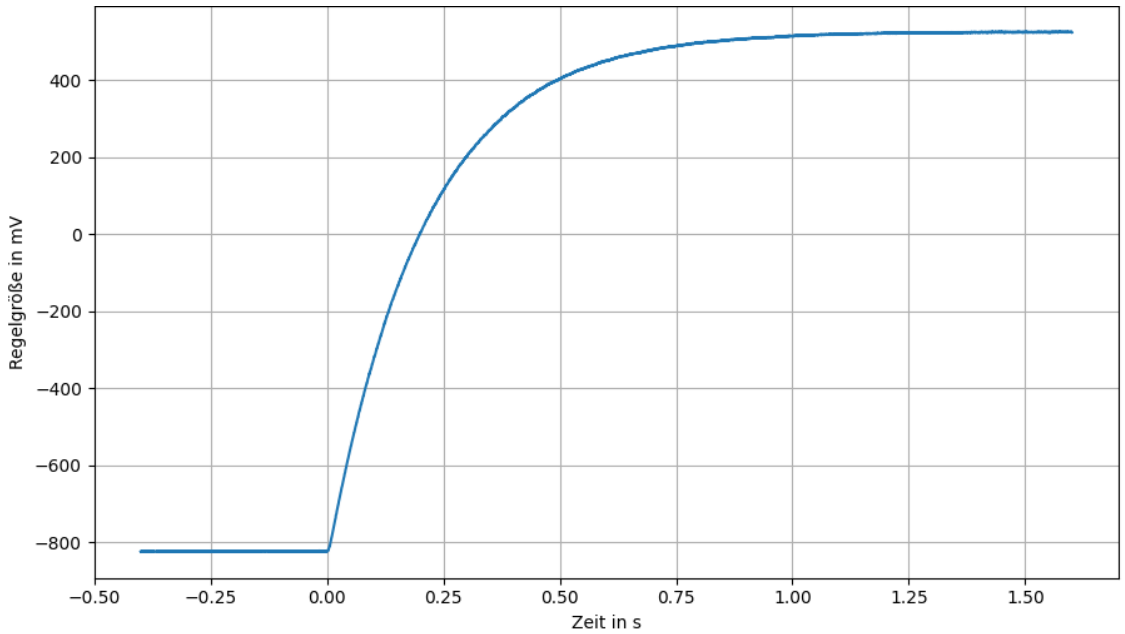

Abb 10: Dynamische Kennlinie

Spannungsbereiche	**Min**	**Max**
U_{pos} mechanischer Anschlag	-5,71V *	5,28V *
U_{pos} Messbereich Eingangsschaltung 1	-7,5V **	7,5V **
U_{pos} Messbereich Eingangsschaltung 2	-1,65V **	1,65V **
U_{pos} Messbereich Eingangsschaltung 3	-0,36V **	0,36V **
U_{Stell} Ausgangsschaltung	-2,16V **	1,86V **

* experimentelle Messungen

** Berechnungen nach Formeln auf Seite 24, experimentelle Prüfung

Ermittlung der Auflösung des DAC in g gemessen im geschlossenen Regelkreis:

Stellwert P_{DAC}	**Masse m**
2216	0g
1635	2500g

$$\frac{\Delta m}{\Delta P_{DAC}} = \frac{0\,g - 2500\,g}{2216 - 1635} = -4{,}3\,g$$

Übertragungsbeiwert im offenen Regelkreis aus statischer Kennlinie gemessen:

Eingangs-schaltung	Übertragungsbeiwert $K_S = \frac{\Delta x}{\Delta y} = \frac{\Delta P_{ADC}}{\Delta P_{DAC}}$	Auflösung ADC $\frac{\Delta m}{\Delta P_{ADC}} = \frac{\Delta m}{\Delta P_{DAC}} : K_S$	Bezogen auf effektive ADC-Punkte: $\frac{\Delta m}{\Delta P_{ADCeff}} = \frac{\Delta m}{\Delta P_{ADC}} \cdot 2^{(16-12{,}75)}$
1	1022	-4,210mg	-40,05mg
2	5590	-0,770mg	-7,33mg
3	25437	-0,169mg	-1,61mg

Messung Quantisierungsrauschen in Ruhe (Grenzfrequenz Tiefpass 8Hz) nach Selbsteinstellung und zweimaliger Optimierung:

Eingangs-schaltung	P-Faktor	I-Faktor	D-Faktor	Filter-Parameter	Standardabweichung Wägeergebnis
1	0,13	0,00072	0,65	0,52	8,56mg
2	0,031	0,00021	0,23	0,47	2,31mg
3	0,0058	0,000056	0,054	0,45	3,38mg
Analogregler					3,14mg

Da die Auflösung des Wägeergebnisses bei 10mg liegt, sind Standardabweichungen unter 5mg als vernachlässigbar zu betrachten.

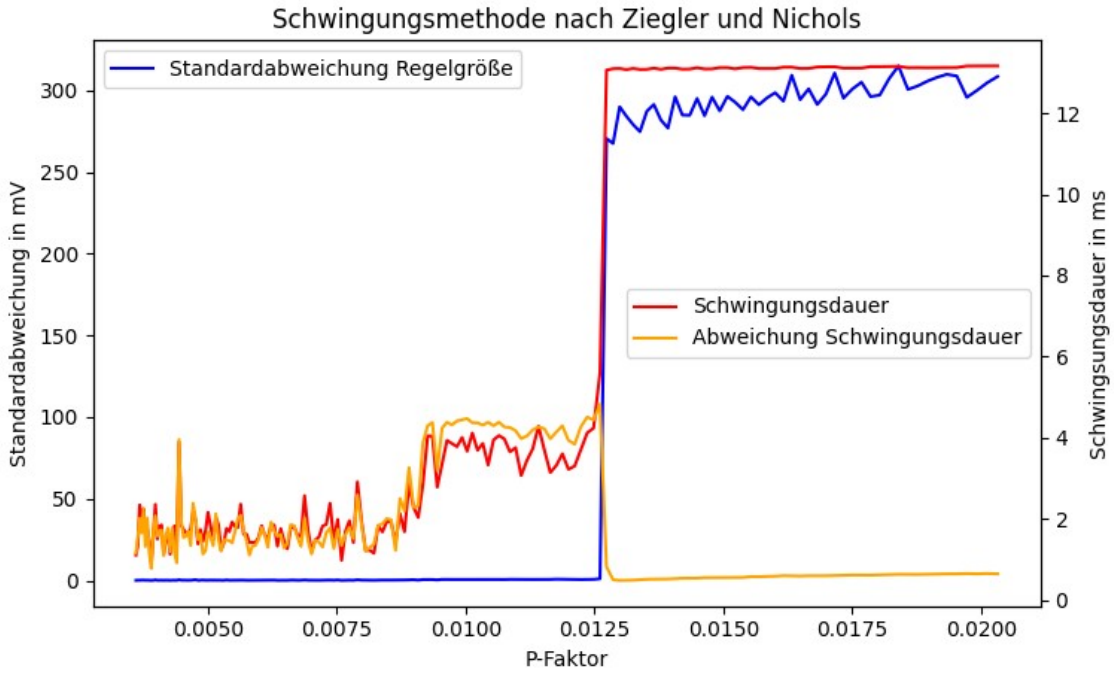

Abb 11: Kriterien zum Erkennen ungedämpfter Schwingungen

Zum Vergleich verschiedener Eingangsschaltungen herangezogen wurden Regler, die für Eingangschaltung 2 und 3 optimiert wurden sowie ein Regler, der die Parametern von Schaltung 3, jedoch die Eingangsschaltung 2 genutzt hat:

Eingangs-schaltung	P-Faktor	I-Faktor	D-Faktor	Filter-Parameter	Regelfläche	Überschwing-weite
2	0,0276	0,000114	0,246	0,65	1662mVs	121mV
2	0.0211·4,5	0.000161·4,5	0.166·4,5	0.89	1957mVs	32mV
3	0.0211	0.000161	0.166	0.89	161mVs	36mV

Mit dem Oszilloskop erfasste Regelreaktion verschiedener Regler auf einen Sprung der Stellgröße bei t=0 um 75 DAC-Punkte (≈325g) gefiltert:

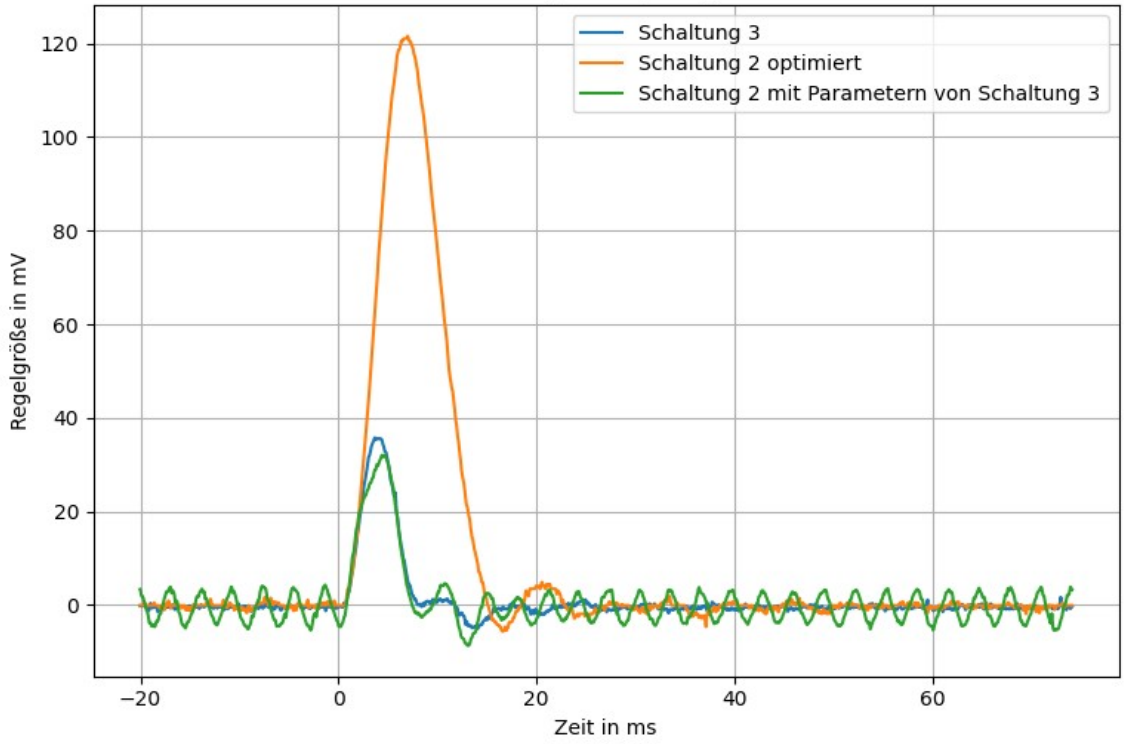

Abb 12: Schaltungsvergleich Störgrößensprünge

Regelparameter und Regelgüte im Verlauf der Selbstoptimierung. Optimierungsdurchlauf 0 entspricht dem Ergebnis der Einstellung durch die Faustformel von Ziegler und Nichols. Regelfläche meint die zeitbeschwerte, betragslineare Regelfläche bei einem Störgrößensprung von 75 DAC-Schritten (≈325g) über die ersten 155ms. Dargestellt sind vier unabhängige Durchgänge.

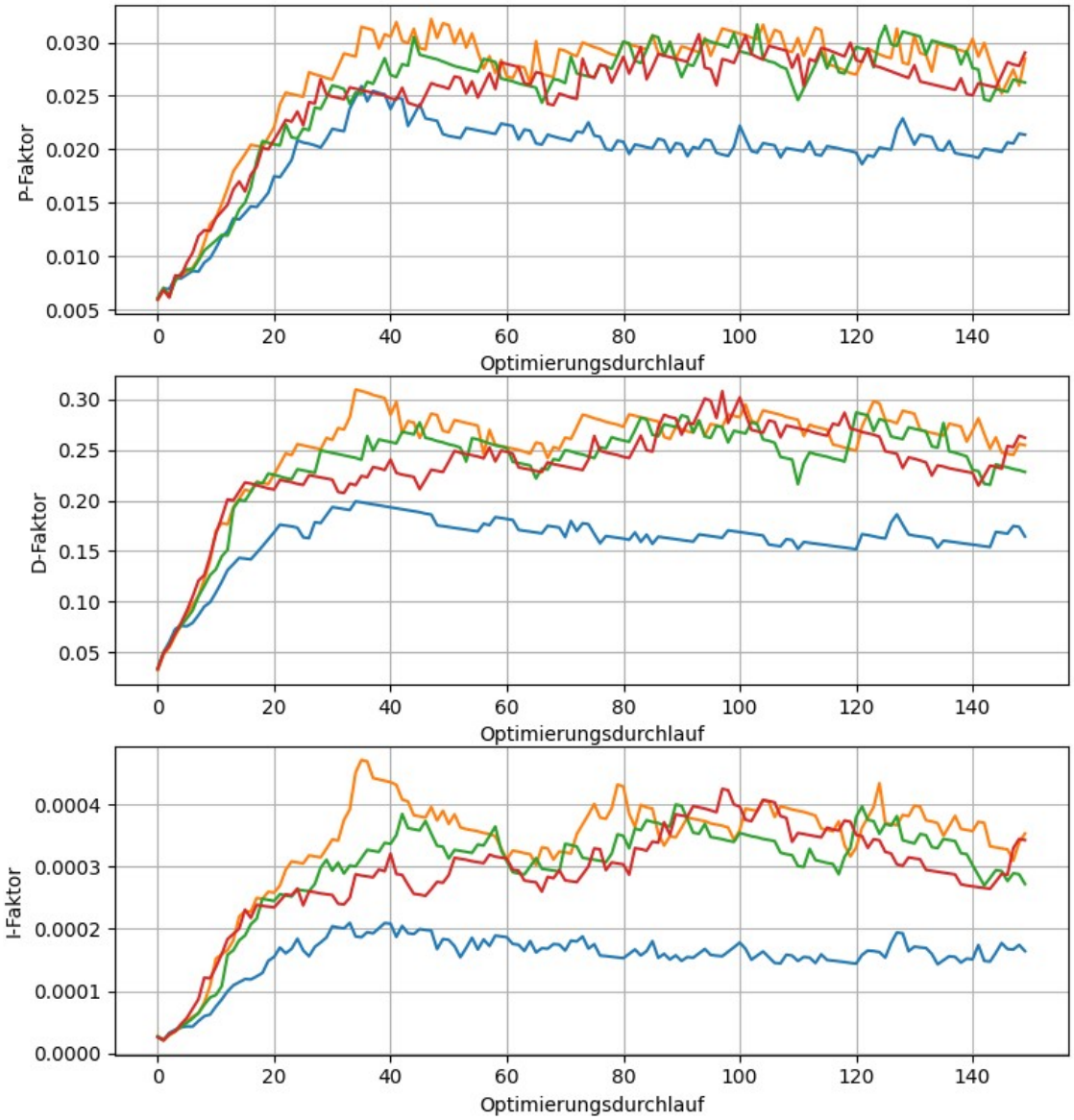

Abb 13: Optimierungsverlauf (Teil 1)

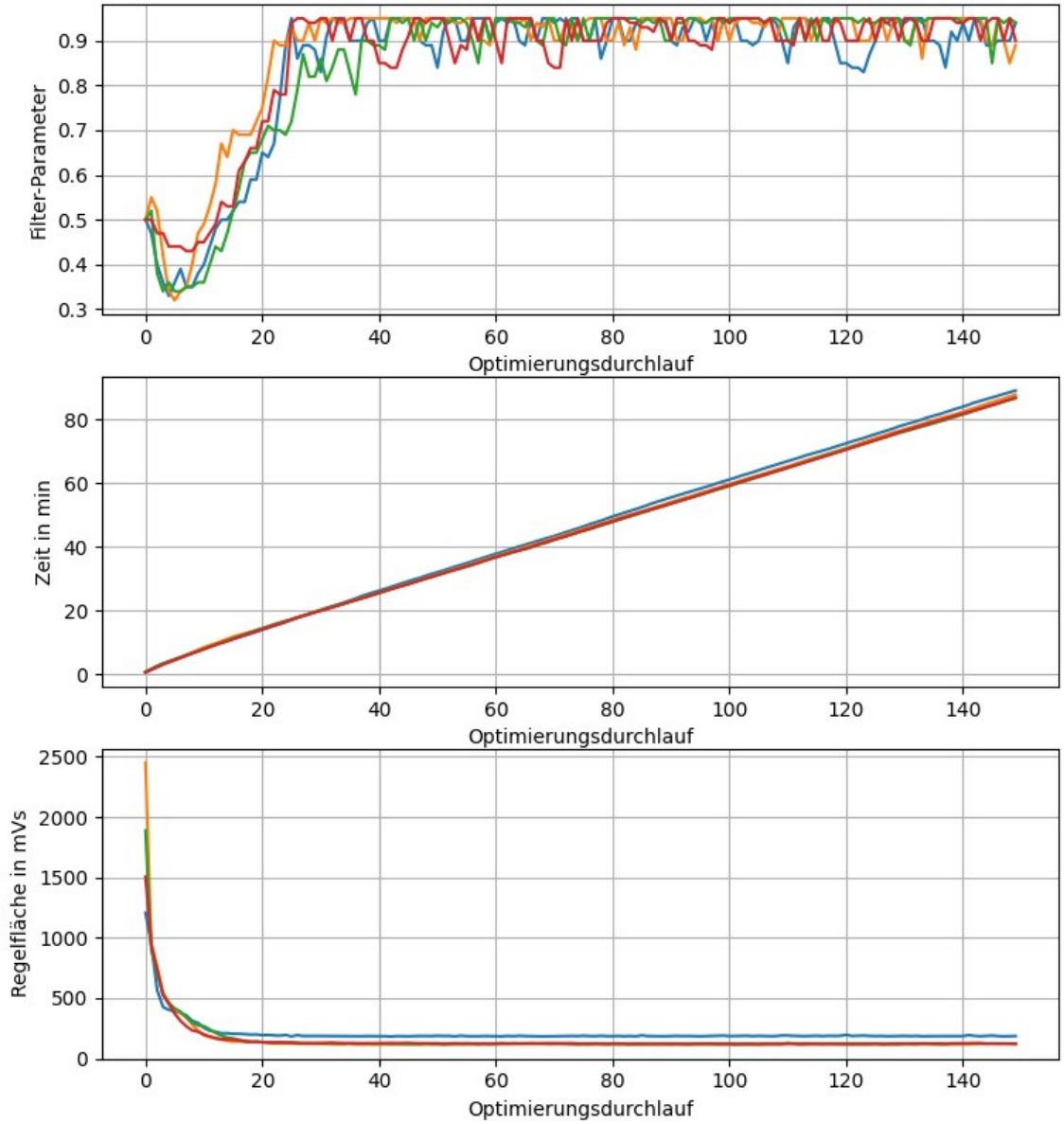

Abb 14: Optimierungsverlauf (Teil 2)

Mit dem Oszilloskop erfasste Regelreaktion verschiedener Regler auf einen Sprung der Stellgröße bei t=0 um 75 DAC-Punkte (≈325g) gemittelt über mehrere Messungen und gefiltert:

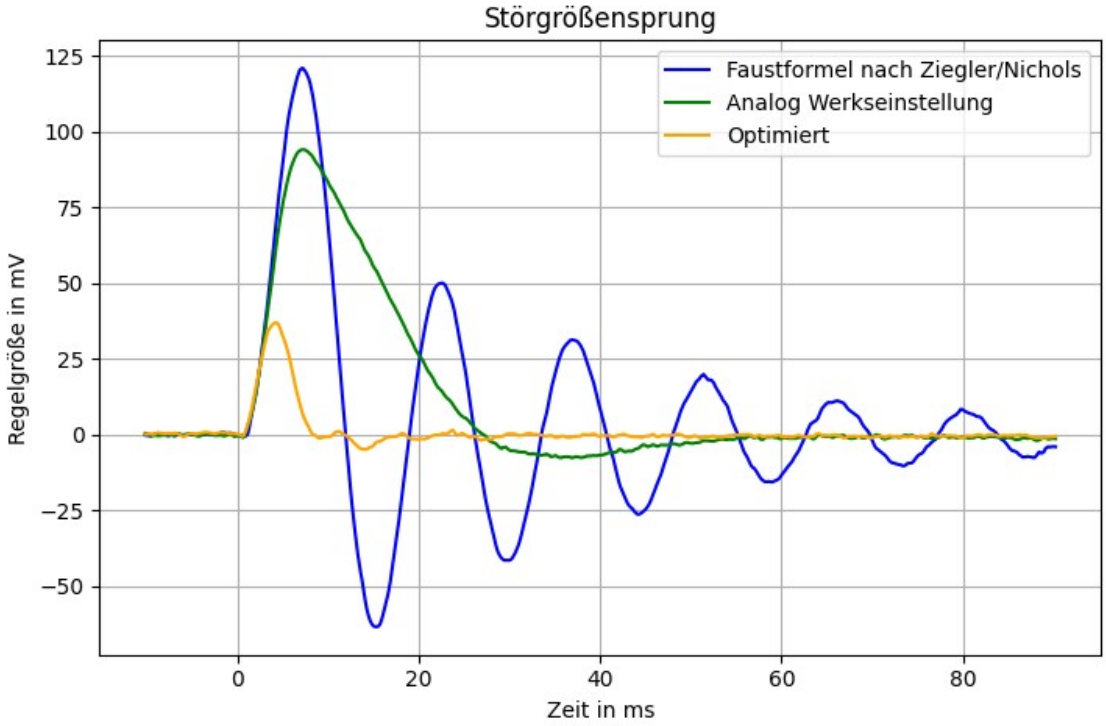

Abb 15: Reglervergleich Störgrößensprünge

6.5 Schaltung

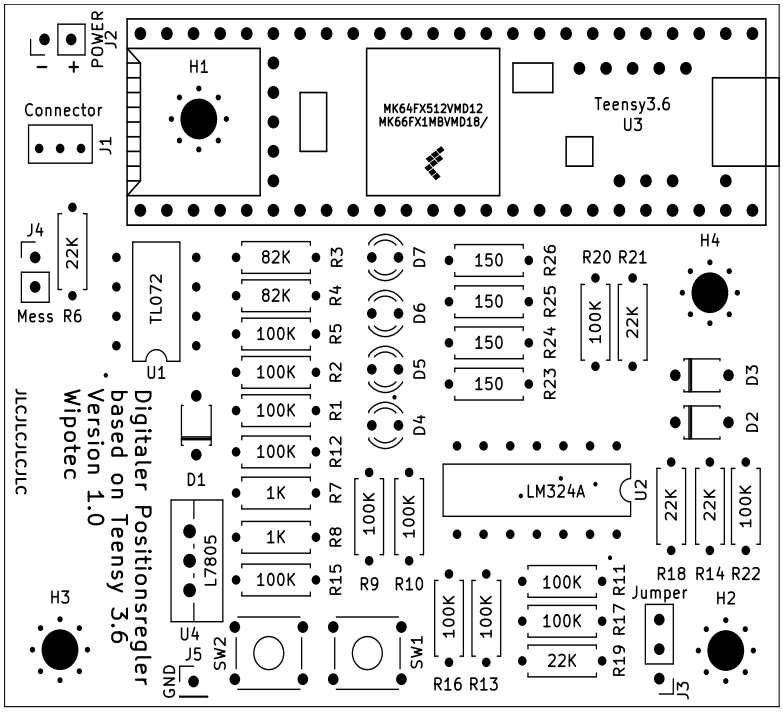

Abb 16: Bestückungsplan (Maßstab 3:2)

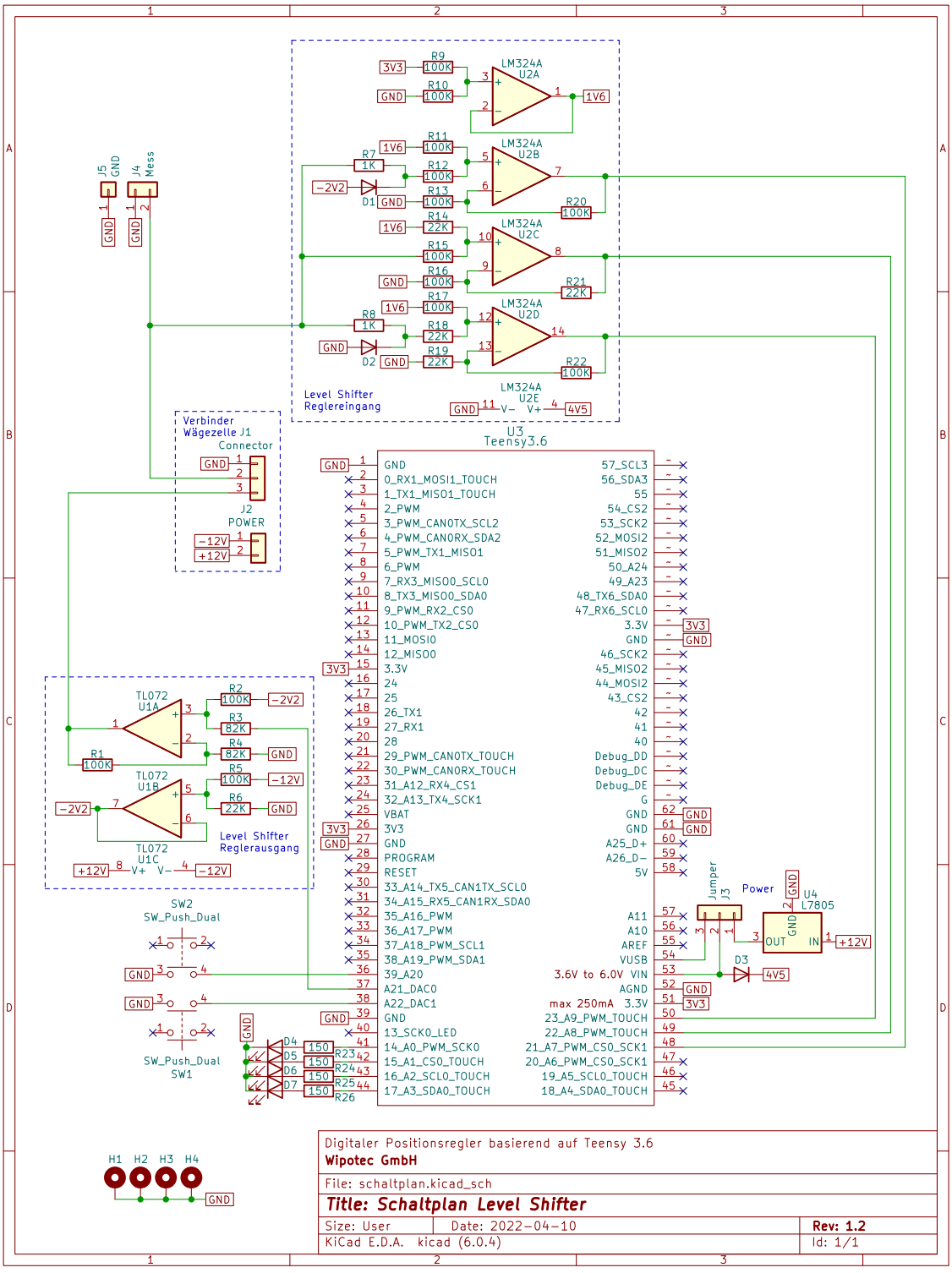

Abb 17: Schaltplan

Eine der verwendeten Grundschaltungen ist ein Spannungsteiler aus zwei Widerständen mit $U_{out}=\frac{U_1 R_2+U_2 R_1}{R_1+R_2}$, der jedoch in dieser Form nur funktioniert, wenn am Ausgang kein Strom fließt. Um dies sicherzustellen wurden sie hier teilweise in Kombination mit Spannungsfolgern / Impedanzwandlern mit $U_{out}=U_1$ eingesetzt, die Stromflüsse an ihrem Ausgang durch die Stromversorgung des Operationsverstärkers bedienen, sodass an ihrem Eingang kein Strom fließt.

Um die Spannungsbereiche der Wägezelle auf die den des Teensy (0-3,3V) zu verschieben, werden diese Level-Shifter mit $U_{out}=U_1+U_2\cdot\frac{R_2}{R_1}$ genutzt, die außerdem U_{out} durch die Versorgungsspannung des Operationsverstärker beschränken. Die Spannungsverstärkung dieser Schaltung liegt damit bei $\frac{R_2}{R_1}$. Sie liegt damit für Eingangsschaltung 1 bei $\frac{22\,k\Omega}{100\,k\Omega}=0{,}22$, für Eingangschaltung 2 bei $\frac{100\,k\Omega}{100\,k\Omega}=1$ und für Eingangsschaltung 3 bei $\frac{100\,k\Omega}{22\,k\Omega}\approx 4{,}5$.

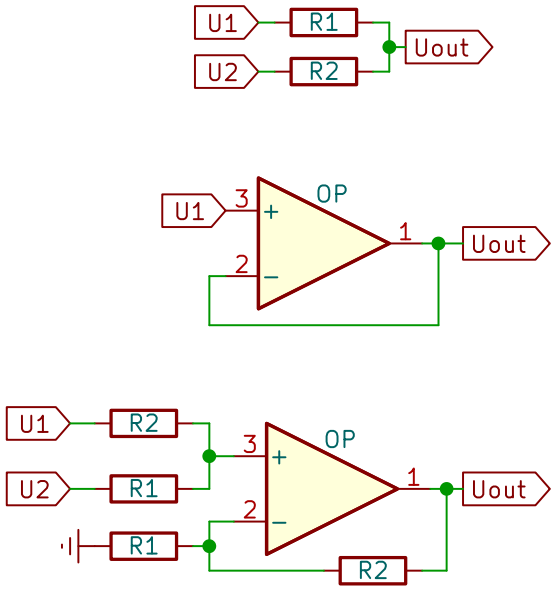

Abb 18: Grundschaltungen

Für die Eingangsschaltungen muss die Offset-Spannung U_1 bei $\frac{3{,}3\,V}{2}=1{,}65\,V$ liegen, um einen Spannungsbereich, der mittig auf GND liegt in die Mitte des Eingangsbereichs des Teensy von 0 – 3,3V zu verschieben.

Da der hier verwendete Operationsverstärker für den positiven Eingang U_+ Absolute-Minimum-Ratings von -0,3V hat, muss gelten:

$$-0{,}3\,V < U_+ = \frac{U_1 R_1+U_2 R_2}{R_1+R_2}$$
$$\Leftrightarrow\ -0{,}3\,V-\frac{U_1 R_1}{R_1+R_2} < \frac{U_2 R_2}{R_1+R_2}$$
$$\Leftrightarrow\ -0{,}3\,V\cdot(R_1+R_2)-U_1 R_1 < U_2 R_2$$
$$\Leftrightarrow\ -0{,}3\,V\cdot\frac{R_1+R_2}{R_2}-U_1\frac{R_1}{R_2} < U_2$$

für Eingangsschaltung 1:

$$U_2 > -0{,}3V \cdot \frac{R_1+R_2}{R_2} - U_1 \frac{R_1}{R_2} = -0{,}3\,V \cdot \frac{100\,k\Omega+22\,k\Omega}{22\,k\Omega} - 1{,}65\,V \cdot \frac{100\,k\Omega}{22\,k\Omega} \approx -9{,}16\,V$$

für Eingangsschaltung 2:

$$U_2 > -0{,}3V \cdot \frac{R_1+R_2}{R_2} - U_1 \frac{R_1}{R_2} = -0{,}3\,V \cdot \frac{100\,k\Omega+100\,k\Omega}{100\,k\Omega} - 1{,}65\,V \cdot \frac{100\,k\Omega}{100\,k\Omega} \approx -1{,}98\,V$$

für Eingangsschaltung 3:

$$U_2 > -0{,}3V \cdot \frac{R_1+R_2}{R_2} - U_1 \frac{R_1}{R_2} = -0{,}3\,V \cdot \frac{22\,k\Omega+100\,k\Omega}{100\,k\Omega} - 1{,}65\,V \cdot \frac{22\,k\Omega}{100\,k\Omega} \approx -0{,}73\,V$$

Da diese Werte für die Schaltungen 2 und 3 oberhalb der unteren Grenze des Spannungsbereiches des Sensors (-5,71V) liegt, wurde hier je eine zusätzliche Diode (D1 und D2) eingebaut, die sich normalerweise in Sperrrichtung befindet. Sollte die Sensorspannung aber unter einen bestimmten Wert fallen, erzeugt sie einen Kurzschluss und begrenzt so die Spannung. Da Siliziumdioden eine Durchlassspannung von ca. 0,5V haben, kann die an der Anode der Diode angelegte Spannung etwa 0,5V über den errechneten Werten liegen.

Der vielfach verwendete digitale Tiefpass wird folgendermaßen berechnet:

$$y = k \cdot x + (1-k) \cdot y_{alt}$$

wobei Grenzfrequenz f_G und Filterparameter k folgendermaßen zusammenhängen:

$$k = 1 - \exp\left(-2\pi \frac{f_G}{f_A}\right)$$

$$\Leftrightarrow \quad f_G = \frac{\log_e(1-k)}{-2\pi} \cdot f_A$$

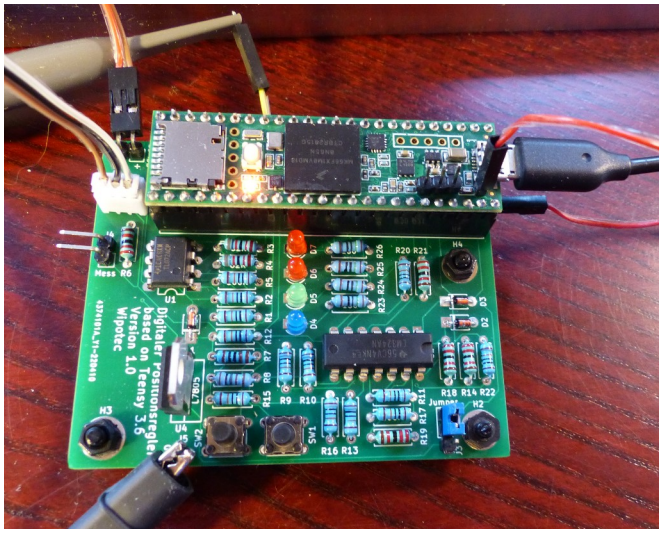

Abb 19: Regler auf Platine

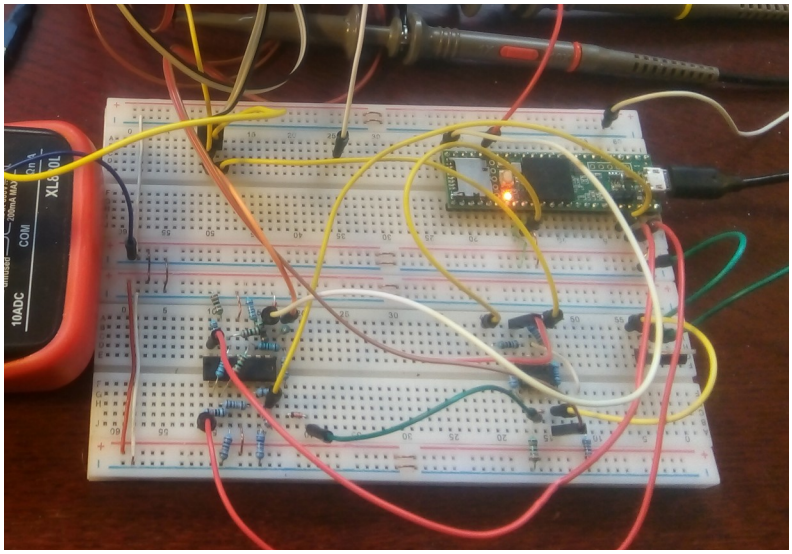

Abb 20: Regler auf Breadboard

6.6 Programmcode

6.6.1 Hauptprogramm Teensy

```
#include <ADC.h>
#include <ADC_util.h>

const int readPinMain = A9;     // Eingangspin zum Regeln    A8, A7, A9
const int readPinTotal = A13;   // Eingangspin mit komplettem Messbereich
const int writePin = A21;       // Ausgangspin für Stellgröße
const int triggerPin = 25;      // Pin als Trigger für Oszilloskop
const int LEDpower = 13;        // Pin für Status-LED: Power
const int LEDrange = 16;        // Pin für Status-LED: Messbereich
const int LEDoverflow = 17;     // Pin für Status-LED: Zahlenüberlauf i_summe
```

```
// relative Änderung der Regelparameter in einem Optimierungsschritt
const float optimierungsschritt = 0.05;
// Verschiebung der Werte in i_summe in Bits,
// um überlaufen der Variable zu vermeiden
const int versatz_i_summe = 0;

long var_copy = 0;   // zum kurzzeitigem Zwischenspeichern zur Ausgabe

// Zeit seit Stellgößen- oder Störgrößensprung in Abtastzeiten
unsigned long counter = 0;
// Zeit seit Beginn der Frequenzmessung in Abtastzeiten
unsigned long frequenzcounter = 0;
// micros() zu Beginn von Frequenzmessung
// bzw. Differenz nach Abschluss der Frequenzmessung
unsigned long frequenzzeit = 0;
// Frequenzmessung aktiv?
bool frequenzmessung = false;

// Größter Wert von abweichung in ADC-Punkten
unsigned long ueberschwingen = 0;
// Zeit nach Störgrößensprung in Abtastzeiten,
// bis Regeldifferenz (abweichung) im Bereich ±500 ADC-Punkte bleibt
unsigned long einregelzeit = 0;
// Zeitbeschwerte betragslineare Regelfläche
// über die ersten 6000 Abtastzeiten (=160ms)
// mit Vorfaktor 1/10 gegenüber Abtastzeiten*ADC-Punkte
unsigned long regelflaeche = 0;
// Mittelwert von ueberschwingen über zwei Sprünge
unsigned long mittelwert_ueberschwingen = 0;
// Mittelwert von einregelzeit über zwei Sprünge
unsigned long mittelwert_einregelzeit = 0;
// Mittelwert von regelflaeche über zwei Sprünge
unsigned long mittelwert_regelflaeche = 0;
// bei Optimierung Wert vor letzter Veränderung
unsigned long last_value = 0;

// Zeit in Abtastzeiten seit letzter Nullstelle
unsigned long t_null = 0;
// Tiefpass (k=0,95  fg=1,2Hz bei fa=150Hz) über t_null bei nächster
// Nullstelle mit Vorfaktor 100
unsigned long t_schwing = 1300;
// Differenz von aktueller Differenz der Nullstellen und Mittelwert
// in Abtastzeit*100
long t_schwingabweichung_aktuell = 0;
// Tiefpass (k=0,98  fg=0,5Hz bei fa=150Hz) über Betrag von
t_schwingabweichung_aktuell
unsigned long t_schwingabweichung_mittelwert = 0;

long i_summe = 0; // differenz_halb aufaddiert

unsigned int sollwert = 34200;  // Sollwert in ADC-Punkten

// Messwert readPinMain
long messwert1 = 0;
// Messwert readPinTotal angepasst
long messwert2 = 0;
// ungefilterter Messwert in ADC-Punkten
```

```
long istwert = 0;
// Wert von istwert im vorangegangenen Zyklus
long istwert_last = 0;
// istwert mit starkem Tiefpass (k=0,99995  fg=0,3Hz) und Vorfaktor 1000
long istwert_mittelwert = sollwert*1000;
// istwert mit variablen Tiefpass für D-Anteil und Vorfaktor 10
long istwert_tiefpass_d = sollwert*10;
// Wert von istwert_tiefpass_d im vorangegangenen Zyklus
long istwert_tiefpass_d_last = sollwert*10;
// istwert mit Tiefpass für Rauschbeurteilung (k=0,998  fg=10Hz) und
// Vorfaktor 10
long istwert_tiefpass_10hz = sollwert*10;

// Regeldifferenz: sollwert - istwert
long differenz = 0;
// Regeldifferenz mit Vorfaktor 1/2^versatz_i_summe genutzt für i_summe
long differenz_halb = 0;
// momentane Differenz von Mittelwert zu Messwert mit Vorfaktor 10
long abweichung = 0;
// betragsmäßige Abweichung mit Tiefpass (k=0.998  fg=12Hz) und Vorfaktor 10
unsigned long standardabweichung = 0;

// Istwertänderung gefiltert mit Vorfaktor 10:
// aenderung_tiefpass = istwert_tiefpass_d_last - istwert_tiefpass_d
long aenderung_tiefpass = 0;

// Rundungsfehler beim Runden auf ganze DAC-Punkte.
// Wird im nächsten Zyklus aufaddiert => PWM
float quantisierungfehler = 0;
// aktueller Stellwert in DAC-Punkten
unsigned int stellwert = 0;
// Offset für Störgrößensprünge und, um i_summe klein zu halten
unsigned int stelloffset = 2150;

// REGELPARAMETER
float p_faktor = 0.02110398;     // differenz * p_faktor = P-Anteil
float i_faktor = 0.00016082;     // i_summe * i_faktor = I-Anteil
float d_faktor = 0.16564891;     // aenderung_tiefpass * d_faktor = D-Anteil
float filter_parameter = 0.89;   // Filterkoeffizient für Tiefpass von
                                 // D-Anteil: k = 1 - filter_parameter

// Kennwerte der Regelstrecke für Stellgrößensprung nach Strejc
boolean strejc_aktiv = false;  // Messung aktiv? Öffnet Regelkreis
unsigned long t10 = 0;    // Zeit in Abtastzeiten bis zur Anpassung auf 10%
unsigned long t20 = 0;    // Zeit in Abtastzeiten bis zur Anpassung auf 20%
unsigned long t80 = 0;    // Zeit in Abtastzeiten bis zur Anpassung auf 80%
unsigned long t90 = 0;    // Zeit in Abtastzeiten bis zur Anpassung auf 90%
long tu = 0;    // durch Faustformeln ermittelter Streckenparameter T_u
long tg = 0;    // durch Faustformeln ermittelter Streckenparameter T_g

// 3 Punkte aus statischer Kennlinie als
// {Stellwert, Regelwert Hauptschaltung, Regelwert Sekundärschaltung}
unsigned int pos1[] = {2122, 26702, 32556};
unsigned int pos2[] = {2122, 26702, 32556};
unsigned int pos3[] = {2123, 51110, 33723};

// Übertragungsbeiwert Ks der Regelstrecke: ΔRegelwert / ΔStellwert
int ks = (pos3[1]-pos1[1])/(pos3[0]-pos1[0]);
```

```
ADC *adc = new ADC();
ADC::Sync_result result;


void statische_kennlinie(long startwert,
                         long stopwert,
                         unsigned int schrittweite,
                         unsigned int intervall) {
  if (startwert < 0)    {startwert = 0;   }
  if (stopwert  > 4096) {stopwert  = 4096;}
  Serial.print(startwert); Serial.print(" - "); Serial.println(stopwert);
  analogWrite(writePin,startwert);
  delay(max(intervall*4, 500));
  pos1[0]=0; pos1[1]=0; pos1[2]=0;
  pos2[0]=0; pos2[1]=0; pos2[2]=0;
  pos3[0]=0; pos3[1]=0; pos3[2]=0;
  for (unsigned int i = startwert; i<stopwert; i=i+schrittweite) {
    analogWrite(writePin, i);
    delay(intervall);
    result = adc->readSynchronizedContinuous();
    result.result_adc0 = (uint16_t)result.result_adc0;
    result.result_adc1 = (uint16_t)result.result_adc1;
    if (pos1[1] == 0 and result.result_adc0 > 10000) {
      pos1[0] = i;
      pos1[1] = result.result_adc0;
      pos1[2] = result.result_adc1;
    } if (pos2[1] == 0 and result.result_adc0 > 20000) {
      pos2[0] = i;
      pos2[1] = result.result_adc0;
      pos2[2] = result.result_adc1;
    } if (pos3[1] == 0 and result.result_adc0 > 32000) {
      pos3[0] = i;
      pos3[1] = result.result_adc0;
      pos3[2] = result.result_adc1;
      if (pos3[0] != pos1[0]) {
        ks = (pos3[1]-pos1[1])/(pos3[0]-pos1[0]);
      } else {
        ks = 0;
      }
      return;
    }
  }
}


void stoersprung(int sprung) {
  delay(500);
  noInterrupts();
  einregelzeit = 0;
  counter = 0;
  ueberschwingen = 0;
  stelloffset = stelloffset + sprung;
  regelflaeche = 0;
  interrupts();
  digitalWrite(triggerPin, HIGH);
  delay(500);
```

```
  noInterrupts();
  mittelwert_einregelzeit = einregelzeit*0.5;
  mittelwert_regelflaeche = regelflaeche*0.5;
  mittelwert_ueberschwingen = ueberschwingen*0.5;
  einregelzeit = 0;
  counter = 0;
  ueberschwingen = 0;
  stelloffset = stelloffset - sprung;
  regelflaeche = 0;
  interrupts();
  digitalWrite(triggerPin, LOW);
  delay(500);
  noInterrupts();
  mittelwert_einregelzeit   = mittelwert_einregelzeit + einregelzeit*0.5;
  mittelwert_regelflaeche   = mittelwert_regelflaeche + regelflaeche*0.5;
  mittelwert_ueberschwingen = mittelwert_ueberschwingen + ueberschwingen*0.5;
  interrupts();
}


void setup() {
  pinMode(LEDpower, OUTPUT); digitalWrite(LEDpower, HIGH);
  pinMode(LEDrange, OUTPUT); digitalWrite(LEDoverflow, HIGH);
  pinMode(LEDoverflow, OUTPUT); digitalWrite(LEDoverflow, HIGH);
  pinMode(triggerPin,  OUTPUT); digitalWrite(triggerPin,  LOW);
  analogWriteResolution(12);

  // starte ADC
  pinMode(readPinMain, INPUT);
  adc->adc0->setReference(ADC_REFERENCE::REF_3V3);
  adc->adc0->setAveraging(4);
  adc->adc0->setResolution(16);
  adc->adc0->setConversionSpeed(ADC_CONVERSION_SPEED:: ADACK_4_0);
  adc->adc0->setSamplingSpeed(ADC_SAMPLING_SPEED::VERY_HIGH_SPEED);
  adc->adc0->enableInterrupts(adc0_isr);

  adc->adc1->setReference(ADC_REFERENCE::REF_3V3);
  adc->adc1->setAveraging(4);
  adc->adc1->setResolution(16);
  adc->adc1->setConversionSpeed(ADC_CONVERSION_SPEED:: ADACK_4_0);
  adc->adc1->setSamplingSpeed(ADC_SAMPLING_SPEED::VERY_HIGH_SPEED);
  //adc->adc1->enableInterrupts(adc0_isr);

  adc->startSynchronizedContinuous(readPinMain, readPinTotal);

  Serial.begin(112500);
}



void loop() {
  if (Serial.available()) {
    char c = Serial.read();
    if (c=='m') {        // Ausgabe Messwert
      noInterrupts();
      Serial.print("Messwert: ");
      Serial.print(messwert1);
      Serial.print(": ");
```

```
  Serial.print(messwert2);
  Serial.print(": ");
  Serial.print(istwert_mittelwert/1000);
  Serial.print(": ");
  Serial.print(istwert_tiefpass_d/10);
  Serial.print(": ");
  Serial.println(istwert);
  interrupts();
}
else if (c=='s') {   // Ausgabe Stellwert
  noInterrupts();
  var_copy = stellwert;
  interrupts();
  Serial.print("Stellwert: ");
  Serial.println(var_copy);
}
else if (c=='+') {   // Sollwert erhöhen
  noInterrupts();
  sollwert = sollwert + 10;
  var_copy = sollwert;
  interrupts();
  Serial.print("Sollwert: ");
  Serial.println(var_copy);
}
else if (c=='-') {    // Sollwert verringern
  noInterrupts();
  sollwert = sollwert - 10;
  var_copy = sollwert;
  interrupts();
  Serial.print("Sollwert: ");
  Serial.println(var_copy);
}
else if (c=='j') {    // Sollwertsprung
  Serial.println("\nSOLLWERTSPRUNG");
  noInterrupts();
  sollwert = sollwert + 11700;
  interrupts();
  digitalWrite(triggerPin, HIGH);
  delay(500);
  noInterrupts();
  sollwert = sollwert - 11700;
  interrupts();
  digitalWrite(triggerPin, LOW);
}
else if (c=='v') {    // Vorlastabgleich
  noInterrupts();
  stelloffset = stellwert;
  var_copy = stellwert;
  interrupts();
  Serial.print("Vorlast: ");
  Serial.println(var_copy);
}
else if (c=='k') {    // statische Kennlinie
  adc->adc0->disableInterrupts();
  Serial.println("\nERMITTLUNG STATISCHE KENNLINIE");
  Serial.print("start first round ... ");
  statische_kennlinie(0,4096,15,30);
  Serial.print("position1 = ");
```

```
  Serial.print(pos1[0]); Serial.print(" : ");
  Serial.print(pos1[1]); Serial.print(" : "); Serial.println(pos1[2]);
  Serial.print("position2 = ");
  Serial.print(pos2[0]); Serial.print(" : ");
  Serial.print(pos2[1]); Serial.print(" : "); Serial.println(pos2[2]);
  Serial.print("position3 = ");
  Serial.print(pos3[0]); Serial.print(" : ");
  Serial.print(pos3[1]); Serial.print(" : "); Serial.println(pos3[2]);
  Serial.print("Ks = "); Serial.println(ks);
  Serial.print("start second round ... ")
  statische_kennlinie(pos2[0]-max(100,(pos3[0]-pos1[0])*5),
                      pos2[0]+max(150,(pos3[0]-pos1[0])*5),
                      3, 50);
  Serial.print("position1 = ");
  Serial.print(pos1[0]); Serial.print(" : ");
  Serial.print(pos1[1]); Serial.print(" : "); Serial.println(pos1[2]);
  Serial.print("position2 = ");
  Serial.print(pos2[0]); Serial.print(" : ");
  Serial.print(pos2[1]); Serial.print(" : "); Serial.println(pos2[2]);
  Serial.print("position3 = ");
  Serial.print(pos3[0]); Serial.print(" : ");
  Serial.print(pos3[1]); Serial.print(" : "); Serial.println(pos3[2]);
  Serial.print("Ks = "); Serial.println(ks);
  Serial.print("start third round ... ");
  statische_kennlinie(pos2[0]-max(25,(pos3[0]-pos1[0])*4),
                      pos2[0]+max(30,(pos3[0]-pos1[0])*4),
                      1, 300);
  Serial.print("position1 = ");
  Serial.print(pos1[0]); Serial.print(" : ");
  Serial.print(pos1[1]); Serial.print(" : "); Serial.println(pos1[2]);
  Serial.print("position2 = ");
  Serial.print(pos2[0]); Serial.print(" : ");
  Serial.print(pos2[1]); Serial.print(" : "); Serial.println(pos2[2]);
  Serial.print("position3 = ");
  Serial.print(pos3[0]); Serial.print(" : ");
  Serial.print(pos3[1]); Serial.print(" : "); Serial.println(pos3[2]);
  Serial.print("Ks = "); Serial.println(ks);
  Serial.print("start fourth round ... ");
  statische_kennlinie(pos2[0]-max(5,(pos3[0]-pos1[0])*2),
                      pos2[0]+max(5,(pos3[0]-pos1[0])*1),
                      1, 2000);
  Serial.print("position1 = ");
  Serial.print(pos1[0]); Serial.print(" : ");
  Serial.print(pos1[1]); Serial.print(" : "); Serial.println(pos1[2]);
  Serial.print("position2 = ");
  Serial.print(pos2[0]); Serial.print(" : ");
  Serial.print(pos2[1]); Serial.print(" : "); Serial.println(pos2[2]);
  Serial.print("position3 = ");
  Serial.print(pos3[0]); Serial.print(" : ");
  Serial.print(pos3[1]); Serial.print(" : "); Serial.println(pos3[2]);
  Serial.print("Ks = "); Serial.println(ks);
  adc->adc0->enableInterrupts(adc0_isr);
}
else if (c=='K') {  // Kennlinie Ausgabe für Diagramm
  adc->adc0->disableInterrupts();
  for (unsigned int i = 1900; i<2200; i=i+1) {
    analogWrite(writePin, i);
    delay(200);
```

```
      result = adc->readSynchronizedContinuous();
      result.result_adc0 = (uint16_t)result.result_adc0;
      result.result_adc1 = (uint16_t)result.result_adc1;
      Serial.print(i);
      Serial.print(";");
      Serial.print(result.result_adc0);
      Serial.print(";");
      Serial.println(result.result_adc1);
    }
    Serial.println("X");
    adc->adc0->enableInterrupts(adc0_isr);
  }
  else if (c=='t') {  // Sprungantwort nach Strejc (not working)
    Serial.println("\nAUSWERTUNG SPRUNGANTWORT NACH STREJC");
    Serial.print("vorbereiten ...");
    adc->stopSynchronizedContinuous();
  //  adc->adc0->disableInterrupts();
    delay(100);
    analogWrite(writePin, pos1[0]);
    delay(7500);
    strejc_aktiv = true;
    Serial.println(" sprung ...");
    digitalWrite(triggerPin, HIGH);
    analogWrite(writePin, pos3[0]);
    counter = 0;
 //    adc->adc0->enableInterrupts(adc0_isr);
    adc->startSynchronizedContinuous(readPinMain, readPinTotal);
    while (t90 == 0) {delay(100);}
    adc->stopSynchronizedContinuous();
//     adc->adc0->disableInterrupts();
    strejc_aktiv = false;
    Serial.print("t10 = "); Serial.println(t10);
    Serial.print("t20 = "); Serial.println(t20);
    Serial.print("t80 = "); Serial.println(t80);
    Serial.print("t90 = "); Serial.println(t90);
    tu = 1.048*t10-0.048*t90;
    tg = 0.455*(t90-t10);
    Serial.print("10/90: tu="); Serial.print(tu);
    Serial.print(" tg="); Serial.println(tg);
    tu = 1.161*t20-0.161*t80;
    tg = 0.721*(t80-t20);
    Serial.print("20/80: tu="); Serial.print(tu);
    Serial.print(" tg="); Serial.println(tg);
  //   delay(2000);
    digitalWrite(triggerPin, LOW);
  //   adc->adc0->enableInterrupts(adc0_isr);
    adc->startSynchronizedContinuous(readPinMain, readPinTotal);
   }
   else if (c=='o') {  // Einstellung nach Oppelt (not working)
    Serial.println("\nREGLEREINSTELLUNG NACH OPPELT");
    noInterrupts();
 //    Serial.println(tg);
 //    Serial.println(tu);
 //    Serial.println(ks);
    p_faktor = 0.8/ks*tg/tu;
    i_faktor = 2/3/tu;
    d_faktor = 0;
    Serial.print("P-Faktor: ");
```

```
    Serial.print(p_faktor*1000);    Serial.println(" x10^(-3)");
    Serial.print("I-Faktor: ");
    Serial.print(i_faktor*1000000); Serial.println(" x10^(-6)");
    interrupts();
  }
  else if (c=='n') {  // Newline
    Serial.println();
  }
  else if (c=='a') {  // Abtastfrequenz
    Serial.println("\nABTASTFREQUENZ MESSEN");
    noInterrupts();
    frequenzzeit = micros();
    frequenzcounter = 0;
    frequenzmessung = true;
    interrupts();
    while (frequenzmessung == true) {delay(50);}
    Serial.print("f = ");
    Serial.print((float)50000000/frequenzzeit); Serial.println("kHz");
    Serial.print("T = ");
    Serial.print((float)frequenzzeit/50000); Serial.println("µs");
  }
  else if (c=='r') {  // Rauschen
    Serial.println("\nRAUSCHEN");
    noInterrupts();
    Serial.print("betragsmäßig: ");
    Serial.println((float)standardabweichung/10);
    Serial.print("absolut: ");
    Serial.println((float)istwert_mittelwert/1000 - sollwert);
    interrupts();
  }
  else if (c=='S') { // Schwingmethode nach Ziegler und Nichols
    Serial.println("\nSCHWINGUNGSMETHODE NACH ZIEGLER UND NICHOLS");
    noInterrupts();
    i_faktor = 0;
    d_faktor = 0;
    p_faktor = max(0.00001, p_faktor*0.3);
    stelloffset = pos2[0];
    interrupts();
    delay(2000);
    noInterrupts();
    var_copy = 0;
    interrupts();
    while (var_copy < 5000) {      // für Selbsteinstellung
//  for (int i=0; i<175; i++) {    // für Diagramm
      noInterrupts();
      p_faktor = p_faktor * 1.01;
      interrupts();
      Serial.print(p_faktor*1000000);
      delay(100);
      noInterrupts();
      Serial.print(";");
      Serial.print(t_schwing);
      Serial.print(";");
      Serial.print(standardabweichung/10);
      Serial.print(";");
      Serial.println(t_schwingabweichung_mittelwert);
      var_copy = standardabweichung;
      interrupts();
```

```
  }
  Serial.println("fertig");
  noInterrupts();
  p_faktor = p_faktor * 0.6;
  i_faktor = p_faktor / (t_schwing / 100 * 0.5) *0.5;
  d_faktor = p_faktor * (t_schwing / 100 * 2) / 10 * 0.12;
  filter_parameter = 0.5;
  interrupts();
  Serial.print("P-Faktor: ");
  Serial.print(p_faktor*1000000);
  Serial.println(" x10^(-6)");
  Serial.print("I-Faktor: ");
  Serial.print(i_faktor*1000000);
  Serial.println(" x10^(-6)");
  Serial.print("D-Faktor: ");
  Serial.print(d_faktor*1000000);
  Serial.println(" x10^(-6)");
  Serial.print("Filter-Parameter: ");
  Serial.println(filter_parameter);
}
else if (c=='T') {  // Störgrößensprung
  Serial.println("\nSTÖRGRÖßENSPRUNG");
  stoersprung(75);
  Serial.print("Einregelzeit: ");
  Serial.println(mittelwert_einregelzeit);
  Serial.print("Überschwingweite: ");
  Serial.println(mittelwert_ueberschwingen);
  Serial.print("Zeitbeschwerte Regelfläche: ");
  Serial.println(mittelwert_regelflaeche);
}
else if (c=='0') {
  Serial.println("\nOFF");
  noInterrupts();
  p_faktor = 0;
  i_faktor = 0;
  p_faktor = 0;
  interrupts();
}
else if (c=='P') {   // P-Anteil Erhöhen
  noInterrupts();
  p_faktor = p_faktor * 1.02;
  var_copy = p_faktor*1000000;
  interrupts();
  Serial.print("P-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='p') {   // P-Anteil Senken
  noInterrupts();
  p_faktor = p_faktor * 0.98;
  var_copy = p_faktor*1000000;
  interrupts();
  Serial.print("P-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='I') {   // I-Anteil Erhöhen
  noInterrupts();
```

```
  i_faktor = i_faktor * 1.02;
  var_copy = i_faktor*1000000;
  interrupts();
  Serial.print("I-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='i') {   // I-Anteil Senken
  noInterrupts();
  i_faktor = i_faktor * 0.98;
  var_copy = i_faktor*1000000;
  interrupts();
  Serial.print("I-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='D') {   // D-Anteil Erhöhen
  noInterrupts();
  d_faktor = d_faktor * 1.02;
  var_copy = d_faktor*1000000;
  interrupts();
  Serial.print("D-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='d') {   // D-Anteil Senken
  noInterrupts();
  d_faktor = d_faktor * 0.98;
  var_copy = d_faktor*1000000;
  interrupts();
  Serial.print("D-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='F') {   // Tiefpass-Anteil Erhöhen
  noInterrupts();
  filter_parameter = filter_parameter * 1.02;
  var_copy = filter_parameter*1000000;
  interrupts();
  Serial.print("F-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='f') {   // Tiefpass-Anteil Senken
  noInterrupts();
  filter_parameter = filter_parameter * 0.98;
  var_copy = filter_parameter*1000000;
  interrupts();
  Serial.print("F-Faktor: ");
  Serial.print(var_copy);
  Serial.println(" x10^(-6)");
}
else if (c=='O') {    // Optimierung
  Serial.println("\nOPTIMIERUNG");
  stoersprung(75);
  // P-FAKTOR
  do {
    last_value = mittelwert_regelflaeche;
```

```
  noInterrupts();
  p_faktor = p_faktor * (1+optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("+P "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
p_faktor = p_faktor * (1-optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("-P "); Serial.println(mittelwert_regelflaeche);
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  p_faktor = p_faktor * (1-optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("-P "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
p_faktor = p_faktor * (1+optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("+P "); Serial.println(mittelwert_regelflaeche);

// I-FAKTOR
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  i_faktor = i_faktor * (1+optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("+I "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
i_faktor = i_faktor * (1-optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("-I "); Serial.println(mittelwert_regelflaeche);
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  i_faktor = i_faktor * (1-optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("-I "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
i_faktor = i_faktor * (1+optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("+I "); Serial.println(mittelwert_regelflaeche);

// D-FAKTOR
```

```
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  d_faktor = d_faktor * (1+optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("+D "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
d_faktor = d_faktor * (1-optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("-D "); Serial.println(mittelwert_regelflaeche);
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  d_faktor = d_faktor * (1-optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("-D "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
d_faktor = d_faktor * (1+optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("+D "); Serial.println(mittelwert_regelflaeche);

// TIEFPASS-PARAMETER
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  filter_parameter = min(1, filter_parameter * (1+optimierungsschritt));
  interrupts();
  stoersprung(75);
  Serial.print("+F "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
noInterrupts();
filter_parameter = filter_parameter * (1-optimierungsschritt);
interrupts();
stoersprung(75);
Serial.print("-F "); Serial.println(mittelwert_regelflaeche);
do {
  last_value = mittelwert_regelflaeche;
  noInterrupts();
  filter_parameter = filter_parameter * (1-optimierungsschritt);
  interrupts();
  stoersprung(75);
  Serial.print("-F "); Serial.println(mittelwert_regelflaeche);
} while (last_value > mittelwert_regelflaeche);
last_value = mittelwert_regelflaeche;
noInterrupts();
filter_parameter = min(1, filter_parameter * (1+optimierungsschritt));
interrupts();
stoersprung(75);
Serial.print("+F "); Serial.println(mittelwert_regelflaeche);
```

```
      Serial.print("P-Faktor: ");
      Serial.print(p_faktor*1000000);
      Serial.println(" x10^(-6)");
      Serial.print("I-Faktor: ");
      Serial.print(i_faktor*1000000);
      Serial.println(" x10^(-6)");
      Serial.print("D-Faktor: ");
      Serial.print(d_faktor*1000000);
      Serial.println(" x10^(-6)");
      Serial.print("Filter-Parameter: ");
      Serial.println(filter_parameter);
    }
  }
}
```

```
void adc0_isr(void) {
  result = adc->readSynchronizedContinuous();
  messwert1 = (uint16_t)result.result_adc0;
  messwert2 = (uint16_t)result.result_adc1;
  /* // Messbereichsumschaltung
  if (1000 < messwert1 and messwert1 < 64000) {
   istwert = messwert1;
  } else{
   messwert2 = (int32_t) messwert2;
   istwert = (messwert2 - pos1[2]) * ((pos3[1]-pos1[1]) / (pos3[2]-pos1[2])) +
pos1[1];
  }
  */
  istwert = messwert1;

  if (!strejc_aktiv) {
    counter++;
    analogWrite(writePin, (uint16_t)stellwert);
    istwert_mittelwert = istwert*0.05 + istwert_mittelwert*0.99995;
    istwert_tiefpass_d = istwert*filter_parameter +
                         istwert_tiefpass_d*(1-(filter_parameter/10));
    istwert_tiefpass_10hz = istwert*0.01696717 + istwert_tiefpass_10hz*0.9983;
    differenz = sollwert-istwert;
    differenz_halb = differenz >> versatz_i_summe;
    aenderung_tiefpass = istwert_tiefpass_d_last - istwert_tiefpass_d;
    abweichung = (istwert_mittelwert/100)-(istwert*10);
  //  abweichung = (istwert_mittelwert/100)-istwert_tiefpass_10hz;
    standardabweichung = standardabweichung * 0.998 + abs(abweichung) * 0.002;


    if (counter < 6000) {
      if (4294967295 - (counter * abs(abweichung)*0.01) >= regelflaeche) {
        regelflaeche = regelflaeche + (counter * abs(abweichung)*0.01);
      } else {
        regelflaeche = 4294967295;
      }
    }
```

```
    t_null = t_null + 1;
    if ((istwert>(istwert_mittelwert/1000)) !=
(istwert_last>(istwert_mittelwert/1000))) {
     // Serial.println("x");
      if (t_null > 10) {
        //Serial.println(t_null);
        t_null = t_null * 100;
        t_schwing = t_schwing*0.95 + t_null*0.05;
        t_schwingabweichung_aktuell = t_null-t_schwing;
        t_schwingabweichung_mittelwert = t_schwingabweichung_mittelwert*0.98 +
                                         abs(t_schwingabweichung_aktuell)*0.02;
        t_null = 0;
      }
    }

    digitalWriteFast(LEDrange, (messwert1 < 10000 or messwert1 > 55536));

    if (abs(abweichung) > 5000) {
      einregelzeit = counter;
    } if (abs(abweichung)/10 > ueberschwingen) {
      ueberschwingen = abs(abweichung)/10;
    }

    if (differenz_halb > 0) {
      if ((2147483647 - differenz_halb) < i_summe) {
        i_summe = 2147483647;
        digitalWriteFast(LEDoverflow, HIGH);
      } else {
        i_summe = i_summe + differenz_halb;
        digitalWriteFast(LEDoverflow, LOW);
      }
    } else {
      if ((-2147483648 - differenz_halb) > i_summe) {
        i_summe = -2147483648;
        digitalWriteFast(LEDoverflow, HIGH);
      } else {
        i_summe = i_summe + differenz_halb;
        digitalWriteFast(LEDoverflow, LOW);
      }
    }


    if (frequenzmessung == true) {
      frequenzcounter++;
      if (frequenzcounter == 50000) {
        frequenzzeit = micros()-frequenzzeit;
        frequenzmessung = false;
      }
    }


    stellwert = stelloffset +
                quantisierungfehler +
                p_faktor*differenz +
                i_summe*i_faktor*(1<<versatz_i_summe) +
                aenderung_tiefpass*d_faktor ;
    if (stellwert<0)    {stellwert=0;}
    if (stellwert>4095) {stellwert=4095;}
```

```
    quantisierungfehler = stellwert - (uint16_t)stellwert;

  } else {
    counter++;
    if (t10==0 and istwert > pos1[1]+0.1*(pos3[1]-pos1[1])) {
      t10 = counter;
    } if (t20==0 and istwert > pos1[1]+0.2*(pos3[1]-pos1[1])) {
      t20 = counter;
    } if (t80==0 and istwert > pos1[1]+0.8*(pos3[1]-pos1[1])) {
      t80 = counter;
    } if (t90==0 and istwert > pos1[1]+0.9*(pos3[1]-pos1[1])) {
      t90 = counter;
    } if (counter >= 1000000) {
      // TIMEOUT
      if (t90 == 0) {
        t90 = counter;
        if (t80 == 0) {
          t80 = counter;
        }
      }
      adc->stopSynchronizedContinuous();
  //    adc->adc0->disableInterrupts();
    }
  }
  istwert_last = istwert;
  istwert_tiefpass_d_last = istwert_tiefpass_d;
}
```

6.6.2 Auswertungsprogramme Python

usb_auswertung_optimierung.py

```
# sendet Wägezelle Befehle, um sich zunächst nach der Faustformel einzustellen
# und sich danach zu optimieren
# zeichnet dabei nach jedem Schritt Reglerparameter, Zeit und Regelgüte auf

import serial
import time

filename = 'optimierungsverlauf13.csv'

startzeit = time.time()
durchgangszaehler = 0
p_faktor = '0'
i_faktor = '0'
d_faktor = '0'
f_faktor = '0'

print('P-Faktor', 'I-Faktor', 'D-Faktor',
      'Durchgang', 'Regelflaeche', sep='\t')
try:
    s = serial.Serial('COM4', baudrate=112500)
    s.write(b'kST')
    while True:
        ergebnis = s.readline().decode('UTF-8').split('\r\n')[0].split(': ')
        if ergebnis[0] == 'P-Faktor':
            p_faktor = ergebnis[1].split(' x10^(-6)')[0]
```

```
        elif ergebnis[0] == 'I-Faktor':
            i_faktor = ergebnis[1].split(' x10^(-6)')[0]
        elif ergebnis[0] == 'D-Faktor':
            d_faktor = ergebnis[1].split(' x10^(-6)')[0]
        elif ergebnis[0] == 'Filter-Parameter':
            f_faktor = ergebnis[1]
        elif ergebnis[0] == 'Zeitbeschwerte Regelfläche':
            ergebnis = ergebnis[1]
            print(p_faktor, i_faktor, d_faktor, f_faktor, sep='\t', end='\t')
            print(durchgangszaehler, ergebnis, sep='\t')
            with open(filename,'a') as file:
                file.write('\n'+str(time.time()-startzeit))
                file.write(';'+str(durchgangszaehler)+';'+ergebnis)
                file.write(';'+p_faktor+';'+i_faktor)
                file.write(';'+d_faktor+';'+f_faktor)
                file.close()
            durchgangszaehler += 1
            s.write(b'OT')
except KeyboardInterrupt:
    s.close()
except SerialException:
    print('connection failed')
```

plot_optimierung_mit_parametern.py

```
# Stellt von usb_auswertung_optimierung.py generierte csv-Dateien dar

import matplotlib.pyplot as plt

fig, axs = plt.subplots(3, 2)
max_points = 150

files = ['optimierungsverlauf9.csv',
         'optimierungsverlauf10.csv',
         'optimierungsverlauf11.csv',
         'optimierungsverlauf12.csv']

for filename in files:
    file  = open(filename)
    daten = file.read().split('\n')[1:]
    file.close()

    points_counter = []
    points_time    = []
    points_p_faktor= []
    points_d_faktor= []
    points_i_faktor= []
    points_f_faktor= []
    points_flaeche = []

    for i in daten:
 #       print(i)
        i = i.split(';')
        points_time.append(float(i[0])/60)
        points_counter.append(int(i[1]))
        points_flaeche.append(int(i[2])*0.0000029541)
        points_p_faktor.append(float(i[3])/1000000)
        points_i_faktor.append(float(i[4])/1000000)
```

```
        points_d_faktor.append(float(i[5])/1000000)
        points_f_faktor.append(float(i[6]))

    axs[2,0].plot(points_counter[:max_points], points_flaeche[:max_points])
    axs[0,1].plot(points_counter[:max_points], points_p_faktor[:max_points])
    axs[2,1].plot(points_counter[:max_points], points_i_faktor[:max_points])
    axs[1,1].plot(points_counter[:max_points], points_d_faktor[:max_points])
    axs[0,0].plot(points_counter[:max_points], points_f_faktor[:max_points])
    axs[1,0].plot(points_counter[:max_points], points_time[:max_points])

axs[2,0].grid()
axs[0,1].grid()
axs[1,0].grid()
axs[1,1].grid()
axs[0,0].grid()
axs[2,1].grid()

axs[2,0].set(xlabel='Optimierungsdurchlauf', ylabel='Regelfläche in mVs')
axs[0,1].set(xlabel='Optimierungsdurchlauf', ylabel='P-Faktor')
axs[2,1].set(xlabel='Optimierungsdurchlauf', ylabel='I-Faktor')
axs[1,1].set(xlabel='Optimierungsdurchlauf', ylabel='D-Faktor')
axs[0,0].set(xlabel='Optimierungsdurchlauf', ylabel='Filter-Parameter')
axs[1,0].set(xlabel='Optimierungsdurchlauf', ylabel='Zeit in min')

plt.show()
```

oszilloskop_plots.py

```
# Stellt Messdaten des Oszilloskops dar
# mehrere Messdurchläufe werden gemittelt und gefiltert

import matplotlib.pyplot as plt

datenpunkte = 614
folder = './oszilloskop/'

files= [['sprung_ziegler1_1kS_aufnahme'  +str(i)+'.csv' for i in range(1,10)],
        ['sprung_analog1_1kS_aufnahme'   +str(i)+'.csv' for i in range(1,10)],
        ['sprung_optimiert1_1kS_aufnahme'+str(i)+'.csv' for i in range(1,13)]]
labels = ['Faustformel Ziegler & Nichols',
          'Analog Werkseinstellung',
          'Optimiert']
offset = [0, 2, -1]

def average(liste):
    return sum(liste)/len(liste)
def filtern(liste):
    for j in range(2):
        liste = [average(liste[max(0,i-1):i+2]) for i in range(len(liste))]
    return liste

def read_files(files):
    xpoints = [0 for i in range(datenpunkte)]
    ypoints = [0 for i in range(datenpunkte)]
    for file in files:
        file = open(folder+file)
        data = file.read()
        file.close()
```

```
        data = data.split('\n')[3:-1]
        for i in range(datenpunkte):
            data[i] = data[i].split(';')
            xpoints[i] += float(data[i][0].replace(',','.'))
            ypoints[i] += float(data[i][1].replace(',','.'))
    xpoints = [i/len(files) for i in xpoints]
    ypoints = [i/len(files) for i in ypoints]
    return xpoints, ypoints

xpoints = [None for i in range(len(files))]
ypoints = [None for i in range(len(files))]
for i in range(len(files)):
    xpoints[i], ypoints[i] = read_files(files[i])
    for j in range(len(ypoints[i])):
        ypoints[i][j] += offset[i]

for i in range(len(files)):
    plt.plot(xpoints[i], filtern(ypoints[i]),  label=labels[i])
plt.xlabel("Zeit in ms")
plt.ylabel("Regelgröße in mV")
plt.legend(loc='best')
plt.title("Störgrößensprung")
plt.grid()
plt.show()
```

auswertung_fastmode.py

```
# Ermittelt die Standardabweichung in Messdaten von Fastmode

folder = './Fastmode 10.05.2022/'
filenames = ['Schaltung 1 h4 8Hz.txt',
             'Schaltung 2 h4 8Hz.txt',
             'Schaltung 3 h4 8Hz.txt',
             'Analogregler h4 8Hz.txt']

for filename in filenames:
    print(filename, end='\t')
    with open(folder+filename) as file:
        data = file.read()
        file.close()
    data = data.split('\n')
    messwerte = []
    mode = False
    for line in data:
        if mode:
            line = line.split('\t')
            if len(line) == 4:
                line = float(line[0].replace(',','.'))
                messwerte.append(line)
        elif line == 'Messwert   Zeit in ms   Trigger      Kanal':
            mode = True
    messwerte = messwerte[10:]
    average = sum(messwerte)/len(messwerte)
    standardabweichung = 0
    for wert in messwerte:
        standardabweichung += abs(wert-average)
    standardabweichung /= len(messwerte)
    print(standardabweichung)
```

usb_auswertung_plot_statische_kennlinie.py

```
# sendet Befehl zum Ausmessen einer statischen Kennlinie an die Wägezelle,
# zeichnet die Antwort auf und stellt die Kennlinie dar

import serial
import matplotlib.pyplot as plt

xpoints = []
y1points = []
y2points = []

try:
    file = input('Filename:  ')
    file = open(file+'.csv','w')
    s = serial.Serial('COM4', baudrate=112500)
    s.write(b'K')
 #   s.reset_input_buffer()
    ergebnis = None
    while ergebnis != 'X':
        ergebnis = s.readline().decode('UTF-8').split('\r\n')[0]
        print(ergebnis)
        file.write(ergebnis+'\n')
        if ergebnis != 'X':
            ergebnis = ergebnis.split(';')
            xpoints.append(int(ergebnis[0]))
            y1points.append(int(ergebnis[1]))
            y2points.append(int(ergebnis[2]))
except KeyboardInterrupt:
    s.close()
s.close()

file.close()

plt.plot(xpoints, [((i*0.72/2**16)-0.36) for i in y1points],
label='Eingangsschaltung 3')
plt.plot(xpoints, [((i*15.1/2**16)-7.56) for i in y2points],
label='Eingangsschaltung 1')
plt.title("Statische Kennline")
plt.xlabel("Stellwert in DAC-Punkten")
plt.ylabel("Regelwert in V")
plt.legend(loc='best')
plt.grid()
plt.show()
```

usb_auswertung_plot_schwingung.py

```
# Sendet Befehl für Schwingungsmethode nach Ziegler und Nichols an Wägezelle,
# zeichnet Antwort auf und stellt verschiedene Kriterien zur
Schwingungserkennung dar.

import serial
import matplotlib.pyplot as plt

xpoints = []
ypoints1= []
ypoints2= []
ypoints3= []
```

```
try:
    file = input('Filename:  ')
    file = open(file+'.csv','w')
    s = serial.Serial('COM4', baudrate=112500)
    s.write(b'S')
 #   s.reset_input_buffer()
    ergebnis = None
    while ergebnis != 'fertig':
        ergebnis = s.readline().decode('ASCII').split('\r\n')[0]
        print(ergebnis)
        if ergebnis.count(';') == 3:
            file.write(ergebnis+'\n')
            ergebnis = ergebnis.split(';')
            xpoints.append(float(ergebnis[0])/1000000)
            ypoints1.append(int(ergebnis[1]))
            ypoints2.append(int(ergebnis[2]))
            ypoints3.append(int(ergebnis[3]))
except KeyboardInterrupt:
    s.close()
    file.close()
s.close()
file.close()


xpoints  = xpoints[10:]
ypoints1 = [(i*0.0005334) for i in ypoints1[10:]]
ypoints2 = [(i*0.0110780) for i in ypoints2[10:]]
ypoints3 = [(i*0.0005334) for i in ypoints3[10:]]

plt.plot(xpoints, ypoints2, color='blue', label='Standardabweichung
Regelgröße')
plt.xlabel("P-Faktor")
plt.ylabel("Standardabweichung in mV")
plt.legend(loc='upper left')
plt.twinx()
plt.plot(xpoints, ypoints1, color='red', label='Schwingungsdauer')
plt.plot(xpoints, ypoints3, color='orange', label='Abweichung
Schwingungsdauer')
plt.ylabel("Schwingsungsdauer in ms")
plt.legend(loc='center right')
plt.title("Schwingungsmethode nach Ziegler und Nichols")
plt.show()
```