

Universität für Baukunst und Metropolenentwicklung
Studiengang Geomatik

Bachelor-Thesis

Low-Cost Umweltmonitoring mit Arduino in Geosensornetzwerken

Vorgelegt am: 17. Dezember 2018

von: Riccardo Samuel Oppermann
geboren am 18.02.1994
in Hamburg

Erster Prüfer: Prof. Dr.-Ing. Harald Sternberg
Zweiter Prüfer: Dipl.-Umweltwiss. Dipl.-Ing. Günter Eppinger

Zusammenfassung

Hochpräzise Vermessungsarbeiten erfordern oft eine genaue Kenntnis des Mediums Luft. Temperatur, Luftdruck und relative Luftfeuchte beeinflussen die benötigte Laufzeit von Wellen, die für Streckenmessungen genutzt werden. Das Ziel dieser Arbeit ist, eine kostengünstige Lösung zu entwickeln, die es ermöglicht, flächendeckend Korrekturwerte für hochpräzise Messungen zu liefern. Dazu wurde mithilfe der Hard- und Softwareplattform Arduino ein Prototyp entwickelt, welcher Temperatur, Luftdruck und Luftfeuchte messen und an eine zentrale Stelle via WLAN weiterleiten kann. Verschiedene Sensoren wurden hinsichtlich ihrer Genauigkeit untersucht und weitere für eine ausfallsichere Datenerfassung benötigte Komponenten ausgewählt. Ein Programm für die Steuerung der Messstationen sowie eines für den zentralen Server wurde entwickelt. Das Ergebnis ist ein Sensornetzwerk, dessen Knoten Materialkosten von etwas über 10 € aufweisen und trotzdem genaue Messwerte erfassen und aufzeichnen.

Abstract

High-precision surveying work often requires precise knowledge of the medium air. Temperature, air pressure and relative humidity influence the required running time of waves used for distance measurements. The aim of this work is to develop a low-cost solution that allows to provide area-wide correction values for high-precision measurements. Using the Arduino hard- and software platform a prototype, which can measure temperature, air pressure and humidity and transmit them to a central server via WLAN, was developed. Various sensors were examined with regard to their accuracy and further components required for fail-safe data acquisition were selected. A program to control the sensor-stations as well as one for the central server was developed. The result is a sensor network, whose nodes have a material cost of just over 10 € and still record and save accurate measured values.

Danksagung

An dieser Stelle möchte ich mich bedanken bei den Herren Prof. Dr.-Ing. Harald Sternberg und Dipl.-Umweltwiss. Dipl.-Ing. Günter Eppinger, die mich bei der Anfertigung dieser Arbeit unterstützt haben.

Besonderer Dank geht auch an meine Frau Lisa, die mich während meines gesamten Studiums motiviert und unterstützt hat.

Ebenfalls möchte ich mich bedanken bei meinem Vater Jens Oppermann, der mir nicht nur mit Interesse und Ideen zur Seite stand, sondern auch die Referenzmessung mit dem Testo Präzisionsfühler an der TU-Hamburg auf schnelle und unkomplizierte Art und Weise ermöglicht hat.

Abschließend danke ich auch meiner Mutter Sabine Oppermann, die ihre Zeit eingesetzt hat, mir bei der Korrekturlesung dieser Arbeit zu helfen.

Inhaltsverzeichnis

1 Einleitung	6
2 Die Plattform Arduino	8
2.1 Software	8
2.2 Hardware	9
2.3 Die I ² C Schnittstelle	10
2.4 Die SPI Schnittstelle	11
3 Auswahl der Module	12
3.1 ESP8266 / ESP12-E	12
3.2 Vergleich gängiger Sensoren	13
3.2.1 BMP085	13
3.2.2 BMP180	13
3.2.3 BMP280 und BME280	14
3.2.4 SHT31-D	14
3.2.5 DS18B20	14
3.2.6 Vergleich der Sensoren	14
3.3 RTC DS3231	15
3.4 Display SH1106	16
3.5 Sonstige Bauteile	16
3.5.1 RGB-LED	16
3.5.2 Reset-Button	17
3.6 Kosten der Module und Bauteile	17
3.7 Layout und Anschluss der Module	18
4 Genauigkeit des BMP180, BME280 und SHT31-D	20
4.1 Genauigkeit der Temperaturmessung	20
4.2 Genauigkeit der Feuchtemessung	22
4.3 Genauigkeit der Luftdruckmessung	23
4.4 Einsatz mehrerer Sensoren pro Station	25

5 Programmierung der Station	26
5.1 Vorbereitung der IDE	26
5.2 Verwendete Bibliotheken	26
5.3 Erklärungen zum Code	27
5.3.1 setup()	27
5.3.2 loop()	29
5.4 Einrichtung einer Station	32
6 Programmierung des Servers	33
6.1 PHP und MySQL	33
6.2 Struktur der Datenbank	33
6.3 Erklärungen zum Code	34
6.3.1 Echtzeitübertragung der Messwerte	35
6.3.2 Übertragung von zwischengespeicherten Werten	37
6.4 Anzeige der Daten	38
7 Fazit und Ausblick	40
7.1 Vergleich mit bestehenden Lösungen am Markt	40
7.1.1 Netatmo Wetterstation	40
7.1.2 Froggit WH3000 SE	41
7.1.3 Vergleich der Genauigkeiten	41
7.2 Zusammenfassung der Ergebnisse	42
7.2.1 Veröffentlichung der Ergebnisse	44
7.3 Ausblick und mögliche Erweiterungen	44
7.3.1 Deepsleep	44
7.3.2 weitere Sensoren	44
Abkürzungsverzeichnis	45
Abbildungsverzeichnis	46
Literaturverzeichnis	47

1 Einleitung

Unsere Umwelt beeinflusst uns jeden Tag aufs neue: wie wir uns kleiden, was wir essen oder welcher Freizeitbeschäftigung wir nachgehen. Daher ist es logisch, dass der Mensch diese Umwelteinflüsse sehr genau beobachtet und aufzeichnet. Dieser Prozess wird als Umweltmonitoring bezeichnet. Gemäß der Brockhaus-Enzyklopädie umfasst Umweltmonitoring die “Gesamtheit des Sammelns und Überwachens umwelt- und gesundheitsrelevanter Daten, die mithilfe technischer Messnetze sowie biotischer und abiotischer Reaktionsindikatoren durchgeführt werden“ (Brockhaus 2018).

Im Bereich der Geodäsie werden heutzutage häufig Laufzeitmessungen durchgeführt, sei es bei der elektrooptischen Distanzmessung mit einem Tachymeter oder einer Positionsbestimmung mithilfe von globalen Navigationssystemen. Die Genauigkeit solcher Laufzeitmessungen ist stark an die Bekanntheit des durchlaufenen Mediums, zum Beispiel die Luft zwischen Satellit und Empfänger, geknüpft (Joeckel et al. 2008, S. 95). Insbesondere Temperatur, Luftdruck und Feuchte in der Luft verändern die Geschwindigkeit, mit der sich eine Welle fortbewegt. Daher müssen diese Größen bei hochpräzisen Messungen genau bestimmt und in Echtzeit oder im Post-Processing berücksichtigt werden.

Da das Wetter aber keine homogene Ausbreitung aufweist, kann es eine Herausforderung darstellen, den exakten Zustand des Mediums Luft an dem Ort der Messung zu bestimmen. Zwar existieren bereits Messnetzwerke, wie zum Beispiel das Bodenmessnetz des Deutschen Wetterdienstes, allerdings sind diese Netze meist zu grobmaschig für die Nutzung in hochpräzisen Messungen. An diesem Punkt setzt diese Bachelor-Thesis an und beschreibt den Entwurf einer kostengünstigen, auf Arduino-basierenden Lösung zur Erfassung von Umweltdaten mithilfe von Sensorstationen, welche in kleiner oder großer Zahl innerhalb eines Sensornetzwerkes eingesetzt werden können.

Im folgenden wird untersucht, wie die Hard- und Softwareplattform Arduino die Grundlage für eine Sensorstation bilden und die Kommunikation mit dem Sensor

ermöglichen kann. Auch werden Microchips für die Kommunikation per WLAN mit einem Server und die Zwischenspeicherung von Daten bei einem Verbindungs ausfall ausgewählt und vorgestellt. Verschiedene Sensoren wurden hinsichtlich ihrer Messgenauigkeit untersucht, dafür wurden Referenzmessungen durchgeführt und ausgewertet. Das entwickelte Programm, welches auf jeder einzelnen Sensorstation läuft, wird vorgestellt und die damit verbundenen Entscheidungen und Überlegungen werden erklärt. Auch das auf PHP und MySQL basierende Serverprogramm wird auf diese Art vorgestellt. Im Schlussteil dieser Arbeit wird die entwickelte Lösung mit bereits bestehenden, kommerziellen Lösungen verglichen und es werden mögliche Erweiterungen besprochen.

2 Die Plattform Arduino

Arduino ist eine Open-Source-Plattform, die auf einfach handzuhabender Hardware und Software basiert. Zielgruppen sind vor allem Einsteiger und fachfremde Personen. Seinen Ursprung hat Arduino im Jahr 2005 am *Ivrea Interaction Design Institute* in Norditalien, wo es von Massimo Banzi und David Cuartielles initiiert wurde. Dort sollte es Studenten ohne große Kenntnisse in den Bereichen Elektrotechnik und Softwareentwicklung helfen, schnell und einfach kleine Prototypen zu entwerfen (Dembowski 2014, S. 9).

Mittlerweile ist Arduino zu einer beliebten Plattform für sowohl Hobbybastler als auch professionelle Anwender herangewachsen. Dies liegt unter anderem auch an der großen Community, die sich gebildet hat und fleißig mit Rat unterstützt sowie Anleitungen für interessante Projekte publiziert. Ende 2018 verzeichnet das offizielle Internet-Forum 3,8 Millionen Beiträge von über 670 000 registrierten Benutzern (Arduino AG 2018a).

2.1 Software

Auf der Homepage der Arduino AG¹ kann kostenlos die Arduino IDE² für Windows, Mac oder Linux heruntergeladen werden. Mithilfe der IDE kann Arduino-Code, ein vereinfachter C-Dialekt (Dembowski 2014, S. 10), geschrieben, überprüft und auf ein Arduino-Board übertragen werden.

Dank der großen Community werden für nahezu jede Aufgabe sogenannte Bibliotheken bereitgestellt. Diese Bibliotheken stellen weitere Funktionen zur Verfügung und erleichtern so zum Beispiel die Ansteuerung von Sensoren, Motoren oder Displays.

¹<https://www.arduino.cc/en/Main/Software>

²Integrated Development Environment

2.2 Hardware

Mittlerweile werden viele verschiedene Boards angeboten, die Arduino-Code ausführen können. Zu den bekanntesten zählt das Arduino Uno (siehe Bild 2.1).

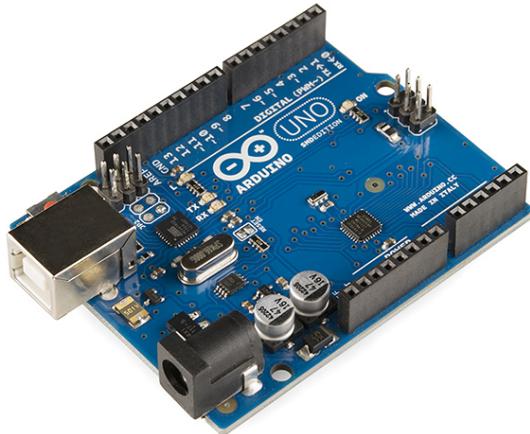


Abb. 2.1: Arduino Uno - R3

Das Uno basiert auf dem Microchip ATmega328P von Atmel. Damit besitzt es 32 Kilobyte Flash-Speicher sowie 2 Kilobyte Arbeitsspeicher und taktet mit 16 Megahertz. Mit der Zeit wurden die Programme allerdings größer und anspruchsvoller, sodass der Speicherplatz und Prozessortakt nicht mehr ausreichte. Neue Boards wie zum Beispiel das Arduino Due mit 512 Kilobyte Speicher und 84 Megahertz wurden angeboten. Der offizielle Shop³ listet mittlerweile mehr als 30 verschiedene Boards für teils sehr verschiedene Anwendungsfälle. So haben manche Boards einen Internetanschluss via RJ-45 Buchse, WLAN⁴ oder GSM⁵, manche sind sehr klein und wieder andere haben besonders viele I/O Pins. Neben den offiziellen Boards der Arduino AG existieren auch noch viele weitere sogenannte Arduino-kompatible Boards, die ebenfalls mit der Arduino-IDE programmiert werden können. Solch ein kompatibles Board wird auch in diesem Projekt eingesetzt.

Das Board an sich übernimmt erst einmal nur die Speicherung und Ausführung des Programms sowie die Kommunikation mit weiterer Hardware. Im nächsten Schritt werden sogenannte Module an das Board angeschlossen. Hierbei gibt es fast endlose

³<https://store.arduino.cc/>

⁴Wireless Local Area Network

⁵Global System for Mobile Communications

Kombinationen aus LED⁶s, Motoren, Schaltern, Sensoren, Lautsprechern, Displays und mehr. Je nachdem, was für eine Aufgabe erfüllt werden soll, wird die passende Hardware angeschlossen. Firmen wie Adafruit⁷ oder Sparkfun⁸ bieten eine große Auswahl an Breakout-Boards, kleine Platinen, die alle benötigten Komponenten beinhalten und unkompliziert mit dem Arduino-Board verbunden werden können. Ebenso stellen diese Firmen oft Anleitungen und Beispielcode bereit, sodass es auch Anfängern leicht fällt, diese Module zu nutzen.

Ebenfalls werden Shields angeboten, dabei handelt es sich um Platinen, die auf bestimmte Arduino-Boards direkt aufgesteckt werden können und so weitere Funktionen wie etwa WLAN, Relais oder eine Motorsteuerung bereitzustellen. Bei Shields muss darauf geachtet werden, dass diese auch auf das genutzte Board passen.

2.3 Die I²C Schnittstelle

Der I²C⁹ Standard wurde von der Firma Philips entwickelt (Dembowski 2014, S. 218), um den Austausch von digitalen Informationen zwischen Microchips und Peripherie-Chips, zum Beispiel Sensoren, möglichst einfach zu gestalten (Margolis 2012, S. 421). Alle gängigen Arduino-Boards unterstützen diesen Standard.

Das I²C-Bussystem funktioniert nach dem Master-Slave-Prinzip. Der Microchip stellt in der Regel den Master dar, an welchen dann unterschiedlichste I²C-Chips angeschlossen werden. Bei diesem Projekt ist also der ESP-12E (siehe Kapitel 3.1) der Master, an den die Slaves BME280, DS3231 sowie das I²C-Display (siehe Kapitel 3.3) angeschlossen sind. Dabei kann die Kommunikation zwischen den Chips bidirektional erfolgen, allerdings nicht gleichzeitig in beide Richtungen. I²C benötigt zwei Leitungen, weshalb es auch manchmal als Two Wire Interface bezeichnet wird. Eine Leitung wird als SDA (Serial Data Line) und die andere als SCL (Serial Clock Line) bezeichnet. Die SLC-Leitung überträgt das Taktsignal, wohingegen die SDA-Leitung die eigentlichen Daten überträgt. I²C unterstützt verschiedene Übertragungsdatenraten, von 100 Kbit/s im Standard-Mode bis zu 5 Mbit/s im Ultra-Fast-Mode (Dembowski 2014, S. 218).

⁶Light Emitting Diode

⁷<https://www.adafruit.com/>

⁸<https://www.sparkfun.com/>

⁹Inter-Integrated Circuit

Jeder Slave-Chip wird über eine Adresse angesprochen. Diese Adresse ist 7 Bit lang, was theoretisch 128 verschiedene Adressen ermöglicht. Da jedoch schon 16 Adressen reserviert sind, ergeben sich 112 in der Praxis nutzbare Adressen (Dembowski 2014, S. 219). So können also 112 Geräte mit unterschiedlichen Adressen angesprochen werden. Allerdings haben viele I²C-Chips feste Adressen, sodass an einem Master nur eines dieser Geräte genutzt werden kann. Andere I²C-Chips geben die Möglichkeit, zwischen mehreren Adressen zu wechseln, indem bestimmte Pins geerdet werden. So können dann mehrere Chips desselben Typs an einem Master betrieben werden. Eine weitere Möglichkeit, mehrere I²C-Chips mit gleicher Adresse an einem Master zu betreiben, wird im Kapitel 4.4 erläutert.

2.4 Die SPI Schnittstelle

SPI¹⁰ wurde von der Firma Motorola entwickelt und sollte wie I²C die Kommunikation zwischen Microcontroller und Peripherie ermöglichen (Dembowski 2014, S. 206). SPI ermöglicht höhere Datenraten und eine gleichzeitige bidirektionale Kommunikation zwischen Master und Slave. Daher wird dieser Standard gerne für Anwendungen wie zum Beispiel Displays, Speicherkarten oder Ethernet-Adapter verwendet. Allerdings benötigt SPI für die Kommunikation mit einem Peripherie-Chip mindestens vier Leitungen und für jeden weiteren Chip eine zusätzliche Leitung (Margolis 2012, S. 421).

Für dieses Projekt ist daher die I²C-Schnittstelle interessanter, da auch bei mehreren angeschlossenen Chips nur zwei Leitungen benötigt werden. Auch werden keine hohen Datenraten benötigt, da nur ab und zu einige Messwerte übertragen werden müssen.

¹⁰Serial Peripheral Interface

3 Auswahl der Module

3.1 ESP8266 / ESP12-E

Bei der Überlegung, welcher Microcontroller die Grundlage für dieses Projekt bilden sollte, spielten Gedanken zu den Kosten, der Größe und der Leistungsfähigkeit eine Rolle. Die klassischen Microcontroller wie der ATmega328P oder ATmega2560 boten nicht genug Leistung und Speicher für dieses Projekt. Außerdem hätte ein zusätzlich benötigtes WIFI-Shield weitere Kosten verursacht.

Daher fiel die Wahl auf den ESP8266 von Espressif Systems, beziehungsweise den darauf basierenden ESP-12E von Ai-Thinker. Mit seinen 80 MHz Taktfrequenz, 4 MB Speicher, Onboard-WLAN, I²C Unterstützung und mehr als 10 GPIO¹-Pins scheint er ideal für dieses Projekt zu sein (Ai-Thinker 2015, S. 4-9).

Da der ESP-12E allein aber aufgrund seiner kompakten Bauweise nur schwer zu nutzen ist, wurde das Open Source Board NodeMCU in der Version 2 (Amica) für die weitere Entwicklung verwendet. Dieses Board stellt zusätzlich einen Spannungsregler, USB-Anschluss und besser erreichbare Pins bereit. Ursprünglich wurde es für Lua Script entwickelt, bietet aber auch volle Arduino-Kompatibilität. Zum Zeitpunkt dieser Arbeit ist auch schon eine Version 3 (LoLin) des NodeMCU erhältlich, diese bietet aber keine nennenswerten Vorteile und ist mit 58 mm × 31 mm × 13 mm auch noch um einiges größer als die Version 2 mit 48 mm × 26 mm × 13 mm.

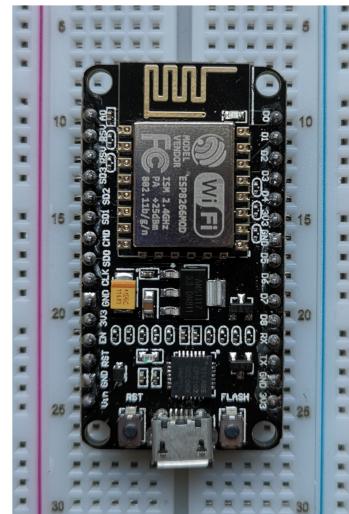


Abb. 3.1: NodeMCU

¹General Purpose Input/Output

3.2 Vergleich gängiger Sensoren

In diesem Unterkapitel sollen gängige, kostengünstige Sensoren miteinander verglichen werden. Um nicht allzu weit auszuholen, werden nur einige Sensoren aufgegriffen, die Temperatur, Luftdruck oder relative Feuchte messen.

3.2.1 BMP085

Der BMP085 von Bosch Sensortec ist ein digitaler Luftdrucksensor, der im Oktober 2009 auf den Markt kam. Seine Funktionsweise basiert auf dem piezoresistiven Effekt. Dieser Sensor kann bei Temperaturen zwischen -40°C und 85°C betrieben werden, erreicht seine größte Genauigkeit allerdings zwischen 0°C und 65°C . Für den Betrieb ist eine Spannung von 3.3 V nötig, allerdings verbauen die meisten Hersteller von Breakout Boards mit diesem Sensor Spannungswandler, sodass auch ein Betrieb mit 5 V möglich ist.

Die Luftdruck- und Temperaturwerte können über eine I²C-Schnittstelle abgerufen werden. Der Luftdruck wird dabei in Hundertstel Hektopascal und die Temperatur in Zehntel Grad Celsius aufgelöst. Die Genauigkeit der Temperaturnessung beträgt bei 25°C typischerweise $\pm 0.5^{\circ}\text{C}$, zwischen 0°C und 65°C typischerweise $\pm 1^{\circ}\text{C}$. Die Luftdruckmessung erfolgt in normalen atmosphärischen Bereichen mit einer Genauigkeit von $\pm 1 \text{ hPa}$. (Bosch Sensortec 2009, S. 6-7)

3.2.2 BMP180

Der BMP180 wurde ebenfalls von Bosch entwickelt und stellt den Nachfolger des BMP085 dar. Er kam im April 2013 auf den Markt. Im direkten Vergleich fällt auf, dass der BMP180 mit seinen 3.6 mm x 3.8 mm etwas kleiner als der BMP085 mit 5 mm x 5 mm ist.

Ansonsten hat sich an den technischen Daten in Bezug auf die Genauigkeit nichts geändert, neu ist allerdings die Angabe zur relativen Genauigkeit der Luftdruckmessung. Diese fällt nämlich mit $\pm 0.12 \text{ hPa}$ in normalen atmosphärischen Bereichen recht genau aus. Dieser Sensor eignet sich also auch für präzise Bestimmungen von Höhenänderungen über den Luftdruck. (Bosch Sensortec 2013, S. 6-7)

3.2.3 BMP280 und BME280

Die 280er Serie von Bosch fällt mit 2.5 mm x 2.55 mm noch einmal etwas kleiner aus. Während der BMP280 nur Druck und Temperatur messen kann, ist der BME280 zusätzlich dazu auch noch in der Lage, die Luftfeuchte zu messen. Die Temperatur wird wie auch schon beim BMP085 und BMP180 mit $\pm 1^{\circ}\text{C}$ im Bereich zwischen 0°C und 65°C gemessen. Auch die relative und absolute Luftdruckmessung erfolgt mit derselben Genauigkeit des BMP180, $\pm 1 \text{ hPa}$ beziehungsweise $\pm 0.12 \text{ hPa}$. Die Luftfeuchte bestimmt der BME280 im Bereich von 20 %rF bis 80 %rF mit einer Genauigkeit von $\pm 3\%$ rF. Auch die Daten des BME280 können über eine I²C-Schnittstelle abgerufen werden, zusätzlich dazu wird auch SPI unterstützt. (Bosch Sensortec 2015, S. 7-10)

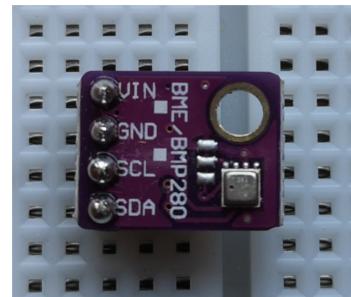


Abb. 3.2: BME280

3.2.4 SHT31-D

Der SHT31-D von Sensirion ist ein weit verbreiteter Feuchtesensor, der im Jahr 2014 auf den Markt kam. Mit 2.5 mm x 2.55 mm ist er recht klein, und dank des breiten Betriebsspannungsbereich von 2.15V bis 5.5V sehr anwenderfreundlich. Dieser Sensor misst die relative Luftfeuchtigkeit laut Hersteller typischerweise mit einer Genauigkeit von $\pm 2\%$ rF im Bereich von 0 %rF bis 100 %rF, die Temperatur mit hochpräzisen $\pm 0.2^{\circ}\text{C}$ im Bereich zwischen 0°C und 90°C .

Als Schnittstelle unterstützt der Sensor I²C. (Sensirion 2015, S. 1-4)

3.2.5 DS18B20

Der digitale Temperatursensor DS18B20 wurde ursprünglich von Dallas Semiconductor entwickelt, diese wurden dann allerdings von Maxim Integrated aufgekauft. Dieser Sensor liefert im Bereich von -10°C bis 85°C Temperaturen mit einer Genauigkeit von $\pm 0.5^{\circ}\text{C}$. Außerdem wird er über ein sogenanntes One-Wire-Interface angesprochen, benötigt also nur eine Leitung zur Datenübertragung. (Maxim Integrated 2008, S. 1)

3.2.6 Vergleich der Sensoren

Um die Sensoren einfach vergleichen zu können, wurde die Tabelle 3.1 erstellt. Alle typischen Genauigkeiten finden sich hier noch einmal aufgelistet, ebenso wurden

die Preise für ein Breakout-Board mit dem jeweiligen Sensor ermittelt. “Kosten Deutschland“ nennt den günstigsten Preis inklusive Versandkosten mit einer Lieferzeit von wenigen Tagen innerhalb Deutschlands. “Kosten China“ nennt den Preis des Moduls inklusive Versand auf der Plattform AliExpress. Hierbei ist natürlich eine längere Lieferzeit von etwa 3-4 Wochen zu beachten, allerdings kann so der Preis einer Messstation stark gesenkt werden.

Letztendlich fiel die Wahl auf den BME280 von Bosch. Dieser Sensor ist in der Lage, alle benötigten Werte mit einer zufriedenstellenden Genauigkeit zu erfassen. Außerdem liegen die Kosten im Low-cost-Bereich und der Sensor kann einfach per I²C angesteuert werden.

	BMP085	BMP180	BME280	SHT31-D	DS18B20
Temperatur	±1 °C	±1 °C	±1 °C	±0.2 °C	±0.5 °C
Luftdruck	±1 hPa	±1 hPa	±1 hPa	-	-
Feuchte	-	-	±3 %rF	±2 %rF	-
Schnittstelle	I ² C	I ² C	I ² C, SPI	I ² C	1-Wire
Kosten Deutschl.	3.00 €	5.00 €	7.00 €	8.00 €	0.70 €
Kosten China	0.50 €	0.70 €	2.50 €	3.70 €	0.50 €

Tabelle 3.1: Vergleich der Genauigkeiten und Kosten

3.3 RTC DS3231

Damit jede Station im Falle eines WLAN-Ausfalls weiterhin Daten aufzeichnen kann, wird für die Zwischenspeicherung eine Uhrzeit beziehungsweise ein Zeitstempel benötigt. Da die Uhrzeit nicht mehr über das Internet abgerufen werden kann und die Schwingungen des Microcontrollers meist zu ungenau sind, wurde eine RTC² in die Station integriert. Diese kann auch über mehrere Wochen oder Monate präzise Zeitangaben liefern, ohne dabei stark zu drifteten.

Der DS3231 von Maxim Integrated ist eine günstige und hochpräzise Echtzeituhr. Hochpräzise, da der Quarzoszillator durch einen integrierten Temperatursensor ausgeglichen wird. Dadurch wird eine Abweichung von maximal ±2 ppm zwischen 0 °C und 40 °C sowie ±3.5 ppm zwischen –40 °C und 85 °C ermöglicht. Angesprochen

²Real-Time Clock

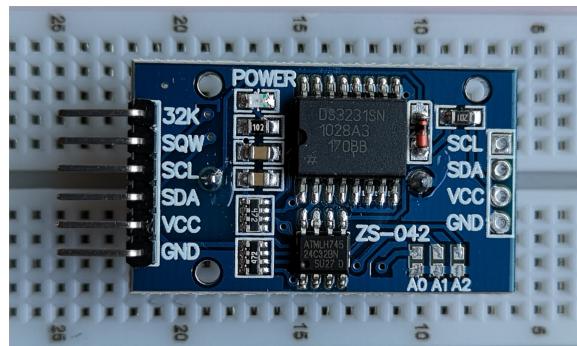


Abb. 3.3: RTC DS3231

wird der DS3231 über I²C (Maxim Integrated 2015, S. 1). Die meisten Breakout-Boards ermöglichen es außerdem, einen LIR2032 Lithiumakku anzuschließen, damit die Uhr auch im Falle eines Stromversorgungsausfalls weiter laufen kann.

3.4 Display SH1106

Bei der Planung der Station kam die Idee auf, dass ein kleines Display, welches die aktuellen Messwerte anzeigt, einen deutlichen Mehrwert bieten könnte. So müsste für eine kurze Auskunft nicht jedes Mal die Datenbank geöffnet oder eine Website aufgerufen werden. Außerdem könnten über das Display auch aussagekräftigere Fehlermeldungen ausgegeben werden.

Um die Steuerung des Displays einfach zu gestalten, wurde eines mit I²C-Schnittstelle gesucht. Recht verbreitet scheint ein OLED³-Display mit 1.3 Zoll Displaydiagonale zu sein, welches auf dem SH1106-Displaytreiber aufbaut. Dieser Chip kann Displays mit bis zu 132 x 64 Pixeln ansteuern (Sino Wealth 2013, S. 1). Damit können gut lesbar vier bis fünf Zeilen Text dargestellt werden.

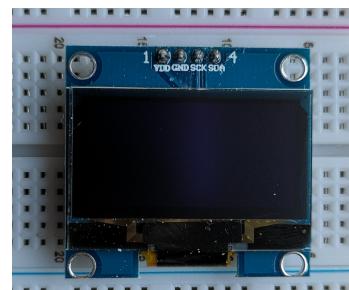


Abb. 3.4: 1.3“ Display

3.5 Sonstige Bauteile

3.5.1 RGB-LED

Um schnell den aktuellen Status einer Station erkennen zu können, sollte noch eine RGB-LED eingebaut werden. Über verschiedene Farben und dauerhaftes oder

³Organic Light Emitting Diode

blinkendes Licht können verschiedene Zustände der Station signalisiert werden. Dafür wurde im ersten Prototyp eine 5 mm Common-Cathode RGB-LED genutzt, später aber auf die bekannte SMD 5050 RGB-LED umgestiegen.

3.5.2 Reset-Button

Damit die Station einfach wieder in den Ausgangszustand versetzt werden kann, wurde noch ein Reset-Button eingeplant. Dabei sollte es sich um einen einfachen Push-Button handeln, welcher ein Signal durchlässt, sobald er gedrückt wird.

3.6 Kosten der Module und Bauteile

Bauteil	Menge	Kosten Deutschland	Kosten China
NodeMCU v2	1	6.50 €	2.50 €
BME280	1	7.00 €	2.50 €
DS3231 RTC	1	5.00 €	1.00 €
1.3“ OLED-Display	1	9.00 €	3.00 €
Pushbutton	1	3.50 € (100 St.)	1.20 € (100 St.)
SMD 5050 RGB-LED	1	16.00 € (100 St.)	2.10 € (100 St.)
Widerstand 1/4 W 10 Ω	2	3.50 € (100 St.)	1.20 € (100 St.)
Widerstand 1/4 W 68 Ω	1	3.50 € (100 St.)	1.20 € (100 St.)
Widerstand 1/4 W 10 kΩ	1	3.50 € (100 St.)	1.20 € (100 St.)
Platine (zweilagig)	1	27.72 € (3 St.)	11.00 € (10 St.)
Gesamtkosten pro Station		37.10 €	10.18 €

Tabelle 3.2: Menge und Kosten aller benötigten Bauteile

Aus der Tabelle 3.2 lassen sich die benötigten Materialien für eine Station ablesen. Die Spalte “Kosten Deutschland“ zeigt die Preise des jeweiligen Bauteils inklusive Versand innerhalb Deutschlands mit einer Lieferzeit von wenigen Tagen. Dabei wurden Preise auf Amazon Deutschland und Ebay Deutschland verglichen. Die Spalte “Kosten China“ zeigt den günstigsten Preis inklusive Versand auf der Plattform Aliexpress.com. Hierbei müssen längere Lieferzeiten von mehreren Wochen in Kauf genommen werden, es besteht aber gerade beim Bau von mehreren Stationen enormes Sparpotential.

Der Platinenfertigungspreis für Deutschland stammt von der Firma

Aisler (<https://aisler.net>), der Preis für eine Fertigung in China stammt von der Firma JLCPCB (<https://www.jlcpcb.com>). Aisler bietet kostenlose Versand und eine Versandzeit von einem Werktag nach Deutschland. JLCPCB bietet deutlich günstigere Fertigungskosten, nur \$2 für 10 Platinen, allerdings kostet der Standardversand \$9.39 und dauert 15-20 Werkstage. Expressversand kostet \$22.41 und dauert 3-5 Werkstage nach Deutschland. Kann man also wenige Tage länger warten, bietet JLCPCB das deutlich bessere Angebot.

3.7 Layout und Anschluss der Module

Mithilfe des Open-Source Programms Fritzing konnte das Layout und die Verkabelung der Station einfach dargestellt werden (siehe Bild 3.5 und 3.6). Dieses Programm beinhaltet bereits eine Vielzahl von Bauteilen, Boards und Modulen. Viele weitere Bauteile wurden von der Community oder Firmen erstellt und können z.B. auf Github heruntergeladen und dann in das Programm importiert werden. Da kein exakt gleich aussehendes Bauteil für den BME280 gefunden wurde, wurde in der Zeichnung das Breakout-Board von Sparkfun verwendet. Ebenso war es beim 1.3“ OLED-Display. Hier wurde einfach ein 0.96“ OLED-Display in der Zeichnung verwendet, da es anschlusstechnisch keinen Unterschied macht.

Funktion	GPIO Nummer	Beschriftung
I ² C SCL	5	D1
I ² C SDA	4	D2
RGB-LED Rot	0	D3
RGB-LED Grün	2	D4
Button	14	D5
RGB-LED Blau	15	D8

Tabelle 3.3: Pinbelegung am NodeMCU

Aufgrund unterschiedlicher GPIO Nummern und Pin-Beschriftungen auf dem NodeMCU Board wurden diese in Tabelle 3.3 noch einmal übersichtlich mit ihrer Funktion aufgelistet.

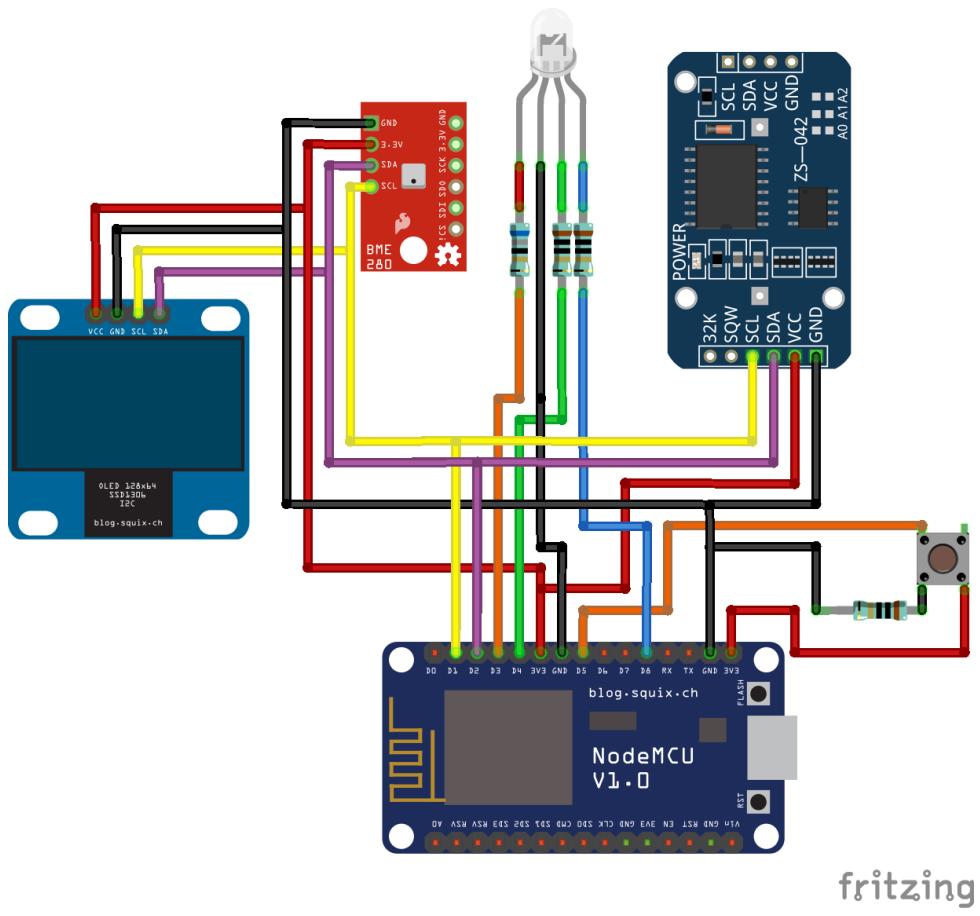


Abb. 3.5: Verkabelung der Module

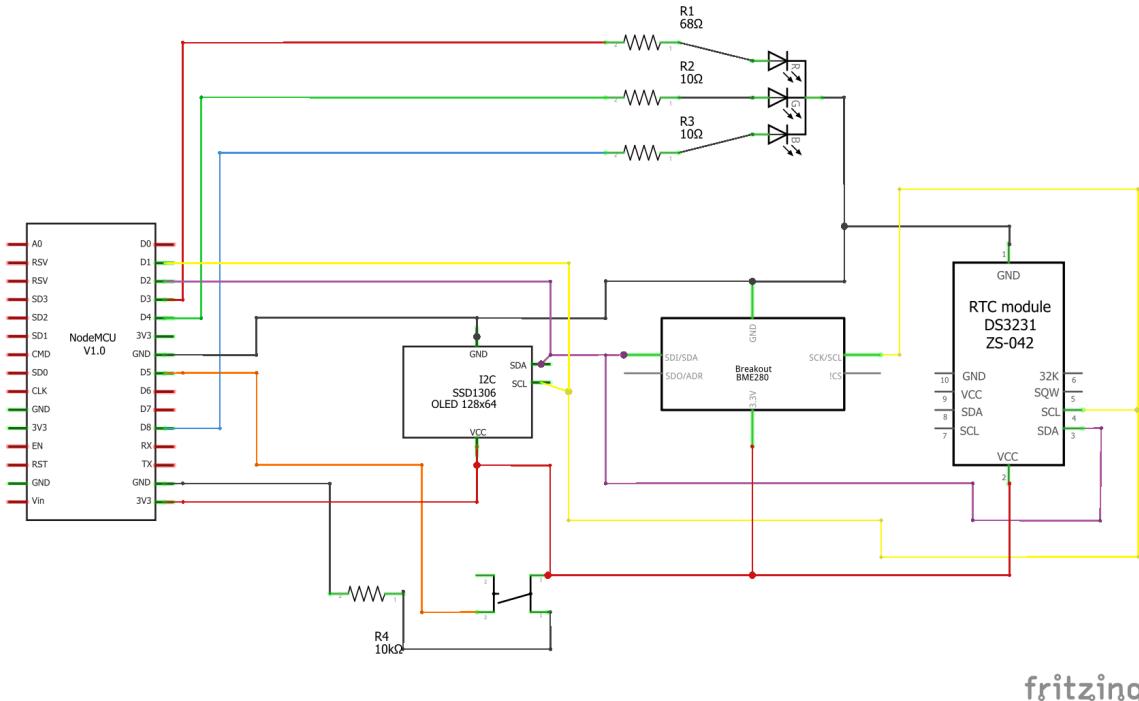


Abb. 3.6: Schaltplan

4 Genauigkeit des BMP180, BME280 und SHT31-D

In diesem Kapitel soll die Genauigkeit der Sensoren BMP180, BME280 und SHT31-D empirisch untersucht werden. Die Hersteller haben zwar Genauigkeitsangaben gemacht (siehe Tabelle 3.1), aber werden diese auch in der Praxis eingehalten?

Um die tatsächlichen Abweichungen zu ermitteln, wurde eine Vergleichsmessung über 24 Stunden mit mehreren BMP180, BME280, SHT31-D und einem Referenzfühler durchgeführt. Als Referenz diente ein Testo 645 (Wert: 560 €) mit einem hochpräzisen Temperatur- und Feuchtefühler (Wert: 535 €). Diesem Fühler lag zum Zeitpunkt der Messung ein aktuelles, vor zwei Monaten ausgestelltes Kalibrierzertifikat bei. Der Hersteller Testo gibt für diesen Fühlertyp eine Genauigkeit bei der Feuchtemessung von $\pm 1\%RH$ im Bereich 10 %RH bis 90 %RH bei 15 °C bis 30 °C an. Die Genauigkeit der Temperaturmessung wird mit $\pm 0.4\text{ }^{\circ}\text{C}$ im Bereich von 0 °C bis 50 °C angegeben (Testo AG n.d.). In Bereichen außerhalb dieser Grenzen ist die Feuchte- und Temperaturmessung etwas ungenauer, allerdings wurden diese während der Vergleichsmessung nicht erreicht.

Die Sensoren BMP180 sowie BME280 enthalten vom Hersteller bestimmte Kalibrierwerte, die auf dem Sensor gespeichert sind. Die verwendeten Arduino-Bibliotheken (siehe Tabelle 5.1) lesen diese Kalibrierwerte aus und beziehen sie in die Berechnung der Messwerte mit ein.

4.1 Genauigkeit der Temperaturmessung

Nach der Auswertung der Daten zeigt sich, dass wie zu erwarten der SHT31-D sehr genaue Messwerte erzeugt.

Aus Tabelle 4.1 lässt sich ablesen, dass die Standardabweichung der Messabweichung zum Referenzfühler bei allen getesteten Sensoren in einem ähnlichen Bereich liegen. Die Messung von Temperaturunterschieden erfolgt also mit ähnlicher Genauigkeit,

Sensor	mittlere Abw. zum Referenzfühler	Standardabw. der Abweichungen
BMP180	0.24 °C	0.10 °C
BME280 1	0.78 °C	0.10 °C
BME280 2	0.71 °C	0.09 °C
BME280 kombi	0.75 °C	0.09 °C
SHT31-D 1	0.03 °C	0.08 °C
SHT31-D 2	0.09 °C	0.07 °C
SHT31-D kombi	0.06 °C	0.07 °C

Tabelle 4.1: berechnete Genauigkeiten der Temperaturmessung

lediglich die Einordnung der gemessenen Werte auf einer absoluten Skala erfolgt mit unterschiedlicher Genauigkeit. Während beide SHT31-D die Werte absolut sehr gut einordnen, sind die des BMP180 und die beider BME280 etwas verschoben. Eventuell kalibriert Sensirion seine Sensoren ab Werk genauer als Bosch es tut. Allerdings zeigt das Ergebnis auch, dass alle getesteten Sensoren die Herstellerangaben erfüllen.

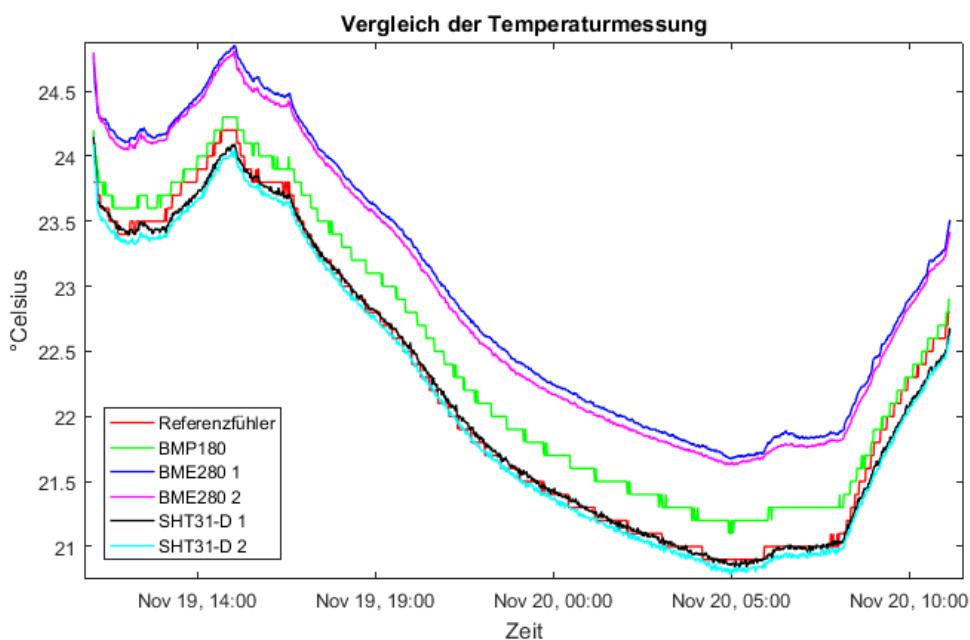


Abb. 4.1: Vergleich der Temperaturmessung

Die Abbildungen 4.1 und 4.2 zeigt die Messreihen noch einmal graphisch. Hier sieht man, wie gut die Werte der SHT31-D Sensoren auf denen des Referenzfühlers liegen. Auch lässt sich gut der Offset der Messreihen des BME280 erkennen. Bestimmt man

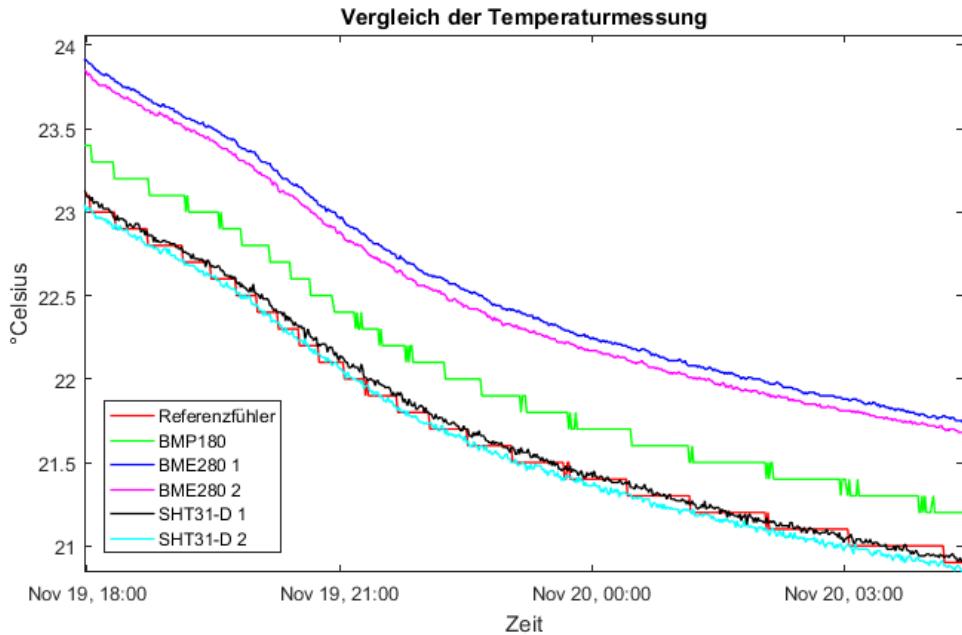


Abb. 4.2: Vergleich der Temperaturmessung (Ausschnitt)

diesen Offset durch eine Referenzmessung wie diese, könnte auch ein BME280 unter Berücksichtigung dieses Wertes präzisere Absolutwerte liefern.

4.2 Genauigkeit der Feuchtemessung

Während bei der Temperaturmessung klar der SHT31-D die genaueren Messwerte lieferte, zeigt die Referenzmessung der Feuchte, dass hier der BME280 geringere Abweichungen aufweist.

Sensor	mittlere Abw. zum Referenzfühler	Standardabw. der Abweichungen
BME280 1	0.75 %rF	0.21 %rF
BME280 2	1.77 %rF	0.19 %rF
BME280 kombi	1.26 %rF	0.20 %rF
SHT31-D 1	2.31 %rF	0.26 %rF
SHT31-D 2	2.65 %rF	0.27 %rF
SHT31-D kombi	2.48 %rF	0.27 %rF

Tabelle 4.2: berechnete Genauigkeiten der Feuchtemessung

Tabelle 4.2 zeigt wieder die mittlere Abweichung der Messwerte zum Referenzfühler sowie die Standardabweichung dieser Messabweichungen. Der BME280 unterschreitet

die Herstellerangabe von $\pm 3\%rF$ in beiden Fällen deutlich, während der SHT31-D die Herstellerangabe von $\pm 2\%rF$ in beiden Fällen überschreitet.

Allerdings zeigen die Standardabweichungen wieder, dass beide Sensoren Feuchteänderungen ähnlich genau messen, nur die absolute Einordnung dieser Änderungen erfolgt unterschiedlich genau. Die Messwerte sind also nur wieder um einen relativ konstanten Offset versetzt. Dies lässt sich auch gut in Abbildung 4.3 erkennen.

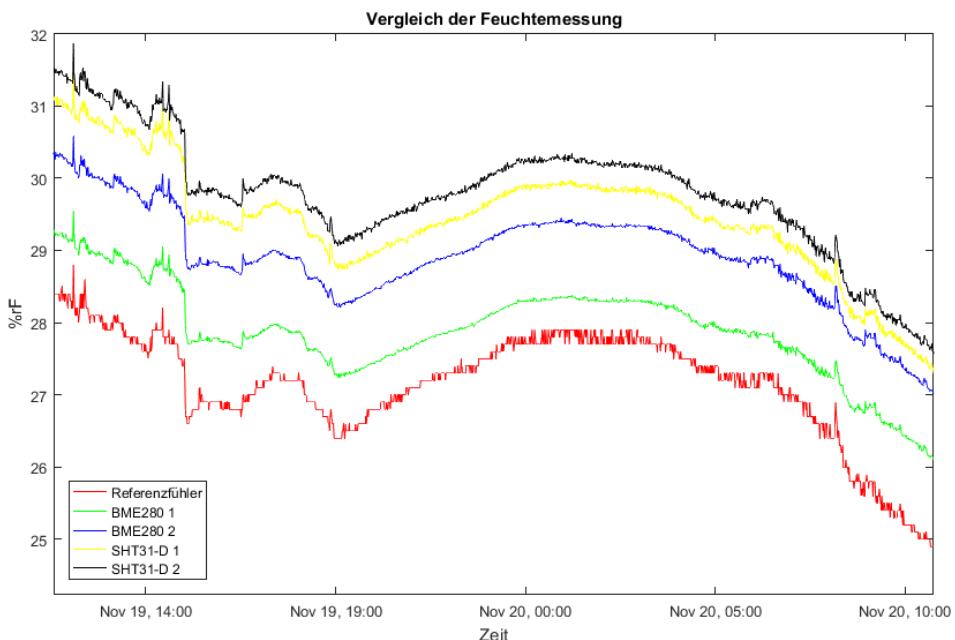


Abb. 4.3: Vergleich der Feuchtemessung

4.3 Genauigkeit der Luftdruckmessung

Für die Vergleichsmessung des Luftdrucks konnte das Referenzgerät von Testo nicht verwendet werden, da dieses nur Feuchte und Temperatur erfasst. Daher wurde ein Model 740 der Firma setra als Referenz verwendet. Der Hersteller gibt für dieses Gerät eine Messgenauigkeit von $\pm 0.02\%FS$ bei 21°C an (setra n.d., S. 34). Berücksichtigt man, dass die Skala des Model 470 bis 1100 hPa geht, ergibt das eine mögliche Messungenauigkeit von $\pm 0.22\text{ hPa}$. Auch diese Messung wurde wieder über 24 Stunden durchgeführt. Dabei wurde darauf geachtet, dass die Temperatur um die 21°C beträgt, damit das Model 470 möglichst genau arbeitet.

Die Tabelle 4.3 zeigt, dass auch bei der Luftdruckmessung alle Sensoren Ände-

rungen des Luftdruckes genau bestimmen können, die Standardabweichungen der Unterschiede zur Referenz liegen alle unter 0.1 hPa. Nur die Einordnung der Werte auf der Hektopascal-Skala erfolgt wieder mit unterschiedlich großen Abweichungen. Aber auch hier erfüllen alle Sensoren die vom Hersteller angegebene Genauigkeit von ± 1 hPa.

Sensor	mittlere Abw. zum Model 470	Standardabw. der Abweichungen
BMP180 1	0.79 hPa	0.06 hPa
BMP180 2	0.71 hPa	0.06 hPa
BMP180 kombi	0.75 hPa	0.05 hPa
BME280 1	0.56 hPa	0.03 hPa
BME280 2	0.87 hPa	0.07 hPa
BME280 3	0.01 hPa	0.04 hPa
BME280 kombi	0.47 hPa	0.04 hPa

Tabelle 4.3: berechnete Genauigkeiten der Luftdruckmessung

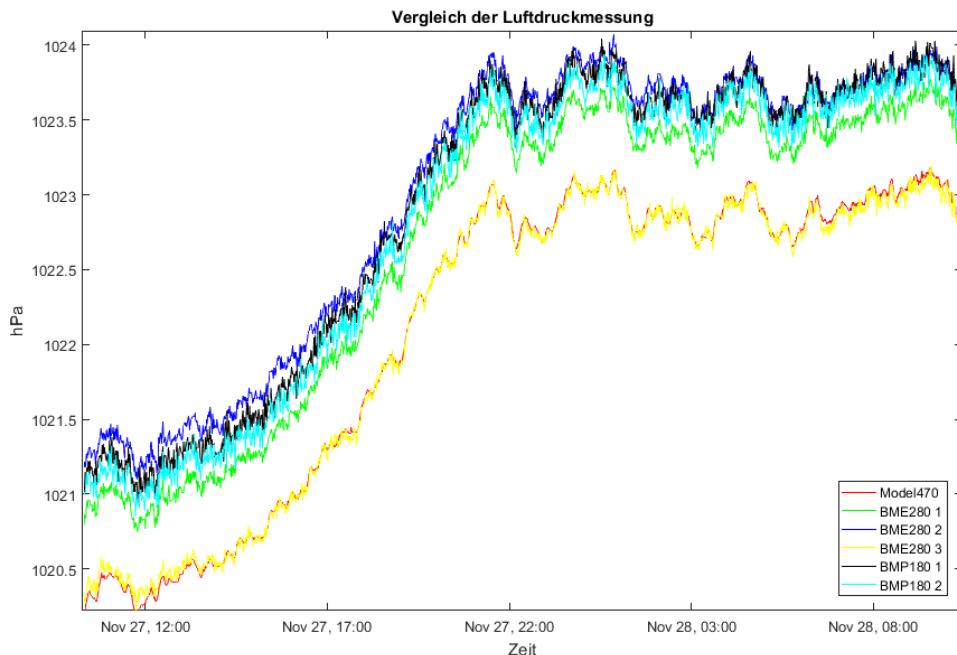


Abb. 4.4: Vergleich der Luftdruckmessung

4.4 Einsatz mehrerer Sensoren pro Station

Die Ergebnisse der Referenzmessungen können dabei helfen, den richtigen Sensor für den gewünschten Zweck auszuwählen: Legt man mehr Wert auf eine genaue Temperaturmessung und kann auf eine Luftdruckbestimmung verzichten, sollte man den SHT31-D verwenden. Möchte man den Luftdruck auch bestimmen und kann sich mit der etwas ungenauerer Temperaturmessung abfinden, sollte der BME280 gewählt werden.

Allerdings existiert auch eine dritte Option: der Einsatz von jeweils einem SHT31-D und einem BME280 pro Station. So könnte man die gute Temperaturmessung des SHT31-D mit der Feuchte- und Luftdruckmessung des BME280 kombinieren.

Zusätzlich könnte man die Vertrauenswürdigkeit der Messwerte noch weiter erhöhen, indem man mehrere Sensoren gleichen Typs auf einer Station einsetzt. So könnten die Abweichungen eines Sensors durch weitere Messwerte anderer Sensoren ausgeglichen werden. Auch wird durch mehrere Sensoren eine größere Ausfallsicherheit erreicht, was besonders bei Langzeitmessungen von Interesse wäre. Hier tritt dann allerdings das Problem auf, dass gleiche Sensoren meist auch gleiche I²C-Adressen haben. Um dieses Problem zu beheben, muss zusätzlich ein I²C-Multiplexer wie zum Beispiel der TCA9548A von Texas Instruments verwendet werden. Dieser Chip ermöglicht es, bis zu 8 Geräte mit gleicher I²C-Adresse nacheinander anzusprechen. Dazu werden alle SCL- und SDA-Leitungen der Geräte mit dem Multiplexer verbunden. Der Multiplexer selber wird per I²C an den Microcontroller angeschlossen und über Steuerbefehle können dann die angeschlossenen Geräte durchgeschaltet werden. Den genannten Multiplexer von Texas Instruments gibt es für etwa 10 € in Deutschland zu kaufen, bestellt man in China bezahlt man etwa 3 €.

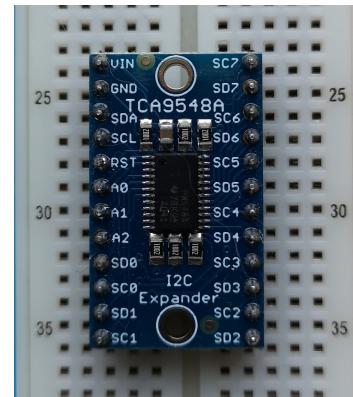


Abb. 4.5: TCA9548A

5 Programmierung der Station

In diesem Kapitel soll die Programmierung der Sensorstation beschrieben werden. Es wird gezeigt, wie die Arduino-IDE vorbereitet wurde, welche Bibliotheken verwendet wurden und welche Überlegungen bei der Programmierung eine Rolle gespielt haben.

5.1 Vorbereitung der IDE

Für die Programmierung dieses Projekts wurde die Arduino-IDE in der Version 1.8.7 eingesetzt. Da die Arduino-IDE standardmäßig das NodeMCU Board nicht unterstützt, muss die Konfiguration nachgeladen werden. Dies funktioniert unkompliziert, indem unter Datei -> Voreinstellungen beim Punkt

Zusätzliche Boardverwalter-URLs die folgende URL eingefügt wird:

http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Im Boardverwalter kann nun das esp8266-Paket mit vielen verschiedenen, auf dem ESP8266-basierenden Boards heruntergeladen werden. Unter dem Menüpunkt Werkzeuge kann dann das Board NodeMCU 1.0 (ESP-12E Module) ausgewählt werden. Beim Unterpunkt Flash Size wurde von mir die Option 4M (3M SPIFFS¹) ausgewählt, so steht maximaler Speicher für Messwerte im Falle eines Verbindungsabfalls zur Verfügung. Unter Erase Flash kann noch ausgewählt werden, ob bei jedem Flashvorgang nur der Speicher mit dem Programm oder auch die eventuell schon gespeicherten WLAN-Einstellungen gelöscht werden sollen.

5.2 Verwendete Bibliotheken

Die Arduino-IDE kommt mit einigen Standardbibliotheken, die häufig benötigte Anwendungsfälle abdecken. Aus diesen wurde die SPIFFS-Bibliothek (FS.h) für die Nutzung des SPIFFS-Speichers sowie die Wire-Bibliothek (Wire.h) für die I²C Kommunikation genutzt. Ebenso werden mit der Installation des NodeMCU-Boards einige ESP-8266-Bibliotheken installiert. Aus diesen wurden die Bibliotheken `ESP8266WiFi.h`, `ESP8266WebServer.h`, `DSNServer.h` und `WifiClientSecure.h` verwendet. Diese Bibliotheken ermöglichen die Bereitstellung eines Webservers inklusive DNS-Server für das Konfigurationsportal

¹Serial Peripheral Interface Flash File System

sowie die Kommunikation über verschlüsselte HTTP²-Anfragen mit einem Webserver. Um die verwendeten Hardwaremodule (Sensor, Uhr, Display) komfortabel anzusprechen und um ein Konfigurationsportal via WiFi anzubieten, wurden einige zusätzliche Bibliotheken verwendet. Diese sind in der Tabelle 5.1 mit Quelle aufgeführt.

Name	Quelle
Adafruit Unified Sensor	github.com/adafruit/Adafruit_Sensor
Adafruit BME280 Library	github.com/adafruit/Adafruit_BME280_Library
RTClib	github.com/adafruit/RTClib
U8g2	github.com/olikraus/u8g2
WiFiManager	github.com/tzapu/WiFiManager
ArduinoJson	github.com/bblanchon/ArduinoJson

Tabelle 5.1: Verwendete Bibliotheken

Die zusätzlichen Bibliotheken können bequem innerhalb der IDE installiert werden. Dafür muss im Menü Sketch der Unterpunkt Bibliothek einbinden ausgewählt werden. Klickt man nun auf den Eintrag Bibliothek verwalten öffnet sich das Bibliotheksverwalter-Fenster. Über die Suche können die benötigten Bibliotheken nun gesucht und installiert werden.

5.3 Erklärungen zum Code

Ein Arduino-Programm besteht im wesentlichen aus zwei Funktionen: `setup()` und `loop()`. Code im `setup()` wird nur einmalig beim Start des Programms ausgeführt, während Code im `loop()` immer wieder in einer Schleife ausgeführt wird. Bevor jedoch diese Funktionen genutzt werden, wurden die im vorherigen Kapitel aufgeführten Bibliotheken importiert, sowie globale Variablen für Sensor, RTC, Display, Logdatei, LED, Button und einige Statussituationen festgelegt.

5.3.1 `setup()`

Zu Beginn werden die Pins für die LED als `OUTPUT` definiert, damit diese Spannung abgeben können. Der Pin für den Button wird als `INPUT` definiert, damit er ein eingehendes Spannungssignal erkennen kann. Die serielle Kommunikation wird mit einer Geschwindigkeit von 9600 Baud (Baud ist ein Maß für die pro Sekunde übertragenen Bits (Margolis 2012, S. 96)) initialisiert und die I²C-Kommunikation wird auf den beiden dafür festgelegten Pins gestartet.

²Hypertext Transfer Protocol

```
void setup() {  
    pinMode(REDPIN, OUTPUT);  
    pinMode(GREENPIN, OUTPUT);  
    pinMode(BLUEPIN, OUTPUT);  
    pinMode(buttonPin, INPUT);  
  
    Serial.begin(9600);  
    Wire.begin(SDAPIN, SCLPIN);  
    delay(1000);  
    ...  
}
```

Dann werden mit der Wire-Bibliothek die I²C-Adressen des DS3231, des Displays und des BME280 angesprochen. An jede Adresse wird ein Stop-Befehl gesendet. Antwortet das Gerät mit dem Statuscode 0 war der Befehl erfolgreich (Arduino AG 2018b). Das Gerät ist also vorhanden und kann initialisiert werden. Wurde das Gerät nicht gefunden, bleibt die entsprechende Statusvariable weiterhin `false`.

```
...  
Wire.beginTransmission(60); // entspricht 0x3C (Display Adresse)  
byte errorDisplay = Wire.endTransmission();  
if (errorDisplay == 0) {  
    Serial.println("I2C Display gefunden");  
    displayConnected = true;  
}  
if (displayConnected) {  
    u8g2.begin();  
    u8g2.enableUTF8Print();  
}  
...  
}
```

Nach der Detektion der angeschlossenen Module wird das SPIFFS geöffnet und nach einer bereits bestehenden Konfigurationsdatei durchsucht. In dieser Konfigurationsdatei steht der Stationstoken, welcher die Station am Server identifiziert. Wird also diese Datei gefunden, wird der Inhalt in die Variable `web_token` geschrieben und die Variable `showWifiPortal` wird auf `false` gesetzt. So wird direkt die gespeicherte WLAN-Verbindung wiederhergestellt und die Anzeige des Konfigurationsportals übersprungen.

Wurde keine Konfigurationsdatei gefunden, wird das Konfigurationsportal in Form eines Wifi-Hotspots gestartet und auf Eingabe der Verbindungsdaten gewartet. Hinweise zu

Einrichtung einer Station finden sich im Kapitel 5.4.

Am Ende des Setups wird noch überprüft, ob neue Konfigurationsdaten über das Konfigurationsportal eingegeben wurden. Ist die entsprechende Variable `shouldSaveConfig == true`, wird die Konfiguration in ein Datei im SPIFFS geschrieben.

5.3.2 loop()

Ein Ziel bei der Programmierung der loop-Funktion war, ohne `delay()` zu arbeiten. So lässt es sich vermeiden, dass das Programm bestimmte Zeiten nicht auf Aktionen, wie zum Beispiel einen gedrückten Button reagieren kann. Auch kann so die LED in bestimmten Abständen ein- und ausgeschaltet, also zum Blinken gebracht werden.

Die loop-Funktion beginnt mit einer Überprüfung, ob der Reset-Button gerade gedrückt wird. Falls dem so ist, wird die Konfigurationsdatei gelöscht und der Chip zurückgesetzt. Eine erneute Einrichtung der Station kann nun erfolgen.

```
void loop() {
    if (buttonConnected) {
        int buttonState = 0;
        buttonState = digitalRead(buttonPin);
        if (buttonState == HIGH) {
            Serial.println("button pressed");
            delay(200);
            SPIFFS.remove("/config.json");
            delay(200);
            wifiManager.resetSettings();
            delay(1000);
            ESP.restart();
        }
    }
    ...
}
```

Danach wird die Funktion `ledBlinker()` aufgerufen. Diese nutzt die Funktion `millis()`, welche die Anzahl Millisekunden, die der Sketch schon läuft, zurückgibt (Margolis 2012, S. 399). So kann, falls der Blinkmodus aktiviert wurde, die LED alle 400 ms ein- oder ausgeschaltet werden. Ebenso wird hier die Farbe der LED über die globalen LED-Farb-Variablen gesetzt.

```
void ledBlinker() {
```

```
if (LEDBLINK) {  
    const unsigned long onZeit = 400; // ms  
    const unsigned long offZeit = 400; // ms  
    if ((millis() % (onZeit + offZeit)) < onZeit) {  
        analogWrite(REDPIN, GREENCOLOR);  
        analogWrite(GREENPIN, REDCOLOR);  
        analogWrite(BLUEPIN, BLUECOLOR);  
    } else {  
        analogWrite(REDPIN, 0);  
        analogWrite(GREENPIN, 0);  
        analogWrite(BLUEPIN, 0);  
    }  
} else {  
    analogWrite(REDPIN, GREENCOLOR);  
    analogWrite(GREENPIN, REDCOLOR);  
    analogWrite(BLUEPIN, BLUECOLOR);  
}  
}
```

Darauf folgend wird die Funktion `processStarter()` aufgerufen. Diese liefert einen boolean-Wert zurück, je nachdem ob eine Messung gestartet werden soll oder nicht. Falls eine RTC angeschlossen ist, wird diese für die Zeitmessung des Messintervalls verwendet, ansonsten wird die `millis()`-Funktion genutzt. Liegt die letzte Messung schon länger als das Messintervall zurück, wird ein neuer Messprozess gestartet.

```
bool processStarter() {  
    if (rtcConnected) {  
        DateTime processNow = rtc.now();  
        if ((processNow.unixtime() - lastProcess) > (intervall * 60)  
            ) {  
            lastProcess = processNow.unixtime();  
            return true;  
        } else {  
            return false;  
        }  
    } else {  
        unsigned long processNow = millis();  
        if ((processNow - lastProcess) > (intervall * 60 * 1000)) {  
            lastProcess = processNow;  
            return true;  
        }  
    }  
}
```

```
    } else {
        return false;
    }
}
```

Falls nun ein neuer Messprozess gestartet wurde, werden als nächstes die Messwerte des Sensors ausgelesen und in den entsprechenden Variablen gespeichert. Dann wird der Daten-String, der per POST-Request an den Server gesendet wird, generiert und mit der Funktion `sendRequest()` abgeschickt. Die Antwort des Servers wird in der Variable `responseObject` gespeichert. Die Antwort ist ein JSON³-Codierter String, der als JSON-Objekt in der Variable `body` gespeichert wird.

Die Variable `body` wird nun durchsucht und der Request-Status, der Stationsname und das Messintervall werden in den entsprechenden Variablen gespeichert beziehungsweise aktualisiert. Ebenfalls liefert der Server mit jeder Antwort die aktuelle Zeit als UNIX-Zeitstempel. Mithilfe dieses Zeitstempels wird die RTC aktualisiert.

Der Request-Status wird nun noch etwas genauer untersucht. Mögliche Statuscodes sind in Tabelle 5.2 mit ihrer Bedeutung aufgeführt. Je nach Fehlercode wird jetzt ein Text auf dem Display ausgegeben und die Farbe der LED angepasst.

Statustext	Bedeutung	LED	Display
success	Sensordaten erfolgreich zum Server übertragen	grün	Uhrzeit des Requests
error_mysql	Fehler beim Speichern der Daten in der Datenbank	blinkt blau	SQL Fehler
error_token	Stationstoken nicht in Datenbank vorhanden	-	-
error_client	Keine WLAN Verbindung vorhanden	blinkt gelb	Wifi Fehler

Tabelle 5.2: Fehlercodes und LED Farben

Falls der Fehler vom Typ `error_sql` oder `error_client` ist, muss davon ausgegangen werden, dass die Messwerte nicht auf dem Server gespeichert werden konnten. Daher werden diese in diesem Fall auf dem SPIFFS mit der Funktion `writeLog()` in einer Datei zwischengespeichert. Allerdings läuft dieser Prozess nur ab, wenn eine RTC angeschlossen ist und das SPIFFS noch genug freien Speicher aufweist.

³.JavaScript Object Notation

Ist der empfangene Statuscode vom Typ `success`, war das Speichern der Messwerte auf dem Server erfolgreich. Es kann also davon ausgegangen werden, dass der Server ordnungsgemäß arbeitet und eine WLAN-Verbindung besteht. Daher wird im Anschluss mit der Funktion `sendLog()` überprüft, ob im SPIFFS Messwerte zwischengespeichert wurden, die zuvor nicht abgeschickt werden konnten. Ist dies der Fall, werden diese Messwerte nun erneut an den Server übermittelt. Tritt dabei wieder ein Fehler auf, werden die Werte erneut in eine Datei geschrieben und es wird später ein erneuter Versuch unternommen. Nur erfolgreich an den Server übermittelte Werte werden aus dem Speicher des NodeMCU entfernt.

5.4 Einrichtung einer Station

Um eine Station einzurichten, muss in der Datenbank ein valider Stationstoken für die Station hinterlegt und ein WLAN-Netzwerk in Reichweite sein.

Nun wird die Station mit dem Netzkabel über die Micro-USB-Buchse des NodeMCU an eine Steckdose angeschlossen. Der Microchip bootet nun automatisch.

Startet der Chip das erste Mal, benötigt er die WLAN-Daten und den Stationstoken. Diese Daten können über das Konfigurationsportal eingegeben werden. Dieses kann aufgerufen werden, indem ein Gerät mit Browser (z.B. Smartphone, Tablet, Laptop) mit dem WLAN-Netzwerk, welches der Chip ausstrahlt, verbunden wird. Der Name des WLAN-Netzwerks beginnt mit den Buchstaben `ESP`, worauf die ID des Chips als Nummer folgt. Es ist kein Passwort für die Verbindung nötig. Öffnet man nun den Browser auf dem verbundenen Gerät, sollten alle Webanfragen auf das Konfigurationsportal weitergeleitet werden. Alternativ kann auch direkt die IP-Adresse `192.168.4.1` aufgerufen werden.

Im Konfigurationsportal klickt man nun auf den Button "Configure WiFi". Der Microchip scannt nun einmal alle WLAN-Netzwerke in Reichweite und zeigt diese mit der jeweiligen Signalstärke an. Durch einen Klick auf das korrekte WLAN-Netzwerk wird dieses ausgewählt und es muss noch das dazugehörige Passwort eingetragen werden. In das Token-Feld wird der entsprechende Stationstoken eingetragen. Mit einem Klick auf den Button "Save" werden die eingegebenen Daten an den Chip übertragen und gespeichert. Die Station ist nun einsatzbereit.

6 Programmierung des Servers

In diesem Kapitel soll es um die Zentrale des Sensornetzwerkes gehen: Ein Computer nimmt die von den einzelnen Stationen gesendeten Daten entgegen und speichert sie in einer Datenbank. Für diese Aufgabe wird ein mit dem Internet verbundener Server verwendet, auf dem ein Apache Webserver in Verbindung mit PHP und MySQL läuft.

6.1 PHP und MySQL

PHP ist eine serverseitige Programmiersprache, die ursprünglich von Rasmus Lerdorf im Jahre 1995 für die Generierung von HTML-Inhalten entworfen wurde (Lerdorf & Tatroe 2002, S. 1) Allerdings lässt sie sich aufgrund ihrer Einfachheit auch gut in diesem Projekt für den Zweck der Weiterverarbeitung der gesendeten Wetterdaten verwenden. Heutzutage zählt PHP immer noch zu den am häufigsten genutzten Web Backend Sprachen (stackoverflow n.d.).

PHP unterstützt eine ganze Reihe von Datenbankmanagementsystemen, darunter natürlich auch die populäre Open Source Datenbank MySQL. MySQL ist eine relationale Datenbank, das bedeutet, die Daten werden in einer oder mehreren Tabellen gespeichert (Yank 2005, S. 89-90). Für die Abfrage von Daten wird SQL (Structured Query Language) genutzt.

6.2 Struktur der Datenbank

Für die Speicherung der gemessenen Daten wurde eine Datenbank mit zwei Tabellen angelegt.

Die erste Tabelle `readings` speichert die Messwerte. Die zweite Tabelle `stations` speichert Informationen und Einstellungen der Stationen. Die Felder `street`, `city`, `lat` und `long` der Tabelle `stations` können in einem nächsten Schritt dazu genutzt werden, die Messwerte auf einer Karte darzustellen. Das Feld `intervall` speichert das Messintervall der jeweiligen Station in Minuten, das Feld `name` speichert einen kurzen Stationsnamen, der auch auf dem Display dargestellt wird.

Die Tabellen `stations` und `readings` sind über eine 1-n Primärschlüssel- Fremdschlüsselbeziehung miteinander verknüpft. Dies wird auch im Diagramm 6.1 deutlich.

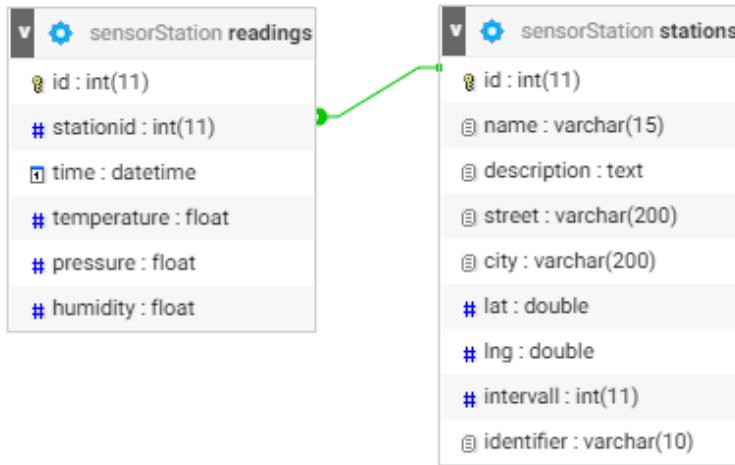


Abb. 6.1: Entity-Relationship-Diagramm

Das Feld `identifier` der Tabelle `stations` enthält einen zehnstelligen, eindeutigen, alphanumerischen Code, mit welchem sich eine Messstation am Server identifizieren kann. Die Messwerte werden dann der entsprechenden Station zugeordnet. Würde man für diesen Zweck einfach die `id` verwenden, könnte ein einfacher Tippfehler (zum Beispiel 64 statt 65) Messwerte einer anderen Station zuordnen. Bei einem zehnstelligen alphanumerischen Code gibt es weitaus mehr Kombinationen und Tippfehler sollten zu einem `error_token`-Fehler führen.

6.3 Erklärungen zum Code

Das vorliegende Serverprogramm wurde entwickelt und getestet mit PHP 7.0.32 und MySQL 5.5.60.

Die von der Station erfassten Daten werden als HTTP-Request an den Server gesendet. In diesem Fall wurde für den Server die Domain `www.weatherweb.de` registriert, so muss nicht immer die IP des Servers verwendet werden. Im Root-Verzeichnis des Webservers wurde der Ordner `api` erstellt, worin das PHP-Programm mit dem Dateinamen `send_data.php` liegt. Die Daten der Station müssen also an die Adresse `www.weatherweb.de/api/send_data.php` geschickt werden.

Zuerst wird dem Script gesagt, alle Zeitangaben beziehen sich auf die Zeitzone “Europa/Berlin“. Ebenso werden die Zugangsdaten für die Datenbank festgelegt.

```

date_default_timezone_set("Europe/Berlin");
$config = array (
    'db_host' => 'localhost',
    'db_name' => 'Datenbankname',
    'db_user' => 'Datenbankbenutzer',
    'db_passwd' => 'Benutzerpasswort',
);

```

Die Verbindung zur Datenbank wird in der Variable \$connect gespeichert.

Dann wird der erste “POST-Parameter“ überprüft. Mit diesem werden zwei Anwendungsfälle unterschieden: In Fall Eins (update) werden die in Echtzeit von der Station gesendeten Daten in die Datenbank gespeichert, in Fall Zwei (send_cache) werden Daten aus dem Zwischenspeicher der Station nach Wiederherstellung der Internetverbindung nun an den Server gesendet.

```

if(isset($_POST['action']) && $_POST['action'] == 'update') {
    ...
}
else if(isset($_POST['action']) &&
$_POST['action'] == 'send_cache') {
    ...
}

```

6.3.1 Echtzeitübertragung der Messwerte

Falls die Daten in Echtzeit gesendet wurden, als action also update angegeben wurde, werden die von der Station ebenfalls als “POST-Parameter“ gesendeten Messwerte überprüft und PHP-Variablen zugewiesen. Um SQL-Injections zu vermeiden, werden die gesendeten Daten mit den Funktionen mysqli_real_escape_string() und strip_tags() überprüft. Sonderzeichen oder SQL-Befehle werden so maskiert und können nicht ausgeführt werden. Ebenfalls wird die aktuelle Zeit als Zeitstempel für den gesendeten Datensatz festgelegt.

```

$now = date('Y-m-d H:i:s');
$temp = isset($_POST['temp']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['temp'])) : '';
$pres = isset($_POST['pres']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['pres'])) : '';
$humid = isset($_POST['humid']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['humid'])) : '';
$token = isset($_POST['token']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['token'])) : '';

```

Damit die Daten hinterher auch den einzelnen Stationen zugeordnet werden können, sendet jede Station bei jedem Request ihren einzigartigen, aus zehn Zeichen bestehenden Identifizierungscode (`token`). Im nächsten Schritt wird überprüft, ob der von der Station gesendete Code korrekt ist und in der Tabelle `stations` vorhanden ist. Wird der Code nicht gefunden, wird der Statuscode `error_token` gesetzt.

```
$queryStation =
    "SELECT * FROM stations WHERE identifier = '$token' LIMIT 1";
$resultStation = mysqli_query($connect, $queryStation);
$stationdata = mysqli_fetch_array($resultStation);
$stations = [];
if (@mysqli_num_rows($resultStation) != 1) {
    $response['status'] = "error_token";
} else {
    // Messwerte in Datenbank schreiben
    ...
}
```

Falls der Stationscode korrekt ist, werden die übermittelten Messwerte in der Datenbank gespeichert. Wenn dies ohne Fehler passiert ist, wird der Statuscode `success` gesetzt, tritt ein Fehler beim Einfügen auf, wird der Statuscode `error_mysql` gesetzt. Ebenfalls wird in der Variable `$response`, welche als Antwort an die Station übermittelt wird, der Stationsname, das für die Station festgelegte Messintervall und die aktuelle Zeit übermittelt.

```
...
else {
    // Messwerte in Datenbank schreiben
$queryInsert =
    "INSERT INTO readings
        (stationid, `time`, temperature, pressure, humidity)
    VALUES
        ((SELECT id FROM stations WHERE identifier='$token'),
        '$now', '$temp', '$pres', '$humi')");
if (mysqli_query($connect, $queryInsert)) {
    $response['status'] = "success";
} else {
    $response['status'] = "error_mysql";
}

$response['stationname'] = $stationdata['name'];
$response['intervall'] = $stationdata['intervall'];
```

```

$response['timestamp'] = time();
}

```

Zum Schluss wird die Antwortvariable \$response noch in das JSON-Format umgewandelt und dann ausgegeben.

```
echo json_encode($response);
```

6.3.2 Übertragung von zwischengespeicherten Werten

Werden nach Wiederherstellung der Internetverbindung einer Station die zwischengespeicherten Messwerte an den Server übertragen, beinhaltet der POST-Parameter action den Wert send_cache. Damit die leistungsschwächere Station die in Textform und mit Semikolon getrennt gespeicherten Messwerte nicht umständlich wieder trennen muss, werden die Daten zeilenweise im POST-Parameter line an den Server gesendet. Eine Zeile enthält nacheinander den Zeitstempel im Unixformat, die Temperatur, den Luftdruck und die Feuchtigkeit.

```
// Beispiel fuer eine Zeile
line=1542372674;24.56;1020.23;43.43
```

Das Script nimmt diese Daten nun entgegen und überprüft sie auf Sonderzeichen und weitere nicht gewünschte Ausdrücke. Daraufhin wird die gesendete Zeile an den Semikolons gebrochen und zu einem Array umgeformt. Die einzelnen Werte des Arrays werden dann den entsprechenden Variablen zugeordnet.

```

else if(isset($_POST['action']) &&
$_POST['action'] == 'send_cache') {
// Verarbeitung von zwischengespeicherten Werten der Station
// nach wiederhergestellter Internetverbindung

$response = [];

$line = isset($_POST['line']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['line'])) : '';
$linedata = explode(";", $line);

$timestamp = $linedata[0];
$temp = $linedata[1];
$pres = $linedata[2];
$humid = $linedata[3];
$token = isset($_POST['token']) ?
    mysqli_real_escape_string($connect, strip_tags($_POST['token'])) : '';
...
}
```

Nun wird wieder wie auch schon bei der Echtzeitübertragung überprüft, ob der gesendete Identifizierungscode der Station korrekt ist. Ist dies der Fall, werden die Daten in die Datenbank geschrieben. Enthält die Zeile fehlerhafte oder unvollständige Daten, wird der Statuscode `error_line` ausgegeben. Auch hier wird in der Antwortvariable `$response` wieder der Stationsname, das für die Station festgelegte Messintervall und die aktuelle Zeit übermittelt.

6.4 Anzeige der Daten

Um die Darstellung der Messdaten zu ermöglichen, wurde eine auf der Javascript Bibliothek amCharts (<https://www.amcharts.com>) basierende Beispielanwendung entwickelt. AmCharts kann frei genutzt werden, solange ein kleines Branding in den erzeugten Diagrammen angezeigt wird.

Die Beispielanwendung ermöglicht die Selektion von Daten nach Station, Datum und anzuzeigendem Zeitraum (Tag, Monat oder Jahr). Abgerufen werden die Daten per Javascript-HTTP-Request über die PHP-Datei `get_data.php`. Diese Datei ruft die entsprechenden Daten aus der Datenbank ab und verkleinert die zu sendende Datenmenge gegebenenfalls durch Mittelbildung. Bei der Anzeige eines Monats werden jeweils die Messwerte einer ganzen Stunde gemittelt, bei der Auswahl eines Jahrs werden die Messwerte für jeden Tag des Jahres gemittelt. Bei der Anzeige eines Tages findet keine Mittelbildung statt. Die Messdaten werden auch ohne Benutzeraktion alle 60 Sekunden erneut abgerufen und aktualisiert. So zeigt das Diagramm immer den aktuellsten Stand.

Ebenfalls wird in die Messdaten der letzten Stunde eine ausgleichende Gerade gelegt, sodass eine Trendabschätzung erfolgen kann. Dieser Trend wird durch Pfeile (steigend oder fallend) für jeden Parameter dargestellt. Das Bild 6.2 zeigt einen Screenshot Anwendung.

Diese Beispielanwendung kann natürlich noch erweitert und angepasst werden. Zum Beispiel könnte eine Authentifizierung eingebaut werden, sodass ein Nutzer nur Zugriff auf eigene Messstationen hat. Auch könnte eine Karte erstellt werden, welche die aktuellen Messwerte georeferenziert darstellt.

Der Quellcode dieser Beispielanwendung kann auch im Github-Repository (siehe Kapitel 7.2.1) gefunden werden.

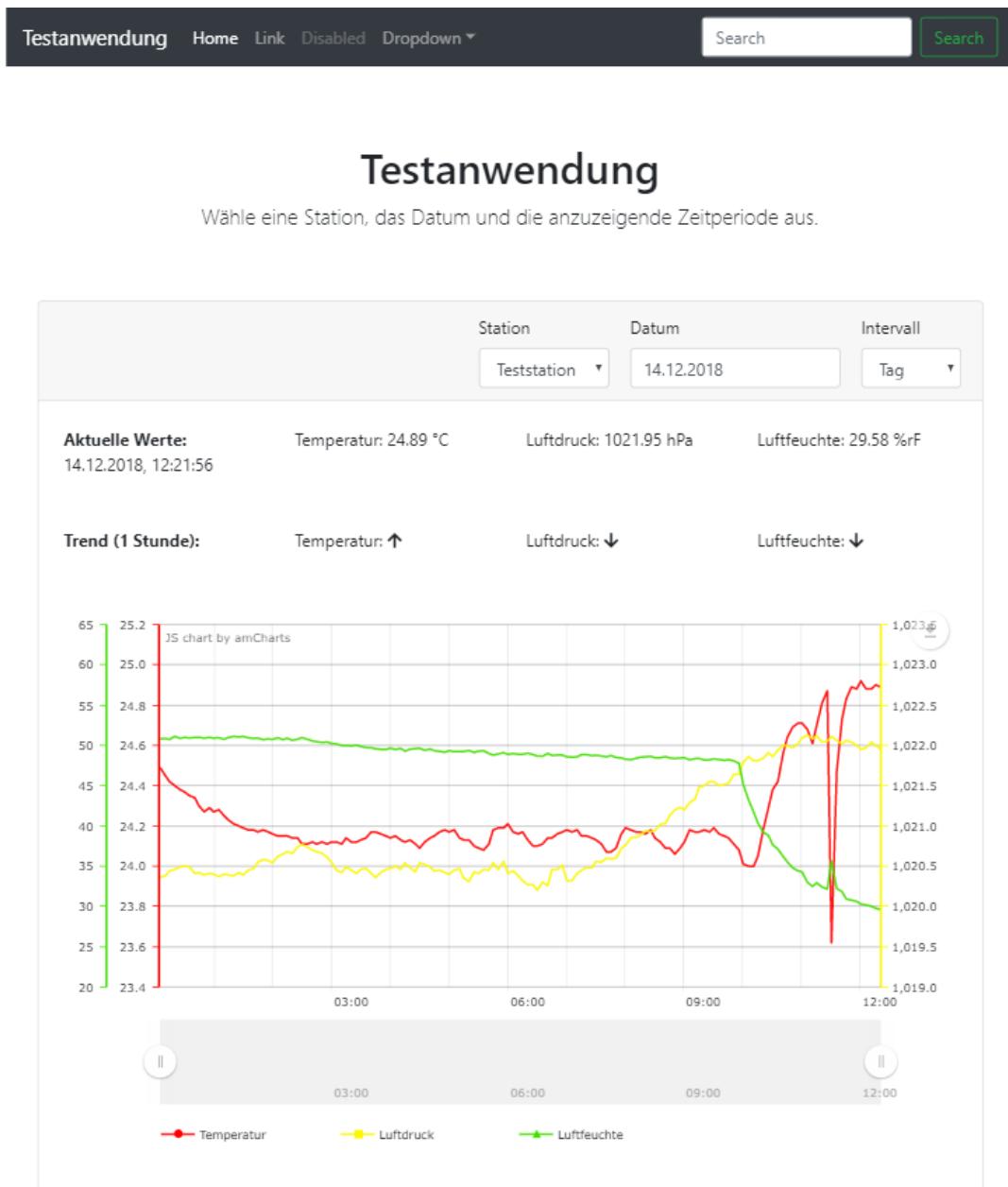


Abb. 6.2: Beispielanwendung zur Anzeige der Messdaten

7 Fazit und Ausblick

In diesem letzten Kapitel sollen die Ergebnisse noch einmal kompakt zusammengefasst, sowie die erstellte Umweltmonitoring-Lösung mit bestehenden Lösungen verglichen werden. Auch sollen Überlegungen zu möglichen zukünftigen Erweiterungen der Station getätigt werden.

7.1 Vergleich mit bestehenden Lösungen am Markt

Im folgenden soll die in dieser Arbeit erstellte Lösung mit zwei bekannten und verbreiteten WLAN-fähigen Lösungen verglichen werden. Dabei sollen Preis, Leistungsumfang und Genauigkeit untersucht werden.

7.1.1 Netatmo Wetterstation

Die Netatmo Wetterstation ist eine weit verbreitete Messstation, die bei einem Preis von etwa 150 € zwei Messstationen, eine für den Innenbereich und eine für außen, mitbringt. Zusätzlich zu Temperatur, Feuchte und Luftdruck erfassen die Netatmo Stationen auch Schalldruck und CO2-Gehalt. Möchte man auch Windgeschwindigkeit sowie Niederschlag erfassen, kann man für etwa 140 € zwei zusätzliche Messgeräte kaufen, die diese Informationen bestimmen. Die Daten können über ein Webinterface oder eine Smartphone-App angesehen werden. Die Innenstation muss dafür dauerhaft mit einer Stromversorgung per Kabel verbunden sein, das Außenmodul wird durch Batterien versorgt (Netatmo 2018).



Abb. 7.1: Netatmo Wetterstation

7.1.2 Froggit WH3000 SE

Die Froggit WH3000 SE kann wie die Netatmo Station über WLAN mit dem Internet verbunden werden. Auch hier können die Daten per Website oder App abgerufen werden. Anders als die Netatmo Station besitzt das Innenmodul auch ein Display, welches die gemessenen Daten darstellt. Gemessen werden Temperatur, Luftdruck, Feuchte, Sonnen-einstrahlung, UV-Strahlung, Niederschlag sowie Windrichtung und Windgeschwindigkeit. Der Preis so einer Station beträgt etwa 165 € (HS Group GmbH & Co. KG 2018).



Abb. 7.2: froggit WH3000 SE

7.1.3 Vergleich der Genauigkeiten

Der Vergleich der Kosten und Genauigkeiten (siehe Tabelle 7.1) zeigt, dass die mit Arduino entwickelte Lösung wirklich kostengünstig ist. Zwar kann sie aktuell nur drei Messgrößen bestimmen, muss sich dabei in puncto Messgenauigkeit nicht verstecken. Die Netatmo Station ist zwar gleich genau, bei der Temperaturbestimmung sogar noch genauer, bietet aber bei einem Preis von 150 € nur noch die zusätzliche CO₂-Messung. Die Bestimmung des Niederschlags und der Windgeschwindigkeit lässt sich Netatmo noch einmal zusätzlich mit 140 € bezahlen. Die froggit WH3000 SE bietet zwar einige zusätzlich bestimmte Parameter, ist dabei aber im allgemeinen doch etwas ungenauer.

	Arduino mit BME280	Netatmo Wetterstation	froggit WH3000 SE
Preis	11 €	150 € + 140 €	165 €
Temperatur	$\pm 1^\circ\text{C}$	$\pm 0.3^\circ\text{C}$	$\pm 1^\circ\text{C}$
Luftdruck	$\pm 1 \text{ hPa}$	$\pm 1 \text{ hPa}$	$\pm 2.7 \text{ hPa}$
Luftfeuchte	$\pm 3 \% \text{ rF}$	$\pm 3 \% \text{ rF}$	$\pm 5 \% \text{ rF}$
Schalldruck	-	k. A.	-
Niederschlag	-	$\pm 1 \text{ mm/h}$	$\pm 10 \%$
Windgeschw.	-	$\pm 1.8 \text{ km/h}$	$\pm 3.5 \text{ km/h}$ bzw. $\pm 10 \%$
CO2-Gehalt	-	$\pm 50 \text{ ppm}$ bzw. $\pm 5 \%$	-
Sonnenstrahlung	-	-	$\pm 15 \%$

Tabelle 7.1: Vergleich mit bekannten Wetterstationen

7.2 Zusammenfassung der Ergebnisse

Der Vergleich verschiedener Sensoren hat gezeigt, dass alle Sensoren, bis auf den SHT31-D im Bereich Feuchte, die Herstellerangaben erfüllen. Der Bosch BME280 ist ein zuverlässiger Sensor, der sowohl Temperatur, Feuchte und Luftdruck mit der vom Hersteller angegebenen Genauigkeit misst. Mit einem Preis von etwa 3€ ist er zudem sehr günstig. Möchte man die Genauigkeit der Temperaturmessung erhöhen, könnte zusätzlich zum BME280 noch der SHT31-D eingesetzt werden. Die Vergleichsergebnisse haben gezeigt, dass dieser Sensor von Sensirion hier ab Werk noch genauere Messwerte liefert.

Allerdings haben die Vergleichsmessungen auch gezeigt, dass alle Sensoren eine jeweils recht konstante Abweichung zum Referenzmessgerät aufwiesen. Die Messung von Temperatur, Luftdruck oder Feuchteunterschieden erfolgt mit unterschiedlichen Sensoren ähnlich genau. Lediglich die vom Hersteller durchgeführte Kalibrierung scheint unterschiedlich präzise zu sein. Bestimmt man also den konstanten Offset mithilfe von kalibrierten Referenzmessgeräten ist auch der BME280 in der Lage, sehr genaue Messwerte zu bestimmen. Die Tabelle 7.2 zeigt zusammengefasst alle mittleren Abweichungen zum Referenzsensor (\bar{x} Abw.) sowie alle Standardabweichungen dieser Abweichungen (σ Abw.).

Das Kapitel 3 hat gezeigt, dass es möglich ist, mit einem Budget von etwas über 10€ eine Sensorstation mit WLAN, Display und Echtzeituhr zu bauen. Bild 7.3 zeigt so eine Station mit einem BME280 auf einer mit EagleCAD entworfenen Platine.

Sensor	Temperatur [°C]		Luftdruck [hPa]		Feuchte [% rF]	
	\bar{x} Abw.	σ Abw.	\bar{x} Abw.	σ Abw.	\bar{x} Abw.	σ Abw.
BMP180 1	0,24	0,10	0,79	0,06	-	-
BMP180 2	-	-	0,71	0,06	-	-
BMP180 kombi	-	-	0,75	0,05	-	-
BME280 1	0,78	0,10	0,56	0,03	0,75	0,21
BME280 2	0,71	0,09	0,87	0,07	1,77	0,19
BME280 3	-	-	0,01	0,04	-	-
BME280 kombi	0,75	0,09	0,47	0,04	1,26	0,20
SHT31-D 1	0,03	0,08	-	-	2,31	0,26
SHT31-D 2	0,09	0,07	-	-	2,65	0,27
SHT31-D kombi	0,06	0,07	-	-	2,48	0,27

Tabelle 7.2: Vergleich aller gemessenen Abweichungen

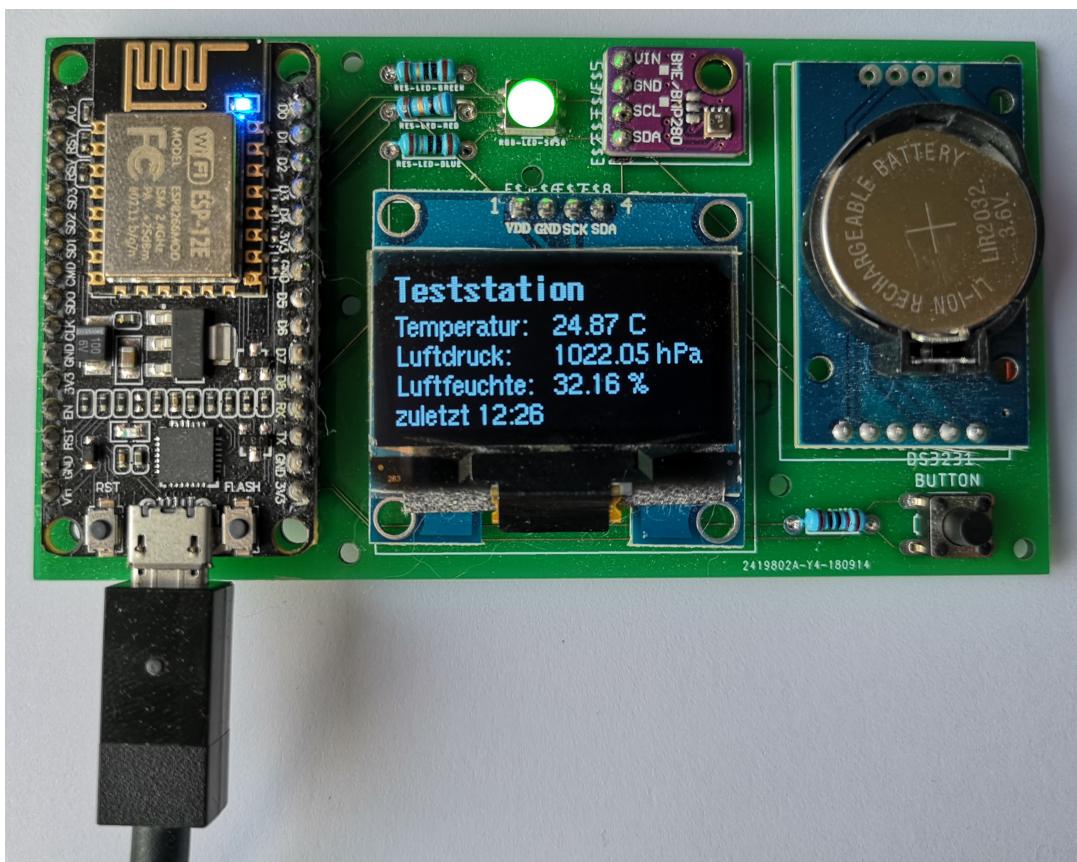


Abb. 7.3: Platine mit Sensorstation

7.2.1 Veröffentlichung der Ergebnisse

Die in dieser Bachelor-Thesis erarbeiteten Ergebnisse (Quellcode, Layout, Schaltplan, Platinenentwurf) wurden auf der Plattform Github unter der Adresse <https://github.com/riccy2/arduino-sensorstation> für jedermann frei zugänglich unter der MIT-Lizenz veröffentlicht.

7.3 Ausblick und mögliche Erweiterungen

Natürlich konnte in den vorherigen Kapiteln nicht das komplette Thema der vernetzten Umweltbeobachtung behandelt werden. Dazu existieren einfach zu viele Parameter, die man beobachten könnte, sowie Möglichkeiten, wie man dies tut. Aber es konnte eine Grundlage gelegt werden, auf die weiter aufgebaut werden kann. Zwei mögliche zukünftige Erweiterungsmöglichkeiten sollen hier noch kurz erläutert werden.

7.3.1 Deepsleep

Die erstellte Sensorstation ist aktuell noch auf eine permanente Stromversorgung per Micro-USB Kabel angewiesen. Allerdings kann der verwendete ESP-8266 auch in einen sogenannten Deepsleep (Tiefschlaf) versetzt werden. Dies hat den Vorteil, dass der Chip und die angeschlossenen Module in diesem Modus kaum Energie benötigen und verbrauchen. Nach einer festgelegten Zeit weckt sich der Chip selbst aus dem Schlaf auf, könnte kurz die Messwerte auslesen und per WLAN weiterschicken, und versetzt sich danach wieder für die Zeit bis zum nächsten Messvorgang in den Schlafmodus.

Gemäß Espressif Inc. (2016) werden im Deepsleep nur etwa $20\text{ }\mu\text{A}$ benötigt. Eine Stromversorgung per Akku, welcher eventuell tagsüber durch ein kleines Solarpanel geladen wird, könnte die Sensorstation dauerhaft autark machen. So könnten Stationen auch an Orten eingesetzt werden, wo keine Stromversorgung vorhanden ist. Für diesen Fall müssten allerdings eventuell auch noch Überlegungen bezüglich eines Gehäuses angestellt werden.

7.3.2 weitere Sensoren

Einige Sensoren, die die Messung von Temperatur, Feuchte und Druck ermöglichen, wurden in dieser Arbeit untersucht. Neben diesen Parametern können natürlich auch viele weitere wie zum Beispiel Lichtintensität, Niederschlag, Windgeschwindigkeit oder Luftqualität gemessen werden. Dafür existieren weitere Sensoren von verschiedenen Herstellern, mit denen die Sensorstation erweitert werden könnte. Falls diese Sensoren ebenfalls den I²C-Standard unterstützen, ist die Einbindung besonders schnell erledigt.

Abkürzungsverzeichnis

GPIO	General Purpose Input/Output
GSM	Global System for Mobile Communications
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
I²C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
LED	Light Emitting Diode
OLED	Organic Light Emitting Diode
RTC	Real-Time Clock
SPI	Serial Peripheral Interface
SPIFFS	Serial Peripheral Interface Flash File System
WLAN	Wireless Local Area Network

Abbildungsverzeichnis

2.1	Arduino Uno - R3, Quelle: SparkFun Electronics	9
3.1	NodeMCU, eigene Aufnahme	12
3.2	BME280, eigene Aufnahme	14
3.3	RTC DS3231, eigene Aufnahme	16
3.4	1.3“ OLED Display, eigene Aufnahme	16
3.5	Verkabelung der Module, eigene Aufnahme erstellt mit fritzing	19
3.6	Schaltplan, eigene Aufnahme erstellt mit fritzing	19
4.1	Vergleich der Temperaturmessung, eigene Aufnahme	21
4.2	Vergleich der Temperaturmessung (Ausschnitt), eigene Aufnahme	22
4.3	Vergleich der Feuchtemessung, eigene Aufnahme	23
4.4	Vergleich der Luftdruckmessung, eigene Aufnahme	24
4.5	TCA9548A, eigene Aufnahme	25
6.1	Entity-Relationship-Diagramm, eigene Aufnahme	34
6.2	RTC Beispielanwendung, eigene Aufnahme	39
7.1	Netatmo Wetterstation, Quelle: www.cyberport.de	40
7.2	froggit WH3000 SE, Quelle: www.froggit.de	41
7.3	Platine mit Sensorstation, eigene Aufnahme	43

Literaturverzeichnis

Ai-Thinker (2015), *ESP-12E WiFi Module*. V. 1.0.

Arduino AG (2018a), ‘Arduino Forum’.

URL: <https://forum.arduino.cc/>, zuletzt besucht: 07.11.2018

Arduino AG (2018b), ‘Arduino Reference’.

URL: <https://www.arduino.cc/en/Reference>, zuletzt besucht: 23.11.2018

Bosch Sensortec (2009), *BMP085 Digital pressure sensor*. Rev. 1.2.

Bosch Sensortec (2013), *BMP180 Digital pressure sensor*. Rev. 2.5.

Bosch Sensortec (2015), *BME280 Combined humidity and pressure sensor*. Rev. 1.1.

Brockhaus (2018), ‘Umweltmonitoring’.

URL: <http://www.brockhaus.de/ecs/enzy/article/umweltmonitoring>, zuletzt besucht: 05.12.2018

Dembowski, K. (2014), *Embedded-Systeme mit der Arduino-Plattform*, VDE Verlag, Berlin.

Espressif Inc. (2016), *ESP8266 Low Power Solutions*. Rev. 1.1.

HS Group GmbH & Co. KG (2018), ‘Froggit WH3000 SE (Second Edition 2018) WiFi App Internet Wetterstation’.

URL: https://www.froggit.de/product_info.php?info=p287_froggit-wh3000-se--second-edition-2018--wifi-app-internet-wetterstation.html, zuletzt besucht: 04.12.2018

Joeckel, R., Stober, M. & Huep, W. (2008), *Elektronische Entfernungs- und Richtungsmesung und ihre Integration in aktuelle Positionierungsverfahren*, Wichmann, Heidelberg.

Lerdorf, R. & Tatroe, K. (2002), *Programmieren mit PHP*, O'Reilly, Köln.

Margolis, M. (2012), *Arduino-Kochbuch: Arduino für Fortgeschrittene; behandelt Arduino 1.0*, 1. Aufl. edn, O'Reilly, Köln.

Maxim Integrated (2008), *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*.

REV: 042208.

Maxim Integrated (2015), *DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*. Rev 10.

Netatmo (2018), ‘Technische Details der Smarten Wetterstation’.

URL: <https://www.netatmo.com/de-de/weather/weatherstation/specifications>, zuletzt besucht: 04.12.2018

Sensirion (2015), *Datasheet SHT3x-DIS*. V. 0.93.

setra (n.d.), *setra Model 470 Digital Pressure Transducers*. Rev. B 4/18/95.

Sino Wealth (2013), *SH1106 132 X 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller*. V. 2.3.

stackoverflow (n.d.), ‘Developer Survey Results 2018’.

URL: <https://insights.stackoverflow.com/survey/2018/>, zuletzt besucht: 15.11.2018

Testo AG (n.d.), *testo 445 / testo 645 Bedienungsanleitung*. V02.00.

Yank, K. (2005), *PHP und MySQL*, dpunkt.verlag.

Universität für Baukunst und Metropolenentwicklung
Studiengang Geomatik

Eidesstattliche Erklärung

Name: Oppermann
Vorname: Riccardo
Matrikelnummer: 6038984
Studiengang: Geomatik, Bachelor

Ich versichere, dass ich die vorliegende Thesis mit dem Titel
Low-Cost Umweltmonitoring mit Arduino in Geosensornetzwerken
selbstständig und ohne unzulässige Hilfe erbracht habe.

Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Hamburg, 17. Dezember 2018

Ort, Datum

Riccardo Oppermann