RAVENSBURG WEINGARTEN UNIVERSITY

BACHELORARBEIT

Entwicklung einer KI-basierten Lösung zur Temperaturregelung eines Wasserbeckens

Autor:
Andreas HARDER

Betreuer:
Prof. Dr.-Ing. Lothar BERGER
Jürgen KNOLL

Diese Abschlussarbeit wurde zur Erlangung des Grades „Bachelor of Engineering“ an der Ravensburg Weingarten University in Zusammenarbeit mit der Firma Uhltronix GmbH eingereicht.

Fakultät für Elektrotechnik und Informatik

22. Mai 2024

RAVENSBURG WEINGARTEN UNIVERSITY

Zusammenfassung

Fakultät für Elektrotechnik und Informatik

Bachelor of Engineering

Entwicklung einer KI-basierten Lösung zur Temperaturregelung eines Wasserbeckens

von Andreas HARDER

In dieser Bachelorarbeit wird die Anwendung von Künstliche Intelligenz (KI) in der Regelungstechnik untersucht, wobei der Fokus auf der Entwicklung und Implementierung von KI-basierten Reglern für die Temperaturregelung eines Wasserbeckens liegt. Das Ziel ist es, das Potenzial des Maschinellen Lernens zur effizienten Bewältigung einfacher Regelungsaufgaben zu erforschen. Im Rahmen der Arbeit werden zwei KI-basierte Regler entwickelt:

- Ein Regler, der aus einem Long Short-Term Memory (LSTM) besteht und darauf trainiert wird, das Verhalten eines vorher entwickelten Proportional-Integral-Derivative (PID)-Reglers zu erlernen und nachzuahmen.
- Ein Model Predictive Control (MPC)-Regler welcher als Vorhersagemodell ein Feedforward Neural Network (FFNN) nutzt. Dieses Netzwerk wird darauf trainiert, das Systemverhalten auf Basis von historischen Betriebsdaten vorherzusagen.

Die Arbeit dokumentiert die Schritte von der Konzeption über die Implementierung bis hin zu Tests und Bewertung der entwickelten Regler. Zur Bewertung der Regler werden diese mit einem herkömmlichen PID-Regler sowie einem konventionellen MPC-Regler verglichen, die ebenfalls im Rahmen dieser Arbeit entwickelt werden.

Die Ergebnisse zeigen, dass die KI-basierten Regler in der Lage sind, die Temperatur des Wasserbeckens mit nur geringfügig schlechterer Leistung als konventionelle Regler zu regeln. Insbesondere der MPC-Regler, der ein datengetriebenes Vorhersagemodell verwendet, zeigt Potenzial für zukünftige Anwendungen, bei denen das Systemverhalten nicht vollständig bekannt ist. Allerdings bringen die KI-basierten Ansätze auch neue Herausforderungen mit sich, wie in dieser Arbeit aufgezeigt wird. Dazu gehören hohe Anforderungen an die Rechenleistung und die Notwendigkeit einer sorgfältigen Parameterauswahl.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Symbole

Symbol	Beschreibung	Einheit
$w(t)$	Führungsgröße (Sollwert)	°C
$y(t)$	Regelgröße (Istwert)	°C
$e(t)$	Regelabweichung	°C
$u(t)$	Stellgröße	%
$d(t)$	Störgröße	
T_d	Totzeit	s
T_s	Abtastzeit	s

Kapitel 1

Einleitung

1.1 Firmenprofil

Die Uhltronix GmbH wurde 2006 von Joachim und Markus Uhl gegründet. Sie hat ihren Sitz in Unterweiler, einer Teilgemeinde von Ostrach (Baden-Württemberg). Das Unternehmen spezialisiert sich auf individuelle Automatisierungslösungen im Maschinenbau und in der Industrie. Mit etwa 15 Mitarbeitern bietet die Uhltronix GmbH ein breites Spektrum an Dienstleistungen von Steuerungsprogrammierung und Visualisierung bis hin zu Elektrokonstruktion und Schaltschrankbau [54].

1.1.1 Abteilung Software-Programmierung

Die primäre Expertise der Firma liegt in der Softwareentwicklung, wobei das Team sich auf zwei Hauptgebiete fokussiert.

1. **SPS-Programmierung:** Hier arbeitet das Team vorrangig mit Siemens und Beckhoff Steuerungen, entwickelt jedoch auch Lösungen mit anderen Systemen. Das Spektrum reicht von der Eigenentwicklung kleinerer Anlagen bis hin zur Unterstützung anderer Maschinenbauer bei umfangreichen Industrieanlagen und Produktionslinien, im Bereich des Sondermaschinenbaus.

2. **Hochsprachenprogrammierung:** Im Bereich der Hochsprachenprogrammierung konzentriert sich das Team auf die Umsetzung von individuellen Softwarelösungen entsprechend den Kundenwünschen. Der Fokus liegt hierbei auf der Entwicklung von Energiemanagement- oder Prozessleitsystemen. Diese maßgeschneiderten Softwarelösungen fungieren häufig als ein zentrales System, welches als Schnittstelle zwischen Enterprise-Resource-Planning (ERP)-Systemen und den verschiedenen Maschinen einer Produktionshalle dient. Ein konkretes Beispiel für ein solches Endprodukt ist eine Software, welche die Visualisierung aller Maschinen in einer Produktionshalle übernimmt und gleichzeitig Prozessdaten aufzeichnet und analysiert.

1.2 Motivation und Relevanz des Themas

In der modernen Technik und Wissenschaft ist eine präzise Reglung von Systemen von grundlegender Bedeutung. Regelungstechnik spielt eine zentrale Rolle in zahlreichen Anwendungen der Industrie, Forschung und Technologie. Obwohl sich diese Arbeit auf die Temperaturreglung eines Wasserbeckens konzentriert, ist sie repräsentativ für ein breites Spektrum an Regelungsproblemen. Die Erkenntnisse und Methoden, die in dieser Arbeit entwickelt werden, haben das Potenzial, über

den spezifischen Anwendungsfall einer Temperaturreglung hinaus auf andere Regelungsaufgaben übertragen zu werden.

Die Relevanz des Themas ergibt sich aus dem Bedarf an verbesserten Regelungsstrategien. Nahezu jede Branche, von der Automobilindustrie, bis hin zur Medizintechnik profitiert von fortschrittlicheren, intelligenten und effizienten Regelungen. Insbesondere die Integration von künstlicher Intelligenz (KI) bietet das Potenzial, traditionelle Regelungsansätze zu revolutionieren und neue Maßstäbe in Bezug auf Leistung, Anpassungsfähigkeit und Effizienz zu setzen.

Es gibt viele verschiedene Methoden zur Durchführung einer Regelung. Abhängig von den Eigenschaften des zu regelnden Systems können bestimmte Regelverfahren besser geeignet sein als andere. Die Auswahl und Feinabstimmung eines geeigneten Verfahrens erfordert fundierte Kenntnisse in der Regelungstechnik und ein Verständnis des Systemverhaltens. Der PID-Regler, der wegen seiner Einfachheit und Effektivität für lineare Systeme weit verbreitet ist, erfordert oft einen aufwendigen Einstellungsprozess, der durch umfangreiche experimentelle Tests und Anpassungen gekennzeichnet ist. Theoretisch können die Regelparameter auch aus mathematischen Modellen abgeleitet werden, was jedoch insbesondere bei komplexen Systemen sehr aufwendig ist und spezielles Fachwissen erfordert. Darüber hinaus stößt der PID-Regler bei anspruchsvollen Systemen mit Nichtlinearitäten oder schwankenden externen Bedingungen an seine Grenzen, wodurch die Leistungsfähigkeit leidet oder die Regelung sogar unmöglich wird.

Hier setzt die Motivation dieser Arbeit an. Es geht darum, KI-basierte Ansätze in die Regelungstechnik zu integrieren, welche den Regelprozess verbessern oder den Einstellvorgang des Reglers vereinfachen. Der Fokus liegt darauf, zu erforschen, wie herkömmliche Verfahren mittels KI optimiert oder gar ersetzt werden können. Eine solche Möglichkeit, könnte Regelungstechnik nicht nur präziser und effizienter machen, sondern auch einfacher in der Anwendung, und somit für ein breiteres Publikum zugänglich. Dadurch eröffnen sich neue Wege für fortschrittliche Anwendungen in Industrie und Forschung.

1.3 Zielsetzung und Forschungsfragen

Zielsetzung:

Die Zielsetzung dieser Bachelorarbeit konzentriert sich auf die Erforschung des Potenzials von maschinellem Lernen (ML) in der Regelungstechnik, insbesondere in Bezug auf einfache Regelungsaufgaben, wie sie oft von Kunden gestellt werden. Diese Aufgaben zeichnen sich dadurch aus, dass eine zufriedenstellende Lösung in kurzer Zeit realisiert werden muss. Dabei steht oftmals nicht die optimale Reglung im Vordergrund, da eine genaue Systemidentifizierung und die Auslegung des entsprechenden Reglers einiges an Expertenwissen benötigt, sehr zeitintensiv und somit unwirtschaftlich sein kann. Der Fokus liegt eher auf der Implementierung eines funktionsfähigen, wenn auch nicht optimalen Reglers, der mit überschaubarem Zeitaufwand realisiert werden kann.

Ein wesentliches Anliegen dieser Arbeit ist es, zu beurteilen, inwiefern ML-basierte Regler für derartige Problemstellungen geeignet sind. Die Grundidee basiert auf der Vermutung, dass ein ML-basierter Regler für einen Automatisierungstechniker ohne

tiefgreifende Kenntnisse in der Regelungstechnik eventuell einfacher zu implementieren sein könnte als ein herkömmlicher Regler. Vor allem, wenn das Systemverhalten der Regelstrecke ohne weiteres nicht bekannt ist. Ziel ist es deshalb, herauszufinden, ob ML-basierte Ansätze eine effiziente und praktikable Lösung für alltägliche Regelungsprobleme darstellen können.

In dieser Arbeit liegt der Schwerpunkt nicht auf der Entwicklung eines marktreifen Produkts, sondern vielmehr auf der grundlegenden Erforschung des Potenzials dieser Technologien am Beispiel der Temperaturregelung eines Wasserbeckens. Diese Untersuchung dient als Grundlage für zukünftige Forschungen, die darauf abzielen, ML in verschiedene regelungstechnische Anwendungen zu integrieren. Ein langfristiges Ziel könnte beispielsweise die Entwicklung eines universell einsetzbaren Reglers sein, der verschiedene Regelaufgaben bewältigen kann. Diese Arbeit soll daher ein tieferes Verständnis für die Einsatzmöglichkeiten und die Grenzen von ML in der Regelungstechnik schaffen.

Forschungsfragen:

Die folgenden Forschungsfragen sollen im Rahmen dieser Bachelorarbeit beantwortet werden:

1. Wie effizient ist ein KI-basierter Regler im Vergleich zu herkömmlichen Reglern bei der Temperaturregelung eines Wasserbeckens?
2. Ist es möglich, einen KI-basierten Regler so zu gestalten, dass er hinsichtlich Bedienung, Implementierung und Nutzerfreundlichkeit auch für Anwender ohne vertiefte technische Kenntnisse zugänglich ist?
3. Welche Herausforderungen und Hürden treten bei der Entwicklung und Implementierung des KI-basierten Reglers auf, und wie können diese überwunden werden?

1.4 Aufbau und Gliederung dieser Arbeit

Dieses Kapitel dient dazu, einen strukturierten Überblick über die vorliegende Arbeit zu geben und den Leser durch die verschiedenen Phasen und Inhalte zu führen. Die Gliederung der Arbeit ist wie folgt aufgebaut:

1. **Einleitung:** Dieses Kapitel setzt den Rahmen für das Thema der Arbeit, erläutert die Motivation und die zentrale Zielsetzung sowie die daraus abgeleiteten Forschungsfragen.
2. **Grundlagen und theoretischer Rahmen:** In diesem Kapitel werden die notwendigen Grundlagen und theoretischen Konzepte erläutert, die für das Verständnis der Arbeit wichtig sind. Dazu gehören die Grundlagen der Regelungstechnik, der künstlichen Intelligenz sowie spezifische Informationen zu den verwendeten Methoden und Technologien.
3. **Stand der Technik:** Dieses Kapitel gibt einen Überblick über die aktuellen Entwicklungen und bestehenden Lösungen im Bereich der KI in der Regelungstechnik.

4. **Auswahl der zu erprobenden KI-Methoden:** Es wird erläutert, welche KI-Methoden ausgewählt wurden, um die definierten Ziele zu erreichen und warum diese Auswahl getroffen wurde.

5. **Projektimplementierung und Identifizierung der Regelstrecke:** Dieses Kapitel beschreibt die Rahmenbedingungen des Projektes, wie z.B. die verwendete Hardware oder die Programmierumgebung und befasst sich mit der Identifikation der Regelstrecke.

6. **Bewertungsmethodik für Regelungsmethoden:** Die Methoden zur Bewertung der entwickelten Regler werden definiert und beschrieben.

7. **Design und Implementierung von Reglern:** Die Entwicklung und praktische Implementierung der verschiedenen Regler wird ausführlich dargestellt. Außerdem werden die entwickelten Regler getestet und bewertet. Im Rahmen des Projekts werden insgesamt vier verschiedene Reglertypen entwickelt und analysiert:

    - **PID-Regler:** Dieser klassische Regler dient als Benchmark, um die Leistung der KI-basierten Regler zu vergleichen.
    - **Konventioneller MPC-Regler:** Dieser Regler verwendet ein mathematisches Vorhersagemodell. Er dient als Grundlage und Vergleichsbasis für weiterentwickelte MPC-Versionen.
    - **Neuronales Netzwerk als PID-Ersatz:** Ein Regler, bei dem ein neuronales Netzwerk die Regelstrategie eines PID-Reglers erlernt und diesen vollständig ersetzt.
    - **MPC-Regler mit neuronalem Netzwerk als Vorhersagemodell:** Dieser Regler nutzt ein neuronales Netzwerk zur Vorhersage des Systemverhaltens.

8. **Zusammenfassung und Ausblick:** Das letzte Kapitel fasst die wichtigsten Erkenntnisse zusammen und gibt einen Ausblick auf mögliche zukünftige Forschungen in diesem Bereich.

9. **Anhänge:** Der Anhang enthält zusätzliche Informationen und Materialien, die zur Vertiefung oder zum besseren Verständnis der Arbeit beitragen.

10. **Literaturverzeichnis:** Alle in der Arbeit zitierten Quellen werden hier aufgeführt.

Kapitel 2

Grundlagen und theoretischer Rahmen

In diesem Kapitel werden gewisse Grundlagen und theoretischen Konzepte betrachtet, welche für das Verständnis dieser Arbeit von Vorteil sind. Es ist jedoch zu beachten, dass eine umfassende Abhandlung aller notwendigen Grundlagen den Rahmen dieser Bachelorarbeit sprengen würde. Für ein vollständiges Verstehen aller Details und Feinheiten der Arbeit, wird daher vorausgesetzt, dass der Leser bereits ein gewisses Vorwissen in diesen Bereichen mitbringt.

2.1 Regelungstechnik

Die Regelungstechnik ist ein fundamentaler Aspekt der Automatisierungstechnik und spielt eine entscheidende Rolle in vielen technischen Systemen. Dieser Abschnitt fokussiert sich auf die Klärung und einheitliche Darstellung der grundlegenden Begriffe und Prinzipien, die in der Regelungstechnik verwendet werden.

2.1.1 Grundstruktur eines Regelkreises

Unter einer Regelung versteht man ein Vorgang, bei dem eine Größe (Regelgröße) fortlaufend erfasst, mit einem gewünschten Sollwert (Führungsgröße) verglichen und bei Abweichungen automatisch korrigiert wird. Dadurch ergibt sich ein geschlossenes System, welches man als Regelkreis bezeichnet. Dieser besteht aus mehreren Komponenten, die zusammenarbeiten, um das gewünschte Ergebnis zu erzielen [33, S. 1 ff.]. In Abbildung 2.1 ist die Grundstruktur eines Regelkreises aufgezeigt. Anschließend werden die wichtigsten darin vorkommenden Begriffe erläutert [33, S. 4 f.].

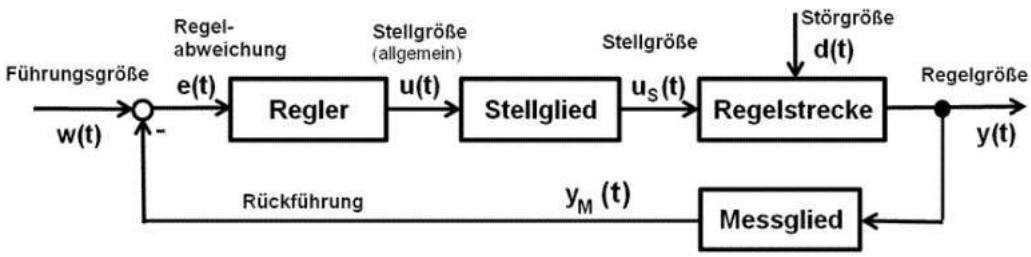


ABBILDUNG 2.1: Grundstruktur eines Regelkreises [69]

- **Führungsgröße (Sollwert) $w(t)$:** Der gewünschte Wert, den die Regelgröße erreichen soll.

- **Regelgröße (Istwert) $y(t)$:** Die zu regelnde Größe im System, die gemessen und auf den gewünschten Wert gebracht werden soll.
- **Regelabweichung $e(t)$:** Die Differenz zwischen Führungsgröße und Regelgröße. Ziel des Reglers ist es, diese Abweichung zu minimieren.
  $e(t) = w(t) - y(t)$
- **Regler:** Der Regler ist dafür zuständig, anhand der Regelabweichung die Stellgröße $u(t)$ so vorzugeben, dass das Regelungsziel erreicht wird.
- **Stellglied/Aktor:** Das Stellglied setzt die Stellgröße $u(t)$ welche der Regler vorgibt, in einen am Prozess wirksamen Wert $u_s(t)$ um. Zum Beispiel könnte das Stellglied ein analoges Ventil sein. Dann wäre $u(t)$ ein Spannungssignal von $0 - 10$ V und die Stellgröße $u_s(t)$ nach dem Ventil dann der tatsächliche Durchfluss des Systems. Die tatsächliche Stellgröße $u_s(t)$ kann durch Faktoren wie mechanische Trägheit, Verzögerungen oder Nichtlinearitäten des Stellglieds von der ursprünglich durch den Regler bestimmten Stellgröße $u(t)$ abweichen.
- **Störgrößen $d(t)$:** Externe Einflüsse, die auf das System einwirken und die Regelgröße verändern können.
- **Messglied/Sensor:** Das Messglied ist dafür zuständig, die aktuelle Regelgröße $y(t)$ kontinuierlich zu erfassen. Es wandelt die physikalischen Zustände oder Veränderungen der Regelgröße in eine für den Regler verarbeitbare Form um.
- **Regelstrecke:** Die Regelstrecke ist der Teil des Systems, der geregelt werden soll. In einem Heizungssystem ist die Regelstrecke der Raum, inklusive aller Eigenschaften und Umgebungsfaktoren, die die Raumtemperatur beeinflussen können. Dazu gehören die Raumisolierung, Fenster, Türen und die Raumgröße. Die Regelstrecke reagiert auf Änderungen der Stellgröße (z.B. die von den Heizelementen abgegebene Wärme) und auf externe Störgrößen wie das Öffnen eines Fensters. Die Regelstrecke bestimmt, wie die Regelgröße (in diesem Fall die Raumtemperatur) auf die vom Regler vorgenommenen Anpassungen der Stellgröße reagiert und welche Dynamik der Regelprozess aufweist.

2.1.2 Verhalten von Regelstrecken

In der Regelungstechnik wird das dynamische Verhalten von Regelstrecken mit Modellen beschrieben, welche dabei helfen die Reaktion der Strecke auf Steuersignale zu verstehen und vorherzusagen. Häufig werden hierfür Standardmodelle wie PT1- und PT2-Glieder verwendet. PT1-Glieder (Systeme erster Ordnung) zeigen eine proportionale Verzögerung, meist verursacht durch einfache Energiespeicher, und sind repräsentativ für thermische Prozesse oder elektrische Schaltungen. PT2-Glieder (Systeme zweiter Ordnung) sind komplexer, können Oszillationen aufweisen und finden sich in Systemen mit mehreren Energiespeichern wie zum Beispiel in mechanischen oder elektrischen Schaltungen [33, S. 201 ff.]. Außerdem können Übertragungssysteme hinsichtlich ihrer Linearität unterschieden werden. Lineare Regelstrecken beschreiben Systeme, wo die Ausgangsreaktion proportional zum Eingangssignal ist und das Superpositionsprinzip gilt. Nichtlineare Strecken hingegen weisen ein komplexeres Verhalten auf, welches Beispielsweise durch Effekte wie Sättigung oder Hysterese geprägt sein kann [69]. In der Regelungstechnik ist es entscheidend, möglichst genau zu verstehen und darzustellen, wie sich die Regelstrecke verhält, um den entsprechenden Regler richtig auslegen zu können.

2.2 Künstliche Intelligenz

Künstliche Intelligenz als treibende Kraft der heutigen Technologie formt unsere Zukunft und eröffnet neue Wege in zahlreichen Disziplinen. In diesem Abschnitt werden einige wesentliche Begriffe erläutert, um ein grundlegendes Verständnis für die anschließenden Diskussionen und Analysen in dieser Arbeit zu schaffen.

2.2.1 Definition Künstliche Intelligenz

Künstliche Intelligenz ist ein Begriff, welcher in der modernen technologischen Welt weit verbreitet und dennoch vielfältig in seiner Bedeutung ist. Eine allgemeingültige einheitliche Definition von KI gibt es nicht. Allgemein bezieht sich KI auf das wissenschaftliche Bestreben, Maschinen oder Computerprogramme zu entwickeln, die Fähigkeiten aufweisen, die als intelligent betrachtet werden und somit Aufgaben erfüllen, die traditionell menschliches Denken erfordern [44]. Der Titel dieser Arbeit verwendet den Ausdruck "KI-basierte Lösung". Um Unklarheiten zu vermeiden, wird zunächst erläutert, was damit gemeint ist. In diesem Kontext bezieht sich KI nämlich vorrangig auf die Anwendung von maschinellem Lernen, welches dazu dient, den Regler zu entwerfen oder zu optimieren.

2.2.2 Maschinelles Lernen

Maschinelles Lernen (ML), auch „Machine Learning“ genannt, ist ein Kernbereich der künstlichen Intelligenz, der sich darauf konzentriert, Computern das Lernen aus Daten und Erfahrungen zu ermöglichen. Durch die Algorithmen wird es dem Modell ermöglicht Muster und Zusammenhänge in umfangreichen Datensätzen zu erkennen, und darauf basierend Entscheidungen oder Prognosen zu treffen. Durch das wiederholte Lernen anhand der Daten verbessert sich das Modell selbstständig in seinem Aufgabenbereich. In den meisten Fällen ist eine große Datenmenge von Vorteil um die Vorhersagegenauigkeit zu verbessern [46]. In dieser Arbeit werden die Begriffe KI und ML oft synonym verwendet, da ML eine wesentliche Methode innerhalb der KI darstellt, um intelligente Lösungen zu entwickeln.

Arten des maschinellen Lernens

Es gibt verschiedene Arten des maschinellen Lernens. Im Folgenden werden die drei wichtigsten beschrieben.

1. **Überwachtes Lernen (Supervised Learning):** Beim überwachten Lernen wird das Modell mit einem Datensatz trainiert, der sowohl die Eingaben (Features) als auch die gewünschten Ausgaben (Labels) enthält. Die erwarteten Ergebnisse, auch als „Ground Truth“ bezeichnet, sind das was der Mensch als richtige Lösung der Aufgabe definiert. Die Qualität der Labels ist entscheidend für den Lernerfolg des Modells. Das Ziel des Trainings ist es, eine bestimmte Funktion zu erlernen, welche die Eingaben auf die entsprechenden Ausgaben abbildet. Diese Funktion beschreibt die Zusammenhänge und Muster der Daten, sodass das Modell Vorhersagen für neue, ungesehene Daten treffen kann. Es ist jedoch zu beachten, dass das Modell dabei keine neuen Erkenntnisse generiert, sondern bestehende Informationen der Trainingsdaten nutzt und auf neue Daten anwendet [37, S. 138 f.].

2. **Unüberwachtes Lernen (Unsupervised Learning):** Unüberwachtes Lernen verzichtet auf vordefinierte Labels im Gegensatz zum überwachten Lernen. Der Algorithmus ist darauf ausgelegt, eigenständig Muster und Strukturen in den Daten zu erkennen, diese zu organisieren und zu klassifizieren. Dies ermöglicht die Entdeckung von Mustern, die einem menschlichen Beobachter möglicherweise nicht auffallen würden. Allerdings liegt es in der Verantwortung des Menschen die Resultate zu interpretieren und zu bewerten, da der Algorithmus keine Erklärungen für die gebildeten Gruppierungen bietet [23].

3. **Lernen durch Verstärkung (Reinforcement Learning):** Reinforcement Learning (RL) ist ein maschinelles Lernverfahren ohne vorgegebenen Trainingsdaten und an die Art des menschlichen Lernens angelehnt. Hier lernt ein Agent, optimale Entscheidungen durch Interaktionen mit seiner Umgebung zu treffen und stützt sich dabei auf Rückmeldungen in Form von Belohnungen oder Bestrafungen. Durch Versuch und Irrtum versucht der Agent selbstständig herauszufinden, welche Aktionen in welcher Situation gut sind. Ziel ist es, eine Strategie zu entwickeln, welche die Belohnung über einen bestimmten Zeitraum maximiert. Die Belohnungen und Strafen werden speziell für die jeweilige Aufgabe definiert und signalisieren dem Agenten den Erfolg oder Misserfolg seiner Aktionen [17, S. 313 ff.].

Überanpassung

Ein wichtiger Begriff, der häufig im Zusammenhang mit maschinellem Lernen vorkommt, ist die Überanpassung, auch bekannt als Overfitting. Diese tritt auf, wenn ein maschinelles Lernmodell zu komplex ist und sich zu stark an die Trainingsdaten anpasst, inklusive des Rauschens und der Ausreißer. Das Modell lernt dadurch nicht die tatsächliche zugrundeliegende Struktur der Daten, sondern spezifische Muster der Trainingsdaten. Dies führt dazu, dass es zwar auf den Trainingsdaten sehr gut funktioniert, aber auf neuen, unbekannten Daten eine geringere Vorhersagegenauigkeit aufweist. Beim Entwerfen eines ML-Modells ist es daher wichtig, Maßnahmen gegen Overfitting zu ergreifen [57].

Künstliche neuronale Netze

Künstliche neuronale Netze (KNNs) sind computergestützte Modelle, die von der Arbeitsweise des menschlichen Gehirns inspiriert sind. Das Netz besteht aus miteinander verbundenen Knoten (Neuronen), die in Schichten angeordnet sind. Wie in Abbildung 2.2 zu sehen, besitzt das Netz eine Eingabeschicht (Input Layer), eine oder mehrere verborgenen Schichten (Hidden Layer) und eine Ausgabeschicht (Output Layer). Jedes Neuron erhält Eingaben, führt eine gewichtete Summation durch und wendet dann eine Aktivierungsfunktion an um eine Ausgabe zu erzeugen, welche an andere Neuronen weitergegeben wird. Während dem Trainingsprozess werden die Gewichte der Neuronen dann durch einen Algorithmus namens Backpropagation in Kombination mit einer Optimierungsmethode wie dem Gradientenabstieg so lange angepasst, bis das Netz die gewünschte Vorhersagegenauigkeit besitzt. Ein KNN welches aus zahlreichen Hidden Layern besteht, bezeichnet man auch als tiefes neuronales Netz (Deep Neural Network). Durch die vielen Schichten wird es dem Netz ermöglicht, komplexe Muster und Zusammenhänge in den Trainingsdaten zu erkennen [64].

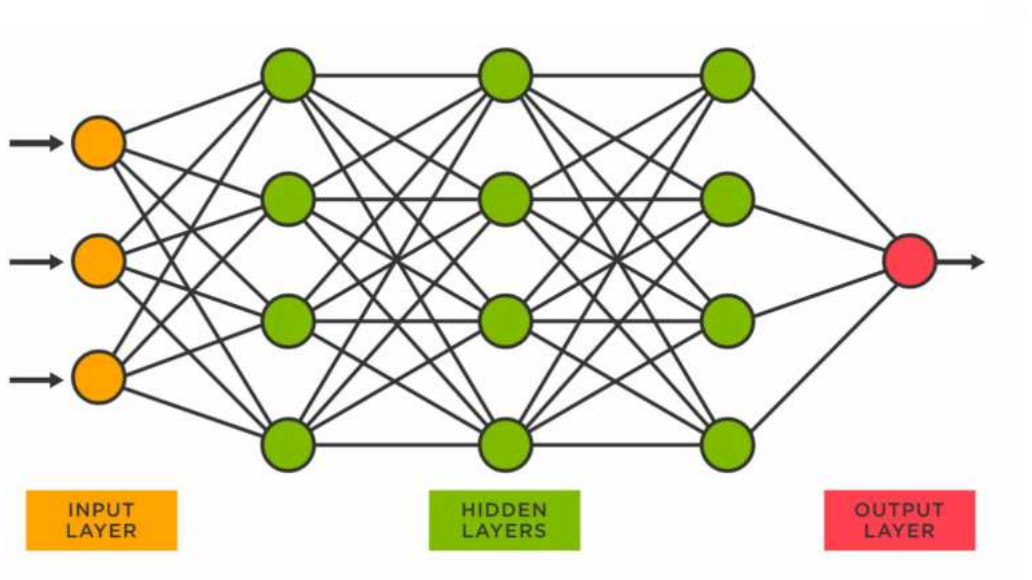


ABBILDUNG 2.2: Aufbau eines neuronalen Netzes [9]

2.3 Optimierungsprobleme

Im Laufe meiner Bachelorarbeit spielen Optimierer eine wesentliche Rolle. Sie sind das Herzstück vieler Algorithmen in der Regelungstechnik und im maschinellen Lernen. Deshalb zielt dieser Abschnitt darauf ab, eine kurze Erklärung zu Optimierern und Optimierungsproblemen zu bieten.

Was ist ein Optimierungsproblem?

Optimierungsprobleme sind ein zentraler Bestandteil in vielen Bereichen der Technik und Wissenschaft, insbesondere in der Regelungstechnik und im maschinellen Lernen. Sie zielen darauf ab, die beste Lösung aus mehreren möglichen zu ermitteln. Die Bewertung erfolgt durch eine Zielfunktion, die es zu maximieren oder minimieren gilt. Entscheidungsvariablen sind dabei die Elemente, welche innerhalb des Problems verändert werden können, um die Zielfunktion zu beeinflussen, während Restriktionen die Grenzen oder Bedingungen definieren, welche eingehalten werden müssen [51, S. 5 f.].

Globale und Lokale Optimierung

Beim Lösen von Optimierungsproblemen wird zwischen globalen Optimierern und lokalen Optimierern unterschieden.

- **Globale Optimierer** sind Algorithmen, die in der Lage sind, den gesamten Lösungsraum eines Problems zu durchsuchen, um das absolute Optimum zu finden. Dies ist besonders wichtig in Fällen, wo es mehrere lokale Optima gibt, aber nur ein globales Optimum, das die beste Lösung darstellt.
- **Lokale Optimierer** hingegen konzentrieren sich auf die Suche in einem begrenzten Bereich des Lösungsraums. Sie sind schnell und effizient in der Findung eines lokalen Optimums nahe einem Startpunkt. Allerdings besteht das Risiko, dass dieses lokale Optimum nicht das beste mögliche Ergebnis über

den kompletten Lösungsraum darstellt. Bei lokalen Optimierern kann die Auswahl des Startpunktes (initial guess) starke Auswirkungen auf das Ergebnis haben [7, S. 9 f.].

Optimierungsmethoden

Die zu optimierenden Funktionen unterscheiden sich in Aspekten wie Stetigkeit, Differenzierbarkeit, Linearität oder in vielen anderen charakteristischen Eigenschaften, welche die Wahl der Optimierungsmethode beeinflussen.Von gradientenbasierten Ansätzen über heuristische Algorithmen bis hin zu evolutionären Verfahren gibt es viele verschiedene Optimierungsmethoden. Abhängig von den Eigenschaften der Funktion, welche optimiert werden soll eignen sich bestimmte Methoden mehr oder weniger. Die genaue Funktionsweise und detaillierte Betrachtung der einzelnen Optimierungsmethoden werden in den entsprechenden Kapiteln dieser Arbeit behandelt, in denen diese Methoden angewendet oder diskutiert werden.

Kapitel 3

Stand der Technik: KI in der Regelungstechnik

In diesem Kapitel werden die Methoden zur Anwendung von KI in der Regelungstechnik diskutiert. Hierfür werden bestehende Arbeiten und Veröffentlichungen betrachtet um das Potenzial und die Relevanz dieser Techniken für den Anwendungsfall dieser Arbeit zu bewerten.

3.1 KI zur Optimierung von Regelparameter

Bei herkömmlichen Reglern wie zum Beispiel dem PID-Regler ist es oft eine Herausforderung die richtigen Regelparameter zu finden, insbesondere bei komplexen oder sich verändernden Umgebungen. Hier kann ML eingesetzt werden, um die Parameterfindung zu vereinfachen oder diese zu optimieren. Bei diesem Ansatz soll der herkömmliche Regler in seiner grundlegenden Funktionsweise nicht verändert werden.

3.1.1 Zusammenfassung Artikel: Self-Tuning PID Controller

Zusammengefasst, geht es in dem Artikel „Optimization of Neural Network-Based Self-Tuning PID Controllers for Second Order Mechanical Systems“ [30] darum, dass die Parameter eines PID-Reglers bei sich änderndem Systemverhalten automatisch angepasst werden sollen. Diese Methode beruht auf der Annahme, dass die meisten realen Systeme durch ein System zweiter Ordnung mit drei Variablen m (Masse), k (Feder) und c (Dämpfer) angenähert werden können. Durch Simulation wurden 10 000 000 Trainingsdaten erzeugt um zwei Neuronale Netze zu trainieren. Das erste KNN wurde darauf trainiert den Systemtyp anhand der entsprechenden Systemreaktion zu identifizieren. Die Werte m, k und c welche das erste KNN ausgibt, werden als Input für das zweite KNN verwendet. Dieses wurde darauf trainiert, für das identifizierte System gute PID-Parameter vorzuschlagen.

Vorteile:

- Diese Methode kann in konservativen Branchen angewendet werden, wo aus Sicherheitsgründen der konventionelle PID-Regler bevorzugt wird.
- Im Betrieb ist eine schnelle Anpassung der PID-Parameter, mit relativ wenig Rechenaufwand möglich.
- Kann mit sich änderndem Systemverhalten umgehen.
- Einfach zu implementieren für Systeme, die durch dasselbe Modell angenähert werden können.

Nachteile:

- Für ein System, welches ein zuvor nicht trainiertes Systemverhalten aufweist oder nicht durch ein System zweiter Ordnung modellierbar ist, müssten die neuronalen Netze neu designet und neu trainiert werden.
- Es können nur Systeme geregelt werden, welche auch PID-regelbar sind.

3.1.2 Fazit: Relevanz für diese Arbeit

Die oben erwähnte Studie zum „Self-Tuning“ eines PID-Reglers demonstriert, wie KI genutzt werden kann, um herkömmliche Regler zu optimieren. Diese Methode zeigt zwar Potenzial die Parameterfindung zu automatisieren, vereinfachen und zu optimieren, jedoch bleibt sie letztlich an die Grenzen und Fähigkeiten des entsprechenden Reglers gebunden. Es gibt noch weitere Ansätze, wo ML zur Optimierung der Regelparameter eingesetzt wird. Ein Beispiel hierfür ist die Bayessche Optimierung, die ausführlich in der entsprechenden Literatur beschrieben wird [18]. Auf diese Methoden wird aber nicht weiter eingegangen, da diese Arbeit primär darauf abzielt das Potenzial von KI als zentrale Komponente in der Regelungsstrategie zu erforschen, und nicht nur als ein Werkzeug zur Optimierung bestehender Regelungssysteme.

3.2 Reinforcement Learning (RL) in der Regelungstechnik

In der aktuellen Forschung wird verstärkt das Potenzial von Reinforcement Learning (RL) in der Regelungstechnik untersucht. Es wird vor allem eingesetzt, um komplexe Regelungsprobleme zu lösen, bei denen traditionelle Ansätze an ihre Grenzen stoßen. RL ist eine Methode des maschinellen Lernens, bei welcher der Agent durch Interaktion mit der Umgebung lernt eine bestimmte Aufgabe zu erfüllen. Vereinfacht gesagt, findet der Regler durch ein systematisches Ausprobieren die passende Regelstrategie und lernt dabei aus seinen Fehlern. RL ist vielseitig einsetzbar, mit zahlreichen Implementierungsansätzen wie modellfreien oder modellbasierten Methoden, on-policy oder off-policy Strategien und verschiedenen Algorithmen. Jede dieser Methoden hat ihre spezifischen Vorteile und Herausforderungen, sodass sie je nach Problemstellung unterschiedlich geeignet sein können. Reinforcement Learning wird nicht nur für autonomes Fahren eingesetzt, sondern findet auch in der Industrie Anwendung. Siemens zum Beispiel wirbt mit einem KI-Regler, der auf Grundlage von RL funktionieren soll [45]. Im Folgenden werden ein paar Vor- und Nachteile von RL in der Regelungstechnik aufgeführt. Die Erkenntnisse sind aus verschiedenen Projekten zusammengetragen [8, 20, 50, 31].

Vorteile:

- Durch den explorativen Ansatz kann RL neue und innovative Lösungsstrategien entdecken, die durch konventionelle regelungstechnische Ansätze möglicherweise nicht erreicht werden.
- Modellfreie RL-Methoden erfordern keine detaillierten mathematischen Modelle der Regelstrecke, was die Implementierung in komplexen Systemen vereinfachen kann.
- Im Vergleich zu Supervised- und Unsupervised-Learning ist für RL kein spezieller Datensatz für das Training erforderlich.

Nachteile:

- Das Trainieren von RL-Agenten kann, insbesondere in Szenarien mit umfangreichen Zustandsräumen, sehr rechen- und zeitaufwendig sein.
- Effektives Lernen erfordert umfangreiche Datenmengen aus der Interaktion mit der Umgebung, was in realen Systemen oft schwierig zu handhaben ist, ohne den normalen Betrieb zu stören. Daher werden häufig Simulationen verwendet, um die Agenten vorab zu trainieren.
- Die Übertragung von in Simulationen trainierten RL-Strategien auf reale Systeme kann problematisch sein, falls die simulierten Bedingungen nicht sämtliche Aspekte der realen Umgebung vollständig abbilden.
- Die Lernprozesse von RL-Agenten können instabil sein und zu nicht vorhersehbaren oder suboptimalen Verhaltensweisen führen. Dies ist besonders kritisch in sicherheitsrelevanten Anwendungen.
- Die Notwendigkeit, neue Situationen zu erkunden, kann in einigen Systemen nicht praktikabel oder sogar gefährlich sein.
- Die Trainingsergebnisse können variieren. Nicht jedes erfolgreiche Training führt zu einem Agenten, der ein zuverlässiges Regelverhalten zeigt.

3.2.1 Fazit: Relevanz für diese Arbeit

RL ist ein fortschrittlicher Ansatz für die Implementierung verschiedener Regelsysteme. Allerdings zeigt eine genauere Betrachtung der Anwendungsbereiche und Projekte, dass RL vorrangig in hochkomplexen Systemen zum Einsatz kommt. Daraus und den vorliegenden Erfahrungsberichten anderer Anwendungen lässt sich schließen, dass die mit RL verbundenen Herausforderungen für weniger komplexe Aufgaben, wie sie in diesem Projekt vorliegen, oft unverhältnismäßig sind. Daher wird RL selten für solch einfache Regelaufgaben verwendet.

3.3 Überwachtes Lernen in der Regelungstechnik

Auch überwachtes Lernen nimmt eine bedeutende Rolle in der Regelungstechnik ein und bietet vielfältige Anwendungsmöglichkeiten. Ein zentrales Einsatzgebiet ist die Systemidentifikation komplexer Systeme mittels historischer Daten, um die Genauigkeit der Systemmodelle zu verbessern. Diese Modelle sind nicht nur für Simulationen wertvoll, die zum Lernen von ML-Reglern genutzt werden können, sondern auch um traditionelle Regelsysteme mittels Simulationsverfahren zu optimieren. Bei der Systemidentifikation können verschiedene Ansätze verfolgt werden. Einerseits kann das vollständige System durch neuronale Netze modelliert werden. Es gibt aber beispielsweise auch Ansätze bei denen nur die Abweichungen von einem theoretischen mathematischen Modell durch neuronale Netze modelliert werden.

Neben der Systemidentifikation kann ein Modell, das durch überwachtes Lernen trainiert wurde, spezifische Funktionen von Reglern übernehmen oder sogar den gesamten Regler ersetzen. In diesem Zusammenhang lassen sich beispielsweise Daten eines effizienten PID-Reglers oder, wie in [74] beschrieben, eines MPCs nutzen, um ein neuronales Netzwerk zu trainieren, das daraufhin die Regelungsdynamiken erlernt.

Die Einsatzmöglichkeiten von überwachtem Lernen in der Regelungstechnik sind theoretisch grenzenlos. Die größten Herausforderungen liegen jedoch oft im Sammeln genügend großer Datenmengen mit den richtigen Labels und im passenden Design des Netzwerks für spezifische Aufgaben.

3.3.1 Erweiterung der Model Predictive Control (MPC) durch maschinelles Lernen

Ein weiterer populärer Ansatz, bei dem ML zum Einsatz kommt, ist in Kombination mit Model Predictive Control (MPC). MPC ist ein fortgeschrittener Regelalgorithmus, der in einem späteren Kapitel detaillierter beschrieben wird. Für die Umsetzung von MPC wird ein Modell benötigt, welches das Systemverhalten vorhersagen kann. Traditionell wird hierfür ein **physikalisches Modell** verwendet. Allerdings gewinnen ML-basierte Ansätze zunehmend an Bedeutung. Eine dieser Möglichkeiten ist das **Black-Box-Modell**, welches ein neuronales Netzwerk nutzt, um das Systemverhalten aus den vorhandenen Daten zu erlernen, ohne die physikalischen Gesetze explizit zu berücksichtigen. Eine Weiterentwicklung davon sind die **Grey-Box-Modelle** oder „Physics Informed Neural Networks (PINNs)“ . Diese verbinden die Vorteile von physikalisch basierten Modellen mit der Flexibilität neuronaler Netze. Sie integrieren physikalisches Vorwissen in die Struktur des neuronalen Netzes, was es ermöglicht, die physikalischen Gesetzmäßigkeiten des Systems zu berücksichtigen , um die Genauigkeit der Vorhersagen zu verbessern. Gleichzeitig kann dadurch die benötigte Trainingsdatenmenge reduziert werden [62, 66]. Das Dokument „Model Predictive Control: Learning-based MPC“ von Alberto Bemporad bietet eine gute Zusammenfassung über die Kombinationsmöglichkeiten von ML und MPC [4]. Verschiedene Ansätze und ML-Modelle, die zur Modellvorhersage eingesetzt werden können, werden darin diskutiert. Die Herausforderungen bei der Implementierung von lernbasierten MPC-Systemen liegen sowohl im Entwurf der ML-Modelle als auch in der Generierung und Aufbereitung geeigneter Trainingsdaten.

Neben dem Entwickeln eines Vorhersagemodells ist auch die Lösung des Optimierungsproblems eine große Herausforderung bei MPC. Das Lösen dieses Problems ist rechenintensiv und muss schnell erfolgen, um eine Regelung in Echtzeit zu ermöglichen. Ein interessanter Ansatz (beschrieben in [74]) besteht darin, den herkömmlichen Optimierungsalgorithmus durch ein neuronales Netzwerk zu ersetzen. Dies ermöglicht es, den rechenintensiven Prozess zu beschleunigen und somit in Echtzeit durchzuführen. Die Trainingsdaten für dieses Netzwerk stammen aus Offline-MPC-Simulationen.

3.3.2 Fazit: Relevanz für diese Arbeit

Überwachtes Lernen hat sich in der Regelungstechnik bereits etabliert und bietet je nach Anwendung unterschiedliche Vor- und Nachteile. Da es viele verschiedene Methoden für den Einsatz des überwachten Lernens in der Regelungstechnik gibt, die alle vielversprechend erscheinen, sollen in diesem Projekt einige davon erprobt und bewertet werden.

Kapitel 4

Auswahl der zu erprobenden KI-Methoden

Im vorangegangenen Kapitel wurden bereits einige Möglichkeiten aufgezeigt, wie maschinelles Lernen in der Regelungstechnik eingesetzt werden kann. Nun geht es darum, spezifische Ansätze auszuwählen, die in dieser Arbeit erprobt und bewertet werden sollen. Ziel ist es, einen Regler zu entwerfen, bei dem künstliche Intelligenz nicht nur eine unterstützende Rolle spielt, sondern als zentrale Komponente fungiert. Im Fokus steht dabei die Temperaturregelung eines Wasserbeckens. Zusätzlich sollen die ausgewählten Methoden das Potenzial haben, dass sie in zukünftigen Projekten für verschiedene Regelungsaufgaben adaptiert werden können, ohne dass dafür ein großer zusätzlicher Implementierungsaufwand nötig ist. Es wäre möglich, Reinforcement Learning (RL) für diesen Anwendungsfall zu erproben. Allerdings zeigt RL vor allem bei hochkomplexen Regelungsaufgaben seine Stärken. Da die Temperaturregelung eines Wasserbeckens sowie andere zukünftige Aufgaben, die der Regler bewältigen soll, eher einfache Single-Input Single-Output (SISO)-Aufgaben darstellen, scheint überwachtes Lernen ein besser geeigneter Ansatz zu sein.

In dieser Arbeit werden deshalb die folgenden zwei Ansätze umgesetzt und bewertet:

- **Nachbildung eines bestehenden Reglers durch ein neuronales Netzwerk:** Ein zuvor entwickelter PID-Regler dient als Grundlage für die Generierung von Trainingsdaten für ein neuronales Netz, das diesen PID-Regler zukünftig ersetzen soll. Dieser Ansatz zielt darauf ab, tiefere Einblicke in den Entwurf von neuronalen Netzwerken, deren Training und die Feinabstimmung der Parameter zu gewinnen. Es wird untersucht, wie effektiv neuronale Netze Regelungsaufgaben übernehmen können und welche Herausforderungen sich dabei ergeben.
- **Erweiterung der Model Predictive Control durch maschinelles Lernen:** In diesem Ansatz wird das herkömmliche Vorhersagemodell der MPC durch ein neuronales Netzwerk ersetzt. Dieses lernt aus historischen Daten das Verhalten des Systems vorherzusagen. Dies ermöglicht die Entwicklung eines Reglers, der effizient funktioniert, auch ohne umfassendes Vorwissen über das spezifische Systemverhalten.

Kapitel 5

Projektimplementierung und Identifizierung der Regelstrecke

In diesem Kapitel geht es um die technische Umsetzung des Projekts, einschließlich der verwendeten Hardware und Software. Es wird auch die Regelstrecke genau untersucht, um ihr Verhalten besser zu verstehen.

5.1 Systembeschreibung

In vorherigen Kapiteln wurde das Ziel dieses Projekts, die Temperaturregelung eines Wasserbeckens, bereits erwähnt. Hier soll nun das System, um das es geht, etwas genauer vorgestellt werden. Es handelt sich um einen Wasserbehälter mit ca. 2 Litern Fassungsvermögen, in dem sich ein PT1000 Temperaturfühler zur Temperaturmessung befindet. Das Erwärmen des Wassers wird durch einen Tauchsieder ermöglicht, welcher im Wasser positioniert ist. Ein wichtiger Aspekt hierbei ist, dass keine aktive Kühlung für das Wasser vorgesehen ist. Stattdessen kühlt sich das Wasser durch natürliche Wärmeabgabe an die Umgebung ab. Die Herausforderung dieses Projekts besteht darin, die Wassertemperatur präzise zu regeln, indem der Tauchsieder entsprechend gesteuert wird, um die gewünschte Temperatur zu erreichen und konstant zu halten. Das Überwachen und Ansteuern des Tauchsieders sowie das Auslesen des Temperaturfühlers erfolgen über einen PC, der als zentrale Einheit fungiert. Die anschließenden Abschnitte dieses Kapitels gehen detailliert auf die Konfiguration der Hardware und die Implementierung der Software ein, die für das Erreichen dieses Regelziels notwendig sind.

5.2 Wahl der Programmiersprache und -Umgebung

Für die Umsetzung dieses Projekts wurde Python als Programmiersprache und PyCharm als Entwicklungsumgebung ausgewählt. Im Zusammenhang mit maschinellem Lernen ist Python eine beliebte und weit verbreitete Programmiersprache. Dies liegt an den umfangreichen Bibliotheken und der Benutzerfreundlichkeit in diesem Gebiet. PyCharm, speziell für Python-Entwicklungen konzipiert, bietet zahlreiche Funktionen, die das Programmieren vereinfachen und effizienter gestalten. Diese Features sind besonders nützlich für komplexe Projekte, wie es hier der Fall ist. Ergänzend werden im Anhang unter Abschnitt A.3 die wichtigsten Python-Bibliotheken, die für das Projekt verwendet werden, kurz beschrieben.

5.3 Hardwareaufbau und Konfiguration

Dieser Abschnitt beschreibt das Hardware-Setup des Projekts, um ein umfassendes Verständnis für die verwendeten Komponenten und deren Zusammenspiel zu schaffen. Zuerst wird das komplette System beschrieben und anschließend die einzelnen Komponenten im Detail betrachtet.

5.3.1 Systemübersicht

In diesem Abschnitt wird der Versuchsaufbau des Projekts vorgestellt, um ein klares Bild von der praktischen Umsetzung zu erhalten. Zuerst wird ein reales Foto des Aufbaus präsentiert, gefolgt von einer detaillierten Skizze, welche die Übersicht auf das Wesentliche reduziert und die Verbindungen zwischen den Komponenten verdeutlicht.

Übersicht des Versuchsaufbaus

Das in Abbildung 5.1 dargestellte System zeigt den Versuchsaufbau für das Projekt. Als zentrales Element dient ein zylinderförmiger, grauer Behälter, der als Wasserbecken fungiert. In diesem Behälter sind sowohl der Tauchsieder als auch der Temperaturfühler positioniert. Ein graues Klemmkästchen übernimmt die Rolle der zentralen Hardware-Schnittstelle, welche die Verbindung zwischen den verschiedenen Komponenten herstellt. Von jedem Gerät führen Kabel zu dieser Verteilerdose, wo die Drähte an Klemmen angeschlossen und entsprechend ihrer Funktionen verbunden werden. Sowohl der Tauchsieder als auch der Temperaturfühler verfügen über Steckverbindungen, was das einfache Trennen des Wasserbehälters vom Rest des Systems ermöglicht. Das rote Gerät mit der Bezeichnung „LabJack T4“ ist ein Eingangs/Ausgangs (E/A)-Modul, welches die Steuerung analoger und digitaler Ein- und Ausgänge durch den PC über eine Universal Serial Bus (USB)- oder Ethernet-Schnittstelle ermöglicht. Links im Bild sieht man den Thyristorsteller für die Phasenanschnittsteuerung, der auf einem Kühlkörper montiert ist.

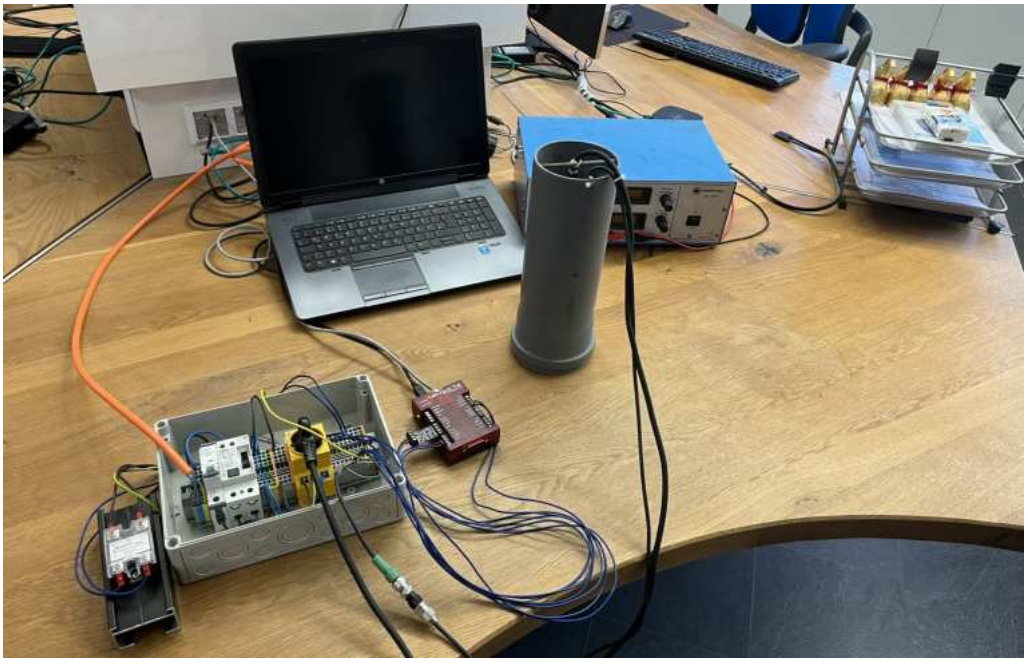


ABBILDUNG 5.1: Darstellung des realen Versuchsaufbaus

Skizzierte Übersicht des Versuchsaufbaus

Im Folgenden wird der Versuchsaufbau anhand einer detaillierten Skizze erläutert, welche in Abbildung 5.2 zu sehen ist. Diese Skizze bietet eine vereinfachte und schematische Darstellung des Systems, wodurch die Verbindungen und Beziehungen zwischen den einzelnen Komponenten klarer hervortreten. Sie veranschaulicht, wie der Tauchsieder, der Temperaturfühler und andere wichtige Elemente innerhalb des Systems angeordnet und miteinander verbunden sind. Das Klemmkästchen ist hierbei nicht aufgeführt, da es lediglich als Verbindungsschnittstelle für die Kabel dient.

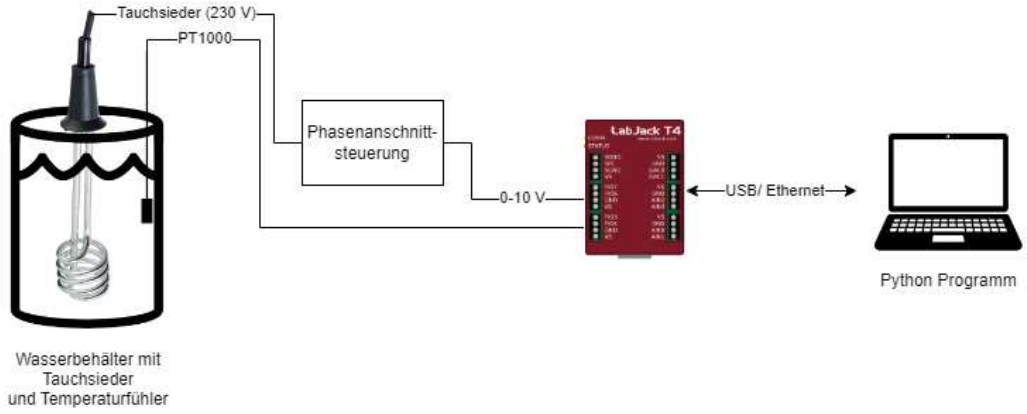


ABBILDUNG 5.2: Skizze des Versuchsaufbaus
mit Elementen aus [16, 28]

Auf dem PC läuft ein Python-Programm, welches über die Ethernet-Schnittstelle mit dem E/A-Modul kommuniziert. Die analogen Ein- und Ausgänge dieses Moduls werden genutzt, um eine Messschaltung für den Temperaturfühler zu realisieren. Zusätzlich kann über einen weiteren analogen Ausgang des LabJack-Moduls ein $0 - 10$ V Signal gesteuert werden. Dieses Signal dient als Steuervorgabe für den Thyristorsteller, an den ein 230 V Signal angeschlossen ist. Der Thyristorsteller moduliert das 230 V Signal mittels Phasenanschnittsteuerung, wodurch eine Leistungsregelung im Bereich von 0 bis 100 Prozent ermöglicht wird. Dieses modulierte Signal wird zu einer Steckdose geführt, an welcher der Tauchsieder angeschlossen ist, um die Temperatur im Wasserbehälter zu regulieren.

5.3.2 Detailbeschreibung der Hardwarekomponenten

In diesem Unterabschnitt werden die spezifischen Hardwarekomponenten, welche im Projekt verwendet werden, im Detail vorgestellt. Jede Komponente wird einzeln betrachtet, um ihre Funktion und Bedeutung im Rahmen des Gesamtsystems zu erläutern.

E/A-Modul „LabJack T4“

Das in Abbildung 5.3 abgebildete LabJack T4 ist ein multifunktionales Datenerfassungsgerät mit welchem über eine USB- oder Ethernet-Schnittstelle kommuniziert werden kann. Es ermöglicht die Ansteuerung und das Auslesen von digitalen und analogen Ein- und Ausgängen direkt vom PC aus. Neben den E/As verfügt das Gerät über Funktionen wie digitale Zähler und Timer sowie eine benutzerfreundliche Graphical User Interface (GUI)-basierte Software [25].

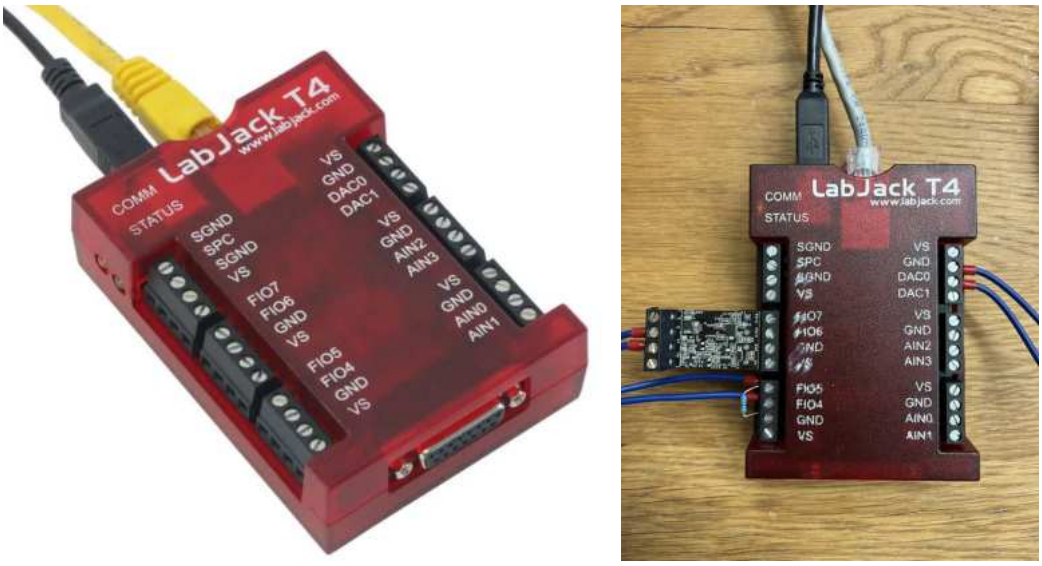


ABBILDUNG 5.3: E/A-Modul LabJack T4 [29]

Im Folgenden werden die Funktionalitäten, welche für dieses Projekt relevant sind genauer beschrieben.

Versorgungsspannung
Das LabJack T4 Modul wird durch eine Gleichspannung von 5 Volt versorgt, die mithilfe eines Netzteils bereitgestellt wird.

Analoge Eingänge:
Für die Auswertung des Temperaturfühlers wird ein analoger Eingang benötigt. Das LabJack T4 verfügt über 12 analoge Eingänge, aufgeteilt in vier Hochspannungs- (AIN0-AIN3) und acht Niederspannungseingänge (AIN4-AIN11). Diese Eingänge können dementsprechend entweder Spannungen im Bereich von ± 10 V oder $0 - 2,5$ V erfassen und bieten eine Auflösung von 12 Bit. Die maximale Datenrate ermöglicht bis zu 40 000 Messwerte pro Sekunde [27, S. 431] .

Analoge Ausgänge:
Die analogen Ausgänge bieten begrenzt durch die Versorgungsspannung, einen Spannungsbereich von $0 - 5$ V. Der maximale Ausgangsstrom ist auf 20 mA beschränkt. Durch den Innenwiderstand von 50 Ω treten an den analogen Ausgängen, abhängig von der Last, leichte Abweichungen vom Sollwert auf [27, S. 680]. Damit der Sollwert des Heizstabes vorgegeben werden kann, wird ein analoges Signal im Bereich von $0 - 10$ V benötigt. Die normalen analogen Ausgänge des LabJack T4 sind hierfür somit nicht ausreichend. Deshalb wurde ein Erweiterungsmodul namens „LJTick-DAC“ verwendet. Das ist die kleine Platine, welche in Abbildung 5.3 im rechten Bild zu sehen ist. Dieses Modul verfügt über insgesamt vier Anschlüsse und wird durch die Bereitstellung von Versorgungsspannung, Ground (GND) und den Anschluss von zwei der digitalen E/As betrieben. Mithilfe einer entsprechenden Software-Konfiguration erweitert dieses Modul die Funktionalität um zwei zusätzliche analoge Ausgangskanäle im Bereich von ± 10 V. Diese zusätzlichen Ausgänge sind als Digital-to-Analog Converter (DAC) ausgelegt. Die Auflösung beträgt 14 Bit (1,22 mV) und für ein Update des Sollwerts wird bei einer Kommunikation über Ethernet ca. 1 ms benötigt, was für dieses Projekt mehr als ausreichend ist [26].

Kommunikation und Konfiguration des LabJack T4:
Für die Kommunikation mit dem LabJack-Modul stehen verschiedene Methoden zur Verfügung. In diesem Projekt wurde Modbus TCP über Ethernet gewählt. Modbus TCP ist ein Kommunikationsprotokoll, das in industriellen Netzwerken weit verbreitet ist. Es ermöglicht die einfache Übertragung von Daten über ein TCP/IP-Netzwerk. Für die Einrichtung der Kommunikation wurde dem LabJack-Modul manuell eine IP-Adresse zugewiesen. Die Kommunikation über Modbus erfolgt durch das Ansprechen spezifischer Register, die verschiedene Funktionalitäten des Geräts steuern. Jedes Register hat eine eindeutige Adresse und ist entweder lesbar, beschreibbar oder beides. Im Datenblatt [27] des LabJack T4 findet sich eine detaillierte Übersicht, welche Register für welche Funktionen zuständig sind. Ein praktisches Beispiel ist das Lesen des Registers 7008 (AIN4_EF_READ_A). Durch das Auslesen dieses Registers erhält man den Wert des analogen Eingangs 4, welcher in diesem Projekt zur Temperaturmessung verwendet wird. Bei jedem Neustart des Python-Projekts werden zunächst die Modbus-Verbindung überprüft und anschließend die Register entsprechend der nötigen Konfiguration für das Projekt beschrieben. Dadurch wird sichergestellt, dass das E/A-Modul korrekt konfiguriert ist und eine zuverlässige Datenübertragung stattfinden kann.

Tauchsieder

Der Tauchsieder, ein zentrales Element in diesem Projekt, ist ein einfaches, jedoch effektives Gerät zur Erwärmung des Wassers im Becken. Er verfügt über eine Nennleistung von 2 kW und einen 230-Volt-Anschluss für die Steckdose. Wie in Abbildung 5.4 zu erkennen, weist der Tauchsieder am Ende eine Heizspirale auf, bei der ein metallischer Stab zu einer spiralförmigen Wicklung geformt ist, um die Wärmeübertragung an das Wasser zu optimieren. Ein Tauchsieder funktioniert nach einem einfachen Prinzip: Er wandelt elektrische Energie in Wärme um. Im Inneren des Tauchsieders befindet sich ein elektrischer Widerstand, der sich erwärmt, wenn Strom durch ihn fließt. Dieser Widerstand ist in einem wärmeleitenden, aber elektrisch isolierenden Material eingebettet, umgeben von einem metallischen, wasserfesten Gehäuse. Wenn der Tauchsieder in das Wasser eingetaucht wird und Strom anliegt, überträgt sich die Wärme des Widerstandes auf das umgebende Wasser, wodurch dieses erwärmt wird.

ABBILDUNG 5.4: Tauchsieder zur Temperaturregelung [16]

Thyristorsteller für Phasenanschnittsteuerung

Um die Leistung des Tauchsieders präzise steuern zu können, ist die Phasenanschnittsteuerung eine effiziente Methode. Dabei wird die anliegende Wechselspannung von 230 V modifiziert, indem Teile der positiven und negativen Sinushalbwellen abgeschnitten werden. Dies erfolgt mittels eines Triacs (Antiparallelschaltung zweier Thyristoren), der nach dem Nulldurchgang der Wechselspannung erst nach einer bestimmten Zeit leitet. Dieser Zeitabschnitt, auch Zündwinkel genannt, wird durch eine Steuerspannung von $0 - 10$ V des Thyristorstellers bestimmt. Umso kleiner die Steuerspannung, desto später der Zündzeitpunkt. Wie in Abbildung 5.5 dargestellt, resultieren verschiedene Steuerspannungen in unterschiedlichen Ausgangsspannungsverläufen. Da der Tauchsieder eine rein ohmsche Last darstellt, folgt der Ausgangsstrom direkt dem modifizierten Spannungsverlauf. Somit kann durch Anpassung der Steuerspannung der Zündwinkel und damit die mittlere Leistung des Tauchsieders gesteuert werden. Eine niedrigere Steuerspannung resultiert in einer geringeren mittleren Leistung. Für ohmsche Lasten wie den Tauchsieder stellt die durch die Phasenanschnittsteuerung erzeugte nicht-sinusförmige Spannung kein Problem dar [47, S. 9 f.].

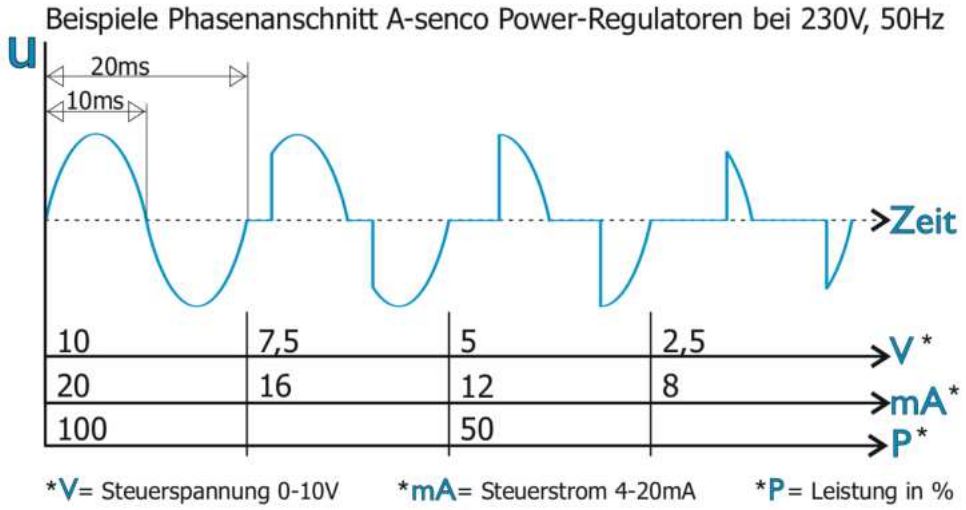


ABBILDUNG 5.5: Spannungsverlauf Thyristorsteller [38, S. 1]

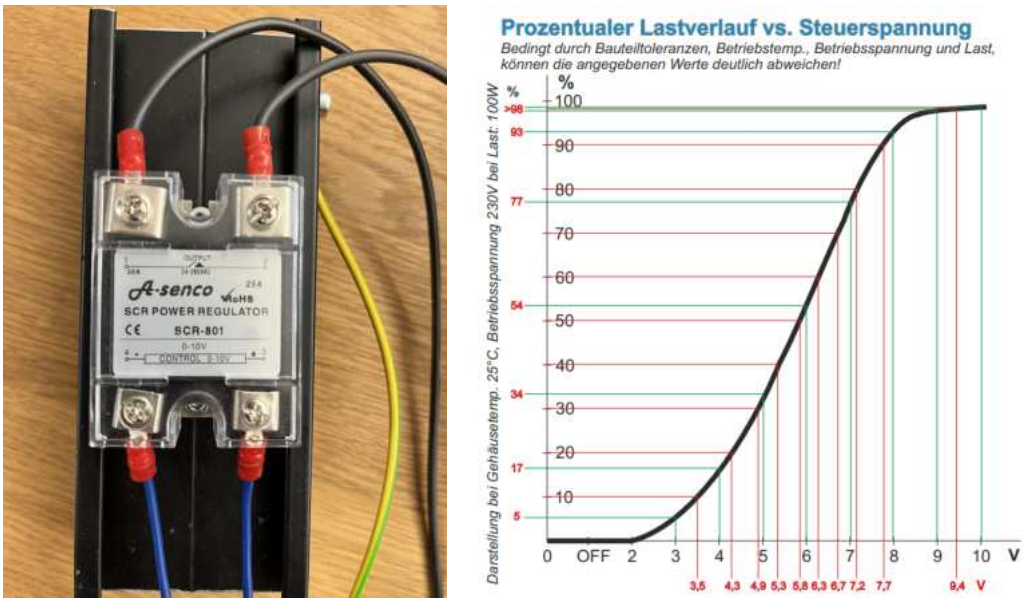
Der im Projekt eingesetzte Thyristorsteller ist für einen maximalen Laststrom von 25 A bei einer Wechselspannung von 230 V ausgelegt. Daraus lässt sich die maximal mögliche Leistung für einen ohmschen Verbraucher ermitteln.

$$P_{\max} = U \cdot I_{\max} \cdot cos\phi = 230\ \text{V} \cdot 25\ \text{A} \cdot 1 = 5{,}750\ \text{kW} \tag{5.1}$$

Somit kann der Tauchsieder welcher eine Nennleistung von 2 kW besitzt, problemlos betrieben werden. Allerdings muss man bei der Phasenanschnittsteuerung und Verbrauchern mit hoher Leistung vorsichtig sein. Die zeitliche Form des Stromflusses weicht stark von der Sinusform ab. Dies kann Spannungsspitzen, elektromagnetische Interferenzen und unerwünschte Oberwellen im Stromnetz verursachen, die andere Geräte stören oder beschädigen können. Da es sich um einen Versuchsaufbau handelt, der nur im Labor genutzt wird, wird diese Thematik in diesem Projekt vernachlässigt. Für eine echte Anwendung müsste man diese Problematik jedoch

genauer betrachten und Lösungen wie Netzfilter oder Drosselfilter in Betracht ziehen [42].

Das linke Bild in Abbildung 5.6 zeigt den eingesetzten Thyristorsteller montiert auf einem Kühlkörper, welcher dazu dient, die bei den Schaltvorgängen entstehende Wärme effektiv abzuleiten und so eine Überhitzung der Bauteile zu vermeiden [39, S. 3]. Die gewünschte mittlere Leistung der Phasenanschnittsteuerung wird mit einem analogen Eingangssignal von $0 - 10$ V gesteuert. Hierbei entspricht 2 V ca. 0 % und 10 V ca. 100 % der Nennleistung des angeschlossenen Geräts [39, S. 1]. Der prozentuale Lastverlauf abhängig von der Steuerspannung ist im rechten Diagramm der Abbildung 5.6 zu sehen.



Thyristorsteller mit Kühlkörper — Lastverlauf in Abhängigkeit der Steuerspannung

ABBILDUNG 5.6: Thyristorsteller und Lastverlauf [39, S. 1]

Im Python-Programm wird der Sollwert für den Tauchsieder als prozentualer Wert der maximal möglichen Leistung angegeben. Bevor der Sollwert an das LabJack-Gerät gesendet wird, muss der prozentuale Leistungswert in den entsprechenden analogen Spannungswert umgerechnet werden. Es wird also eine mathematische Formel gesucht, welche den Leistungswert ($0 - 100$ %) auf einen Spannungswert ($0 - 10$ V) abbildet. Basierend auf den Datenpunkten der Kennlinie in Abbildung 5.6, wurde der linke Plot in Abbildung 5.7 erstellt, welche den notwendigen analogen Spannungswert in Abhängigkeit der gewünschten Leistung zeigt. Das Datenblatt des Thyristorstellers gibt allerdings an, dass die Kennlinie aufgrund von Bauteiltoleranzen, Betriebstemperatur, Last und weiteren Einflüssen variieren kann. Deshalb wurde die tatsächliche Leistung des Tauchsieders bei verschiedenen analogen Spannungswerten zwischen 0 und 10 Volt mittels eines Leistungsmessgeräts gemessen und eine neue Kennlinie erstellt. Diese ist auf der rechten Seite in Abbildung 5.7 dargestellt, wobei die blauen Punkte die gemessenen realen Werte repräsentieren. Um eine Umrechnung von jedem Leistungswert in den entsprechenden analogen Spannungswert zu ermöglichen und die Datenlücken zu füllen, wurde ein Polynom 6. Grades zur Approximation verwendet. Dies wurde durch die „polyfit“-Methode

aus der „Numpy“ Bibliothek realisiert. Nach dem Testen verschiedener Polynomgrade stellte sich nämlich heraus, dass ein Polynom 6. Grades einen guten Kompromiss zwischen Genauigkeit und Komplexität bietet und die realen Datenpunkte gut approximiert. Die resultierende Funktion 6. Grades ist in Abbildung 5.7 im rechten Diagramm in Orange eingezeichnet und wird durch die folgende Formel ausgedrückt. Hierbei ist U_{analog} die analoge Spannung und P die Leistung in Prozent.

$$\begin{aligned} U_{\text{analog}} = {} & 3.05 \cdot 10^{-10} P^6 - 8.38 \cdot 10^{-8} P^5 + 8.68 \cdot 10^{-6} P^4 - \\ & 4.14 \cdot 10^{-4} P^3 + 8.92 \cdot 10^{-3} P^2 - 3.40 \cdot 10^{-2} P + 2.56 \end{aligned} \tag{5.2}$$

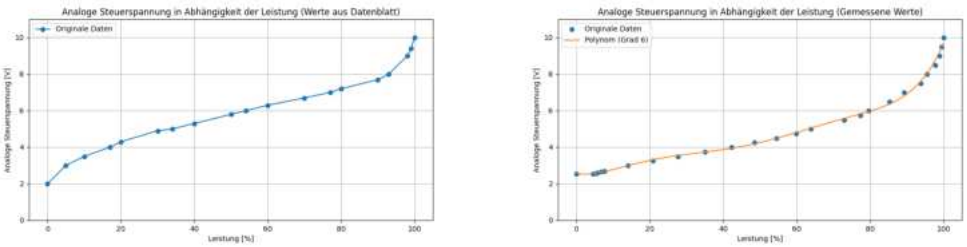


ABBILDUNG 5.7: Analoge Steuerspannung in Abhängigkeit der Leistung

Im weiteren Verlauf des Projektes wird der Sollwert des Tauchsieders ausschließlich in Prozent angegeben, wobei 0 % einer Leistung von 0 kW und 100 % einer vollen Leistung von 2 kW entsprechen. Die oben aufgeführte Umrechnung in einen Analogwert und die Realisierung mittels Phasenanschnittsteuerung treten dabei in den Hintergrund und sind für die folgenden Kapitel nicht weiter von Bedeutung.

Temperaturfühler (PT1000)

Für die Temperaturüberwachung im Wasserbecken wird ein Widerstandsthermometer, bekannt als Resistance Temperature Detector (RTD), verwendet. Diese Art von Temperaturfühlern besteht aus einem Kabel und einem Messwiderstand, dessen Widerstand sich temperaturabhängig ändert. Durch das Messen dieses Widerstands kann die Temperatur bestimmt werden. Unter den gängigen RTDs sind der PT100 und der PT1000 besonders verbreitet. Der wesentliche Unterschied zwischen diesen beiden Typen liegt in ihrem Nennwiderstand. Bei einer Temperatur von 0 °C weist der PT100 einen Widerstand von 100 Ω auf, während der PT1000 bei der gleichen Temperatur einen Widerstand von 1000 Ω zeigt. Ein weiteres entscheidendes Merkmal der Sensoren ist die in Abbildung 5.8 dargestellte Kennlinie. Beide haben eine näherungsweise lineare Kennlinie, aber die des PT1000 ist etwa zehnmal steiler als die des PT100. Diese größere Steigung führt zu genaueren Ergebnissen bei geringfügigen Temperaturänderungen, da der PT1000 empfindlicher auf solche Änderungen reagiert [43]. In der Praxis bedeutet dies, dass der PT1000 für jede Temperaturänderung von 1 K seinen Widerstand um ungefähr 4 Ω ändert, während der PT100 eine Änderung von 0,4 Ω aufweist. Die höhere Empfindlichkeit des PT1000 macht ihn ideal für Anwendungen, bei denen es auf Genauigkeit ankommt, da er weniger anfällig für äußere Störeinflüsse wie Leitungswiderstände ist. Aus diesem Grund wurde für dieses Projekt der PT1000 ausgewählt.

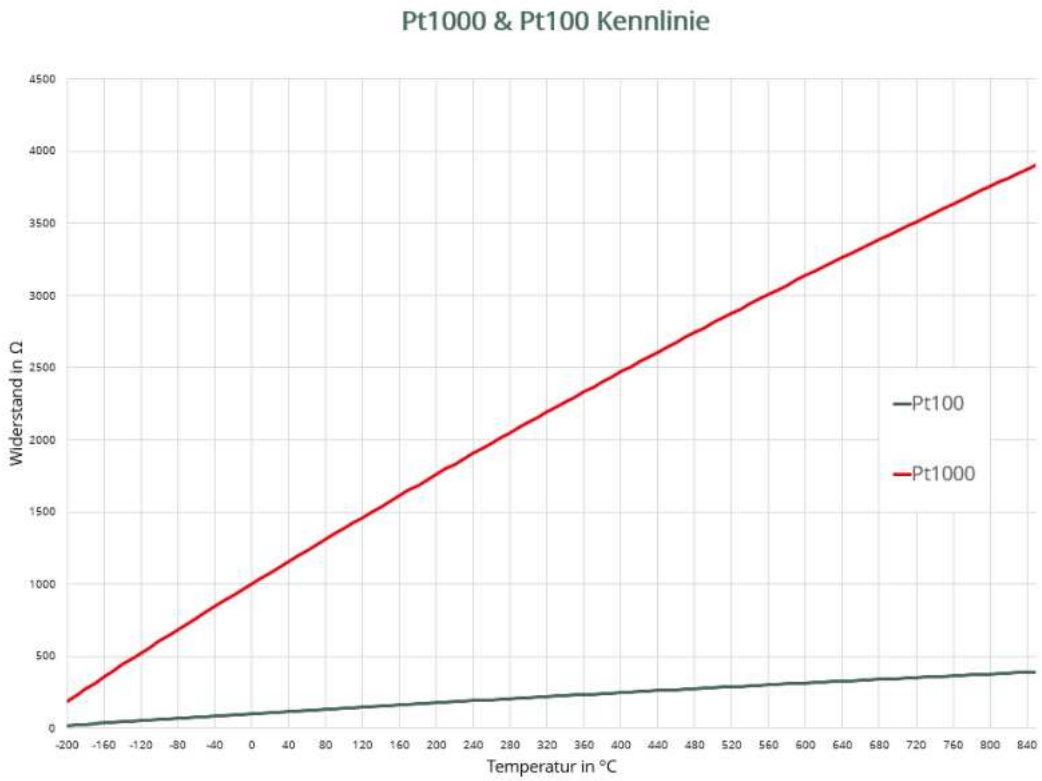


ABBILDUNG 5.8: Pt100 und PT1000 Kennlinie im Vergleich [43]

Messschaltung zur Temperaturbestimmung:
Für die Bestimmung der Temperatur ist eine geeignete Messschaltung erforderlich. Das LabJack T4 bietet die Funktionalität, den Widerstand eines RTDs zu messen und diesen direkt in einen Temperaturwert umzurechnen [27, S. 480 ff.]. Verschiedene empfohlene Messschaltungen sind im Datenblatt detailliert dargestellt [27, S. 468 ff.]. Allerdings werden nicht alle diese Methoden von der T4-Modellreihe unterstützt. Andere benötigen eine externe Konstantstromquelle oder sind anfällig für Störungen. Die in Abbildung 5.9 gezeigte Schaltung bietet den besten Kompromiss zwischen einfacher Implementierung und hoher Messgenauigkeit.

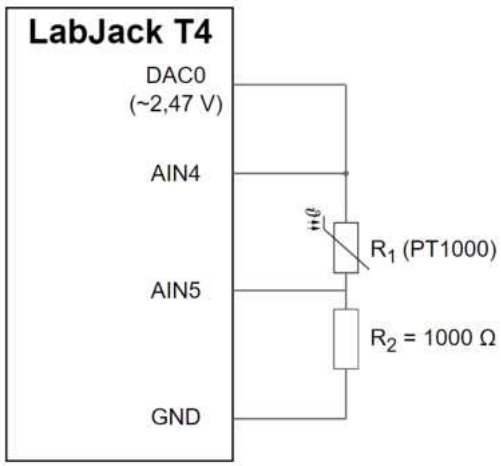


ABBILDUNG 5.9: Messschaltung PT1000

Für die Schaltung zur Temperaturmessung wird eine Spannungsquelle sowie zwei analoge Eingänge benötigt. Die Wahl fällt auf die Eingänge „AIN4“ und „AIN5“, die einen Messbereich von $0 - 2,5$ V bieten. Diese Auswahl ist vorteilhaft gegenüber den Eingängen mit einem Messbereich von ± 10 V, da bei einem Bereich von $0 - 2,5$ V eine effektive Ausnutzung des gesamten Messbereichs leichter zu realisieren ist. Dadurch lässt sich eine höhere Auflösung erzielen, was die Genauigkeit der Temperaturmessung verbessert. Als Spannungsquelle dient der analoge Ausgang „DAC0“, welcher auf 2,6 V eingestellt wird. Durch die Messschaltung wird der Ausgang belastet und es entsteht ein Spannungsabfall, sodass an „AIN4“ noch ungefähr 2,47 V anliegt. Dieser Wert liegt somit noch innerhalb des zulässigen Bereichs von maximal 2,5 V. Durch die Messung der Spannungen an „AIN4“ und „AIN5“ lässt sich der Widerstand des PT1000 bestimmen. Daraus kann dann der Temperaturwert des Temperaturfühlers ermittelt werden. Diese Berechnung wird intern vom LabJack-Gerät durchgeführt. Es ist lediglich notwendig, die verschiedenen Funktionen den Ein- und Ausgängen zuzuordnen und die entsprechende Konfiguration vorzunehmen, wie beispielsweise die Einstellung der Temperatureinheit in °C oder den Wert des Widerstands R_2.

Wahl des Widerstands R_2:
Der Widerstand R_2 bildet zusammen mit dem PT1000 ein Spannungsteiler. Somit beeinflusst der Wert von R_2 die Spannung über dem PT1000 und somit die Empfindlichkeit der Schaltung gegenüber Temperaturänderungen. Bei Temperaturschwankungen ist es entscheidend, die Spannungsänderung an „AIN5“ möglichst groß zu halten, um selbst geringfügige Temperaturänderungen präzise erfassen zu können. Für dieses Projekt wird $R_2 = 1000\ \Omega$ gewählt. Damit wird die maximale Empfindlichkeit bei 0 °C erreicht, wo der PT1000 einen Widerstandswert von 1000 Ω besitzt. Aber auch die restlichen Arbeitspunkte besitzen eine ausreichende Auflösung.

Auflösung der Temperaturerfassung
Durch eine Verlängerung der Abtastzeit von 0,07 ms auf 0,5 ms, was für diese Anwendungen immer noch ausreichend ist, kann die Auflösung der analogen Eingänge von 12 auf 13,2 Bit gesteigert werden. Dies erfolgt durch Anpassen des Parameters „Resolution Index“ [27, S. 645 f.]. Diese Verbesserung führt zu einer präziseren Erfassung kleiner Spannungsänderungen. Die kleinste messbare Spannungsänderung, die der analoge Eingang detektieren kann, lässt sich wie folgt berechnen:

$$U_{\text{res}} = \frac{V_{\text{max}} - V_{\text{min}}}{2^n} = \frac{2,5\ \text{V} - 0\ \text{V}}{2^{13.2}} = 0,266\ \text{mV} \tag{5.3}$$

Zur Abschätzung der resultierenden minimalen Temperaturänderung, die erfasst werden kann, wird ein Arbeitspunkt von 100 °C betrachtet. Dieser Punkt repräsentiert eine Art Worst-Case-Szenario, da mit steigender Temperatur die Auflösung tendenziell abnimmt, bedingt durch die zunehmende Differenz zwischen den Widerständen R_1 (PT1000) und R_2. In diesem Projekt wird das Wasser nicht über 100 °C erhitzt, daher stellt die Festlegung von 100 °C als Arbeitspunkt eine sinnvolle Basis dar, um die Leistungsfähigkeit und Grenzen der Schaltung realistisch zu bewerten.

Die Spannung am analogen Eingang „AIN4“ beträgt $U_{\text{AIN4}} = 2,47$ V, und der Widerstand R_2 hat einen Wert von $R_2 = 1000\ \Omega$. Diese Werte sind für die folgenden Berechnungen notwendig.

Zunächst wird der Widerstand des PT1000 bei einer Temperatur von 100 °C berechnet. R_0 ist der Nennwiderstand bei 0 °C, θ die Temperatur und α der Temperaturkoeffizient eines Platin Messwiderstandes. Für den Temperaturbereich von 0 bis 100 °C liefert die folgende Formel unter Verwendung des angegebenen mittleren Werts für α eine angemessene Näherung [67].

$$\begin{aligned} R_1(100\,^\circ\mathrm{C}) &= R_0 \cdot (1 + \alpha \cdot \theta) \\ &= 1000\,\Omega \cdot (1 + 3.85 \cdot 10^{-3}\,^\circ\mathrm{C}^{-1} \cdot 100\,^\circ\mathrm{C}) = 1385\,\Omega \end{aligned} \tag{5.4}$$

Die Spannung an „AIN5“ kann über die Spannungsteiler Formel berechnet werden.

$$U_{\mathrm{AIN5}} = U_{\mathrm{AIN4}} \cdot \frac{R_2}{R_1 + R_2} = 2,47\ \mathrm{V} \cdot \frac{1000\,\Omega}{1385\,\Omega + 1000\,\Omega} = 1,035639\ \mathrm{V} \tag{5.5}$$

Ändert sich nun die Temperatur und somit die Spannung U_{AIN5} um 0,266 mV ergibt sich:

$$U_{\mathrm{AIN5neu}} = U_{\mathrm{AIN5}} + \Delta u_{\mathrm{AIN5}} = 1,035639\ \mathrm{V} + 0,000266\ \mathrm{V} = 1,035905\ \mathrm{V} \tag{5.6}$$

Nun kann der neue Widerstand des PT1000 berechnet werden.

$$R_{1\mathrm{neu}} = R_2 \cdot \frac{U_{\mathrm{AIN4}} - U_{\mathrm{AIN5neu}}}{U_{\mathrm{AIN5neu}}} = 1000\,\Omega \cdot \frac{2,47\ \mathrm{V} - 1,035905\ \mathrm{V}}{1,035905\ \mathrm{V}} = 1384,3885\,\Omega \tag{5.7}$$

Stellt man die Gleichung 5.4 um, kann man die daraus resultierende Temperatur berechnen.

$$\theta_{\mathrm{neu}} = \frac{R_{1\mathrm{neu}} - R_0}{\alpha \cdot R_0} = \frac{1384,3885\,\Omega - 1000\,\Omega}{3.85 \cdot 10^{-3}\,^\circ\mathrm{C}^{-1} \cdot 1000\,\Omega} = 99,84\,^\circ\mathrm{C} \tag{5.8}$$

Dies verdeutlicht, dass eine Temperaturänderung von 0,16 K eine Spannungsänderung von 0,266 mV am analogen Eingang „AIN5“ bewirkt, welcher die kleinstmögliche messbare Veränderung darstellt. Daraus folgt, dass die Temperaturmessung bei einer Temperatur von 100 °C eine Auflösung von 0,16 K erreicht. Bei niedrigeren Temperaturen verbessert sich die Messgenauigkeit geringfügig. Für die Anforderungen dieses Projekts ist diese Auflösung ausreichend.

Anwendung eines gleitenden Mittelwertfilters
Wenn der Sensor bei einer gleichbleibenden Temperatur ausgelesen wird, ist ein gewisses Maß an Rauschen zu erkennen. Dieses Rauschen entsteht durch Messfehler und äußere Störungen. Um die Messdaten zu verbessern, wurde in Python ein gleitender Mittelwertfilter implementiert. Dieser Filter verwendet ein Zeitfenster von drei Sekunden, um die Temperaturdaten zu glätten. In Abbildung 5.10 sieht man deutlich den Unterschied zwischen dem gefilterten und dem ungefilterten Messwert.

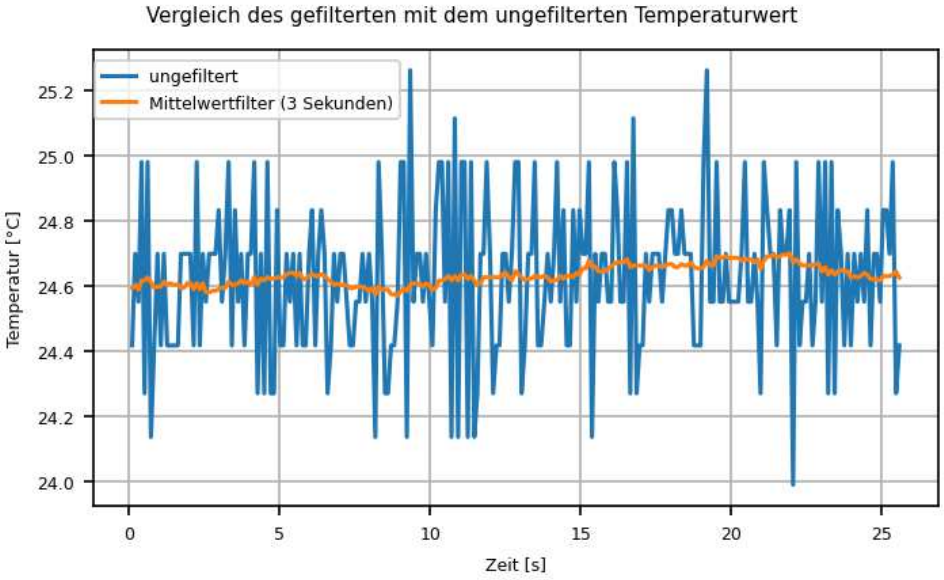


ABBILDUNG 5.10: Vergleich des gefilterten mit dem ungefilterten Temperaturwert

Aufbau der Verteilerdose

Die Verteilerdose ist das graue Klemmkästchen welches in Abbildung 5.11 zu sehen ist. Es fungiert als zentrale Schnittstelle für die Hardware. Sie verbindet die Kabel von allen Geräten. Jedes Kabel wird auf Klemmen aufgelegt und entsprechend seiner Funktion verdrahtet. Die Stromversorgung des Systems erfolgt durch eine 230 V Wechselspannungszuleitung, die mit einem Schutz-Kontakt (Schuko)-Stecker ausgestattet ist. Zum Schutz vor Überlast und Kurzschlüssen ist ein Leitungsschutzschalter mit C10-Charakteristik eingebaut. Zusätzlich gibt es einen Fehlerstrom- Schutzschalter (RCD), der vor Fehlerströmen schützt. Dieser ist besonders wichtig in feuchten Umgebungen und bei Anwendungen wo Wasser im spiel ist. Die gelbe Steckdose innerhalb der Verteilerdose empfängt das durch die Phasenanschnittsteuerung modifizierte Signal. Dort wird der Stecker für den Tauchsieder angeschlossen. Auch der Temperatursensor wurde mit einem kleinen Sensorstecker versehen, was das Trennen des Wasserbehälters vom System vereinfacht.

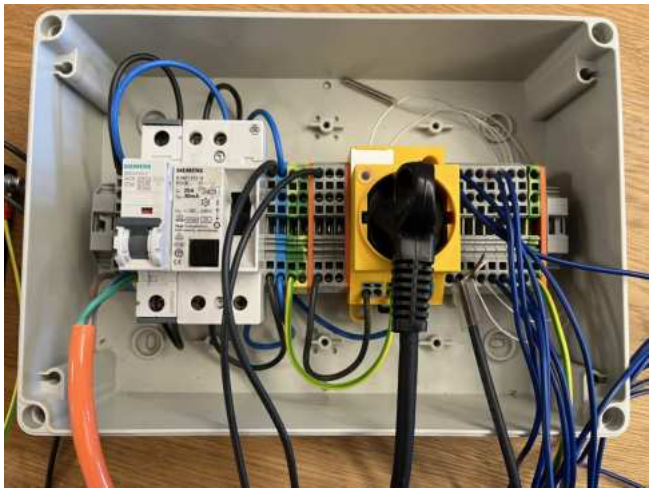
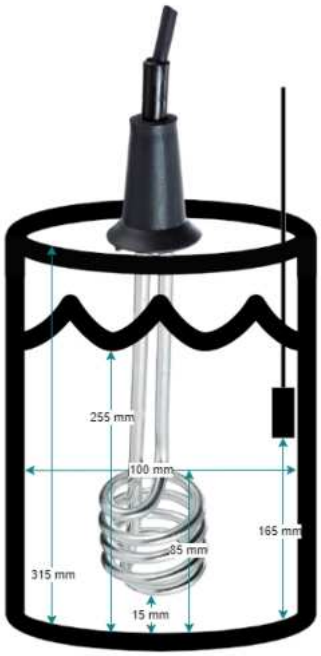


ABBILDUNG 5.11: Verteilerdose des Systems

Beschreibung des Wasserbehälters

Der Behälter besteht aus einem HT-Rohr, das hohe Temperaturen aushalten kann. Zwei Gewindestangen sind daran befestigt, um den Tauchsieder und den Temperatursensor zu fixieren. Diese Anordnung sorgt dafür, dass der Sensor immer an derselben Stelle bleibt. Der Tauchsieder hat zwar etwas Spiel nach links und rechts, ist aber immer auf gleicher Höhe. Der Behälter wird mit 2 Liter Wasser befüllt. Um zu verhindern, dass bei hohen Temperaturen viel Wasser verdunstet, wird er mit einer Plexiglasscheibe abgedeckt. In Abbildung 5.12 ist links das tatsächliche System und rechts eine skizzierte Seitenansicht mit Maßen zu sehen.

Realer Aufbau



Skizze des Aufbaus mit Elementen aus [16]

ABBILDUNG 5.12: Wasserbehälter mit Tauchsieder und PT1000

5.4 Identifizierung der Regelstrecke

Das Kernziel dieses Abschnitts besteht darin, das Verhalten des zu regelnden Systems eingehend zu analysieren und präzise zu modellieren. Durch die Entwicklung eines mathematischen Modells wird eine solide Basis für eine effiziente Temperaturregelung geschaffen.

5.4.1 Begrenzung der Arbeitstemperatur

Beim Annähern an den Siedepunkt, welcher bei ungefähr 100 °C liegt, verändert sich das thermische Verhalten von Wasser signifikant. Diese Phänomene erschweren eine präzise Temperaturregelung. Um diese Regelungsherausforderungen zu vermeiden und eine gleichbleibende Systemdynamik zu gewährleisten, wird der Arbeitstemperaturbereich dieses Projekts auf 20 °C bis 80 °C beschränkt. Der gewählte untere Grenzwert von 20 °C kommt daher, dass das System nicht aktiv gekühlt werden

kann und Temperaturen unterhalb der Raumtemperatur praktisch nicht zu erreichen sind, es sei denn durch manuelle Aktionen wie das Zufügen von kaltem Wasser. Diese Temperaturbegrenzung stellt sicher, dass die Regelung innerhalb eines praktikablen und vorhersagbaren Bereichs erfolgt.

5.4.2 Komponenten des Regelkreises

Um das Systemverhalten angemessen zu erfassen, ziehen wir nochmals die Darstellung einer typischen Regelstrecke aus Unterabschnitt 2.1.1 heran. Diese Visualisierung hilft uns, die Funktionen der einzelnen Komponenten unseres Systems zu verstehen und zuzuordnen.

- **Führungsgröße $w(t)$:** Dies ist der vom Benutzer über die grafische Benutzeroberfläche vorgegebene Sollwert in Grad Celsius, der die Zieltemperatur des Wassers im Behälter definiert.
- **Stellglied:** Das Stellglied setzt sich aus der Phasenanschnittsteuerung und dem Tauchsieder zusammen. Dabei repräsentiert die Stellgröße $u(t)$ die Heizleistung des Tauchsieders, ausgedrückt in Prozent.
- **Regelstrecke:** Die Regelstrecke umfasst den Wasserbehälter und alle relevanten Faktoren, die die Wassertemperatur beeinflussen können. Dazu zählen die Umgebungstemperatur und die Eigenschaften des Behälters wie Materialdicke und Wärmeleitfähigkeit. Auch die Wassermenge und die Verdunstungsrate des Wassers spielen eine wichtige Rolle für die Dynamik der Temperaturänderung.
- **Externe Störgrößen $d(t)$:** In diesem Projekt befindet sich der Wasserbehälter in einem Raum mit minimalen externen Einflüssen. Wäre dies nicht der Fall, könnten externe Störungen wie Temperaturschwankungen im Raum oder direkte Sonneneinstrahlungen auftreten.
- **Messglied:** Das Messglied im System ist der Temperatursensor, der die Regelgröße $y(t)$, also die aktuelle Wassertemperatur im Behälter erfasst.
- **Regler:** Die Implementierung des Reglers findet im Rahmen des Python-Projekts statt. Er verarbeitet die Führungsgröße und die Rückführung des Istwertes um die geeignete Stellgröße für die Temperaturanpassung zu bestimmen.

5.4.3 Reaktionszeit des Temperatursensors

Um ein besseres Verständnis für die Auswirkungen der Reaktionszeit des Temperatursensors auf die Messergebnisse zu erlangen, wurde eine spezifische Messung durchgeführt. Dabei wurde der Wasserbehälter zunächst erhitzt und eine Messaufzeichnung gestartet. Nach ca. 70 Sekunden wurde schlagartig kaltes Wasser hinzugefügt, um die Reaktionsgeschwindigkeit des Sensors auf diese abrupte Temperaturänderung zu testen. Die Ergebnisse dieser Messung sind in Abbildung 5.13 dargestellt, wobei die blaue Linie den Zeitpunkt des Hinzufügens des kalten Wassers markiert. Es dauert etwa 1,5 Sekunden, bis eine sichtbare Reaktion des Sensors auf die Temperaturänderung zu sehen ist. Ab dort zeigt die Temperaturkurve einen exponentiellen fallenden Verlauf. Dieses Verhalten ist typisch für PT1000 Temperatursensoren, da ihre Reaktionszeit aufgrund der physikalischen Eigenschaften und der Wärmeübertragung zwischen Sensor und Umgebung natürlich begrenzt ist.

Angesichts der Tatsache, dass Temperaturänderungen in diesem Projekt generell eher langsam erfolgen und abrupte Sprünge selten sind, erweist sich die Reaktionszeit des Sensors als ausreichend für die Anforderungen. Zur Vereinfachung in zukünftigen Kapiteln, wird angenommen, dass der Sensor den Istwert genau und ohne Verzögerung erfasst, obwohl in Wirklichkeit eine geringfügige Verzögerung und Verfälschung des Messwertes durch den Sensor vorliegt.

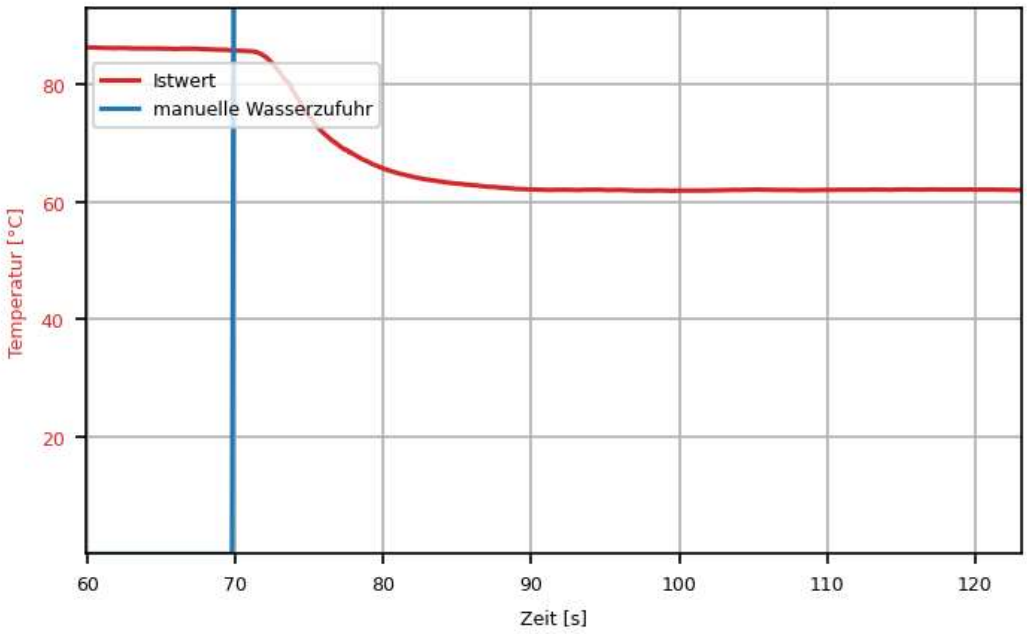


ABBILDUNG 5.13: Messung der Reaktionszeit des Temperatursensors

5.4.4 Wahl des Modells

Wie bereits in Unterabschnitt 2.1.2 diskutiert, kann ein thermischer Prozess wie das Erhitzen eines Wasserbehälters häufig mithilfe eines Systems erster Ordnung, bekannt als PT1-System, modelliert werden. Dieses Modell ist besonders geeignet, um Systeme zu beschreiben, die über einen primären Energiespeicher verfügen. Insbesondere Heizvorgänge, werden häufig mittels eines PT1-Modells simuliert, da es in der Lage ist, die thermische Trägheit und Wärmekapazität des Systems zu erfassen. Darüber hinaus ist die Berücksichtigung einer Totzeit oft notwendig, um die verzögerte Wirkung des Heizelements auf die Wassertemperatur realistisch zu modellieren. Die Totzeit repräsentiert die Zeitdifferenz zwischen dem Moment der Leistungsänderung am Heizelement und dem Beginn der sichtbaren Temperaturänderung im Wasser [68, 72]. Um zunächst das Verhalten eines PT1-Glieds ohne Totzeit im Zeitbereich zu beschreiben, betrachten wir die Differenzialgleichung (DGL) eines einfachen PT1-Glieds.

$$T \cdot \dot{y}(t) + y(t) = K \cdot u(t) \tag{5.9}$$

Hierbei ist:

- $y(t)$ die Ausgangsgröße des Systems (z.B. die Temperatur des Wassers),
- $u(t)$ die Eingangsgröße des Systems (z.B. die Leistung des Heizelements),
- T die Zeitkonstante des Systems. Diese gibt an, wie schnell das System auf Änderungen reagiert. Konkret ist T die Zeit, die das System braucht, um nach

einer Änderung des Eingangssignals 63, 2 % der Gesamtänderung zu erreichen und sich einem neuen stationären Ausgangszustand anzunähern.

- K der Verstärkungsfaktor des Systems, welcher das Verhältnis zwischen der Änderung der Ausgangsgröße und der Änderung der Eingangsgröße im stationären Zustand angibt und somit die Stärke der Systemantwort auf Eingangssignale beschreibt.

In den nachfolgenden Abschnitten wird die Reaktion des Systems auf verschiedene Eingangswerte gemessen und analysiert. Darauf aufbauend werden die entsprechenden Parameter (K, T, Totzeit) identifiziert, um das Systemverhalten mithilfe eines PT1-Glieds mit Totzeit zu beschreiben.

5.4.5 Ermittlung der Totzeit des Systems

In diesem Unterabschnitt liegt der Fokus auf der Analyse und Bestimmung der Totzeit des Systems. Die Totzeit ist die Verzögerung zwischen dem Zeitpunkt einer Aktion, wie der Anpassung der Heizleistung des Tauchsieders, und dem Zeitpunkt, an dem sich die Auswirkung dieser Aktion auf die Temperatur des Wassers im Behälter messen lässt. Zur Ermittlung der Totzeit wird die Systemreaktion auf einen plötzlichen Sprung des Stellwerts von 0 % auf 100 % untersucht. Nachdem das Wasser für eine gewisse Zeit erhitzt wurde, erfolgt eine Rückstellung des Stellwerts von 100 % auf 0 %. Wie in Abbildung 5.14 zu sehen zeigt der Istwert der Temperatur erst 27 Sekunden nach dem ersten Stellwertsprung eine Veränderung und beginnt zu steigen. Nach Abschalten des Tauchsieders ist ein weiterer Temperaturanstieg zu beobachten, welcher nach ungefähr 28 Sekunden zum Stillstand kommt. Basierend auf diesen Beobachtungen wird für das System eine angenäherte Totzeit (T_d) von 27 Sekunden angenommen.

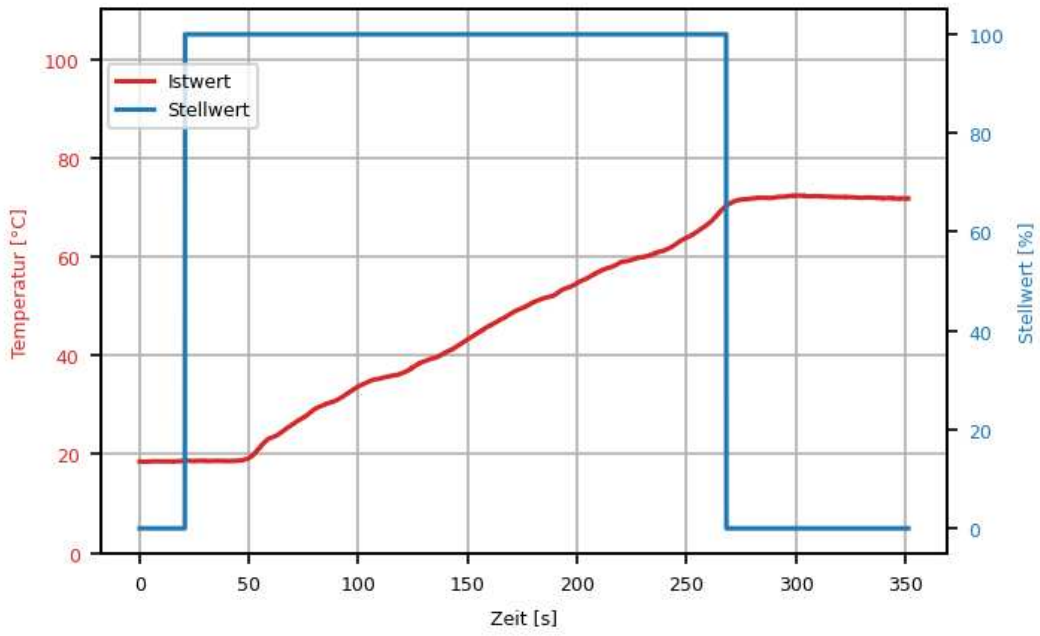


ABBILDUNG 5.14: Messung zur Ermittlung der Totzeit

5.4.6 Ermittlung der PT1-Parameter des Systems

Für die Analyse der PT1-Strecke wird zunächst die erwartete Reaktion und der Verlauf der Sprungantwort betrachtet. Wenn man davon ausgeht, dass das Wasser die vom Tauchsieder bereitgestellte Energie restlos aufnimmt, liegt die Annahme eines anfänglich linearen Temperaturanstiegs nahe. Allerdings beginnt das Wasser, mit zunehmender Erwärmung mehr Wärme an die Umgebung abzugeben. Deshalb nimmt trotz kontinuierlicher Energiezufuhr die Geschwindigkeit, mit welcher die Temperatur steigt ab und die Temperaturkurve wird mit der Zeit flacher. Ab einer bestimmten Temperatur ist die abgegebene Energie gleich der zugeführten Energie, und es stellt sich ein stationärer Zustand ein.

Aus der Sprungantwort eines PT1-Systems lassen sich normalerweise direkt der Verstärkungsfaktor K und die Zeitkonstante T ermitteln. Betrachtet man jedoch die gemessene Sprungantwort bei einer Heizleistung von 0 % auf 100 %, die in Abbildung 5.15 dargestellt ist, zeigt sich, dass die Temperatur zunächst nahezu linear ansteigt und kurz vor dem Siedepunkt bei 100 °C dann abflacht und schließlich konstant bleibt. Das Problem bei der Analyse der Sprungantwort bei voller Heizleistung (100 %) ist, dass die Temperatur nahe des Siedepunkts ihr Verhalten ändert und ab dem Siedepunkt nicht weiter ansteigt. Deshalb ist es schwer, aus dieser Sprungantwort den Verstärkungsfaktor K direkt zu bestimmen. Um trotzdem sinnvolle PT1-Parameter zu erhalten, kann die Heizleistung so weit reduziert werden, dass die Sprungantwort einen eindeutigen stationären Zustand zeigt.

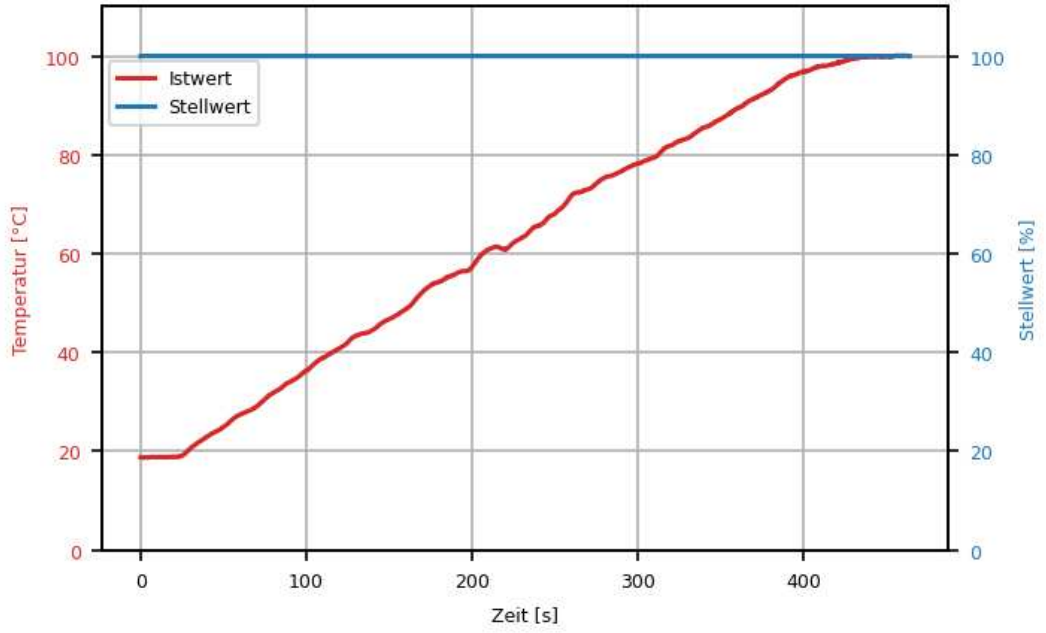


ABBILDUNG 5.15: Sprungantwort bei einer Heizleistung von 0 auf 100%

Die Sprungantwort bei einer Heizleistung von 5 %, gezeigt in Abbildung 5.16, führt zu einem stabilen Zustand bei 89 °C. Aus dieser Beobachtung lassen sich der Verstärkungsfaktor K und die Zeitkonstante T ableiten.

1. **Verstärkungsfaktor**:
   Der Verstärkungsfaktor K ist das Verhältnis der Änderung der Ausgangsgröße

(in diesem Fall die Temperatur) zur Änderung der Eingangsgröße (die Heizleistung). Unter der Annahme, dass sich das System ohne Heizleistung (0 % Stellwert) auf die Umgebungstemperatur von $T_{\text{amb}} = 23\ ^\circ\text{C}$ einstellt und bei einer Heizleistung von 5 % eine stabile Temperatur von 89 °C erreicht, lässt sich K wie folgt berechnen.

$$K = \frac{\Delta T}{\Delta U} = \frac{T_{\text{stationär}} - T_{\text{amb}}}{5\ \% - 0\ \%} = \frac{89\ ^\circ\text{C} - 23\ ^\circ\text{C}}{5\ \%} = 13,2\ (\text{K}\,\%^{-1}) \tag{5.10}$$

2. **Zeitkonstante**:
   Die Zeitkonstante T beschreibt, wie schnell das System auf Änderungen reagiert und errechnet sich aus der Zeit, die das System braucht, um 63,2 % des Wegs zum neuen stabilen Zustand zurückzulegen. Um T zu bestimmen, schaut man in dem Plot in Abbildung 5.16 wann die Temperatur den Wert 64,71 °C erreicht, was 63,2 % der Temperaturänderung entspricht. Die bis zu diesem Zeitpunkt verstrichene Zeit repräsentiert die Zeitkonstante. Aus der Grafik ermittelt, resultiert somit ein Wert von $T = 6000$ Sekunden.

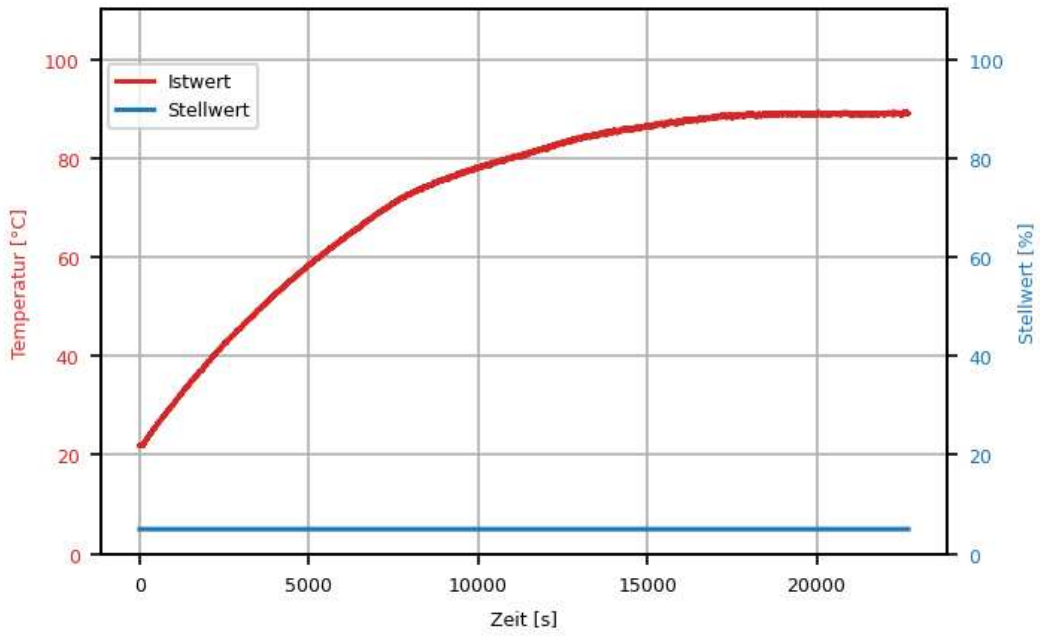
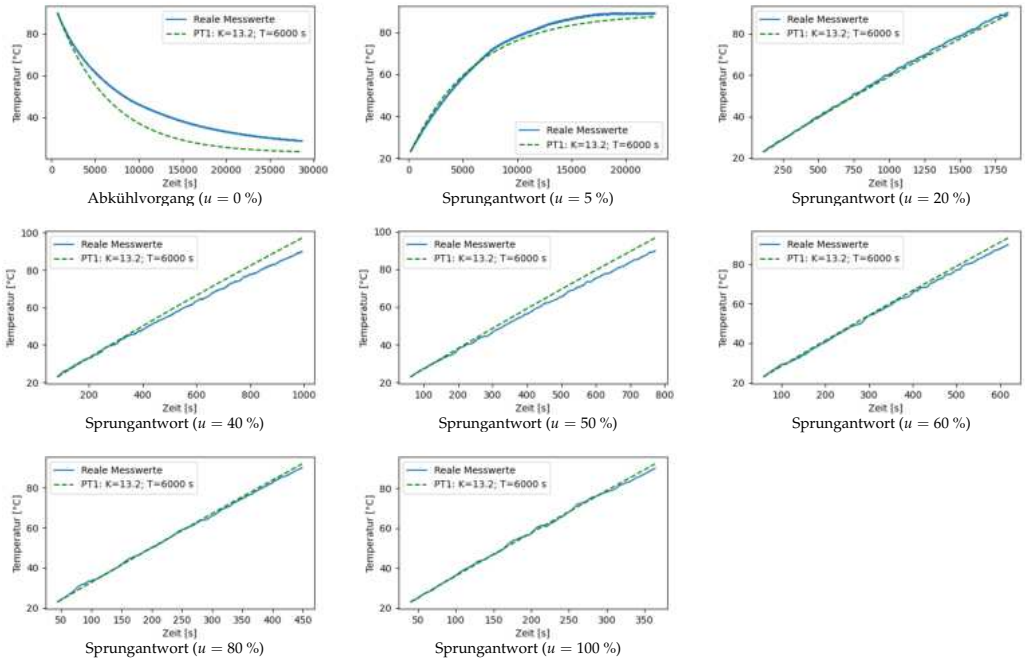


ABBILDUNG 5.16: Sprungantwort bei einer Heizleistung von 0 auf 5%

Für die Analyse der Sprungantworten des Wasserbeckens bei weiteren Leistungsstufen des Tauchsieders wurde ein Python-Skript „GetPt1CharacteristicODE.py“ entwickelt. Dieses Skript nutzt die Differenzialgleichung eines PT1-Gliedes, wie in Gleichung 5.9 beschrieben, um einen grafischen Vergleich zwischen dem theoretischen Verlauf eines PT1-Systems und dem tatsächlichen Verhalten des Wasserbeckens zu ermöglichen. Um die realen Bedingungen präzise abzubilden, wurde die Gleichung um eine Offset-Verschiebung erweitert. Dies berücksichtigt die Feststellung, dass das Wasser bei einer Leistung von $U = 0$ % eine Starttemperatur von 23 °C zeigt, entsprechend der Umgebungstemperatur. Die angepasste Differenzialgleichung, die das System beschreibt, lautet:

$$T \cdot \dot{y}(t) + y(t) = K \cdot u(t) + 23\,^{\circ}\mathrm{C} \tag{5.11}$$

In Abbildung 5.17 sind sowohl Messdaten des Abkühlprozesses als auch Daten der Sprungantworten bei verschiedenen Leistungsstufen visualisiert. Dabei markiert die blaue Linie die realen Messwerte, während die grün gestrichelte Linie den theoretisch vorhergesagten Verlauf eines PT1-Systems mit den oben ermittelten Parametern gemäß der DGL in Gleichung 5.11 abbildet.



Abkühlvorgang ($u = 0\,\%$) Sprungantwort ($u = 5\,\%$) Sprungantwort ($u = 20\,\%$)

Sprungantwort ($u = 40\,\%$) Sprungantwort ($u = 50\,\%$) Sprungantwort ($u = 60\,\%$)

Sprungantwort ($u = 80\,\%$) Sprungantwort ($u = 100\,\%$)

ABBILDUNG 5.17: Vergleich PT1 System mit Realen Messdaten

Die Analyse der Ergebnisse zeigt, dass die Annäherung des Modells an die realen Messdaten durchaus zufriedenstellend ist. Auf Grundlage dieser Erkenntnisse wird in den folgenden Kapiteln angenommen, dass das Systemverhalten durch ein PT1-Modell mit den Parametern $K = 13,2$ und $T = 6000$ s angenähert werden kann.

Kapitel 6

Bewertungsmethodik für Regelungsmethoden

In den nachfolgenden Kapiteln erfolgt die Untersuchung verschiedener Regelungsmethoden zur Temperaturregelung des Wasserbeckens. Um eine objektive und gründliche Bewertung zum Vergleich dieser Ansätze zu gewährleisten, bedarf es einer systematischen Bewertungsmethodik, welche in diesem Kapitel definiert werden soll. Aspekte wie Energieeffizienz, der Aufwand für die Implementierung sowie die Flexibilität in Bezug auf wechselnde Umgebungsbedingungen stellen zentrale Bewertungskriterien für Regelungssysteme dar, welche gegen Ende dieser Arbeit noch betrachtet werden. Zunächst liegt der Fokus aber auf der Definition eines Prüfverfahrens, das die Regelqualität und Regelgüte bewertet. Aus diesem Grund wird ein spezifischer Testablauf festgelegt, der es ermöglicht, einen direkten Vergleich der Systeme unter einheitlichen Bedingungen durchzuführen.

6.1 Testablauf

Für die Auswahl der Sollwerte eines geeigneten Testablaufes ist es wichtig, ein breites Spektrum an Betriebsbedingungen abzudecken, um die Leistungsfähigkeit und Flexibilität der Regelungsmethoden umfassend zu bewerten. Daher werden Sollwerte sowohl im niedrigen, mittleren als auch im hohen Temperaturbereich festgelegt. Dabei sollen geringfügigen Anpassungen des Sollwertes bis hin zu großen sprunghaften Temperaturänderungen untersucht werden. Zusätzlich werden Störgrößen in Form von kaltem Wasser eingeführt, um die Reaktion und Robustheit der Regelung gegenüber unvorhergesehenen Veränderungen zu testen. Außerdem ist interessant, wie präzise und schnell die Systeme nach einer Abkühlphase den definierten Sollwert erreichen und stabilisieren können. Aufbauend auf diesen Überlegungen ergibt sich folgender Testablauf:

- **Schritt 1: Erwärmen auf Ausgangspunkt 30 °C**
- **Schritt 2: Geringfügiges Erwärmen auf 31,5 °C**
- **Schritt 3: Sprunghafte Sollwertänderung auf 40 °C**
- **Schritt 4: Sprunghafte Sollwertänderung auf 80 °C**
- **Schritt 5: Sollwertänderung auf 75 °C (Abkühlvorgang)**
- **Schritt 6: Neu befüllen des Behälters mit kaltem Wasser**
- **Schritt 7: Sollwert 55 °C mit Störgröße**
  Kurz bevor der Zielwert von 55 °C erreicht wird, erfolgt bei einer Temperatur von 53 °C die Zugabe von 200 ml kaltem Wasser.

- **Schritt 8: Neu befüllen des Behälters mit kaltem Wasser**
- **Schritt 9: Erhitzen auf 65 °C**
- **Schritt 10: Abkühlen auf 60 °C mit Störgröße**
  Kurz bevor der Zielwert von 60 °C erreicht wird, erfolgt bei einer Temperatur von 61 °C die Zugabe von 200 ml kaltem Wasser.
- **Schritt 11: Erhitzen auf 80 °C**
  Untersucht wird, wie die veränderte Wassermenge das Regelverhalten beeinflusst, im Vergleich zu den Ergebnissen aus Schritt 4.
- **Schritt 12: Neu befüllen des Behälters mit kaltem Wasser**
- **Schritt 13: Rampe als Sollwertvorgabe**
  Der Sollwert folgt einer linear ansteigenden Rampe mit einer Steigung von $0,05\ \mathrm{K\,s^{-1}}$. Diese Steigung kann ungefähr bei einer konstanten Heizleistung von 50 % erreicht werden.

6.2 Bewertungskriterien

Dieser Abschnitt zielt darauf ab, die maßgeblichen Parameter und Merkmale zu definieren, anhand derer die Qualität der unterschiedlichen Regelungsmethoden beurteilt werden kann. Bei einem Regler geht es darum den Sollwert möglichst schnell und mit möglichst wenig Überschwingen zu erreichen. In Abbildung 6.1 wird eine beispielhafte Änderung eines Sollwertes sowie die daraus resultierende Sprungantwort dargestellt. Die in der Grafik hervorgehobenen Bewertungskriterien für die Sprungantwort sind nachfolgend detailliert erläutert.

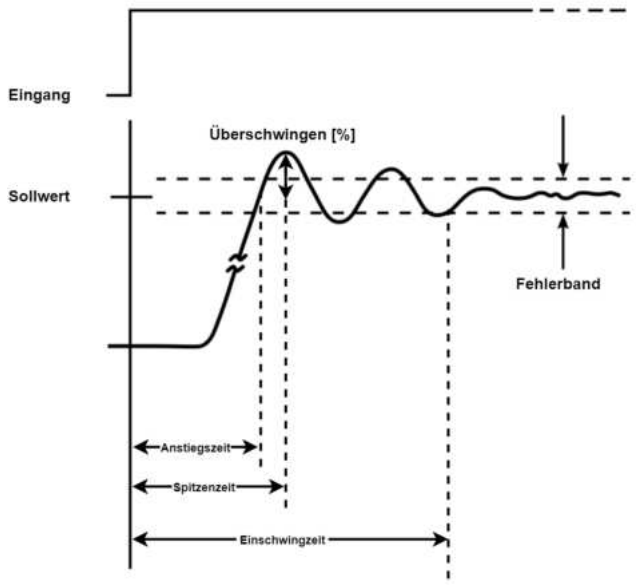


ABBILDUNG 6.1: Spezifikationen für das Einschwingverhalten mit Elementen aus [60]

1. **Stationärer Fehler:**
   Der stationäre Fehler (Steady State Error) ist die dauerhafte Abweichung zwischen der Regelgröße und ihrem Sollwert im stabilen Endzustand des Systems. Das Ziel eines Regelsystems ist diesen stationären Fehler zu minimieren.

2. **Anstiegszeit:**
   Die Zeitspanne, welche benötigt wird, bis die Regelgröße erstmals einen bestimmten Prozentsatz ihres Sollwertes erreicht. Sie ist ein Maß für die Schnelligkeit der Systemreaktion [49]. In diesem Projekt wird zwischen zwei spezifischen Anstiegszeiten unterschieden: Zum einen wird die Zeit gemessen, die benötigt wird, bis 90 % des Sollwertes erreicht sind ($Anstiegszeit_{90\%}$), und zum anderen die Zeitspanne bis zur vollständigen Erreichung des Sollwertes, also 100 % ($Anstiegszeit_{100\%}$).

3. **Spitzenzeit**:
   Die vergangene Zeit bis die Regelgröße ihren ersten maximalen Wert erreicht [49].

4. **Einschwingzeit**:
   Die Zeit, die vergeht, bis die Regelgröße stabil innerhalb eines bestimmten Toleranzbereichs um den Sollwert bleibt. Sie ist ein Indikator dafür, wie schnell das System sich nach einer Störung stabilisiert [49]. Für dieses Projekt wurde ein Fehlerband von 3 % gewählt. Diese Entscheidung berücksichtigt die benötigte Präzision und das Messrauschen. Bei der Temperaturregelung des Wasserbeckens wird eine Abweichung bis 3 % vom Sollwert als ausreichend genau betrachtet.

5. **Überschwingweite**:
   Dieser Wert beschreibt wie stark die Regelgröße über den Sollwert hinausschießt, bevor sie ihren endgültigen Wert erreicht. Üblicherweise wird das Überschwingen in Prozent des Sollwerts ausgedrückt [49].

6. **Integral of Time-multiplied Absolute value of Error (ITAE):**
   Das ITAE ist ein Maß, das den Fehler über die Zeit bewertet, wobei die Fehler mit der Zeit immer stärker gewichtet werden. Es bewertet sowohl die Genauigkeit als auch die Geschwindigkeit, mit der das System den Sollwert erreicht und beibehält. Durch den Gewichtungsfaktor t wird eine schnelle Stabilisierung des Systems als vorteilhaft bewertet. Das ITAE kann mit folgender Formel berechnet werden [63].

$$ITAE = \int_0^\infty t \cdot |e(t)|\, dt \tag{6.1}$$

7. **Integral of Squared Error (ISE):**
   Das ISE ist ein Gütemaß, das die quadratischen Fehler über die Zeit integriert. Im Gegensatz zum ITAE, welches den Fehler über die Zeit gewichtet, legt das ISE ein stärkeres Gewicht auf größere Fehler, da die Fehlerquadrate summiert werden. Dies macht das ISE besonders empfindlich gegenüber großen Abweichungen vom Sollwert. Das ISE ist nützlich, um die Gesamtenergie des Fehlers über die Zeit zu bewerten und wird oft in Anwendungen verwendet, wo große Fehler besonders unerwünscht sind [35]. Es kann mit der folgenden Formel berechnet werden:

$$ISE = \int_0^\infty e(t)^2 dt \tag{6.2}$$

   Die Minimierung des ISE führt zu einer Regelung, die große Abweichungen vom Sollwert effektiv reduziert, was für viele Anwendungen wünschenswert ist.

6.3 Relevanz der Bewertungskriterien

Es ist nicht möglich, alle Bewertungskriterien für Regelungsprozesse gleichzeitig zu optimieren, da die Wichtigkeit dieser Kriterien je nach Anwendungsfall variiert. Daher ist es wichtig, zu Beginn eines Projekts festzulegen, welche Kriterien Vorrang haben. Zum Beispiel könnte bei einigen Aufgaben ein schnelles Erreichen des Zielwerts wichtiger sein, auch wenn dabei ein größeres Überschwingen toleriert wird. Bei anderen Anwendungen wiederum ist es entscheidend, Überschwingen komplett zu vermeiden. Für dieses Projekt muss also klar definiert werden, welche Kriterien am wichtigsten sind, um die Regelungsstrategie entsprechend auszurichten.

Bei der Temperaturregelung eines Wasserbeckens liegt der Fokus darauf, die Differenz zwischen der tatsächlichen und der gewünschten Temperatur so gering wie möglich zu halten. Deshalb sind das ISE und ITAE gute Kriterien, um die Zielsetzung des Projekts zusammenfassend auszudrücken. Das ISE gewichtet größere Abweichungen mehr, während das ITAE die Abweichungen mit der Zeit immer mehr gewichtet. Obwohl diese beiden Kriterien in der Theorie umfassend sind, bleibt die Bewertung von Anstiegszeiten, Spitzenzeit und Einschwingzeit relevant, um die Schlussfolgerungen von ISE und ITAE zu überprüfen. Zudem ist es essenziell, den stationären Fehler zu betrachten, wobei leichte Schwankungen, verursacht durch Messrauschen des Temperaturfühlers, akzeptabel sind. Die Überschwingweite spielt in diesem Projekt eine untergeordnete Rolle. Entscheidender sind die Auswirkungen eines Überschwingens, die sich in den anderen Bewertungskriterien niederschlagen. Ein moderates Überschwingen ist vertretbar, sofern es den Regelprozess beschleunigt. Da allerdings bei dem System kein aktives Kühlen möglich und der natürliche Abkühlvorgang sehr langsam ist, führt eine Übersteuerung des Istwertes meistens zu einer lang anhaltenden Regelabweichung.

Das ISE und ITAE wird somit zusammen mit dem Stationären Fehler zum Hauptbewertungskriterium in diesem Projekt, wobei darauf geachtet werden muss, dass der Zeitraum, über den das Integral berechnet wird, bei den zu vergleichenden Messungen vergleichbar bleibt und nicht stark variiert. Die anderen Bewertungskriterien bleiben weiterhin relevant und dienen als zusätzliche Validierung der Ergebnisse.

Kapitel 7

Design und Implementierung eines PID-Reglers

7.1 Theoretische Grundlagen des PID-Reglers

Ein PID-Regler ist ein weit verbreitetes Instrument in der Regelungstechnik, das zur Regelung von Prozessen eingesetzt wird. Dieser lässt sich durch eine Parallelschaltung eines P-, eines I- und eines D-Reglers mit den Koeffizienten K_P , T_I und T_D realisieren. PID steht für Proportional-, Integral- und Differenzialregelung, wobei jeder Teil des Reglers eine spezifische Aufgabe übernimmt [52, S. 146 ff.].

- **Proportionaler Anteil:** Dieser reagiert direkt auf die aktuelle Regelabweichung $e(t)$ zwischen dem Ist- und dem Sollwert. Der Wert des P-Anteils ist proportional zur Regelabweichung. Die Formel lautet:

$$P = K_P \cdot e(t) \tag{7.1}$$

- **Integraler Anteil:** Der Integralanteil summiert die Regelabweichungen über die Zeit. Dies hilft, den systematischen Fehler zu eliminieren und sorgt dafür, dass der Istwert den Sollwert exakt erreicht. Der I-Anteil kann folgendermaßen berechnet werden:

$$I = \frac{1}{T_I} \cdot \int_0^t e(\tau) d\tau \tag{7.2}$$

- **Differenzialer Anteil:** Der Differenzialteil reagiert auf die Änderungsrate der Regelabweichung. Er kann zukünftige Abweichungen antizipieren und das System stabilisieren, indem er übermäßige Reaktionen oder Schwingungen dämpft. Er wird folgendermaßen berechnet:

$$D = T_d \cdot \frac{d}{dt} e(t) \tag{7.3}$$

Die Formel für den PID-Regler wird häufig in einer alternativen Produktform dargestellt, die durch die Koeffizienten K_P, T_n und T_v charakterisiert wird. Hierbei wird T_n als Nachtstellzeit und T_v als Vorhaltezeit bezeichnet [59].

$$u(t) = K_P \cdot \left(e(t) + \frac{1}{T_n} \int_0^t e(\tau) d\tau + T_v \frac{d}{dt} e(t) \right) \tag{7.4}$$

7.2 Digitalisierung und Implementierung des PID-Reglers

Um den analogen PID-Regler im Python-Projekt implementieren zu können, ist eine Digitalisierung des Regelmechanismus erforderlich. Für diese Digitalisierung wird die Rückwärts-Differenzierungsmethode, auch bekannt als Euler-Methode, herangezogen. Diese Methode approximiert die Ableitung eines Signals durch die Bildung der Differenz zwischen dem aktuellen Wert und dem vorherigen Wert, dividiert durch die Abtastzeit:

$$\Delta x[n] = \frac{x[n] - x[n-1]}{T_s} \tag{7.5}$$

Nun wird die Euler Methode auf den durch Gleichung 7.4 beschriebenen PID-Regler angewendet. Somit ergibt sich unter Berücksichtigung der Regelabweichung $e[n]$ und der Abtastzeit T_s folgender Regelalgorithmus.

$$P[n] = K_P \cdot e[n] \tag{7.6}$$

$$I[n] = I[n-1] + \frac{K_P}{T_n} \cdot e[n] \cdot T_s \tag{7.7}$$

$$D[n] = K_P \cdot T_v \cdot \frac{e[n] - e[n-1]}{T_s} \tag{7.8}$$

$$u[n] = P[n] + I[n] + D[n] \tag{7.9}$$

Filterung des D-Anteils:
In der Praxis wendet man auf die Regelabweichung des Differenzialanteils in PID-Reglern oft einen Filter an, um das durch Messrauschen verursachte hochfrequente Signalrauschen zu reduzieren und so die Stabilität und Effektivität der Regelung zu verbessern. Für die Filterung wird ein Tiefpassfilter mit folgender Formel angewendet. Hierbei steht e_f für die gefilterte Regelabweichung.

$$e_f[n] = \alpha \cdot e[n] + (1 - \alpha) \cdot e_f[n-1] \tag{7.10}$$

Der D-Anteil lässt sich somit mit der folgenden angepassten Formel berechnen:

$$D[n] = K_P \cdot T_v \cdot \frac{e_f[n] - e_f[n-1]}{T_s} \tag{7.11}$$

Der Parameter α bestimmt die Glättungsstärke des Filters, wobei ein höheres α dem aktuellen Messwert mehr Gewicht gibt und somit weniger Glättung bewirkt. Alpha kann durch $\alpha = \frac{T_s}{T_f}$ angenähert werden, wobei T_f die Filterzeitkonstante ist. Das ist die Zeit nach welcher der Tiefpassfilter etwa 63,2 % des Endwerts einer sprunghaften Änderung des Eingangssignals erreicht hat. Bei der Auswahl von T_f muss beachtet werden, diese bedeutend größer als die Abtastzeit (0,1 Sekunden) zu wählen. Da das zu regelnde System eher träge ist, wird ein T_f von 2 Sekunden gewählt um hochfrequentes Rauschen effektiv zu filtern, ohne dabei die Systemdynamik zu sehr zu beeinflussen [41].

Anti-Windup
Wenn beim PID-Regler die Stellgröße ihre physikalischen Limits erreicht, tritt ein Wind-up-Effekt auf. Dabei arbeitet die Integration des Reglers weiter, ohne dass die Stellgröße zunimmt. Dies kann zu einer übermäßigen Überschwingung und verzögerten Stabilisierung des Systems führen, wenn das Stellglied wieder in einen

nicht gesättigten Zustand zurückkehrt [70]. Um dies zu verhindern wurde eine Anti-Windup-Maßnahme implementiert. Diese greift ein, sobald die Summe aus dem Proportional- und Integralanteil den maximalen oder minimalen Stellwert überschreitet. Sie friert dann den Integralanteil ein und verhindert somit ein weiteres Anwachsen oder Absinken.

Implementierung des PID Reglers im Programm:
Die Umsetzung des PID-Reglers erfolgt durch die Klasse „Pid“, die alle zugehörigen Prozesse steuert. Sie aktiviert den Regler mit einer Abtastzeit von 100 ms und protokolliert Istwerte, Sollwerte sowie Stellgrößen zusammen mit Zeitstempeln in einer Datei. Die Einstellungen für die Parameter des PID-Reglers können bequem über eine grafische Benutzeroberfläche, die in Abbildung 7.1 zu sehen ist, vorgenommen werden. Der Code für die Implementierung des PID-Regelalgorithmus, ist im Anhang unter Unterabschnitt A.4.1 dargestellt.

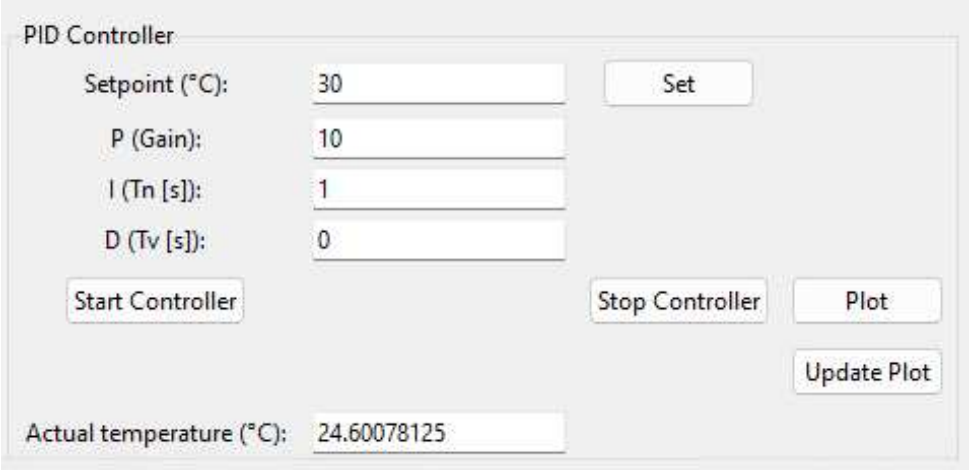


ABBILDUNG 7.1: Benutzeroberfläche PID-Regler

7.3 Bestimmung der PID-Regelparameter

Theoretische Grundlagen

Die Ermittlung der PID-Parameter stellt oft eine herausfordernde Aufgabe dar, deren Erfolg maßgeblich von der Genauigkeit des verfügbaren Modells der Regelstrecke abhängt. Es existieren vielfältige Methoden zur Parametereinstellung, darunter die Analyse mittels Bode-Diagramm, Wurzelortskurve oder im Pol-Nullstellen-Diagramm. Zusätzlich bieten spezialisierte Softwaretools Unterstützung bei der Analyse und Feinabstimmung der Reglerparameter. Bewährte Faustformeln, wie die von Ziegler-Nichols oder das Verfahren nach Chien, Hrones und Reswick, bieten praktische Anhaltspunkte für eine erste Parametereinstellung. Simulationen ergänzen das Spektrum der genannten Methoden zusätzlich [58]. Während Ansätze wie der Reglerentwurf basierend auf der Wurzelortskurve oder dem Bode-Diagramm fundiertes Fachwissen in Regelungstechnik und ein tiefgreifendes Verständnis der Regelstrecke voraussetzen, zielen andere Methoden darauf ab, auch ohne umfangreiche Vorkenntnisse zu funktionierenden PID-Parametern zu führen. Hierbei liegt der Nachteil oft in einer weniger präzisen Anpassung an das spezifische System.

Trotz Anwendung dieser Verfahren ist es in der Praxis oft notwendig, dass ein Experte den Regler später durch gezieltes Experimentieren und Anpassen feinjustiert.

Möchte man den Regler und das System nun mithilfe des Bode-Diagramms analysieren, ist dafür eine mathematische Beschreibung für das System notwendig. Der für dieses Projekt entwickelte PID-Regler weist mehrere besondere Eigenschaften auf, welche eine mathematische Modellierung mit anschließender Analyse sehr komplex und zeitaufwändig machen. Zu diesen Eigenschaften gehören beispielsweise der Anti-Windup-Mechanismus und eine Begrenzung des Stellbereichs, die den Stellwert im negativen Bereich und oberhalb von 100 % limitiert. Außerdem ergeben sich Herausforderungen durch den zusätzlichen Parameter für den Tiefpassfilter im Differenzialanteil oder aus der Digitalisierung und der diskreten Abtastung des Systems.

In diesem Projekt, das sich durch ein überschaubares Regelungsproblem auszeichnet, wird ein pragmatischer Ansatz gewählt, der auf bewährten Faustregeln und gezieltem Experimentieren basiert. Es wird erwartet, dass man so relativ schnell zu einem guten Ergebnis kommt.

Entwurf nach Faustformelverfahren
Faustformelverfahren, wie die Methoden von Ziegler und Nichols oder die Einstellregeln nach Chien, Hrones und Reswick, sind praxisorientierte Ansätze zur Einstellung der Parameter von PID-Reglern. Diese Verfahren basieren auf empirischen Erkenntnissen und wurden durch Beobachtung des Regelverhaltens verschiedener Systeme unter Standardbedingungen entwickelt. Die Annahme hinter diesen Methoden ist, dass sich viele industrielle Regelungsprobleme trotz ihrer Unterschiedlichkeit in bestimmten grundlegenden Verhaltensweisen ähneln und daher mit einem einheitlichen Satz von Regelparametern effektiv gesteuert werden können. Ziel dieser Faustformelverfahren ist es, eine akzeptable Regelgüte mit einfacher und schneller Einstellung zu erreichen, auch wenn dies möglicherweise nicht die theoretisch optimale Lösung für jedes einzelne System darstellt. Diese Methoden basieren auf der Auswertung der Totzeit T_d, der stationären Verstärkung K und der Zeitkonstante T, die direkt aus der Sprungantwort des Regelkreises entnommen werden können. Mit diesen drei Werten lassen sich dann mithilfe spezifischer Tabellen die Einstellungen für P-, PI- oder PID-Regler bestimmen [61]. Im folgenden Unterabschnitt wird die detaillierte Anwendung der Faustformelverfahren nicht weiter ausgeführt. Für die spezifischen Tabellen und eine umfassende Anleitung sei auf die entsprechende Literatur verwiesen, beispielsweise in [61] oder dem Vorlesungsskript aus Regelungstechnik 1 [6]. Die Regeln von Chien, Hrones und Reswick bieten die Möglichkeit, zwischen einem Fokus auf Störverhalten oder Führungsverhalten zu wählen. In diesem Projekt sind kontinuierliche Störgrößen, kaum vorhanden. Daher werden die Einstellungen bevorzugt, die das Führungsverhalten verbessern.

Praktische Umsetzung

Umsetzung mittels P-Regler
Zu Beginn wird die Regelung ausschließlich mit einem P-Regler durchgeführt, um ein grundlegendes Verständnis für das Verhalten der Regelstrecke zu entwickeln. Für diesen Zweck wurden verschiedene Messreihen bei einem Sollwert von 60 °C aufgenommen. Zur Analyse dieser Messungen dient das selbst erstellte Python-Skript „EvaluateMeasurement.py“, das ein Messfenster von bis zu 1000 Sekunden

auswertet. Dieses Skript führt automatisch Berechnungen durch, um den Maximalwert, die Überschwingweite, Anstiegs-, Spitzen- und Einschwingzeit zu bestimmen. Zudem werden das ISE und ITAE evaluiert. Die Zeitpunkte in diesen Messungen beziehen sich alle auf eine festgelegte Referenzzeit, die dem Moment entspricht, in dem die Ist-Temperatur die Marke von 25 °C überschreitet. Für die Berechnung der Integrale ISE und ITAE wird ein Zeitraum von 900 Sekunden, beginnend ab der Referenzzeit, herangezogen. Die durchgeführten Messungen mit unterschiedlichen Verstärkungsfaktoren K_P werden in Abbildung 7.2 dargestellt. Eine vergrößerte Ansicht dieser Messergebnisse findet sich in Anhang B für eine detailliertere Betrachtung. Bei der Auswahl des Verstärkungsfaktors kamen zunächst die Faustformelverfahren zum Einsatz, wobei die Berechnung auf Basis der Streckenparameter $K = 13,2; T = 6000; T_d = 27$ s erfolgte. Die Auswahl der P-Faktoren für die weiterführenden Messungen erfolgten auf Grundlage von Einschätzungen, die sich aus den Ergebnissen vorheriger Messreihen ergaben, um Aspekte zu betrachten, die besonders aufschlussreich erschienen.

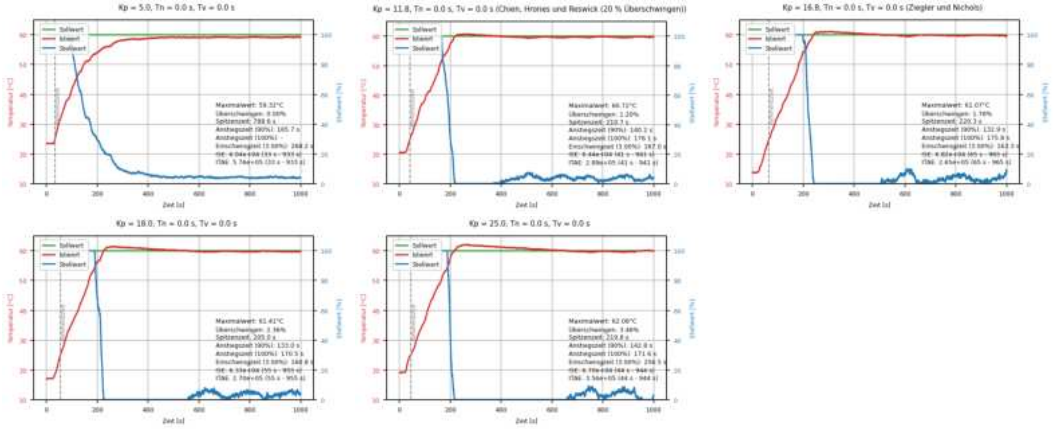


ABBILDUNG 7.2: Vergleich P-Regler mit verschiedener Verstärkung

Eine Erhöhung des P-Anteils bei einem Regler führt typischerweise zu einem steileren Anstieg des Istwertes, was eine kürzere Anstiegszeit und in der Regel auch ein stärkeres Überschwingen zur Folge hat. Die Auswertung der Messdaten zeigt, dass dieses Verhalten tendenziell erkennbar ist, allerdings nicht durchgängig. In der Theorie sollte ein höherer P-Anteil nicht mit einer längeren Anstiegszeit einhergehen, was jedoch hier teilweise der Fall ist. Dies verdeutlicht, dass bei den Messungen bestimmte Toleranzen existieren und eine absolute Reproduzierbarkeit nicht gegeben ist. Dieser Aspekt sollte bei der Auswertung berücksichtigt werden. Auch anhand des ISE ist das beschriebene Verhalten zu erkennen. Zusammengefasst lässt sich feststellen, dass bereits mit einem P-Regler eine relativ gute Regelung erzielt werden kann. Jedoch führt die Charakteristik der Strecke ohne Ausgleich zu einer permanenten Regelabweichung, die hier zwar relativ gering ausfällt, aber dennoch die Einführung eines I-Anteils erforderlich macht, um diese vollständig zu kompensieren.

Umsetzung mittels PI-Regler

Für den PI-Regler wurden zunächst Messungen unter Verwendung der durch die beiden Faustformelverfahren ermittelten Parameter durchgeführt. Anschließend erfolgte eine manuelle Feinabstimmung dieser Werte. Die Ergebnisse dieser Messungen sind in Abbildung 7.3 dargestellt.

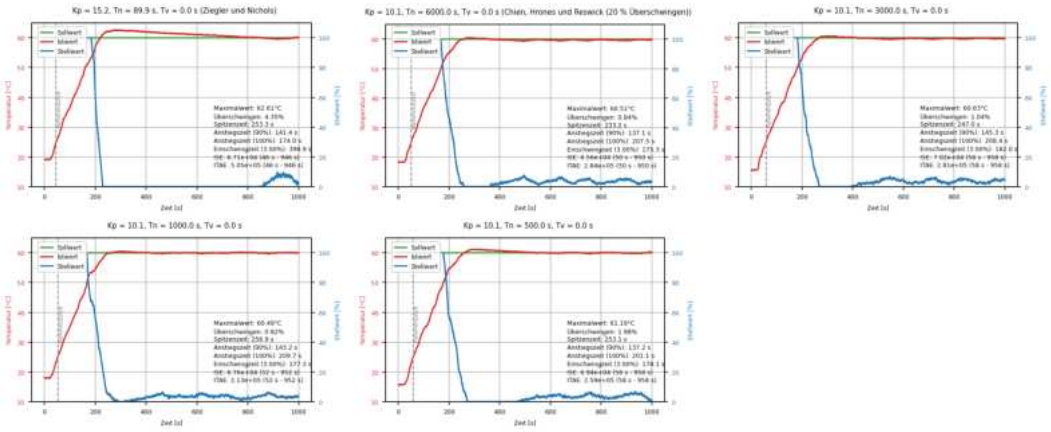


ABBILDUNG 7.3: Vergleich PI-Regler

Vergleicht man die Messungen, sind in der Anstiegszeit nur geringe Unterschiede zu erkennen. Allerdings führt ein zu hoher I-Anteil (kleines T_n) zu einem verstärkten Überschwingen. Eine detaillierte Analyse der Sprungantwort und des stationären Fehlers zeigt, dass die Messung mit $K_P = 10.1$ und $T_n = 1000$ das überzeugendste Gesamtergebnis liefert. Dieses Ergebnis spiegelt sich ebenfalls im ITAE wider.

Umsetzung mittels PID-Regler
Die Ergebnisse der Implementierung eines PID-Reglers mit verschiedenen Parametern sind in Abbildung 7.4 dargestellt.

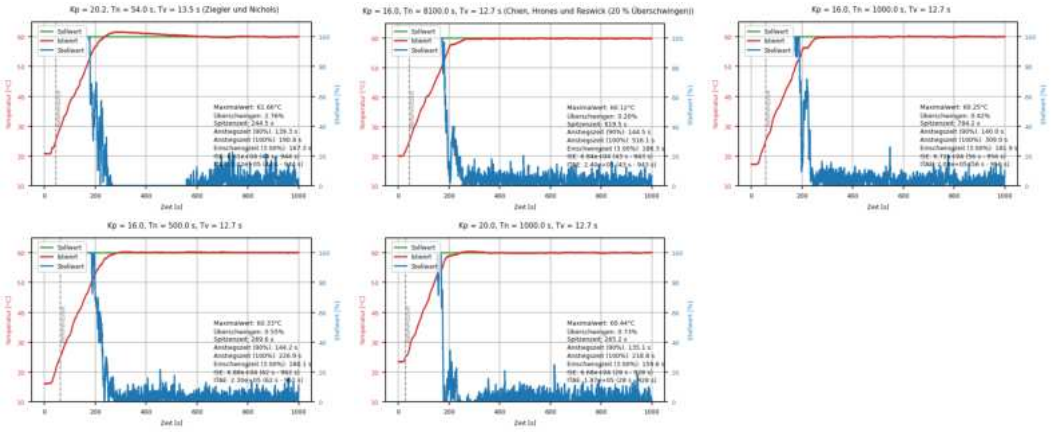


ABBILDUNG 7.4: Vergleich PID-Regler

Die detaillierte Auswertung der Messdaten zeigt, dass die zuletzt dargestellte Messung mit den Regelparametern $K_P = 20$, $T_n = 1000$ und $T_v = 12,7$ zum bisher besten Ergebnis führt. Es sollte jedoch erwähnt werden, dass sich nur geringfügige Verbesserungen im Vergleich zum PI-Regler offenbaren.

Zusammenfassend kann man sagen, dass sowohl der PI-Regler, als auch der PID-Regler zufriedenstellende Ergebnisse liefert. Dabei ist allerdings zu berücksichtigen, dass diese Ergebnisse spezifisch für einen festgelegten Sollwert von 60 °C ermittelt wurden. Eine erneute Bewertung der PID-Parameter bei einem angepassten Sollwert von 30 °C (Abbildung 7.5) zeigt, dass das Überschwingen über den Sollwert in diesem niedrigeren Temperaturbereich etwas stärkere Auswirkungen hat, vor allem weil der Abkühlprozess sehr träge ist. Trotzdem liefern die Regler auch unter diesen

Bedingungen akzeptable Leistungen. Folglich werden die PID-Parameter $K_P = 20$, $T_n = 1000$ und $T_v = 12,7$ als zufriedenstellend eingestuft und für zukünftige Experimente übernommen.

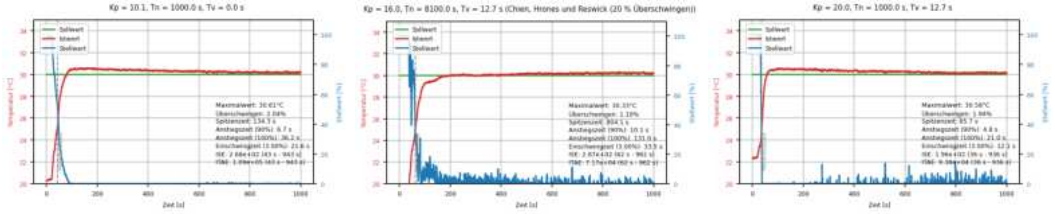


ABBILDUNG 7.5: Vergleich PID-Regler bei Sollwert 30 °C

7.4 Testdurchlauf PID-Regler

Basierend auf den im vorherigen Kapitel definierten PID-Reglerparametern ($K_P = 20$, $T_n = 1000$ und $T_v = 12,7$) wird nun der in Abschnitt 6.1 beschriebene Testdurchlauf durchgeführt und dokumentiert. Die daraus resultierenden Ergebnisse sind in Abbildung 7.6 zu sehen.

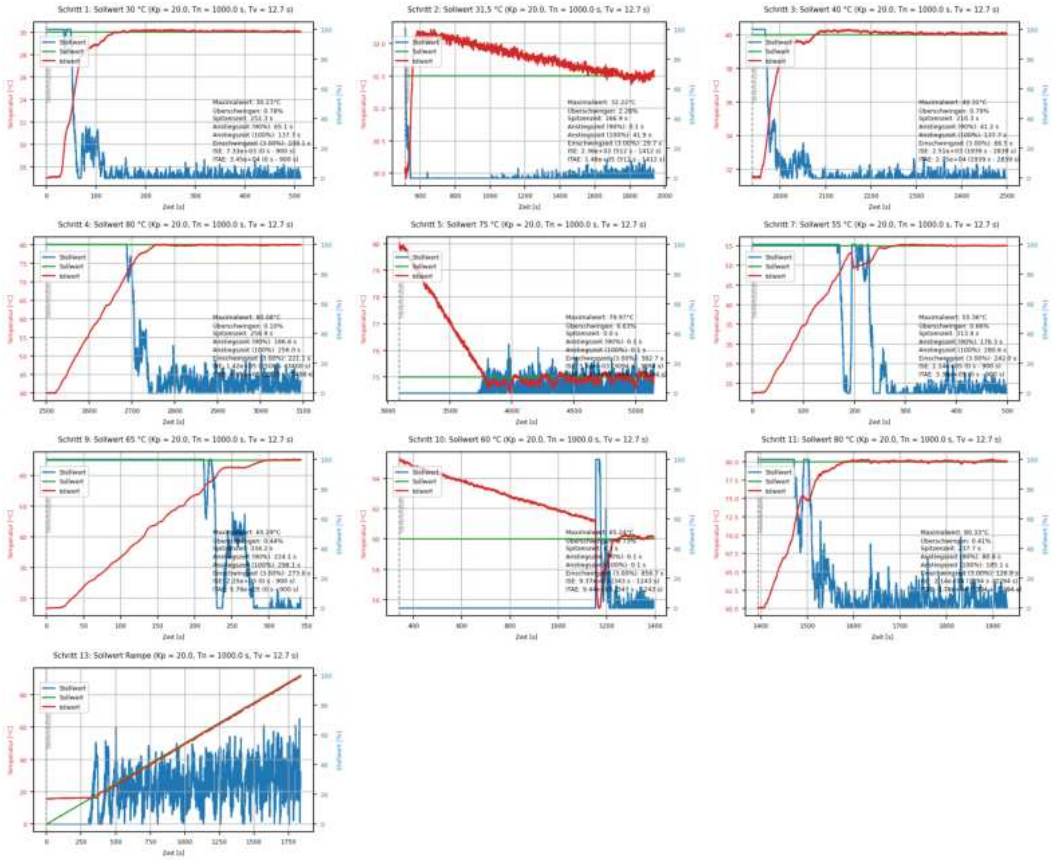


ABBILDUNG 7.6: Testdurchlauf PID-Regler

Die Analyse des Testdurchlaufes zeigt, dass bei der geringfügigen Sollwertanpassung von 30 °C auf 31,5 °C in Schritt 2 ein leichtes Überschwingen zu beobachten ist. Folglich benötigt das System eine gewisse Zeit, um sich abzukühlen und den neuen Sollwert zu stabilisieren. Bei den restlichen Messungen zeigt der Regler ein durchaus akzeptables Regelverhalten, einschließlich der Reaktion auf Störeinflüsse.

7.5 Fazit

Dieses Kapitel hat gezeigt, dass PID-Regler schnell und effektiv funktionieren können, allerdings ist häufig ein gewisses Maß an Experimentieren und Feinjustierung erforderlich. Mit zunehmender Systemkomplexität steigt auch die Schwierigkeit bei der Reglerentwicklung. Trotz verfügbarer Optimierungstechniken und Software Tools können Situationen entstehen, in denen einfache PID-Regler an ihre Grenzen gelangen. Auch wenn dieses Kapitel nicht auf fortgeschrittene Reglerentwurfsmethoden eingeht, ist es erfahrenen Fachleuten bekannt, dass der Entwurfsprozess mit der Komplexität des Systems zunehmend komplizierter und zeitaufwendiger wird.

Kapitel 8

Design und Implementierung eines Reglers mittels Model Predictive Control

8.1 Theoretische Grundlagen Model Predictive Control (MPC)

MPC ist eine fortgeschrittene Regelungsmethode, die unter anderem für die Regelung von nichtlinearen Systemen eingesetzt wird. Sie basiert darauf, zukünftige Prozessverläufe vorherzusagen und nutzt diese Informationen, um die besten Steueraktionen zu berechnen. Im Verlauf dieses Abschnittes wird das Grundprinzip und die Funktionsweise von MPC erläutert. Für eine detaillierte Auseinandersetzung mit MPC, einschließlich der zugrunde liegenden Theorien und Methoden, wird auf das Skript der Vorlesung „Regelungstechnik 2“ verwiesen [5].

8.1.1 Grundprinzip von MPC

MPC nutzt ein mathematisches Modell des Systems, um vorherzusagen, wie es sich in der Zukunft verhalten wird. Die Vorhersage geschieht über einen festgelegten Zeitraum, den sogenannten Vorhersagehorizont. Auf Grundlage dieser Prognosen ermittelt MPC die notwendigen Steueraktionen, um spezifische Ziele zu erreichen, die durch eine Kosten- oder Zielfunktion festgelegt sind. Dies führt zu einem Optimierungsproblem, welches gelöst werden muss, um die Zielfunktion unter Berücksichtigung aller relevanten Beschränkungen zu optimieren [19, S. 142 f.].

8.1.2 Funktionsweise von MPC

Im Folgenden wird die Funktionsweise von MPC detailliert erläutert, wobei der Fokus auf einer verständlichen Darstellung liegt, ohne allzu tief in komplexe mathematische Formulierungen einzusteigen. Diese Erläuterung orientiert sich konkret am Beispiel dieses Projekts, um die Funktionsweise greifbar und anschaulich darzustellen.

Bei der Anwendung von MPC sind der Vorhersagehorizont p und die Abtastzeit von zentraler Bedeutung. Sie legen fest, wie weit in die Zukunft der Algorithmus blicken kann und wie genau die Reaktionen des Systems erfasst werden. Zudem ist es notwendig, eine Zielfunktion, oft auch als Kostenfunktion bezeichnet, zu definieren. Diese Funktion zielt üblicherweise darauf ab, die Abweichung zwischen dem Sollwert $w[k]$ und dem Istwert $y[k]$ zu minimieren. Für diesen spezifischen Fall sieht die Funktion folgendermaßen aus:

$$f(\mathrm{w,y}) = \sum_{k=0}^{p} (w_k - y_k)^2 \tag{8.1}$$

Zu dieser Basisfunktion kommt nun ein zusätzlicher Term zur Glättung der Stellwerte hinzu, der abrupte Änderungen zwischen aufeinanderfolgenden Steueraktionen verhindern soll. Dieser Glättungsterm, gewichtet mit β, bestraft große Schwankungen in den Stellgrößen $u[k]$. Dies sorgt für eine sanftere Anwendung der Steuerbefehle und trägt somit zur Systemstabilität und Regelgenauigkeit bei. Der Faktor α bestimmt die Gewichtung der Differenz zwischen Soll- und Istwerte. Demnach kann die Kostenfunktion wie in Gleichung 8.2 formuliert werden.

$$f(\mathrm{w,y}) = \alpha \cdot \sum_{k=0}^{p} (w_k - y_k)^2 + \beta \sum_{k=0}^{p} (u_k - u_{k-1})^2 \tag{8.2}$$

In Bezug auf Abbildung 8.1 müssen nun die Stellgrößen $u[k]$, dargestellt in Blau, so angepasst werden, dass die Differenz zwischen dem Sollwert (rote Linie) und dem Istwert (gelbe Linie) so gering wie möglich ausfällt. Durch Anwendung des mathematischen Modells des Systems lässt sich eine Gleichung formulieren, die den Ausgang $y[k]$ zu verschiedenen Zeitpunkten in Abhängigkeit von $u[k]$ und den vorherigen Ausgangswerten vorhersagt. Diese Vorhersage wird in die Zielfunktion eingesetzt. Anschließend gilt es, das Optimierungsproblem zu lösen, mit dem Ziel, die Werte für $u[k]$ so zu bestimmen, dass die Zielfunktion minimiert wird und dadurch die Abweichung zwischen Soll- und Istwert so gering wie möglich ausfällt. Nachdem das Optimierungsproblem gelöst und die Stellwerte $u[0]$ bis $u[p]$ ermittelt wurden, wird in der Praxis lediglich der erste Stellwert $u[0]$ auf das System angewendet. Sobald die Abtastzeit verstrichen ist, wird der gesamte Prozess erneut durchgeführt, einschließlich der Lösung des Optimierungsproblems. Dieser iterative Prozess wiederholt sich fortlaufend, wobei der Vorhersagehorizont jeweils mit dem Zeitverlauf mitwandert. Die kontinuierliche Neuberechnung und Anpassung der Stellwerte ermöglicht es, die Steuerungsstrategie stets an die aktuelle Situation anzupassen und somit flexibel auf Änderungen zu reagieren [19, S. 145 ff.].

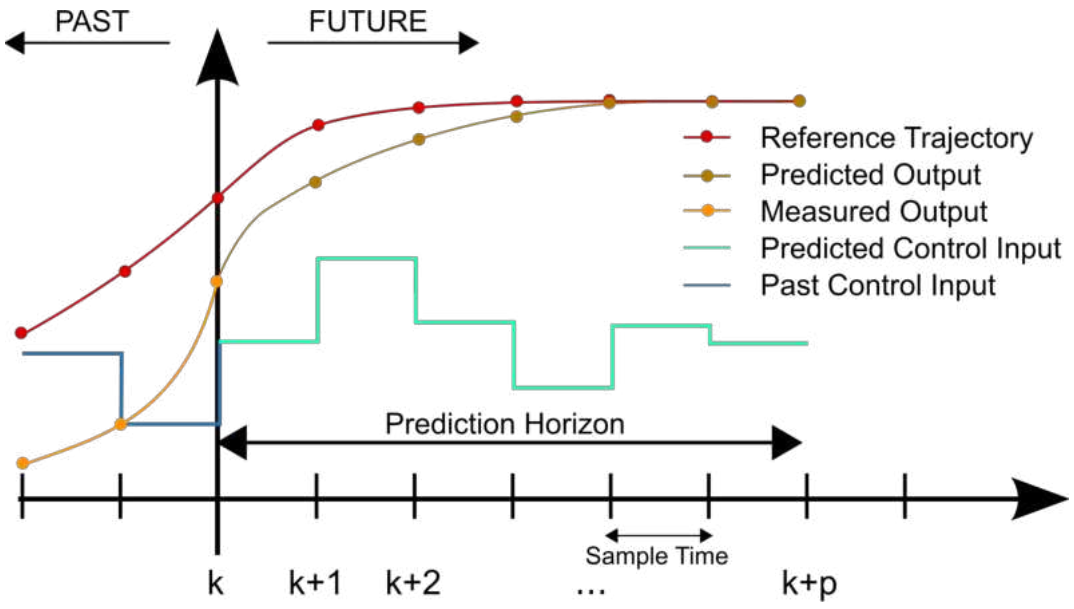


ABBILDUNG 8.1: MPC Beispiel [65]

8.1.3 Vor- und Nachteile von MPC

Model Predictive Control bietet eine vorausschauende Strategie zur Regelung komplexer Systeme, die sowohl Vorteile als auch bestimmte Herausforderungen mit sich bringt [36, S. 1048].

Vorteile:

- **Anpassungsfähigkeit von MPC:** MPC erlaubt eine flexible Definition von Zielfunktionen und die Einbeziehung spezifischer Beschränkungen, wodurch eine maßgeschneiderte Anpassung an jede Aufgabe möglich wird.
- **Vorausschauende Fähigkeit:** MPC verwendet ein mathematisches Modell, um vorherzusagen, was als Nächstes passiert. Das hilft dabei, schon im Voraus auf Veränderungen zu reagieren.
- **Flexibel bei Störungen:** Durch die kontinuierliche Neuberechnung und Anpassung der Steuerbefehle kann MPC effektiv auf unvorhergesehene Änderungen im System oder in der Umgebung reagieren.
- **Benutzerfreundliche Einstellungen:** MPC verwendet einfache, leicht verständliche Parameter, sodass auch Einsteiger diese nachvollziehen können.
- **Unterstützung für Multiple-Input Multiple-Output (MIMO)-Systeme:** MPC zeichnet sich durch seine Fähigkeit aus, Systeme mit mehreren Eingängen und Ausgängen gleichzeitig zu regeln.

Nachteile:

- **Hoher Rechenaufwand:** Die ständige Lösung des Optimierungsproblems, insbesondere bei komplexen Systemen mit langen Vorhersagehorizonten, erfordert signifikante Rechenkapazitäten, was die Anwendung in Echtzeitsystemen mit begrenzten Ressourcen einschränken kann.
- **Modellabhängigkeit:** Die Leistung von MPC hängt stark von der Genauigkeit des verwendeten mathematischen Modells ab.
- **Komplexität in der Implementierung:** Die Entwicklung eines MPC-Reglers ist nicht einfach. Dies liegt daran, dass man zunächst ein präzises Modell des Systems erstellen muss. Zusätzlich ist es notwendig, ein Optimierungsproblem zu formulieren und zu lösen.

Zusammenfassend lässt sich sagen, dass MPC eine leistungsstarke und flexible Regelungsmethode für komplexe Systeme bietet. Allerdings bringt die Implementierung auch einige Herausforderungen mit sich. Es muss für den entsprechenden Anwendungsfall abgewägt werden, ob die Vorteile die Schwierigkeiten übersteigen.

8.2 Implementierung von MPC

Die Umsetzung des MPC-Reglers wird in der Python-Klasse „MpcController“ realisiert. Für die erfolgreiche Implementierung werden die folgenden Schritte durchgeführt. Ein Ausschnitt des Codes für die Implementierung des MPC-Reglers ist im Anhang unter Unterabschnitt A.4.2 dargestellt.

1. **Modellbildung:**
   Zuerst wird ein mathematisches Modell des Wasserbeckens erstellt. Dieses Modell beschreibt, wie die Wassertemperatur auf verschiedene Einflüsse reagiert, beispielsweise auf die Heizleistung des Tauchsieders oder auf Änderungen der Umgebungstemperatur. Das Modell muss nicht perfekt sein, aber es sollte die grundlegenden dynamischen Eigenschaften des Systems erfassen. Für dieses Projekt wird das Verhalten des Wasserbeckens mittels einer Differenzialgleichung modelliert, die auf den PT1-Parametern basiert, welche in Abschnitt 5.4 bestimmt wurden. Zur numerischen Lösung dieser Differenzialgleichung wird die Euler-Methode eingesetzt, woraus sich folgende Formel für die Berechnung der Temperatur ergibt. In dieser Formel repräsentiert T die Temperatur, τ die Zeitkonstante, und K den Verstärkungsfaktor der PT1-Strecke:

$$T(t + \Delta t) = T(t) + \frac{\Delta t}{\tau}(K \cdot u(t) - T(t) + T_{\text{Umgebung}}) \tag{8.3}$$

   Zusätzlich muss noch die Totzeit des Systems von $T_d = 27$ s berücksichtigt werden. Dies wird umgesetzt, indem der Steuerbefehl, der zum aktuellen Zeitpunkt berechnet wird, erst nach einer Verzögerung von 27 Sekunden im System wirksam wird. Um dies in der numerischen Simulation zu berücksichtigen, wird der Steuerbefehl $u(t - T_d)$ anstelle von $u(t)$ in die Berechnung der Temperaturänderung eingesetzt.

2. **Abtastzeit definieren:**
   Die Definition der Abtastzeit ist ein wesentlicher Schritt in der Implementierung von MPC, da sie das Zeitintervall festlegt, in dem das System Messdaten erfasst und neue Steuerbefehle generiert. Die Abtastzeit beeinflusst sowohl die Schnelligkeit, mit der das System auf Veränderungen reagieren kann, als auch die Präzision, mit der Sollwerte verfolgt werden. Während eine zu lange Abtastzeit die Reaktionsfähigkeit des Systems verringert, kann eine zu kurze Abtastzeit die Rechenanforderungen unnötig erhöhen und den Regelalgorithmus sowie die Hardware belasten [19, S. 147 f.]. Für die Implementierung dieses Projekts wurde eine **Abtastzeit von 1 s** festgelegt. Diese Entscheidung basiert auf der Überlegung, dass diese Zeitspanne sowohl eine angemessene Schnelligkeit für die Regelungsprozesse bietet als auch sicherstellt, dass die Rechenressourcen effizient genutzt werden, ohne übermäßige Belastung.

3. **Vorhersagehorizont festlegen:**
   Der Vorhersagehorizont bestimmt den Zeitraum, über den das Systemverhalten vorhergesagt wird. Die Länge des Horizonts hängt von der Dynamik des Systems und den praktischen Anforderungen der Regelungsaufgabe ab [19, S. 148 f.]. Für die Temperaturregelung des Wasserbeckens muss ein Vorhersagehorizont gewählt werden, der ausreichend lang ist, um die Auswirkungen der Stellwerte auf die Wassertemperatur angemessen zu erfassen und kurz genug, um Rechenzeit zu sparen und somit auf Veränderungen zeitnah reagieren zu können. Auf Basis einer Abschätzung wird vorerst ein **Vorhersagehorizont von 80 Samples** festgelegt, was bei einer Abtastzeit von 1 s einer Periode von 80 Sekunden entspricht.

4. **Kontrollhorizont festlegen:**
   Der Kontrollhorizont bestimmt den Zeitraum, in dem aktive Steuermaßnahmen zur Beeinflussung des Systemverhaltens ergriffen werden. Während der

Systemzustand über den gesamten Vorhersagehorizont prognostiziert wird, erfolgen Anpassungen der Steuerbefehle nur innerhalb des Kontrollhorizonts. Nach Ablauf dieser Zeit bleiben die Stellwerte unverändert, um Rechenressourcen zu schonen [19, S. 148 f.]. Für die Implementierung wird anfangs ein **Kontrollhorizont von 40 Samples** angenommen, was bei einer Abtastzeit von 1 s einer Dauer von 40 Sekunden entspricht.

5. **Optimierungsproblem formulieren:**
   Die Implementierung des Optimierungsproblems wird durch die Methode „objective“ realisiert. Diese verwendet das mathematische Modell von Gleichung 8.3, um die Temperatur über einen festgelegten Vorhersagezeitraum zu berechnen. Die Hauptaufgabe dieser Methode ist es eine Reihe von Steuerbefehle mittels der Zielfunktion (beschrieben in Gleichung 8.2) zu beurteilen. Einfach ausgedrückt erhält die Methode Steuerbefehle, von u_0 bis u_p als Eingangsdaten. Basierend darauf prognostiziert das Modell den zukünftigen Temperaturverlauf. Anschließend wird die aus Gleichung 8.2 abgeleitete Kostenfunktion angewendet, um die Abweichungen zwischen dem vorhergesagten Temperaturverlauf und den Zieltemperaturen zu bewerten. Diese Funktion beinhaltet auch Strafterme für abrupte Änderungen in den Steuerbefehlen, um eine möglichst sanfte Regelung zu gewährleisten. Die resultierenden Kosten, die von der „objective“-Methode zurückgegeben werden, dienen somit als Maß für die Effektivität der Steuerbefehle. Durch die Methode können jetzt verschiedene Steuerbefehle auf einen einzigen Wert abgebildet werden, der zeigt, wie gut sie funktionieren. Das Ziel ist nun, das Optimierungsproblem zu lösen und somit die beste Kombination dieser Befehle zu finden, die für die effektivste Steuerung sorgt.

   Um die Kostenfunktion zu berechnen, ist die Festlegung der Gewichtungsfaktoren α und β erforderlich. Basierend auf vorläufigen Tests erscheinen die Werte $\alpha = 1$ und $\beta = 0,001$ als eine geeignete Ausgangsbasis.

6. **Lösung des Optimierungsproblems:**
   Um das Optimierungsproblem zu lösen, wird die Funktion „minimize“ aus der Bibliothek „scipy.optimize“ eingesetzt. Diese Bibliothek dient der Lösung von Optimierungsproblemen verschiedenster Art, einschließlich Minimierungsaufgaben. „minimize“ wählt ohne eine spezifisch vorgegebene Methode eine passende Standardmethode aus einer Auswahl, die auf Heuristiken basiert. Diese Heuristiken berücksichtigen Aspekte wie die Dimensionalität des Problems und das Vorhandensein von Grenzwerten, um die am besten geeignete Methode für das vorliegende Problem zu identifizieren [48]. Ein potenzieller Nachteil dieses allgemeinen Ansatzes ist die Unsicherheit bezüglich des genauen Lösungsweges, besonders die Frage, ob das gefundene Minimum lokal oder global ist. Da das Kapitel primär die Grundlage für spätere Implementierungen bilden soll, ist zunächst eine funktionierende Lösung ausreichend, auch wenn die genauen Details des Lösungsweges nicht bekannt sind.

   Für den Optimierungsprozess werden der Funktion „minimize“ die Zielfunktion „objective“, Grenzen für die Steuerbefehle ($0\,\% - 100\,\%$) und ein initialer Startwert übergeben. Ein Startwert von 0 für alle Steuerwerte gibt dem Algorithmus, den Ausgangspunkt für den Optimierungsprozess vor.

8.3 Testdurchlauf des MPC-Reglers

Um einen Eindruck über die Funktionsweise des entwickelten MPC-Reglers zu bekommen, wird der Testlauf, definiert in Abschnitt 6.1, durchlaufen. Die Auswahl der MPC-Parameter basierte auf intuitiven Abschätzungen. Für diesen Test wurden ein Vorhersagehorizont von 80 Samples und ein Kontrollhorizont von 40 Samples bei einer Abtastzeit von einer Sekunde festgelegt. Die Gewichtungsfaktoren für die Soll-Istwertabweichung und die Stellwertänderung wurden auf $\alpha = 1$ und $\beta = 0,001$ eingestellt, um ein ausgewogenes Verhältnis zwischen Regelgenauigkeit und Aktionsdynamik zu schaffen. Die Messaufzeichnungen des Testdurchlaufs sind in Abbildung 8.2 dargestellt.

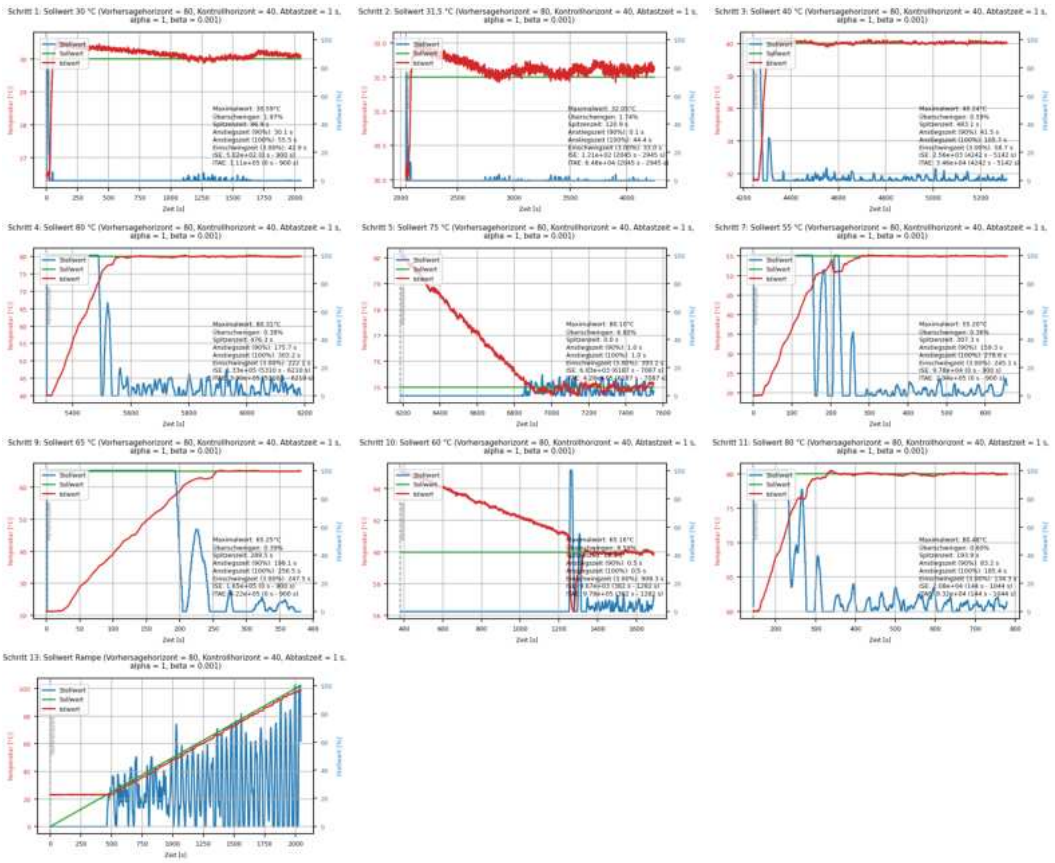


ABBILDUNG 8.2: Testdurchlauf MPC-Regler

Betrachtet man die Messergebnisse und vergleicht diese mit den Ergebnissen des PID-Reglers aus Abschnitt 7.4, kann man erkennen, dass die Ergebnisse ähnlich gut sind und keine der beiden Methoden sehr positiv oder negativ heraussticht. Ein auffälliger Punkt ist jedoch, dass bei der letzten Messung unter Verwendung einer Rampe eine konstante Abweichung von ungefähr 2 Kelvin vom Zielwert auftritt, was darauf hindeutet, dass der Regler in dieser spezifischen Situation nicht optimal funktioniert. Dieses Phänomen lässt sich darauf zurückführen, dass der Regler in seiner aktuellen Implementierung darauf ausgerichtet ist, die Abweichung zu minimieren, indem er annimmt, dass der Sollwert über den Vorhersagehorizont konstant bleibt, was hier nicht der Fall ist.

8.4 Fazit

Die Untersuchung hat gezeigt, dass MPC bei der Regelung des Wasserbeckens zufriedenstellende Leistungen erbringt, jedoch im Vergleich zu einem traditionellen PID-Regler keine signifikanten Vorteile für diese Art von einfachen Regelungsaufgaben bietet. Der wahre Wert des MPC wird bei der Handhabung komplexerer und nichtlinearer Systeme deutlich, vorausgesetzt, diese lassen sich durch mathematische Modelle ausreichend genau beschreiben. Hervorzuheben ist jedoch, dass die Effektivität von MPC stark von der Genauigkeit des Vorhersagemodells abhängt. Weicht das Modell von der Realität ab, können Probleme wie stationäre Fehler auftreten, welche die Regelgüte beeinträchtigen. In solchen Situationen kann der Einsatz eines PID-Reglers als ergänzende Maßnahme sehr nützlich sein. Die Kombination von MPC und PID nutzt die Vorteile der präzisen Vorhersagefähigkeit des MPC sowie die zuverlässige Korrektur des Restfehlers durch den PID-Regler.

Kapitel 9

Nachbilden eines bestehenden Reglers durch ein Neuronales Netz

In diesem Kapitel wird die Möglichkeit untersucht, einen bestehenden Regler durch ein neuronales Netz zu ersetzen. Das Ziel ist es, durch überwachtes Lernen ein Netzwerk so zu trainieren, dass es die Funktion eines bereits funktionierenden Reglers nachahmt, indem es dessen historische Daten nutzt. Dadurch soll das Neuronale Netz lernen, ähnliche Entscheidungen wie der bestehende Regler zu treffen, um die Temperatur unter verschiedenen Bedingungen zu regeln. Für die praktische Umsetzung dieses Kapitels wird der zuvor entworfene PID-Regler verwendet, dessen Dynamik das neuronale Netzwerk nachbilden soll.

Da das neuronale Netz mit Daten eines bereits bestehenden Reglers trainiert wird, liegt die Annahme nahe, dass seine Leistung nicht über die des PID-Reglers hinausgeht. Die Wahrscheinlichkeit, dass das Netz eigenständig neue Schlüsse zieht oder Strukturen erkennt, die einen Mehrwert gegenüber dem existierenden Regler bieten, ist gering. Daher ist kein unmittelbarer Vorteil gegenüber dem PID-Regler, der ersetzt werden soll, zu erwarten. Der Schwerpunkt dieses Kapitels liegt vielmehr darauf, ein Verständnis dafür zu entwickeln, inwiefern sich ein neuronales Netzwerk für Regelaufgaben eignet und welche Herausforderungen sowie Schwierigkeiten bei der Entwicklung und Implementierung auftreten können.

9.1 Wahl des Netzwerktyps

Bei der Entwicklung eines neuronalen Netzes gibt es verschiedene Netzwerkarchitekturen, die in Betracht gezogen werden können. Daher ist es wichtig, zunächst zu entscheiden, welche Art von Netzwerk eingesetzt werden soll.

Hierfür wird zunächst nochmals der PID-Regler betrachtet, den es zu ersetzen gilt. Wie im Blockdiagramm in Abbildung 9.1 gezeigt, nimmt dieser einen Soll- und Istwert auf und ermittelt daraus den Stellwert. Jetzt geht es darum, eine passende Netzwerkarchitektur zu finden, die mit ähnlichen Inputs ähnliche Ergebnisse liefert. Dafür werden nun unterschiedliche Ansätze untersucht.

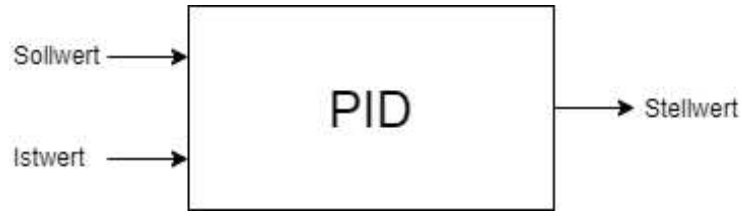


ABBILDUNG 9.1: PID-Regler Blockdiagramm

1. **Feedforward Neural Network (FFNN):**
   FFNNs sind die einfachste Art von neuronalen Netzen, bei denen Informationen nur in eine Richtung fließen. Diese Art von Netzwerk besitzt von sich aus kein Gedächtnis und bildet die aktuellen Eingänge auf die Ausgänge ab [55]. Das bedeutet, dass bei einem fertig trainierten FFNN dieselben Inputs immer zu demselben Output führen. Diese Art von Netzwerk kann für einfache Regler wie einem P-Regler funktionieren. Für die Integration eines I-Anteils, der vergangene Werte berücksichtigt, ist jedoch eine Form von Gedächtnis erforderlich. Eine denkbare Herangehensweise ist es, die Eingangsdaten um historische Informationen zu erweitern, wie es beispielhaft in Abbildung 9.2 dargestellt wird. Beim Verwenden von FFNNs muss einem bewusst sein, dass das Netz ausschließlich mit den Informationen arbeitet, die es über die Eingaben erhält. Eine eigenständige Gedächtnis- oder Speicherfunktion besitzt es nicht.

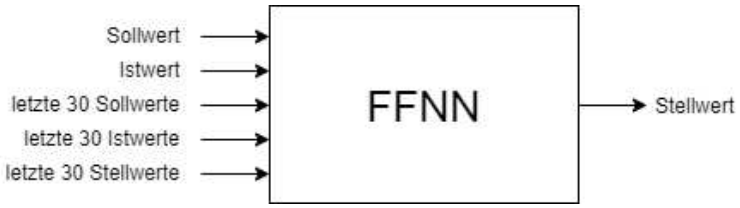


ABBILDUNG 9.2: Mögliches Blockdiagramm FFNN als Regler

2. **Recurrent Neural Network (RNN):**
   Im Gegensatz zu FFNNs besitzen RNNs wie der Name schon sagt rekurrente Verbindungen von Neuronen einer Schicht zu Neuronen derselben oder einer vorangegangenen Schicht. Abbildung 9.3 zeigt ein Beispiel mit verschiedenen Arten von Rückkopplungen. Diese rekursiven Schleifen erlauben es dem Netzwerk, einen Zustand über vergangene Eingaben zu bewahren, was sie ideal für Aufgaben macht, bei denen zeitliche Abhängigkeiten eine Rolle spielen, wie z.B. bei der Spracherkennung oder der Zeitreihenanalyse [71]. RNNs haben allerdings oft mit dem Verschwinden oder Explodieren von Gradienten zu kämpfen, was das Erlernen von langfristigen Mustern schwierig macht. Dieses Problem entsteht durch wiederholte Multiplikationen kleiner/großer Gewichte beim Backpropagation-Prozesses [55]. Daher erkennen RNNs zwar zeitliche Muster in Daten, aber oft nur über einen kurzen Zeitraum.

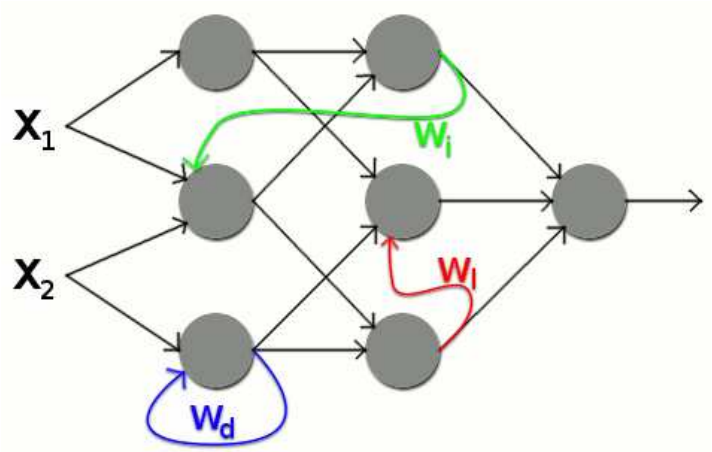


ABBILDUNG 9.3: Beispiel RNN [71]

3. **Long Short-Term Memory (LSTM):**
   LSTM-Netzwerke sind eine spezialisierte Form von RNNs, die entwickelt wurden, um die Herausforderungen des Verschwindens oder Explodierens von Gradienten zu überwinden. Ihre komplexe interne Struktur, welche in Abbildung 9.4 zu sehen ist, besteht aus verschiedenen „Gates“ (Input-, Forget- und Output-Gate), die steuern, wann Informationen gespeichert, aktualisiert oder gelöscht werden. Der „Cell State“ dient als Langzeitgedächtnis, während der „Hidden State“ kurzfristige Informationen verarbeitet und für die Ausgabe des Netzwerks genutzt wird. Die Aufgabe der Gates ist es, bei jedem Berechnungsschritt, relevante von irrelevanten Informationen zu unterscheiden und somit die „States“ entsprechend zu aktualisieren. Nach der Aktualisierung der „States“ und dem Durchlaufen der Datensequenz berechnet das Netzwerk dann seine Ausgaben [32]. Diese Struktur erlaubt es LSTMs, sowohl kurz- als auch langfristige Muster in Daten zu erkennen. Das macht sie ideal für sequenzielle Anwendungen wie Sprachverarbeitung, Texterzeugung und Zeitreihenanalysen.

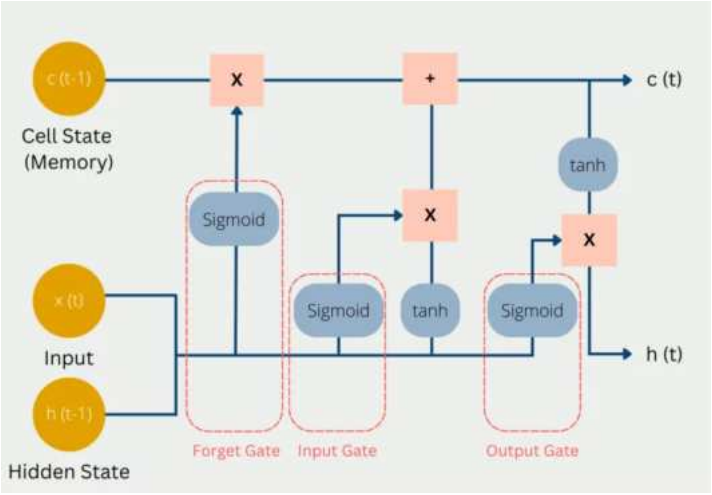


ABBILDUNG 9.4: Aufbau LSTM-Zelle [32]

Bei der Wahl des geeigneten Netzwerktyps für die Nachbildung eines PID-Reglers fällt die Entscheidung auf das LSTM-Netzwerk. LSTM-Netzwerke bieten im Vergleich zu FFNNs und klassischen RNNs den entscheidenden Vorteil, sowohl kurz- als auch langfristige zeitliche Abhängigkeiten in den Daten erfassen und modellieren zu können. Ein potenzieller Nachteil von LSTM-Netzwerken könnte ihre komplexere Struktur sein, die in der Regel längere Trainingszeiten und umfangreichere Datensätze für ein effektives Training erfordert. Dieser Aspekt muss allerdings während der praktischen Umsetzung erprobt werden.

9.2 Design und Training des LSTM-Reglers

Für die Umsetzung und Implementierung des Reglers dient die Arbeit „Automation with LSTM Network“ [1] als Grundlage, in der ein PID-Regler durch ein LSTM-Netzwerk ersetzt wurde. Dies bietet eine solide Basis für die Festlegung der Parameter, das Design und die Implementierungsstrategie. Die Datenaufbereitung und das Training des LSTMs wurde in dem Google Colab Notebook „LSTM_ersetzt_PID.ipynb“ durchgeführt. Google Colab ist ein kostenfreier, cloud-basierter Service, der die Ausführung von Python-Code in sogenannten Notebooks ohne eine lokale Softwareinstallation ermöglicht. Diese Notebooks sind besonders im Bereich des maschinellen Lernens beliebt, da sie eine übersichtliche Strukturierung von Code neben dazugehöriger Dokumentation ermöglichen. Codeabschnitte können zudem einzeln ausgeführt werden, was eine flexible und interaktive Arbeitsweise fördert. Die in den Notebooks trainierten Modelle werden anschließend exportiert und in dem PyCharm-Projekt importiert, wo sie mit dem restlichen Projektcode verwendet werden können. Im Anhang in Unterabschnitt A.4.3 finden sich Codeausschnitte, die den Prozess des Trainings des LSTM-Modells sowie dessen spätere Implementierung als Regler veranschaulichen.

9.2.1 Entwurf der Grundlegenden Netzwerkarchitektur

Bevor mit der Generierung der Trainingsdaten begonnen wird, ist es notwendig, die grundlegende Architektur des zu trainierenden LSTM-Netzwerks festzulegen. Das Netzwerk erhält Sequenzen von Eingangsdaten, die eine bestimmte Anzahl historischer Informationen umfassen. Zum Beispiel könnte dies eine Sequenz der letzten 15 Sollwerte und Regelabweichungen sein, jeweils im Abstand von zwei Sekunden. Auf der Grundlage dieser Daten soll das LSTM-Netzwerk den nächsten Stellwert vorhersagen. Das zugrundeliegende Design lässt sich in vereinfachter Form, so wie in Abbildung 9.5, darstellen.

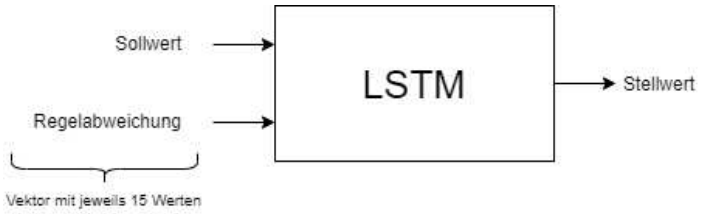


ABBILDUNG 9.5: Grundstruktur LSTM

9.2.2 Generierung der Trainingsdaten

Für das Training des LSTM-Netzwerks wurden, wie in Abbildung 9.6 dargestellt, Trainingsdaten über einen Zeitraum von etwa 90 000 Sekunden generiert. Hierbei wurden zufällige Sollwerte im Bereich von 25 °C bis 100 °C für zufällige Zeitintervalle zwischen 500 und 7000 Sekunden vorgegeben. Aufgezeichnet wurde jeweils die resultierenden Stellwerte, die der PID-Regler lieferte. Diese Daten wurden zunächst in einer simulierten Umgebung unter Verwendung des PT1-Euler-Modells erzeugt, nicht am realen System. Die Simulation bietet den Vorteil, dass sie Herausforderungen wie Messungenauigkeiten, die im realen Einsatz auftreten würden, vermeidet. Zudem ermöglicht sie eine beschleunigte Datengenerierung verglichen mit dem zeitaufwendigen Prozess am realen System.

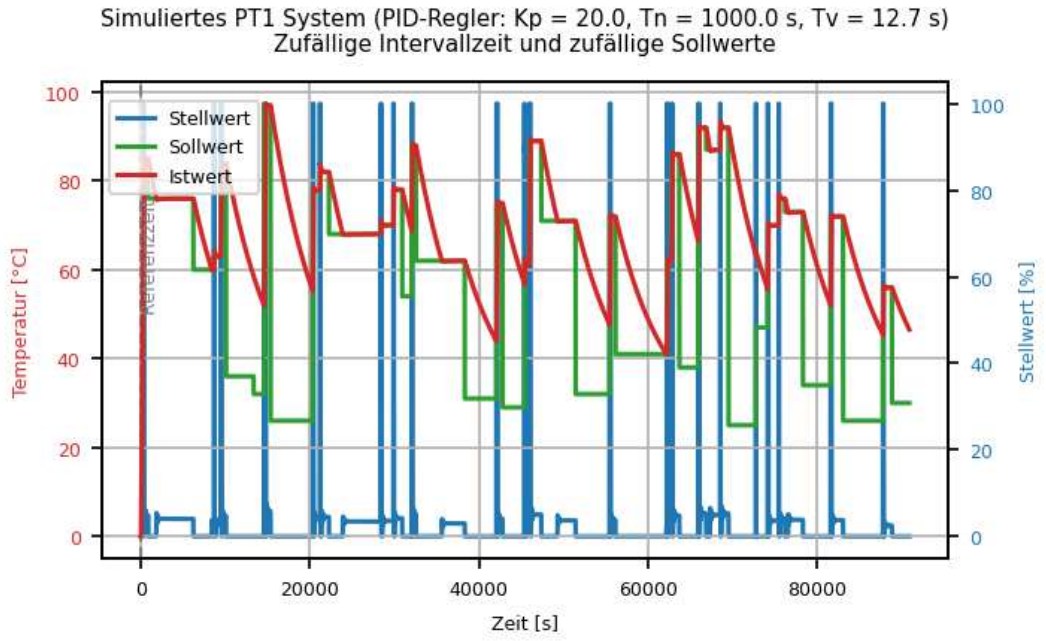


ABBILDUNG 9.6: Simulation des PID-Reglers

Datenvorverarbeitung

Um mit den generierten Daten das LSTM zu trainieren, ist eine Vorverarbeitung dieser Daten nötig.

Feature Engineering:
Zunächst gilt es, die Eingaben für das Netzwerk festzulegen. Neben den bestehenden Daten wurde die Regelabweichung (Error) als zusätzliches Feature eingeführt. Um die Bedeutung der einzelnen Features für die Zielvariable zu evaluieren, kam die Funktion „SelectKBest“ aus der „sklearn“-Bibliothek zum Einsatz. Diese Funktion führt einen statistischen Test durch, um die Stärke der Beziehung zwischen jedem Feature und der Zielvariable zu messen. Der grundlegende Gedanke dabei ist, dass Features mit einer stärkeren Beziehung zur Zielvariable vermutlich von größerer Relevanz für das Modell sind. Indem man den Fokus auf diese relevanten Features legt, kann nicht nur die Genauigkeit des Modells gesteigert, sondern auch Overfitting minimiert und die Trainingszeit verkürzt werden [13]. Die Anwendung dieser Funktion auf die Trainingsdaten ergab den in Abbildung 9.7 dargestellten Plot, welcher die Relevanz der Features im Verhältnis zueinander anzeigt. Auf Grundlage

dieser Ergebnisse wurde entschieden, den Istwert als Feature zu eliminieren, um Redundanzen möglichst zu vermeiden.

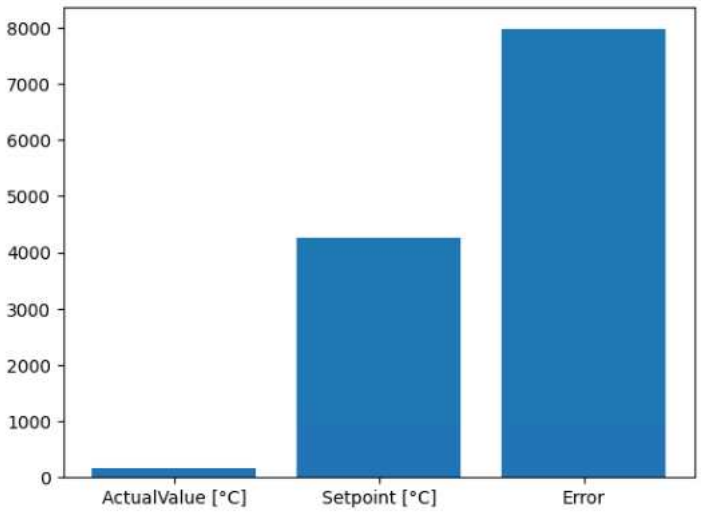


ABBILDUNG 9.7: Ergebnis „SelectKBest“

Daten skalieren:
Für eine verbesserte Modellgenauigkeit ist das Skalieren der Daten oft unerlässlich, da viele Algorithmen im maschinellen Lernen auf unterschiedliche Wertebereiche der Features sensibel reagieren. Durch das Skalieren werden alle Merkmale auf einen einheitlichen Maßstab gebracht, um zu verhindern, dass bestimmte Merkmale aufgrund ihrer Größenordnung einen überproportionalen Einfluss haben. Es gibt verschiedene Skalierungsmethoden wie zum Beispiel der „MinMaxScaler“, welcher die Daten eines Features in einen Bereich zwischen 0 und 1 transformiert, indem er die Werte in Relation zu deren minimalen und maximalen Werten skaliert. Der verwendete „MaxAbsScaler“ hingegen passt die Daten so an, dass sie zwischen -1 und 1 liegen, wobei das ursprüngliche Vorzeichen der Daten erhalten bleibt [12].

Sequenzen erstellen:
Für das LSTM-Netzwerk ist das Umwandeln der Trainingsdaten in Sequenzen ein wesentlicher Schritt. LSTMs benötigen sequenzielle Daten, um zeitliche Muster effektiv zu lernen. Dabei werden die Daten in mehrere Abschnitte aufgeteilt, wobei jeder Abschnitt eine Folge von aufeinanderfolgenden Datenpunkten enthält. Die Wahl der Sequenzlänge ist dabei wichtig, da sie bestimmt, wie weit das Netzwerk in die Vergangenheit blicken kann, um seine Vorhersagen zu treffen. Die Aufteilung der Datenpunkte wurde so gestaltet, dass das LSTM-Netzwerk eine Sequenz der letzten 15 Messungen als Eingabe erhält, mit einem Intervall von 2 Sekunden zwischen den einzelnen Werten. Diese Struktur ermöglicht es dem Netzwerk, zeitliche Muster zu erkennen, indem es Informationen aus den vorangegangenen 30 Sekunden berücksichtigt.

Aufteilung Trainings-, Test- und Evaluierungsdaten:
Für eine effektive Bewertung des LSTM-Netzwerks werden die Daten in drei Teile aufgeteilt (Training-, Test- und Evaluierungsdaten). Die Trainingsdaten, welche 64 % des Gesamtdatensatzes ausmachen, sind für das Anlernen der Modellparameter vorgesehen. Die Testdaten (16 %), ermöglichen während des Trainingsprozesses,

nach jeder Epoche, eine vorläufige Beurteilung des Modells anhand bisher unbekannter Daten. Die Evaluierungsdaten (20 %) werden am Ende verwendet, um die endgültige Leistungsfähigkeit des Modells zu beurteilen.

9.2.3 Erstellen und Trainieren des Modells

Das Erstellen des LSTM-Modells beginnt mit dem Design der Netzwerkarchitektur. Entscheidend hierbei ist die Festlegung der Anzahl an LSTM-Schichten sowie der Neuronen je Schicht. Außerdem sind die Auswahl eines geeigneten Optimierers, die Integration von Dropout-Layern und die Bestimmung der Trainingsparameter von Bedeutung. Eine detaillierte Diskussion dieser Konfigurationsmöglichkeiten und ihrer Auswirkungen auf die Modellleistung wird in Unterabschnitt 9.2.5 vertieft behandelt. Nach Abschluss dieser Vorbereitungen beginnt das Training des Modells, welches anschließend zusammen mit den verwendeten Skalierern gespeichert wird, um eine spätere Anwendung oder Weiterentwicklung zu erleichtern. Die in Abbildung 9.8 dargestellten Fortschrittsbalken zeigen während des Trainings den „loss“, welcher den Fehler zwischen den Vorhersagen des Modells und den tatsächlichen Trainingsdaten angibt, und den „val_loss“, der den Fehler auf dem Validierungsdatensatz misst und somit die Modellleistung auf unbekannten Daten bewertet. Der am Ende erzeugte Plot veranschaulicht den Trainingserfolg, wobei das Ziel darin besteht, dass mit fortschreitenden Epochen sowohl der Trainings- als auch der Validierungsfehler kontinuierlich abnehmen.

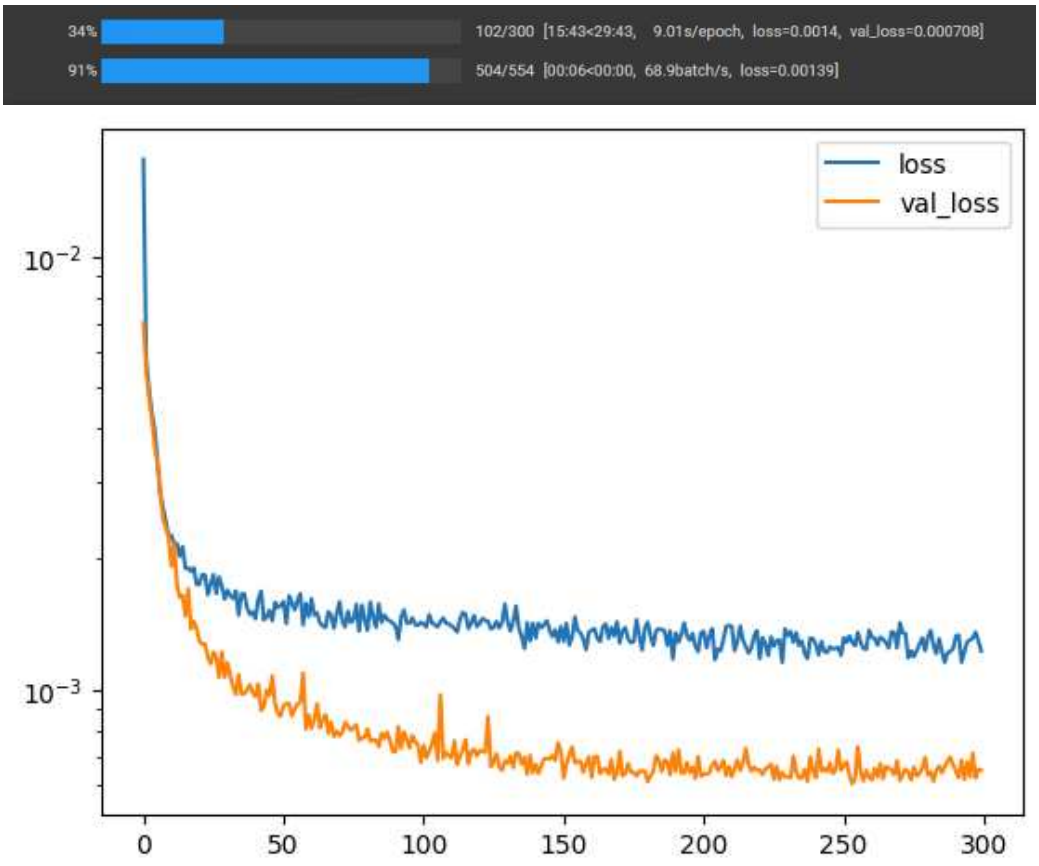


ABBILDUNG 9.8: Darstellung Trainingsfortschritt

9.2.4 Evaluieren des Modells

Die Evaluierung umfasst nicht nur die ursprünglich separierten 20 % der Daten, sondern auch einen extra Datensatz aus einer neuen 100 000-Sekunden-Simulation, um die Bewertung aussagekräftiger zu gestalten. Mit dem trainierten Modell werden die Stellwerte für diese Evaluierungsdaten prognostiziert. Das Diagramm, welches in Abbildung 9.9 zu sehen ist, stellt die vom LSTM-Modell vorhergesagten Stellwerte den Werten des PID-Reglers gegenüber, mit dem Ziel, eine hohe Übereinstimmung zu erreichen. Darüber hinaus wurde der mittlere quadratische Fehler (MSE) ermittelt, um einen direkten Vergleich der Modellgenauigkeit mit früheren Modellen zu ermöglichen. Zusätzlich kann zur Modellbewertung, der Trainings- und Validierungsfehler über die Epochen als Indikatoren herangezogen werden. Ein sinkender Trainingsfehler bei gleichbleibendem oder steigendem Validierungsfehler weist zum Beispiel auf eine Überanpassung an die Trainingsdaten hin. Wenn hingegen beide Fehlerarten nicht abnehmen, deutet dies auf Probleme beim Lernprozess hin, die eine Überprüfung der Netzwerkarchitektur, der Lernparameter oder der Trainingsdaten erfordern.

```
Quadratischer mittlerer Fehler (MSE) zwischen PID und LSTM: 7.161861850139367
```

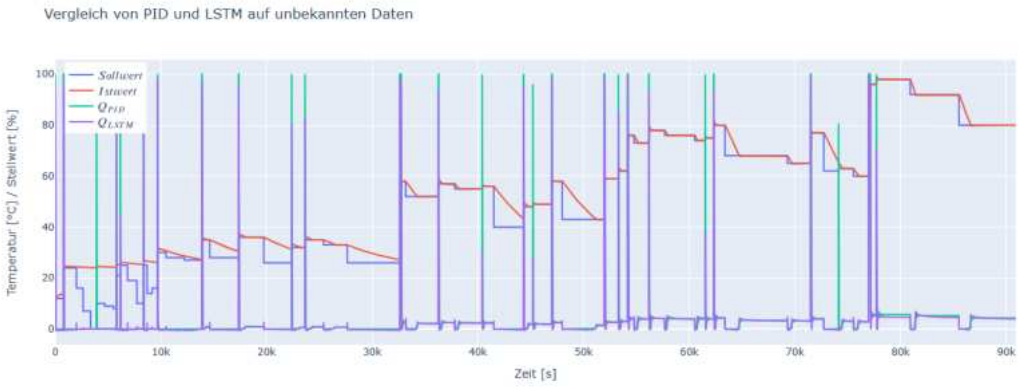


ABBILDUNG 9.9: Vergleich PID vs. LSTM auf unbekannten Daten

9.2.5 Optimierung und Feinabstimmung des LSTM-Modells

Nach der initialen Implementierung des LSTM-Modells folgt eine Phase der Optimierung und Feinabstimmung, die entscheidend für das Erreichen einer akzeptablen Modellleistung ist. Dieser Prozess wird iterativ gestaltet, wobei durch gezielte Anpassung einzelner Parameter das Modell wiederholt trainiert, evaluiert und analysiert wird. Obwohl automatisierte Verfahren zur Parameteroptimierung existieren, basiert dieser Ansatz vorwiegend auf manuellen Experimenten. Dieses Vorgehen zielt darauf ab, ein intuitives Gefühl für die Auswirkungen unterschiedlicher Einstellungen auf die Leistung des Modells zu erlangen. Da das wiederholte Trainieren des Netzwerks mit allen Trainingsdaten zeitaufwendig ist, kann eine effektive Strategie sein, nur einen Teil der Daten zu nutzen oder die Anzahl der Trainingsdurchläufe (Epochen) zu verringern. Diese Vorgehensweise ermöglicht es, vorläufig zu beurteilen, ob und wie sich der Fehler durch gemachte Anpassungen verändert. Bei zufriedenstellenden vorläufigen Ergebnissen kann das Modell anschließend unter vollständigen Bedingungen für eine umfassende Bewertung trainiert werden.

Netzwerkarchitektur und Parameter

Die Auswahl passender Netzwerkarchitekturen und Parameter kann herausfordernd und zeitaufwendig sein und verlangt sowohl Erfahrung als auch ein gutes Verständnis für das Modell und die Daten. Die folgende Tabelle fasst die wichtigsten Anpassungen zusammen, die zur Leistungssteigerung des Modells vorgenommen wurden, einschließlich der final genutzten Einstellungen. Der Weg zu diesen Entscheidungen, geprägt von zahlreichen Versuchen, Trainingsdurchläufen und Beobachtungen, wird hier nicht im Detail beschrieben. Jedoch enthält die Tabelle zu einigen Parametern kurze Notizen oder Besonderheiten, die im Verlauf aufgefallen sind.

Parameter	**Beschreibung**	**Gewählte Einstellung**	**Anmerkung**
Fenster	Bestimmt die Anzahl der vorangegangenen Werte, die als Eingabe in das LSTM-Netzwerk fließen. In Kombination mit dem Parameter „step_interval“ legt dies fest, wie weit das LSTM in die Vergangenheit schauen kann. Ein hoher Wert ermöglicht es dem Modell, Muster über längere Zeiträume hinweg besser zu erkennen, kann allerdings das Training verlangsamen und das Risiko von Überanpassung erhöhen. Ein niedriger Wert hingegen beschleunigt zwar das Training, könnte jedoch die Fähigkeit des Modells einschränken, langfristige Muster zu identifizieren.	15	Größeres Fenster hat nur geringfügige Verbesserungen gebracht während Trainingszeit stark anstieg.
Zeitintervall	Definiert das Zeitintervall zwischen den einzelnen Werten in einer Sequenz. Dies beeinflusst, wie detailliert das LSTM, zeitliche Veränderungen innerhalb des festgelegten Fensters wahrnimmt. Ein längeres Intervall kann die Komplexität reduzieren und das Training beschleunigen, riskiert jedoch den Verlust wichtiger Informationen zwischen den Punkten.	2 Sekunden	

Parameter	**Beschreibung**	**Gewählte Einstellung**	**Anmerkung**
Wahl der Input Features	Definiert welche Datenpunkte oder Features als Input für das LSTM-Netzwerk verwendet werden. Eine sorgfältige Auswahl ist entscheidend, da sie direkt die Qualität der Modellvorhersagen beeinflusst. Zu viele oder irrelevante Features können das Modell verwirren oder zu Overfitting führen, während bei zu wenigen Features, möglicherweise wichtige Informationen ausgelassen werden.	Sollwert, Regelabweichung	Der Istwert ist in Sollwert und Regelabweichung enthalten, wodurch seine separate Berücksichtigung eine unnötige Redundanz darstellen würde.
Wahl des Skalierers	Bestimmt die Methode, mit der die Eingabedaten normiert oder standardisiert werden, um sie auf einen einheitlichen Wertebereich zu bringen.	MaxAbs -Scaler [14]	Die Wahl des „MaxAbsScalers“ basierte auf einer intuitiven Entscheidung, da dieser das ursprüngliche Vorzeichen der Daten (bei Regelabweichung relevant) beibehält. Andere Skalierer zeigten in der Performance geringfügig schlechtere Ergebnisse.
Shuffle	Steuert, ob die Reihenfolge der Datensequenzen vor der Aufteilung in Trainings- und Testsets zufällig gemischt wird. Das Beibehalten der ursprünglichen Reihenfolge kann für Zeitreihendaten entscheidend sein, um die zeitliche Abfolge und die damit verbundenen Muster nicht zu stören.	False	Das Ergebnis ist ähnlich, in Foren wird aber empfohlen die Daten bei LSTMs nicht zu mischen.
Anzahl LSTM-Hidden-Layer	Gibt die Tiefe des Netzwerks an, also wie viele aufeinanderfolgende verdeckte Schichten verwendet werden. Mehr Schichten können dem Netzwerk helfen, komplexere Muster zu erkennen, erfordern jedoch mehr Rechenleistung und können das Risiko von Overfitting erhöhen.	2	Siehe „Ergänzung zur Bestimmung der Anzahl der Layer und Neuronen“
Anzahl der Neuronen pro Schicht	Definiert, wie viele Neuronen im jeweiligen LSTM-Layer enthalten sind. Eine höhere Anzahl kann die Modellkomplexität und die Fähigkeit zur Mustererkennung steigern, bringt aber auch die Gefahr von Overfitting mit sich und erhöht den Rechenaufwand.	Hidden-Layer 1: 5 Neuronen, Hidden-Layer 2: 3 Neuronen	Siehe „Ergänzung zur Bestimmung der Anzahl der Layer und Neuronen“

Parameter	Beschreibung	Gewählte Einstellung	Anmerkung
Dropout-Schichten und Dropout-Rate	Dropout ist eine Technik zur Vermeidung von Overfitting, indem zufällig ausgewählte Neuronen während des Trainings ignoriert werden. Die Rate bestimmt den Anteil der Neuronen, die in jeder Epoche ausgelassen werden. Die Platzierung der Dropout-Schichten innerhalb des Netzwerks spielt eine wesentliche Rolle für deren Wirksamkeit [56].	Jeweils nach den beiden Hidden-Layern ein Dropout-Layer mit einer Rate von 0,1	
Aktivierungs -funktionen	Diese bestimmen, wie Neuronen ihre Signale verarbeiten und weiterleiten. Sie sind zum Beispiel entscheidend für die Fähigkeit des Netzwerks Nichtlinearitäten in den Daten zu modellieren.	LSTM-Layer: Standard, Output-Layer: Linear	
Optimierer	Wählt den Algorithmus, der zur Anpassung der Gewichte basierend auf den Trainingsdaten verwendet wird. Ein gut gewählter Optimierer verbessert die Effizienz des Trainings und die Genauigkeit des Modells.	Adam	Der Adam-Optimierer ist wegen seiner bewährten Effektivität eine weit verbreitete Wahl.
Lernrate	Steuert, wie stark die Gewichte nach jedem „Batch“ während des Trainings angepasst werden. Ist sie zu hoch, kann das Modell über das Ziel hinausschießen und nicht konvergieren. Ist sie zu niedrig, kann das Training unnötig lange dauern oder in einem lokalen Minimum stecken bleiben.	0.0001	
Batch-Größe	Sie gibt an, wie viele Trainingsbeispiele gleichzeitig verarbeitet werden, bevor die Gewichte aktualisiert werden. Die Batch-Größe beeinflusst die Geschwindigkeit und Genauigkeit des Lernens, da sie bestimmt, wie oft die Gewichte angepasst werden und wie viele Daten bei jeder Anpassung berücksichtigt werden.	50	
Anzahl Epochen	Epochen bestimmen, wie oft das Modell den gesamten Trainingsdatensatz durchläuft. Mehr Epochen bieten zwar zusätzliche Chancen zum Lernen, können jedoch auch zu Overfitting führen. (Ein Anzeichen für Overfitting ist es, wenn der Validierungsfehler nach einer gewissen Anzahl von Epochen wieder zu steigen beginnt.)	300	Ab 200 Epochen ist keine signifikante Reduzierung des Fehlers mehr zu beobachten.

TABELLE 9.1: Zusammenfassung der wichtigsten Parameter

Ergänzung zur Bestimmung der Anzahl der Layer und Neuronen
Die Bestimmung der optimalen Architektur eines Neuronalen Netzes, einschließlich der Anzahl an Layern und Neuronen, ist ein wichtiger Schritt, der wesentlich zur Leistung des Modells beiträgt. Es gibt keine allgemeingültige Formel, die die perfekte Konfiguration für jedes Problem liefert, jedoch existieren Richtlinien und Heuristiken, die als Ausgangspunkt verwendet werden können. Von dort aus kann durch systematisches Experimentieren eine passende Struktur gefunden werden.

In einem neuronalen Netzwerk entspricht die Anzahl der Neuronen in der Eingangsschicht üblicherweise der Anzahl der Merkmale im Datensatz. Für regressive Modelle, wie in diesem Projekt, wird in der Ausgangsschicht normalerweise ein Neuron verwendet [22]. Die Anzahl der Hidden-Layer sind ausschlaggebend für die Fähigkeit des Netzwerks, komplexe Muster zu erkennen. Einfache Probleme erfordern weniger Neuronen und Schichten, während komplexere Aufgaben von tieferen Architekturen profitieren können. Netzwerke ohne Hidden-Layer sind nur für linear separierbare Probleme geeignet, während ein Netzwerk mit einem Hidden-Layer nichtlineare Probleme lösen kann. Zwei versteckte Schichten reichen aus, um nahezu jede Funktion zu modellieren, weshalb meist ein bis zwei dieser Schichten für praktische Anwendungen genügen. Während tiefere Netzwerke mit vielen Parametern zwar komplexe Probleme effektiver lösen können, führen sie gleichzeitig zu längeren Trainingszeiten und einem erhöhten Bedarf an umfangreichen Datensätzen, um dem Risiko einer Überanpassung entgegenzuwirken [73]. Eine Faustformel für die Anzahl der Neuronen in den Hidden-Layern besagt, dass diese zwischen der Anzahl der Eingabe- und Ausgabeneuronen liegen sollte, aber nicht mehr als das Doppelte der Eingabeneuronen betragen darf [22]. Diese Regel dient lediglich als grobe Orientierung und bezieht sich primär auf FFNNs. Je nach spezifischem Einsatzgebiet und insbesondere bei der Verwendung von LSTMs kann es jedoch durchaus angebracht sein, von dieser Faustformel abzuweichen.

9.3 Testen und Auswerten des Modells

9.3.1 Auswertung anhand der Evaluierungsdaten

Das LSTM wurde für die Parameterfindung nach jedem Training bereits oberflächlich ausgewertet. Anschließend sollen die Ergebnisse des final trainierten Modells etwas genauer analysiert und bewertet werden.

Zunächst wird das Gesamtbild der Evaluierungsdaten betrachtet, welches in Abbildung 9.10 dargestellt ist. Es zeigt verschiedene Sollwerte sowie die resultierenden Stell- und Istwerte einer Simulation eines PID-Reglers, wobei das PT1-Euler-Modell zur Simulation des Umgebungsverhaltens eingesetzt wurde. Diese Daten dienen zur Bewertung der Vorhersagen des LSTM-Netzwerks, visualisiert durch eine lila Linie. Im Idealfall sollte diese lila Linie die Stellwerte des PID-Reglers, angezeigt durch eine hellblaue Linie, möglichst genau wiedergeben.

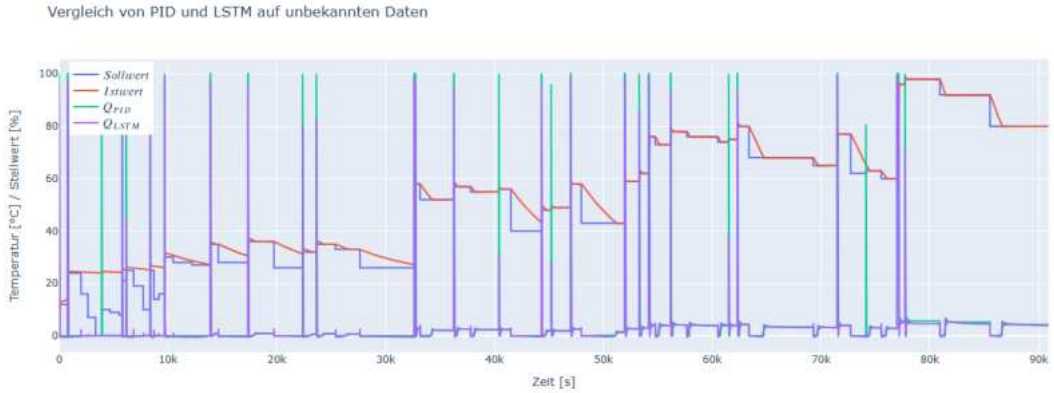


ABBILDUNG 9.10: LSTM-Ausgabe Auf Evaluierungsdaten

Auf den ersten Blick scheint das LSTM den PID-Regler recht gut zu approximieren. Ein genauerer Blick auf das vergrößerte Diagramm (Abbildung 9.11), zeigt jedoch, dass es in einigen Situationen noch zu leichten Abweichungen kommt.

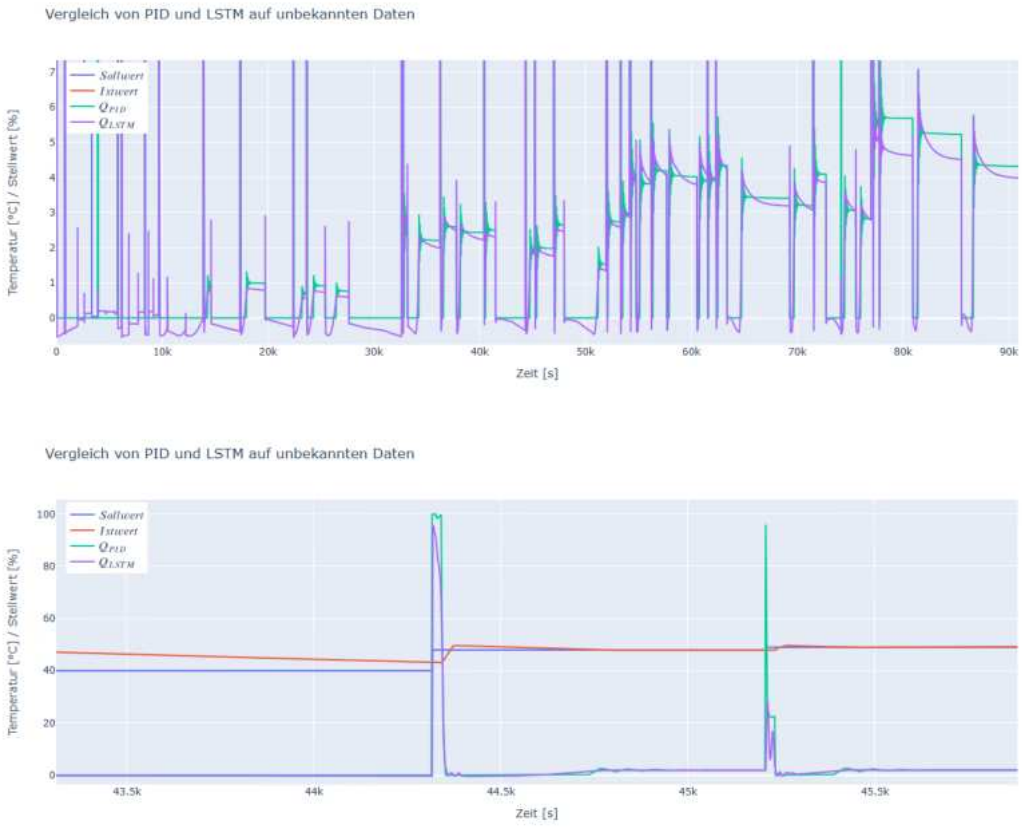


ABBILDUNG 9.11: LSTM-Ausgabe Auf Evaluierungsdaten gezoomt

9.3.2 Testen des LSTM-Reglers im Betrieb (Simulationsumgebung)

Trotz geringer Abweichungen auf den Validierungsdaten wird das Modell für den Einsatz im tatsächlichen Betrieb exportiert, sodass es innerhalb der Simulationsumgebung im PyCharm-Projekt genutzt werden kann. Dort wird der LSTM-Regler

verwendet, um verschiedene, willkürlich gewählte Sollwerte zu regeln. Einige Ausschnitte dieser Simulation, sind in Abbildung 9.12 zu sehen. Um einen Vergleichswert zu haben wurde zusätzlich mit aufgezeichnet, welchen Stellwert der PID-Regler, den es zu reproduzieren gilt, ausgegeben hätte.

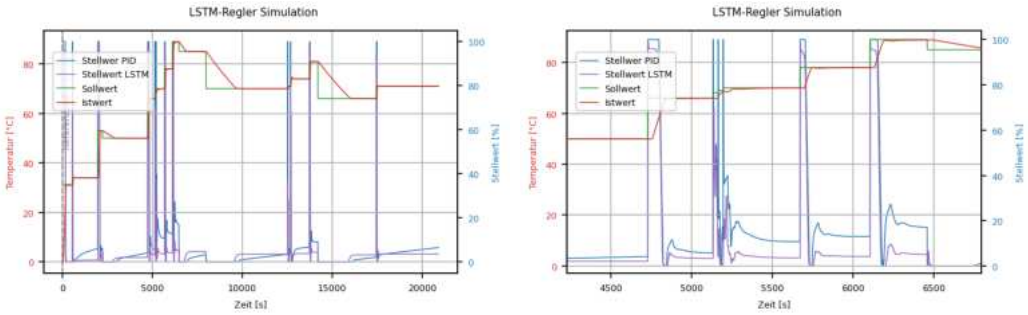


ABBILDUNG 9.12: Simulation LSTM-Regler

Obwohl es Situationen gibt, in denen der LSTM-Regler nicht ganz so wie der PID-Regler reagiert, werden die Sollwerte dennoch relativ präzise erreicht. Es scheint, dass die Abweichungen hauptsächlich in Szenarien auftreten, die in den Trainingsdaten nicht ausreichend vertreten sind. Es ist zu erkennen, dass das LSTM Schwierigkeiten hat, das I-Verhalten des PID-Reglers exakt nachzubilden.

9.3.3 Testen des LSTM-Reglers im Betrieb (Reales System)

Es soll nun überprüft werden, ob der Regler, der ursprünglich mit Simulationsdaten trainiert wurde, auch im realen System effektiv funktioniert. Dafür wurden bestimmte Sollwerte angefahren, wie in Abbildung 9.13 dargestellt.

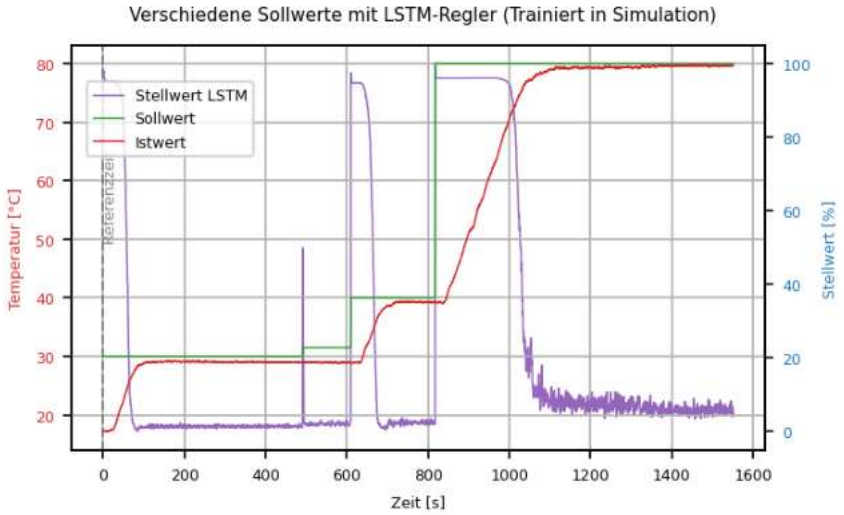


ABBILDUNG 9.13: Test des LSTM-Regler (Trainiert mit Simulationsdaten) am realen System

Die Auswertung der Ergebnisse zeigt, dass der Regler in der Lage ist, sich den Sollwerten anzunähern, jedoch sind deutlich stationäre Fehler zu erkennen, und die Sollwerte werden nicht präzise erreicht. Es fehlt das I-Verhalten, das erforderlich wäre, um diese Fehler über die Zeit auszugleichen.

9.4 Untersuchung der Eignung von realen Betriebsdaten für das LSTM-Training

In den vorherigen Kapiteln wurde die Generierung von Trainingsdaten mittels Simulation durchgeführt, um effizient eine große Menge an Daten in kurzer Zeit zu sammeln. Jetzt wird die Möglichkeit untersucht, reale Daten eines PID-Reglers aus dem tatsächlichen Betrieb für das Training zu nutzen. Zu diesem Zweck wurden über einen Zeitraum von etwa 14 000 Sekunden verschiedene Sollwerte vorgegeben und die Reaktionen des Systems aufgezeichnet, wie in Abbildung 9.14 dargestellt. Mit diesen realen Daten wurde das Modell anschließend trainiert. Bei den Ergebnissen auf unabhängigen Validierungsdaten, dargestellt in Abbildung 9.15, fällt auf, dass das Modell das Rauschen in den Daten etwas filtert. Insgesamt ist das Ergebnis jedoch recht gut und übertrifft die Leistung des mit Simulationsdaten trainierten Reglers.

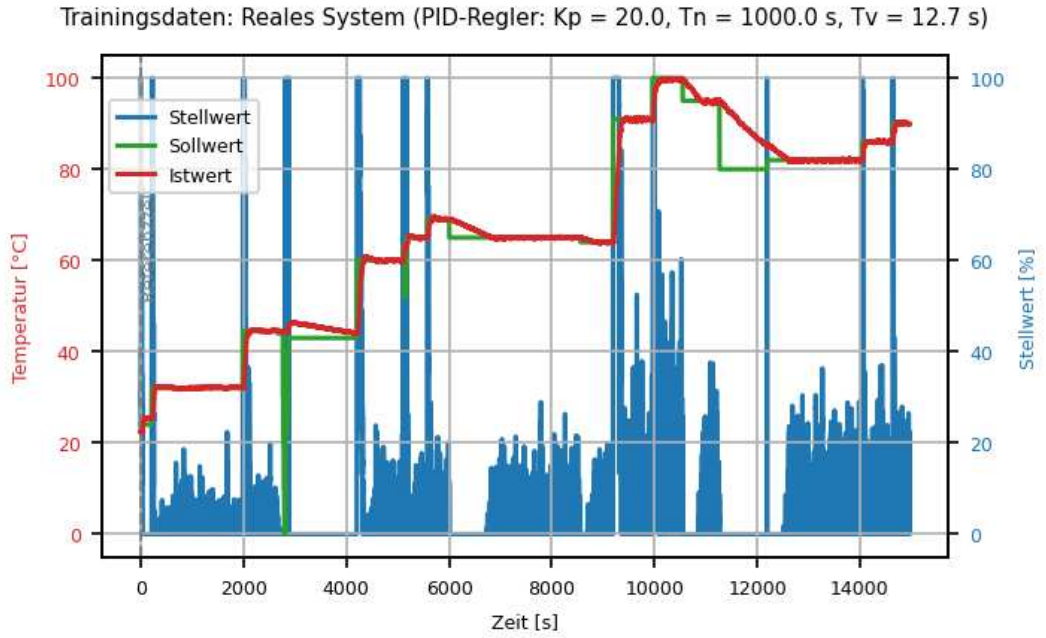


ABBILDUNG 9.14: Trainingsdaten reales System

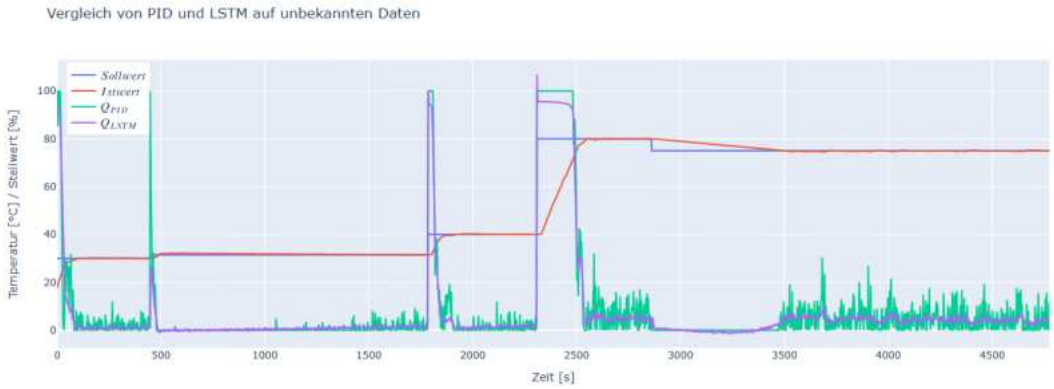


ABBILDUNG 9.15: Ausgabe auf Validierungsdaten (Auf realen Daten trainiertes LSTM)

Zum Testen des Reglers wird nun der in Abschnitt 6.1 beschriebene Testdurchlauf am realen System durchgeführt. Die daraus resultierenden Ergebnisse sind in Abbildung 9.16 zu sehen.

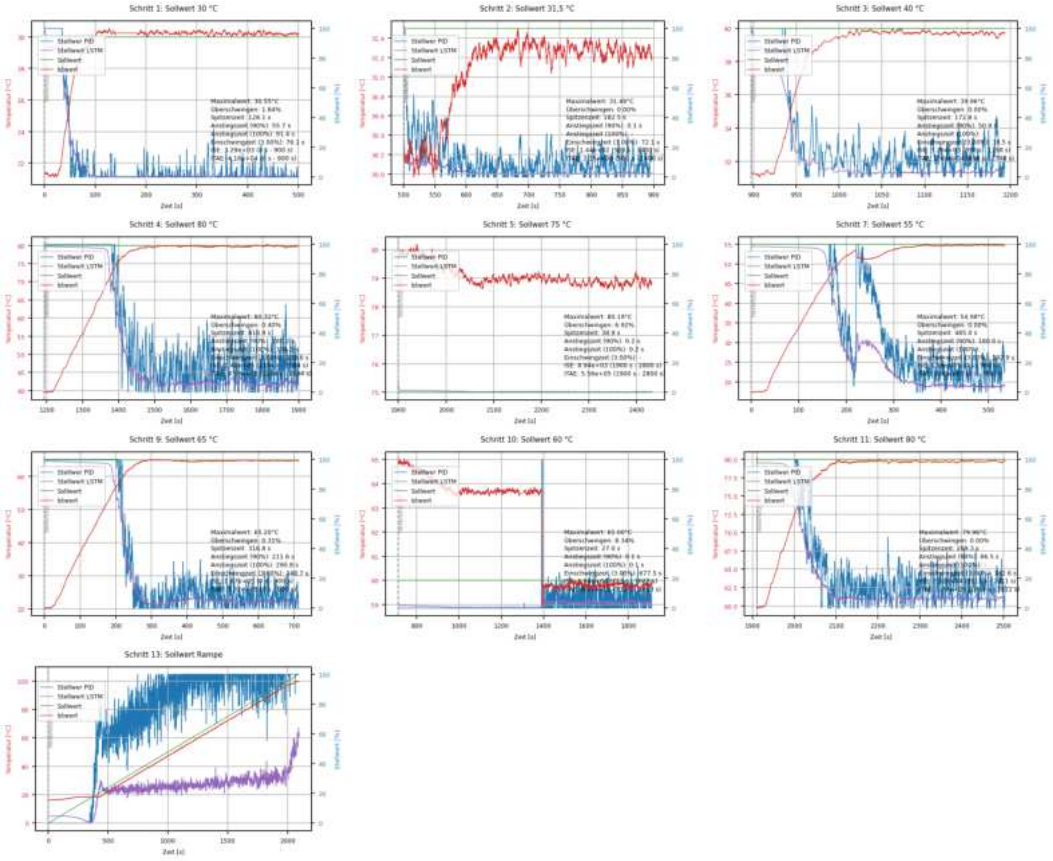


ABBILDUNG 9.16: Testdurchlauf LSTM-Regler (Trainiert mit realen Daten)

Bei der Auswertung des Testlaufs zeigt sich, dass es stationäre Fehler gibt und der Sollwert nicht immer genau getroffen wird, weil das I-Verhalten fehlt. Der stationäre Fehler ist allerdings geringer als bei dem Regler, der mit Simulationsdaten trainiert wurde. Dies liegt wahrscheinlich daran, dass dieser Regler anhand echter Systemdaten gelernt hat und somit die Unterschiede zwischen dem Simulationsmodell und der Wirklichkeit keinen Einfluss haben. Besonders auffällig ist, dass bei Abkühlphasen wie zum Beispiel in Schritt 5 der Stellwert nicht auf 0 gesetzt wird, was dazu führt, dass der Sollwert nicht erreicht wird.

9.5 Bewertung der Ergebnisse und Verbesserungspotenzial

Dieser Abschnitt analysiert und bewertet die Ergebnisse der Experimente mit dem LSTM. Hierbei geht es darum, die Probleme, welche aufgetreten sind, zu verstehen und zu überlegen, wie man sie potenziell lösen könnte. Die Umsetzung dieser Lösungsansätze wird im Umfang dieser Arbeit jedoch nicht weiter verfolgt, da der Fokus darauf lag, ein grundlegendes Verständnis für LSTM-Regler zu entwickeln. Im Folgenden sollen die wesentlichen Erkenntnisse aus den Experimenten bewertet werden.

- **Regler lernt Systemdynamik:**
  Betrachtet man die Ergebnisse des mit Simulations- und Echtdaten trainierten Reglers, ergibt sich die Erkenntnis, dass nicht das allgemeine PID-Verhalten des Reglers erlernt wurde, sondern das Regelverhalten des kompletten Systems, welches auch die Systemdynamik der PT1-Strecke mit einschließt. Je nach Anwendungsfall kann dies sowohl Vor- als auch Nachteile mit sich bringen. Ein wesentlicher Vorteil ist, dass der Regler dadurch ein besseres Verständnis für die Systemdynamik entwickeln kann, anstatt sich nur auf PID-Mechanismen zu beschränken. Dies kann jedoch auch zum Nachteil werden, falls das Systemverhalten stark von dem Zustand abweicht, auf den der Regler trainiert wurde. Die Auswahl der Trainingsdaten spielen hierbei eine entscheidende Rolle, um das Verhalten des LSTMs zu beeinflussen. Indem man zum Beispiel auf allgemeinere und umfassendere Trainingsdaten setzt, die kein PT1-System nachbilden, könnte man den Regler dazu anleiten, sich mehr wie ein universeller PID-Regler zu verhalten.

- **Abweichungen des Stellwertes zwischen PID-Regler und LSTM:**
  Obwohl das LSTM den PID-Regler meist gut imitiert, gibt es manchmal Unterschiede in den Stellwerten. Diese Abweichungen könnten durch fehlende Informationen in den Trainingsdaten entstehen. Eine vollständige Eliminierung dieser Abweichungen ist aufgrund des Bestrebens, eine Überanpassung zu vermeiden, nicht realisierbar. Dennoch lässt sich die Genauigkeit verbessern, indem man ein breiteres Spektrum an Trainingsdaten nutzt. Demnach wären zusätzliche Trainingsdaten erforderlich, die nicht nur quantitativ, sondern auch qualitativ umfassender sind und ein weites Spektrum an Sollwerten über verschiedene Temperaturbereiche abdecken. Die Herausforderung hierbei besteht darin, eine Methode zu finden, mit der man schnell und effektiv diese umfassenden Daten erheben kann.

- **Stationäre Fehler:**
  Auffällig in den Ergebnissen ist der stationäre Fehler, also die dauerhafte Differenz zwischen Soll- und Istwert bei bestimmten Temperaturbereichen. In den vorhandenen Trainingsdaten erreicht der Regler den Sollwert meist zügig. Szenarien in denen der Istwert langfristig unter oder über dem Ziel liegt, sind darin nicht vorhanden. Folglich fehlt dem Regler die Erfahrung, mit solchen Situationen umzugehen. In der Praxis können diese Zustände allerdings auftreten, etwa durch Unterschiede im Systemverhalten oder wenn das LSTM den PID-Regler nicht perfekt nachbildet. Ein Lösungsansatz hierfür wäre, die Trainingsdaten gezielt um solche Situationen zu erweitern, in denen der Regler bisher nicht optimal arbeitet. Dadurch könnte der Regler quasi ein Integral-Verhalten erlernen, was ihm hilft, diese stationären Fehler zu minimieren.

- **Stellwert wird beim Abkühlvorgang nicht 0:**
  Ein weiteres Problem, das vor allem beim mit realen Daten trainierten LSTM auftritt, ist, dass der Stellwert beim Abkühlvorgang nicht immer auf 0 reduziert wird und es somit zu bleibenden Regelabweichungen kam. Dies lässt sich wahrscheinlich auch durch eine Erweiterung des Trainingsdatensatzes beheben. Aufgrund der langsamen Abkühlprozesse des Systems waren diese Szenarien in den vorhandenen Trainingsdaten unterrepräsentiert, was zu diesem Mangel im Lernprozess geführt hat.

Zusammenfassend lässt sich sagen, dass die Auswahl und Generierung der Trainingsdaten eine entscheidende Rolle dafür spielen, was das Netzwerk lernt und

wie gut es funktioniert. Viele der Probleme, die während der Experimente auftraten, könnten durch eine gezielte Auswahl der Trainingsdaten gemildert oder vermieden werden. Besonders beim Training eines LSTMs mit realen Daten ist eine durchdachte Strategie erforderlich, um eine breite Palette von Szenarien in den Trainingsdaten zu erfassen. Zudem bietet das Experimentieren mit Netzwerkparametern, Sequenzlängen und Eingabefeatures weitere Chancen, die Leistung des Netzes zu verbessern. Es muss allerdings hervorgehoben werden, dass dieser Prozess des Ausprobierens und Testens sehr zeitaufwendig und komplex sein kann.

9.6 Fazit

Das Ziel dieses Kapitels war es zu erkunden, ob ein bestehender Regler, sei es ein PID-Regler oder ein MPC-Regler, durch ein LSTM ersetzt werden kann. Die durchgeführten Experimente haben bestätigt, dass dies grundsätzlich möglich ist, allerdings handelt es sich um keinen einfachen Prozess. Die Auswahl der richtigen Trainingsdaten, eines geeigneten Netzwerkdesigns und angemessener Parameter erweist sich als anspruchsvoll. Zudem ist es oft schwer nachzuvollziehen, warum der Regler bestimmte Entscheidungen trifft. Auch wenn der Regler generell funktioniert, gibt es immer ein Risiko für Situationen, die er nicht kennt und auf die er dann unerwartet reagiert.

Nun stellt sich die Frage, warum man so viel Aufwand betreiben sollte, um ein LSTM zu entwickeln, welches letztendlich keine signifikanten Vorteile gegenüber dem bestehenden Regler bietet und zudem potenzielle Fehlerquellen und Nachteile mit sich bringt. Ein interessanter Gedanke für die Zukunft der Regelungstechnik könnte die Entwicklung eines universellen LSTM-Netzwerks sein, das in der Lage ist, unterschiedliche Systeme effektiv zu regeln. Hierfür könnte man historische Daten von verschiedenen Reglern, die jeweils auf ein bestimmtes System zugeschnitten sind, verwenden um das LSTM zu trainieren. Das Idealziel wäre dann, dass dieses LSTM in der Lage ist, abhängig von der jeweiligen Systemdynamik, angepasste Regelstrategien zu erlernen und anzuwenden. Somit hätte man einen universellen Regler, der eine Vielzahl von Systemen regeln kann, indem er sich dynamisch an das Systemverhalten anpasst. Jedoch bringt dieser universelle Ansatz, der verschiedene Regler vereint, auch neue Herausforderungen mit sich, auf die in dieser Arbeit nicht näher eingegangen wird.

Kapitel 10

Erweiterung der Model Predictive Control durch maschinelles Lernen

In Kapitel 8 wurde die Model Predictive Control (MPC) eingeführt, die primär darauf abzielt, das zukünftige Verhalten eines Systems vorherzusagen und darauf basierend optimale Steuerbefehle zu bestimmen. Da es in der Praxis oftmals eine Herausforderung ist, ein mathematisches Modell zu finden, welches das Systemverhalten hinreichend genau beschreibt, beschäftigt sich dieses Kapitel mit der Untersuchung, ob ein traditionelles MPC-Modell durch ein datengetriebenes neuronales Netzwerk ersetzt werden kann. Ziel ist es, ein Netzwerk zu entwickeln, das selbstständig aus historischen Daten Muster und Zusammenhänge lernt und diese nutzt, um das zukünftige Systemverhalten vorherzusagen. Dieses Modell könnte dann als Vorhersagemodell innerhalb einer MPC Anwendung eingesetzt werden. Das Kapitel zielt darauf ab zu prüfen, ob dieser Ansatz praktikabel ist und wie gut ein solches Modell im Vergleich zu traditionellen MPC-Methoden abschneidet. Für diesen Zweck erfolgt ein Vergleich mit dem MPC, beschrieben in Kapitel 8, bei dem zur Vorhersage ein PT1-Modell mit Totzeit eingesetzt wurde.

In der Simulink-Umgebung von Matlab wird ein Tool namens „Neural Network Predictive Controller“ angeboten, das theoretisch die Umsetzung der soeben genannten Konzepte ermöglichen soll. Es nutzt neuronale Netzwerke, um Vorhersagen über das Systemverhalten zu treffen und darauf basierend Steuerbefehle abzuleiten. Allerdings funktioniert das Tool weitgehend als eine Blackbox, was heißt, dass man nicht genau sehen kann, wie es funktioniert. Man kann nur anhand der Beschreibungen und Parameter einige Vermutungen aufstellen [34]. In diesem Kapitel wird das Simulink-Tool zwar nicht direkt verwendet. Es dient jedoch als Inspirationsquelle für die Entwicklung eines Reglers mit ähnlicher Funktionalität. Die auf der Webseite beschriebenen Parameter liefern wichtige Anhaltspunkte, die genutzt werden können, um maschinelles Lernen effektiv in MPC zu integrieren.

10.1 Wahl des Netzwerktyps

Zunächst muss geklärt werden, welche Art neuronaler Netzwerke für die Modellierung des Systems gut geeignet ist. **LSTM-Netzwerke** sind gut darin, lange und komplexe Abhängigkeiten in Daten zu erkennen. Sie sind also gut für Anwendungen geeignet, wo die Daten viele zeitliche Zusammenhänge enthalten. Allerdings benötigen sie relativ viel Rechenleistung, was bei der Implementierung in Echtzeitsystemen oder auf Hardware mit begrenzten Ressourcen ein Nachteil sein kann. Außerdem kann ihre Komplexität zu langen Trainingszeiten und einem erhöhten

Bedarf an Trainingsdaten führen, was die Entwicklungszeiträume signifikant verlängert. **Feedforward-Neural-Networks (FFNNs)** zeichnen sich hingegen durch ihre geringere Komplexität und dementsprechend schnelleren Trainings- und Ausführungszeiten aus. Das macht sie attraktiv für den Einsatz in ressourcenbeschränkten Umgebungen oder Anwendungen, bei denen schnelle Entscheidungen erforderlich sind. Allerdings besitzen FFNNs von Natur aus keine interne Mechanik, um zeitliche Sequenzen direkt zu modellieren. Um dennoch zeitlich relevante Informationen für die Vorhersage nutzbar zu machen, ist ein entsprechendes Feature-Engineering notwendig.

Unter Berücksichtigung der genannten Faktoren, insbesondere des geringeren Rechenbedarfs und der einfacheren Struktur, fällt die Wahl für die Modellvorhersage auf das FFNN. Da MPC aufgrund des zu lösenden Optimierungsproblems ohnehin schon rechenintensiv und zeitkritisch ist, stellen FFNNs im Vergleich zu LSTMs die praktikablere Option dar.

10.2 Design des FFNN-Netzwerks

Das Ziel des FFNNs ist es, basierend auf historischen Istwerten und Stellgrößen des Systems, zukünftige Istwerte vorherzusagen. Um dies zu erreichen, muss zunächst das Grundgerüst für das neuronale Netzwerk entworfen werden.

Abtastzeit:
Ein wichtiger Parameter beim Einsatz von MPC ist die Wahl der Abtastzeit. Eine zu lange Abtastzeit kann dazu führen, dass wesentliche Informationen über die Systemdynamik verloren gehen, während eine zu kurze Abtastzeit den Rechenaufwand erheblich erhöht. Auch für das Training des Vorhersagemodells spielt die Abtastzeit eine entscheidende Rolle. Das verwendete neuronale Netz erhält als Input, Vektoren, die aus aufeinanderfolgenden historischen Daten bestehen, und lernt auf dieser Grundlage, den nächsten Istwert vorherzusagen. Aus dieser Überlegung ergeben sich zwei zusätzliche Parameter welche von Bedeutung sind. Die zeitlichen Abstände zwischen zwei aufeinanderfolgenden historischen Datenpunkten sowie der spezifische Zeitschritt, für welchen der zukünftige Istwert prognostiziert werden soll. Diese Zeiträume können flexibel festgelegt werden und müssen nicht zwangsläufig gleich sein. Es ist jedoch von entscheidender Wichtigkeit, dass die einmal gewählten Zeitabstände während des Trainingsprozesses über die verschiedenen Datenpunkte hinweg und ebenso in der späteren Anwendungsphase konsistent beibehalten werden. Um die Komplexität des Systems zu minimieren und den Prozess der Datensammlung für das Training zu vereinfachen, wird die Zeitspanne zwischen den einzelnen historischen Daten sowie der Zeitschritt für die zukünftigen Vorhersagen des Netzwerks einheitlich an die Abtastzeit des MPCs angeglichen.

Es ist nun erforderlich, eine geeignete Abtastzeit für den Betrieb des MPCs und des dazugehörigen Vorhersagemodells festzulegen. Dabei ist zu berücksichtigen, dass der Abkühlprozess des Systems deutlich langsamer verläuft als der Heizprozess. Eine Abtastzeit von einer Sekunde oder kürzer würde während des Abkühlens wahrscheinlich keine signifikanten Änderungen erfassen. Eine **Abtastzeit von 5 Sekunden** erscheint daher angemessen, da sie ausreichend Zeit bietet, um Veränderungen

am System zu erkennen, ohne dass die Dynamik des Systems dadurch zu stark beeinflusst wird.

Ausgabe des Netzwerkes:
Für die Vorhersage zukünftiger Zustände mit einem neuronalen Netz in Model Predictive Control gibt es hauptsächlich zwei verschiedene Ansätze. Der iterative Ansatz prognostiziert den nächsten Zustand basierend auf aktuellen und vergangenen Daten und wiederholt diesen Vorgang für jeden Schritt im Vorhersagehorizont. Dies ist zwar flexibel, kann aber rechenintensiv sein, da das Netzwerk mehrfach ausgeführt wird. Außerdem besteht das Risiko der Fehlerfortpflanzung. Mit jeder Iteration kann sich ein anfänglicher Modellierungsfehler weiter ausbreiten und die Vorhersagegenauigkeit über den Zeitraum hinweg verschlechtern. Der direkte Ansatz hingegen trainiert das Netzwerk darauf, alle zukünftigen Zustände innerhalb des Horizonts in einem Durchlauf vorherzusagen, was effizienter in der Rechenzeit ist, jedoch ein komplexeres Design und sorgfältiges Training erfordert. Für dieses Projekt fällt die Wahl auf den iterativen Ansatz, hauptsächlich aufgrund seiner geringeren Komplexität, die wiederum den Bedarf an Trainingsdaten reduziert. Das Ziel ist nämlich, mit einer geringen Menge an Trainingsdaten bereits effektive Vorhersagen zu erzielen. Daher erscheint der iterative Ansatz als die passendere Methode. Folglich ist das FFNN so gestaltet, dass es lediglich einen Ausgang hat, der den nächsten zukünftigen Istwert liefert.

Eingangsdaten:
Die Eingangsdaten für das Netzwerk setzen sich zusammen aus historischen Istwerten und den dazugehörigen Stellgrößen des Systems. Diese Kombination ermöglicht es dem Netzwerk, Muster und Abhängigkeiten zu erkennen, die entscheidend für die Vorhersage des nächsten Istwertes sind. Eine wichtige Entscheidung betrifft die Menge der vergangenen Messwerte, die als Eingabe dienen. Es ist entscheidend, dass das Netzwerk genügend Informationen aus der Vergangenheit erhält, um eine sinnvolle Vorhersage treffen zu können, ohne jedoch von irrelevanten Daten überflutet zu werden. Da das System eine Totzeit von ungefähr 27 Sekunden aufweist, muss der betrachtete Zeitraum der Stellwerte in der Vergangenheit mindestens diese Dauer umfassen. Mit einer festgelegten Abtastzeit von fünf Sekunden zwischen den Messwerten erscheint eine Auswahl der letzten 5 Istwerte und der letzten 10 Stellwerte sinnvoll. Dadurch kann das Netzwerk in Bezug auf die Stellwerte 50 Sekunden und bei den Istwerten 25 Sekunden zurückblicken. Ein kürzerer Rückblick bei den Istwerten erscheint ausreichend. Diese Zeitspanne gewährleistet, dass das Netzwerk genug relevante Informationen erhält, um die Systemdynamik zu verstehen und genaue Vorhersagen des nächsten Istwertes zu treffen.

Basierend auf den zuvor festgelegten Konfigurationen lässt sich das Grundgerüst des FFNNs für die Modellvorhersage, wie in Abbildung 10.1 dargestellt, visualisieren. Demnach empfängt das Modell einen Eingabevektor, der 17 Datenpunkte umfasst. Diese setzen sich aus dem aktuellen Istwert, dem aktuellen Stellwert sowie 15 historischen Datenpunkten zusammen.

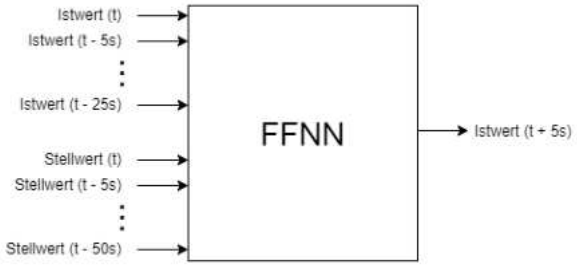


ABBILDUNG 10.1: Grundstruktur FFNN zur Modellvorhersage

10.3 Festlegung der Netzwerkparameter

Für die Festlegung der wichtigsten Parameter des Netzwerks, wie die Anzahl der Schichten, Anzahl der Neuronen pro Schicht, Aktivierungsfunktionen und Dropout-Layer, kam eine Bibliothek für Bayesianische Optimierung zum Einsatz. Dies ist ein Ansatz, der darauf abzielt, die bestmöglichen Hyperparameter für ein maschinelles Lernmodell zu finden. Im Gegensatz zu anderen Methoden, die Hyperparameter zufällig oder durch systematisches Ausprobieren aller Kombinationen testen, nutzt die Bayesianische Optimierung frühere Ergebnisse, um intelligente Vermutungen darüber anzustellen, welche Hyperparameter-Kombinationen voraussichtlich die besten Leistungen erbringen. Dieser Prozess beruht auf dem Prinzip der Wahrscheinlichkeit und zielt darauf ab, den vorgegebenen Suchraum effizient zu erkunden, indem Bereiche mit hohem Potenzial bevorzugt getestet werden. So ermöglicht es eine schnellere und effizientere Optimierung im Vergleich zu anderen Methoden [3, 21]. Die Implementierung und Durchführung der Bayesianischen Optimierung ist im Google Colab Notebook mit dem Titel „FFNN_vorhersageTemp_Simulation.ipynb“ realisiert worden. Ein spezifischer Codeausschnitt dazu kann im Anhang in Unterabschnitt A.4.4 eingesehen werden. Für diese Analyse wurden 500 000 zufällige Datenpunkte generiert, die auf dem zuvor entwickelten Simulationsmodell des PT1-Systems basieren. Um ein breites Spektrum an Hyperparametern zu evaluieren, wurden 100 verschiedene Konfigurationen getestet, jeweils in zwei Durchgängen über 10 Epochen hinweg. Die Aufteilung der Datensätze erfolgte mit 80 % für das Training und 20 % für die Validierung. Als Leistungsindikator für das Modell diente der mittlere quadratische Fehler (MSE).

Die Parameter für das endgültige Design des FFNNs sowie die Trainingsparameter wurden basierend auf den Erkenntnissen der Bayesianischen Optimierung, den Erfahrungen aus dem Kapitel über das LSTM-Modell und weiteren Tests festgelegt. Die wichtigsten Parameter sind in Tabelle 10.1 zusammengefasst.

Parameter	Gewählte Einstellung	Anmerkung
Wahl des Skalierers	MinMaxScaler [15]	
Shuffle	True	Im Gegensatz zum LSTM ist es hier kein Problem, wenn die Trainings- und Testdaten beim Aufteilen gemischt werden.
Anzahl Hidden-Layer	1	Ausgewählt aufgrund der Ergebnisse der Bayesianischen Optimierung
Anzahl der Neuronen pro Schicht	2	Ausgewählt aufgrund der Ergebnisse der Bayesianischen Optimierung
Dropout-Schichten und Dropout-Rate	Keine Dropout-Schicht	Ausgewählt aufgrund der Ergebnisse der Bayesianischen Optimierung
Aktivierungsfunktionen	Hidden-Layer: relu, Output-Layer: Linear	Ausgewählt aufgrund der Ergebnisse der Bayesianischen Optimierung
Optimierer	Adam	Der Adam-Optimierer ist wegen seiner bewährten Effektivität eine weit verbreitete Wahl.
Lernrate	0,008	Ausgewählt aufgrund der Ergebnisse der Bayesianischen Optimierung
Batch-Größe	32	Standardwert
Anzahl der Epochen	300	

TABELLE 10.1: Zusammenfassung der wichtigsten Trainingsparameter für das FFNN

10.4 Trainieren und Bewerten des Neuronalen Netzwerks

10.4.1 Generieren der Trainingsdaten

Damit das neuronale Netzwerk die Temperatur des Wasserbeckens möglichst genau vorhersagen kann, ist eine Sammlung von vielseitigen Trainingsdaten unerlässlich. Diese Daten sollten ein möglichst breites Spektrum an Situationen abdecken, in denen das System arbeitet. Deshalb ist es wichtig, ein Verfahren zu entwickeln, das unterschiedliche Stellwerte über den gesamten Temperaturbereich testet und die Reaktionen des Systems aufzeichnet. Eine Herausforderung ergibt sich daraus, dass der Abkühlvorgang des Systems sehr langsam ist. Im Gegensatz zum Aufheizprozess, bei dem das Erreichen eines beliebigen Sollwertes oft nur wenige Minuten in Anspruch nimmt, kann die Abkühlung des Systems mehrere Stunden dauern. Deshalb ist eine sorgfältige Auswahl der Zielwerte im Datengenerierungsprozess entscheidend, um möglichst schnell eine breite Palette an Daten zu erfassen. Das Python-Skript „DnMpcGenerateTrainingData.py“ steuert diesen Prozess der Datenerstellung. Der darin implementierte Algorithmus setzt zufällige Stellwerte über zufällig gewählte Intervallzeiten an und lässt dem System Pausen zum Abkühlen. Dabei werden die Solltemperaturen nahe der aktuellen Temperatur gewählt, um in kurzer Zeit eine Vielzahl von Situationen abzubilden. Für das Training des neuronalen Netzwerks wurden insgesamt vier solcher Messreihen durchgeführt, von denen eine in Abbildung 10.2 dargestellt ist. Diese Messungen dauerten zusammen etwa 13 Stunden. Zur Validierung der Ergebnisse wurde ein zusätzlicher Messdurchlauf durchgeführt, der etwas über vier Stunden dauerte.

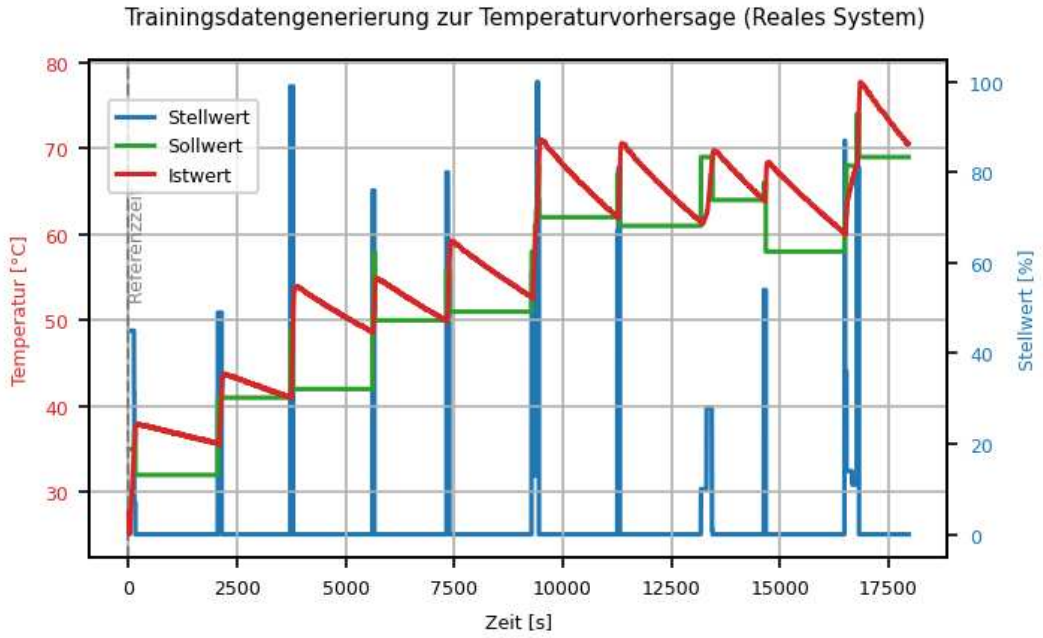


ABBILDUNG 10.2: Trainingsdaten für das FFNN

10.4.2 Training des Neuronalen Netzes

Die Datenvorverarbeitung, das Training und die Validierung des FFNNs werden im Google Colab Notebook „TrainingMitRealenDaten_FFNN_vorhersageTemp.ipynb“ durchgeführt. Zuerst müssen die Trainingsdaten importiert, aufbereitet und skaliert werden, damit sie für das Training des Modells geeignet sind. Das Training des FFNNs erstreckt sich über 300 Epochen, wobei der Fortschritt des Trainingsprozesses in Abbildung 10.3 visualisiert ist. Nach dem Training wird das Modell exportiert, um es später wiederverwenden zu können.

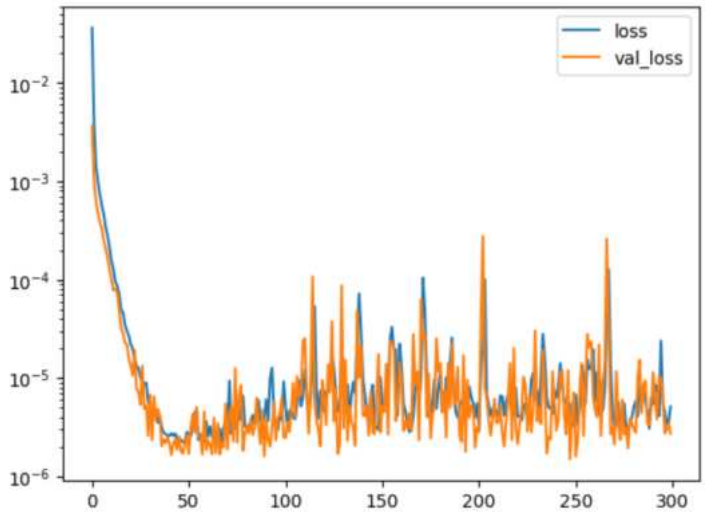


ABBILDUNG 10.3: Trainingsfortschritt FFNN

10.4.3 Validierung des Neuronalen Netzes

Um die Leistung des Modells auf unbekannten Daten zu prüfen, wurde ein Plot erstellt, der die Vorhersagen (einen Schritt voraus) des neuronalen Netzwerks mit den real gemessenen Werten aus dem Validierungsdatensatz vergleicht. Zusätzlich wird die Differenz zwischen Vorhersage und tatsächlichem Wert visualisiert und der mittlere absolute Fehler (MAE) über alle Datenpunkte berechnet. Diese Ergebnisse, sind in Abbildung 10.4 dargestellt. Zusätzlich ist in Abbildung 10.5 ein Ausschnitt des Vergleichsplots in vergrößerter Ansicht zu sehen.

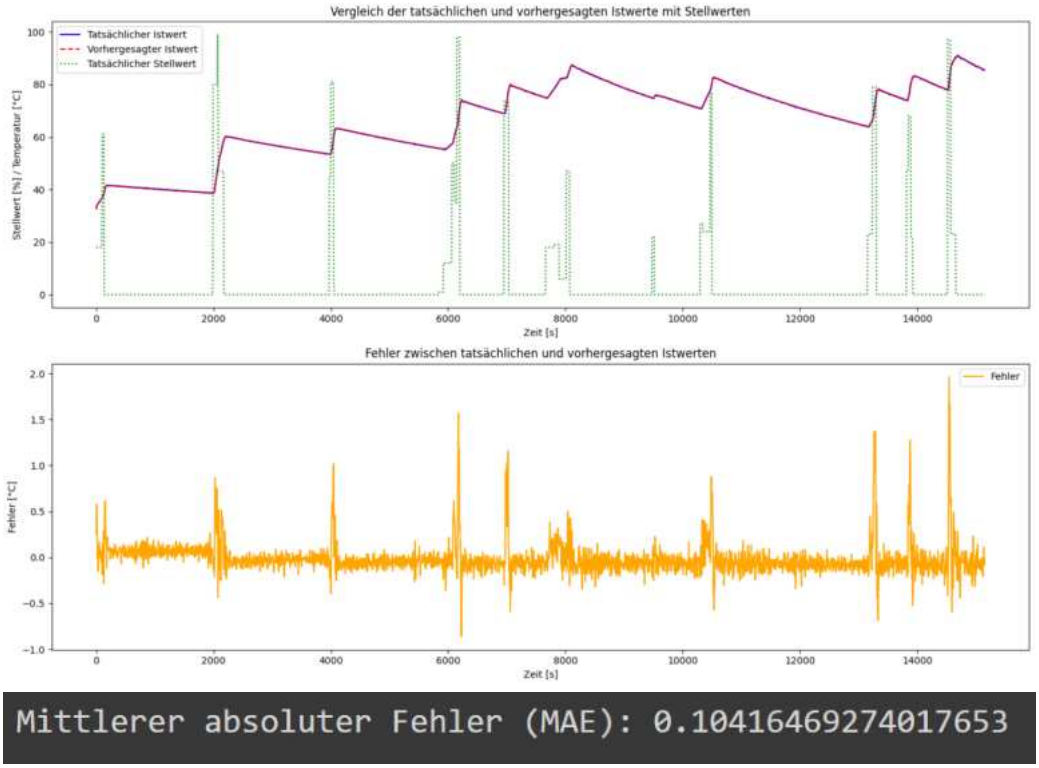


ABBILDUNG 10.4: Vorhersage auf Validierungsdaten (einen Schritt voraus)

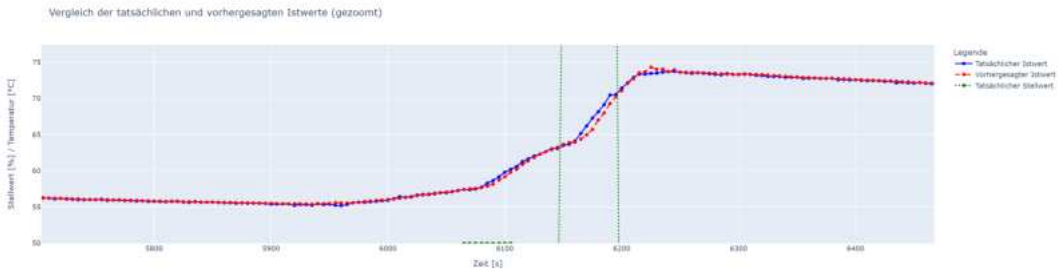


ABBILDUNG 10.5: Vorhersage auf Validierungsdaten (gezoomt)

Die Analyse der Grafiken zeigt, dass die Vorhersagegenauigkeit während der Heizphasen geringer ist als in den Abkühlphasen. Dies lässt sich durch die höhere Dynamik der Temperatur beim Aufheizen erklären. Zudem sind die Aufheizphasen in den Trainingsdaten weniger präsent als die Abkühlphasen, weil sie nicht so lange

andauern. Über die gesamte Messreihe hinweg weist das Modell einen mittleren absoluten Fehler (MAE) von $0,1\ °C$ auf, was als akzeptabel betrachtet wird, besonders wenn man das Messrauschen des Sensors berücksichtigt, das zu Datenungenauigkeiten führt.

Für den Einsatz des Modells in MPC reicht es allerdings nicht, dass das Modell nur für den nächsten Schritt genaue Vorhersagen treffen kann. Es muss auch in der Lage sein, die Temperatur über einen längeren Vorhersagezeitraum genau zu prognostizieren. Es besteht nämlich das Risiko, dass durch Ungenauigkeiten des Modells bei einem langen Vorhersagehorizont der Fehler mit jedem Schritt größer wird und schließlich ein Level erreicht, bei dem das Modell unbrauchbar wird. Um das zu testen, wurde das Modell dazu verwendet, die Temperatur über einen Zeitraum von zehn Schritten vorauszusagen. Diese Vorhersagen wurden dann mit den tatsächlichen Werten aus den Validierungsdaten verglichen, wie im Plot in Abbildung 10.6 dargestellt. Für die initiale Vorhersage dienten die tatsächlichen Datenwerte als Ausgangspunkt. Bei jeder weiteren Vorhersage wird der zuletzt vom neuronalen Netzwerk prognostizierte Wert als Basis für die nächste Vorhersage genutzt. Dieser Vorgang wiederholt sich über zehn Iterationen. Nach jeweils zehn Schritten erfolgt ein Reset, woraufhin der Prozess erneut mit dem tatsächlichen Istwert als Startpunkt beginnt.

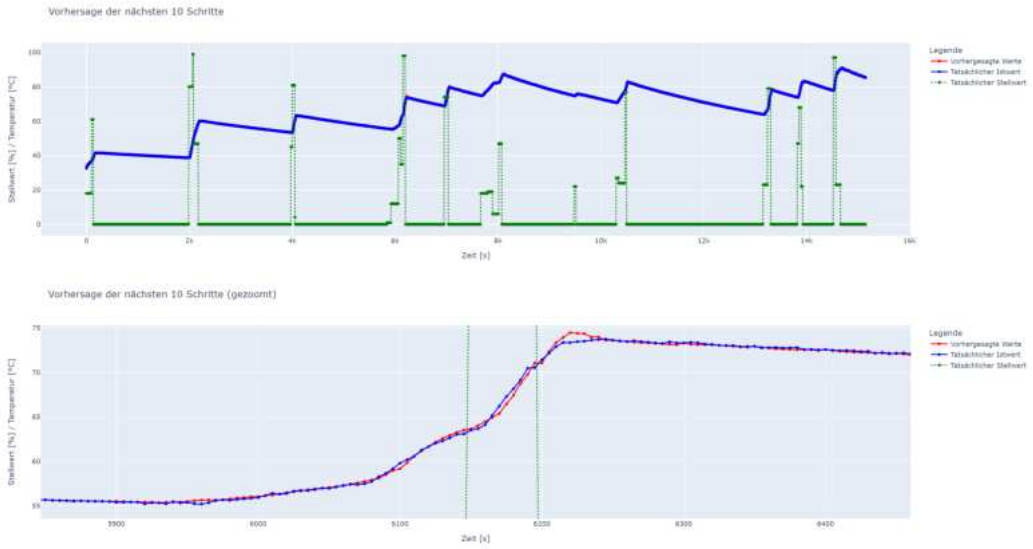


ABBILDUNG 10.6: Vorhersage auf Validierungsdaten (10 Schritte voraus)

Die Grafik zeigt, dass die roten und blauen Datenpunkte eng beieinanderliegen, was darauf hindeutet, dass das Modell die tatsächlichen Werte mit hoher Genauigkeit vorhersagt. Bei einer näheren Betrachtung der Grafik, insbesondere wenn man hineinzoomt, werden die geringfügigen Abweichungen zwischen den Vorhersagen und den realen Werten deutlicher sichtbar. Trotz dieser kleinen Unterschiede lässt sich festhalten, dass die Vorhersageleistung des Modells für praktische Anwendungen durchaus akzeptabel ist.

10.5 Integration des FFNNs in Model Predictive Control

Die Einbindung des trainierten FFNNs in MPC wird durch die Python-Klasse „dn MpcController.py“ realisiert. Diese Klasse umfasst alle erforderlichen Funktionen, um das FFNN in den MPC-Prozess zu integrieren. Dazu gehört das Laden des trainierten Modells, das Skalieren und Aufbereiten der Eingabedaten sowie das Protokollieren der Ergebnisse. Der Regler wird in einem festen Zyklus von 5 Sekunden aufgerufen, um das Optimierungsproblem zu lösen und den optimalen Stellwert zu ermitteln. Die wichtigste Funktion in der Klasse ist die „objective“-Methode. Diese Methode nimmt einen Vektor mit 5 Stellwerten (Steuerhorizont = 5) entgegen und prognostiziert aufgrund dessen den zukünftigen Temperaturverlauf des Systems . Um diese Vorhersage zu treffen, nutzt sie das trainierte neuronale Netzwerk. Anschließend wird die Kostenfunktion, wie in Gleichung 8.2 beschrieben, berechnet. Diese Funktion bewertet unter anderem den Fehler zwischen dem Sollwert und dem vom Netzwerk vorhergesagten Istwert über den Zeitraum. Das Ziel ist es nun, die Kombination von Stellwerten zu finden, die die Kostenfunktion minimiert und somit die Regelabweichung über den gesamten Vorhersagehorizont von 10 Schritten (entspricht 50 Sekunden) reduziert. Für die Lösung dieses Optimierungsproblems kommt die „minimize“-Methode aus der „scipy.optimize“-Bibliothek zum Einsatz. Diese nutzt den Broyden–Fletcher–Goldfarb–Shanno (BFGS) Algorithmus, ein numerisches Verfahren zur Lösung von nichtlinearen Optimierungsproblemen. Zur weiteren Vertiefung sind die entsprechenden Codeausschnitte im Anhang (Unterabschnitt A.4.4) dargestellt.

Anmerkung:
Der Optimierungsalgorithmus arbeitet mit einem numerischen Verfahren, das die „objective“-Funktion potenziell tausendfach aufrufen kann. Die Häufigkeit dieser Aufrufe hängt davon ab, wie schnell eine Lösung gefunden wird und wie komplex das zu lösende Problem ist. Deshalb ist es wichtig den Berechnungsprozess so effizient wie möglich zu gestalten. Eine Methode die zur Effizienzsteigerung des Optimierungsalgorithmus angewendet wurde, ist die intelligente Wahl der initialen Stellwerte. Dafür werden die optimalen Stellwerte des letzten Durchlaufs als Startpunkte für die nächste Suche verwendet. Diese Herangehensweise verhindert, dass bei jedem Optimierungsdurchgang von vorne mit zufälligen Werten begonnen werden muss, und nutzt die erfolgreichen Ergebnisse der letzten Suche als solide Basis für die weitere Optimierung.

Ein weiteres Hindernis zu Beginn war die lange Ausführungsdauer der „predict“-Methode aus der Keras-Bibliothek, die für jede Vorhersage des Modells etwa 30 ms in Anspruch nahm und den Optimierungsprozess somit auf mehrere Sekunden ausdehnte. Teilweise schaffte es der Optimierer nicht innerhalb der Abtastzeit von 5 s eine Lösung zu finden. Eine deutliche Verbesserung wurde erreicht, indem die Gewichte des Modells extrahiert und das FFNN manuell rekonstruiert wurde, was die Ausführungszeit erheblich verkürzte. Dadurch ist es nun möglich, dass der Optimierer in weniger als 100 ms zuverlässig eine Lösung findet.

10.6 Reglerbewertung anhand des Testdurchlaufs

Um die Leistung des FFNNs in Kombination mit MPC im realen Anwendungsfall zu bewerten, wird der Testlauf aus Abschnitt 6.1 durchlaufen. Die Ergebnisse dieses Tests und die entsprechenden Messdaten sind in Abbildung 10.7 visualisiert.

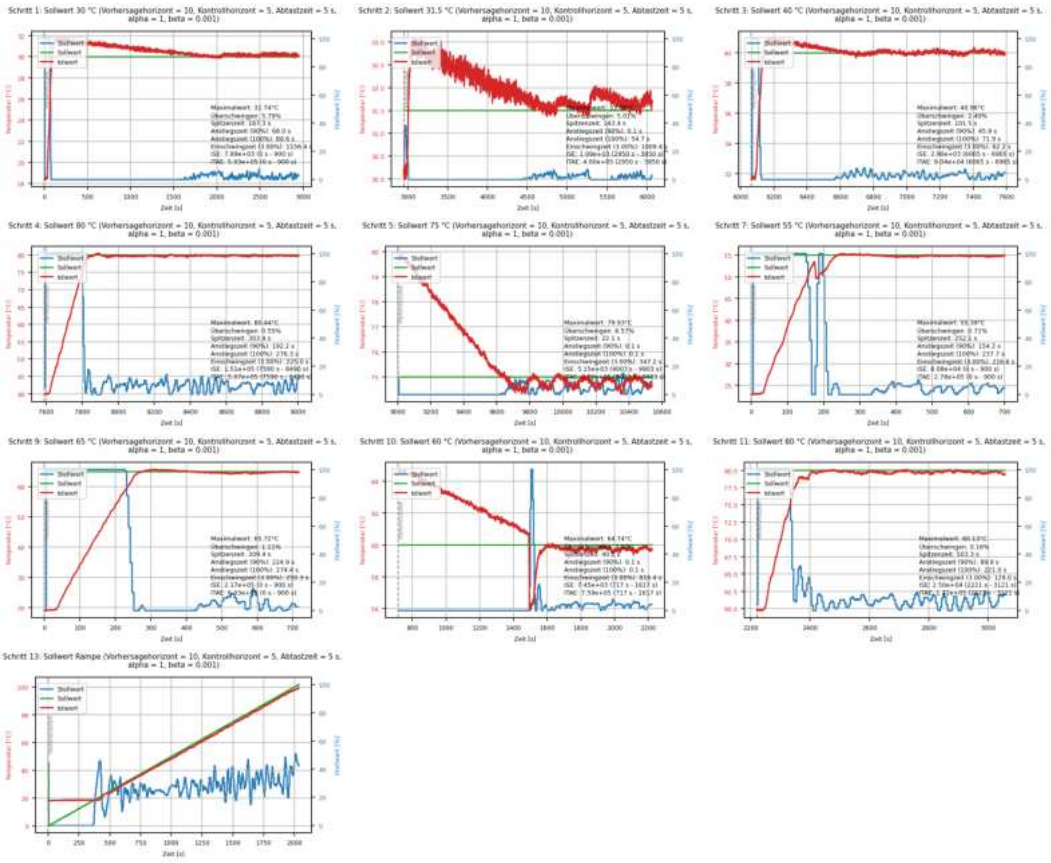


ABBILDUNG 10.7: Testdurchlauf MPC-Regler mit FFNN zur Modellvorhersage

Bei der Auswertung der Messergebnisse, zeigt sich, dass der FFNN-basierte MPC-Regler überraschend gute Resultate liefert und in vielen Szenarien ähnlich performt wie der MPC-Regler aus Abschnitt 8.3. Es gibt allerdings Situationen, in denen der FFNN-MPC-Regler schlechtere Ergebnisse erzielt, insbesondere erreicht er den Sollwert nicht immer mit der gleichen Präzision oder zeigt gerade zu Beginn im niedrigen Temperaturbereich ein stärkeres Überschwingen. Dies könnte darauf zurückzuführen sein, dass die Trainingsdaten diese Temperaturbereiche nicht ausreichend abdecken und das Modell hier weniger genau ist. Zusammenfassend lässt sich feststellen, dass die Sollwerte bereits recht genau erreicht werden. Eine Erweiterung der Trainingsdaten, besonders im niedrigeren Temperaturbereich, kann die Vorhersagegenauigkeit des Modells weiter verbessern. Mit solchen Anpassungen könnte der Regler Ergebnisse liefern, die für praktische Anwendungen durchaus zufriedenstellend sind.

10.7 Fazit und Ausblick auf zukünftige Entwicklungen

In diesem Kapitel wurde beschrieben, wie ein MPC-Regler, welcher als Vorhersagemodell ein FFNN nutzt, designet und implementiert werden kann. Dieses Modell wurde ausschließlich mit realen Daten trainiert. Wie der Testdurchlauf gezeigt hat, ist der Regler in der Lage vorgegebene Temperatursollwerte mit hinreichender Genauigkeit und Schnelligkeit zu erreichen. In diesem spezifischen Anwendungsfall, der Temperaturregelung eines Wasserbeckens, hat sich der verwendete Regler als zufriedenstellend erwiesen. Im Folgenden werden die gewonnenen Erkenntnisse

zusammengefasst und sowohl die Herausforderungen als auch die Möglichkeiten und das Verbesserungspotenzial der Methode dargestellt. Dabei wird nicht nur die Leistung im Rahmen dieser Arbeit betrachtet, sondern auch das Potenzial für zukünftige Entwicklungen. Es wird auch die Idee diskutiert, ein Tool zu entwickeln, das diesen Prozess automatisieren könnte. Dadurch könnte man auf diese Weise Regler sowohl für unbekannte Systeme als auch für andere Regelungsaufgaben entwerfen.

10.7.1 Verbesserungspotenzial für zukünftige Implementierungen

Um die Leistung des in diesem Kapitel implementierten Reglers zu verbessern, gibt es verschiedene Möglichkeiten:

- Ein erster, naheliegender Ansatz besteht darin, systematisch mit verschiedenen Systemparametern zu experimentieren. Dazu zählen zum Beispiel der Vorhersagehorizont, der Kontrollhorizont, die Abtastzeit sowie das Design des neuronalen Netzwerks. Durch Anpassungen dieser Parameter könnte die Effizienz und Genauigkeit des Reglers verbessert werden.
- Ein weiterer wichtiger Punkt ist die Generierung einer ausreichend großen und repräsentativen Datenmenge für das Training des neuronalen Netzwerks. Wie in Abbildung 10.2 zu sehen, basierten die Trainingsdaten bisher auf einer eher zufälligen Auswahl von Temperatursollwerten, was zu Genauigkeitseinbußen in weniger repräsentierten Temperaturbereichen führte. Zukünftige Verbesserungen könnten von einem intelligenteren Algorithmus einhergehen, der systematisch eine breite Palette von Ist- und Sollwertkombinationen erfasst. Ziel wäre es, diese Daten schnell und intelligent zu sammeln, um das Training des neuronalen Netzwerks möglichst effektiv zu gestalten.
- Zusätzlich besteht Verbesserungspotenzial bei der Wahl des verwendeten Optimierungsalgorithmus zur Bestimmung der optimalen Stellwerte über den Vorhersagehorizont. In diesem Projekt kam der BFGS-Algorithmus zum Einsatz, der in diesem spezifischen Anwendungsfall erfolgreich war. Es lässt sich jedoch nicht garantieren, dass er unter anderen Bedingungen oder bei anderen FFNN-Designs das Optimierungsproblem genauso schnell und effizient lösen wird. Insbesondere ist nicht immer sichergestellt, ob das gefundene Minimum nur lokal oder auch global optimal ist. Es könnte daher sinnvoll sein, andere Optimierungsalgorithmen zu prüfen oder den aktuellen Algorithmus so anzupassen, dass er besser auf diese spezifische Art von Optimierungsproblem zugeschnitten ist.
- Eine weitere Verbesserungsmöglichkeit besteht darin, das Modell kontinuierlich während des Betriebs zu aktualisieren. In diesem Projekt erfolgte das Training des Modells nur einmalig mit einem festen Datensatz. Zukünftig könnte man einen Automatismus entwickeln, der während des Betriebs laufend Daten sammelt und filtert. Dabei könnten zum Beispiel Daten, die über längere Zeit gleiche Sollwerte zeigen und daher weniger Aussagekraft haben, automatisch herausgefiltert werden. Die verbleibenden, relevanten Daten könnten dann verwendet werden, um das Modell regelmäßig nachzutrainieren oder neu zu trainieren. Dies würde die Genauigkeit des Modells verbessern und es an eventuelle Änderungen im System anpassen.

Diese Verbesserungen könnten den Regler effizienter und zuverlässiger gestalten und somit eine solide Basis für zukünftige Entwicklungen bieten. Sie überschreiten jedoch den Rahmen dieser Arbeit aufgrund ihres Umfangs.

10.7.2 Stärken und Vorteile des MPC-Reglers mit neuronalem Netz als Vorhersagemodell

- **Lernfähigkeit aus echten Daten:**
  Ein wesentlicher Vorteil eines MPC-Reglers mit einem neuronalen Netzwerk (FFNN) als Vorhersagemodell liegt in seiner Fähigkeit, direkt aus echten Systemdaten zu lernen. Im Gegensatz zu traditionellen MPC-Reglern, die auf mathematischen Modellen basieren und detaillierte Kenntnisse der physikalischen Vorgänge erfordern, bietet das FFNN eine größere Flexibilität. Es kann aus den vorhandenen Daten lernen, ohne dass explizite Modellannahmen nötig sind, was besonders in Systemen nützlich ist, in denen das physikalische Verhalten unbekannt oder schwer zu modellieren ist.

- **Anpassungsfähigkeit an Systemveränderungen:**
  Ein weiterer Aspekt des FFNN-basierten MPC-Reglers ist seine Fähigkeit, sich an Veränderungen im Systemverhalten anzupassen. Während traditionelle Modelle oft durch starre Modellannahmen eingeschränkt sind, die die Realität nicht immer genau abbilden, kann ein neuronales Netzwerk durch kontinuierliches Training mit aktuellen Daten fortlaufend verbessert und an Änderungen im Systemverhalten angepasst werden. Dies ermöglicht es dem Regler, seine Genauigkeit und Effektivität mit zunehmender Datenmenge zu steigern.

- **Anwendbar bei komplexen Systemen:**
  Im Vergleich zu PID-Reglern bietet der FFNN-basierte MPC-Regler den zusätzlichen Vorteil, effektiv mit komplexen oder nichtlinearen Systemen umgehen zu können. PID-Regler stoßen oft an ihre Grenzen, wenn es um die Handhabung von Systemen mit komplexer Dynamik geht.

10.7.3 Nachteile und Limitierungen des Reglermodells

- **Datenabhängigkeit:**
  Ein wesentlicher Nachteil des FFNN-basierten MPC-Reglers ist seine starke Abhängigkeit von der Qualität und Quantität der verfügbaren Trainingsdaten. Eine nicht ausreichend große oder repräsentative Datenmenge kann die Vorhersagefähigkeit des neuronalen Netzwerks und somit die Leistung des Reglers beeinträchtigen.

- **Transparenz und Interpretierbarkeit:**
  Neuronale Netzwerke, insbesondere tiefere Architekturen, können als „Black-Box“ angesehen werden, da ihre Entscheidungsfindungsprozesse oft nicht transparent sind. Zusätzlich trägt der Optimierungsalgorithmus des MPC-Reglers zur mangelnden Transparenz bei, da auch seine Funktionsweise und Entscheidungsfindung oft nicht vollständig durchschaubar sind. Diese doppelte Intransparenz kann die Fehlersuche und die Optimierung des Regelungssystems erheblich erschweren, da nicht immer eindeutig ist, wie oder warum das Netzwerk oder der Algorithmus eine bestimmte Entscheidung trifft. Diese Undurchsichtigkeit stellt besonders in kritischen Anwendungen ein Problem dar, bei denen Fehlentscheidungen des Reglers ernsthafte Konsequenzen haben können.

- **Geeignete Wahl der Parameter:**
  Das Lernen des Modells aus realen Daten erlaubt es, einen Regler auch ohne genaue Kenntnis des physikalischen Systems zu entwickeln. Allerdings müssen einige Parameter wie Abtastzeit, Vorhersagehorizont, die Menge der historischen Daten und die Anzahl der Neuronen im FFNN gut überlegt sein. Hierfür ist ein gewisses Maß an Erfahrung oder intuitives Verständnis notwendig, da ein planloses Herumprobieren vermutlich nicht zielführend sein wird. Es ist wichtig, ein Gefühl für die Systemdynamik zu haben und zu verstehen, wie sich unterschiedliche Parameter auf die Leistung des Reglers auswirken können.

  Beispiel: Bereits die Auswahl der richtigen Abtastzeit erwies sich in diesem Projekt als eine Herausforderung. Beim Aufheizen des Systems sind deutliche Temperaturänderungen innerhalb einer Sekunde zu beobachten, während beim Abkühlen in derselben Zeitspanne kaum Unterschiede, abgesehen von Messrauschen, festzustellen sind. Da das FFNN darauf trainiert wird, die Temperatur jeweils für einen Zeitschritt vorherzusagen, kann eine zu kurze Abtastzeit dazu führen, dass das Modell die Temperatur während des Abkühlvorganges nicht richtig vorhersagen kann. Ist die Abtastzeit zu lang, kann der Regler eventuell nicht schnell genug auf die Temperaturänderungen beim Heizvorgang reagieren. Eine Abtastzeit von fünf Sekunden stellte in diesem Fall einen guten Kompromiss dar. Würde sich allerdings die Dynamik zwischen Aufheizen und Abkühlen noch stärker unterscheiden, würde das zu Problemen bei der Wahl der Abtastzeit führen. Ähnliche Abwägungen sind auch bei anderen Parametern erforderlich, wie zum Beispiel der Anzahl der historischen Eingangsdaten für das FFNN oder dem Vorhersagehorizont. Auch hier müssen Kompromisse gefunden werden, um sicherzustellen, dass wichtige Details erhalten bleiben, ohne den Rechenaufwand und die Komplexität des Systems unnötig zu steigern.

- **Trainingsaufwand:**
  Bevor der Regler zum Einsatz kommen kann, ist es nötig, passende Trainingsdaten zu sammeln. Dieser Vorgang kann je nach Systemtyp sehr lange dauern, vor allem wenn es gilt, den gesamten Arbeitsbereich des Systems zu erfassen. Bei manchen Systemen kann es sogar riskant oder praktisch unmöglich sein, gewisse Zustände sicher zu erreichen. Außerdem beansprucht das Training des neuronalen Netzwerks hohe Rechenkapazitäten, weshalb es nicht auf allen Systemen problemlos durchgeführt werden kann. Allerdings besteht die Möglichkeit, das Training auf externe Ressourcen auszulagern, um dieses Problem zu umgehen. Auch die Möglichkeit, das Modell regelmäßig zu aktualisieren, um es an neue oder sich ändernde Betriebsbedingungen anzupassen, bringt zusätzlichen logistischen Aufwand mit sich und erfordert einen kontinuierlichen Ressourceneinsatz.

- **Hohe Rechenanforderungen im Betrieb:**
  Nicht nur für das Training, sondern auch im laufenden Betrieb benötigt der MPC-Regler mit neuronalem Netzwerk gewisse Rechenanforderungen. In Regelsystemen mit hoher Dynamik, wo schnelle Anpassungen nötig sind, kann der hohe Rechenaufwand eine Umsetzung in Echtzeit erschweren oder sogar unmöglich machen. Im Gegensatz zum PID-Regler ist es schwierig, diesen

Reglertyp auf Geräten mit begrenzter Rechenleistung, wie kleinen Mikrocontrollern, zu betreiben.

10.7.4 Fazit

Die in diesem Kapitel beschriebene Entwicklung und Implementierung eines MPC-Reglers mit einem FFNN als Vorhersagemodell hat gezeigt, dass diese Methode gute Ergebnisse bei der Erreichung der Solltemperaturen liefert und von der Fähigkeit des neuronalen Netzwerks profitiert, direkt aus realen Daten zu lernen. Die Anwendung dieser Strategie bietet signifikante Vorteile, stellt aber auch einige Herausforderungen dar. Der größte Vorteil ist die Möglichkeit, ohne umfangreiches Vorwissen über das Systemverhalten, effektive Regelungsstrategien zu entwickeln. Trotzdem ist ein gewisses Grundverständnis erforderlich, um die Parameter angemessen zu verstehen und effektiv einzustellen. Auch die Implementierung kann logistisch anspruchsvoll sein und ist nicht auf allen Systemen direkt möglich. Darüber hinaus bleibt die Anwendung traditioneller Regelungsmethoden die bevorzugte Wahl, wenn Genauigkeit, Transparenz und Zuverlässigkeit entscheidend sind und das Systemverhalten bekannt und modellierbar ist. Der hier diskutierte Ansatz eröffnet jedoch spannende Möglichkeiten für zukünftige Weiterentwicklungen und könnte, mit weiterer Optimierung und Vereinfachung der Prozesse, auch breitere Anwendungsfelder erfolgreich erschließen. Zum Beispiel kann ein Tool entwickelt werden, das diese Methoden nutzt, sodass auch Anwender ohne umfangreiche Erfahrungen im Bereich der Regelungstechnik funktionierende Regler für verschiedene Anwendungsfälle entwerfen und anwenden können. Die Entwicklung dieses Tools wird in diesem Projekt nicht weiter dokumentiert. Um dennoch einen Eindruck zu vermitteln, wie es letztendlich funktionieren könnte, wird im nächsten Unterabschnitt ein möglicher Entwurfsansatz dazu vorgestellt.

10.7.5 Möglichkeit für eine automatisierte Prozessführung mittels eines Entwicklungsassistenten

Die in diesem Kapitel dargelegten Schritte, beginnend mit der Erzeugung der Trainingsdaten, über das Training des neuronalen Netzes (FFNN), bis hin zum Einsatz des Reglers, können durch ein Tool automatisiert werden. Ein solches Tool würde als interaktiver Entwicklungsassistent fungieren, der den Nutzer durch jeden einzelnen Schritt führt. Am Ende dieses Prozesses steht dem Anwender ein einsatzbereiter Regler zur Verfügung, den er nur noch starten muss. Eine mögliche Funktionsweise und das Design dieses Tools werden in den nachfolgenden Bildern näher beschrieben.

Im Hauptfenster des Entwicklungsassistenten, dargestellt in Abbildung 10.8, können Nutzer die grundlegenden MPC-Parameter einstellen. Ist das Vorhersagemodell bereits trainiert, bietet das Tool die Möglichkeit, einen manuellen Sollwert vorzugeben und den Regler zu starten und zu stoppen. Im Betrieb erfasst das Tool kontinuierlich Daten wie Soll-, Ist- und Stellwerte, die sich grafisch in einem Plot darstellen lassen. Sollte das Vorhersagemodell noch nicht trainiert sein, führt ein Klick auf „Einstellungen Vorhersagemodell“ zu einem weiteren Fenster, das die Einrichtung und das Training des Modells ermöglicht.

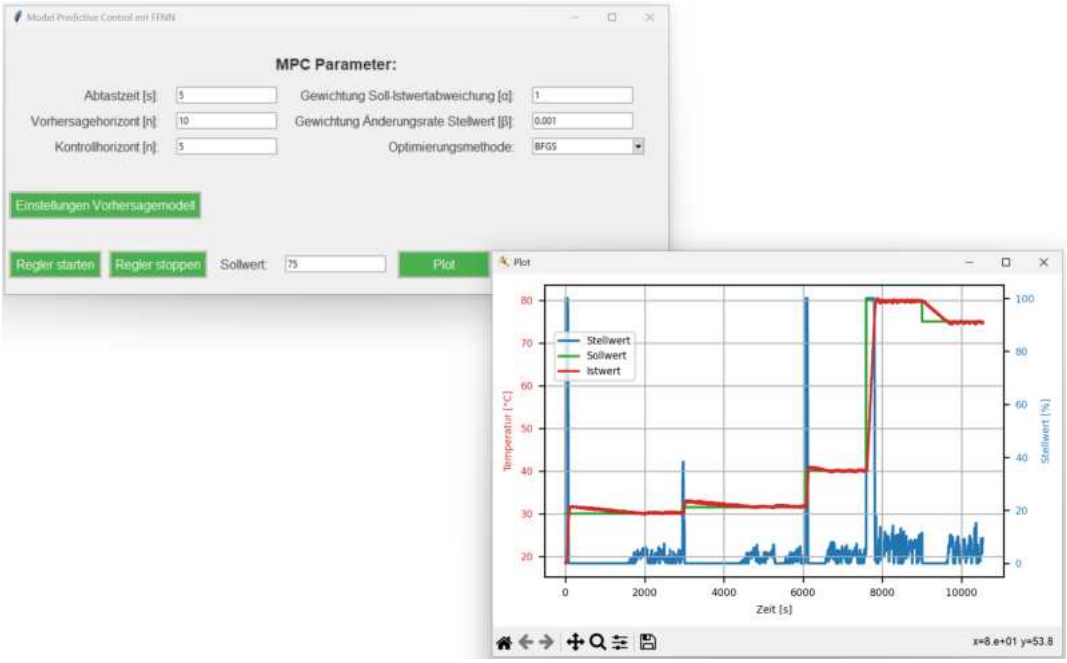


ABBILDUNG 10.8: Hauptfenster des Entwicklungsassistenten

Im folgenden Fenster (Abbildung 10.9) können Nutzer die Einstellungen für das FFNN, das für die Vorhersage der Systemreaktion zuständig ist, anpassen. Es besteht die Möglichkeit, ein schon trainiertes Modell zu importieren oder zu exportieren. Falls noch kein Modell vorliegt, kann ein neues erstellt werden.

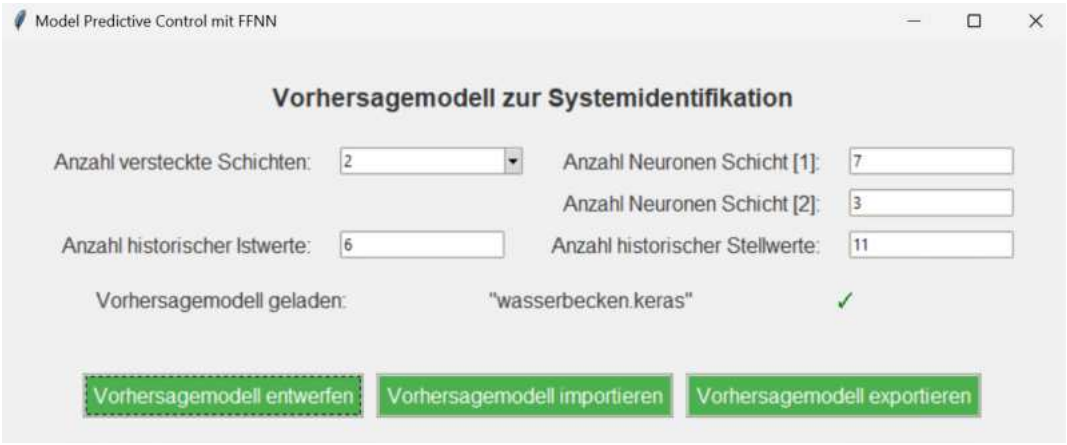


ABBILDUNG 10.9: Entwicklungsassistent: Einstellungen Vorhersagemodell

Entscheidet man sich für die Entwicklung eines neuen Modells, öffnet sich das Fenster, das in Abbildung 10.10 dargestellt ist. Hier wird der Benutzer durch den Trainingsprozess geführt. Sollten keine Trainingsdaten vorhanden sein, können diese durch das Einstellen der notwendigen Parameter und das Betätigen des Buttons „Trainingsdaten generieren“ erstellt werden. Dabei führt das System verschiedene Stellwerte innerhalb des festgelegten Bereichs aus und zeichnet die Reaktionen des realen Systems auf. Nachdem die Aufzeichnung abgeschlossen ist und die Daten als zufriedenstellend angesehen werden, kann das eigentliche Training des Modells erfolgen. Hierfür werden die Anzahl der Trainingsepochen festgelegt und die Aufteilung der Daten für das Training und die Validierung bestimmt. Nach erfolgreichem

Training lässt sich das FFNN evaluieren, indem es Vorhersagen auf Basis der Validierungsdaten generiert. Mithilfe von Diagrammen, die die tatsächliche Systemreaktion mit den Modellvorhersagen vergleichen, lässt sich beurteilen, ob das Modell den Anforderungen entspricht. Ist das Modell zufriedenstellend, kann es im Regelprozess eingesetzt und getestet werden.

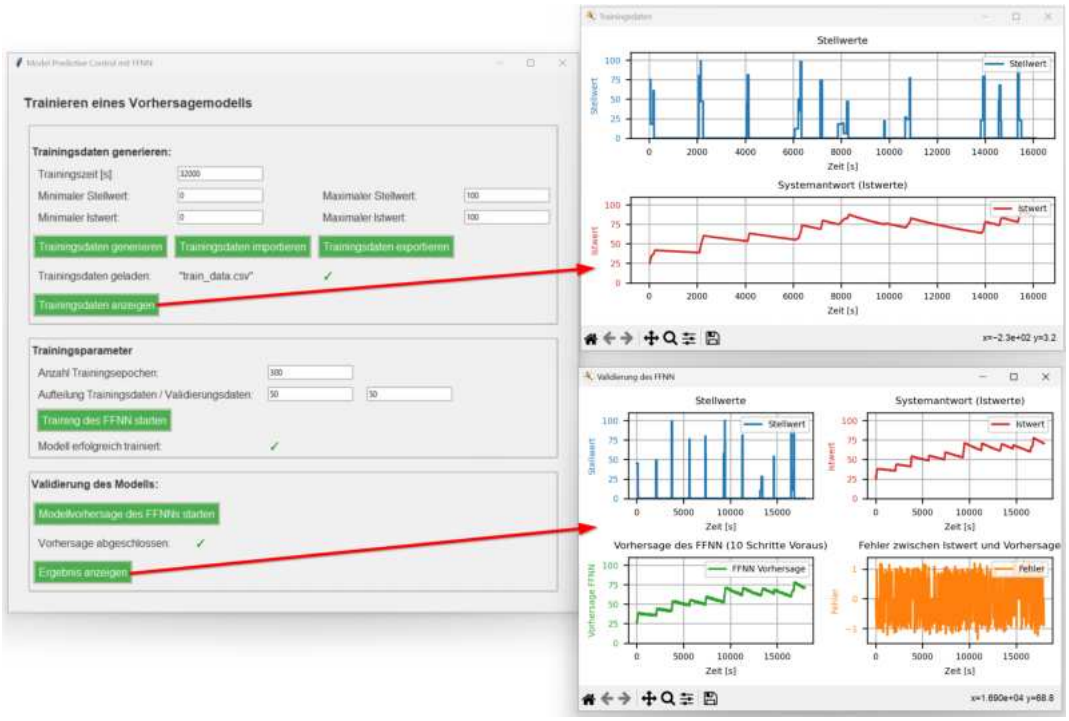


ABBILDUNG 10.10: Entwicklungsassistent: Trainieren des Vorhersagemodells

Ein Tool wie dieses erleichtert nicht nur die schnelle Entwicklung eines Reglers, sondern macht fortschrittliche Regelungstechniken auch für Anfänger zugänglich, die möglicherweise noch keine tiefgreifenden Kenntnisse in MPC oder dem Training von neuronalen Netzen haben. Es erfordert allerdings ein gewisses Grundverständnis dafür, welche Parameter potenziell sinnvoll sein können.

Kapitel 11

Zusammenfassung und Ausblick

11.1 Zusammenfassung

In dieser Arbeit wurde die Anwendung von künstlicher Intelligenz in der Regelungstechnik am Beispiel der Temperaturregelung eines Wasserbeckens untersucht. Der Fokus lag auf dem gesamten Prozess von der Identifikation des PT1-Systems bis hin zur Entwicklung sowohl konventioneller als auch KI-basierter Regelsysteme.

Zu Beginn wurde ein konventioneller PID- und ein MPC-Regler entwickelt. Anschließend erfolgte der Entwurf zweier KI-basierter Regelsysteme. Der erste dieser Regler, bestehend aus einem Long Short-Term Memory (LSTM)-Netzwerk, zielte darauf ab, das Regelverhalten des PID-Reglers nachzuahmen. Der Fokus lag dabei auf der grundsätzlichen Eignung des LSTM-Modells als Regler und den damit verbundenen Herausforderungen, weniger auf einer Perfektionierung des Systems.

Das zweite Projekt konzentrierte sich auf die Entwicklung und das Training eines Feedforward-Neuronalen Netzwerks (FFNN), welches zur Vorhersage des Verhaltens der PT1-Regelstrecke diente und als Vorhersagemodell für einen MPC-Regler fungierte. Für das Training des Netzwerks wurden Daten aus einer 13-stündigen Aufzeichnung des realen Systemverhaltens genutzt.

Beide KI-basierten Regelsysteme waren in der Lage, die Temperatur des Wasserbeckens effektiv zu regeln. Während das LSTM-Modell allerdings in einigen Szenarien noch Schwierigkeiten zeigte, welche zu stationären Regelabweichungen führten, konnte der KI-basierte MPC-Regler Ergebnisse erzielen, die mit den Leistungen der konventionellen Regler vergleichbar waren.

Die Arbeit zeigte auch, welche Herausforderungen die Implementierung dieser Reglertypen mit sich bringt. Ein wichtiger Punkt dabei ist die Qualität der Trainingsdaten für die neuronalen Netzwerke. Während der Tests wurde deutlich, dass die Regler besonders in den Betriebszuständen Probleme hatten, welche in den Trainingsdaten nicht ausreichend vertreten waren. Das macht klar, wie wichtig es ist, eine breite Palette von Betriebsbedingungen in den Trainingsdaten zu erfassen. Außerdem ist das Design des jeweiligen Netzwerks entscheidend, insbesondere die Auswahl und Strukturierung der Eingabedaten. Es ist wichtig, die Netzwerke mit ausreichend Informationen zu versorgen, ohne sie mit irrelevanten Daten zu überfluten. Zusätzlich stellen die notwendigen Rechenressourcen, sowohl für das Training der Netzwerke als auch für den Betrieb der Regler, eine weitere Herausforderung dar. Auch die Komplexität und „Black Box“-Eigenschaften von KI-Systemen können in kritischen Anwendungen problematisch sein, wo Transparenz und Vorhersagbarkeit besonders wichtig sind.

11.2 Ausblick

Die in dieser Arbeit untersuchten Reglertypen haben gezeigt, dass sie grundlegende Aufgaben wie die Temperaturregelung eines Wasserbeckens erfolgreich bewältigen können. Obwohl sie die Leistung herkömmlicher Regler nicht übertreffen konnten, zeigen sie dennoch interessante Perspektiven für zukünftige Anwendungen.

Ein möglicher Ansatz zur Weiterentwicklung des LSTM-Reglers wäre die Implementierung eines neuronalen Netzes, das durch überwachtes Lernen die Dynamik verschiedener existierender Regler erfasst. Ein solches System könnte nicht nur Daten von einem einzelnen Reglertyp, sondern von vielen verschiedenen Reglern und Systemen mit unterschiedlichem Verhalten verwenden. Auf diese Weise könnte das Netz grundlegende Zusammenhänge zwischen Systemverhalten und Reglerreaktion lernen und so als universeller Regler dienen. Dieser Regler würde dann zwar die Performance der einzelnen Regler nicht übertreffen, hätte aber den Vorteil, dass er für verschiedene Systeme eingesetzt werden kann und sich dynamisch an veränderte Systembedingungen anpasst. Eine der Herausforderungen liegt jedoch in der Beschaffung einer umfangreichen und qualitativ hochwertigen Datensammlung von bestehenden Reglern.

Der Einsatz eines neuronalen Netzes als Vorhersagemodell für MPC bietet die Möglichkeit, ein Tool zu entwickeln, das den Entwurf von Reglern für unbekannte Systeme ermöglicht, ohne dass detailliertes Wissen über das Systemverhalten erforderlich ist. Der Schlüssel zu dieser Methode liegt im Trainingsprozess des Reglers. Dieser ist nämlich in der Lage, die benötigten Daten selbstständig zu generieren, indem er verschiedene Stellgrößen am System testet und aus den Reaktionen lernt. Eine Grundvoraussetzung dafür ist jedoch die Möglichkeit, diese Eingriffe gefahrlos am System durchführen zu können. Außerdem muss für diesen Prozess der Datengenerierung ausreichend Zeit und Systemverfügbarkeit gegeben sein. Ein Vorteil dieses Ansatzes besteht darin, dass der Regler während des regulären Betriebs kontinuierlich Daten erfassen kann, welche zur laufenden Verbesserung des Modells genutzt werden können.

Die beiden vorgestellten Implementierungen haben das grundsätzliche Potenzial dieser Art von Regelsystemen aufgezeigt. Für einen industrietauglichen und prozesssicheren Einsatz sind jedoch noch zusätzliche Forschungen und Entwicklungen erforderlich. Bisher wurde beispielsweise nicht erforscht, wie effektiv diese Methoden bei Regelungsaufgaben jenseits der Temperaturregelung eines Wasserbeckens sind. Zudem besteht Optimierungsbedarf in der Art und Weise, wie historische Daten in die Netzwerke integriert werden (also die Gestaltung der Eingabefeatures). Ebenso wichtig ist die Entwicklung intelligenter Algorithmen, die schnell und effizient qualitativ hochwertige Trainingsdaten generieren können. Bei erfolgreicher Umsetzung dieser Verbesserungen könnte aus diesen Ansätzen ein innovatives Produkt in der Regelungstechnik entstehen, das auch von Laien effektiv eingesetzt werden kann.

Anhang A

Python-Programm-Übersicht

A.1 Übersicht der Eigenentwickelten Python-Klassen

Klasse	Beschreibung
Gui	Diese Klasse umfasst alle Funktionen, die für die grafische Benutzeroberfläche notwendig sind. Sie wird zu Beginn des Programms innerhalb der main-Funktion instanziiert. Über die Schaltflächen der Benutzeroberfläche können dann verschiedene Aktionen gesteuert werden, wie das Starten und Stoppen der Regler sowie das Einstellen verschiedener Parameter.
IoModule	Diese Klasse beinhaltet alle Funktionalitäten bezüglich des E/A-Moduls. Beim Start des Programms wird die Verbindung zum E/A-Modul überprüft und die Register des Moduls entsprechend der nötigen Konfiguration beschrieben. Es wird ein Thread gestartet in welchem im 100 ms Takt der Temperatursensor ausgelesen und geglättet wird. Außerdem bietet diese Klasse Methoden um den Sollwert des Tauchsieders zu schreiben und den zuletzt aktualisierten Temperaturwert zu lesen. Der Sollwert des Tauchsieders wird in $0 - 100$ % der Leistung vorgegeben. Die Umrechnung von % in die entsprechende Spannung übernimmt die Methode „LinearizeVoltage“.
DataLoader	Diese Klasse verfügt über Funktionen zum Lesen von CSV-Dateien, die genutzt werden, um zuvor erstellte Log-Dateien von Messungen zu importieren und für weitere Analysen aufzubereiten.
DataLogger	Diese Klasse bietet Funktionen, um ausgewählte Daten in einer CSV-Datei zu speichern.
SimPt1	Diese Klasse dient dazu das PT1-Streckenverhalten zu simulieren.
Pid	Diese Klasse umfasst sämtliche Funktionen im Zusammenhang mit dem PID-Regler. Ein separater Thread führt den Regler mit einer Abtastfrequenz von 100 ms aus. Zudem werden Daten wie Sollwert, Istwert und Stellgröße zusammen mit den jeweiligen Zeitstempeln in einer Log-Datei dokumentiert.
Mpc	Diese Klasse umfasst sämtliche Funktionen im Zusammenhang mit dem MPC-Regler. Ein separater Thread führt den Regler mit einer eingestellten Abtastfrequenz aus. Zudem werden Daten wie Sollwert, Istwert und Stellgröße zusammen mit den jeweiligen Zeitstempeln in einer Log-Datei dokumentiert.

Klasse	**Beschreibung**
Lstm	Diese Klasse umfasst sämtliche Funktionen im Zusammenhang mit dem LSTM-Regler. Ein separater Thread führt den Regler mit einer Abtastfrequenz von 100 ms aus. Zudem werden Daten wie Sollwert, Istwert und Stellgröße zusammen mit den jeweiligen Zeitstempeln in einer Log-Datei dokumentiert.
dnMpc	Diese Klasse umfasst sämtliche Funktionen im Zusammenhang mit dem MPC-Regler, welcher zur Modellvorhersage ein Neuronales Netz verwendet. Ein separater Thread führt den Regler mit einer Abtastfrequenz von 5 s aus. Zudem werden Daten wie Sollwert, Istwert und Stellgröße zusammen mit den jeweiligen Zeitstempeln im 100 ms-Takt, in einer Log-Datei dokumentiert.

TABELLE A.1: Übersicht Python-Klassen

A.2 Übersicht der einzelnen Python-Code-Files und Notebooks

Name	Beschreibung
`MatlabPlot_Phasenanschnittsteuerung.py`	Dieses Programm plottet die Datenpunkte (Steuerspannung in Abhängigkeit der gewünschten Leistung) der Kennlinie des Thyristorstellers für die Phasenanschnittsteuerung und erzeugt die Grafiken, welche in Abbildung 5.7 zu sehen sind.
`GetPt1CharacteristicODE.py`	Das Ziel des Python-Skripts war es ursprünglich, mithilfe eines Optimierungsalgorithmus die Parameter K und T für das PT1-System so zu bestimmen, dass eine bestmögliche Annäherung an die realen Messdaten durch ein theoretisches Modell erreicht wird. Überraschenderweise stellte sich heraus, dass die vom Optimierer ermittelten Parameter den durch grafische Analyse gefundenen Werten sehr nahe kamen, wodurch die Unterschiede vernachlässigbar wurden. Aufgrund dieser Erkenntnis fand vorrangig die Plot-Funktion des Skripts Anwendung, welche einen direkten Vergleich zwischen den realen Messdaten und den durch die Differenzialgleichung vorhergesagten Werten ermöglicht, und somit eine visuelle Bestätigung der Modellgenauigkeit bietet.
`Evaluate Measurement.py`	Dieses Python Skript erstellt ein Plot von einzelnen Messungen und ermittelt gleichzeitig Bewertungskriterien wie Maximalwert, Überschwingweite, Anstiegs-, Spitzen- und Einschwingzeit sowie ISE und ITAE.
`LSTM_ersetzt_PID.ipynb`	In diesem Notebook geht es um die Entwicklung, das Training und die Auswertung eines LSTM-Modells, das darauf abzielt, die Funktion eines PID-Reglers nachzubilden. Das Notebook führt durch den gesamten Prozess von der Datenvorbereitung über das Design des neuronalen Netzes bis hin zur Evaluierung der Modellleistung.
`TrainingMit RealenDaten_FFNN_vorhersageTemp.ipynb`	Dieses Notebook beschäftigt sich mit der Entwicklung, dem Training und der Auswertung eines FFNNs, das darauf ausgelegt ist, die zukünftige Temperatur des Wasserbeckens basierend auf historischen Stell- und Istwerten vorherzusagen. Dieses FFNN wird anschließend exportiert und in der MPC als Vorhersagemodell verwendet. Das Notebook führt durch den gesamten Prozess von der Datenvorbereitung über das Design des neuronalen Netzes bis hin zur Evaluierung der Modellleistung.
`FFNN_vorhersageTemp_Simulation.ipynb`	Ähnlich dem vorherigen Notebook, fokussiert dieses sich auf die Entwicklung, das Training und die Auswertung eines FFNNs. Allerdings werden für das Training Simulationsdaten erzeugt und verwendet und keine realen Daten.

TABELLE A.2: Übersicht Python-Code-Files und Notebooks

A.3 Übersicht der verwendeten Bibliotheken

In diesem Abschnitt werden die relevanten Python-Bibliotheken beschrieben, welche in diesem Projekt verwendet wurden.

- **Matplotlib** ist eine Open-Source-Bibliothek zur Erstellung von Grafiken. Sie bietet eine Vielzahl von Funktionen zum Erstellen von Diagrammen, Plots, Histogrammen und anderen visuellen Darstellungen von Daten [10]. Sie wurde in diesem Projekt zur Erstellung verschiedener Plots eingesetzt.

- **Tkinter** ist das Standard-Toolkit in Python, um grafische Benutzeroberflächen zu erstellen. Diese Bibliothek erleichtert das Erstellen von Fenstern, Schaltflächen, Menüs und weiteren grafischen Elementen, um benutzerfreundliche Interfaces zu schaffen [24].

- **NumPy** ist eine Python-Bibliothek, die für effiziente numerische Berechnungen entwickelt wurde und insbesondere im Bereich des maschinellen Lernens von großer Bedeutung ist. Sie ist optimiert, um komplexe Berechnungen schnell und einfach durchzuführen, ermöglicht leistungsstarke Operationen auf großen Datensätzen und integriert sich gut mit anderen Python-Bibliotheken, was sie für Algorithmen im maschinellen Lernen unverzichtbar macht [2].

- **Pandas** ist eine leistungsfähige Bibliothek, die sich ideal für die Datenvorbereitung im maschinellen Lernen eignet. Sie ermöglicht das effiziente Bearbeiten großer Datensätze und bietet eine Vielzahl nützlicher Funktionen wie das Zusammenführen, Umformen und Zusammenfassen von Daten, bevor sie in ML-Modelle integriert werden [2].

- **SciPy** ist eine Bibliothek für wissenschaftliche Berechnungen, die auf NumPy aufbaut und dessen Funktionen erweitert. Sie bietet Möglichkeiten zur numerischen Integration, Optimierung, Signalverarbeitung und Statistik. Das im Projekt verwendete Modul „optimize“ beinhaltet verschiedene Algorithmen für das Lösen von komplexen Optimierungsproblemen [11].

- **Scikit-learn**, oft abgekürzt als sklearn, ist eine Open-Source-Bibliothek, die eine breite Palette von Werkzeugen für maschinelles Lernen bereitstellt. Sie zeichnet sich durch ihre Benutzerfreundlichkeit aus und konzentriert sich auf traditionelle Methoden des maschinellen Lernens, einschließlich überwachtem und unüberwachtem Lernen [2]. In diesem Projekt wurden hauptsächlich die Datenvorverarbeitungsfunktionen von sklearn verwendet, wie zum Beispiel Funktionen zur Skalierung von Daten oder zur Aufteilung von Datensätzen in Trainings- und Testdatensätze.

- **TensorFlow** ist ein von Google entwickeltes Open-Source-Framework für maschinelles Lernen. Es wurde speziell für die Erstellung und das Training komplexer Modelle, insbesondere tiefer neuronaler Netze, entwickelt [2]. In diesem Projekt wurde TensorFlow verwendet, um die neuronalen Netze zu entwickeln und zu trainieren.

- **Keras** ist eine benutzerfreundliche Schnittstelle für Frameworks wie TensorFlow, welche die Implementierung von neuronalen Netzen vereinfacht. Sie erleichtert schnelles Experimentieren und Anpassen von komplexen Lernmodellen [2].

A.3.1 Unterschiede zwischen Bibliotheken für das Maschinelle Lernen

Es gibt mehrere Bibliotheken, die zur Erstellung und zum Training neuronaler Netzwerke verwendet werden können. Zu den bekanntesten gehören Scikit-learn, TensorFlow und PyTorch. Im Folgenden werden die grundlegenden Unterschiede dieser Bibliotheken beschrieben und die Entscheidung für TensorFlow in diesem Projekt erläutert.

Scikit-learn:
Scikit-learn ist eine Bibliothek, die sich auf traditionelles maschinelles Lernen konzentriert. Sie bietet eine breite Palette an Algorithmen, die ohne tiefgreifende Lernmechanismen auskommen. Diese Bibliothek ist ideal für einfache maschinelle Lernprojekte, die keine komplexen neuronalen Netzwerkstrukturen erfordern.

TensorFlow und PyTorch:
Im Gegensatz dazu sind TensorFlow und seine High-Level-API Keras speziell für tiefes Lernen und die Arbeit mit komplexen neuronalen Netzwerken konzipiert. Ein alternatives Produkt zu TensorFlow ist das von Facebook entwickelte PyTorch, das ebenfalls für maschinelles Lernen mit großen Datenmengen und komplexen Modellen geeignet ist. Der Hauptunterschied zwischen TensorFlow und PyTorch liegt in der Art und Weise, wie sie Berechnungsgraphen handhaben:

- TensorFlow verwendet ein statisches Berechnungsgraphenmodell, das vor der Ausführung definiert wird. Dies ermöglicht Optimierungen, wie die effizientere Nutzung von Hardware-Ressourcen und eine verbesserte Leistung während des Trainings.
- PyTorch hingegen nutzt ein dynamisches Modell, bei dem der Graph während der Laufzeit erstellt wird. Dies bietet eine größere Flexibilität und erleichtert das Debugging.

Aufgrund dieser Eigenschaften wird TensorFlow häufiger in Produktionsumgebungen eingesetzt, während PyTorch in der Forschung sehr beliebt ist [40].

Entscheidung für TensorFlow:
Beide Bibliotheken sind sehr leistungsfähig und könnten den Anwendungsfall dieses Projekts problemlos abdecken. Die Entscheidung fiel aufgrund persönlicher Präferenzen auf TensorFlow, da bereits Erfahrungen damit vorhanden waren. Ein einfaches Netzwerk wie ein FFNN hätte zwar auch mit Methoden aus Scikit-learn erstellt und trainiert werden können. Für die Implementierung von LSTM-Netzwerken, die eine komplexere Netzwerkstruktur für die Verarbeitung sequenzieller Daten darstellen, sind jedoch TensorFlow oder PyTorch besser geeignet. Daher wurde TensorFlow in Kombination mit Keras für eine möglichst einfache Implementierung gewählt.

Es ist wichtig zu beachten, dass Bibliotheken wie TensorFlow und PyTorch für das Training komplexer Modelle mit langen Trainingszeiten und die Vorhersage großer Datenmengen optimiert wurden. Sie sind jedoch nicht sehr performant, wenn es darum geht, nur eine einzelne Modellvorhersage zu machen. Da das Netzwerk in diesem Projekt im Regelalgorithmus verwendet wird, wo die Ausführungszeit eine Rolle spielt, ist dies nicht vorteilhaft. Um dieses Problem zu umgehen, wurden für

das Training zwar die Keras-Bibliothek verwendet, für die Implementierung im Regler, wo es auf die Performance ankommt, jedoch die trainierten Gewichte extrahiert und das neuronale Netz manuell nachkonstruiert.

A.4 Wichtige Python-Codeausschnitte

Im Rahmen des Projekts wurde eine große Menge an Python-Code entwickelt. Dazu gehören Skripte mit zahlreichen Funktionen, wie zum Beispiel zur Auswertung verschiedener Daten, zur Generierung von Trainingsdaten, zum Training von ML-Modellen oder zur Erstellung von Plots. Außerdem wurde der Code für verschiedene Regler, das Datalogging und die GUI zur Parametereingabe implementiert. Es wäre nicht sinnvoll, den gesamten Code hier darzustellen, da dies zu umfangreich und unübersichtlich wäre. Stattdessen werden in diesem Abschnitt einige wichtige Codeausschnitte vorgestellt und kurz beschrieben.

A.4.1 PID-Regler

Diese Klasse „PidController“ enthält die Implementierung des digitalen PID-Reglers, einschließlich Anti-Windup und Tiefpassfilter für den D-Anteil. Die „Run“-Methode wird übergeordnet im 100-ms-Takt aufgerufen, um den Regler auszuführen.

```
from general.GuiParameter import GuiParameter
from general.Constants import Constants as c


class PidController:
    def __init__(self, pGuiParameter: GuiParameter, pOutMin=0, pOutMax
                     ↪ =100, pSup=0.0):
        self.sup = pSup  # Noise suppression
        self.guiParameter = pGuiParameter
        self.kp = self.guiParameter.KP  # Controller gain
        self.tn = self.guiParameter.TN  # Integral time of the
                     ↪ controller
        self.tv = self.guiParameter.TV  # Derivative time of the
                     ↪ controller

        self.OutMin = pOutMin  # Lower output limit
        self.OutMax = pOutMax  # Upper output limit
        self.integral = 0.0  # Integral value
        self.error = 0.0  # Control deviation
        self.lastError = 0.0  # previous Control deviation
        self.errorFiltered = 0.0  # Control deviation
        self.lastErrorFiltered = 0.0  # previous Control deviation

    def Run(self, pSet, pAct, pTime, pManualControlValue=None):
        if pManualControlValue is not None:
            print("Manual ControlValue: " + str(pManualControlValue))
            return pManualControlValue

        # actualize the parameters from the GUI
        self.kp = self.guiParameter.KP  # Controller gain
        self.tn = self.guiParameter.TN  # Integral time of the
                     ↪ controller
        self.tv = self.guiParameter.TV  # Derivative time of the
                     ↪ controller

        # Noise suppression
        if abs(pSet - pAct) > self.sup:
            self.error = pSet - pAct
```

```
34          else:
35              self.error = 0.0
36
37          # P component
38          p = self.kp * self.error
39
40          # I component
41          if self.tn == 0.0:
42              ki = 0
43              self.integral = 0
44          else:
45              ki = self.kp * 1.0 / self.tn
46
47          error = (self.error + self.lastError) * 0.5
48          valueToAdd = error * pTime * ki
49          self.integral += valueToAdd
50          i = self.integral
51
52          # Limit integrator (anti windup)
53          if (p+i > self.OutMax and i > 0):  # freeze integrator
54              if self.tn != 0.0:
55                  self.integral -= valueToAdd  # freeze
56              else:
57                  self.integral = 0.0
58          elif (p+i < self.OutMin and i < 0):  # freeze integrator
59              if self.tn != 0.0:
60                  self.integral -= valueToAdd
61              else:
62                  self.integral = 0.0
63          i = self.integral
64
65          # Low Pass filter for the D component
66          alpha = pTime / c.filterTimeDerivative
67          self.errorFiltered = alpha * self.error + (1 - alpha) * self.
                        ↪ lastErrorFiltered
68          # D component
69          if pTime > 0:
70              kd = self.tv * self.kp
71              d = kd * (self.errorFiltered - self.lastErrorFiltered) /
                        ↪ pTime
72          else:
73              d = 0.0
74
75          self.lastError = self.error
76          self.lastErrorFiltered = self.errorFiltered
77
78          # PID
79          output = p + i + d
80          #print(f"PID: P = {p}, I = {i}, D = {d}, Output = {output}")
81
82          # Limit output
83          if output > self.OutMax:
84              output = self.OutMax
85          elif output < self.OutMin:
86              output = self.OutMin
87          return output
88
89      def Reset(self):
90          self.integral = 0.0
91          self.error = 0.0
92          self.lastError = 0.0
93          self.errorFiltered = 0.0
94          self.lastErrorFiltered = 0.0
```

A.4.2 Klassischer MPC-Regler

Diese Klasse „MpcController“ implementiert einen konventionellen MPC-Regler, der als Vorhersagemodell die Euler-Approximation der PT1-Strecke verwendet. Zusätzlich wird die Totzeit des Systems berücksichtigt. Die Methode „objective“ prognostiziert die Kosten verschiedener Stellwerte über den Vorhersagehorizont hinweg. Auf diese Methode wird der Optimierer („scipy.optimize.minimize“) angewendet um die optimalen Stellwerte zu ermitteln. Die „Run“-Methode, welche den gesamten Prozess steuert, wird zyklisch im 1-Sekunden-Takt aufgerufen.

```
import scipy.optimize as opt
from general import GuiParameter
from general.Constants import Constants as c
import time


class MpcController:
    def __init__(self, pGuiParameter: GuiParameter, pOutMin=0, pOutMax=100):
        self.guiParameter = pGuiParameter
        self.alpha = c.alpha
        self.beta = c.beta

        self.outMin = pOutMin  # Lower output limit
        self.outMax = pOutMax  # Upper output limit

        deadTimeSteps = int(c.Tu / self.guiParameter.SampleTimeMpc)  # Number of steps to simulate the dead time
        self.historicalControlActions = [(0, self.guiParameter.SampleTimeMpc)] * deadTimeSteps  # List of historical control actions [action, time]


    def GetOptimalControlAction(self, pSet, pAct, pTime, pLastControlAction):
        Tau = c.Tau  # in seconds
        Gain = c.Gain
        Tu = c.Tu  # in seconds

        predictionHorizon = self.guiParameter.PredictionHorizonMpc  # Number of steps to predict
        controlHorizon = self.guiParameter.ControlHorizonMpc  # Number of steps to control

        deadTimeSteps = int(c.Tu / self.guiParameter.SampleTimeMpc)  # Number of steps to simulate the dead time

        if len(self.historicalControlActions) < deadTimeSteps: # If the list is not filled yet
            self.historicalControlActions = [(0,
                                              self.guiParameter.SampleTimeMpc)] * deadTimeSteps  # List of historical control actions [action, time]

        if len(self.historicalControlActions) >= deadTimeSteps:
            for i in range(len(self.historicalControlActions) - deadTimeSteps):  # pop as much elements as necessary
                self.historicalControlActions.pop(0)
        self.historicalControlActions.append((pLastControlAction, pTime))

```

```
39          def objective(u):
40                  T = pAct  # initial temperature
41                  error = 0
42                  controlActionChange = 0
43                  # Simulate a delay of Tu
44                  for step in range(len(self.historicalControlActions)):
45                      historicalControlAction, time = self.
                            ↪ historicalControlActions[step]  # gets the
                            ↪ corresponding historical control action
46                      T = T + (1 / Tau) * (Gain * historicalControlAction
                            ↪  - T + 23) * time
47                      error += (T - pSet) ** 2
48
49                  for t in range(predictionHorizon):
50                      controlAction = u[min(t, controlHorizon - 1)]
51                      T = T + (1 / Tau) * (Gain * controlAction - T + 23)
                            ↪  * pTime
52                      error += (T - pSet) ** 2
53
54                      # Calculate the change of the control action
55                      if t == 0:
56                          controlActionChange += (u[min(t, controlHorizon
                                ↪  - 1)] - pLastControlAction) ** 2
57                      if t > 0:  # no change at the first step
58                          controlActionChange += (u[min(t, controlHorizon
                                ↪  - 1)] - u[min(t - 1, controlHorizon - 1)])
                                ↪  ** 2
59
60                  penalty = c.alpha * error + c.beta *
                            ↪ controlActionChange
61
62                  return penalty
63
64
65          initialGuess = [0] * controlHorizon  # initial guess for
                            ↪ control actions
66          bounds = [(self.outMin, self.outMax) for _ in range(
                            ↪ controlHorizon)]
67
68          print("Historical control actions: ")
69          print(self.historicalControlActions)
70
71          optimizingTime = time.time()
72          result = opt.minimize(objective, initialGuess, bounds=bounds)
73          optimizingTime = time.time() - optimizingTime
74
75
76          if result.success:
77              print("Optimal control action: ")
78              print(result)
79              print("Optimizing time: " + str(optimizingTime))
80              return result.x[0]
81          else:
82              raise ValueError("Could not find optimal control action")
83
84      def Run(self, pSet, pAct, pTime, pLastControlAction):
85          output = self.GetOptimalControlAction(pSet, pAct, pTime,
                            ↪ pLastControlAction)
86
87          # Limit output
88          if output > self.outMax:
89              output = self.outMax
90          elif output < self.outMin:
```

```
91              output = self.outMin
92          return output
93
94      def Reset(self):
95          print("Reset not implemented yet")
```

A.4.3 LSTM-Regler

Trainieren des Reglers

Das Training des LSTMs, welches das Regelverhalten eines PID-Reglers erlernen soll, erfolgt in einem Google Colab Notebook. Der folgende Code zeigt den Aufbau des LSTM-Modells und dessen Training. Die Datenvorverarbeitung sowie die Validierung und Bewertung des Modells sind nicht explizit aufgeführt.

```
1
2  # Keras LSTM model
3  model = Sequential()
4
5  model.add(Input(shape=(Xtrain.shape[1], Xtrain.shape[2])))
6
7  # First layer specifies input_shape and returns sequences
8  model.add(LSTM(units=5, return_sequences=True))
9  # Dropout layer to prevent overfitting
10 model.add(Dropout(rate=0.1))
11
12 # Last layer doesn't return sequences (middle layers should return
                              ↪ sequences)
13 model.add(LSTM(units=3))
14 model.add(Dropout(rate=0.1))
15
16 # Dense layer to return prediction
17 model.add(Dense(1))
18
19
20 # Compile model; adam optimizer, mse loss
21 adam = Adam(learning_rate=0.0001)
22 model.compile(optimizer='adam', loss='mean_squared_error')
23
24 es = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=25)
25
26 result = model.fit(Xtrain, ytrain, verbose=0, validation_split=0.2,
27                    callbacks = [TqdmCallback(verbose=1)],#es
28                    batch_size=50,
29                    epochs=300)
30
31 # Plot loss and save model
32 epochs = es.stopped_epoch
33 plt.semilogy(result.history['loss'],label='loss')
34 plt.semilogy(result.history['val_loss'],label='val_loss')
35 plt.legend()
36
37 model.save('/content/drive/MyDrive/LSTMPID/model.keras')
38
```

Anwenden des Reglers

Die Klasse „KiControllerLstm“ verwendet das vorab trainierte LSTM-Modell als Regler. Zu Beginn wird dieses Modell zusammen mit den erforderlichen Skalierern

einmalig geladen. Nach dem Start des Reglers wird die „Run“-Methode in regelmäßigen Abständen von 100 Millisekunden aufgerufen. In dieser Methode wird das LSTM genutzt, um mittels den historischen Ist- und Sollwerten, den anzuwendenden Stellwert zu bestimmen.

```
from general.GuiParameter import GuiParameter
from general.Constants import Constants as c
import numpy as np
import joblib

from keras.models import load_model

class KiControllerLstm:
    def __init__(self, pGuiParameter: GuiParameter, pOutMin=0, pOutMax=100):
        self.scalerX = joblib.load("C:\\Users\\ux\\PycharmProjects\\BA\\models\\scaler_x_real.joblib")
        self.scalerY = joblib.load("C:\\Users\\ux\\PycharmProjects\\BA\\models\\scaler_y_real.joblib")
        self.model = load_model("C:\\Users\\ux\\PycharmProjects\\BA\\models\\model_real.keras")

        self.guiParameter = pGuiParameter

        self.OutMin = pOutMin  # Lower output limit
        self.OutMax = pOutMax  # Upper output limit

        self.window = c.windowLstm
        self.ActValues = np.zeros(self.window)
        self.Setpoints = np.zeros(self.window)

        self.counter = 0


    def Run(self, pActValue, pSetpoint):

        # safe last x historical Values for LSTM input
        # take every 20th value  20 * 100 ms = 1 s
        if self.counter % 20 == 0:
            self.ActValues = np.roll(self.ActValues, -1)
            self.ActValues[-1] = pActValue
            self.Setpoints = np.roll(self.Setpoints, -1)
            self.Setpoints[-1] = pSetpoint
        self.counter += 1

        # Calculate error (necessary feature for LSTM input)
        err = self.Setpoints - self.ActValues

        # Format data for LSTM input
        #X = np.vstack((self.ActValues, self.Setpoints, err)).T
        X = np.vstack((self.Setpoints, err)).T
        Xs = self.scalerX.transform(X)
        Xs = np.reshape(Xs, (1, Xs.shape[0], Xs.shape[1]))

        # Predict Q for controller and unscale
        outputScaled = self.model.predict(Xs, verbose=0)
        output = self.scalerY.inverse_transform(outputScaled)[0][0]

         # Ensure output is between min and max
        if output > self.OutMax:
            output = self.OutMax
        elif output < self.OutMin:
```

```
            output = self.OutMin

        return output

    def Reset(self):
        self.counter = 0
        self.ActValues = np.zeros(self.window)
        self.Setpoints = np.zeros(self.window)
```

A.4.4 MPC-Regler mit FFNN als Vorhersagemodell

Hyperparameter finden mittels Bayesianischer Optimierung

Um das Design und die Hyperparameter für das Training des FFNNs optimal zu gestalten, wurde eine Bayesianische Optimierung durchgeführt. Diese wurde folgendermaßen umgesetzt.

```
def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units_0', min_value=2, max_value=30,
                        ↪ step=1),
                    activation=hp.Choice('activation_0', ['relu', 'tanh
                        ↪ ']),
                    input_shape=(inputShape,)))

    model.add(Dropout(rate=hp.Float('dropout_0', min_value=0.0,
                        ↪ max_value=0.5, step=0.1)))

    # Number of hidden layers
    #for i in range(hp.Int('num_layers', min_value=0, max_value=2, step
                        ↪ =1)):
    #    model.add(Dense(units=hp.Int('units_' + str(i+1), min_value=2,
                        ↪  max_value=30, step=1),
    #                    activation=hp.Choice('activation_' + str(i+1),
                        ↪  ['relu', 'tanh'])))
        # Dropout-Rate
    #    model.add(Dropout(rate=hp.Float('dropout_' + str(i+1),
                        ↪ min_value=0.0, max_value=0.5, step=0.1)))

    model.add(Dense(1, activation='linear'))
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Float('learning_rate', min_value=1e-4, max_value=1e-2,
                        ↪ sampling='log')),
        loss='mean_squared_error',
        metrics=['mean_squared_error'])
    return model

inputShape = Xtrain.shape[1]

tuner = BayesianOptimization(
    build_model,
    objective='val_mean_squared_error',
    max_trials=150,
    executions_per_trial=2,
    directory='my_dir',
    project_name='bayesian_optimization'
)

# Output summary
tuner.search_space_summary()

```

```
# Split training data again
XtrainSplit, Xval, ytrainSplit, yval = train_test_split(Xtrain, ytrain,
                          ↪  test_size=0.2, random_state=None)

# Starting the search
tuner.search(XtrainSplit, ytrainSplit, epochs=100, validation_data=(
                          ↪ Xval, yval), verbose=2)

# Getting the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]  #
                          ↪ num_trials = only the best

# Customise the output to include all hyperparameters
num_layers = 0 # best_hps.get('num_layers')

# Basis Hyperparameter
print(f"""
Basis Hyperparameter:
    - Units Schicht 0: {best_hps.get('units_0')}
    - Activation Schicht 0: {best_hps.get('activation_0')}
    - Dropout-Rate Schicht 0: {best_hps.get('dropout_0')}
    - Anzahl der Schichten: {num_layers}
    - Lernrate: {best_hps.get('learning_rate')}
""")

# Hyperparameters for each hidden layer
for i in range(num_layers):
    print(f"""
    Schicht {i+1} Hyperparameter:
        - Units: {best_hps.get('units_' + str(i+1))}
        - Activation: {best_hps.get('activation_' + str(i+1))}
        - Dropout-Rate: {best_hps.get('dropout_' + str(i+1))}
    """)
```

Trainieren des Netzwerkes

Mit den durch die Bayesianische Optimierung vorgeschlagenen Parametern wurde das FFNN trainiert. Die Datenvorverarbeitung sowie die Validierung und Bewertung des Modells sind dabei nicht explizit aufgeführt.

```
# Recreate model manually
# definition of hyperparameters
units_0 = 2
activation_0 = 'relu'
dropout_0 = 0.0
units_1 = 6
activation_1 = 'relu'
dropout_1 = 0.1
learning_rate = 0.01
input_shape = Xtrain.shape[1]  # Stellen Sie sicher, dass Xtrain
                          ↪ definiert ist

# Build model
modelManual = Sequential()
modelManual.add(Dense(units=units_0, activation=activation_0,
                          ↪ input_shape=(input_shape,)))
modelManual.add(Dropout(rate=dropout_0))

#modelManual.add(Dense(units=units_1, activation=activation_1))
#modelManual.add(Dropout(rate=dropout_1))

# Output layer
```

```
modelManual.add(Dense(1, activation='linear'))

# Compile model
modelManual.compile(optimizer=keras.optimizers.Adam(learning_rate=
                            ↪ learning_rate), loss='mean_squared_error',
                            ↪ metrics=['mean_squared_error'])

result = modelManual.fit(Xtrain, ytrain, validation_data=(Xtest, ytest)
                            ↪ , epochs=300)

model = modelManual

# Plot loss
#epochs = es.stopped_epoch
plt.semilogy(result.history['loss'],label='loss')
plt.semilogy(result.history['val_loss'],label='val_loss')
plt.legend()

modelpath = '/content/drive/MyDrive/FFNN/modelReal.keras'
model.save(modelpath)
```

Implementierung des Reglers

Das zuvor trainierte FFNN wird in der Klasse „DnMpcController“ verwendet, um zusammen mit dem MPC-Algorithmus einen Regler zu realisieren. Ähnlich wie beim konventionellen MPC-Regler prognostiziert die „objective“-Methode die Kosten verschiedener Stellwerte über den Vorhersagehorizont hinweg. Der Unterschied besteht darin, dass diesmal die Vorhersagen des FFNNs anstelle des physikalischen PT1-Modells genutzt werden. Diese Methode wird durch den Optimierer „scipy. optimize.minimize“ angewendet, um die optimalen Stellwerte zu ermitteln. Die „Run“-Methode, welche den gesamten Prozess steuert, wird zyklisch alle 5 Sekunden aufgerufen.

```
import scipy.optimize as opt
from general import GuiParameter
from general.Constants import Constants as c
from keras.models import load_model
import numpy as np
import time


class DnMpcController:
    def __init__(self, pGuiParameter: GuiParameter, pOutMin=0, pOutMax
                            ↪ =100):
        self.guiParameter = pGuiParameter
        self.alpha = c.alpha
        self.beta = c.beta

        self.outMin = pOutMin  # Lower output limit
        self.outMax = pOutMax  # Upper output limit

        # historical control actions and actual values
        self.historicalControlActions = [0] * c.
                            ↪ dnMpcNumberDelayedControlValues  # List of
                            ↪ historical control actions [action, time]
        self.historicalActualValues = None  # List of historical actual
                            ↪  values
        self.lastActualValue = 0

        self.ControlHorizon = self.guiParameter.ControlHorizonDnMpc
```

```
24          self.initialGuess = [0] * self.ControlHorizon
25
26          # Load Model
27          self.model = load_model("C:\\Users\\ux\\PycharmProjects\\BA\\
                        ↪ models\\DnMpc\\modelReal.keras")
28          #self.model = load_model("C:\\Users\\ux\\PycharmProjects\\BA\\
                        ↪ models\\DnMpc\\model.keras")
29
30          # extract weights and biases for the first dense layer
31          self.weights0, self.biases0 = self.model.layers[0].get_weights
                        ↪ ()
32          # extract weights and biases for the output layer
33          self.weights1, self.biases1 = self.model.layers[-1].get_weights
                        ↪ ()
34
35      # since all control values are in the range 0 -100 % and most
                        ↪ actual values are also between 0 - 100,
                        ↪ scaling can be done manually
36      def ScaleMinMax(self, pValue, pMin=0, pMax=100):
37          return (pValue - pMin) / (pMax - pMin)
38      def InverseScaleMinMax(self, pScaledValue, pMin=0, pMax=100):
39          return pScaledValue * (pMax - pMin) + pMin
40
41      def predictWithNumpy(self, pInputData):
42          def relu(x):
43              return np.maximum(0, x)
44
45          # Processing through the first dense layer
46          layer0Output = relu(np.dot(pInputData, self.weights0) + self.
                        ↪ biases0)
47          # If there would be a second layer:
48          # layer_1_output = relu(np.dot(layer_0_output, weights_1) +
                        ↪ biases_1)
49
50          # Output layer (linear activation)
51          finalOutput = np.dot(layer0Output, self.weights1) + self.
                        ↪ biases1
52          return finalOutput
53
54
55      def GetOptimalControlAction(self, pSet, pAct, pTime,
                        ↪ pLastControlAction):
56          # Update historical values
57          if self.historicalActualValues is None:
58              self.historicalActualValues = [pAct] * c.
                        ↪ dnMpcNumberDelayedActualValues
59
60          self.lastActualValue = pAct
61          self.historicalActualValues.append(self.lastActualValue)
62          self.historicalControlActions.append(self.lastControlAction)
63          if len(self.historicalActualValues) > c.
                        ↪ dnMpcNumberDelayedActualValues:
64              self.historicalActualValues.pop(0)
65          if len(self.historicalControlActions) > c.
                        ↪ dnMpcNumberDelayedControlValues:
66              self.historicalControlActions.pop(0)
67
68          predictionHorizon = self.guiParameter.PredictionHorizonDnMpc
69          controlHorizon = self.guiParameter.ControlHorizonDnMpc
70
71
72          # Objective function to minimize
73          def objective(controlSequence):
```

```
74              # Initialisiere den Fehlerwert
75              error = 0
76              controlActionChange = 0
77
78              # inizialize the predicted actual values with the
                    ↪ historical actual values
79              predictedActuals = self.historicalActualValues.copy()
80              # Update the last known actual value for the first
                    ↪ prediction
81              lastActualValue = self.lastActualValue
82
83
84              for i in range(predictionHorizon):
85                  # Create the input vector for the neural network
86                  # For the first step, use the last known values, then
                        ↪ the predicted values
87
88                  if i == 0:
89                      pastActuals = self.historicalActualValues[-c.
                            ↪ dnMpcNumberDelayedActualValues:]
90                      pastControls = self.historicalControlActions[-c.
                            ↪ dnMpcNumberDelayedControlValues:]
91                  else:
92                      pastActuals = predictedActuals[-c.
                            ↪ dnMpcNumberDelayedActualValues:]
93
94                      # Extract the necessary historical control actions
95                      historicalControlActionsNp = np.array(
96                          self.historicalControlActions[-(c.
                                ↪ dnMpcNumberDelayedControlValues - i):]
97                      )
98
99                      # Control sequence up to the current step i
100                     controlSequenceCurrentStep = controlSequence[:min(i
                            ↪ , len(controlSequence))]
101
102                     # controlHorizon:
103                     # If i is greater than the length of the
                            ↪ controlSequence, fill with the last value
                            ↪ of the controlSequence
104                     if i > len(controlSequence):
105                         lastValue = controlSequence[-1]
106                         fillValues = np.full((i - len(controlSequence))
                                ↪ , lastValue)
107                         controlSequenceFilled = np.concatenate([
                                ↪ controlSequenceCurrentStep, fillValues])
108                     else:
109                         controlSequenceFilled =
                                ↪ controlSequenceCurrentStep
110
111                     # Combine historical control actions and filled
                            ↪ control sequence
112                     if historicalControlActionsNp.size > 0:
113                         pastControls = np.concatenate([
                                ↪ historicalControlActionsNp,
                                ↪ controlSequenceFilled])
114                     else:
115                         pastControls = controlSequenceFilled
116
117
118                 currentControlAction = controlSequence[min(i, len(
                        ↪ controlSequence) - 1)]
119
```

```
120                 # Skalieren der Ist- und Stellwerte
121                 scaledPastActuals = self.ScaleMinMax(np.array(
                            ↪ pastActuals).reshape(-1, 1)).flatten()
122                 scaledPastControls = self.ScaleMinMax(np.array(
                            ↪ pastControls).reshape(-1, 1)).flatten()
123                 scaledLastActualValue = self.ScaleMinMax(np.array([
                            ↪ lastActualValue]).reshape(-1, 1)).flatten()
124                 scaledCurrentControlAction = self.ScaleMinMax(np.array
                            ↪ ([currentControlAction]).reshape(-1, 1)).
                            ↪ flatten()
125
126                 # Creating the scaled input vector for the neural
                            ↪ network
127                 inputVector = np.concatenate((scaledPastActuals,
                            ↪ scaledPastControls, scaledLastActualValue,
                            ↪ scaledCurrentControlAction))
128
129                 # Make the prediction with the neural network
130                 start_time = time.time()
131                 #predictedValueScaled = self.model.predict(inputVector.
                            ↪ reshape(1, -1), verbose=0)
132                 predictedValueScaled = self.predictWithNumpy(
                            ↪ inputVector).reshape(1, -1) # this is much
                            ↪ faster than the keras predict function
133                 #print(predictedValueScaled)
134                 predictedValue = self.InverseScaleMinMax(
                            ↪ predictedValueScaled)[0][0]
135                 end_time = time.time()
136                 #print("Prediction time: ", end_time - start_time)
137
138                 # Update the error based on the difference between the
                            ↪ prediction and the target value
139                 error += (predictedValue - pSet) ** 2
140                 # Update the list of predicted actual values
141                 predictedActuals.append(lastActualValue)
142                 lastActualValue = predictedValue
143
144                 # Calculate the change in the control action
145                 controlActionChange += (currentControlAction -
                            ↪ pastControls[-1]) ** 2
146
147
148             # Total penalty based on the error and the changes to the
                            ↪ control actions
149             penalty = self.alpha * error + self.beta *
                            ↪ controlActionChange
150
151             return penalty
152
153         bounds = [(self.outMin, self.outMax) for _ in range(
                            ↪ controlHorizon)]
154
155         # Run optimization
156         start_time = time.time()
157         result = opt.minimize(objective, self.initialGuess, bounds=
                            ↪ bounds)
158         end_time = time.time()
159         print("Optimization time: ", end_time - start_time)
160
161         if result.success:
162             optimalControlSequence = result.x  # Return entire sequence
163             self.initialGuess = np.roll(optimalControlSequence, -1)
```

```
164             self.initialGuess[-1] = self.initialGuess[-2]  # Duplicate
                    ↪ the second last value
165             optimalControlAction = optimalControlSequence[0]
166             print(result)
167         else:
168             print("Optimization failed")
169             optimalControlAction = pLastControlAction  # fallback to
                    ↪ last control action
170         return optimalControlAction
171
172
173
174
175     def Run(self, pSet, pAct, pTime, pLastControlAction):
176         if self.ControlHorizon != self.guiParameter.ControlHorizonDnMpc
                    ↪ :
177             self.ControlHorizon = self.guiParameter.ControlHorizonDnMpc
178             self.initialGuess = [0] * self.ControlHorizon
179
180         self.lastActualValue = pAct
181         self.lastControlAction = pLastControlAction
182
183         output = self.GetOptimalControlAction(pSet, pAct, pTime,
                    ↪ pLastControlAction)
184
185         self.historicalActualValues.pop(0)
186         self.historicalActualValues.append(pAct)
187         self.historicalControlActions.pop(0)
188         self.historicalControlActions.append(pLastControlAction)
189
190
191         # Limit output
192         if output > self.outMax:
193             output = self.outMax
194         elif output < self.outMin:
195             output = self.outMin
196         return output
197
198     def Reset(self):
199         self.historicalControlActions = [0] * c.
                    ↪ dnMpcNumberDelayedControlValues  # List of
                    ↪ historical control actions [action, time]
200         self.historicalActualValues = None
201         self.lastActualValue = 0
202
203         self.initialGuess = [0] * self.ControlHorizon
204
205
```

Anhang B

Messergebnisse

B.1 P-Regler (Sollwert 60 °C)

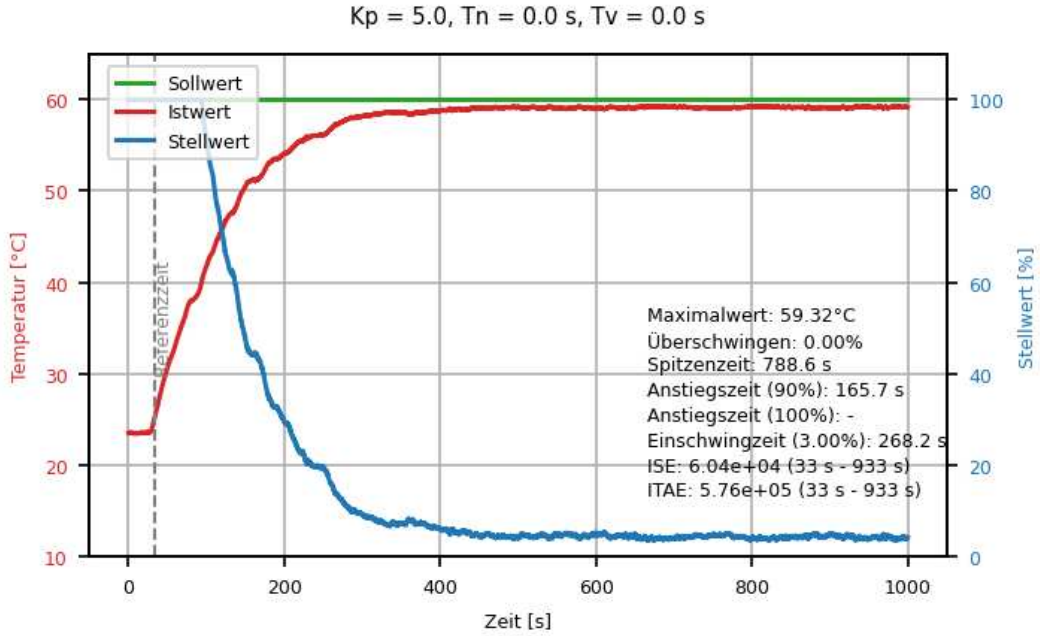


ABBILDUNG B.1: PID-Regler; Sollwert 60 °C
$K_P = 5,0; T_n = 0; T_v = 0$

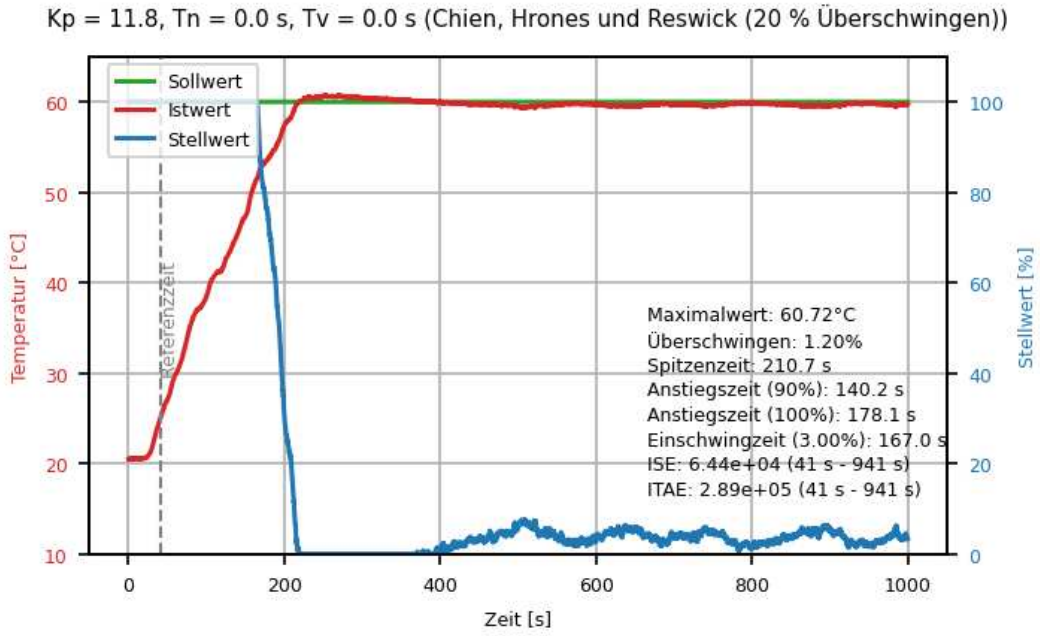


ABBILDUNG B.2: PID-Regler; Sollwert 60 °C
$K_P = 11,8; T_n = 0; T_v = 0$

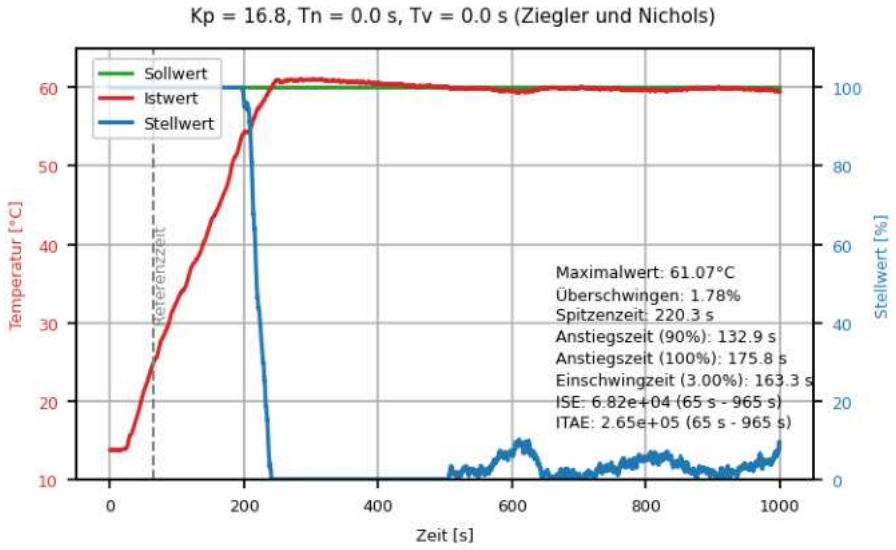


ABBILDUNG B.3: PID-Regler; Sollwert 60 °C
$K_P = 16,8; T_n = 0; T_v = 0$

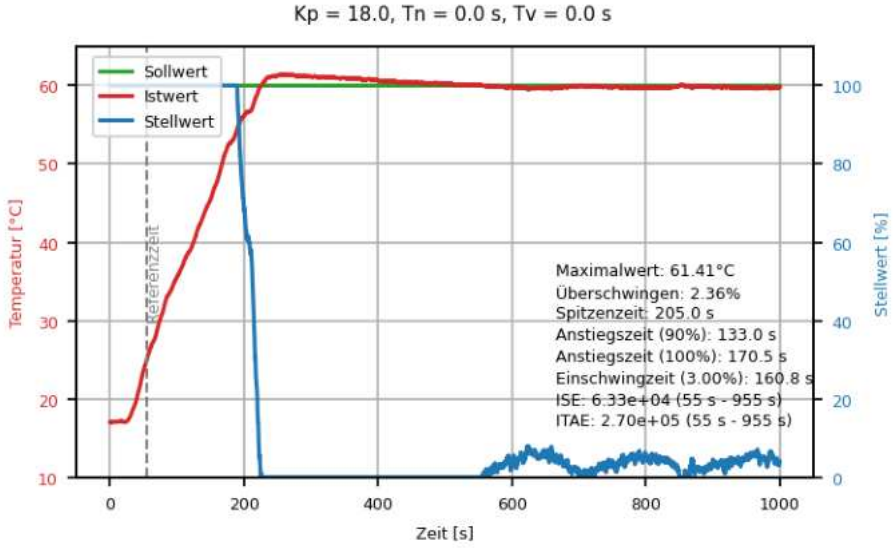


ABBILDUNG B.4: PID-Regler; Sollwert 60 °C
$K_P = 18,0; T_n = 0; T_v = 0$

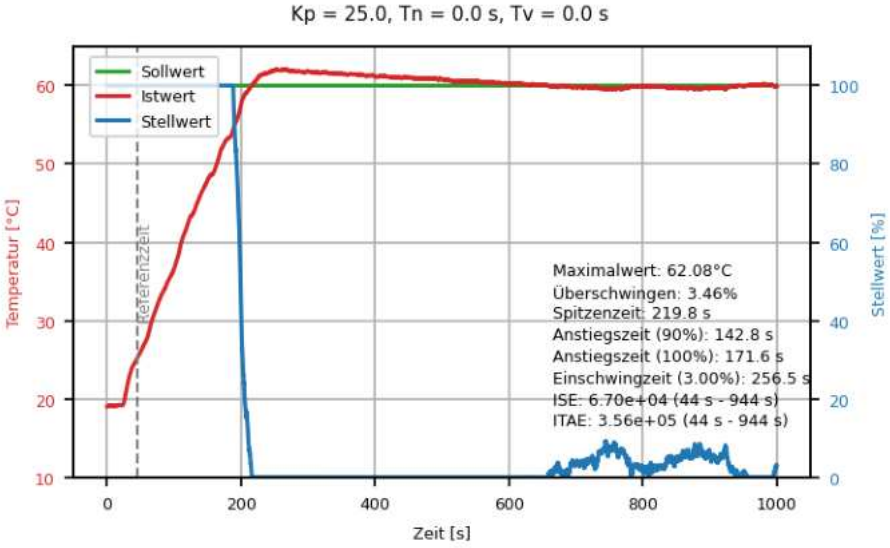


ABBILDUNG B.5: PID-Regler; Sollwert 60 °C
$K_P = 25,0; T_n = 0; T_v = 0$

B.2 PI-Regler (Sollwert 60 °C)

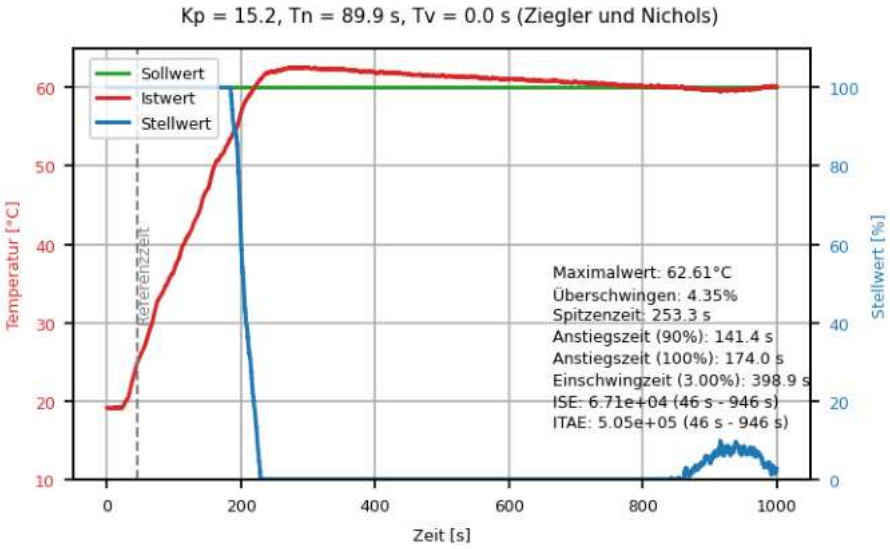


ABBILDUNG B.6: PID-Regler; Sollwert 60 °C
$K_P = 15,2; T_n = 89,9; T_v = 0$

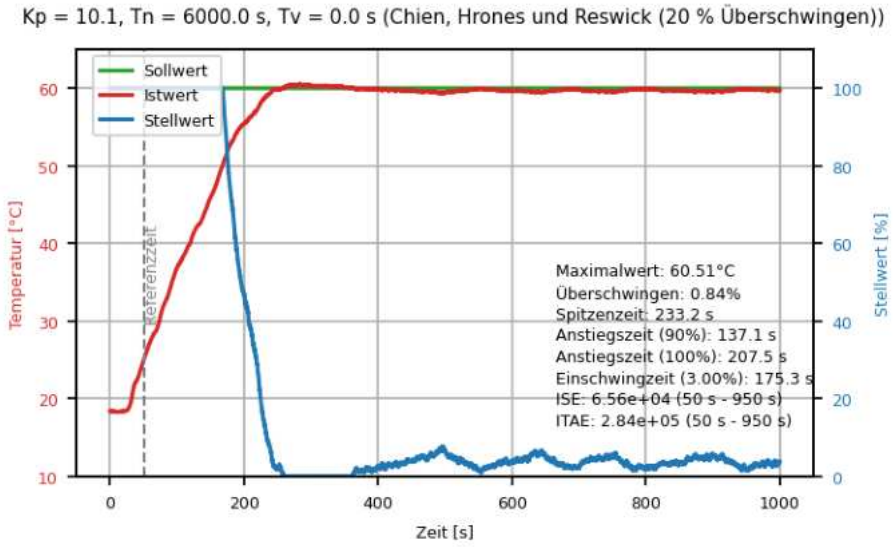


ABBILDUNG B.7: PID-Regler; Sollwert 60 °C
$K_P = 10,1; T_n = 6000,0; T_v = 0$

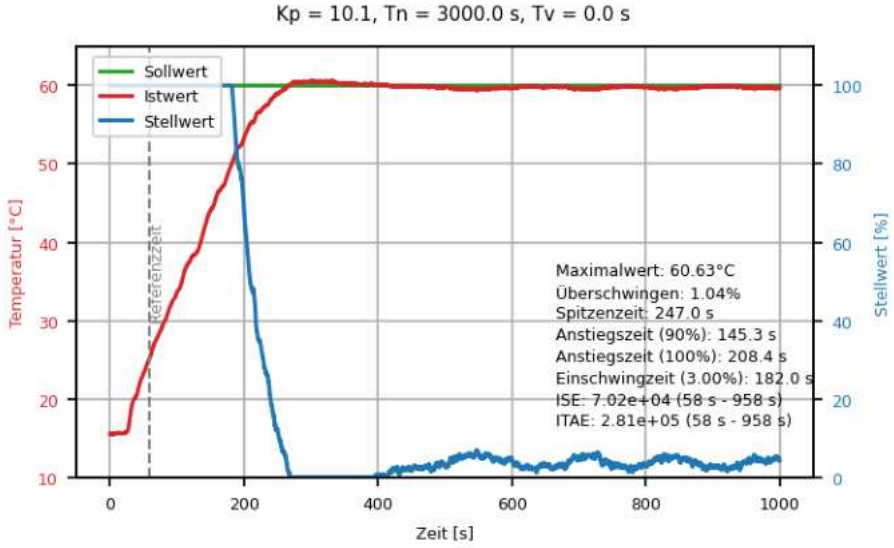


ABBILDUNG B.8: PID-Regler; Sollwert 60 °C
$K_P = 10,1; T_n = 3000,0; T_v = 0$

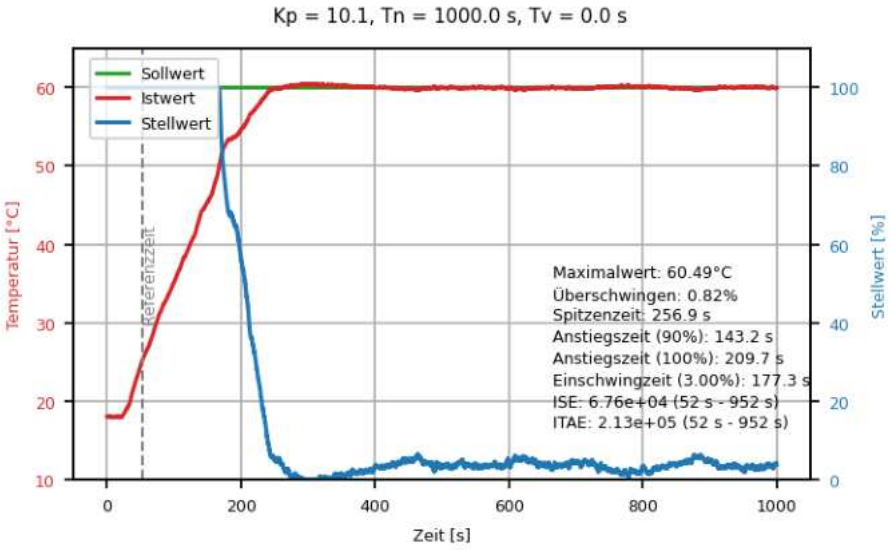


ABBILDUNG B.9: PID-Regler; Sollwert 60 °C
$K_P = 10,1; T_n = 1000,0; T_v = 0$

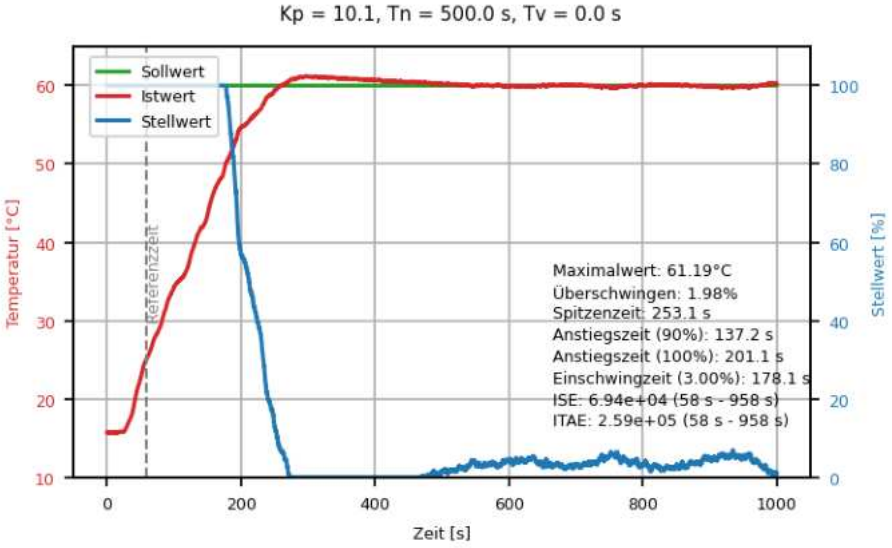


ABBILDUNG B.10: PID-Regler; Sollwert 60 °C
$K_P = 10,1; T_n = 500,0; T_v = 0$

B.3 PID-Regler (Sollwert 60 °C)

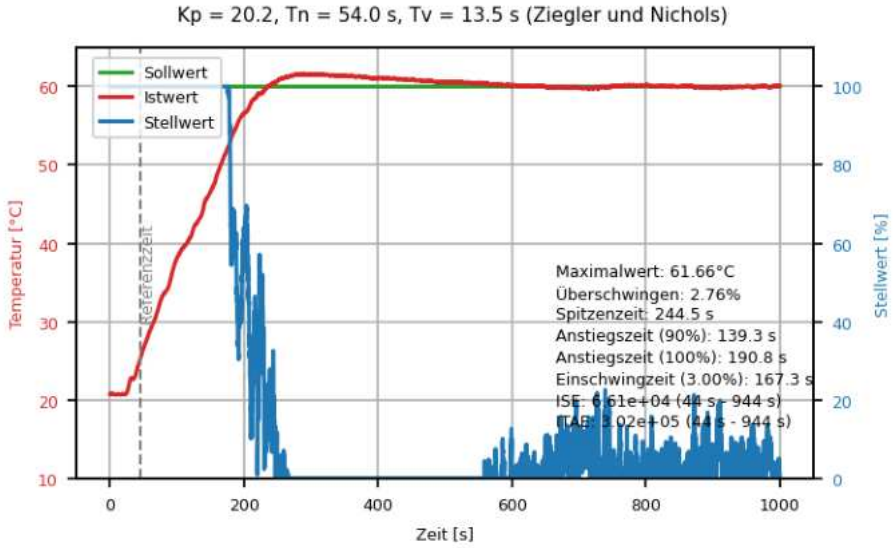


ABBILDUNG B.11: PID-Regler; Sollwert 60 °C
$K_P = 22,2; T_n = 54,0; T_v = 13,5$

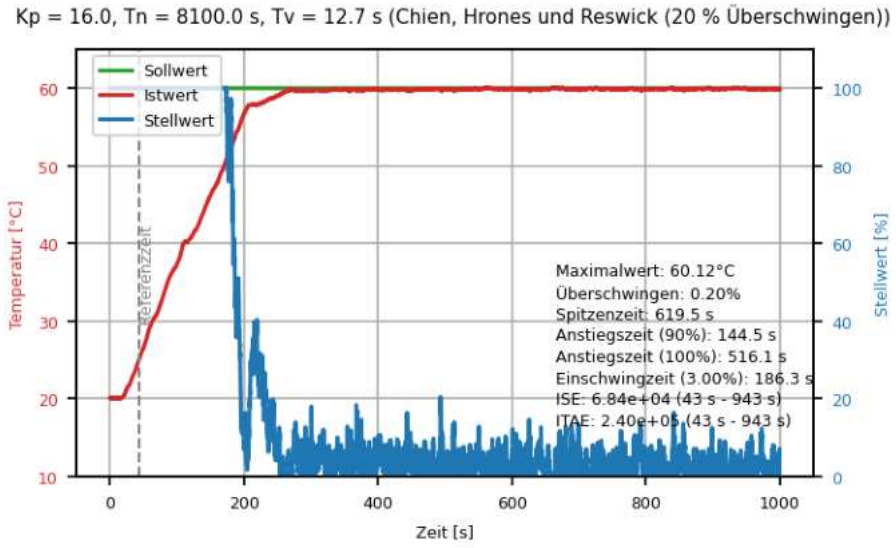


ABBILDUNG B.12: PID-Regler; Sollwert 60 °C
$K_P = 16,0; T_n = 8100,0; T_v = 12,7$

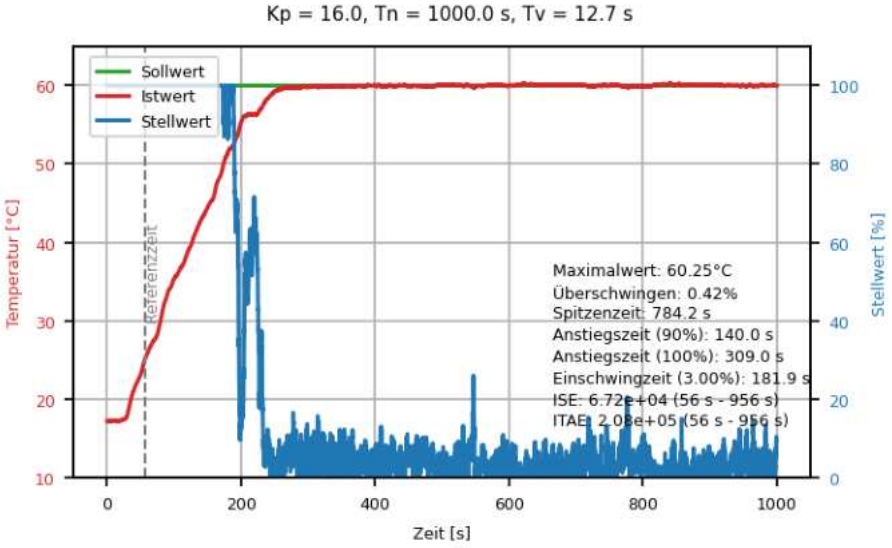


ABBILDUNG B.13: PID-Regler; Sollwert 60 °C
$K_P = 16,0; T_n = 1000,0; T_v = 12,7$

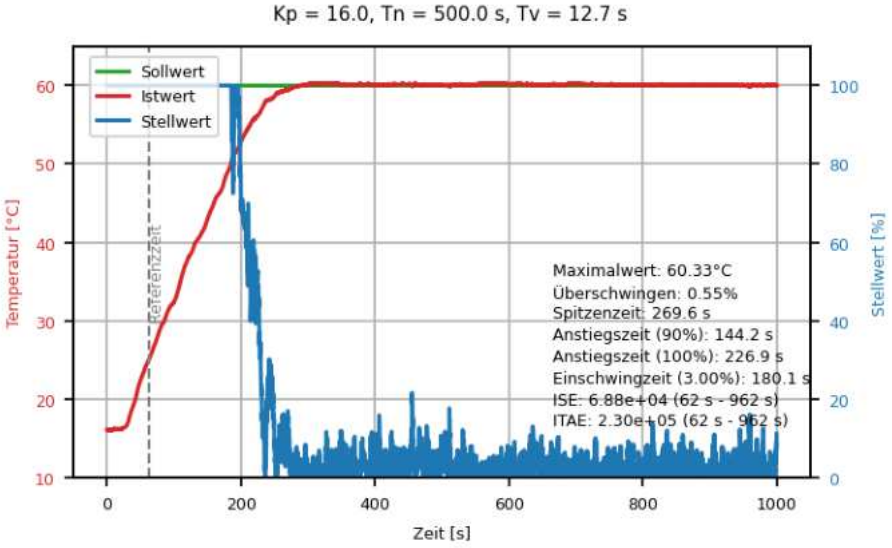


ABBILDUNG B.14: PID-Regler; Sollwert 60 °C
$K_P = 16,0; T_n = 500,0; T_v = 12,7$

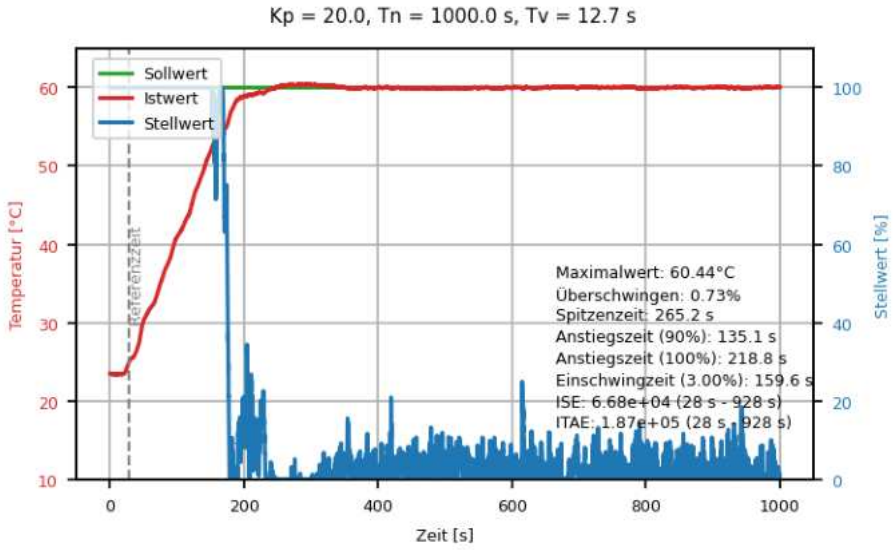


ABBILDUNG B.15: PID-Regler; Sollwert 60 °C
$K_P = 20,0; T_n = 1000,0; T_v = 12,7$

B.4 PID-Regler (Sollwert 30 °C)

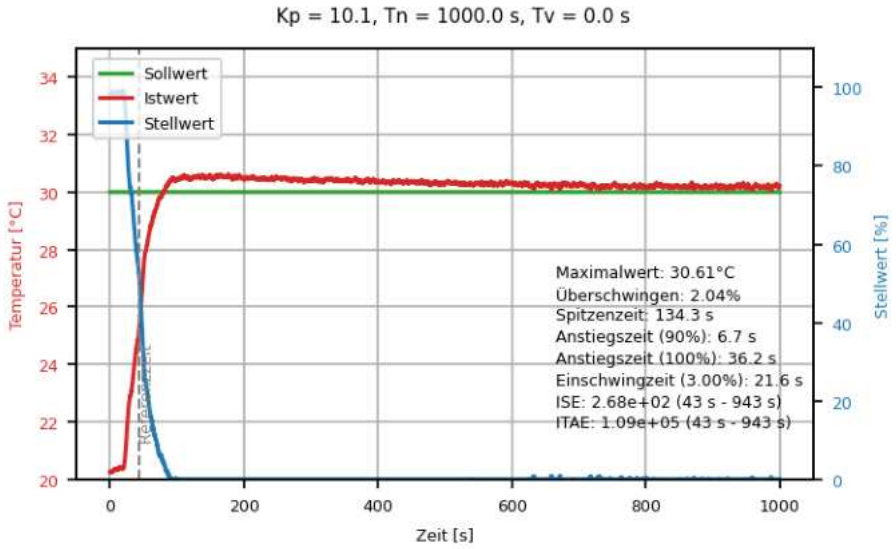


ABBILDUNG B.16: PID-Regler; Sollwert 30 °C
$K_P = 10,1; T_n = 1000,0; T_v = 0,0$

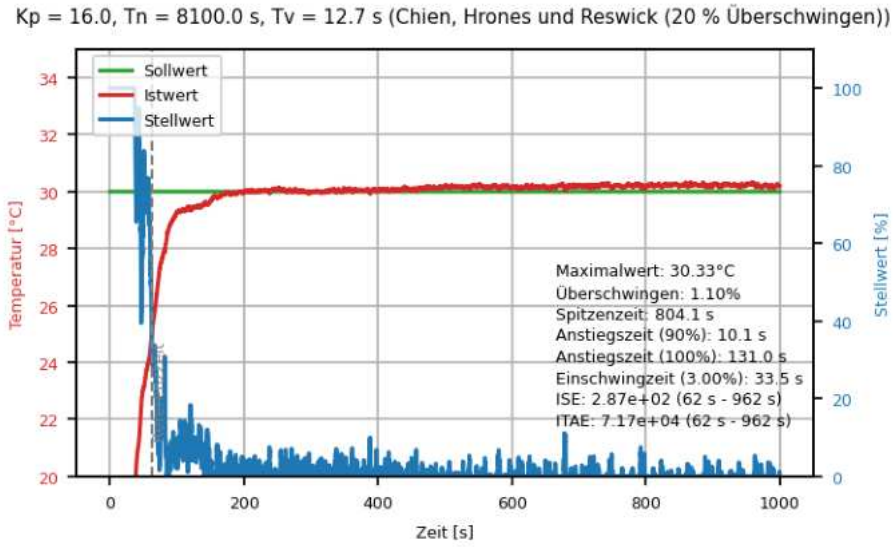


ABBILDUNG B.17: PID-Regler; Sollwert 60 °C
$K_P = 16,0; T_n = 8100,0; T_v = 12,7$

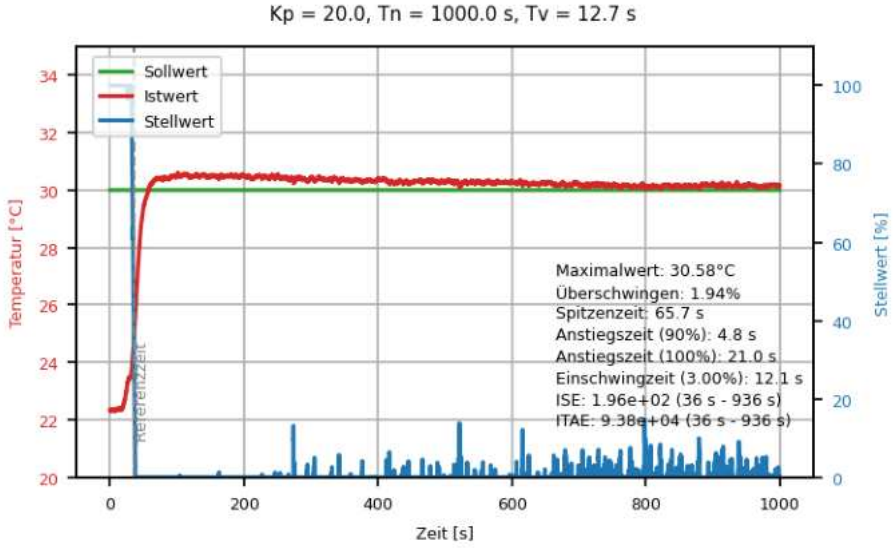


ABBILDUNG B.18: PID-Regler; Sollwert 60 °C
$K_P = 20,0; T_n = 1000,0; T_v = 12,7$

B.5 Testdurchlauf PID-Regler

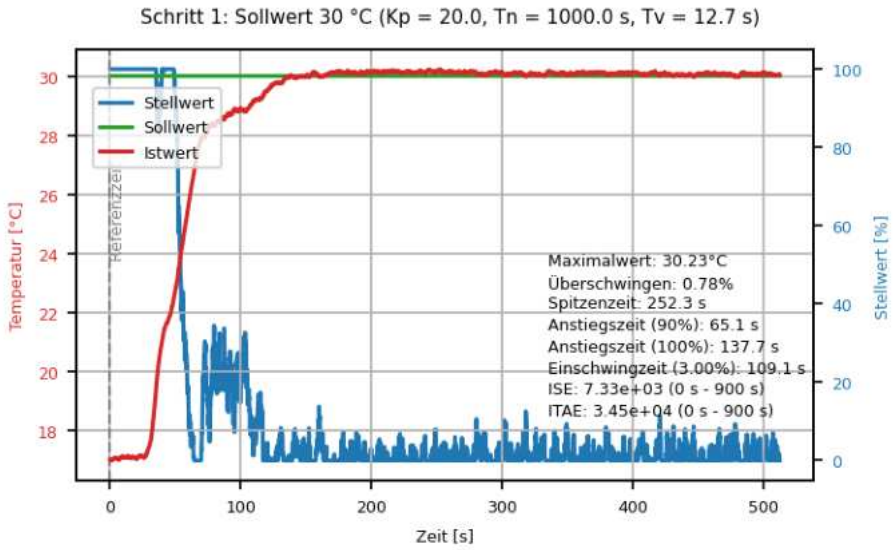


ABBILDUNG B.19: PID-Regler Testdurchlauf Schritt 1

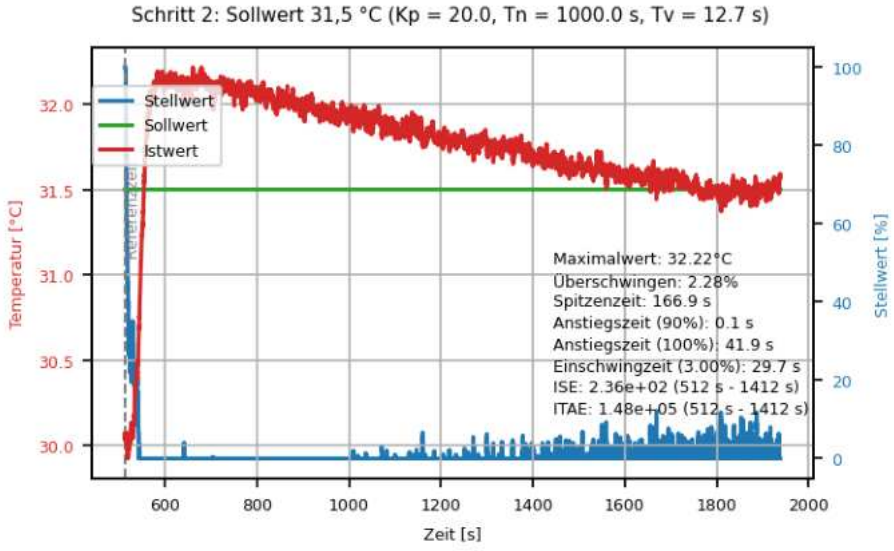


ABBILDUNG B.20: PID-Regler Testdurchlauf Schritt 2

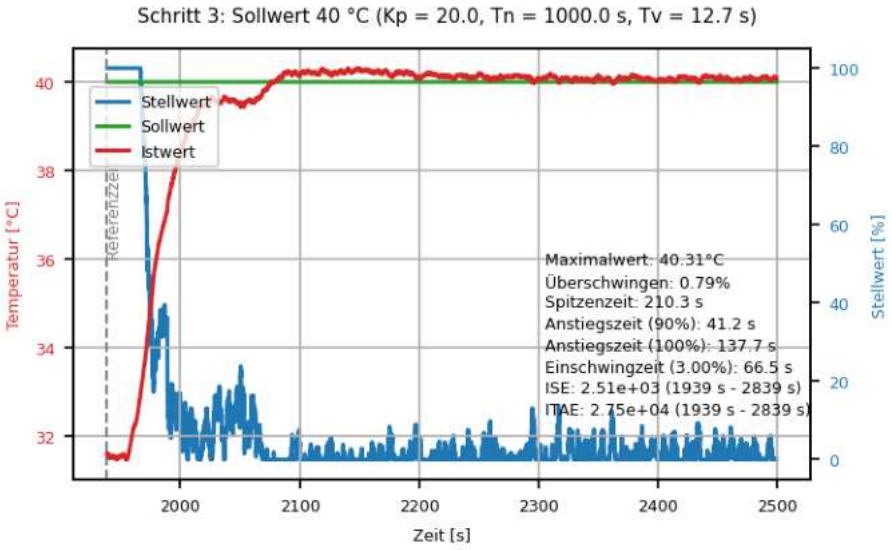


ABBILDUNG B.21: PID-Regler Testdurchlauf Schritt 3

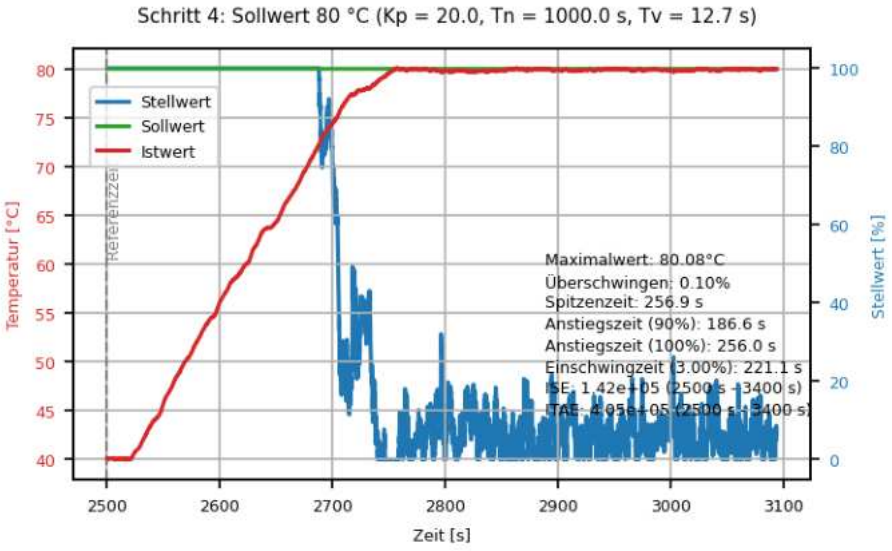


ABBILDUNG B.22: PID-Regler Testdurchlauf Schritt 4

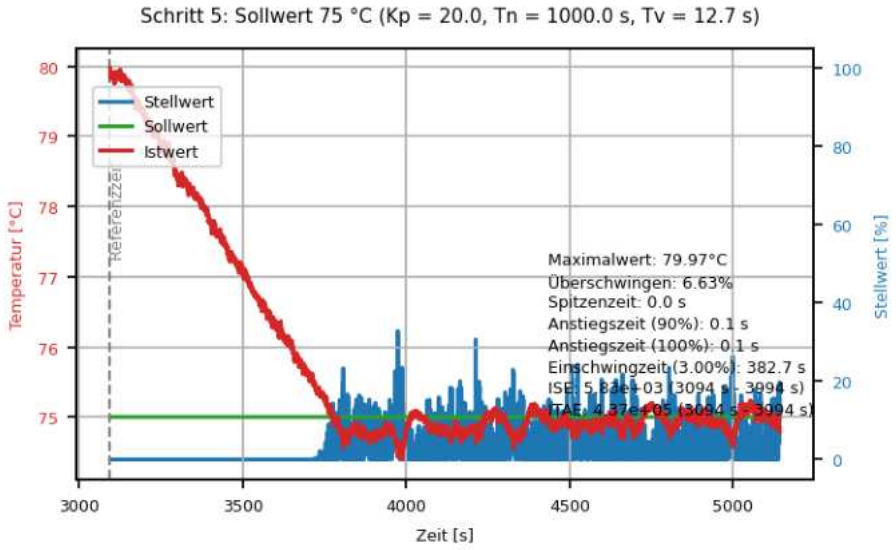


ABBILDUNG B.23: PID-Regler Testdurchlauf Schritt 5

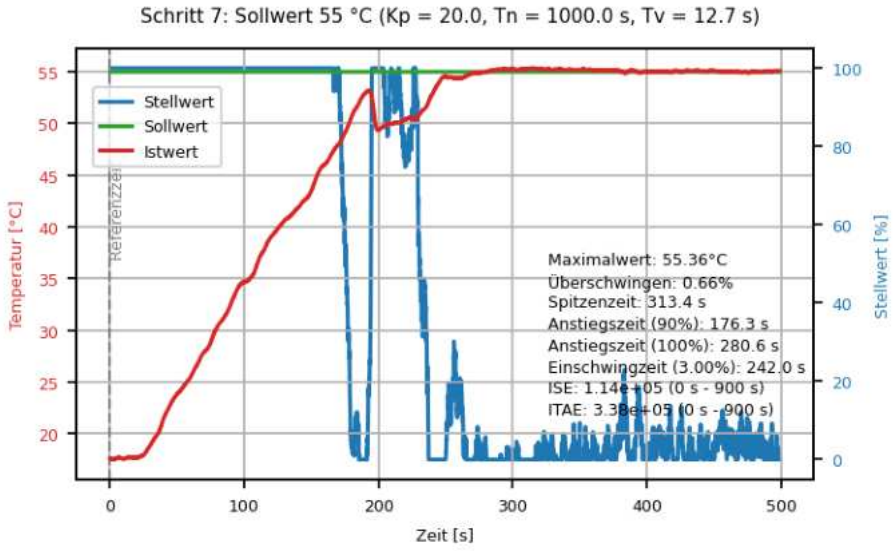


ABBILDUNG B.24: PID-Regler Testdurchlauf Schritt 7

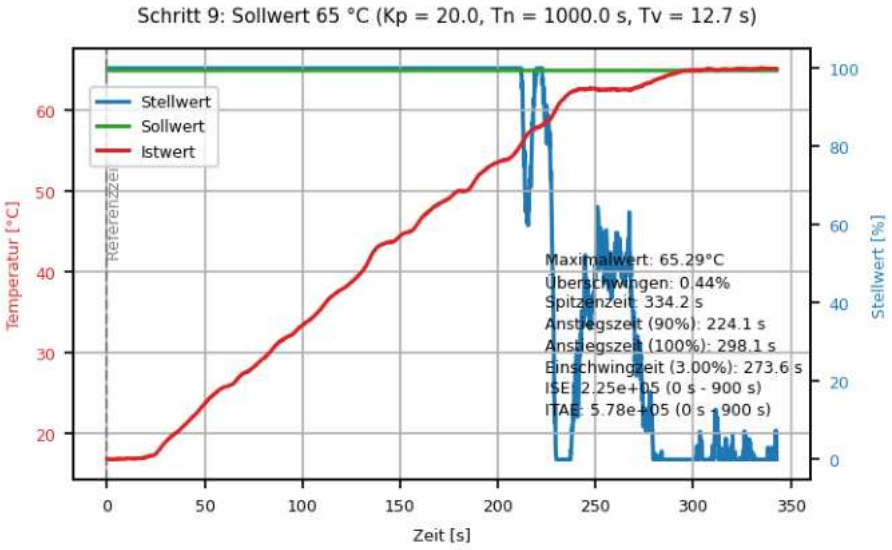


ABBILDUNG B.25: PID-Regler Testdurchlauf Schritt 9

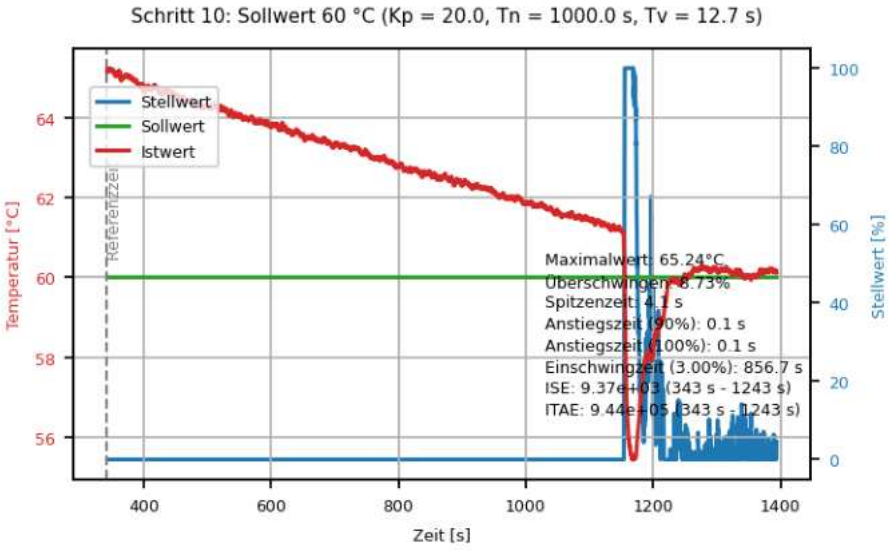


ABBILDUNG B.26: PID-Regler Testdurchlauf Schritt 10

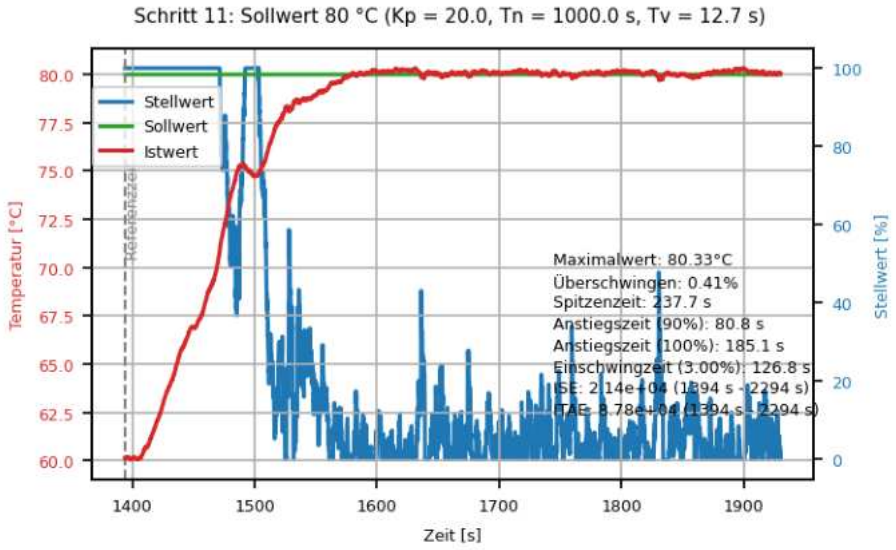


ABBILDUNG B.27: PID-Regler Testdurchlauf Schritt 11

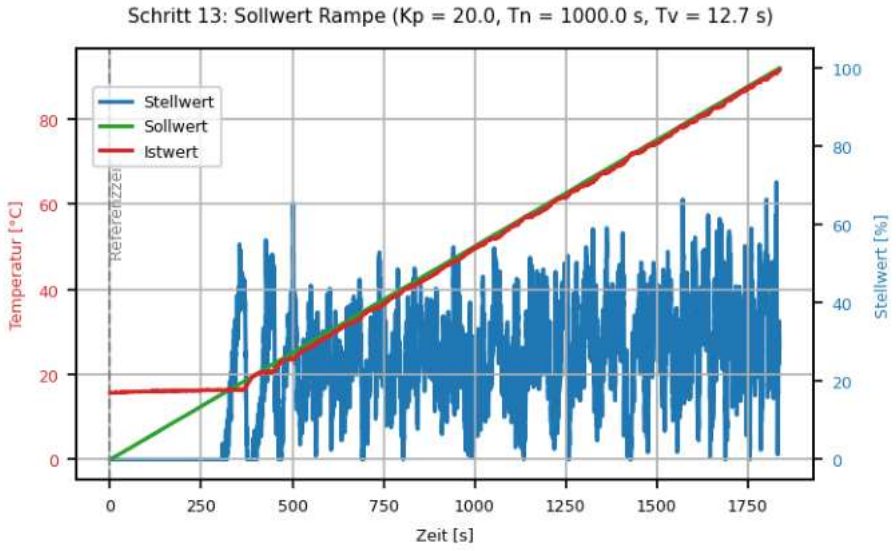


ABBILDUNG B.28: PID-Regler Testdurchlauf Schritt 13

B.6 Testdurchlauf MPC-Regler

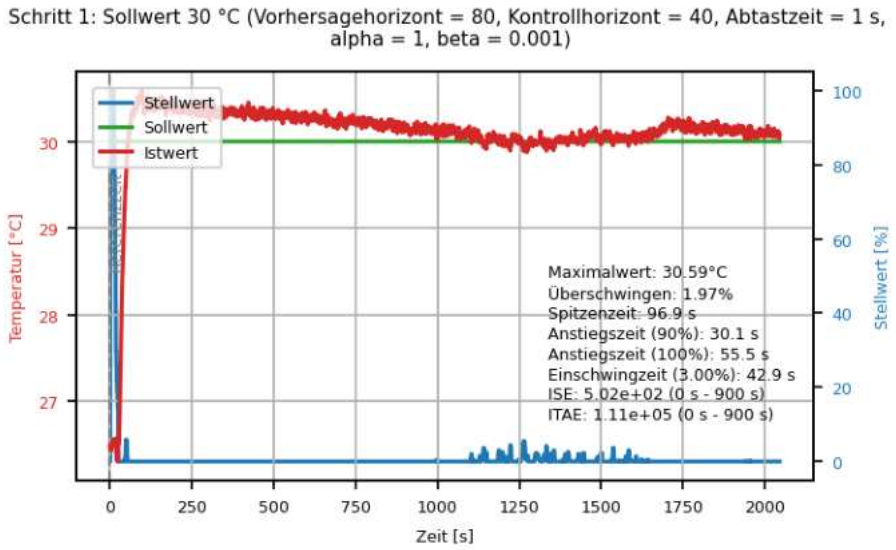


ABBILDUNG B.29: MPC-Regler Testdurchlauf Schritt 1

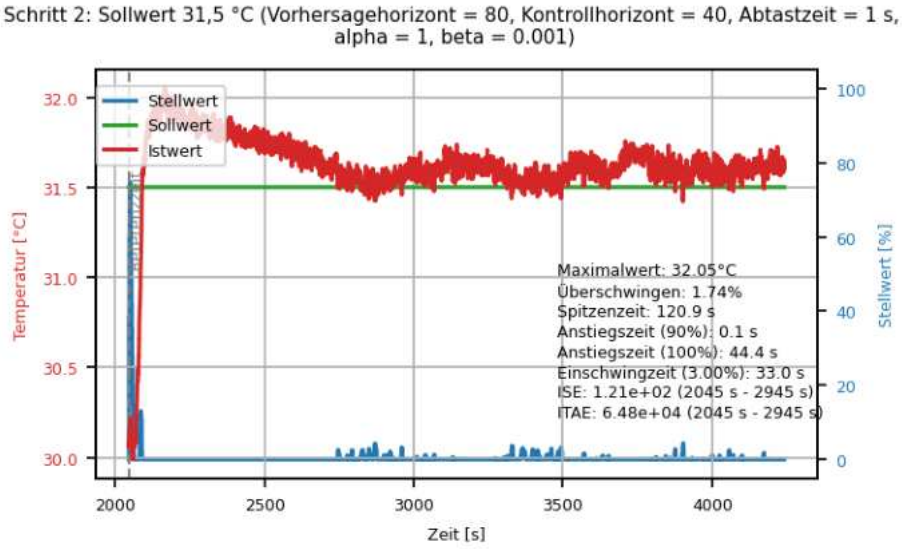


ABBILDUNG B.30: MPC-Regler Testdurchlauf Schritt 2

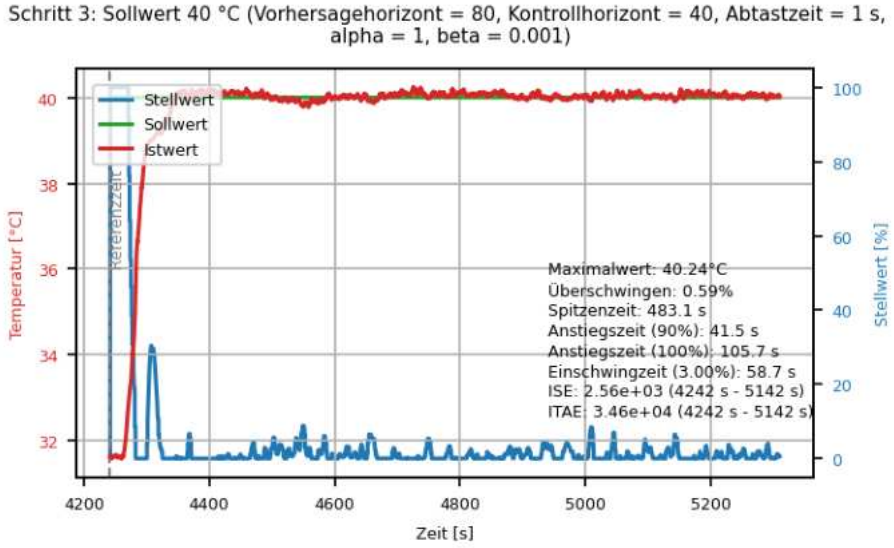


ABBILDUNG B.31: MPC-Regler Testdurchlauf Schritt 3

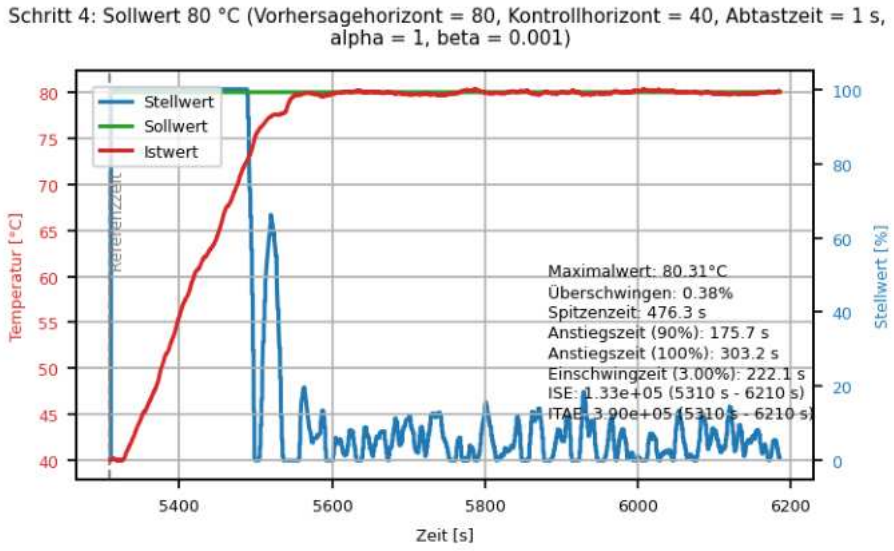


ABBILDUNG B.32: MPC-Regler Testdurchlauf Schritt 4

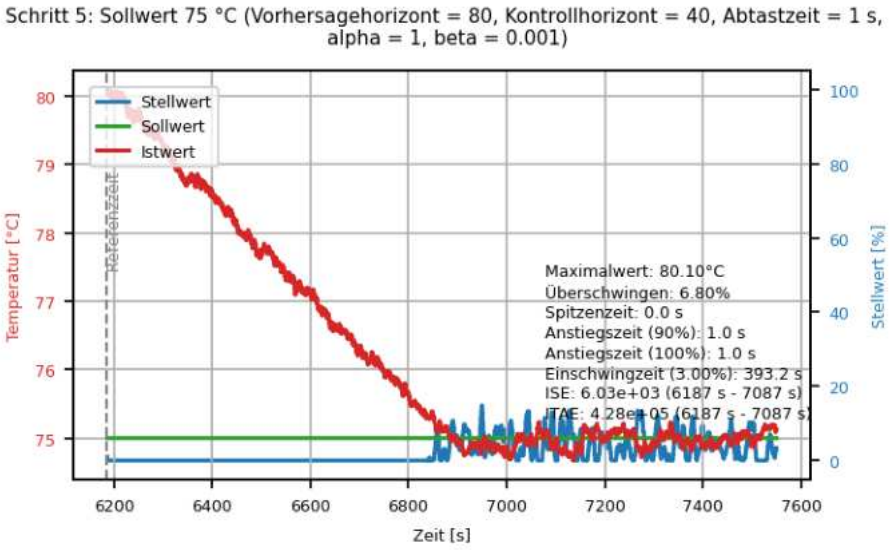


ABBILDUNG B.33: MPC-Regler Testdurchlauf Schritt 5

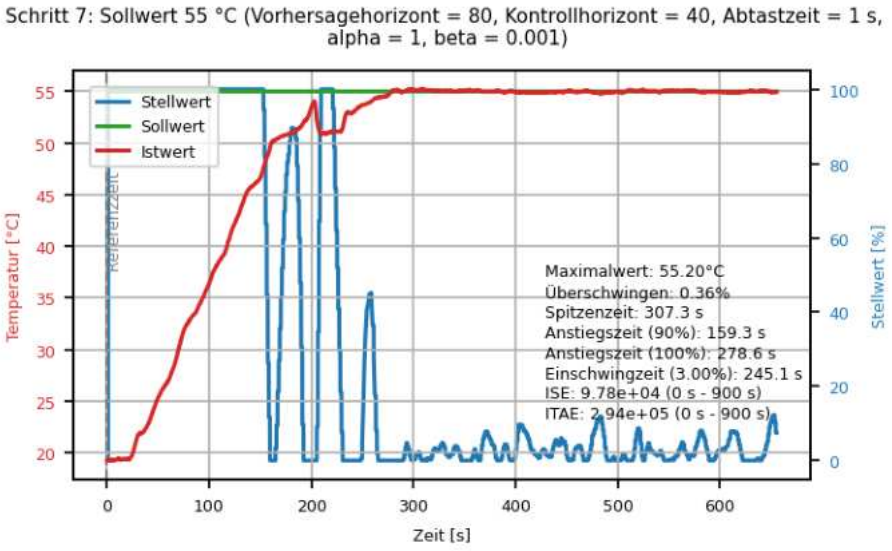


ABBILDUNG B.34: MPC-Regler Testdurchlauf Schritt 7

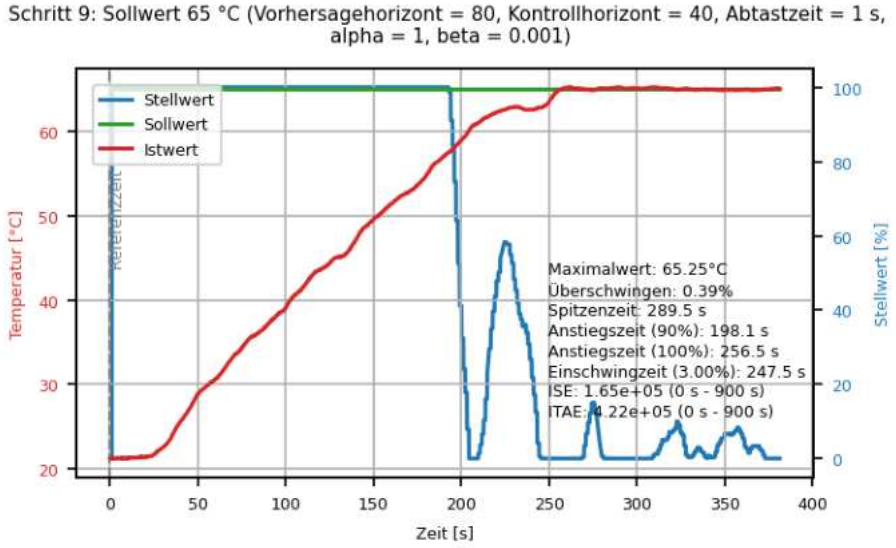


ABBILDUNG B.35: MPC-Regler Testdurchlauf Schritt 9

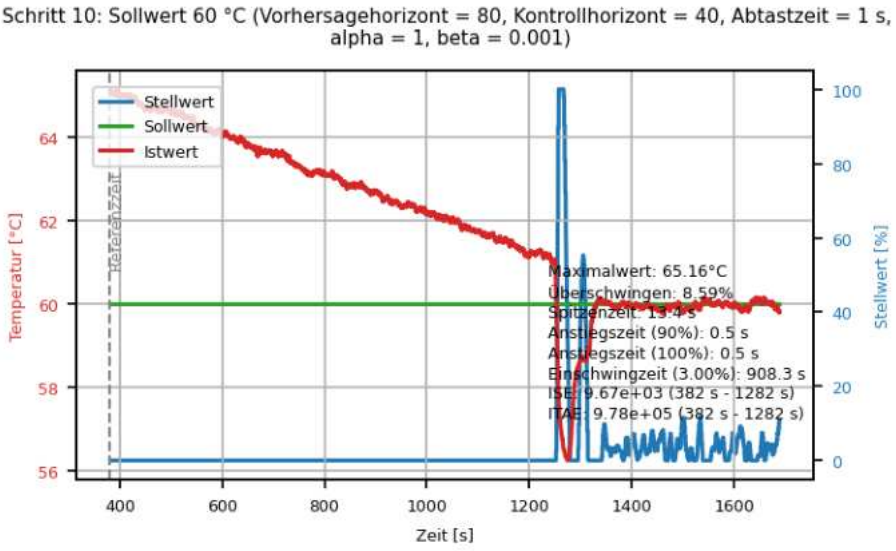


ABBILDUNG B.36: MPC-Regler Testdurchlauf Schritt 10

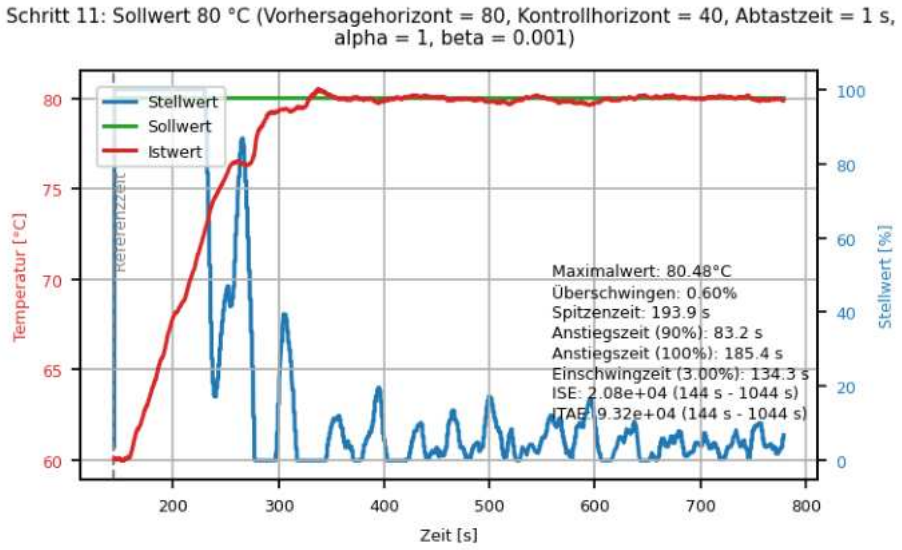


ABBILDUNG B.37: MPC-Regler Testdurchlauf Schritt 11

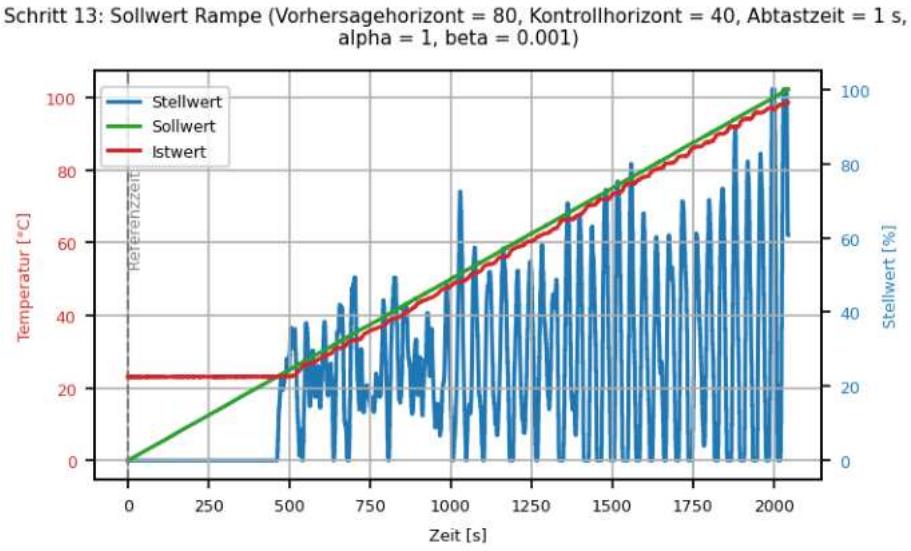


ABBILDUNG B.38: MPC-Regler Testdurchlauf Schritt 13

B.7 Testdurchlauf LSTM-Regler (Trainiert mit realen Daten)

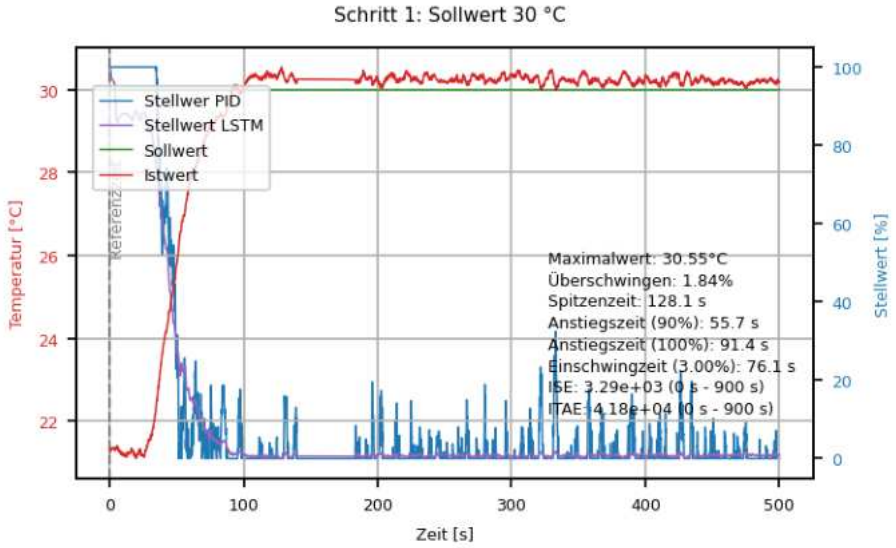


ABBILDUNG B.39: LSTM-Regler Testdurchlauf Schritt 1

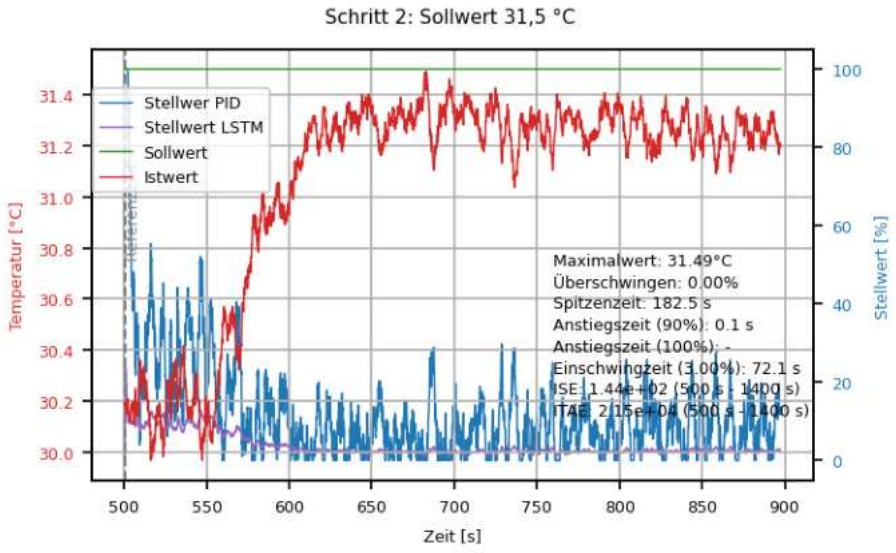


ABBILDUNG B.40: LSTM-Regler Testdurchlauf Schritt 2

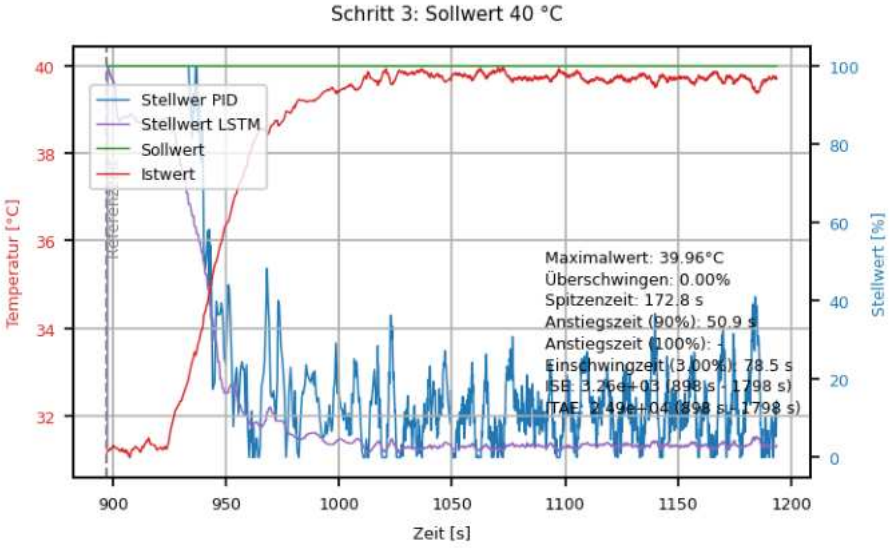


ABBILDUNG B.41: LSTM-Regler Testdurchlauf Schritt 3

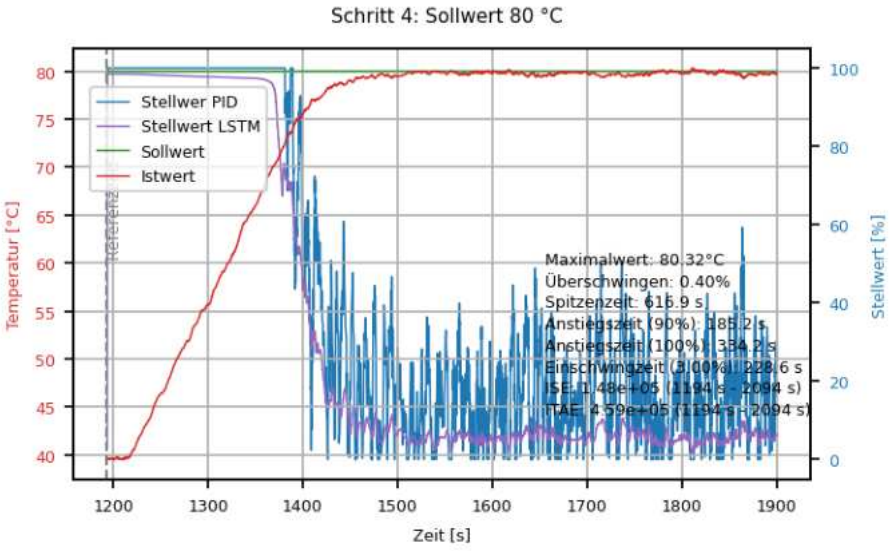


ABBILDUNG B.42: LSTM-Regler Testdurchlauf Schritt 4

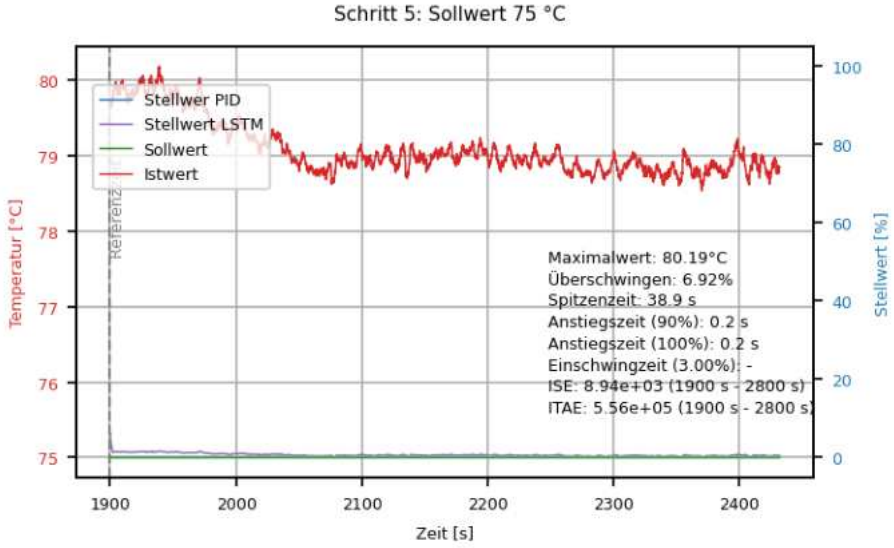


ABBILDUNG B.43: LSTM-Regler Testdurchlauf Schritt 5

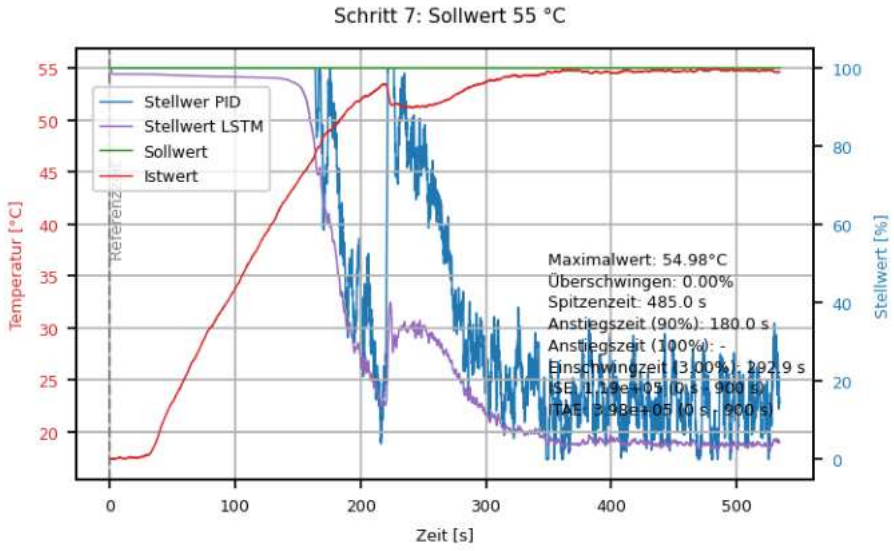


ABBILDUNG B.44: LSTM-Regler Testdurchlauf Schritt 7

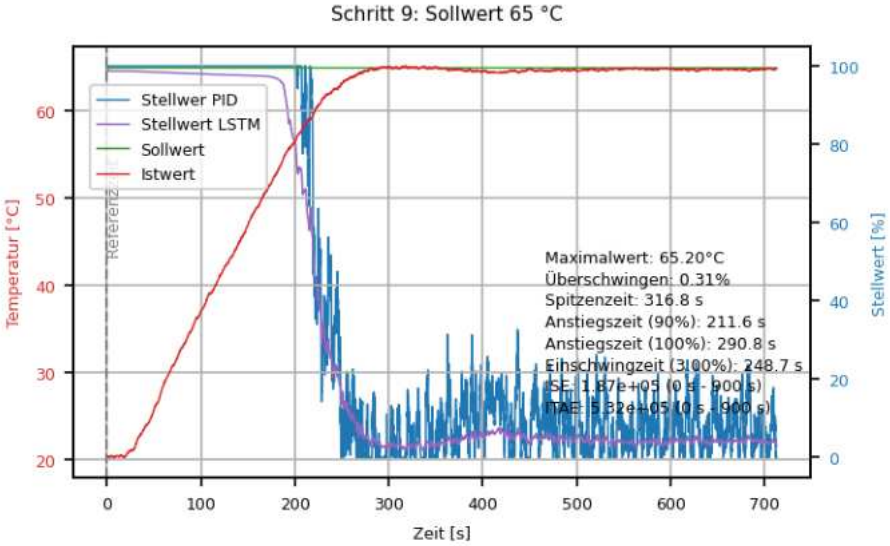


ABBILDUNG B.45: LSTM-Regler Testdurchlauf Schritt 9

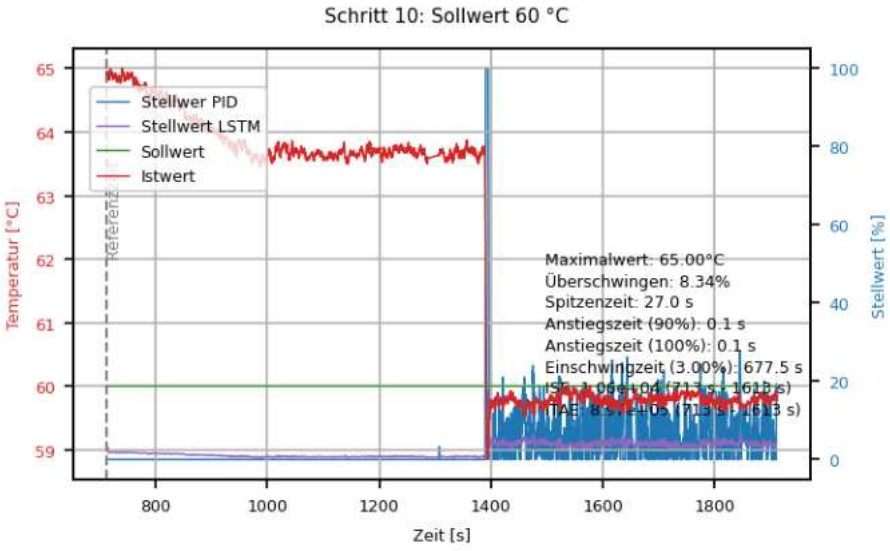


ABBILDUNG B.46: LSTM-Regler Testdurchlauf Schritt 10

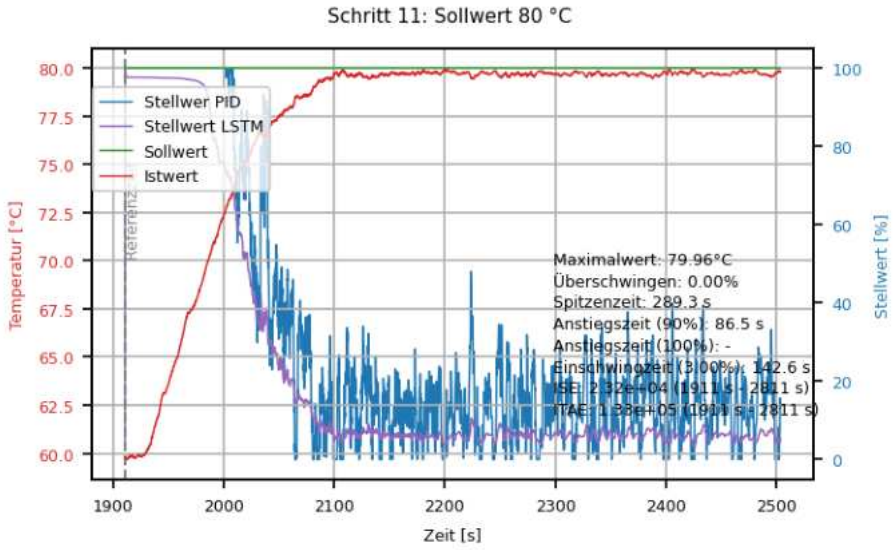


ABBILDUNG B.47: LSTM-Regler Testdurchlauf Schritt 11

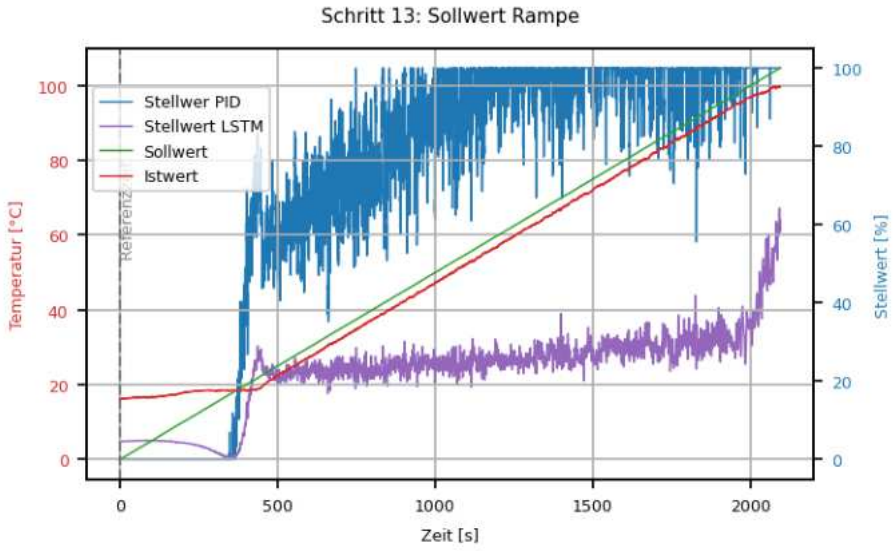


ABBILDUNG B.48: LSTM-Regler Testdurchlauf Schritt 13

B.8 Testdurchlauf MPC-Regler mit FFNN

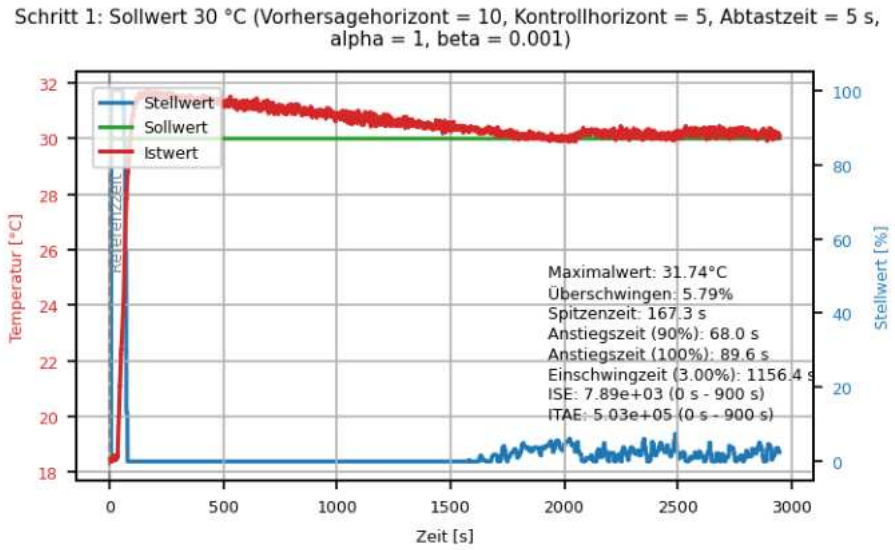


ABBILDUNG B.49: FFNN-MPC-Regler Testdurchlauf Schritt 1

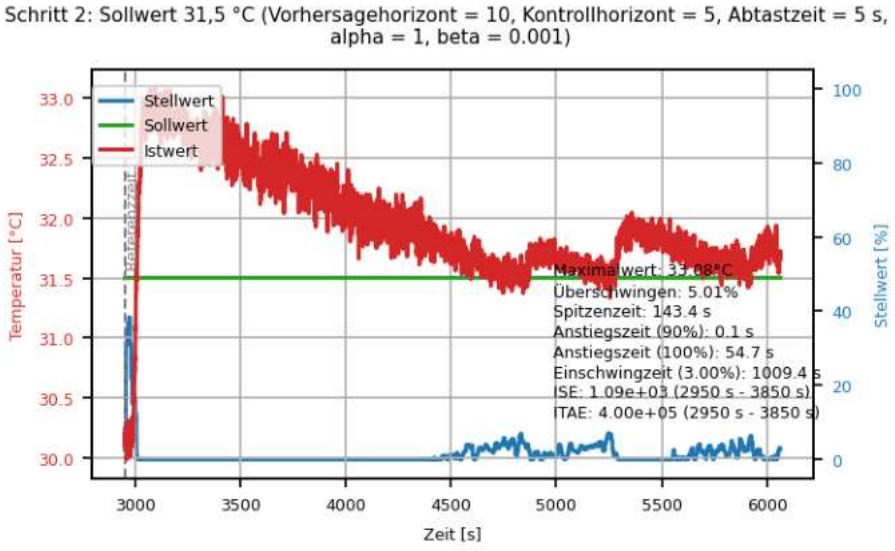


ABBILDUNG B.50: FFNN-MPC-Regler Testdurchlauf Schritt 2

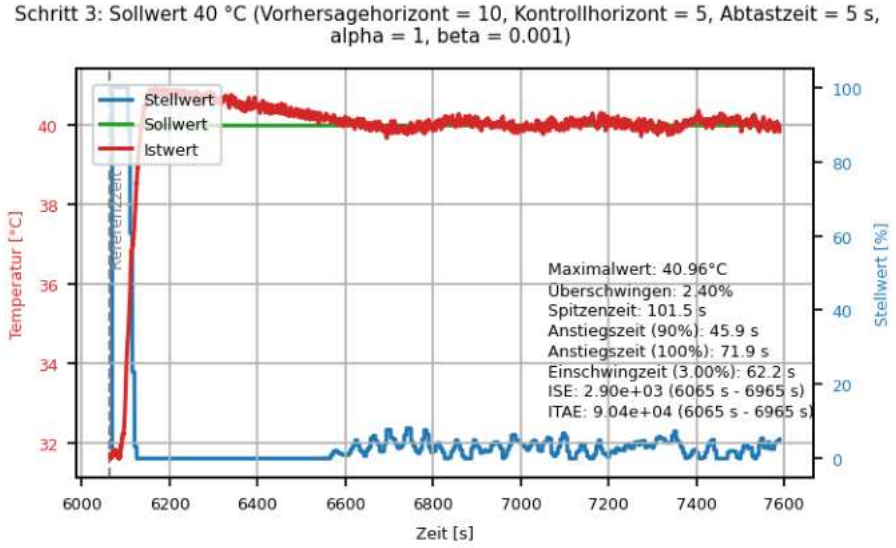


ABBILDUNG B.51: FFNN-MPC-Regler Testdurchlauf Schritt 3

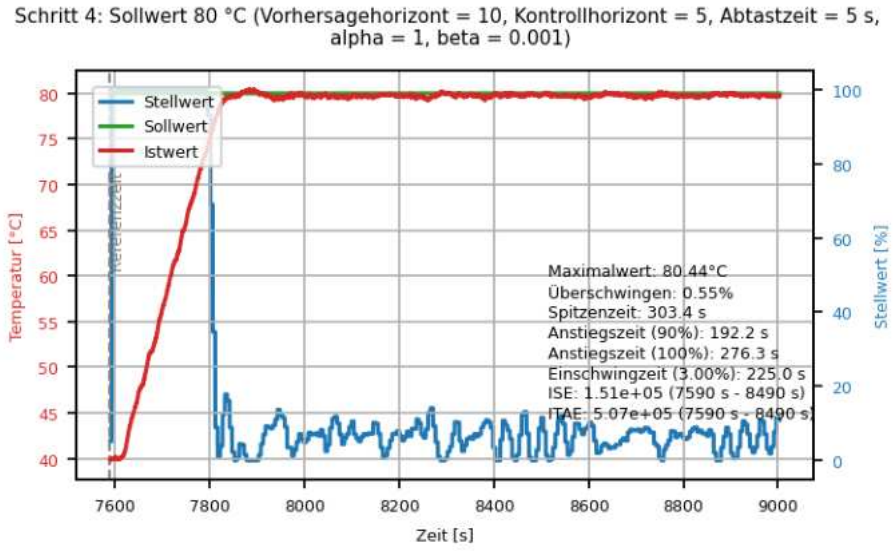


ABBILDUNG B.52: FFNN-MPC-Regler Testdurchlauf Schritt 4

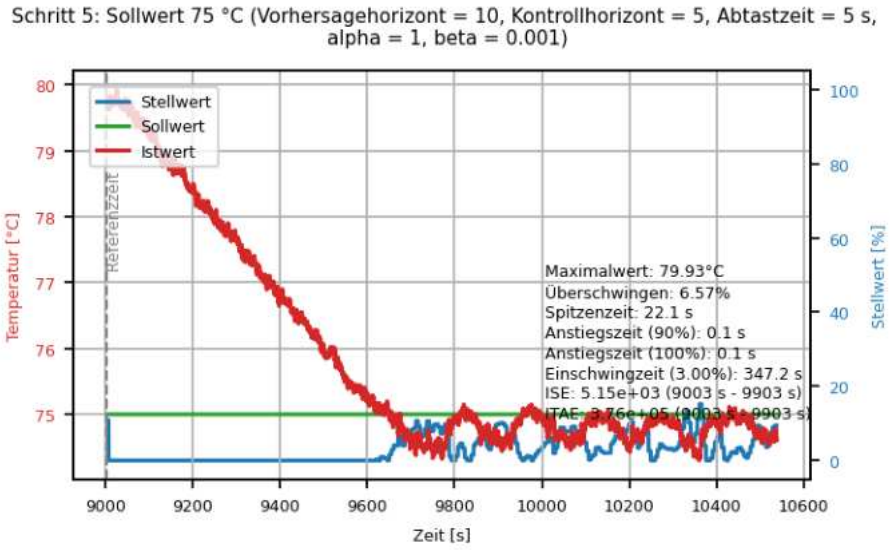


ABBILDUNG B.53: FFNN-MPC-Regler Testdurchlauf Schritt 5

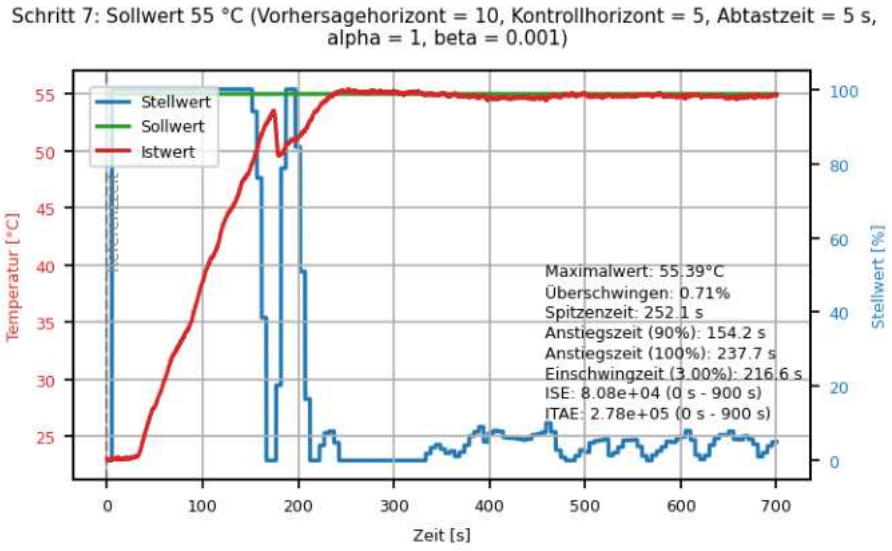


ABBILDUNG B.54: FFNN-MPC-Regler Testdurchlauf Schritt 7

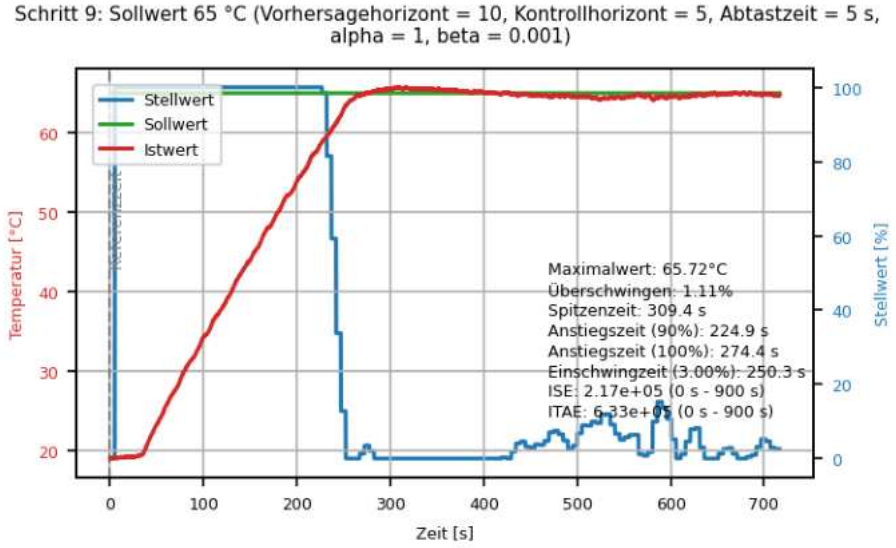


ABBILDUNG B.55: FFNN-MPC-Regler Testdurchlauf Schritt 9

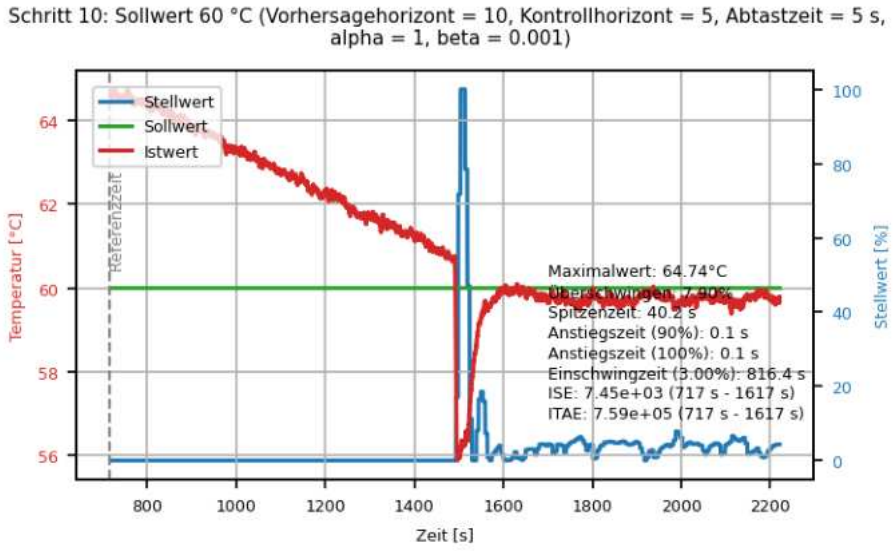


ABBILDUNG B.56: FFNN-MPC-Regler Testdurchlauf Schritt 10

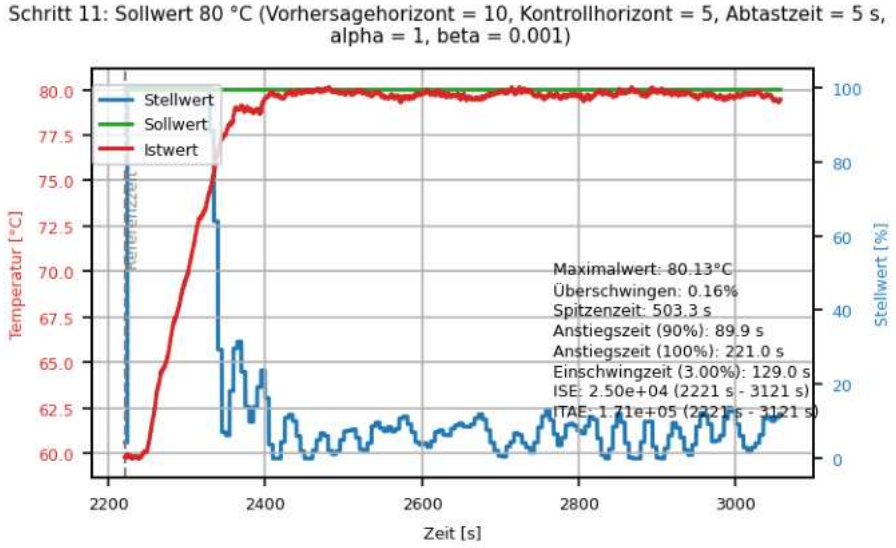


ABBILDUNG B.57: FFNN-MPC-Regler Testdurchlauf Schritt 11

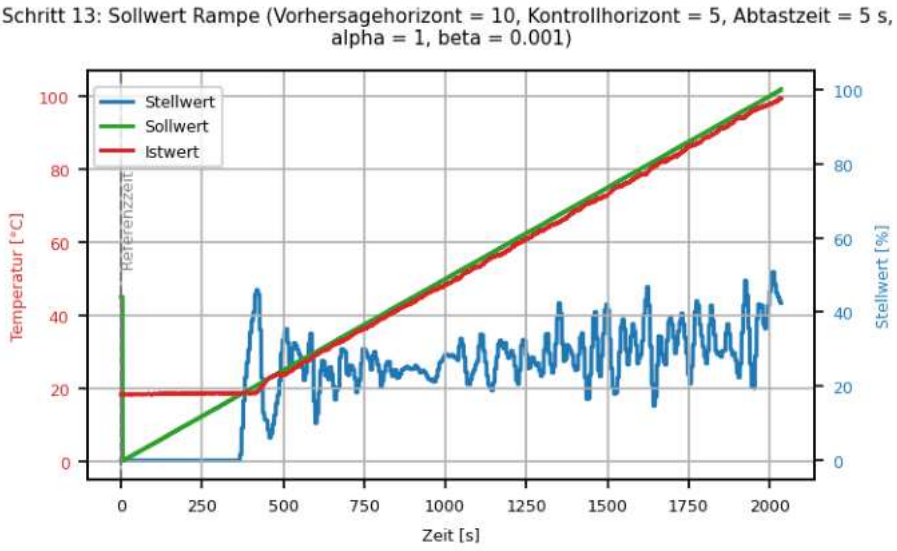
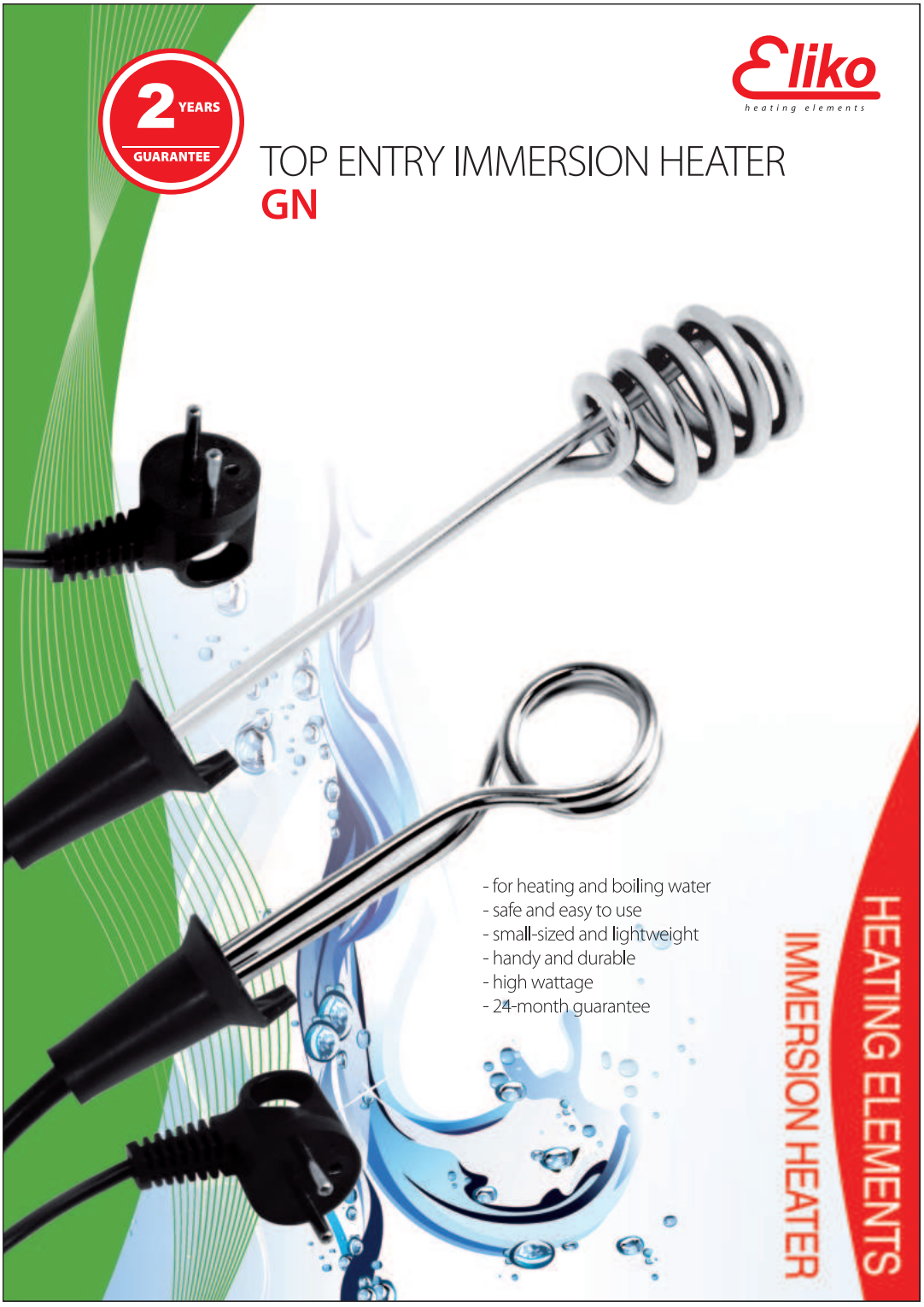


ABBILDUNG B.58: FFNN-MPC-Regler Testdurchlauf Schritt 13

Anhang C

Datenblätter

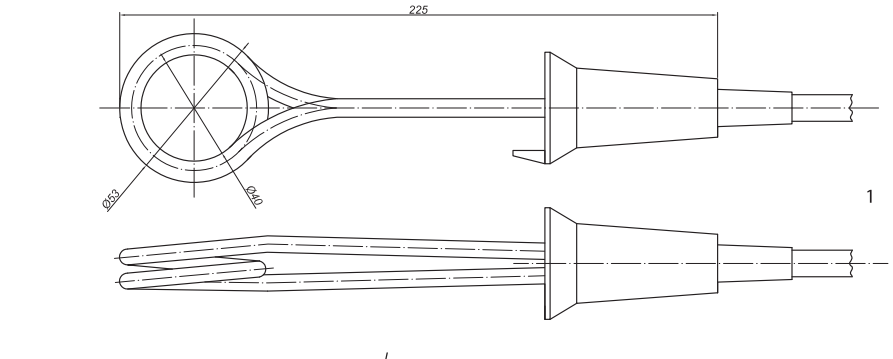
C.1 Datenblatt ELIKO Tauchsieder

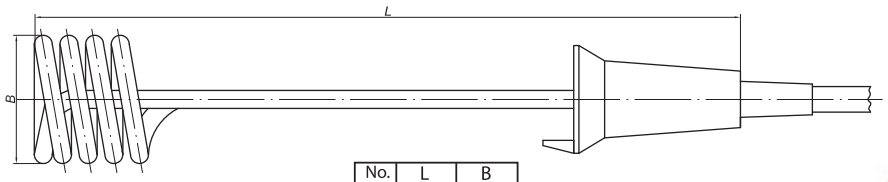


2 YEARS GUARANTEE

Eliko heating elements

TOP ENTRY IMMERSION HEATER
GN





No.	L	B
2	275	50
3	310	58

Top entry immersion heaters are used for heating and boiling water. These cheap, safe and easy to use heating units have been very popular with customers for years. Depending on their size and wattage, they can be used for heating suitable amounts of water. They are indispensable when travelling, on a camping or around the household.

SPECIFICATIONS

1.	W004	IMMERSION HEATERS 900W	900W	230V
2.	W005	IMMERSION HEATERS 1400W	1400W	230V
3.	W040	IMMERSION HEATERS 2000W	2000W	230V

HEATING ELEMENTS
IMMERSION HEATER

Eliko® elementy grzejne

equal

ELIKO Przedsiębiorstwo Produkcyjno Usługowo Handlowe
Elżbieta Mońka, Roman Mońka - Sp. Jawna
ul. Częstochowska 30, 42-274 Konopiska
Telefon: (+48) 34 328 23 21, (+48) 34 328 20 45, (+48) 34 328 37 48
Fax: (+48) 34 328 20 44, info@eliko.pl, www.eliko.pl

www.eliko.pl

Datenblatt entnommen aus [53]

C.2 Datenblatt Thyristorsteller zur Phasenanschnittsteuerung

CE

Montage-, und Betriebsanleitung Thyristorsteller

A-senco® Phasenanschnittsteuerung

Die A-senco Power Thyristorsteller steuern entsprechend einem analogen Eingangssignal (z. B. 0 (2) -10V oder 4 - 20mA) die Leistung einer angeschlossenen Last. Dabei wird durch Phasenanschnitt der positiven und negativen Sinushalbwellen eine stufenlose Leistungssteuerung von Verbrauchern möglich. Im Ergebnis ist dies mit einer Dimmerschaltung einer Glühbirne vergleichbar.
Durch die kompakte Bauweise ist ein vielfältiger Einsatz in Wechselstromnetzen, sowohl zur Steuerung von ohmschen Lasten (z. B. el. Heizungen), als auch induktiven Lasten (z. B. Motoren) möglich. Bei Steuerung insbesondere von induktiven Lasten (z. B. Drehzahlregelung von Motoren) ist vor Anwendung abzuklären, ob der Verbraucher aufgrund seiner Bauweise für den Betrieb durch Phasenanschnitt geeignet ist.

Die Vorzüge der A-senco Power-Thyristorsteller sind:
Kontaktlos, verschleißfrei, funkenlos und stufenlose Leistungssteuerung bei geringer Verlustleistung und dadurch langlebig. Kompakte Abmessungen, hohe Strombelastbarkeit und montagefreundliche Handhabung.
Die Gehäusekonstruktion besteht aus flammwidrigen Epoxy-/Kunststoffkonstruktionen, widerstandsfähig gegen hohe mech. Belastung und Vibrationen.

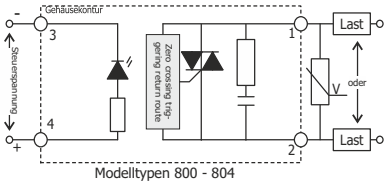
Modelltypen:

Zur Leistungssteuerung
von Wechselstromlasten
und 0(2)-10VDC Input

Art.-Nr. SCR-800
Art.-Nr. SCR-801
Art.-Nr. SCR-802
Art.-Nr. SCR-804

Technik / Funktion:

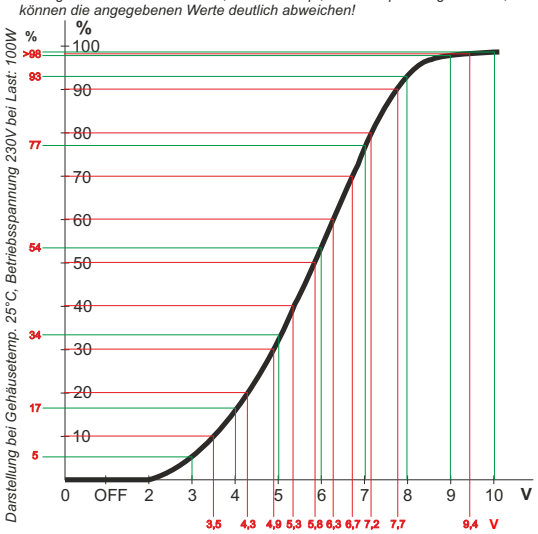
Innerer Aufbau (vereinfacht)



Modelltypen 800 - 804

Prozentualer Lastverlauf vs. Steuerspannung

Bedingt durch Bauteiltoleranzen, Betriebstemp., Betriebsspannung und Last, können die angegebenen Werte deutlich abweichen!



Hinweise für den Betrieb:

- Thyristorsteller mit Phasenanschnitt sind naturgemäß nur zur Schaltung von Wechselspannungen geeignet.

Durch die Integration einer RC-Schutzschaltung in unseren Thyristorstellern, kann auch im OFF-Zustand ein sehr geringer Wechselstrom im einstelligen mA-Bereich fließen. Für den Anschluss von Heizungen, Motoren oder ähnl. Lasten hat dies keine Bedeutung. Der Betrieb erfordert eine zu steuernde Mindestlast von ca. 25W.
Die zu steuernden Spannungen müssen sich kontinuierlich im angegebenen Spannungsbereich (angegebene Ausgangsspannung) befinden.
Überstrom bei Kurzschlüssen oder Überlast sind die häufigste Ursache für Ausfälle bei Halbleiterschaltern. Wir empfehlen Ihnen den Einsatz eines für die Schaltung Ihrer Last geeigneten Varistors (parallel zum Ausgang an Klemme 1 und 2 verdrahtet) zur Absicherung gegen Überspannungsspitzen aus dem Netz oder der Last. Varistoren sind unabhängig der Art der zu schaltenden Last anwendbar und haben keinen Einfluss auf die Funktion.
Je nach Anwendung (insbesondere induktive Lasten) kann in Verbindung mit dem verbauten Gesamtequipment, eine zusätzliche Funkentstörung notwendig sein (z.B. Art.-Nr. EB-613 für Lasten bis 3A)

Bitte beachten Sie, dass defekte Halbleiterbausteine von Garantieleistungen ausgeschlossen sind!
Die Wärmeerzeugung des Relais liegt bei ca. 1,5 Watt / Ampere geschalteter Last. Die angegebenen Temperaturen dürfen dabei nicht überschritten werden. Wir empfehlen Ihnen unser nachfolgend aufgelistetes Zubehör an Kühlkörpern.

Belegung der Anschlussterminals:

Last (Klemme 1 und 2): M4 Schraubklemme
Verwenden Sie zur Konnektierung nur 4,2mm Gabelschuhe od. Ringösen. Ab 25A Last nur Ringösen.
Terminals Steuerspannung (Klemme 3 und 4): M3 Schraubterminal
Verwenden Sie zur Konnektierung 3,2mm Gabelschuhe od. Ringösen.

Empfohlene Varistoren:

Für Modelltypen SSR-800 ...814: Varistor 275 V-Typ (Varistorspannung / 1mA) Art.-Nr. EB-25
(nur für 230V-Lasten. Je nach Last können andere Volttypen zur Anwendung kommen.)

Empfohlene Kühlkörper:

Für alle hier aufgeführten Modelltypen passend: siehe Anlage 1 "Bemessung von Kühlkörpern" zu dieser Bedienanleitung.
Für Lasten unterhalb 1A können Vollmetall-Hutschienenadapter verwendet werden. Art.-Nr. HSA-1

Berechnung der schaltbaren Nennlast:

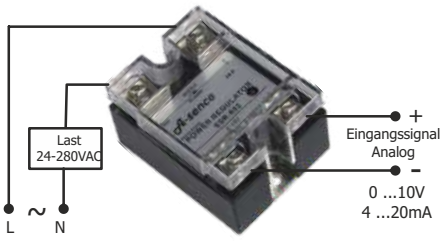
Versch. Lasten produzieren beim Einschalten hohe Einschalt-Stromspitzen. Um eine Überlastung der Thyristoren zu vermeiden, finden Sie nachstehend einige beispielhafte Angaben zur Auslegung. Diese sind unverbindliche Richtwerte und können ggf. auch stark abweichen.

Beispiel: Wertangabe 0,8 bedeutet:
Der im Datenblatt je nach Modelltyp angegebene max. Laststromwert (beispielhaft 40A), darf im Fall einer Glühbirne 40A x0,8 = 32 A betragen.

Lasttyp	Faktor
Rein ohmsche Last ohne erhöhten Einschaltstrom	1
Glühbirne	0,8
1-Phasen Motor	0,12/0,24
3-Phasen-Motor	0,18/0,33

Anschlussschema:

Modelltypen 800 ... 804



Maße: (LxBxH) 58x45x29mm (32mm mit Cover)
Lochmaß: zentrisch 48mm Lochabstand; 2x Durchmesser: 4,2mm

Technische Daten:

Modelltyp: SSR-	800	801	802	804		
Max. Laststrom	10A AC	25A AC	40A AC	80A AC		
Last-Spannungsbereich (Akzeptierte Spannung der Last)	180 - 280VAC					
Last-Frequenzbereich (Akzeptierte Frequenz des Laststroms)	47 ...53 Hz					
Spannungsabfall am Ausgang	< 1,5 V					
Steuerstrom in mA						
Steuerspannung in Volt	2-10V	2 - 10V	2 - 10V	2- 10V		
Wärmeerzeugung kontinuierlich pro Ampere Last	max. 1,5 Watt bei 100% ED					

Zulässige Umgebungstemperatur	-25°C ...+50°C
Zulässige max. Bauteiltemperatur	-25°C ...+60°C
Spannungsfestigkeit zwischen Eingang und Ausgang	2500 V AC 1 Minute
Eingangswiderstand	Bei 4V ca. 1mA, steigegnd bei 7V ca. 2,5mA, bei 10V ca. 3mA

Sicherheitshinweise:

- Bauen Sie das Relais in ein dafür zugelassenes Gehäuse ein, dessen Einbausituation den Anforderungen der Schutzart Ip20 oder größer entspricht. Schutzart Ip20 besteht für das Relais nur mit zugehöriger Schutzkappe!
  Achten Sie beim Einbau auf eine ausreichende Wärmeabfuhr über die metallische Kontaktfläche.
- Bei Schaltung größerer Lasten besteht die Möglichkeit einer schädlichen Auswirkung auf das vorhandene Stromnetz. Klären Sie mit Ihrem Netzlieferanten die Zulässigkeit der Verwendung von Phasenanschnittsteuerungen in der von Ihnen gewünschten Leistung ab.

- Zur Integration von SCR-Relais in steuertechnische Prozesse kann eine individuelle Gefahrenanalyse notwendig sein. Beachten Sie in diesem Zusammenhang die Tatsache, dass bei Ausfall von Halbleiterrelais (sog. Durchlegieren) in der Regel die Last kontinuierlich durchgeschaltet bleibt. Besteht die Möglichkeit, dass bei Ausfall des SCR-Relais eine Gefahr entsteht, sind zusätzliche Maßnahmen erforderlich (z. B. Sicherheitsabschaltung).
- Benutzen Sie das Relais nicht in explosionsgefährdeter Atmosphäre oder in der Nähe brennbarer Flüssigkeiten oder Gase.
- Bedenken Sie, dass ein unqualifizierter Umgang mit Strom Schmerzen, bleibende gesundheitliche Schäden oder Ihren Tod zur Folge haben kann. Zu den Folgen des Todes informieren Sie sich in Ihrer Bibel.
- Diese Bedienungsanleitung setzt eine Qualifikation im Umgang mit el. Betriebsmitteln voraus. Wenden Sie sich an Ihren örtlichen Elektroinstallateur, falls Sie keine fachliche Qualifikation besitzen!

Bitte beachten Sie bei einer Außerbetriebnahme, dass SCR-Relais entsprechend der Elektronikschrottverordnung dem Recycling zugeführt werden. Bitte erkundigen Sie sich nach der am Betriebsstandort zum Zeitpunkt der Außerbetriebsetzung gültigen abfalltechnischen Behandlung bei Ihrer zuständigen kommunalen Behörde.

Vertrieb / Kundendienst Deutschland:

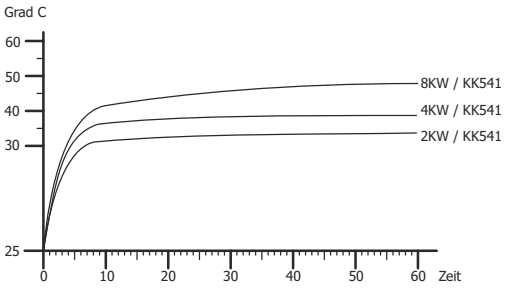
Pohltechnik.com GbR
Schnaitbergstraße 4
D-73457 Essingen
info@pohltechnic.com
0049 7365 964942-0 Tel.
0049 7365 964942-9 Fax

Trotz sorgfältiger Erstellung dieser Anleitung können Fehler in der Dokumentation, insbesondere durch techn. Änderungen nicht ausgeschlossen werden. Wir freuen uns über Verbesserungsvorschläge und Anregungen, welche die Verständlichkeit unserer Produkte erhöhen und sind dankbar für Ihre Nachricht per Mail.

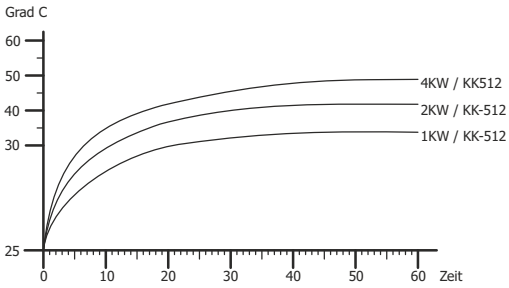
www.Pohltechnic.com

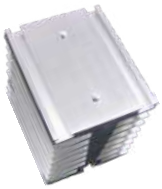
Bemessung von Kühlkörpern bei SCR-Relais und Thyristorstellern:

Je nach angeschlossener Last sind entsprechende Kühlkörper zu verwenden.
Die folgenden Diagramme wurden bei frei stehenden Kühlkörpern bei Umgebungstemperaturen von 24 -26°C ermittelt. Einschaltdauer= 100%.
Temperaturangaben bei Verwendung als passive Kühlkörper (ohne Gebläse) bei stehender Luft
Die Temperaturangabe wurde mittels Messung der Kerntemperatur des Kühlkörpers direkt unterhalb der Wärmetauschfläche des SCR-Thyristorstellers ermittelt. Durch äußere Einflüsse oder unterschiedliche Montagesituation können Abweichungen entstehen.
Die Kennlinien sind deshalb nur unverbindliche Orientierungswerte. Bei ungünstigen Bedingungen empfiehlt sich die Verwendung des nächst größeren Kühlkörpers. Die max. Temperatur der Wärmetauschfläche eines Halbleiterschalters beträgt ca. 60°. Ab 70°C können Schaltfehler auftreten, sowie Ausfälle aufgrund zu hoher Temperaturen entstehen.

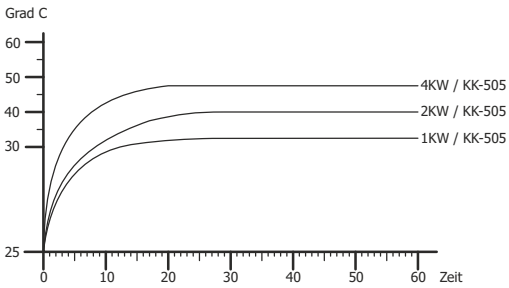


Art.-Nr. KK-541
Vollaluminium, Gewicht ca. 0,76 kg
Grundfläche: 150x99mm Höhe: 80mm

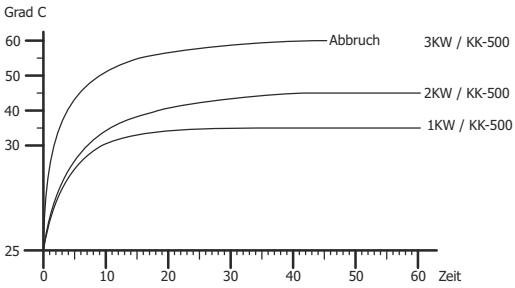




Art.-Nr. KK-512
Vollaluminium, Gew. ca. 0,41kg
Grundfläche: 80x66mm Höhe: 80mm



Art.-Nr. KK-505
Vollaluminium, Gew.0,2kg
Grundfläche: 125x75mm Höhe: 51mm



Art.-Nr. KK-500
Vollaluminium Gewicht ca. 0,1 kg
Grundfläche: 80x50mm Höhe: 50mm

www.Pohltechnic.com

Preiswert ● Industriequalität ● Techn. Service ● Tel. Beratung z. Normaltarif ● Einbau- u. Fertiggeräte

Art.-Nr.: SCR-800 ...804 Anlage Kühlkörper **Betriebsanleitung: Ausgabe 20120609** **Anlage 01**

Datenblatt entnommen aus [39]

C.3 Datenblattauszüge E/A-Modul LabJack

Subsections

14.1 AIN Extended Features

14.2 Extended Channels (T7 Only)

AIN Summary By Device

T4

Analog Inputs:
4 high-voltage (AIN0-AIN3)
8 low-voltage (AIN4-AIN11)
Voltage Ranges:
±10 V or 0-2.5 V (Appendix A-3-1-1 and Appendix A-3-1-3)
Resolution:
12-bit
Max Data Rate:
40000 samples/second in stream mode (Appendix A-1)
Sampling Modes:
Single-ended

T7

Analog Inputs:
14 (AIN0-AIN13)
Voltage Ranges:
±10 V, ±1 V, ±0.1 V, ±0.01 V (Appendix A-3-2-1 and Appendix A-3-2-3)
Resolution:
T7: 16-bit
T7-Pro: 24-bit
Effective Resolution:
T7: 16 to 19 bit*
T7-Pro: 16 to 22 bit*
*At gain 1x. For more details, see Appendix A-3-2-2.
Max Data Rate:
100000 samples/second in stream mode (Appendix A-1)
Sampling Modes:
Configurable as single-ended or differential
Extended Channels:
The number of analog inputs can be extended to 84 with a MUX80 (AIN48-AIN127)

T8

Analog Inputs:
8 (AIN0-AIN7)
Voltage Ranges:
±11 V, ±9.6 V, ±4.8 V, ±2.4 V, ±1.2 V, ±0.6 V, ±0.3 V, ±0.15 V, ±0.075 V, ±0.036 V,

14.1.0.1 Excitation Circuits [T-Series Datasheet]

Overview

AIN-EF indices that need to measure resistance (Resistance, RTD, Thermistor) can use different types of excitation circuits. The excitation circuit converts the varying resistance to a voltage signal that can be measured by the LabJack.

Individual AIN-EF indices may only support a subset of the circuits listed here.

For AIN-EF indices that require excitation circuits, the circuit indices below are written to AIN#_EF_CONFIG_B.

The most commonly used circuit is #4, as it is designed for the LJTick-Resistor.

These circuits all use a voltage source or current source for excitation. Note that any noise in the excitation source will result in proportionate noise in the sensor signal. Sources designed for excitation (e.g. voltage reference) are recommended rather than power supplies (e.g. VS).

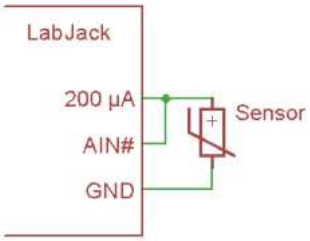
Current Source Excitation Circuits

The first 3 excitation circuits are specific to current source excitation. A current source varies voltage so it can provide the specified fixed current.

Circuit 0 – 200 µA Current Source - T7:

This excitation circuit uses the 200 µA current source available on the T7 to excite a sensor. It calculates resistance based on the measured voltage and the stored factory calibration value for 200UA. This circuit is useful for smaller resistances such as RTDs.

The following figure shows a basic single sensor connection, but if the AIN is configured and connected as differential, multiple sensors can be put in series as shown in figures from the 200UA/10UA documentation.



200µA Current Source

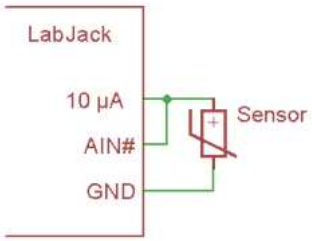
Configuration registers:

- AIN#_EF_CONFIG_C - Ignored
- AIN#_EF_CONFIG_D - Ignored
- AIN#_EF_CONFIG_E - Ignored

Circuit 1 – 10 µA Current Source - T7:

This excitation circuit uses the 10 µA current source available on the T7 to excite a sensor. It calculates resistance based on the measured voltage and the stored factory calibration value for 10UA. This circuit is useful for larger resistances such as thermistors.

The following figure shows a basic single sensor connection, but if the AIN is configured and connected as differential, multiple sensors can be put in series as shown in figures from the 200UA/10UA documentation.



10 µA current Source

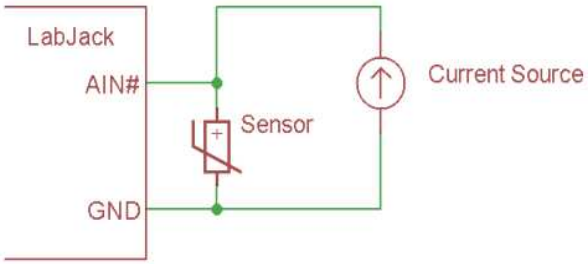
Configuration registers:

- AIN#_EF_CONFIG_C - Ignored
- AIN#_EF_CONFIG_D - Ignored
- AIN#_EF_CONFIG_E - Ignored

Circuit 2 – Custom Current Source:

This excitation circuit uses a current source external to the LabJack. The current provided by the source is specified during configuration of the AIN#_EF.

The following figure shows a basic single sensor connection, but if the AIN is configured and connected as differential, multiple sensors can be put in series.



Configuration registers:

- AIN#_EF_CONFIG_C - Ignored
- AIN#_EF_CONFIG_D - Excitation Amps
- AIN#_EF_CONFIG_E - Ignored

Resistive Divider Excitation Circuits

Divider circuits rely on an excitation source and a fixed resistor in series with the sensor.

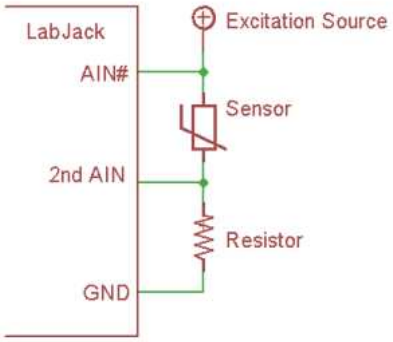
Circuit 3 – Divider with Measured Excitation - Differential:

This circuit has an excitation source in series with a sensor which is then in series with a fixed resistor to ground. The device takes the single-ended reading of 2ndAIN to get Vresistor, and takes the differential reading of AIN# to get Vsensor. Vresistor is divided by the specified fixed resistance to get current, and then Vsensor is divided by that current to get the resistance of the sensor.

The drawing shows the most common way of connecting. AIN# is a positive differential channel (e.g. AIN2) and 2ndAIN is the negative associated with that channel (e.g. AIN3).

Alternatively, any differential pair of analog inputs can be connected across the sensor, and 2ndAIN can be any single-ended analog input. This allows multiple sensors to be connected in series with a single excitation source.

AIN# must be pre-configured by the user as differential or an error will be thrown. Thus this circuit is supported on the T7 but not the T4.

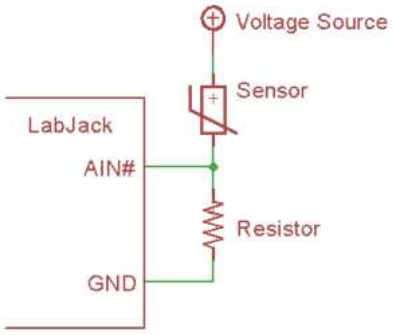


Configuration registers:

- AIN#_EF_CONFIG_C - 2nd AIN: Channel Number to Measure Vresistor
- AIN#_EF_CONFIG_D - Ignored
- AIN#_EF_CONFIG_E - Fixed Resistor Ohms

Circuit 4 – Voltage Source with Specified Value:

This excitation circuit uses a voltage source and a shunt resistor. Values for the output of the voltage source and the resistor must be provided during AIN#_EF configuration. When using this circuit, the LabJack will measure the voltage between the sensor and the resistor, then calculate the resistance of the sensor.



Configuration registers:

- AIN#_EF_CONFIG_C - Ignored
- AIN#_EF_CONFIG_D - Excitation Volts
- AIN#_EF_CONFIG_E - Fixed Resistor Ohms.

Note for the LJTick-Resistance: This excitation circuit #4 is the most common circuit used with the LJTick-Resistance. The "Resistor" shown above is built into the LJTick-Resistance, so to create this circuit simply connect one side of the RTD to LJTR-Vref and the other side of the RTD to LJTR-VINx.

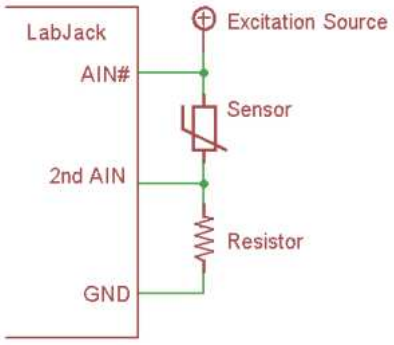
Note for the T8: The T8 has two 3.3 V reference voltage outputs that could be suitable for excitation.

Circuit 5 – Divider with Measured Excitation - Single-Ended:

This circuit has an excitation source in series with a sensor which is then in series with a fixed resistor to ground. The device takes the single-ended reading of 2ndAIN to get Vresistor, and takes the single-ended reading of AIN# minus the reading from 2ndAIN to get Vsensor. Vresistor is divided by the specified resistance to get current, and then Vsensor is divided by current to get the resistance of the sensor.

Must be connected exactly as shown in the drawing. AIN# and 2ndAIN can be any channels.

This excitation circuit looks the same as circuit #3, but the voltage across the sensor is determined by the difference of 2 single-ended readings rather than a single differential reading. That means this circuit is supported on all devices and is limited to a single sensor in series with the excitation source.

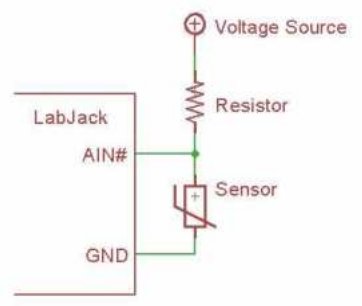


Configuration registers:

- AIN#_EF_CONFIG_C - 2nd AIN: Channel Number to Measure Vresistor
- AIN#_EF_CONFIG_D - Ignored
- AIN#_EF_CONFIG_E - Fixed Resistor Ohms

Circuit 1004 – Voltage Source with Specified Value (swapped fixed resistor position) - T4/T7

This excitation circuit uses a voltage source and a shunt resistor. It is the same as circuit 4 except with the fixed resistor and sensor positions swapped. Values for the output of the voltage source and the resistor must be provided during AIN#_EF configuration. When using this circuit, the LabJack will measure the voltage between the sensor and the resistor, then calculate the resistance of the sensor.



Configuration registers:

- AIN#_EF_CONFIG_C - Ignored
- AIN#_EF_CONFIG_D - Excitation Volts
- AIN#_EF_CONFIG_E - Fixed Resistor Ohms.

Note for the T8: The T8 has two 3.3 V reference voltage outputs that could be suitable for excitation.

14.1.3 RTD [T-Series Datasheet]

Overview

AIN#_EF_INDEX values:

40: PT100
41: PT500
42: PT1000

This RTD Extended Feature automatically performs calculations for a Resistance Temperature Detector (RTD). RTD types are listed above.

When AIN#_EF_READ_A is read, the T-series device reads an analog input and calculates the resistance of the RTD. Temperature is then calculated using the rational polynomial technique.

An RTD (aka PT100, PT1000) is a type of temperature sensor. See the Temperature Sensors App Note.

An RTD provides a varying resistance, but the LabJack measures voltage, so some sort of circuit must be used to convert the varying resistance to a varying voltage. This AIN-EF supports various excitation circuits. The best option is usually the LJTick-Resistance, which would be excitation circuit #4.

The resistance to temperature conversion is done using the RTD Rational Polynomial technique. The polynomial coefficients are fixed and assume the most common RTD characteristics, where a PT100 (for example) has a resistance of 100.0 ohms at 0 °C and a coefficient of 0.00385. Math for non-standard RTDs will have to be handled by the user. PT500 is assumed to have 5 times the resistance of a PT100, and a PT1000 is assumed to have 10 times. More information about the polynomial can be found here: http://www.mosaic-industries.com/embedded-systems/microcontroller-projec...

Configuration

To configure, write to the following registers.

AIN#_EF_CONFIG_A - Options: Selects temperature units:

- 0 = K
- 1 = °C
- 2 = °F

AIN#_EF_CONFIG_B - Excitation Circuit Index: The index of the voltage divider excitation circuit to be used.

See 14.1.0.1 Excitation Circuits for circuit indices.

AIN#_EF_CONFIG_C - 2nd AIN: Channel Number to Measure Vresistor: For excitation circuits 3 and 5 this is the extra AIN used to measure the voltage across the fixed resistor. Ignored for other excitation circuits.

AIN#_EF_CONFIG_D - Excitation Volts or Amps: For excitation circuit 2 this is the fixed amps of the current source. For excitation circuit 4 this is the fixed volts of the voltage source. Ignored for other excitation circuits.

AIN#_EF_CONFIG_E - Fixed Resistor Ohms: For excitation circuits 3, 4 and 5, this is the ohms of the fixed resistor.

Remarks

The normal analog input settings are used for negative channel, resolution index, settling, and range.

T7: If the voltage will stay below 1.0V, use the 1.0V range for improved resolution and accuracy.

T8: For improved resolution and accuracy, use the lowest voltage range that is appropriate for your signal's voltage min/max. For example, if the signal's min/max is -0.0064V/0.0549V, use the T8 voltage range of ±0.075V.

Results

Retrieve the results by reading the following registers.

AIN#_EF_READ_A: Calculated temperature.
AIN#_EF_READ_B: Resistance of the RTD.
AIN#_EF_READ_C: Voltage across the RTD.
AIN#_EF_READ_D: Current through the RTD.

Only reading AIN#_EF_READ_A triggers a new measurement, so you must always read A before reading B, C or D.

Example

The LJTick-Resistance-1k is the best and easiest way to measure an RTD, but if you don't have an LJTR the 200UA source on the T7 is a quick way to get readings.

200UA current source, circuit #0:

Connect 200UA to AIN0 and connect a PT100 RTD from AIN0 to GND:

```
AIN0_EF_INDEX = 40      // Set AIN_EF0 to RTD100.
AIN0_EF_CONFIG_A = 0    // Set result units to
kelvin.
AIN0_EF_CONFIG_B = 0    // Set excitation circuit to
0.
```

Now each read of AIN0_EF_READ_A will measure the voltage on AIN0, use that to calculate resistance based on the factory stored value for 200UA, and use that to calculate temperature.

LJTick-Resistance-1k, circuit #4:

Connect a PT100 RTD from Vref to VINA on an LJTick-Resistance-1k that is plugged into the AIN0/AIN1 block.

```
AIN0_EF_INDEX = 40       // Set AIN_EF0 to RTD100.
AIN0_EF_CONFIG_A = 0     // Set result units to kelvin.
AIN0_EF_CONFIG_B = 4     // Set excitation circuit to 4.
AIN0_EF_CONFIG_D = 2.50  // Vref on the LJTR is 2.50 volts.
AIN0_EF_CONFIG_E = 1000  // We are using the 1k version of the LJTR.
```

Now each read of AIN0_EF_READ_A will measure the voltage on AIN0, do the voltage divider math to determine the resistance of the RTD, and use that to calculate temperature.

A-3-1-2 T4 Noise and Resolution [T-Series Datasheet]

ADC Noise and Resolution

T-series devices use an internal analog-to-digital converter (ADC) to convert analog voltage into digital representation. The ADC reports an analog voltage in terms of ADC counts, where a single ADC count is the smallest change in voltage that will affect the reported ADC value. A single ADC count is also known as the converter's least significant bit (LSB) voltage. The ADC's resolution defines the number of discrete voltages represented over a given input range. For example, a 16-bit ADC with a ±10 input range can report 65536 discrete voltages (2^16) and has an LSB voltage of 0.305 mV (`20 V ÷ 2^16`).

The stated resolution for an ADC is a theoretical, best-case value assuming no channel noise. In reality, every ADC works in conjunction with external circuitry (amplifiers, filters, etc.) which all possess some level of inherent noise. The noise of supporting hardware, in addition to noise of the ADC itself, all contribute to the channel resolution. In general, the resolution for an ADC and supporting hardware will be less than what is stated for the ADC. The combined resolution for an in-system ADC is termed effective resolution (aka ENOB). Simply put, the effective resolution is the equivalent resolution where analog voltages less than the LSB voltage are no longer differentiable from the inherent hardware noise.

The effective resolution is closely related to the error free code resolution (EFCR) or *flicker-free* code resolution. The EFCR represents the resolution on a channel immune to "bounce" or "flicker" from the inherent system noise. The EFCR is not reported in this appendix. However, it may be closely approximated by the following equation:

```
EFCR = effective resolution - 2.7 bits
```

The T4 and the T7 offer user-selectable effective resolution through the resolution index parameter on any one AIN channel. Internally, the ADC hardware uses modified sampling methods to reduce noise. Valid resolution index values are:

- 0-5 for the T4
- 0-8 for the T7
- 0-12 for the T7-Pro

Increasing the resolution index value will improve the channel resolution, but doing so will usually extend channel sampling times. See section 14.0 AIN for more information on the resolution index parameter and its use.

Noise and Resolution Data

High-Voltage Channels (AIN0-AIN3)

Resolution Index	Effective Resolution [bits]	Effective Resolution [mV]	AIN Sample Time [ms]
1	11.0	10.4	0.07
2	11.4	7.7	0.11
3	12.3	4.2	0.16
4	12.9	2.7	0.29
5*	13.2	2.2	0.5

Low-Voltage Channels (Applicable FIO & EIO)

Resolution Index	Effective Resolution [bits]	Effective Resolution [mV]	AIN Sample Time [ms]
1	10.8	1.2	0.07
2	11.6	0.69	0.11
3	12.0	0.52	0.16
4	12.2	0.43	0.29
5*	12.8	0.29	0.51

* Default command-response Resolution Index for the T4.

A-4 Analog Output [T-Series Datasheet]

Specifications for analog output channels (DAC0 and DAC1) are shown below.

T4

Table A4-1. T4 DAC Information. All specs at room temperature unless otherwise noted.

	Conditions	Min	Typical	Max	Units
Nominal Output Range [1]	No Load	0.01	-	4.98	Volts
	@ ±2.5 mA	0.30	-	4.69	Volts
Resolution	-	-	10	-	Bits
Absolute Accuracy	5% to 95%, No Load	-	±0.15	-	% FS
	-	-	±7.5	-	mV
Integral Linearity Error	-	-	-	±1	counts
Differential Linearity Error	-	-	±0.1	±0.5	counts
Noise [2]	-	-	±1	-	counts
Source Impedance [3]	-	-	110	±10	Ω
Current Limit [4]	Max to GND	-	32	-	mA
Time Constant	-	-	1	-	µs

[1] Maximum and minimum analog output voltage is limited by the supply voltages (VS and GND). The specifications assume VS is 5.0 volts. Also, the ability of the DAC output buffer to driver voltages close to the power rails, decreases with increasing output current.

[2] Specified with no load and set to 0 V. Noise will increase with load and voltage, to a maximum of 3 counts @ 4V with a 100 Ω load. The DAC's ability to reject noise decreases as the output voltage nears the Vs supply.

[3] The source impedance is a combination of fixed resistance and the impedance from the output buffer. The impedance from the buffer will vary depending on the operating conditions. When set to 4 V with a 100 Ω load, Rs is ~ 112 Ω. When set to 4 V with a 10 kΩ load, Rs is ~110 Ω.

[4] The output buffer will limit current to about 30 mA and can maintain this value continuously without damage. Take, for example, a 1.5 ohm resistor from DAC0 to GND, with the internal source impedance of 110 ohms, and DAC0 set to 4.5V. A simple calculation would predict a current of 4.5/(110+1.5) = 40 mA, but the output buffer will limit the current to 32 mA.

Datenblattauszüge entnommen aus [27]

Literaturverzeichnis

[1] APMonitor. *Automation with LSTM Network*. 2023. URL: https://apmonitor.com/pds/index.php/Main/LSTMAutomation (besucht am 14. 03. 2024).

[2] Aleks Basara. *8 Python-Bibliotheken für Machine Learning im Jahr 2024*. 2024. URL: https://www.fragment-studio.com/de/posts/8-python-libraries-for-machine-learning-in-2024 (besucht am 13. 05. 2024).

[3] *Bayesianische Optimierung: Definition und Funktionsweise*. 2023. URL: https://datascientest.com/de/bayesianische-optimierung-definition-und-funktionsweise (besucht am 28. 03. 2024).

[4] Alberto Bemporad. *Model Predictive Control Learning-based MPC*. 2023. URL: http://cse.lab.imtlucca.it/~bemporad/teaching/mpc/imt/8-data_driven_mpc.pdf (besucht am 05. 03. 2024).

[5] Prof. Dr. Lothar Berger. *Advanced Control Systems*. 2022.

[6] Prof. Dr. Lothar Berger. *Grundlagen der Regelungstechnik*. 2022.

[7] Stephen Boyd. *Convex Optimization*. Cambridge University Press, New York, 2013, S. 5–14. ISBN: 9780511804441. URL: https://doi.org/10.1017/CBO9780511804441.

[8] Dominic Brown und Martin Strube. *Simulationsgestützte Auslegung von Reglern mithilfe von Machine Learning*. 2020. URL: https://www.cea-wismar.de/asimsst/data/020-ASIM-SST-2020-brown.pdf (besucht am 08. 01. 2024).

[9] Emily Chen und Angela Zhao. *Quantum Neural Networks*. 2023. URL: https://medium.com/mit-6-s089-intro-to-quantum-computing/quantum-neural-networks-7b5bc469d984 (besucht am 08. 01. 2024).

[10] DataScientest. *Matplotlib: Alles über die Python-Bibliothek von Dataviz*. 2023. URL: https://datascientest.com/de/matplotlib-alles-wissen (besucht am 13. 05. 2024).

[11] DataScientest. *SciPy: Alles über die Python-Bibliothek für Machine Learning*. 2023. URL: https://datascientest.com/de/scipy-alles-uber-die-python-bibliothek-fur-machine-learning (besucht am 13. 05. 2024).

[12] scikit-learn developers. *Preprocessing data*. URL: https://scikit-learn.org/stable/modules/preprocessing.html (besucht am 19. 03. 2024).

[13] scikit-learn developers. *sklearn.feature_selection.SelectKBest*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (besucht am 19. 03. 2024).

[14] scikit-learn developers. *sklearn.preprocessing.MaxAbsScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html (besucht am 19. 03. 2024).

[15] scikit-learn developers. *sklearn.preprocessing.MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (besucht am 28. 03. 2024).

[16] *Eliko GN 2,0 Tauchsieder Wasserkocher Reisetauchsieder 2kW 2000 Watt 230 V, schwarz*. URL: https://m.media-amazon.com/images/I/611gSZKxWWL._AC_SL1500_.jpg (besucht am 22. 01. 2024).

[17] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. Springer Fachmedien Wiesbaden, 2016, S. 313–315. ISBN: 978-3-658-13549-2. DOI: 10.1007/978-3-658-13549-2. URL: https://doi.org/10.1007/978-3-658-13549-2.

[18] Marcello Fiducioso1 u. a. *Safe Contextual Bayesian Optimization for Sustainable Room Temperature PID Control Tuning*. URL: https://las.inf.ethz.ch/files/fiducioso19_bo4pid.pdf (besucht am 23. 02. 2024).

[19] William Holderbaum, Feras Alasali und Ayush Sinha. *Modellprädiktive Regelung*. Springer International Publishing, 2023, S. 141–158. ISBN: 978-3-031-45471-4. DOI: 10.1007/978-3-031-45471-4_5. URL: https://doi.org/10.1007/978-3-031-45471-4_5.

[20] Till Hörger. *Reinforcement Learning Framework zur optimalen Regelung von Orbitalantrieben unter Berücksichtigung von Robustheit und Betriebsbereichseinschränkungen*. 2021. URL: https://elib.dlr.de/143369/1/H%C3%B6rger%20Master%202021.pdf (besucht am 08. 01. 2024).

[21] Luca Invernizzi u. a. *Getting started with KerasTuner*. 2021. URL: https://keras.io/guides/keras_tuner/getting_started/ (besucht am 28. 03. 2024).

[22] Sandhya Krishnan. *How do determine the number of layers and neurons in the hidden layer?* 2021. URL: https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3 (besucht am 20. 03. 2024).

[23] Louisa Krutzfeldt. *Was ist Machine Learning? Algorithmen, Methoden und Beispiele*. 2023. URL: https://stackfuel.com/de/blog/machine-learning-algorithmen-data-analytics/ (besucht am 07. 01. 2024).

[24] Veikko Krypczyk. *Python: Eine schicke Oberfläche für intelligente Skripte*. 2022. URL: https://www.informatik-aktuell.de/entwicklung/programmiersprachen/python-eine-schicke-oberflaeche-fuer-intelligente-skripte.html (besucht am 13. 05. 2024).

[25] LabJack. *LabJack T4*. URL: https://labjack.com/products/labjack-t4 (besucht am 24. 01. 2024).

[26] LabJack. *LJTick-DAC Datasheet*. 2023. URL: https://labjack.com/pages/support?doc=%2Fdatasheets%2Faccessories%2Fljtick-dac-datasheet%2F (besucht am 25. 01. 2024).

[27] LabJack. *T-Series Datasheet*. 2023. URL: https://files.labjack.com/datasheets/LabJack-T-Series-Datasheet.pdf (besucht am 24. 01. 2024).

[28] *LabJack T4*. URL: https://labjack.com/cdn/shop/products/labjack-t4-276115.png?v=1674769838 (besucht am 22. 01. 2024).

[29] *LabJack T4*. URL: https://labjack.com/cdn/shop/products/labjack-t4-324916.jpg?v=1674763614 (besucht am 24. 01. 2024).

[30] Yong-Seok Lee und Dong-Won Jang. „Optimization of Neural Network-Based Self-Tuning PID Controllers for Second Order Mechanical Systems“. In: *Applied Sciences* 11.17 (2021). ISSN: 2076-3417. DOI: 10.3390/app11178002. URL: https://www.mdpi.com/2076-3417/11/17/8002.

[31] Christian Lemke. *Reinforcement Learning als Methode auf dem Weg hin zu einer generellen Künstlichen Intelligenz*. 2023. URL: https://www.alexanderthamm.com/de/blog/einfach-erklaert-so-funktioniert-reinforcement-learning/ (besucht am 08. 01. 2024).

[32] *Long Short-Term Memory Networks (LSTM) – einfach erklärt!* 2022. URL: https://databasecamp.de/ki/lstm (besucht am 15. 03. 2024).

[33] Prof. Dr. Jan Lunze. *Regelungstechnik 1*. Springer Fachmedien Berlin, Heidelberg, 2020, S. 1–7. ISBN: 978-3-662-60746-6. DOI: 10.1007/978-3-662-60746-6. URL: https://doi.org/10.1007/978-3-662-60746-6.

[34] MathWorks. *Design Neural Network Predictive Controller in Simulink*. URL: https://de.mathworks.com/help/deeplearning/ug/design-neural-network-predictive-controller-in-simulink.html (besucht am 26. 03. 2024).

[35] *Measures of controlled system performance*. URL: https://www.online-courses.vissim.us/Strathclyde/measures_of_controlled_system_pe.htm (besucht am 15. 02. 2024).

[36] University of Michigan. *Chemical Process Dynamics and Controls*. URL: https://ia800701.us.archive.org/28/items/ChemicalProcessDynamicsAndControls/ChemicalProcessDynamicsAndControls.pdf (besucht am 22. 02. 2024).

[37] Andreas Mockenhaupt. *Digitalisierung und Künstliche Intelligenz in der Produktion: Grundlagen und Anwendung*. Springer Fachmedien Wiesbaden, 2021, S. 138–142. ISBN: 978-3-658-32773-6. DOI: 10.1007/978-3-658-32773-6_3. URL: https://doi.org/10.1007/978-3-658-32773-6_3.

[38] *Montage-, und Betriebsanleitung Thyristorsteller*. URL: https://www.pohltechnik.com/media/files_public/b4c39b284da82f62e0192c27f395b55d/Bedien_SSR-812.pdf (besucht am 26. 01. 2024).

[39] *Montage-, und Betriebsanleitung Thyristorsteller*. URL: https://www.pohltechnik.com/de/scr-thyristorsteller/thyristorsteller-1-x-phasenanschnitt-230v-0-10v-dc-10-a?dl_media=527 (besucht am 26. 01. 2024).

[40] Learnbay Official. *Pytorch vs. Tensorflow: Major Difference Among Deep Learning*. 2023. URL: https://medium.com/@Learnbay_official/pytorch-vs-tensorflow-major-difference-among-deep-learning-cea7bf35ad11 (besucht am 13. 05. 2024).

[41] Pieter P. *PID Controllers*. 2023. URL: https://tttapa.github.io/Pages/Arduino/Control-Theory/Motor-Fader/PID-Controllers.html (besucht am 13. 02. 2024).

[42] Dr. Rüdiger Paschotta. *Phasenanschnittsteuerung*. 2023. URL: https://www.energie-lexikon.info/phasenanschnittsteuerung.html (besucht am 13. 05. 2024).

[43] *Pt100 oder Pt1000: der Unterschied*. URL: https://temperatur-profis.de/temperaturfuehler/pt100-oder-pt1000/ (besucht am 29. 01. 2024).

[44] Jann Raveling. *Was ist künstliche Intelligenz?* 2023. URL: https://www.wfb-bremen.de/de/page/stories/digitalisierung-industrie40/was-ist-kuenstliche-intelligenz-definition-ki (besucht am 07. 01. 2024).

[45] Katja Rietbrock und Volkmar Sterzing. *Künstliche Intelligenz steuert Ihre Anlage auf beste Weise*. 2021. URL: https://blog.siemens.com/2021/07/kunstliche-intelligenz-steuert-ihre-anlage-auf-beste-weise/ (besucht am 04. 03. 2024).

[46] SAP. *Was ist Machine Learning?* 2023. URL: https://www.sap.com/germany/products/artificial-intelligence/what-is-machine-learning.html (besucht am 07. 01. 2024).

[47] Manfred Schleicher und Winfried Schneider. *Thyristor-Leistungssteller – Grundlagen und Tipps für den Praktiker*. Seitenzahlen beziehen sich auf die Online-PDF-Version. JUMO, 2023, S. 9–10. ISBN: 978-3-935742-04-7. URL: https://campus.jumo.de/dl.cfm?h=C1D9D97DDE6433188A619804DB2664E2F70F421099F146844631FEB524777DD7.

[48] SciPy. *scipy.optimize.minimize*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html (besucht am 22. 02. 2024).

[49] *Spezifikationen für das Einschwingverhalten: Spitzenzeit, Einschwingzeit, Anstiegszeit, Überschwingen und prozentuales Überschwingen*. 2022. URL: https://aleksandarhaber.com/transient-response-specifications-peak-time-settling-time-rise-time-and-percent-overshoot/ (besucht am 12. 02. 2024).

[50] Pascal Stubel. *Untersuchung der Eignung von Reinforcement Learning Algorithmen zur Regelung einer Lüftungsanlage im Vergleich zu Standardreglern*. 2022. URL: https://reposit.haw-hamburg.de/bitstream/20.500.12738/14311/1/Bachelorarbeit_PascalStubel_geschw%C3%A4rzt.pdf (besucht am 08. 01. 2024).

[51] Nathan Sudermann-Merx. *Einführung in Optimierungsmodelle*. Springer Fachmedien Berlin, Heidelberg, 2023, S. 5–14. ISBN: 978-3-662-67381-2. DOI: 10.1007/978-3-662-67381-2. URL: https://doi.org/10.1007/978-3-662-67381-2.

[52] Karl-Dieter Tieste und Oliver Romberg. *Keine Panik vor Regelungstechnik!* Vieweg + Teubner Verlag Wiesbaden, 2012, S. 146–155. ISBN: 978-3-8348-2329-8. DOI: 10.1007/978-3-8348-2329-8. URL: https://doi.org/10.1007/978-3-8348-2329-8.

[53] *Top Entry Immersion Heater*. URL: http://www.eliko.pl/en/products/immersion-heaters/immersion-heaters-with-a-power-of-1400w-and-2000w (besucht am 22. 01. 2024).

[54] *Uhltronix Innovativ, jung & dynamisch*. 2018. URL: https://www.uhltronix.de/unternehmen (besucht am 28. 12. 2023).

[55] *Understanding Deep Learning: DNN, RNN, LSTM, CNN and R-CNN*. 2019. URL: https://medium.com/@sprhlabs/understanding-deep-learning-dnn-rnn-lstm-cnn-and-r-cnn-6602ed94dbff (besucht am 15. 03. 2024).

[56] *Was ist die Dropout Layer?* 2023. URL: https://databasecamp.de/ki/dropout-layer (besucht am 20. 03. 2024).

[57] *Was ist Overfitting?* 2022. URL: https://databasecamp.de/ki/overfitting (besucht am 07. 01. 2024).

[58] Waste. *Regelungstechnik*. 2023. URL: https://rn-wissen.de/wiki/index.php/Regelungstechnik (besucht am 14. 02. 2024).

[59] Wikipedia. *Digitaler Regler*. 2024. URL: https://de.wikipedia.org/wiki/Digitaler_Regler (besucht am 12. 02. 2024).

[60] Wikipedia. *Einschwingzeit*. URL: https://de.wikipedia.org/wiki/Einschwingzeit (besucht am 12. 02. 2024).

[61] Wikipedia. *Faustformelverfahren (Automatisierungstechnik)*. 2024. URL: https://de.wikipedia.org/wiki/Faustformelverfahren_(Automatisierungstechnik) (besucht am 16. 02. 2024).

[62] Wikipedia. *Grey box model*. 2021. URL: https://en.wikipedia.org/wiki/Grey_box_model (besucht am 13. 05. 2024).

[63] Wikipedia. *ITAE*. 2022. URL: https://de.wikipedia.org/wiki/ITAE (besucht am 12. 02. 2024).

[64] Wikipedia. *Künstliches neuronales Netz*. 2023. URL: https://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz (besucht am 08. 01. 2024).

[65] Wikipedia. *Model Predictive Control*. 2022. URL: https://de.wikipedia.org/wiki/Model_Predictive_Control (besucht am 20. 02. 2024).

[66] Wikipedia. *Physics-informed neural networks*. 2024. URL: https://en.wikipedia.org/wiki/Physics-informed_neural_networks (besucht am 13. 05. 2024).

[67] Wikipedia. *Platin-Messwiderstand*. URL: https://de.wikipedia.org/wiki/Platin-Messwiderstand (besucht am 31. 01. 2024).

[68] Wikipedia. *PT1-Glied*. URL: https://de.wikipedia.org/wiki/PT1-Glied (besucht am 02. 02. 2024).

[69] Wikipedia. *Regelstrecke*. 2023. URL: https://de.wikipedia.org/w/index.php?title=Regelstrecke&oldid=236932641 (besucht am 03. 01. 2024).

[70] Wikipedia. *Regler*. 2024. URL: https://de.wikipedia.org/wiki/Regler (besucht am 14. 02. 2024).

[71] Wikipedia. *Rekurrentes neuronales Netz*. 2023. URL: https://de.wikipedia.org/wiki/Rekurrentes_neuronales_Netz (besucht am 15. 03. 2024).

[72] Wikipedia. *Totzeit (Regelungstechnik)*. URL: https://de.wikipedia.org/wiki/Totzeit_(Regelungstechnik) (besucht am 02. 02. 2024).

[73] Daniel Winkler. *Was ist wichtiger: die Anzahl der neuronen pro Schicht oder die Anzahl der Schichten?* Forum Diskussion. 2019. URL: https://de.quora.com/Was-ist-wichtiger-die-Anzahl-der-neuronen-pro-Schicht-oder-die-Anzahl-der-Schichten (besucht am 20. 03. 2024).

[74] Daniel Martin Xavier, Ludovic Chamoin und Laurent Fribourg. *Data-driven MPC applied to non-linear systems for real-time applications*. 2024. URL: https://hal.science/hal-04465225/document (besucht am 05. 03. 2024).