Zentralinstitut für Engineering, Elektronik und
Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

Realisierung eines Temperatur-reglers mittels MQTT-basierter Sensordatenübertragung

Dominik Hoven

Jül-4417

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Zentralinstitut für Engineering, Elektronik und Analytik (ZEA) · Systeme der Elektronik (ZEA-2)

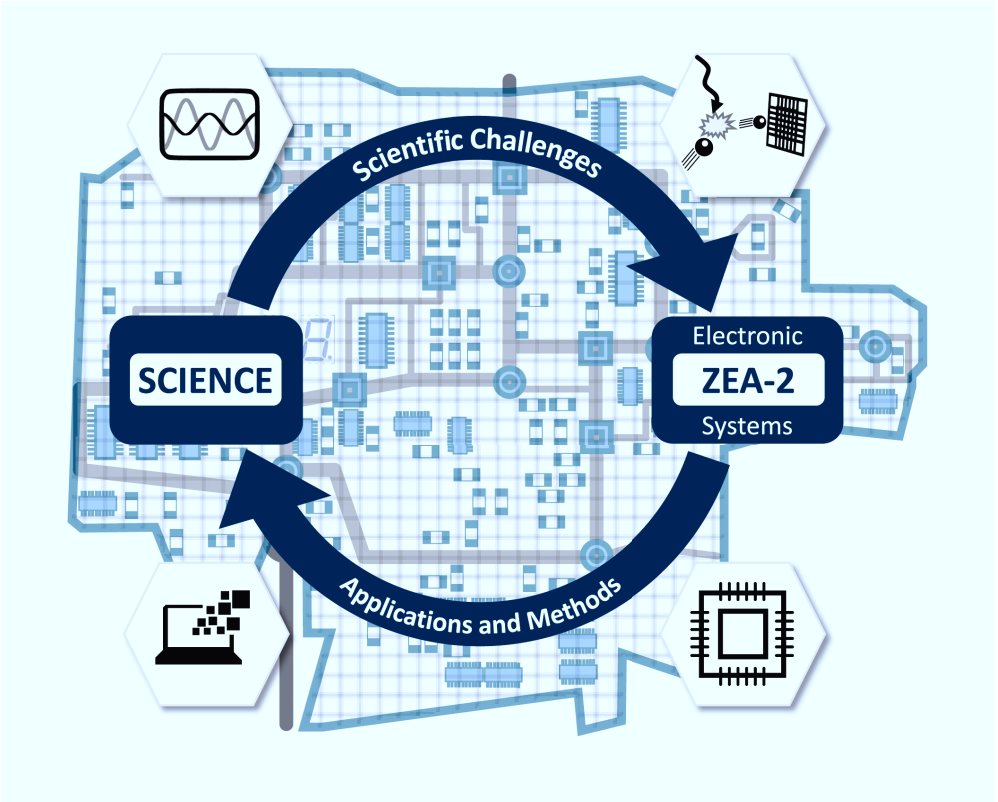
Realisierung eines Temperatur-reglers mittels MQTT-basierter Sensordatenübertragung

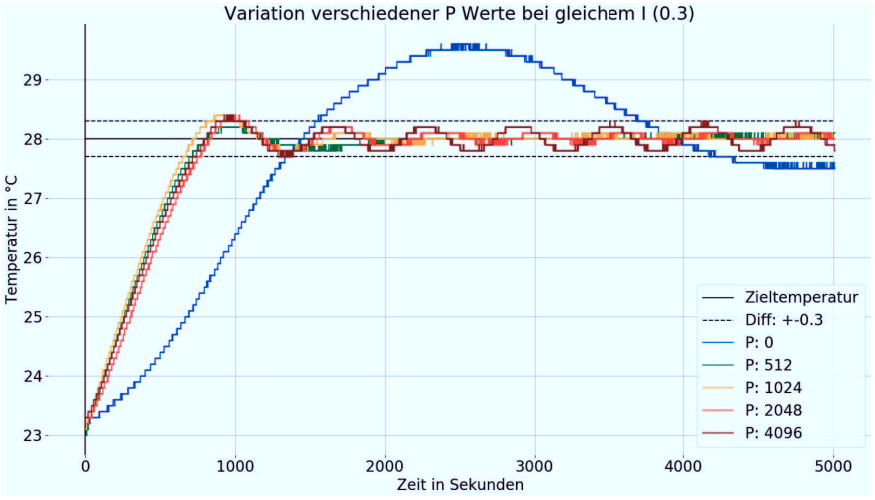
Dominik Hoven

Berichte des Forschungszentrums Jülich
Jül-4417 · ISSN 0944-2952
Zentralinstitut für Engineering,
Elektronik und Analytik (ZEA)
Systeme der Elektronik (ZEA-2)
(Bachelor, FH Aachen, Campus Jülich, 2018)

Vollständig frei verfügbar über das Publikations-
portal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess

Forschungszentrum Jülich GmbH · 52425 Jülich
Zentralbibliothek, Verlag
Tel.: 02461 61-5220 · Fax: 02461 61-6103
zb-publikation@fz-juelich.de
www.fz-juelich.de/zb


Scientific Challenges
SCIENCE
Electronic
ZEA-2
Systems
Applications and Methods


Variation verschiedener P Werte bei gleichem I (0.3)
Temperatur in °C
Zeit in Sekunden
29
28
27
26
25
24
23
0
1000
2000
3000
4000
5000
Zieltemperatur
Diff: +-0.3
P: 0
P: 512
P: 1024
P: 2048
P: 4096

Kurzfassung

Diese Arbeit hat zum Ziel, einen Temperaturregler zu implementieren, der die Informationsdaten für den Regler aus einem Sensornetz, das auf dem Message Queue Telemetry Transport Protokoll basiert, bezieht. Da der Regler später in einem hochgradig elektromagnetisch empfindlichen Versuchsaufbau genutzt werden soll, entstehen zusätzliche Anforderungen an den Aufbau des Systems. So wird das System in eine erste Einheit, in der sich später der empfindliche Experimentaufbau mit möglichst wenig Elektronik befindet, und eine zweite Einheit, in der sich die Bauteile befinden, die zum Heizen benötigt werden, aufgeteilt.
Nach einer Beschreibung des MQTT-Protokolls wird auf die Implementierung eingegangen. Um die elektromagnetische Störung gering zu halten, basiert das Sensornetz auf ESP8266 Mikrocontrollern. Diese besitzen ein integriertes WLAN-Modul, wodurch auf ihnen das MQTT-Protokoll genutzt werden kann. So wird auf diesen die uMQTTBroker-Bibliothek, die eine Client- und Broker-Funktionalität für diesen Mikrocontroller bereitstellt, installiert, die auf GitHub zur Verfügung gestellt wird. Darauf folgt die Validierung der Anzahl möglicher Clients, die in der Dokumentation der Bibliothek angegebenen wird. Bei dieser wurde festgestellt, dass es nicht möglich war, wie in der Bibliothek angegeben, acht Clients zu nutzen. Stattdessen konnten nur fünf Clients genutzt werden. Darauf wurde mit diesen Clients die minimale Übertragungsperiodendauer von 500 ms bestimmt.
Nach einer Beschreibung verschiedener Reglerarten wird das vorliegende System analysiert und ein passender Regler gewählt. Aufgrund der langen Totzeit und der sich nicht schnell ändernden Temperaturen wurde ein Proportional-Integral-Regler gewählt. Dieser und die Steuerung des Messsystems wurden in Python realisiert.
Zuletzt mussten die Regel-Parameter ermittelt werden. Nachdem diese gefunden wurden, konnte der Regler in diesem System getestet werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

In diesem Kapitel wird zunächst das Zentralinstitut für Engineering, Elektronik und Analytik vorgestellt. Hiernach wird auf die Motivation eingegangen.

1.1. ZEA

Das Zentralinstitut für Engineering, Elektronik und Analytik (ZEA) des Forschungszentrum Jülich arbeitet mit Wissenschaftlern aus aller Welt zusammen. Als Technologielieferant für das Forschungszentrum Jülich ermöglichen die entwickelten Geräte, Verfahren, Anlagen und Systeme Spitzenforschung zu betreiben.
Als „Systeme der Elektronik“, kurz auch ZEA-2, wird der Institutsbereich bezeichnet, in dem komplexe elektronische und informationstechnische Systemlösungen entwickelt werden. Dabei erfolgen Kooperationen mit Wissenschaftlern, Universitäten und anderen wissenschaftlichen Einrichtungen weltweit. Neben der erfolgreichen Ausbildung in technischen und naturwissenschaftlichen Berufen werden auch Doktoranden betreut.
In dem Kompetenzteam „Softwareentwicklung“ werden Softwarelösungen geplant und implementiert. Dabei muss häufig auf spezialisierte Hardwarelösungen zur Datenerfassung und Steuerung der komplexen wissenschaftlichen Aufgabestellungen eingegangen werden. [1]

1.2. Motivation

In dem Projekt EMI [2] werden elektromagnetische Induktionsmessgeräte zur Untersuchung von Böden entwickelt. Da Halbleiterbauelemente und Messspulen verwendet werden, können aufgrund von Temperaturdrifts Veränderungen in den Messdaten entstehen. Eine möglichst gleichbleibende Temperatur sollte diese Veränderungen während der Messungen verhindern. Dafür wird eine Temperaturregelung benötigt. Da die entwickelten Messgeräte hoch elektromagnetisch sensibel sind, sollen so wenig elektronische Bauteile wie möglich in deren Nähe stehen. Aus diesem Grund müssen Messgeräte und Temperaturregelung getrennt aufgebaut werden. Sie werden in zwei Styroporboxen, die mit zwei Schläuchen verbunden sind, untergebracht.
Der realisierte Temperaturregler soll die Temperatur innerhalb von 2000 Sekunden auf eine Zieltemperatur mit einer Abweichung von maximal $\pm 0.3\,^{\circ}\mathrm{C}$ bringen. Zudem soll die Zieltemperatur mit einer möglichst geringen Abweichung gehalten werden. Aufgrund der Sensorungenauigkeit ist dabei ein Intervall von $\pm 0.1\,^{\circ}\mathrm{C}$ ausreichend.

2. Versuchsaufbau

In diesem Kapitel wird zunächst die Hardware und darauf der Versuchsaufbau zum Testen des Sensornetzwerkes beschrieben. Danach wird auf den Aufbau des Regelsystems eingegangen.

2.1. Elemente

Beide Versuchsaufbauten bestehen aus verschiedenen Bauelementen. Diese werden zuerst im Folgenden beschrieben.

2.1.1. Adafruit Huzzah Feather Board

Bei dem in Abbildung 2.1 gezeigten Adafruit Huzzah Board handelt es sich um ein sogenanntes „All-in-One“-Entwicklungsboard, auf dem sich ein ESP8266 Microcontroller befindet.

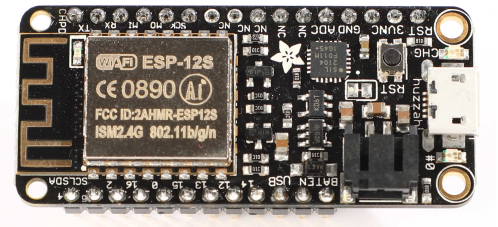


Abbildung 2.1.: Ein Adafruit Huzzah Feather Board.

Der in Abbildung 2.2 abgebildete Microcontroller besitzt einen WLAN-Chip, über den die Kommunikation mit dem MQTT-Netzwerk möglich wird. Der Microcontroller besitzt einen 32-Bit Prozessorkern und läuft mit einem Systemtakt von 80 MHz bis 160 MHz.

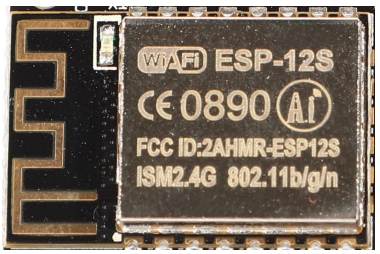


Abbildung 2.2.: Ein ESP-12S WiFi-Modul auf einem Adafruit Huzzah Feather Board (Ausschnitt aus Abbildung 2.1).

Das Board besitzt ein eingebautes USB- und Batterieladegerät. Zudem ist ein USB-Seriell-Konverter in das Board integriert, der durch maximal 921600 Baud[1] einen schnellen Upload ermöglicht. Das Board ist 52 mm ∗ 23 mm ∗ 8 mm groß und wiegt 9.7 g. [3]

2.1.2. DHT22

Der in Abbildung 2.3 abgebildete DHT22-Sensor ist ein digitaler Temperatur- und Luftfeuchtesensor. Der messbare Temperaturbereich reicht von −40 bis +80 °C, wobei die Temperaturmessgenauigkeit ±0.5 °C und die Temperaturauflösung ±0.1 °C beträgt. Der Messbereich für die Luftfeuchte reicht von 0 bis 100 % relative Luftfeuchte. Dabei wird mit einer Messgenauigkeit von ±2 % RH[2] gemessen. [4]

Abbildung 2.3.: Ein DHT22-Sensor.

2.1.3. Lithium Ionen Polymer Batterie

In Abbildung 2.4 ist eine handelsübliche einzellige Lithium Ionen Polymer Batterie zu sehen. Der Anschluss dieser Batterie an das Adafruit Huzzah Board erfolgt über einen zweipoligen JST-Stecker. Bei einer Spannung von 3.7 V hat die Batterie eine elektrische Ladung von 1.1 Ah. Zudem besitzt die Batterie einen eingebauten Schutz gegen Über- und Unterspannung. Die Batterie wiegt 22.2 g bei einer Größe von 61 mm ∗ 37 mm ∗ 5 mm. [5]

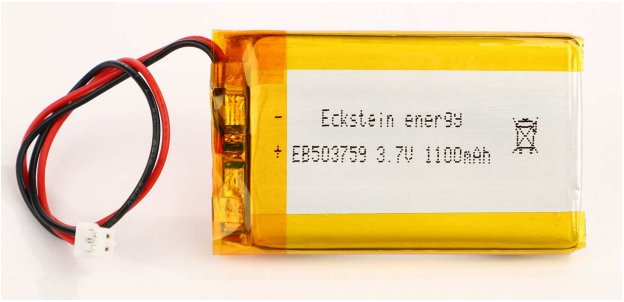


Abbildung 2.4.: Eine Lithium Ionen Polymer Batterie.

[1] Baud ist die Einheit für die Symbolrate und ist ein Maß für die Übertragungsgeschwindigkeit. Wenn 1 Symbol pro Sekunde übertragen wird, ist die Geschwindigkeit 1 Baud. Unter einem Symbol wird dabei eine Änderung des Signals verstanden.

[2] RH steht für „relative humidity“, Relative Luftfeuchtigkeit

2.1.4. Heizwiderstand

Bei dem Heizwiderstand handelt es sich um zwei in Reihe geschaltete Widerstände, die auf einem Kühlkörper befestigt sind.

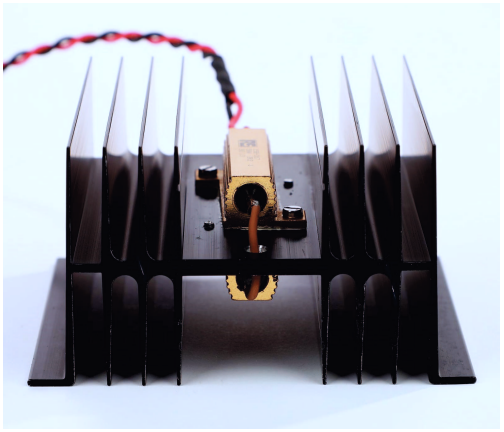


Abbildung 2.5.: Der Heizwiderstand.

Dieser ist in Abbildung 2.5 dargestellt. Da jeder Widerstand einen ohmschen Widerstand von 2.2Ω besitzt und für eine Leistung von 50 W ausgelegt ist, besitzt der Heizwiderstand einen ohmschen Widerstand von 4.4Ω und kann für eine Leistung von 100 W genutzt werden.
Zu der Steuerung der mittleren elektrischen Leistung des Heizwiderstandes wird eine Pulsweitenmodulation (PWM) genutzt. Bei dieser Modulationsart handelt es sich um eine Zeitmodulation, bei der zwischen 100% (An) und 0% (Aus) gewechselt wird. Dazu wird bei einer konstanten Frequenz die Breite der Impulse verändert.

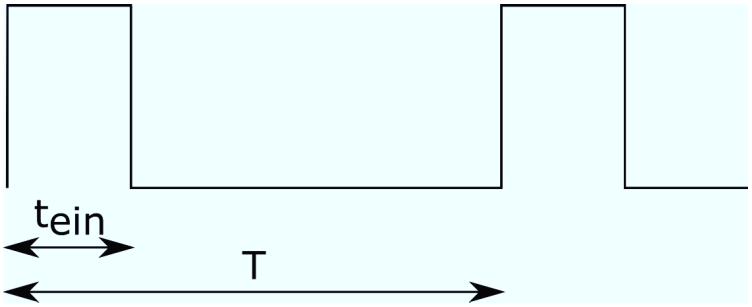


Abbildung 2.6.: Beispiel eines PWM-Signals mit einem Tastgrad von 25%.

In der Abbildung 2.6 ist als Beispiel ein PWM-Signal mit einem Tastgrad von 25% zu sehen.

$$\frac{t_{ein}}{T} = 0.25 = 25\% \tag{2.1}$$

Die Rechnung für das Tastverhältnis ist in (2.1) dargestellt. Dabei ist T die Periodendauer und t_{ein} die Einschaltzeit. Die Einschaltzeit ist die Zeit in einem Zyklus, in dem das Signal auf „high“ ist. Für die Leistung bedeutet dies, dass in t_{ein} die Leistung 100% und in der restlichen Zeit bis zu dem Periodenende die Leistung 0% ist. Die mittlere elektrische Leistung

kann durch wie in Gleichung (2.2) beschrieben berechnet werden. Dabei ist P_{HM} die mittlere elektrische Leistung des Heizwiderstandes für eine Periode T, $P_H(t)$ die zeitabhängige Leistung und P_{Max} die maximale Leistung.

$$P_{HM} = \frac{1}{T} \cdot \int_0^T P_H(t)dt = \frac{1}{T} \cdot (\int_0^{t_{ein}} P_H(t)dt + \int_{t_{ein}}^T P_H(t)dt) = 25\% \cdot P_{Max} \quad (2.2)$$

Zur Verstärkung PWM-Signals des Boards wird der in Abbildung 2.7 abgebildete Schaltverstärker benötigt.

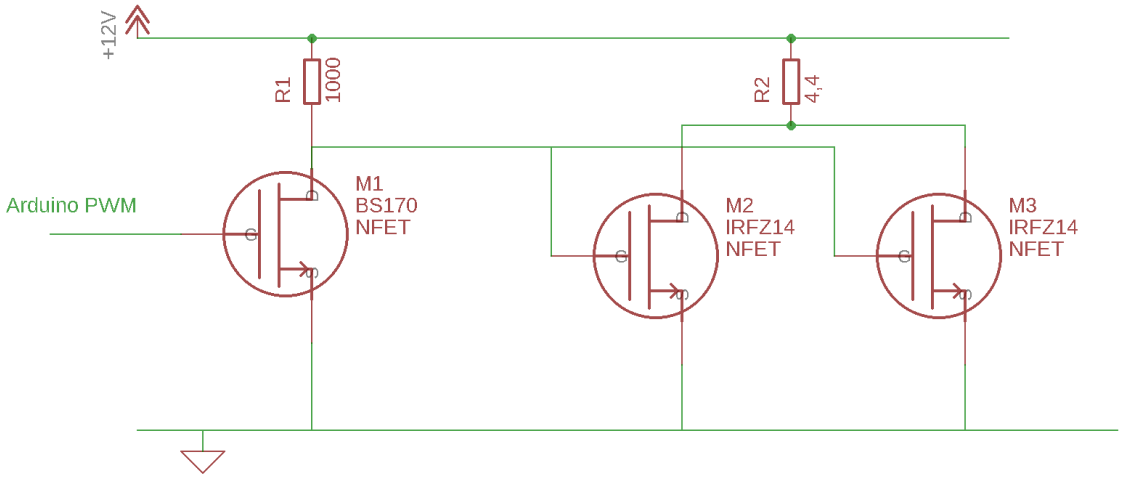


Abbildung 2.7.: Ein Schaltverstärker.

2.1.5. Bürstenloser Gleichstrommotor in einem 3D-gedruckten Gehäuse

Der bürstenlose Gleichstrommotor, auch Brushless DC Motor oder BLDC-/BL-Motor, ist ein Motor, der auf dem Funktionsprinzip einer Drehstrom-Synchronmaschine mit Erregung durch Permanentmagnete basiert. Der Rotor ist bei diesen Motoren mit Permanentmagneten versehen und die Spulen befinden sich in einem fixierten Stator. Bei dem verwendeten Motor wurde eine Außenläuferform gewählt. Sie zeichnet sich durch eine hohe Effizienz aus.

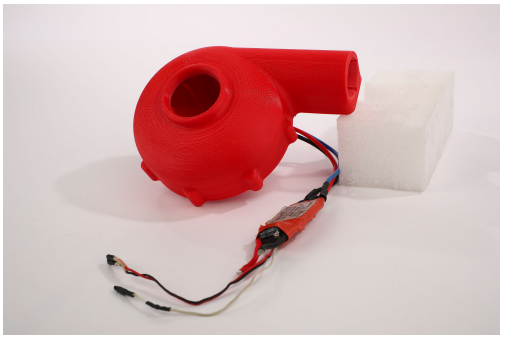


Abbildung 2.8.: Außenläufermotor in einem 3D-gedruckten Gehäuse.

Der in das 3D-gedruckte Gehäuse eingebaute Motor ist in Abbildung 2.8 zu sehen.
Dieses Gebläse wird zum Lufttransport innerhalb des Systems genutzt. Durch die Bauform des Gehäuses und der Motorart ist es besonders gut geeignet, da es einen erhöhten Druck ausüben kann. Dadurch kann es die Luft besonders gut durch die Schläuche drücken. Dies wird benötigt, um eine Luftzirkulation in dem Aufbau zu ermöglichen.

2.1.6. WLAN-Modul

Das genutzte WLAN-Modul stammt von der Firma *tp-link* und trägt die Bezeichnung „TL-WR702N“. Es ist ein tragbarer 150Mbit/s-WLAN-Nano-Router, der mehrere Betriebsarten für unterschiedliche Einsatzszenarien bereit hält. In diesem Fall wurde die Betriebsart „Access Point“ gewählt. Das Modul hat eine Größe von $57\,\text{mm} * 57\,\text{mm} * 18\,\text{mm}$. Die Abbildung 2.9 zeigt dieses WLAN-Modul. [6]

Abbildung 2.9.: Ein WLAN-Modul.

2.2. Aufbau der Tests des Sensornetzes

Zum Test des Sensornetzes werden mehrere Adafruit Huzzah Boards verwendet. Minimal sind dazu vier Boards für MQTT-Broker, den Auslese-, Heiz- und -Sensor-Client nötig. Die genauen Aufgaben der verschiedenen Clients werden in Kapitel 3.2.2 erklärt.

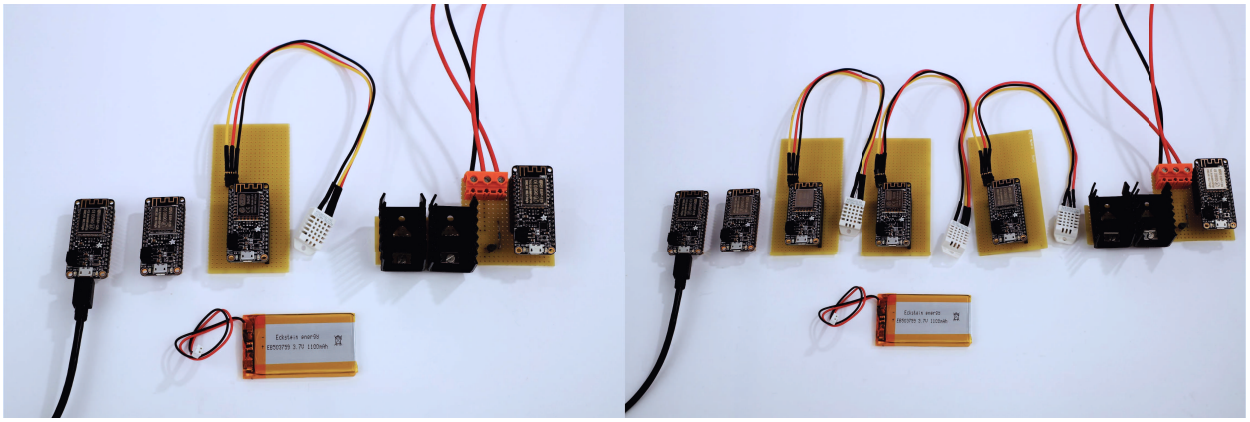


Abbildung 2.10.: Links: Minimaler, rechts: maximaler Aufbau.

In Abbildung 2.10 ist in der linken Bildhälfte der minimale und in der rechten der maximale Aufbau gezeigt. Der minimale Aufbau besteht aus: Auslese-Client, MQTT-Broker, Sensor-Client und Heiz-Client (v.l.n.r). Von dem Auslese-Client geht ein USB-Kabel ab. Dies zeigt, dass er an einen PC angeschlossen ist, von dem die Stromversorgung übernommen wird. Die anderen Boards werden von einer Batterie mit Strom versorgt. Dabei besitzt jedes Board eine eigene Batterie. Aus Zwecken der Übersichtlichkeit im Bild wurde davon abgesehen, je eine Batterie darzustellen. Stattdessen wurde eine unangeschlossene Batterie zur Verdeutlichung hinzugefügt. In diesen minimalen Aufbau können optional weitere Publisher hinzugefügt werden. So werden in dem maximalen Aufbau drei Sensor-Clients gezeigt.

2.3. Temperaturregelungsaufbauten

Die beiden im Folgenden beschriebenen Aufbauten wurden für Tests der Temperaturregelung verwendet.

2.3.1. Grundaufbau

Der Grundaufbau bei den Tests der Temperaturregelung lässt sich in zwei Seiten aufteilen. Diese werden im folgenden beschrieben. In Abbildung 2.11 ist der Grundaufbau einmal ohne und einmal mit Deckel zu sehen. Die Verbindung der beiden Boxen fehlt.

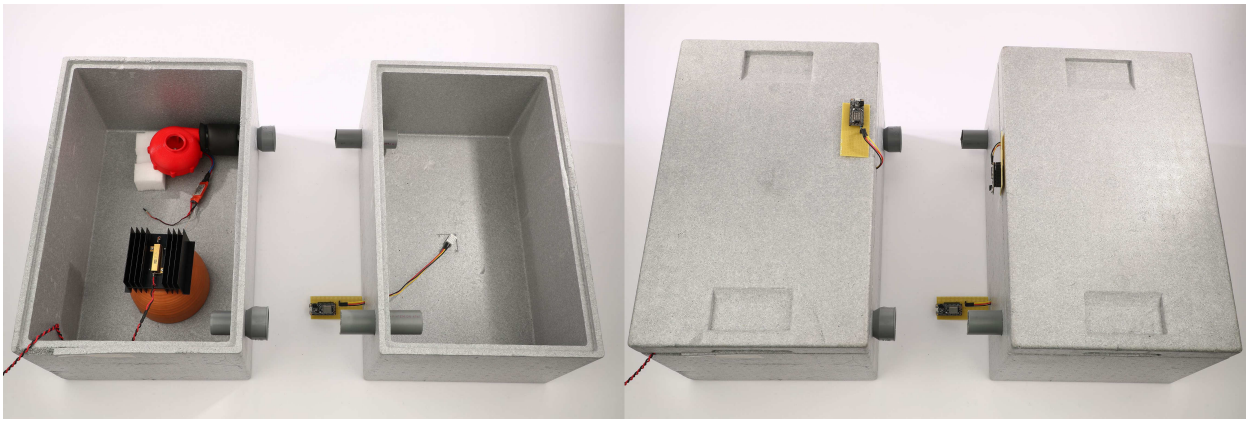


Abbildung 2.11.: Der Grundaufbau ohne eine Verbindung der Boxen.

2.3.1.1. Heiz-Seite

Die Heiz-Seite beinhaltet die Komponenten, die zur Heizung des Versuchaufbaus benötigt werden. Alle Komponenten dieser Seite sind in einer handelsüblichen Styroporbox, die auch als Thermobox Verwendung findet, untergebracht. Diese Box dient dem Erhalt der schon dem System zugefügten Wärme. In dieser Box befindet sich, neben dem Gebläse, auch der Heizwiderstand. Das Gebläse befindet sich im oberen Teil der Box, wohingegen der Heizwiderstand auf einem Blumentopf aus Ton steht, damit er in einiger Entfernung zu dem Styropor steht. Dies soll vermeiden, dass das Styropor anfängt zu schmelzen. Die Heiz-Seite ist in Abbildung 2.12 dargestellt.

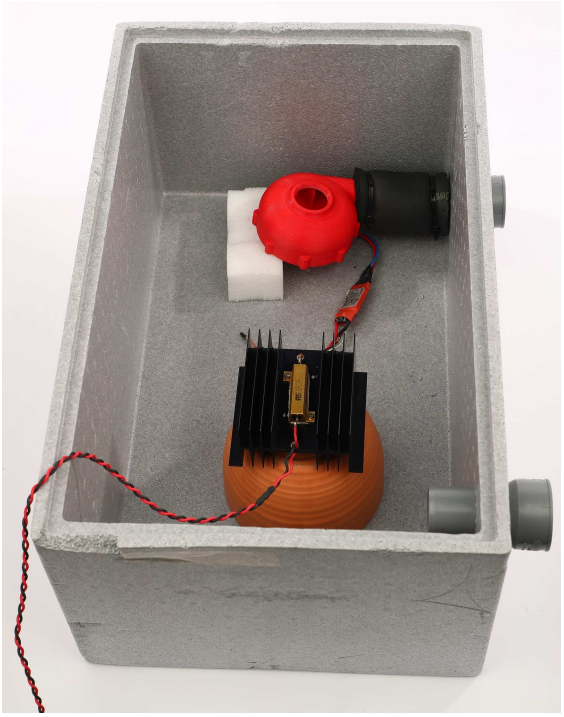


Abbildung 2.12.: Die Heiz-Seite des Grundaufbaus.

In dem Deckel dieser Box ist ein DHT22-Temperatursensor befestigt. Das zugehörige Board befindet sich außerhalb dieser Box. Dies ist in Abbildung 2.13 zu sehen.
Das Board, das die Steuerung des Heizwiderstandes übernimmt, befindet sich, wie auch die Elektronik, die zur Steuerung des Motors genutzt wird, außerhalb der Styroporbox. Sie sind vorerst an ein Netzteil angeschlossen. Dieses ist so eingestellt, dass es die Leistung der später genutzten Batterien (zur Betreibung des Heizwiderstandes und des Motors) ausgibt.

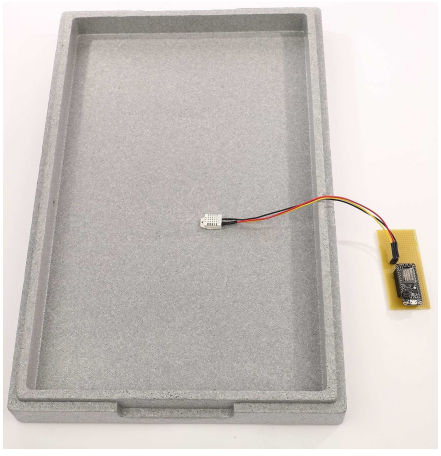


Abbildung 2.13.: Der Deckel der Styroporbox im Grundaufbau.

2.3.1.2. Experiment-Seite

Die Experiment-Seite besteht aus zwei ESPs, deren DHT22-Sensoren sich in einer Styroporbox befinden, die der in Kapitel 2.3.1.1 genannten Box gleicht. Um messen zu können, ob ein Temperaturunterschied zwischen Boxdeckel und Boden besteht, wurden die DHT22-Sensoren entsprechend an Boden und Deckel befestigt.

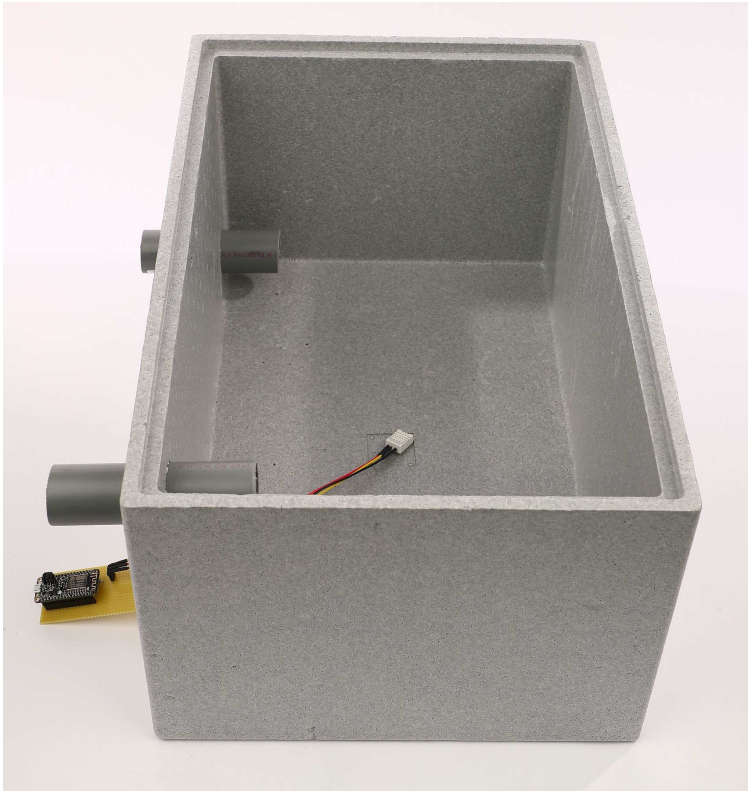


Abbildung 2.14.: Die Experiment-Seite des Grundaufbaus.

In der Abbildung 2.14 ist zu sehen, wie der DHT22 am Boden der Box befestigt ist. Auch hier befindet sich nur der Sensor selbst in der Box, nicht aber das gesamte Board. Der Deckel ist, wie schon in Abbildung 2.13 gezeigt, aufgebaut.
Da das Gebläse viel Luft in diese Box blasen kann, kann der Deckel herausgedrückt werden. Dadurch entweicht bereits erwärmte Luft. Um dies zu vermeiden, wird der Deckel der Box beschwert.

2.3.1.3. Verbindung

Die Verbindung beider Seiten wird durch Isolierschläuche ermöglicht. Dazu wurden in beide Styroporboxen jeweils zwei Löcher gebohrt, in die jeweils ein Rohraufsatz gesteckt wurde. Auf diesen können dann die Schläuche gesteckt werden.
Zwischen den beiden Boxen ist zudem das WLAN-Modul und der ESP, auf dem der MQTT-Broker läuft, befestigt, um die optimale Erreichbarkeit von beiden Seiten zu ermöglichen.

2.3.2. Aufbau für Störung

Dieser Aufbau baut auf dem in Kapitel 2.3.1 beschriebenen Aufbau auf. So wird der schon beschriebene Aufbau um eine weitere Styroporbox erweitert.
In dieser großen Styroporbox steht die Experiment-Seite, welche in Kapitel 2.3.1.2 aus dem Grundaufbau thematisiert wurde. Dadurch wird die Simulation einer Störung ermöglicht. Um das Scheinen der Sonne auf die Styroporbox zu simulieren, wodurch sich die Styroporbox unweigerlich erwärmt, wird die Luft zwischen der äußeren und der inneren Box mit einem Heizföhn erwärmt. In Abbildung 2.15 sind beide Styroporboxen zu sehen.

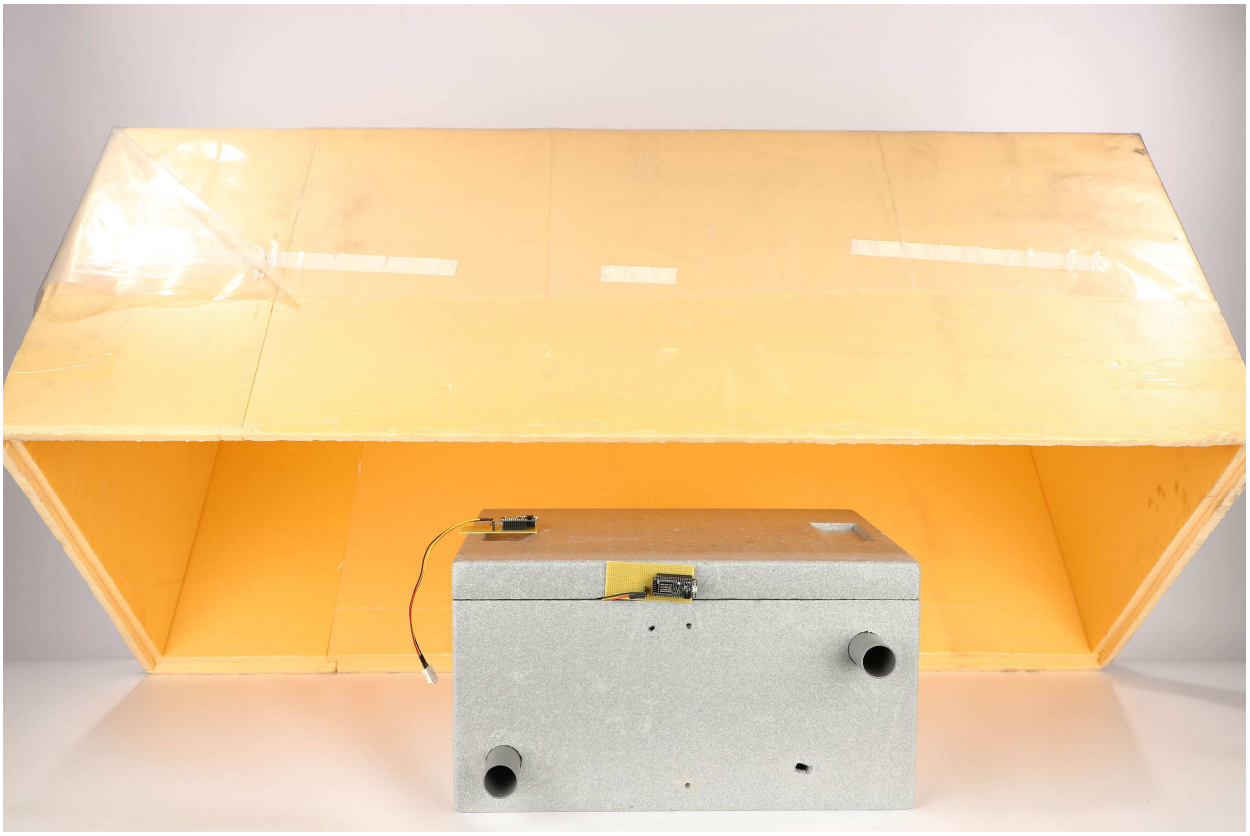


Abbildung 2.15.: Die Erweiterung Grundaufbaus für Störungen.

3. MQTT-Sensornetz

In diesem Kapitel wird auf das MQTT-Sensornetz eingegangen. Nach einer kurzen Einführung in das MQTT-Protokoll wird die Realisierung des MQTT-Sensornetzes beschrieben.

3.1. MQTT-Protokoll

Das Message Queue Telemetry Transport Protokoll (MQTT) ist ein Nachrichtenprotokoll, das zur Machine-to-Machine-Kommunikation[7] als auch im Internet-of-Things (IoT)[8] genutzt wird. Im ISO[1]-OSI[2]-Modell agiert das Protokoll in der Anwendungsebene, während es auf dem Übertragungssteuerungsprotokoll/Internetprotokoll (TCP/IP) ausgeführt wird.

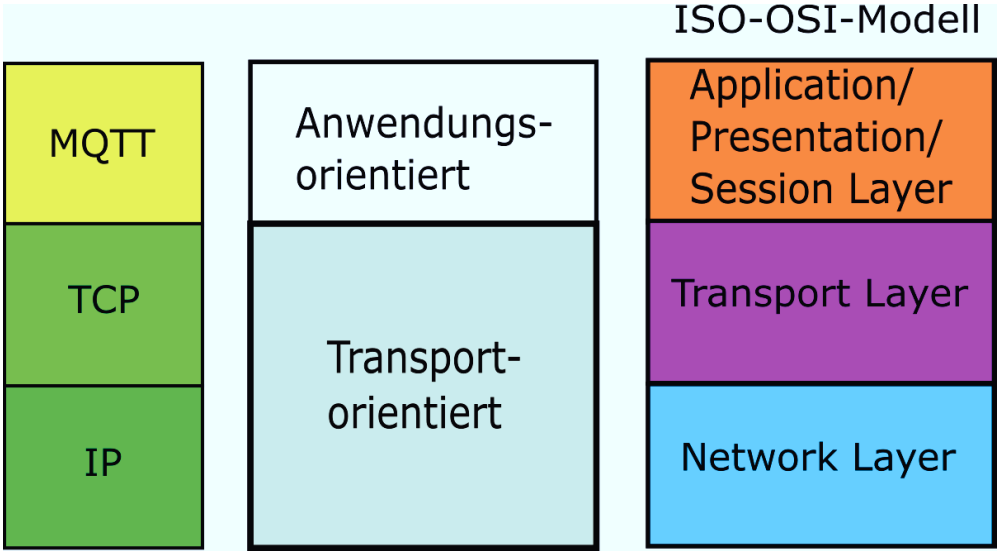


Abbildung 3.1.: MQTT über dem TCP- und IP-Stack.

In der Abbildung 3.1 wird verdeutlicht, wie die Position von MQTT im ISO-OSI-Modell ist. Dabei ist MQTT anwendungsorientiert und TCP/IP transportorientiert. MQTT beinhaltet die Schichten 5-7 (Application/ Presentation/ Session Layer), TCP die 4. Schicht (Transport Layer) und IP die 3. Schicht (Network Layer) des ISO-OSI-Modells.
Zur Datenübertragung nutzt das Protokoll eine brokerbasierte Publish-Subscribe-Architektur[9]. Bei dieser Architektur werden Clients, die Daten empfangen wollen, als Subscriber bezeichnet. Sie registrieren sich bei dem Broker, über den alle Daten transferiert werden, für ein Thema. Dieser Broker wird in verschiedenen Versionen auch als Server bezeichnet. Der Client kann jedoch auch mehrere Themen subscriben. Der Broker leitet anschließend dem Subscriber die Nachricht weiter, wenn der Broker eine Nachricht zu einem Thema erhält,

[1] International Organization for Standardization (Internationale Organisation für Normung).
[2] Open System Interconnection (Offenes System für Kommunikationsverbindungen).

das der Subsriber subscribt hat. Auf der anderen Seite gibt es, neben dem Subscriber, auch einen sogenannten Publisher.
Der Publisher produziert die Daten und sendet sie mit einem bestimmten Thema an den Broker, der diese daraufhin weiterleitet.

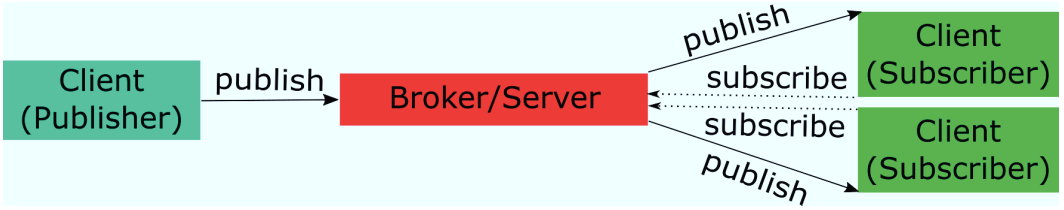


Abbildung 3.2.: Schematischer Beispielaufbau einer Publish-Subscribe-Architektur.

In Abbildung 3.2 wurde die Architektur beispielhaft dargestellt, um ein besseres Verständnis zu ermöglichen. Dabei sind mehrere Clients mit einem Server verbunden. Ein Client ist Publisher und die anderen Clients sind Subscriber. Der Server leitet die Daten an die Subscriber weiter.
Einer der großen Vorteile der brokerbasierten Publish-Subscribe-Architektur ist, dass der Datenproduzent seine Abnehmer nicht kennen muss, da die Datenverteilung über den Broker geregelt wird. Es müssen sich jedoch alle Clients, egal ob Publisher oder Subscriber, mit dem Broker verbinden. Dadurch ist es mit einem leistungsfähigen Server möglich, dass ein leistungsschwacher Datenproduzent Daten publisht und diese Tausende oder mehr Interessenten erreicht. [10]
Ein Thema, unter dem Daten gepublisht werden können, kann in Unterthemen eingeteilt werden. Dies ist durch das Trennzeichen „/“möglich. Das Trennzeichen erlaubt die genauere Spezifizierung der an den Interessenten zu übertragenden Daten. So können übergeordnete Themen subscribt werden, wodurch die Daten aller zugehörigen Unterthemen zusätzlich dem Interessenten übermittelt werden.
Wenn Nachrichten verschickt werden, kann das „Retained Flag“ auf „true“ gesetzt werden. Die Nachricht wird dann als „Retained Message“ bezeichnet. Die letzte „Retained Message“ und die zugehörige Quality-of-Service-Ebene eines Themas werden von dem Broker gespeichert. Meldet sich ein Client neu für ein Thema an, für das eine „Retained Message“ von dem Broker gespeichert wurde, wird diese dem Client weitergeleitet.
Bei der Übermittlung der Daten ist es von dem Protokoll vorgesehen, dass es unterschiedliche Quality-of-Service-Stufen gibt. In der genutzten MQTT-Bibliothek wurde bisher nur die QoS-Stufe 0 implementiert. Daneben gibt es noch zwei weitere QoS-Stufen. Im folgenden sind die verschiedenen Stufen aufgelistet.

0. Höchstens einmalige Lieferung: Diese QoS-Ebene bietet die gleiche Garantie wie das zugrunde liegende TCP-Protokoll. Sie wird auch häufig mit „fire and forget“ beschrieben, da der Absender die Nachricht nur an den Empfänger schickt und nichts weiter dabei passiert. Im Vergleich mit den anderen QoS-Ebenen besitzt diese den geringstmöglichen Overhead.

1. Mindestens einmalige Lieferung: Diese QoS-Ebene fügt dem Ziel, das die Nachricht erhalten soll, eine Bestätigungsanforderung hinzu. Dadurch wird gewährleistet, dass die Nachricht mindestens einmal an den Empfänger zugestellt wird. Es kann jedoch auch sein, dass dadurch Duplikate entstehen.

2. Genau einmalige Lieferung: Diese QoS-Ebene garantiert, dass die Nachricht nur einmal an den Empfänger zugestellt wird. Diese QoS-Stufe hat den höchsten Overhead im Vergleich zu den beiden anderen Stufen. Sie erfordert zwei Datenübertragungen zwischen Sender und Empfänger.

Es ergibt sich also, dass die QoS-Stufe auf die Hardware und die Daten, die übertragen werden sollen, abgestimmt sein sollte. Sollten Publisher und Subscriber unterschiedliche QoS-Anforderungen haben, muss der MQTT-Broker die QoS-Stufe auf die niedrigste von dem Subscriber angeforderte QoS-Stufe herabstufen.
MQTT besitzt eine sogenannte Testamentfunktion. Durch das Setzten von Flags bei einer Nachricht kann diese als eine „WillMessage“ von dem Broker interpretiert werden. Diese wird unter einem in der Nachricht angegebenen Thema von dem Server gepublisht, wenn der MQTT-Client von dem Server getrennt wird oder die Verbindung zum MQTT-Server abbricht. Dabei wird die in der Nachricht festgelegte QoS-Stufe von dem Server verwendet. [11]

3.2. Realisierung des MQTT-Sensornetzes

Die Realisierung des MQTT-Sensornetzes basiert von der Hardwareseite auf Adafruit Huzzah Feather Boards und einem Modul der Firma *tp-link*, das den WLAN Access Point bereitstellt.
Die Softwareseite basiert auf der uMQTTBroker-Bibliothek [12] des GitHub Nutzers „martin-ger“ [13]. Diese erweitert die MQTT-Client-Bibliothek [14] des GitHub Nutzers „Tuan PM“ [15]. Neben der Einarbeitung einiger MQTT-Broker Funktionen hat „martin-ger“ die schon bestehenden Client-Funktionalitäten mit in seine Bibliothek übernommen. Diese wurden neben Funktionen des ESP8266-Systems für die Clients genutzt. In Abbildung 3.3 ist grafisch verdeutlicht, wie die Beziehungen der entwickelten Software aussehen.

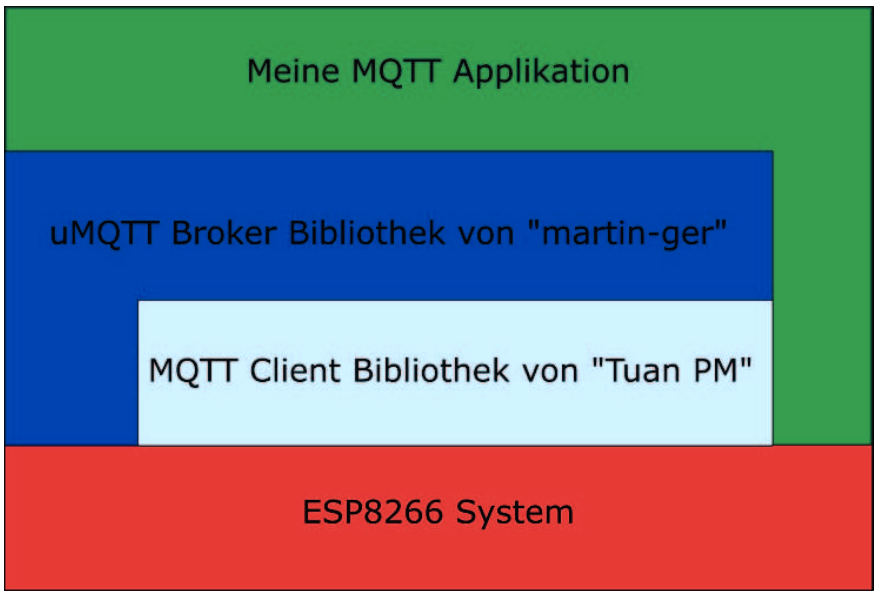


Abbildung 3.3.: Verschiedenen Beziehungen in der Software.

In dem GitHub-Repository [12] zu der uMQTTBroker-Bibliothek werden die unterstützten und nicht unterstützten Funktionen aufgeführt. Bei der folgenden Aufzählung handelt es sich um die nicht unterstützten Funktionen.

- andere QoS-Stufen als 0
- viele TCP (=MQTT) Clients
- nicht persistente Verbindungen
- keine TLS (Transport Layer Security)

Bei diesen nicht unterstützten Funktionen ist die Einschränkung der Clientanzahl am gravierendsten, gefolgt von der Einschränkung auf die QoS-Stufe 0. Dies ist darin begründet, dass die Anzahl der Clients die Anzahl von Sensormodulen und die QoS-Stufen die Datenübertragungssicherheit limitiert. Die Anzahl der benötigten Clients liegt jedoch in dem Rahmen der unterstützten Verbindungen und die Sensordaten sind durch die Aufnahme von tausenden Werten, welche sich nicht sehr schnell in ihrer Größe ändern, übertragungskritisch.

3.2.1. MQTT-Broker

Bei diesem Sketch[3] handelt es sich um die Implementierung des MQTT-Broker. Er wird auf den ESP aufgespielt und übernimmt die schon beschriebene Aufgabe der Datenverteilung. Der Grundaufbau der Implementierung konnte aus dem Beispielsketch von GitHub [16] übernommen werden.
In dem Sketch muss noch der Name und das WLAN-Passwort umgesetzt werden. In dem Versuch handelte es sich hierbei um die Daten des WLAN-Moduls.

Listing 3.1: Broker IP-Variablen

```
23 IPAddress ip(192, 168, 21 ,42);
24 IPAddress subnet(255, 255, 255, 0);
25 IPAddress gt(192, 168, 21, 1);
```

Zudem wurden Variablen für IP-Adresse, Subnetz und den Gateway, wie in Listing 3.1 gezeigt ist, hinzugefügt. Mit diesen kann in der *setup*-Funktion das WiFi und der MQTT-Broker konfiguriert werden. Zudem wird in dieser Funktion der WiFi-Modus des ESP auf „station“ gesetzt. Darauf kann die Verbindung mit dem WLAN begonnen und solange gewartet werden, bis sich der Client mit dem WLAN verbunden hat.

Listing 3.2: Broker IP-Variablen

```
56   WiFi.config(ip, gt, subnet);
57   WiFi.mode(WIFI_STA);
58   WiFi.begin(ssid, pass);
59   while (WiFi.status() != WL_CONNECTED) {
60     delay(500);
61     Serial.print(".");
62   }
```

[3] In der Arduino IDE ist ein Sketch ein Programm, welches durch die Umgebung auf einen Mikrocontroller geladen wird. Es besteht im Grundaufbau aus einer *setup*- und einer *loop*-Funktion.

Dies ist in Listing 3.2 gezeigt. Da es von keinem Interesse ist, dass der Broker eine aufsteigende Zahl, die sich jede Sekunde erhöht, publisht, konnte der Inhalt der *loop*-Funktion des Beispiel Broker Sketches gelöscht werden.
Die Publish-Subscribe-Architektur sieht nur einen einzigen Broker vor. Aus diesem Grund gibt es auch hier nur einen einzigen ESP, auf dem die Software für den Broker läuft.

3.2.2. MQTT-Clients

Neben dem Broker gibt es noch die MQTT-Clients. Diese sind in der Realisierung des Sensornetzes in drei Arten unterteilbar. Ein Beispielsketch zu einem MQTT-Client konnte in dem GitHub-Repository von „i-n-g-o“(Ingo Randolf)[17] gefunden werden.
Die genauen Unterschiede zu dem Beispielsketch werden in jedem Unterkapitel einzeln aufgeführt.

3.2.2.1. Sensor-Client

Der Sensor-Client publisht die Temperatur- und Luftfeuchtedaten des Sensors. Zudem ist er Subscriber des Themas „time“. Dieses Thema wird verwendet, um den Zeitstempel der Daten zu synchronisieren. Es werden mehrere Clients dieser Art im Versuch verwendet.
In dem Sketch des Sensor-Clients müssen weitere Bibliotheken für den Temperatur- und Luftfeuchtesensor eingebunden werden. Zudem wurde der Sketch um einige *define*-Direktiven erweitert. Mit diesen werden grundlegende Einstellungen, wie der WLAN-Name und das WLAN-Passwort als auch Informationen wie die Server-IP des MQTT-Brokers oder die Sensornummer des Sensor-Clients, in dem Programm ersetzt. Weiterhin wurden Variablen deklariert, die zur Nutzung des Sensors, als auch zu weiteren Einstellungen im Programm nötig sind.
Um den Sketch für einen weiteren Client zu benutzten, muss nur die Direktive, mit der die Sensornummer festgelegt wird, geändert werden. Das heißt, dass nur eine Zahl geändert werden muss.
Im Vergleich zu dem Beispielsketch sind die *setup*- und *loop*-Funktion umfangreicher.
So wurde in der *setup*-Funktion vor dem Beginn der WiFi-Verbindung der WiFi-Modus des ESP auf „station“ gesetzt. Dies ist nötig, da sich mit einem Access Point verbunden werden soll. Zudem wird der DHT22-Sensor gestartet.
Nach dem Verbindungsversuch mit dem MQTT-Broker wird solange gewartet, bis sich der Client mit diesem verbunden hat. Darauffolgend wird das Thema „PC/time“ subscribt. An dieses Thema wird von dem Auslese-Client, welcher in Kapitel 3.2.2.3 thematisiert wurde, gepublisht, wenn die entsprechenden Daten von dem PC auf die serielle Schnittstelle geschrieben werden. Als serielle Schnittstelle wird dabei USB genutzt. Nach dem Subscriben publisht der Client an das Thema „*Sensornummer*/getTime“. Daraufhin wird so lange von dem Client gewartet, bis der Auslese-Client die Nachricht verarbeitet und an das Thema „PC/time“ gepublisht hat.
Die *loop*-Funktion unterscheidet sich vollkommen von der aus dem Beispielsketch.

Listing 3.3: Sensor-Client Datenauslese

```
109   float h = dht.readHumidity();
110   float t = dht.readTemperature();
111
112   while (isnan(h) || isnan(t)){
113     Serial.println("Failed to read from DHT sensor!");
114     delay(500);
115     h = dht.readHumidity();
116     t = dht.readTemperature();
117   }
118   float hic = dht.computeHeatIndex(t, h, false);
```

In Listing 3.3 ist der Code zum Auslesen der Sensordaten zu sehen.
So werden in Zeile 109 und 110 die Sensordaten ausgelesen und in zwei Variablen gespeichert. Darauf werden die Sensordaten in einer while-Schleife solange erneut ausgelesen, bis diese gültig sind. Dies kann zum Beispiel passieren, wenn der DHT nicht angeschlossen ist. Daraufhin kann aus den Variablen der Hitzeindex[4] errechnet werden. Dies ist in Zeile 118 zu sehen. Dabei wird neben dem Feuchtigkeits- und Temperaturwert *false* angegeben. Damit wird angegeben, dass die Temperatur nicht in Fahrenheit, sondern in Celsius ist.

Listing 3.4: Sensor-Client Datenstring

```
122 String data = "Humidity="  +(String) h
123   +"&Temperature_Cel="+(String) t
124   +"&HeatIndex_Cel="  +(String) hic
125   +"&Time="           +(String) (millis()-nulltime+timegone)
```

In dem Listing 3.4 ist zu sehen, wie die zu sendenden Daten zusammengesetzt werden. Dabei handelt es sich zum einen um die Temperatur und die Luftfeuchte mit dem berechneten Hitzeindex und zum anderen um einen Zeitstempel. Dieser wird aus der aktuellen Systemzeit, dem Nullzeitpunkt und der vergangenen Zeit seit Steuerungsstart errechnet.
Die Sensordaten werden darauf unter dem Thema „*Sensornummer*/sensordata“ gepublisht. Durch Zeitstempel und die *delay*-Funktion, mit der das Programm die angegebene Zeit in Millisekunden wartet, wird ermöglicht, dass jede Sekunde Daten gepublisht werden.
Neben diesen beiden Hauptfunktionen wurde die *myDataCb*-Funktion erweitert. Diese Funktion wird aufgerufen, sobald an ein Thema gepublisht wird, das der Client subscribt hat. In dieser Funktion wird bei jeder ankommenden Nachricht geprüft, ob die ankommende Nachricht zu dem Thema „PC/time“ gehört. Ist dies der Fall, so wird die Variable *nulltime*, die den Startpunkt der Zeitzählung markiert, auf die aktuelle Systemzeit und die Variable *timegone*, die die schon in der Steuerung vergangene Zeit widerspiegelt, auf die übertragenen Daten gesetzt. Mit diesen beiden Daten wird der Zeitstempel errechnet.

[4] Es handelt sich um eine Größe, die der Beschreibung der gefühlten Temperatur, auf Basis der gemessenen Lufttemperatur und der relativen Luftfeuchtigkeit, dient.

3.2.2.2. Heiz-Client

Der Heiz-Client wird genutzt um die Heizung, die durch einen Heizwiderstand realisiert wurde, anzusteuern. Da es nur einen Heizwiderstand gibt, gibt es nur einen Client dieser Art. Um die Heizung zu steuern, subscribt dieser Client das Thema „regler“ und wartet auf Daten. Wurden Daten zu diesem Thema gepublisht, wandelt er die Daten in einen Integer um und nutzt diesen Wert als Ausgabewert eines PWM-Signals, mit dem der Stromfluss des Heizwiderstandes gesteuert wird.
Um die genannten Funktionen zu gewährleisten, musste wieder der Beispielsketch verändert werden. So wurden auch hier wie bei dem Sensor-Client 3.2.2.1 *define*-Direktiven verwendet. Da bei diesem Client kein Sensor angeschlossen ist, sind diese Direktiven nicht vorhanden. Stattdessen wurde eine Direktive für den Pin eingeführt, an dem später das PWM-Signal ausgegeben wird.
Analog zu dem Sensor-Client muss in der *setup*-Funktion der WiFi-Modus des ESP auf „station“ gesetzt werden. Der Client subscribt das Thema „PC/regler“ an das der Auslese-Client publisht, wenn die entsprechenden Daten von dem PC auf die serielle Schnittstelle geschrieben werden.
Die *loop*-Funktion ist bei dem Client komplett leer, da er nur darauf warten muss, dass an das oben genannte Thema gepublisht wird. Deshalb wird auch bei diesem Client die *myDataCb*-Funktion verändert. Es wird wie bei dem Sensor-Client eine Abfrage nach dem Thema, zu dem die Daten gehören, eingeführt.

Listing 3.5: Heiz-Client

```
119     analogWrite(PWM_PIN, atoi(data.c_str()));
```

Wurden die Daten also zu dem Thema „PC/regler“ gepublisht, so wird auf den durch die *define*-Direktive angegebenen GPIO[5]-Pin ein PWM-Signal, mit der in der Nachricht angegebenen Einschaltdauer, gegeben. Die Daten müssen nicht geprüft werden, da nur Daten an dieses Thema gepublisht werden, die von dem PC kommen und dort korrekt gesendet werden. Dies ist in Listing 3.5 gezeigt. Für das Anlegen des PWM-Signals wird dabei die *analogWrite*-Funktion genutzt, während die übertragenen Daten zu einem Integer mit der *atoi*-Funktion umgewandelt werden.

3.2.2.3. Auslese-Client

Der Auslese-Client wird an dem PC angeschlossen, über den der Versuch gestartet wird. Dieser Client hat alle Themen subscribt und schreibt das Thema, als auch die zugehörigen Daten, auf die serielle Schnittstelle, sobald er etwas empfängt. Zudem liest er die serielle Schnittstelle aus und publisht die über die Schnittstelle übermittelten Daten zu dem ebenfalls übertragenen Thema.
Im Folgenden sind zwei Beispieldatensätze zu sehen. Sie haben den Aufbau, wie sie auch auf die serielle Schnittstelle geschrieben werden.

- `1/getTime,1`
- `2/sensordata,Humidity=45.50&Temperature_Cel=22.50&HeatIndex_Cel=22.00&Time=35598`

[5] GPIO – general purpose input/output. Es handelt sich um einen Kontaktstift, dem kein vordefinierter Zweck zugewiesen ist. Das Verhalten ist durch logische Programmierung frei bestimmbar.

In der ersten Zeile ist eine Anfrage nach der Zeit von dem Sensor 1 und in der zweiten Zeile gepublishte Sensordaten von dem Sensor 2 dargestellt. Dies ist an der beginnenden 2 zu erkennen.
Die Zeit, die von dem PC auf die serielle Schnittstelle geschrieben wird und von dem Client gepublisht wird, hat folgende Form:

- `PC/time,35798`

Es gibt nur einen Auslese-Client, da durch das Subscriben aller Themen alle Daten empfangen werden. Theoretisch ist es möglich, dass mehrere Clients dieser Art existieren. Dies wäre in der Praxis nutzbar, wenn die Daten auf verschiedenen PCs verarbeitet werden sollen.
Die oben genannten Funktionen führen in der Implementation zu einer Abweichung von dem Beispielsketch. Wie zuvor werden durch *define*-Direktiven Angaben zu WLAN, Sensornummer und Server gemacht. Weiterhin kommt eine Direktive hinzu, die beschreibt welche Themen später subscribt werden sollen. In diesem Fall ist es das Doppelkreuz „#“. Dabei handelt es sich in MQTT um ein Wildcard-Zeichen. Wird es subscribt, werden alle Nachrichten aller Themen an den Subscriber weitergeleitet.
In der *setup*-Funktion wird wieder der WiFi-Modus auf „station“ gesetzt und zum Schluss werden durch einen Aufruf der *subscribe*-Funktion mit dem Parameter „TOPIC“, der durch die *define*-Direktive mit einem „#“ ersetzt wird, alle Themen subscribt.
Die *loop*-Funktion weicht ebenfalls wieder von dem Beispiel ab. In dieser wird, solange etwas auf der seriellen Schnittstelle liegt, von dieser eingelesen. Daraufhin wird geprüft, ob die gelesenen Daten nicht nur fehlerhafte Werte enthalten. Dazu wird geprüft, ob die Länge des aus den eingelesenen *chars* bestehenden *strings* größer 0 ist. Ist dies der Fall, werden die Daten von dem Thema nach einem Trennzeichen – hier ist es ein Komma – getrennt und daraufhin werden die Daten unter dem entsprechenden Thema gepublisht.
Die Übertragung der empfangenen Daten erfolgt durch ein Schreiben auf die serielle Schnittstelle. Dazu muss nichts in dem Beispielsketch verändert werden, da in diesem schon die Ausgabe über die serielle Schnittstelle erfolgt.

3.2.3. Konfiguration der Arduino IDE

Die Installation der IDE[6] wird im Anhang A.1 erklärt.
Zuerst muss unter *Datei→Voreinstellungen→Zusätzliche Boardverwalter-URLs* der Link „http://arduino.esp8266.com/stable/package_esp8266com_index.json“ hinzugefügt werden. Um das ESP8266 Modul mit der Arduino IDE benutzen zu können, muss der ESP im Bibliotheken Manager der Arduino IDE unter *Werkzeuge→Board:…→Boardverwalter…→Filter:* „ESP8266“ ausgewählt werden. Von dort aus können die Ergebnisse bis zu dem Eintrag „esp8266 by ESP8266 Community“ durchsucht werden. Dies muss installiert werden. Darauf kann das Adafruit Feather Huzzah ESP8266 unter *Werkzeuge→Board:…* ausgewählt werden. Zusätzlich müssen noch ein Port und die Upload-Geschwindigkeit (in Baud) gesetzt und ein Board ausgewählt werden. Der Port muss im Betriebssystem nachgesehen werden und als Upload-Geschwindigkeit wird 115200 ausgewählt. Unter *Werkzeuge→Erase Flash:…* wird „Sketch + WiFi Settings“ gewählt. Zudem sollte für die uMQTTBroker-Bibliothek unter *Werkzeuge→lwIP*[7] *Variant* die lwIP-Version „1.4 High Bandwidth“ gewählt werden.

[6] Integrierte Entwicklungsumgebung, von englisch „integrated development environment“
[7] lwIP ist die Abkürzung für „lightweight IP“. Es ist ein weit verbreiteter Open-Source-TCP/IP-Stack für eingebettete Systeme.

Zudem müssen noch weitere Bibliotheken zur Nutzung der Sensoren installiert werden. Dazu zählen:

- DHT Sensor Library
- ESP8266Wifi
- uMQTTBroker

Die ersten zwei Bibliotheken können in der Arduino IDE im Blibliothek-Manager unter *Sketch→Include Library→Library Manager* gesucht, ausgewählt und dann installiert werden. Die dritte Bibliothek wird aus dem GitHub Repository [12] heruntergeladen und in den Standardordner „*Benutzer*\Documents\Arduino\libraries“ entpackt. Nach einem Neustart der IDE kann auf diese Bibliothek zugegriffen werden.

4. Kapazität des Sensornetzes

In diesem Kapitel wird die Kapazität des Sensornetzes getestet. Das bedeutet, dass getestet wird, wieviele Publisher maximal in einem Netz möglich sind. Es wird sozusagen der Flaschenhals[1] im System gesucht.
Da in der Dokumentation der uMQTTBroker-Bibliothek gesagt wird, dass bis zu acht Clients getestet wurden, wurde versucht dies zuerst zu validieren.

4.1. Validierung der Anzahl möglichen Clients

Da die lwIP-Version 2.0 dazu führt, dass bei mehr als fünf Verbindungen das Socket blockiert, rät „martin-ger“, der die uMQTTBroker-Bibliothek geschrieben hat, dazu, die lwIP Version „1.4 High Bandwidth“ zu verwenden. Diese Einstellung wird auch bei dem Aufspielen auf die Mikrocontroller gewählt.
Bei der Validierung wurden zuerst einfache Sketche geschrieben, die nur Daten mit einer eigenen ID jede Sekunde publishen. Dabei wurden acht Boards vorbereitet, wobei die Boards nummeriert wurden. Die ID, die neben den Daten gepublisht wird, ist die Boardnummer. Es wurden nach und nach die Boards eingeschaltet, nachdem bei der Überwachung der Daten bei dem Broker gesehen wurde, dass Daten ankommen.
Dies wurde fünf Mal durchgeführt. Bei allen fünf Versuchen konnten sich nicht mehr als fünf Clients mit dem Broker verbinden. Der sechste Client konnte sich zwar jedes mal mit dem WLAN verbinden, aber es war nicht möglich, dass er sich mit dem Broker verbindet.
Aus diesem Grund wurden verschiedene Ansätze entworfen, dieses Problem zu beheben, die im Folgenden beschrieben werden.

4.1.1. Reduzierung der zu übertragenden Daten

Als erster Ansatz wurde die Anzahl der zu übertragenden Daten reduziert. Dazu wurde der Sketch so umgeschrieben, dass nur alle 5 Sekunden Daten gepublisht werden.
Dies führte jedoch zu keiner Verbesserung. Der sechste Client kann sich immer noch nicht mit dem Broker verbinden.

4.1.2. Löschen von Verbindungen im TCP *time-wait*-Zustand

Als zweiter Ansatz wurde das Löschen von Verbindungen im TCP *time-wait*-Zustand ausprobiert. Der *time-wait* Zustand hilft TCP, zwei aufeinanderfolgende Verbindungen nicht zu verwechseln, wenn die erste bereits geschlossen ist, aber immer noch doppelte Pakete im Internet verloren gehen, die später während der zweiten ankommen.
Nach der Dokumentation des ESP8266 Arduino Kerns[18] ist die Anzahl der Verbindungen

[1] Auch bekannt als Bottleneck. Die Ressource im System, welche den niedrigsten Durchsatz in dem gesamten System hat und damit den Durchfluss begrenzt. Dadurch stellt sie eine eine Kapazitätsgrenze für das Gesamtsystem dar.

in diesem Zustand auf fünf begrenzt und sie werden bei dem Überschreiten entfernt. Daneben gibt es noch einen manuellen Workaround, bei dem durch den Aufruf einer Funktion die Verbindungen bereinigt werden.
Dieser Workaround wurde auf dem Broker installiert, hat jedoch keine Veränderung bewirkt.

4.1.3. Änderung der lwIP-Version

Als dritter Ansatz wurde die lwIP-Version verändert. Zur Auswahl stehen neben der von „martin-ger“ empfohlenen Version „1.4 High Bandwidth“ die Versionen:

- v2 Lower Memory
- v2 Higher Bandwith
- v1.4 Compile from Source

Die Version „v1.4 Compile from Source“ führt zu einem Fehler bei dem Kompilieren, der nicht behoben werden konnte. Diese Version bringt also keine Vorteile.
Die beiden „v2“-Versionen kompilieren. Mit beiden verbindet sich das sechste Board zuerst mit dem WLAN und danach auch scheinbar mit dem Broker. Der Mikrocontroller publisht Daten, von denen jedoch keine einzige bei dem Broker ankommt. Es ist also auch hier nicht möglich, einen weiteren Client zu nutzen.

4.1.4. Änderung der Boardversion

Als vierter Ansatz wurde versucht, die Boardversion zu verändern. Dazu muss in der IDE unter *Werkzeuge→Board→Boardverwalter…* im Suchfeld nach „esp8266“ gesucht werden. Wird der Eintrag angeklickt, kann unten links die Version ausgewählt werden. Bisher war die neuste Version 2.4.2 eingestellt.
Als erstes wurde die Boardversion 2.4.1 ausgewählt. Darauf wurden erneut alle lwIP-Versionen getestet, es kam dabei zu den gleichen Ergebnissen wie zuvor.
Anschließend wurde die Boardversion 2.4.0 gewählt. In dieser Boardversion gibt es eine andere Auswahl an lwIP-Versionen diese sind im Folgenden aufgelistet:

- v2 Prebuilt (MSS=536)
- v2 Prebuilt (MSS=1460, unstable)
- v1.4 Prebuilt
- v1.4 Open Source

Wie auch zuvor kompiliert die letzte Version „v1.4 Open Source“ als einzige nicht. Version „v1.4 Prebuilt“ verhält sich wie Version „1.4 High Bandwidth“ und die beiden „v2“-Versionen wie auch zuvor die „v2“-Versionen. Diese Boardversion hat also auch keine Besserung gebracht.
Als letzte Boardversion wurde die Version 2.3.0 getestet. Bei dieser ist keine Auswahl der lwIP-Version möglich. Der sechste Client verhält sich jedoch auch hier wie bei der lwIP-Version „1.4 High Bandwidth“ bei der Boardversion 2.4.2. Es wurde also keine Verbesserung erzielt.

4.2. Minimale Übertragungsperiodendauer

Da nicht mehr als fünf Verbindungen mit dem Broker möglich sind, wurde anschließend getestet, was die minimale Übertragungsperiodendauer ist. Da von den fünf möglichen Verbindungen eine die des Auslese-Clients ist, sind nur noch vier Verbindungen frei. Würde der Heiz-Client um einen Sensor erweitert, wäre es möglich, die Daten von vier Sensoren zu publishen. Ein solcher maximaler Aufbau ist auch in Abbildung 2.10 zu sehen.
Für den Test liest der Auslese-Client alle gepublishten Daten aus und schreibt sie auf die serielle Schnittstelle, damit diese von dem PC ausgelesen werden können. Die anderen vier Clients publishen in einem sich verändernden Takt (50-1000ms) statische Daten.
Dieser Datensatz hat folgende Form:

Sensornummer`/sensordata,`↩
`Humidity=45.50&Temperature_Cel=22.50&HeatIndex_Cel=22.00&Time=42023`

Dabei verändert sich nur die Sensornummer, die in dem Thema enthalten ist, zu dem die Daten gepublisht werden.
Die gepublishten Daten werden bei allen Clients über den seriellen Monitor ausgelesen. Aus den von den vier Clients gepublishten Daten kann bestimmt werden, wieviele Daten effektiv gesendet worden sind. Damit kann der prozentuale Anteil errechnet werden, der den Broker, beziehungsweise den Auslese-Client, erreicht.

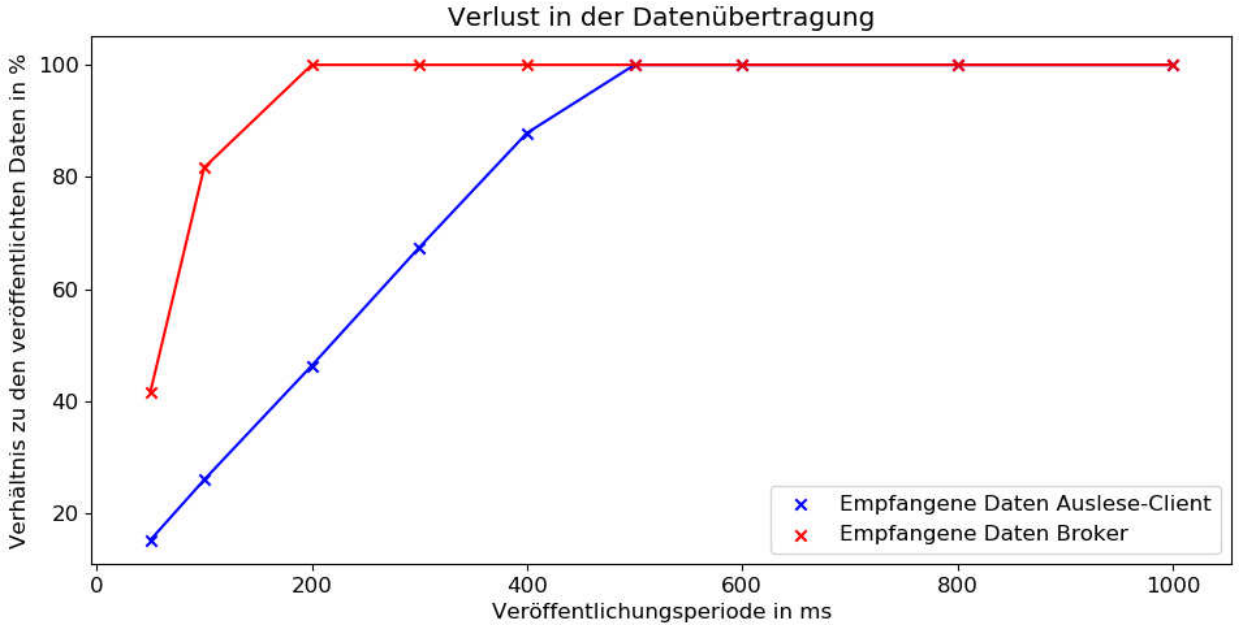


Abbildung 4.1.: Der Datenverlust in Abhängigkeit der Übertragungsperiodendauer.

In der Abbildung 4.1 ist zu sehen, wie sich der Datenverlust bei einer sinkenden Periodendauer der publish-Operationen verhält. In der Abbildung ist zum einen angegeben, wieviel Prozent der gepublishten Daten bei dem Broker (rot) und dem Auslese-Client (blau) ankommen. Ab einem Ausführen der publish-Operation alle 500 ms, kommen alle Daten an. Unter 400 ms kommen nicht mehr alle Daten bei dem Auslese-Client an, während erst unter 100 ms Datenverluste bei dem Broker zu sehen sind.
Dies bedeutet, dass Temperaturdaten von vier Sensoren und ein Zeitstempel alle 500 ms

sicher gepublisht werden können. Da es für Temperaturdaten eher ungewöhnlich ist, dass sie sich so schnell verändern, bedeutet dies, dass der Zeitabstand der publish-Operationen, der eine Sekunde beträgt, für die Temperaturwerte beibehalten werden kann. Zudem ist es möglich, Daten weiterer Sensoren zu publishen, da das Sensornetz noch nicht ausgelastet ist.

5. Temperaturregler

In diesem Kapitel wird zuerst beschrieben, welche verschiedenen Reglerarten es gibt. Daraufhin wird genauer auf den PI-Regler eingegangen. Schließlich wird sowohl auf die Realisierung des Reglers, sowie dessen Verifikation, eingegangen.

5.1. Regler

Ein Regler verfolgt im allgemeinen den Ansatz eines geschlossenen Wirkungskreislaufs. Er wird auch als Regelkreis bezeichnet. Bei einem Regelkreis gibt es mehrere Größen, die wichtig sind. Im Folgenden werden diese kurz beschrieben.

- Bei der *Regelgröße* handelt es sich um die Größe, die gemessen wird.
- Der *Sollwert* ist der gewünschte Wert, den die Regelgröße haben soll.
- Die *Stellgröße* wird von dem Regler erzeugt und in einem Prozess umgewandelt, um die Regelgröße zu beeinflussen.
- Die *Störgröße* ist eine Größe die sich auf den Regelkreis auswirkt, aber oft unerwünscht ist.

In der Regelungstechnik werden Geräte und Programme als Regler genutzt, um einen vorhandenen Zustand eines Prozesses in einen festgelegten gewünschten Zustand zu bringen. Dazu wird aus der Regeldifferenz eine Stellgröße berechnet, mit der die Regeldifferenz minimiert werden soll.

$$\text{Regeldifferenz} = \text{Sollwert} - \text{Regelgröße (Istwert)} \tag{5.1}$$

Die Berechnung der Regeldifferenz wurde in Gleichung (5.1) gezeigt. Die Regeldifferenz wird auch als Fehler, oder im englischen *Error*, bezeichnet.
Bei dem Regeln wird auf die Sensorik und Aktorik zurückgegriffen. Die Aktorik, auch Informationseinwirkung genannt, führt die Stellgröße in eine weitere Größe über, die eine Einwirkung auf den Prozess hat. Um diese Änderung messen zu können, wird die Sensorik benötigt. Mit der Sensorik, die auch als Informationsgewinnung bezeichnet wird, kann eine sogenannte Rückmeldung über die Wirkung der Aktorik gewonnen werden.

5.1.1. Proportional-Regler

Der Proportional-Regler wird auch als P-Regler bezeichnet. Er erhält seinen Namen durch die Proportionalität zwischen der Regeldifferenz und der Stellgröße. Die Regeldifferenz wird in diesem Zusammenhang als $e(t)$ bezeichnet, wobei das e für den „Error“ steht, der von der Zeit abhängig ist. Daneben gibt es noch den Faktor K_P.

Zur Regelung wird die Regeldifferenz $e(t)$ mit einem Faktor K_P multipliziert. Das Ergebnis ist die Stellgröße, die als $U_M(t)$ bezeichnet wird. Dies wird durch Gleichung (5.2) verdeutlicht.

$$U_M(t) = K_P \cdot e(t) \tag{5.2}$$

Ein Problem bei diesen Reglern ist, dass der Sollwert nie erreicht werden kann, da bei $e(t) = 0$ auch $U_M = 0$ ist. Es entsteht also eine bleibende Regeldifferenz. Dies liegt daran, dass die Stellgröße einen Wert annimmt, an dem keine Änderung mehr geschieht.

5.1.2. Integral-Regler

Bei einem Integral-Regler, auch als I-Regler bezeichnet, sind die Änderungen der Stellgröße proportional zu dem Integral der Regeldifferenz über der Zeit.[19]
Zur Regelung wird das Integral der Regeldifferenz $e(t)$ mit einem Faktor K_I multipliziert. Das Ergebnis ist die Stellgröße $U_M(t)$. Die Gleichung (5.3) beschreibt den I-Regler.

$$U_M(t) = K_I \cdot \int e(t)dt \tag{5.3}$$

Das Integral in der Regeldifferenz führt dazu, dass die Stellgröße, sobald der Sollwert erreicht ist, nicht mehr weiter ansteigt. Steigt die Regelgröße über den Sollwert wird durch das Integral die negative Regeldifferenz mit eingerechnet. Dadurch wird das Integral wieder kleiner, wodurch auch die Stellgröße kleiner wird.

5.1.3. Vergleich der P-und I-Regler

Der P-Regler hat den Vorteil, dass er schnell und robust regelt. Der Nachteil ist, dass er den Sollwert nicht erreichen kann. Im Vergleich dazu kann der I-Regler den Sollwert erreichen, wobei er jedoch sehr langsam regelt.
Am besten wäre also eine Kombination der Vorteile der beiden Regler.

5.1.4. Proportional-Integral-Regler

Zu dem proportionalen Anteil des P-Reglers wird der integrierender Anteil des I-Reglers hinzugenommen. Mit dieser Kombination der beiden Regler wird versucht die positiven Eigenschaften der beiden Regler zu übernehmen. Daher erhält der Regler den Namen Proportional-Integral-Regler, der auch als PI-Regler bezeichnet wird. Bei dem Regler wird also auf die von dem P-Regler errechnete Stellgröße das Integral der Regeldifferenz $\int e(t)dt$ multipliziert mit einem Faktor K_I addiert. Die genaue Berechnung der Stellgröße bei einem PI-Regler wird in Gleichung (5.4) beschrieben.

$$U_M(t) = K_P \cdot e(t) + K_I \cdot \int e(t)dt \tag{5.4}$$

Die Berechnung der Stellgröße ist in Abbildung 5.1, die das Blockschaltbild eines PI-Reglers zeigt, ebenfalls zu erkennen.

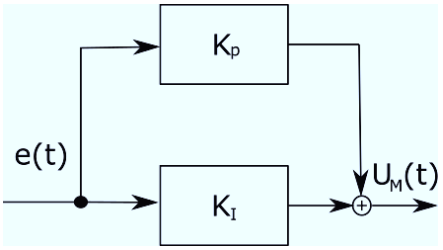


Abbildung 5.1.: Blockschaltbild eines PI-Reglers.

5.2. Systemanalyse

In diesem Kapitel wird das bestehende System analysiert.
Bei dem System besteht der Regelkreis aus den beiden Styroporboxen, die durch die Schläuche miteinander verbunden sind, als auch aus dem Gebläse, dem Heizwiderstand und den Sensoren. In diesem System wird durch das Gebläse eine Luftzirkulation ermöglicht.
Die Temperatur, die von dem Temperatursensor an der Styroporboxdecke der Experiment-Seite, welche in Kapitel 2.3.1.2 thematisiert wurde, gemessen wird, ist die Regelgröße.
Die Temperatur in dieser Box soll eine Temperatur, die zu Beginn einer Messung angegeben wurde, den Sollwert, erreichen und konstant halten.
Bei der Stellgröße handelt es sich um einen Wert, der in einem Umwandlungsprozess auf einen Wert zwischen 0 und 1023 gesetzt wird. Der Wert dient als Grundlage für ein PWM-Signal, das über einen GPIO ausgegeben wird. Der Wert 1023 bedeutet dabei ein Tastgrad von 0%, während ein Wert von 0 für einen Tastgrad von 100% steht.
Bei dem Versuchsaufbau gibt es mehrere Störgrößen. Diese sind im folgenden aufgelistet.

- Wärmeverlust (Undichtigkeit der Boxen/Schläuche/Verbindungen, Wind, Luftzirkulation)
- Wärmzufuhr (Sonne, erhöhte Außentemperatur)

Das PWM-Signal ist ein Stellsignal und führt mit dem Heizwiderstand zu einer aktiven Veränderung der Regelgröße. Bei dem Heizwiderstand handelt sich also um einen Aktor, wohingegen die Temperatur an dem Deckel der Box mit einem Sensor gemessen wird.
Das gesamte System besitzt zudem eine Totzeit. Der Hauptteil dieser entsteht dadurch, dass der Heizwiderstand nicht schlagartig heizen kann. Ebenfalls besitzt dieser eine Restwärme, selbst wenn die Stromzufuhr auf null geregelt wird. Der Heizwiderstand benötigt Zeit, um eine bestimmte Temperatur zu erreichen, und braucht nach dem Stopp des Stromflusses auch noch einige Zeit, bis er wieder abgekühlt ist. Deshalb ist das System ein nichtlineares System. Ein weiterer Totzeitanteil entsteht durch die Bewegung der Luft von einer Styroporbox in die andere. Diese Zeit ist durch das starke Gebläse jedoch verschwindend gering im Vergleich zu der Aufheizzeit des Heizwiderstandes. Zudem besitzt die Styroporbox eine Totzeit, da sie die Temperatur, die ankommt, nicht direkt annimmt.
In Abbildung 5.2 ist das Signalflussbild des Regelkreises abgebildet. Dabei steht $w(t)$ für den Sollwert. Die Regeldifferenz ist durch $e(t)$ bezeichnet, die der Eingangswert für den Regler ist. Die Ausgabe des Reglers ist die Stellgröße $u(t)$, die zu einer Veränderung des Stromdurchflusses im Heizwiderstand führt. Dies hat eine Wirkung auf die Regelstrecke. Zudem haben die Störgrößen $z(t)$ eine Auswirkung auf den Regelkreis. Darauf kann der Messwert $y(t)$ wieder zur Berechnung für die Regeldifferenz eingehen.

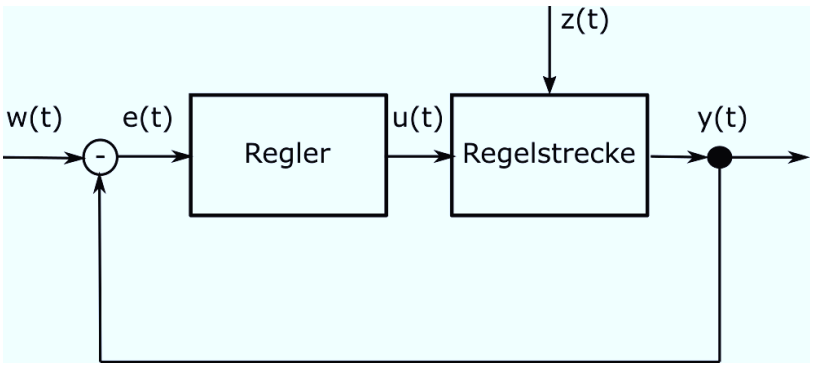


Abbildung 5.2.: Signalfluss im Regelkreis.

5.3. Realisierung

In diesem Kapitel wird auf die Realisierung des Reglers eingegangen. Dabei wird zuerst beschrieben, wie der Regler implementiert und daraufhin das Steuerungsskript für den Versuch geschrieben wurde.

5.3.1. PI-Regler

Der PI-Regler wurde in Python realisiert. Dazu wurde die Klasse *PI* erstellt.
Die Klasse *PI* besitzt einen Konstruktor, der einen P- und einen I-Wert übergeben bekommt. Sind keine Werte angegeben werden die Standardwerte $P = 512$ und $I = 0,3$ gewählt. Diese Werte wurden aus den Messreihen in Kapitel 5.4 gewonnen. Daraufhin werden die Klassenvariablen *Kp* und *Ki*, bei denen es sich um die schon in Kapitel 5.1.4 beschriebenen Konstanten handelt, auf die Werte von P und I gesetzt. Zudem wird die Klassenvariable *current_time* auf die Zeit des Aufrufes gesetzt und *last_time* auf die gerade neu gesetzte Variable *current_time*. Dabei wird die *last_time* genutzt, um den letzten Regelzeitpunkt zu speichern und *current_time* ist der Zeitpunkt, an dem neu geregelt wird. Daraufhin wird die Funktion *clear* aufgerufen.
In der Funktion *clear* wird die Klassenvariable *SetPoint*, bei der es sich um den Sollwert handelt, als auch die Fehlerterme für P und I, die zuletzt gemessene Regeldifferenz und der Stellwert auf 0 gesetzt.
Die wohl wichtigste Funktion der Klasse ist die *update*-Funktion. Sie erhält die Regelgröße als Übergabeparameter. Daraufhin kann aus diesem und dem Sollwert die Regeldifferenz errechnet werden. Nach dem Berechnen der Zeit, die seit dem letzten Aufruf der Funktion vergangen ist, kann aus diesen die Zeitdifferenz der Messungen errechnet werden. Mit den damit errechneten Werten können die P- und I-Terme errechnet werden. Dazu wird für den P-Term die Regeldifferenz mit der Konstanten *Kp* multipliziert und für den I-Term wird auf den schon bestehenden I-Term das Ergebnis aus der Multiplikation von Zeitdifferenz und Regeldifferenz aufaddiert. Daraufhin wird die Klassenvariable *last_time* auf die Zeit des Aufrufes gesetzt. Danach kann die Klassenvariable *output*, bei der es sich um die Stellgröße handelt, berechnet werden. Dazu werden der P-Term und das Ergebnis der Multiplikation aus der Konstanten *Ki* und des I-Terms, dem Fehlerintegral, addiert. Anschließend wird von 1024 dieses Ergebnis abgezogen und wenn dann die Variable *output* größer als 1023 ist, wird sie auf 1023 gesetzt und ist sie kleiner als 0 wird sie auf 0 gesetzt. Diese Berechnung ist in Listing 5.1 zu sehen.

Listing 5.1: PI-Regler

```
26 def update(self, feedback_value):
27     error = self.SetPoint - feedback_value
28
29     self.current_time = time.time()
30     delta_time = self.current_time - self.last_time
31
32     self.PTerm = self.Kp * error
33     self.ITerm += error * delta_time
34
35     self.last_time = self.current_time
36     self.output = 1024 - (self.PTerm + (self.Ki * self.ITerm))
37     if self.output>1023:
38         self.output = 1023
39     if self.output<0:
40         self.output = 0
```

5.3.2. Steuerung

Auch die Steuerung des Reglers wurde in Python implementiert.
In dem Programm wird zuerst die Nullzeit gesetzt. Dabei handelt es sich um die Zeit, zu der relativ alle Daten gespeichert werden. Daraufhin wird geprüft, ob überhaupt serielle Schnittstellen vorhanden sind. Ist dies der Fall, wird die erste serielle Schnittstelle gewählt. Da keine anderen Clients oder Geräte, die einen seriellen Port öffnen, an dem PC angeschlossen sind, handelt es sich hierbei um den Auslese-Client.
Bei dem Öffnen des seriellen Ports muss darauf geachtet werden, dass dabei der Zusatz *rtscts=true* gewählt wird. Dadurch wird die RTS/CTS-Flusskontrolle eingeschaltet; *RTS* steht für „Ready To Send“ und *CTS* für „Clear To Send“. Die Bezeichnung *RTS* ist in den letzten Jahren gegen *RFR* ausgetauscht worden, da es sich eigentlich um „Ready For Receiving“ handelt. Das Öffnen des Ports mit der Einstellung *rtscts=true* ist unabdingbar, da ansonsten der ESP durch das Öffnen resettet wird. Nach dem Öffnen des Ports mit dieser Einstellung, wird er wieder geschlossen, da das Setzen der Leitungen zum Senden und Empfangen einen großen Aufwand darstellt. Danach kann der serielle Port wieder mit der Standardeinstellung, bei der *rtscts=off* ist, geöffnet werden.
Daraufhin wird „PC/time,*Systemzeit-Nullzeit*“ auf die serielle Schnittstelle mittels der Funktion *sendTime* geschrieben. Dabei handelt es sich um die schon vergangene Zeit seit Start der Steuerung. Daraufhin werden zwei Threads gestartet. Der erste Thread wird mit der Funktion *readPort* aufgerufen und dient dem Abspeichern der Daten und der Regelung und der zweite Thread wird mit der Funktion *displayData* aufgerufen und dient zum Plotten der Daten.
In der Funktion *readPort* wird solange in einer while-Schleife von dem Port gelesen, wie die Variable *readThePort* auf *true* steht. Nach dem Lesen wird versucht, das gelesene Zeichen zu decodieren. Ist dies erfolgreich, wird geprüft, ob es nicht leer ist. Ist dies der Fall wird geprüft, ob es sich um das Zeilenendezeichen „\r“ handelt. Trifft dies nicht zu, so wird das Zeichen der Variablen *line*, die aus einer Kette von Zeichen besteht, angefügt. Ist das Zeichen jedoch ein „\r“, so kann die Variable *line* verarbeitet werden. Es wird geprüft, ob ein Komma in der Zeile vorhanden ist, da dies das Trennzeichen von Thema und Daten ist.

Ist dies der Fall, so wird zuerst das Thema durch split-Operationen in der Variable *topic* gespeichert. Dabei wird nur die zweite Themenebene gewählt. Daraufhin wird geprüft, ob es sich um das Thema „getTime“ handelt. Sollte dies der Fall sein, wird die oben schon erwähnte Funktion *sendTime* aufgerufen. Handelt es sich jedoch um das Thema „sensordata“, so werden durch split-Operationen die Daten und die Sensornummer bestimmt. Daraufhin werden die Daten entsprechend der Sensornummer in eine Datei geschrieben. Zudem werden die Daten in Arrays geschrieben, wobei die Auswahl des Arrays durch if-Abfragen auf die Sensornummer geschieht. Diese Arrays werden zudem von dem zweiten Thread für das Plotten der Daten benötigt. Weiterhin wird bei Daten, die zu der Sensornummer *1* gehören, die Funktion *tempRegler* mit den Temperaturdaten als Übergabeparameter aufgerufen. In der Funktion wird dann die *update*-Funktion des Reglers mit diesem Temperaturwert aufgerufen. Danach wird „PC/regler,“ und der Wert der Klassenvariable *output* auf die serielle Schnittstelle geschrieben. Nach diesem Bearbeiten der Daten, wird die Variable *line* wieder auf einen leeren String gesetzt und das Folgezeichen des „\r“, das „\n“, wird von der seriellen Schnittstelle gelesen. Darauf beginnt die while-Schleife von vorne.
In dem zweiten Thread werden die in Arrays abgespeicherten Daten mit der *scatter*-Funktion der Bibliothek *matplotlib* geplottet.

5.4. Verifikation und Parameterbestimmung

Es wurde davon abgesehen, das System zu simulieren, da es zu viele Eigenschaften des Systems gibt, die nicht genau bekannt sind. Aus diesem Grund wurden verschiedene Tests durchgeführt, um die für das Experiment besten P- und I-Werte für den PI-Regler zu finden. Diese werden im folgenden erläutert.

5.4.1. I-Variation

Bei diesem Test wurden die I-Werte des Reglers bei einem festen P-Wert verändert.
In Abbildung 5.3 sind die Temperaturkurven bei verschiedenen PI-Werten dargestellt. Dabei wurde der P-Wert gleich belassen und der I-Wert verändert. Die Wahl des P-Wertes fiel auf 512, da es die Hälfte der Anzahl verschiedener PWM-Signale (1024) ist. Dies bedeutet, dass ein reiner P-Regler bis zu einer Regeldifferenz von 1024/512 =2 °C mit voller Leistung heizen würde und danach langsam weniger heizt. Da der I-Anteil hinzu kommt, ist es möglich einen exakten Sollwert von 28 °C Grad zu erreichen. Zudem wird durch das Heizen bis zu der 2 °C-Grenze sichergestellt, dass der Regler zügig die 2 °C-Grenze zu dem Sollwert erreicht. Es wurde hier die Grenze von 22 °C gewählt, da der Heizwiderstand immer eine Restwärme besitzt. Im Vergleich zu dem P Wert erscheint der I-Wert eher klein gewählt, aber da Heizzeiten sehr lang sind, kann das Fehlerintegral sehr groß werden.
In Abbildung 5.3 ist das Verhalten eines einfachen P-Reglers (blaue Kurve) zu sehen. Um einen solchen handelt es sich, wenn bei einem PI-Regler der I-Wert 0 gewählt wird. In der Abbildung wird dies durch die blaue Kurve dargestellt.
Der Wert des P-Reglers steigt zügig an und erreicht nach einem Überschwung, auf den ein kleiner Unterschwung folgt, einen relativ konstanten Wert. Bei diesem handelt es sich nicht um den Sollwert, da ein einfacher P-Regler diesen nicht erreichen kann. Die Schwankungen bei der Temperatur ergeben sich dadurch, dass der Temperatursensor nur eine Auflösung von 0.1 °C besitzt und sich der Temperaturwert genau zwischen der Grenze zweier Werte befindet und eine kleine Änderung zu dem Wechsel zwischen den Werten führt. Ein weiterer

Punkt dabei ist die Trägheit des Heizkörpers.
Mit der Erhöhung des I-Wertes zeigt sich, dass sich die Schwingfrequenz verkleinert. Dies ist durch die Verdopplung des I-Wertes besonders gut bei $I = 0.3$, $I = 0.6$ und $I = 1.2$ erkennbar. Zudem erhöht sich das Überschwingen mit steigendem I-Wert.
Zu der Ermittlung des besten Wertepaares, wurde zuerst verglichen, mit welchen Werten sich der Regler in unter 2000 s in einem Intervall von $\pm 0.3\,°C$ befindet. Dies ist der Fall bei $I = 0.15$ und $I = 0.3$ nach circa 700 s und bei $I = 0.6$ nach circa 1900 s. Von diesen dreien sind die Messwerte von $I = 0.3$ am besten, da dieser sogar nach circa 2000 s ziemlich konstant den Sollwert von 28 °C hält, wohingegen mit $I = 0.15$ und $I = 0.6$ der Sollwert sogar innerhalb von insgesamt 5000 s nicht konstant gehalten wird. Dies liegt daran, dass in dem nichtlinearen System zwar das Heizen linear steuerbar ist, das Kühlen jedoch nicht.

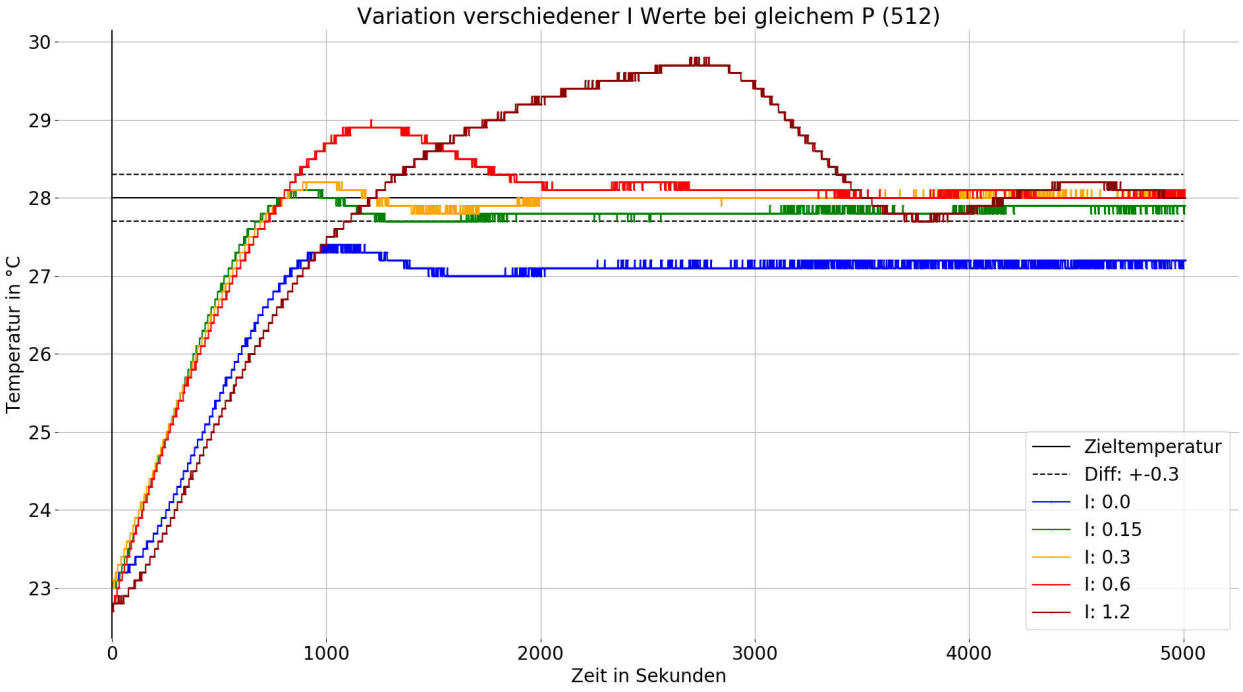


Abbildung 5.3.: Verlauf der Temperatur bei verschiedenen PI-Werten.

5.4.2. P-Variation

Da bei der Variation der I-Werte festgestellt wurde, dass bei einem P-Wert von 512 und einem I-Wert von 0.3 die besten Messergebnisse erzielt werden konnten, wurde ein I-Wert von 0.3 festgelegt. Mit diesem wurden Messungen mit verschiedenen P-Werten wiederholt. In Abbildung 5.4 sind die Messergebnisse zu sehen, bei denen der I-Wert festgelegt und der P-Wert geändert wurde.
Die Messdaten eines reinen I-Reglers werden in der blau gefärbten Kurve dargestellt. Dabei zeigt die Kurve zunächst einen geringen Anstieg. Ihr Anstieg beginnt jedoch bis zu einem Wendepunkt bei circa 1100 s zu wachsen, fällt dann aber wieder. Nachdem ein Maximum bei circa 2500 s erreicht wurde, fällt die Kurve wieder ab. Der I-Regler zeigt damit das erwartete Verhalten bei einem System mit Totzeit. Zudem besitzt diese Kurve eine deutlich kleinere Schwingfrequenz als die PI-Regler.

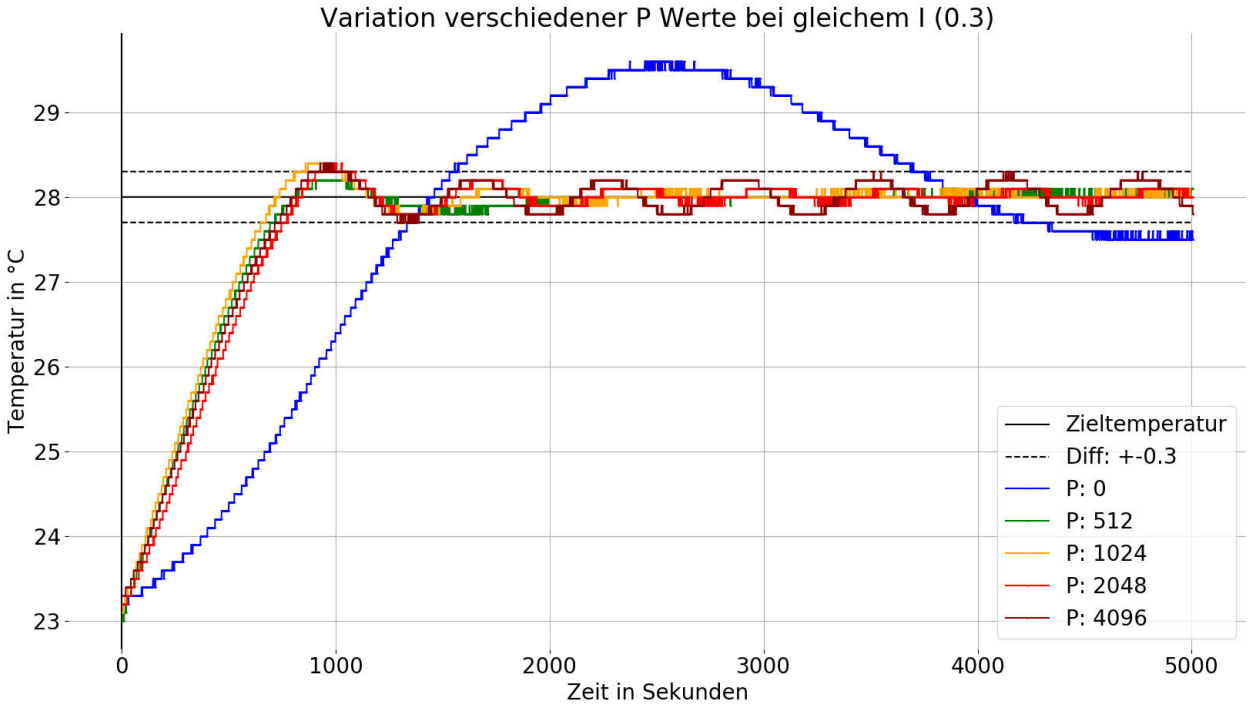


Abbildung 5.4.: Verlauf der Temperatur bei verschiedenen PI-Werten.

Die PI-Regler mit $P = 1024$, $P = 2048$ und $P = 4096$ befinden sich nach circa 1000 s in dem Intervall von $\pm 0.3\,°C$. Besser ist jedoch der PI-Regler mit $P = 512$, der sogar schon nach circa 700 s dieses Intervall erreicht hat.
Die Schwingfrequenz wird mit steigenden P-Werten größer. Dies ist besonders gut bei den Messdaten mit $P = 1024$, $P = 2048$ und $P = 4096$ zu erkennen. Daneben ist ersichtlich, dass sich bei den Messdaten mit $P = 4096$ eine anhaltende Schwingung ausgebildet hat. Anhaltende Schwingungen sind unerwünscht, da durch diese auch anhaltende Temperaturdrifts entstehen.
Im abschließenden Vergleich zu den in Abbildung 5.4 gezeigten Messergebnissen liefert der PI-Regler mit $P = 512$ und $I = 0.3$ erneut die besten Werte. Das Zielintervall, als auch der Sollwert, wird schnell erreicht. Zudem ist kein Schwingen zu sehen.

5.4.3. Störung des Regelsystems

Im folgenden wurde ein PI-Regler mit $P = 512$ und $I = 0.3$ verwendet, da er bei den durchgeführten Tests am besten abgeschlossen hat. Es wurde nicht versucht noch bessere Ergebnisse zu erreichen, da die benötigte Genauigkeit erreicht wurde. Stattdessen wurde getestet, wie der Regler auf größere Störungen reagiert.
Eine Grobübersicht auf die Reaktion auf drei Störimpulse ist in Abbildung 5.5 dargestellt. Als Kurven sind die Außentemperatur und die Temperatur am Deckel der Styroporbox zu sehen. Die gesamte Abbildung ist in zwei Hälften unterteilbar. Dabei ist in der ersten Hälfte die normale Regelung in einer temperaturstabilen Umgebung zu sehen. Sie reicht bis circa 11 800 s, was umgerechnet circa 3 h 17 min entspricht. Darauf folgt dann die zweite Hälfte in der die drei Störungen zu sehen sind.

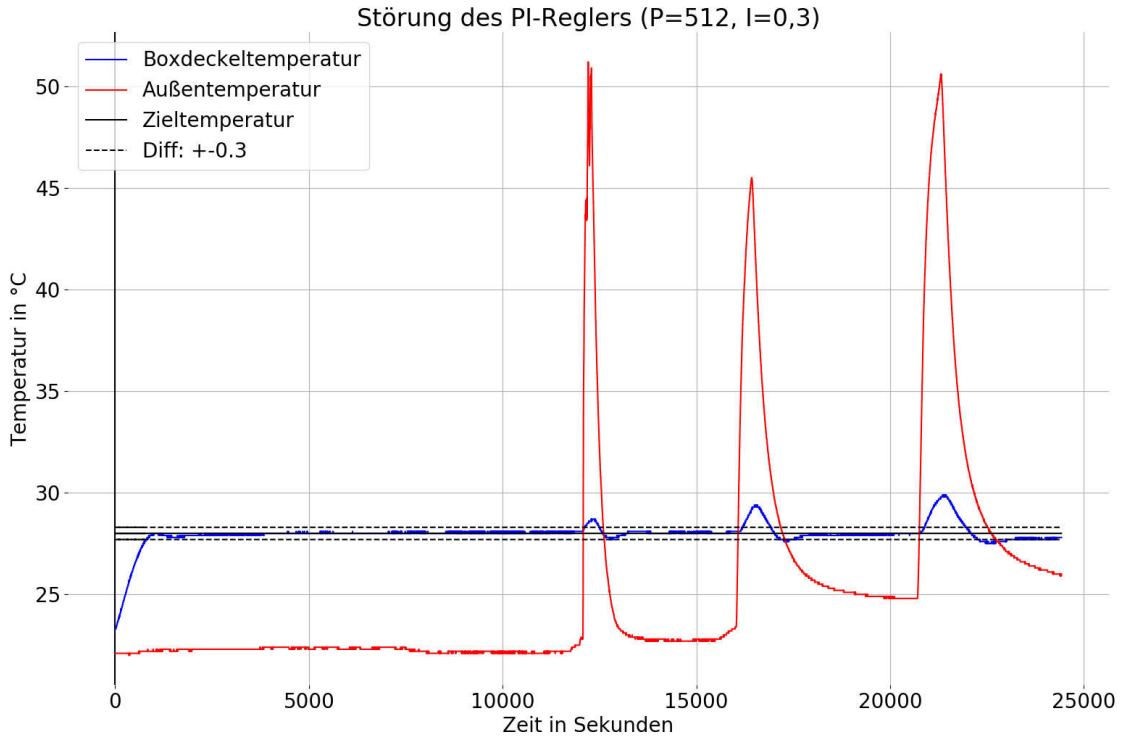


Abbildung 5.5.: Temperaturkurven mit eingebrachten Störungen.

5.4.3.1. Regelung in der ersten Hälfte

In der ersten Hälfte regelt der Regler den Temperaturwert auf den Sollwert.
In Abbildung 5.6 ist dies zu sehen. Die Außentemperatur ist recht konstant, auch wenn es eine maximale Abweichung von 0.3 °C gibt. Die Temperaturwerte am Deckel der Box nähern sich wie zuvor zügig den 28 °C an. Es ist zu sehen, dass die Temperatur am Deckel eine maximale Abweichung von ±0.1 °C besitzt, was jedoch an der Sensorgenauigkeit liegt. Die Zieltemperatur wird aber davon abgesehen konstant gehalten.

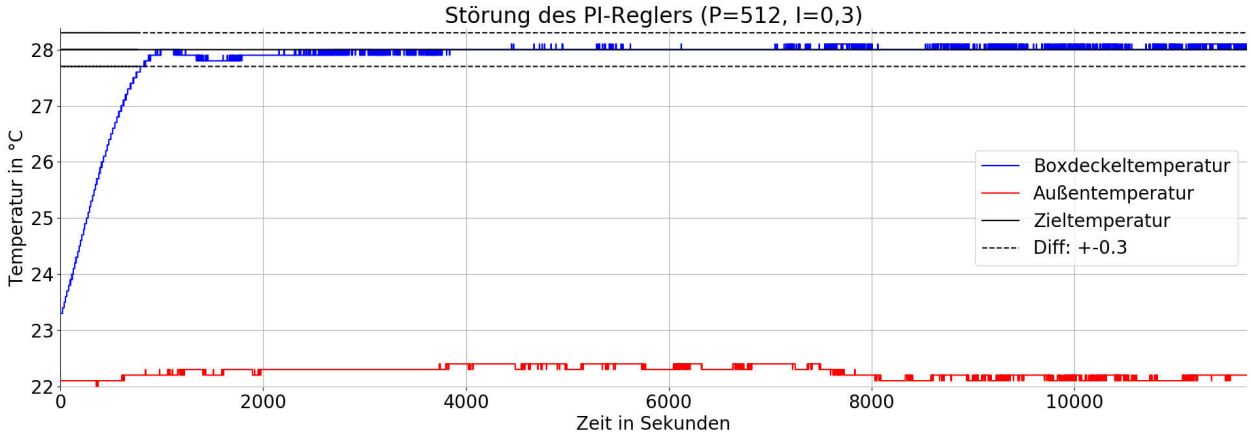


Abbildung 5.6.: Temperaturkurven ohne Störung.

5.4.3.2. Erste Störung

Bei der ersten Störung wurde, nachdem der Regler die Temperatur in der Box auf die Zieltemperatur gebracht hat, ein Heizföhn auf den Deckel gerichtet. Es wurde für 5 Minuten mit einer bei dem Heizföhn eingestellten Temperatur von 100 °C geheizt.

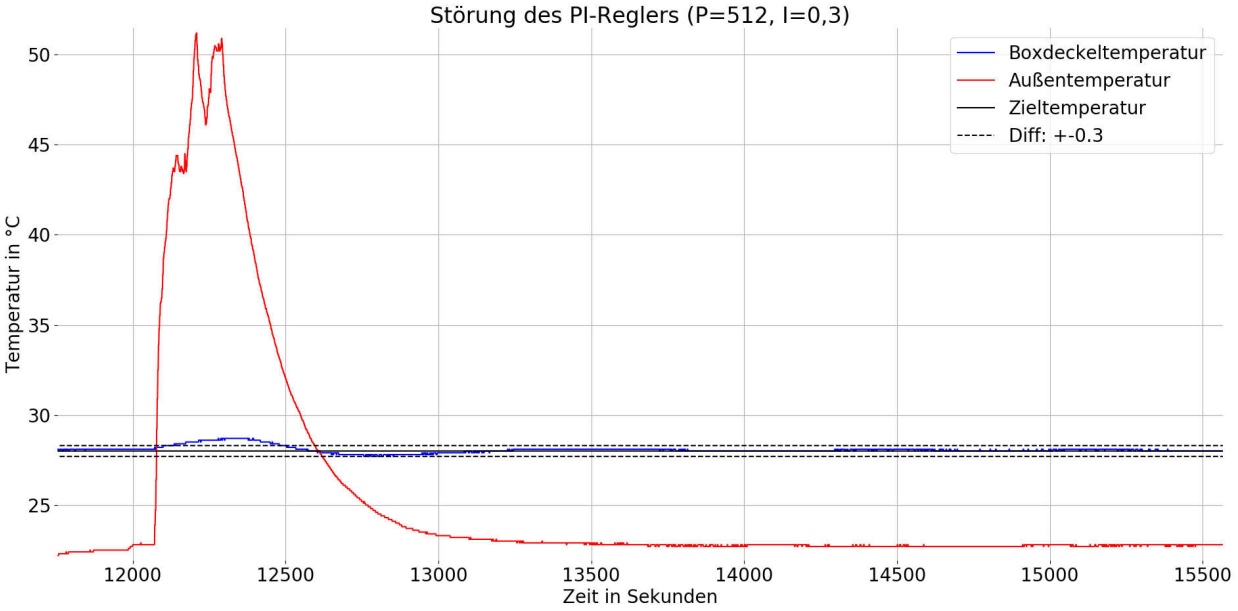


Abbildung 5.7.: Temperaturkurven für den Zeitraum der ersten Störung.

In Abbildung 5.7 sind die Temperaturkurven der ersten Störung veranschaulicht. Dabei ist gut zu erkennen, dass in der Kurve der Außentemperatur 2 Maxima vorhanden sind. Dies ist dadurch entstanden, dass der Heizföhn für einen kurzen Zeitraum nicht direkt auf den Außensensor gehalten wurde. Nach dem Heizen kann ein Temperaturabfall der Außentemperatur gemäß dem Newtonschen Abkühlungsgesetz[1] beobachtet werden.
In Abbildung 5.8 ist zu sehen, wie sich die Temperatur innerhalb der Box verändert. Dabei ist besonders gut zu erkennen, dass wenn die Box von außen punktuell erhitzt wird, diese auch innen sehr schnell wärmer wird. Dies wird hier vor allem dadurch verstärkt, dass sich der Temperatursensor in der Box auch an der Decke befindet. Weiterhin zeigt sich, dass die Temperatur in der Box nach dem Abkühlen der Außentemperatur unter die Zieltemperatur absinkt, danach etwas überschwingt und erst darauf den Sollwert erreicht. Dies ist dadurch zu erklären, dass während dem Überschwingen das Fehlerintegral des Reglers kleiner wird und der im Verhältnis zu dem I-Term große P-Term die Heizung aus stellt, da sie weit über der Zieltemperatur ist. Nach einiger Zeit unterhalb der Zieltemperatur konnte das Fehlerintegral wieder soweit anwachsen, dass der I-Term groß genug ist, dass die Zieltemperatur erreicht wird.

[1] Ein einfacher Ansatz zur Beschreibung des Temperaturausgleichs zwischen einem Körper und seiner Umgebung. Es lässt sich durch ein Anfangswertproblem beschreiben, dessen Lösung zu einem exponentiellen Abnehmen der Temperaturdifferenz führt.

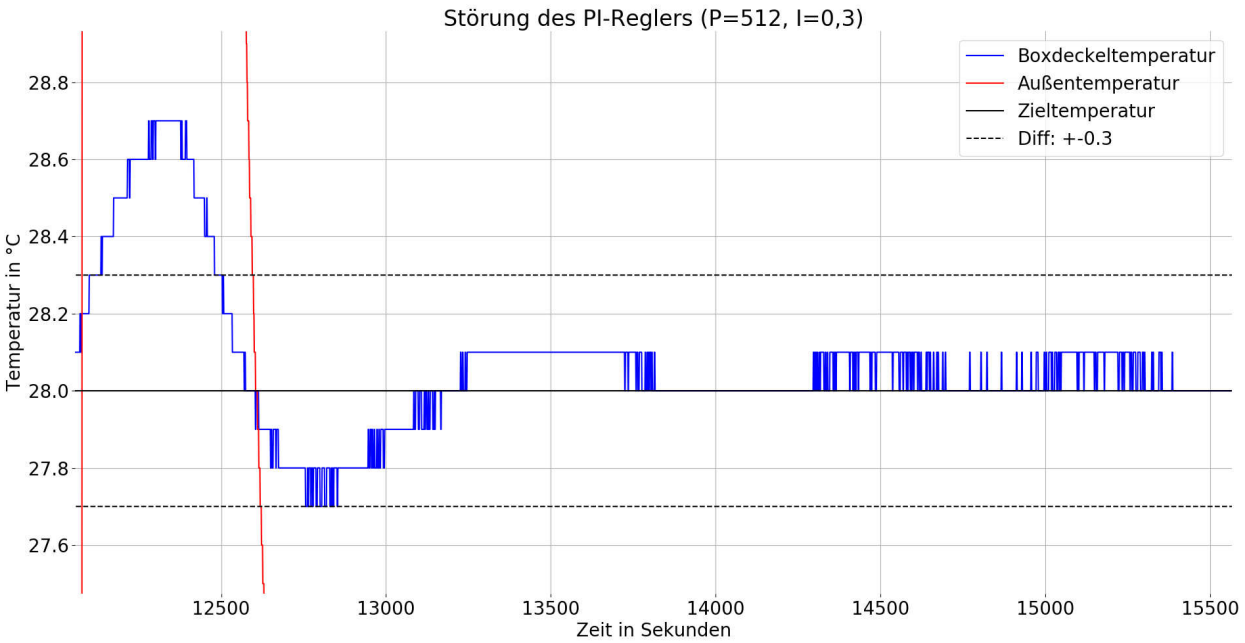


Abbildung 5.8.: Teile der Außentemperaturkurve und die Boxdeckeltemperatur der ersten Störung.

5.4.3.3. Zweite Störung

Bei der zweiten Störung wurde der Versuchsaufbau, der schon in Kapitel 2.3.2 beschrieben wurde, genutzt. Es wurde mit einem Heizföhn mit 100 °C heißer Luft die Luft zwischen den Styroporkisten erwärmt um eine gleichmäßige Außentemperatur für die innere Styroporbox herzustellen. Dabei wurde über einen Zeitraum von 5 Minuten geheizt. Die Temperatur zwischen beiden Boxen wird im Folgenden als Außentemperatur bezeichnet.

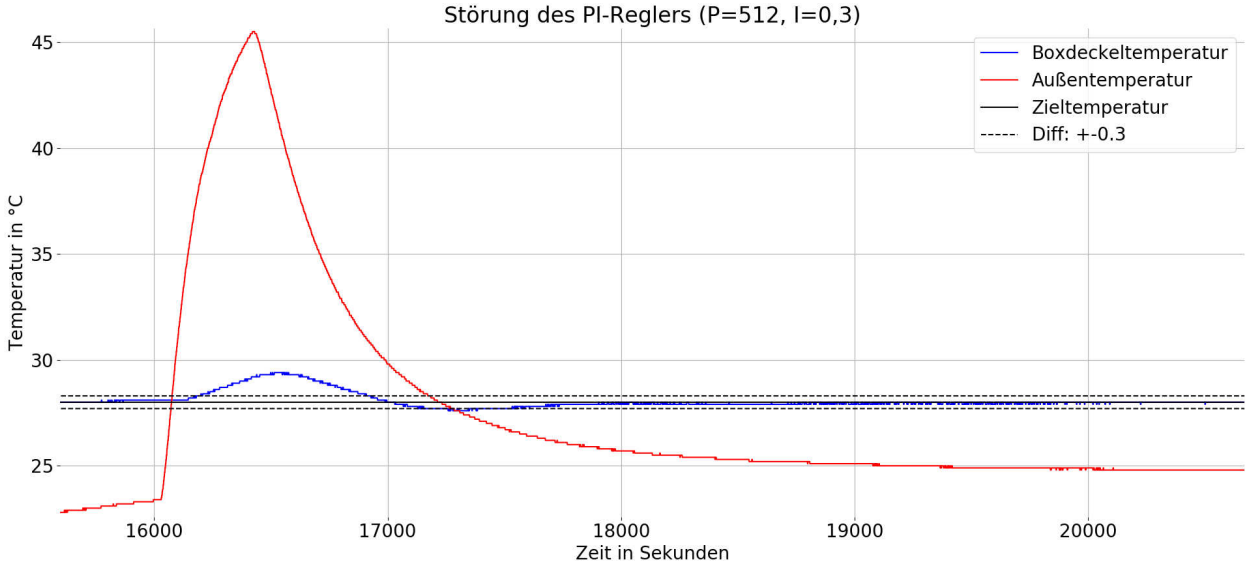


Abbildung 5.9.: Temperaturkurven der zweiten Störung.

In Abbildung 5.9 sind die Temperaturkurven der zweiten Störung zu sehen. Bei dem Aufheizen ist festzustellen, dass die Außentemperatur nach einiger Zeit nicht mehr so schnell wie zu Beginn ansteigt. Zudem kann erneut der Temperaturabfall nach Heizstopp gemäß dem Newtonschen Abkühlungsgesetz beobachtet werden. In diesem Überblick ist gut zu erkennen, dass die Temperatur im Boxdeckel erst mit einer Verzögerung auf die erwärmte Außentemperatur, mit einem Temperaturanstieg, reagiert.
In Abbildung 5.10 wird die Boxdeckeltemperatur detaillierter dargestellt. Im Vergleich zu der ersten Störung ist die Außentemperatur geringer, da jedoch die gesamte Außenfläche der inneren Box dieser Wärme ausgesetzt ist, erhitzt sich das Innere der inneren Box deutlich stärker. Nach dem Heizende fällt auch die Temperatur in der Box. Durch die Regelung hat der Heizwiderstand nicht mehr geheizt, da die Regelgröße weit über dem Sollwert war, wodurch ein Unterschwingen entsteht. Daraufhin nähert sich die Temperatur nur von unten an den Sollwert an, da das Fehlerintegral durch die zu hohe Temperatur verkleinert wurde.

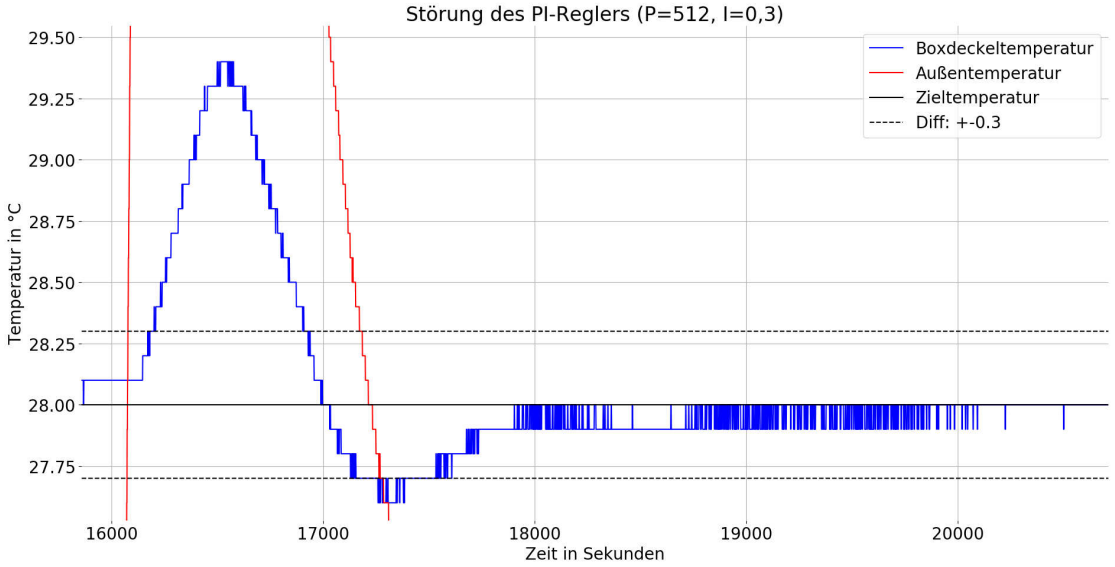


Abbildung 5.10.: Teile der Außentemperaturkurve und die Boxdeckeltemperatur in dem Zeitraum der zweiten Störung.

5.4.3.4. Dritte Störung

Im folgenden wird die dritte Störung behandelt. Der einzige Unterschied zu der zweiten Störung besteht darin, dass für 10 Minuten mit dem Heizföhn geheizt wurde. In Abbildung 5.11 ist die dritte Störung dargestellt. Die Außentemperaturkurve verhält sich wie bei der zweiten Störung. Die Boxdeckeltemperatur erwärmt sich erneut mit einer Verzögerung, was in Abbildung 5.12 zu sehen ist.
Im Vergleich zu der zweiten Störung ist die Erwärmung durch die Verlängerung des Heizens von 5 auf 10 Minuten im Maximum um etwa 0.5 °C größer. Dies führt dazu, dass sich das Fehlerintegral erneut deutlich verringert. Jedoch ist die Verringerung des Fehlerintegrals so stark, dass sie dazu führt, dass die Boxdeckeltemperatur zuerst nur 27.7 °C erreicht.

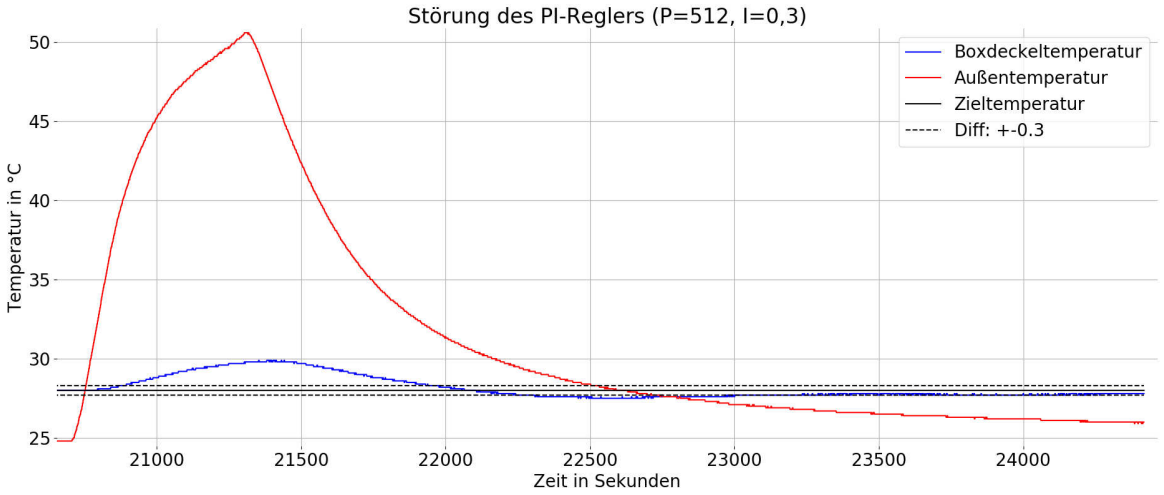


Abbildung 5.11.: Temperaturkurven der dritten Störung.

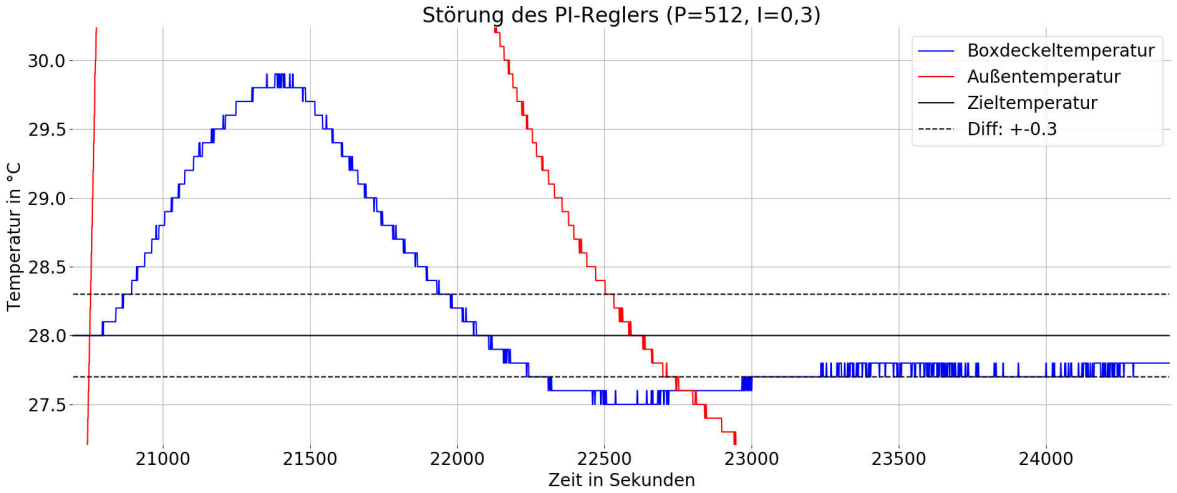


Abbildung 5.12.: Teile der Außentemperaturkurve und die Boxdeckeltemperatur in dem Zeitraum der dritten Störung.

6. Zusammenfassung der Ergebnisse

Durch die Nutzung der uMQTTBroker-Bibliothek im Rahmen des Sensornetzes ist es möglich, das Sensornetz auf Boards mit dem ESP8266 basieren zu lassen. Dies ermöglicht einen geringen Stromverbrauch und die Benutzung von LiPo-Batterien. Da die Bibliothek, entgegen der Dokumentation, nicht mit acht Clients genutzt werden konnte, wurden verschiedene Ansätze verfolgt, die Bibliothek mit acht Clients zu nutzen. Diese bestanden daraus die übertragenen Daten zu reduzieren, die Verbindungen in dem *time-wait*-Zustand zu löschen, die lwIP-Version und die Boardversion zu verändern. Diese Ansätze haben keine Veränderung bewirkt. Die Bibliothek kann nur mit fünf Clients genutzt werden. Darauf wurde die minimale Übertragungsperiodendauer getestet. Dabei wurde festgestellt, dass vier Clients problemlos alle 500 ms Daten publishen können. Da in der Anwendung jede Sekunde Temperaturwerte gepublisht werden, ist das Sensornetz noch nicht voll ausgelastet. Deshalb ist es möglich, neben den Temperaturwerten, auch noch Daten weiterer Sensoren zu publishen. Da jeder Client an einem Ort Temperaturdaten sammelt, können mit fünf Clients weniger Messpunkte im Regelkreis realisiert werden. Trotz der geringeren Anzahl an Clients ist es möglich, mit diesen Daten einen Temperaturregler zu realisieren und zudem die Temperatur in dem System an maximal vier Punkten zu beobachten.
Die Analyse des Systems hat ergeben, dass mehrere Störgrößen, die die Temperatur beeinflussen, vorhanden sind und auch eine Totzeit existiert. Der Hauptteil der Totzeit entsteht durch den Heizwiderstand, ein geringerer Teil durch die Bewegung der Luft durch das Gebläse. Zudem handelt es sich bei dem vorhandenen System um ein nichtlineares System, da mit dem Heizwiderstand das Heizen kontrolliert werden kann, das Abkühlen jedoch nicht. Eine Ersetzung des Heizwiderstandes kam bei dem vorliegenden Versuchsaufbau nicht in Frage, da andere Bauelemente deutlich mehr Strom verbrauchen, als eine Batterie, die zur Stromversorgung genutzt wird, leisten kann.
Zur Temperaturregelung, als auch zur Steuerung des Messsystems, wurde ein PI-Regler in der Programmiersprache Python entwickelt. Die Bestimmung der Regelparameter für den PI-Regler erfolgte durch Tests, bei denen ein Regelparameter unverändert blieb, während die andere variiert wurde. Die für das System besten Regelparameter sind $P = 512$ und $I = 0.3$. Mit den ermittelten Parametern ist der Regler in der Lage, trotz äußerer Störungen, die Temperatur zuverlässig nahe am Sollwert zu halten. Bei Einschalten des Systems heizt der Regler schnell auf den Sollwert. Nach einer Störung, zum Beispiel der Erhitzung der Box über den Sollwert, heizt der Regler die zunächst abkühlende Temperatur erneut auf den Sollwert.
Der vorhandene Versuchsaufbau bietet keine Möglichkeit zur Kühlung der Luft im System. Dementsprechend hört der PI-Regler auf zu heizen, wenn die Solltemperatur für längere Zeit überschritten wird. Aus diesem Grund ist darauf zu achten, dass die Zieltemperatur über der höchstmöglichen Außentemperatur gewählt wird, da der Regler ansonsten nicht in der Lage ist, die Temperatur, durch Aufheizen des Systems, zuverlässig zu halten.

7. Ausblick

Bisher konnte das entwickelte Sensornetz nicht mit mehr als fünf Clients betrieben werden. Es kann versucht werden, das Netzwerk so zu verändern, dass die Verwendung von mehr als fünf Clients möglich wird. Ein Lösungsansatz könnte eine Veränderungen in dem lwIP-Stack sein. Weiterhin kann ein Raspberry PI als MQTT-Broker genutzt werden. Dieser ist leistungsfähiger als der verwendete Mikrocontroller, jedoch benötigt dieser eine andere MQTT Bibliothek. Durch die Nutzung einer anderen Bibliothek können möglicherweise mehr als fünf Clients genutzt werden. Der Regler und die Speicherung der Sensordaten wurden über einen Desktop Computer realisiert. Für die Regelung, als auch die Speicherung der Daten, kann alternativ ein Mikrocontroller verwendet werden.
Neben den beschrieben Temperaturreglern gibt es weitere Arten von Temperaturreglern die bei dem vorhandenen System funktionieren könnten. So kann zum Beispiel ein PID-Regler implementiert werden, bei dem neben den Regelanteilen des PI-Reglers ein differentiell wirkender Anteil hinzu kommt. Daneben gibt es noch die Möglichkeit die Regelung mit einer Fuzzylogik zu implementieren. Bei dieser Reglerart wird eine verbale Beschreibung des Regelungsvorganges genutzt.
Als erweiterte Regelkreisstruktur könnte auch ein Smith-Prädiktor implementiert werden. Prädiktoren nutzen dabei direkt das Wissen des Regelstreckenmodells, um damit Vorhersagen zukünftiger Regelgrößenverläufe zu treffen. Dadurch entstehen Vorteile bei stark totzeitbehafteten Systemen. Der Nachteil dieses Prädiktors ist jedoch, dass das Regelstreckenmodell gut mit der Regelstrecke übereinstimmen muss.
Zur Verbesserung des Regelkreises könnten andere Regelparameter zu besseren Ergebnissen führen. Die Bestimmung der Parameter könnte durch eine Simulation des Systems vereinfacht werden.

A. Programmierumgebung

Als Programmierumgebung wird die Open-Source Arduino Software(IDE) verwendet.

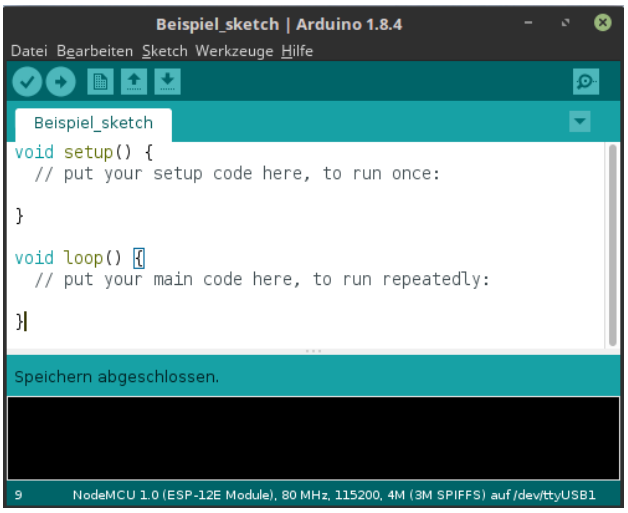


Abbildung A.1.: Die Arduino IDE.

Die Oberfläche der IDE ist in Abbildung A.1 zu sehen. Die plattformübergreifende Software läuft auf Windows, Linux und Macintosh OSX Betriebssystemen. Dies erleichtert das Schreiben von Code und das Hochladen dieses Codes auf Microcontroller.

A.1. Installation der Entwicklungsumgebung

Zuerst muss die Software für die IDE von der offiziellen Seite von Arduino https://www.arduino.cc/en/Main/Software heruntergeladen werden. Dabei kann zwischen einer .exe und einem zip-Paket ausgewählt werden. Durch die Installation mit der .exe wird alles, was für die Arduino IDE benötigt wird, installiert, dazu zählen auch Treiber.
Sobald der Download fertig ist, kann mit der Installation begonnen werden. Bei der Treiberinstallation kann eine Warnung von dem Betriebssystem kommen. Dann muss die Treiberinstallation explizit erlaubt werden. Danach können die Komponenten, die installiert werden sollen, und anschließend das Installationsverzeichnis ausgewählt werden. Dabei wird von Herstellerseite empfohlen das Default-Verzeichnis beizubehalten. Darauf folgt die Extraktion und Installation aller benötigten Daten, um die Arduino IDE vollständig nutzen zu können.

Literatur

[1] *ZEA-2 – Systemhaus für die Forschung*. URL: https://www.fz-juelich.de/zea/zea-2/DE/Home/home_node.html (besucht am 23. 07. 2018).

[2] *EMI*. URL: https://www.fz-juelich.de/zea/zea-2/DE/Forschung/Messsysteme/_node.html (besucht am 08. 08. 2018).

[3] *Adafruit Huzzah Feather Board*. URL: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-feather-huzzah-esp8266.pdf (besucht am 19. 07. 2018).

[4] *DHT22*. URL: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf (besucht am 19. 07. 2018).

[5] *Lithium-Ion Polymer Batterie*. URL: https://eckstein-shop.de/LiPo-Akku-Lithium-Ion-Polymer-Batterie-37V-1100mAh-JST-PH-Connector (besucht am 25. 07. 2018).

[6] *tp-link TL-WR702N WLAN-Router*. URL: https://www.tp-link.com/de/products/details/TL-WR702N.html (besucht am 25. 07. 2018).

[7] Hsiang-Wen Chen und Fuchun Joseph Lin. "Converging MQTT Resources in ETSI Standards Based M2M Platform". In: *2014 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, and IEEE Cyber, Physical and Social Computing, iThings/GreenCom/CPSCom 2014, Taipei, Taiwan, September 1-3, 2014*. IEEE Computer Society, 2014, S. 292–295. ISBN: 978-1-4799-5967-9. DOI: 10.1109/iThings.2014.52. URL: https://doi.org/10.1109/iThings.2014.52.

[8] Manuel Guttenberger und Klemens Waldhör. "xHealth: Eine MQTT und REST basierte Architektur zum Zugriff auf Sensordaten smarter Objekte". In: *46. Jahrestagung der Gesellschaft für Informatik, Informatik 2016, 26.-30. September 2016, Klagenfurt, Österreich*. Hrsg. von Heinrich C. Mayr und Martin Pinzger. Bd. P-259. LNI. GI, 2016, S. 1851–1864. ISBN: 978-3-88579-653-4. URL: https://dl.gi.de/20.500.12116/1070.

[9] Dinesh Thangavel u. a. "Performance evaluation of MQTT and CoAP via a common middleware". In: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 21-24, 2014*. IEEE, 2014, S. 1–6. ISBN: 978-1-4799-2843-9. DOI: 10.1109/ISSNIP.2014.6827678. URL: https://doi.org/10.1109/ISSNIP.2014.6827678.

[10] Patrick Th. Eugster u. a. "The many faces of publish/subscribe". In: *ACM Comput. Surv.* 35.2 (2003), S. 114–131. DOI: 10.1145/857076.857078. URL: http://doi.acm.org/10.1145/857076.857078.

[11] Gastón C. Hillar. *MQTT Essentials - A Lightweight IoT Protocol*. Packt Publishing, 2017. ISBN: 9781787287815.

[12] *uMQTTBroker*. URL: https://github.com/martin-ger/uMQTTBroker (besucht am 18. 07. 2018).

[13] *martin-ger*. URL: https://github.com/martin-ger (besucht am 18. 07. 2018).

[14] *esp MQTT*. URL: https://github.com/tuanpmt/esp_mqtt (besucht am 18. 07. 2018).

[15] *Tuan PM*. URL: https://github.com/tuanpmt (besucht am 18. 07. 2018).

[16] *Beispiel MQTT-Broker Sketch für den ESP8266*. URL: https://github.com/martin-ger/uMQTTBroker/blob/724bf8f8308cb251d985cb31e0c87440d5bb50d8/examples/uMQTTBrokerSample/uMQTTBrokerSample.ino (besucht am 24. 07. 2018).

[17] *Beispiel MQTT-Client Sketch für den ESP8266*. URL: https://github.com/i-n-g-o/esp-mqtt-arduino/tree/master/examples (besucht am 24. 07. 2018).

[18] *ESP8266 Arduino Kern FAQ*. URL: http://arduino-esp8266.readthedocs.io/en/latest/faq/readme.html#how-to-clear-tcp-pcbs-in-time-wait-state (besucht am 04. 08. 2018).

[19] Hildebrand Walter. *Grundkurs Regelungstechnik*. 3. Aufl. Springer Vieweg, 2013. ISBN: 9783834814203.

[20] *2014 IEEE International Conference on Internet of Things, IEEE Green Computing and Communications, and IEEE Cyber, Physical and Social Computing, iThings-/GreenCom/CPSCom 2014, Taipei, Taiwan, September 1-3, 2014*. IEEE Computer Society, 2014. ISBN: 978-1-4799-5967-9. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7058329.

[21] Heinrich C. Mayr und Martin Pinzger, Hrsg. *46. Jahrestagung der Gesellschaft für Informatik, Informatik 2016, 26.-30. September 2016, Klagenfurt, Österreich*. Bd. P-259. LNI. GI, 2016. ISBN: 978-3-88579-653-4. URL: https://dl.gi.de/handle/20.500.12116/616.

[22] *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 21-24, 2014*. IEEE, 2014. ISBN: 978-1-4799-2843-9. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6820824.

[23] *Heizwiderstand Datenblatt*. URL: https://docs-emea.rs-online.com/webdocs/15a2/0900766b815a2313.pdf (besucht am 07. 08. 2018).

Jül-4417 • Dezember 2018
ISSN 0944-2952

Mitglied der Helmholtz-Gemeinschaft