HTBLuVA Salzburg

Höhere Lehranstalt für
Elektronik und Technische Informatik

DIPLOMARBEIT

Gesamtprojekt

LRS

Entwicklung eines Luftreinigungssystems

André Haumtratz 5CHEL Betreuer: Prof. Dipl. Ing. Siegbert Schrempf

Samuel Kurzmann 5CHEL

Kooperationspartner:

ausgeführt im Schuljahr 2021/22

Abgabevermerk:

Datum: 04.04.2022 übernommen von:

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Salzburg, am 05.04.2022

Verfasserinnen / Verfasser:

Samuel Kurzmann

André Haumtratz

htl bildung mit zukunft	**HÖHERE TECHNISCHE BUNDESLEHR- UND VERSUCHSANSTALT Salzburg**
	Elektronik und Technische Informatik

DIPLOMARBEIT
DOKUMENTATION

Namen der Verfasserinnen / Verfasser	Samuel Kurzmann André Haumtratz
Jahrgang Schuljahr	5CHEL 2021/22
Thema der Diplomarbeit	Entwicklung eines Luftreinigung Systems
Kooperationspartner	

Aufgabenstellung	Luftverschmutzung verursacht gesundheitliche Schäden. Die drei aussagekräftigsten Schadstoffe sind Feinstaub, Stickstoffdioxid und Ozon. Benötigt wird ein Luftreinigungssystem, welches zu einer besseren Luftqualität beiträgt. Besonders empfindliche Personen werden dadurch weniger belastet.

Realisierung	Es wurde ein Luftreinigungssystem entwickelt, welches einen Lüfter betreibt und ein Potentiometer zum Steuern der Lüftergeschwindigkeit besitzt. Es werden Sensoren zur Schadstofferkennung eingebaut. Die Sensor-Daten werden an einem Raspberry-Pi geschickt und ausgegeben, und dann online und auf einem Display dargestellt.

Ergebnisse	Der Lüfter funktioniert, und seine Geschwindigkeit kann eingestellt werden. Der Feinstaub-Sensor schickt die Daten und veranschaulicht sie mithilfe von Graphen.

Prototyp	Der aktuelle Prototyp (sieh Abbildung) besteht aus einem Gehäuse, in welchem sich das Lüftersystem und Filter befindet.

Teilnahme an Wettbewerben, Auszeichnungen	

Möglichkeiten der Einsichtnahme in die Arbeit	

Approbation (Datum / Unterschrift)	Prüferin / Prüfer	Direktorin / Direktor Abteilungsvorständin / Abteilungsvorstand

htl bildung mit zukunft	**COLLEGE OF ENGINEERING** **Salzburg**
	Electronics and Computer Engineering

DIPLOMA THESIS
Documentation

Author(s)	Samuel Kurzmann André Haumtratz
Form Academic year	5CHEL 2021/22
Topic	Development of an air purifier
Co-operation Partners	

Assignment of Tasks	Air pollution causes damage to health. The three most significant pollutants are particulate matter, nitrogen dioxide and ozone. The Development Goal is an air purification system that contributes to better air quality. Particularly sensitive people will be less burdened

Realisation	An air purification system has been developed, which has a fan drive and a potentiometer to control the fan speed. Sensors are also built in to detect pollutants. The sensor data is sent to a Raspberry Pi and the output is displayed online and on a display.

Results	The fan works and its speed can be adjusted. The fine dust sensor sends the data and illustrates it with the help of graphs.

bildung mit zukunft	**COLLEGE OF ENGINEERING** **Salzburg**
	Electronics and Computer Engineering

Illustrative Graph, Photo (incl. explanation)	The current prototype (see figure) consists of a housing containing the filter, fan and sensory system.

Participation in Competitions Awards	

Accessibility of Diploma Thesis	

Approval (Date / Sign)	Examiner	Head of College Head of Department

Vorwort

Luftverschmutzung ist ein Allgegenwertiges Problem und verursacht gesundheitliche Schäden. Diese Schäden fallen besonders bei empfindlichen Personen auf. Es wurde nach einer Lösung gesucht, die nicht nur die Luft filtert, sondern auch andere Schadstoffe misst. Nach intensivem Brainstorming und Recherche kamen wir auf die Schlussfolgerung, dass Feinstaub, Ozon und Stickstoffdioxid die aussagekräftigsten Schadstoffe in der Luft sind. Daher entwickeln wir ein System, das den Feinstaubgehalt misst und filtert. Der Ozon- und Stickstoffdioxidgehalt werden ebenfalls gemessen.

Danksagung

Unser herzlicher Dank gilt unserem Projektbetreuer Prof. Dipl.-Ing. Siegbert Schrempf, der uns während der Arbeit an unserem Projekt hilfreich zur Seite stand.

Zusätzlich möchten wir uns bei sämtlichen Werkstättenlehrern, darunter BEd. FL. Ing. Dipl.-Päd. Herbert Pölzer, FL Ing. BEd. Norbert Wen und FL Ing. Dipl.-Päd. Wolfgang Straßl, bedanken, die uns bei der Fertigung unserer Hardware- und Gehäusekomponenten tatkräftig zur Seite standen.

Inhaltsverzeichnis

1 Überblick 12
2 Systemspezifikation 13
2.1 Zielbestimmungen 13
2.1.1 Musskriterien 13
2.1.2 Wunschkriterien 13
2.1.3 Abgrenzungskriterien 13
2.2 Produkteinsatz 14
2.2.1 Anwendungsbereiche 14
2.2.2 Zielgruppen 14
2.2.3 Betriebsbedingungen 14
2.2.4 Produktfunktionen´ 14
2.3 Produktleistungen 16
2.4 Benutzungsoberfläche 16
2.4.1 Display 16
2.4.2 Website 16
2.5 Qualitätszielbestimmungen 16
2.6 Globale Testszenarien und Testfälle 17
2.7 Entwicklungsumgebung 18
2.7.1 Software 18
2.7.2 Hardware 18
2.7.3 Orgware 18
3 Organisation - Projektmanagement 19
3.1 Projektmanagement 19
3.1.1 Überblick 19
3.2 Individuelle Aufgabenstellungen inkl. Arbeits- und Terminplan (GANTT-Diagramme) 20

3.2.1 Samuel Kurzmann 20
3.2.2 André Haumtratz 21
4 Grundlagen und Methoden 22
4.1 Physiologische Grundlagenforschung 22
4.2 Sensorik 25
4.2.1 PM2.5-Sensor 25
4.2.2 UART-Schnittstelle (Universal Asynchronous Receiver Transmitter) 27
4.2.3 Verbindung des Sensors mit dem Raspberry-Pi 29
4.2.4 Python Code 30
4.2.5 Kommunikation Protokoll Feinstaub-Sensor 33
4.2.6 Testen des Sensors mittels Logic-Analyzer 34
4.2.7 Ozon-Sensor 37
4.2.8 Pin-Belegung des Ozonsensors (MQ-131) 39
4.2.9 Kommunikationsaufbau mit Microcontroller 39
4.2.10 Pinbelegung des MCP3008 40
4.2.11 SPI-Schnittstelle 41
4.2.12 Python-Code 43
4.2.13 Stickstoffdioxid-Sensor 47
4.2.14 Stickstoffdioxid-Sensor mit Arduino-Uno 49
4.2.15 Umstellung Arduino-Uno zu Raspberry-Pi 51
4.2.16 Kommunikation LCD-Display zu Microcontroller 54
4.2.17 I2C-Schnittstelle 54
4.2.18 Display Implementierung in Python 56
4.3 Lüftersystem 57
4.3.1 Einstellung der Lüftergeschwindigkeit 57
4.3.2 ADC Baustein 58

4.3.3 PWM 60
4.3.4 Tacho Signal 61
4.4 Onlinevisualisierung 65
4.4.1 MQTT-Grundlagen 66
4.4.2 Node-RED 69
4.4.3 Influx-DB 79
4.4.4 Grafana 80
5 Ergebnisse – Abnahme 84
6 Literaturverzeichnis 86
7 Verzeichnis der Abbildungen, Tabellen und Abkürzungen 88
7.1 Abbildungen 88
8 Begleitprotokoll gemäß § 9 Abs. 2 PrO 92
8.1 Begleitprotokoll Kurzmann 92
8.2 Begleitprotokoll Haumtratz 93

1 Überblick

Durch ein Luftreinigungssystem werden die Feinstaubpartikel in der Luft gefiltert. Dazu werden Sensoren verbaut die denn Feinstaub, Ozon und Stickstoffdioxidgehalt messen. Die Messdaten werden per Display sowie online veranschaulicht.

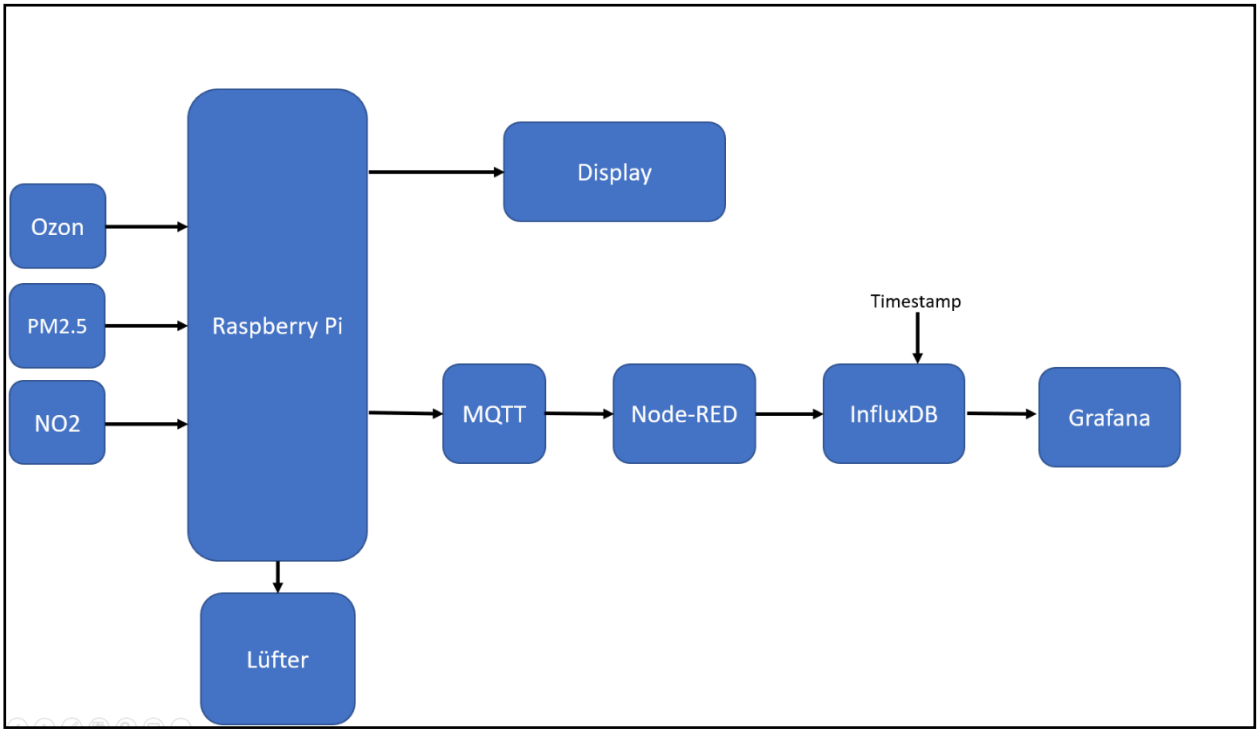


Abbildung 1: Grober Entwurf des Projektes

2 Systemspezifikation

2.1 Zielbestimmungen

2.1.1 Musskriterien

- Lüfter ist ansteuerbar
- Sensorauswertung funktional
- Die Sensoren Daten werden am Display und Online angezeigt

2.1.2 Wunschkriterien

- Das Gehäuse hält der Alltagsbenutzung stand

2.1.3 Abgrenzungskriterien

- Es wird nur der Feinstaubgehalt verbessert (kein Ozon und NO2)
- Das LRS-Projekt wird für die Zimmeranwendung und nicht für die industrielle Anwendung gebaut

2.2 Produkteinsatz

2.2.1 Anwendungsbereiche

Das Luftreinigungssystem ist primär für die Eigenheim gedacht und soll vor allem den Feinstaubgehalt verringern. Das System sollte nicht in großen Werkstätten bzw. Räumen aufgestellt werden, da der Prototyp nicht für so einen Einsatz gedacht ist.

2.2.2 Zielgruppen

Die Hauptzielgruppe besteht aus privaten Nutzern, welche die Luftqualität in ihrem Eigenheim verbessern wollen.

2.2.3 Betriebsbedingungen

Ein idealer Betriebszustand wird bei Raumtemperatur und Zugang zu Netzversorgung erreicht. Es sollte möglichst frei und offen ausgerichtet sein, da sonst weniger Luft gefiltert werden kann.

2.2.4 Produktfunktionen´

/F010/ System mit Spannung versorgen

Die Netzspannung versorgt den Motor

/F020/ Funktionierender Lüfter

Der Lüfter dreht seine Rotoren

/F021/ Lüfter ansteuern

Der Lüfter kann seine Geschwindigkeit mittels Potentiometer ändern

/F030/ Messen

Messwerterfassung durch die Sensorik

/F031/ Messung von Feinstaub

Der PM2.5-Sensor misst den Feinstaubgehalt

/F032/ Messung von Ozon

Der Ozongehalt wird gemessen

/F033/ Messung von NO2

Der NO2 Gehalt wird gemessen

/F040/ Darstellen der Sensor-Daten

Sensor-Daten können dargestellt werden

/F041/ Display zeigt gemessene Daten

Das LCD-Display zeigt die Sensor-Daten

/F042/ Website visualisiert gemessene Daten

Die Sensor-Daten werden auf einer Website veranschaulicht

2.3 Produktleistungen

/L010/ Robustheit

Das Gehäuse schützt vor allgemeinen Benutzer Schäden

/L020/ Effizienz

Das System soll so viel Luft wie möglich filtern

/L030/ Benutzerfreundlichkeit

Der Benutzer sollte ohne Einführung das System intuitiv bedienen können

2.4 Benutzungsoberfläche

2.4.1 Display

Das LDC-Display, soll vereinfacht, kompakt und übersichtlich die Sensor-Daten anzeigen. Die Daten werden automatisch bei Anschalten des Systems dargestellt.

2.4.2 Website

Die Website stellt die Daten online da, falls der Benutzer mal nicht in Reichweite des Geräts ist.

2.5 Qualitätszielbestimmungen

	sehr wichtig	wichtig	weniger wichtig	unwichtig
Robustheit		x		
Zuverlässigkeit	X			
Korrektheit		x		
Benutzerfreundlichkeit	X			
Effizienz		x		
Portierbarkeit		x		

2.6 Globale Testszenarien und Testfälle

/F010/ Funktionierender Lüfter

Die Lüftergeschwindigkeit kann geändert werden

/F022/ Luftfiltern

Der Feinstaubgehalt verbessert sich im Innenraum, während das Lüftersystem angeschaltet ist.

/F030/ Messen

Die Messwerte sind nicht immer die gleichen

/F031/ Messung von Feinstaub

Der Feinstaubgehalt wird gemessen

/F032/ Messung von Ozon

Der Ozongehalt wird gemessen

/F033/ Messung von NO2

Der Stickstoffdioxidgehalt wird gemessen

/F041/ Display zeigt gemessene Daten

Die Messdaten werden am Display dargestellt

/F042/ Website visualisiert gemessene Daten

Die Messdaten werden auf Grafana dargestellt

2.7 Entwicklungsumgebung

2.7.1 Software

- Thonny IDE für das Programmieren der Sensor und Lüfter-Software (3.3.13)
- MQTT senden der Daten an Node-RED
- Node-RED Verarbeitung der Daten und speichern in die Datenbank
- InfluxDB ist eine Datenbank
- Grafana zur Graphen Darstellung der Sensor Daten
- Fusion 360 für den Entwurf und Druck des Gehäuses
- Eagle zum Entwerfen der Platine

2.7.2 Hardware

- Raspberry Pi 3 als zentrale Steuereinheit
- PC-Lüfter für das Lüftersystem
- PM2.5-Sensor zur Messung (PMS3005)
- Ozon-Sensor zur Messung (MQ-131)
- NO2-Sensor zur Messung (MICS2714)
- LCD-Display zur Veranschaulichung der Messdaten

2.7.3 Orgware

- Kanbanflow.com für das Projekt Managements
- Draw.io
- Discord
- Microsoft Teams
- Microsoft Word

3 Organisation - Projektmanagement

3.1 Projektmanagement

3.1.1 Überblick

Samuel Kurzmann

- Entwicklung des Hardware-Prototyps
- Darstellung der Sensor-Daten durch Grafana
- Mechanischer Aufbau des Prototyps
- Gehäuse Filtersystem
- Vertiefende Grundlagenforschung: Datenübertragung in einem lokalen Netzwerk und Untersuchung von IOT Anwendungen

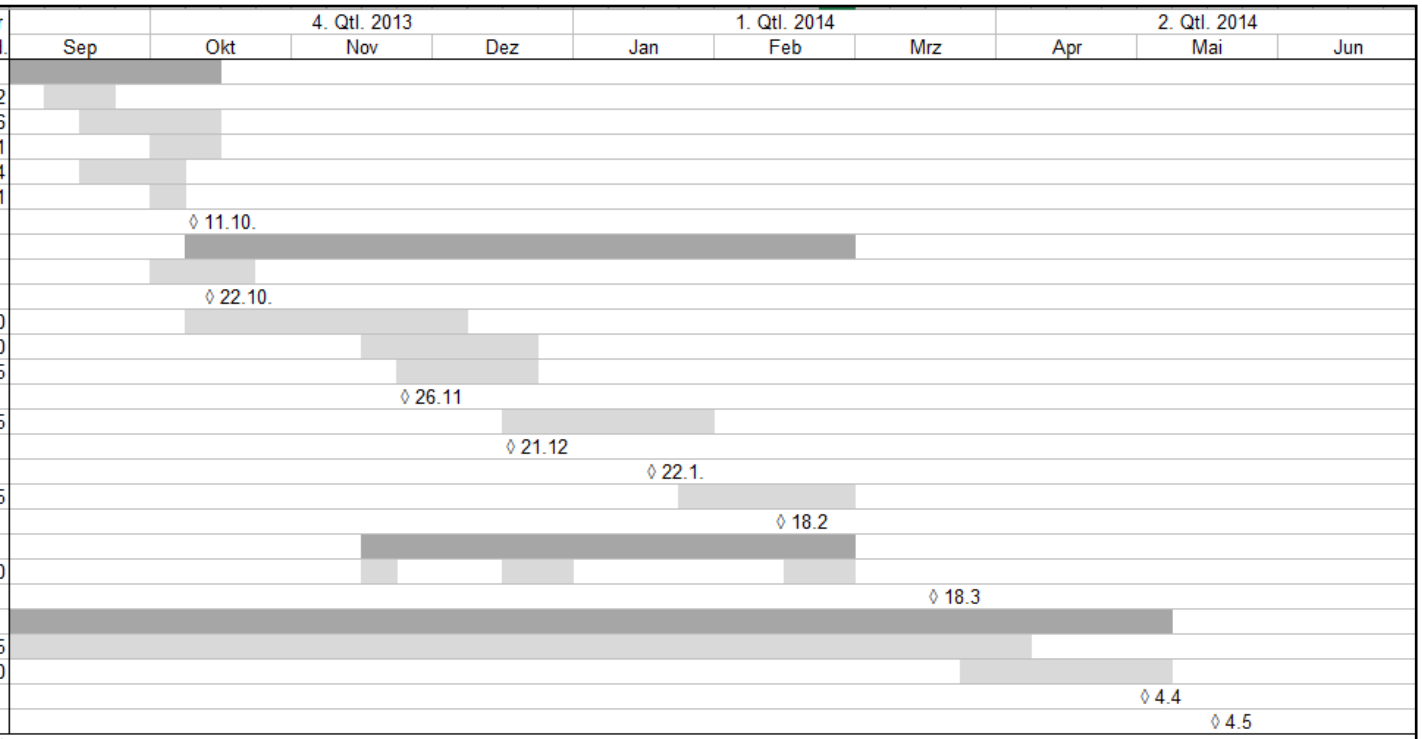
André Haumtratz

- Messung und Auswertung der Sensordaten
- Visualisierung der Sensordaten
- Gehäusekonstruktion
- Systemintegration und Qualitätskontrolle
- Vertiefende Grundlagenforschung: Übertragungsprotokolle

3.2 Individuelle Aufgabenstellungen inkl. Arbeits- und Terminplan (GANTT-Diagramme)

3.2.1 Samuel Kurzmann

Nr.	Vorgangsname	Dauer in Std.
1	**Planungsphase**	
2	Analyse	2
3	Grobentwurf	6
4	Kostenkalkulation	1
5	DA-Antrag erstellen	4
6	Projektplan erstellen	1
7	DA-Antrag Abgabe	
8	**Entwicklungs- u. Implementierungsphase**	
9	Einarbeitungsphase	
10	1. Review Systemspezifikation	
11	Motoransteuerung	30
12	Lüfterintegration	10
13	Filtersystem	5
14	2. Review Prototyp	
14	Mechanischer Aufbau des Prototyps	15
15	1. Präsentation	
16	2. Review	
17	Onlinedarstellung der Sensor-Daten	25
18	3. Review	
19	**Testphase**	
20	Systemtests und Fehlerbehebung	30
21	4. Review	
22	**Dokumentation**	
23	Dokumentation	25
24	Vertiefende Grundlagenarbeit	30
25	DA-Abgabe	
26	**Schlusspräsentation**	
	Summe	184



Entwicklung eines Luftreinigungssystem

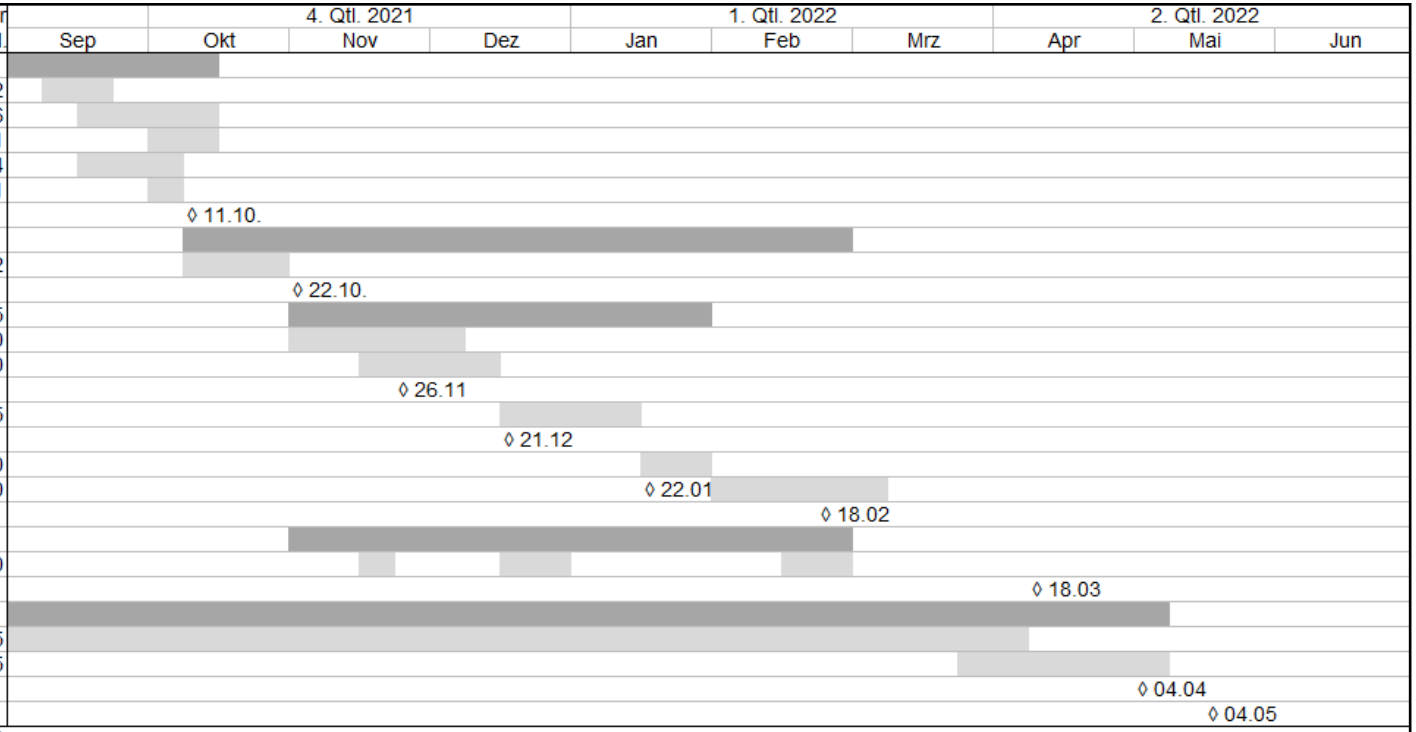
Samuel Kurzmann

Vorgang Meilenstein ◊ Sammelvorgang

Abbildung 2: GANTT-Diagramm KurS

3.2.2 André Haumtratz

Nr.	Vorgangsname	Dauer in Std.
1	**Planungsphase**	
2	Analyse	2
3	Grobentwurf	6
4	Kostenkalkulation	1
5	DA-Antrag erstellen	4
6	Projektplan erstellen	1
7	DA-Antrag Abgabe	
8	**Entwicklungs- u. Implementierungsphase**	
9	Einrichtung der Programmierumgebung	2
10	1. Review Systemspezifikation	
11	Inbetriebnahme der Sensoren	25
12	Ausgabe der Sensor-Daten	30
13	Installierung eines Displays	10
14	2. Review Prototyp	
15	Darstellung der Sensor-Daten am Display	15
16	1. Präsentation	
17	Übergabe der Daten zur Online-Darstellun	10
18	Tag der offenen Tür	10
19	3. Review 2. Prototyp	
20	**Testphase**	
21	Systemtests und Fehlerbehebung	30
22	4. Review	
23	**Dokumentation**	
24	Dokumentation	25
25	Vertiefende Grundlagenarbeit	15
26	DA-Abgabe	
27	**Schlusspräsentation**	
	Summe	186



Entwicklung eines Luftreinigungssystem

André Haumtratz

Vorgang Meilenstein ◊ Sammelvorgang

Abbildung 3: GANTT-Diagramm HauA

4 Grundlagen und Methoden

4.1 Physiologische Grundlagenforschung

Allgemeines

Durch die menschengeschaffene Luftverschmutzung werden immer mehr Schadstoffe erzeugt. Darunter sind die drei aussagekräftigsten Schadstoffe: Feinstaub, Ozon und Stickstoffdioxid. Daher wird ein Filtersystem benötigt, welches den Feinstaub in der Luft filtert und so empfindliche Personen hilft und schützt.

Lüftergeschwindigkeit

Die Lüftergeschwindigkeit eines normalen Luftreinigers ist sehr unterschiedlich, meist liegen die Geschwindigkeiten zwischen 300-3000 Umdrehungen/min. Wir haben ein Mittelmaß gesucht, um genug Luft durch den Filter zu blassen. Einen Motor mit einer höheren Umdrehungszahl ist sehr kostspielig, daher wählen wir ein Mittelmaß mit ungefähr 1000 Umdrehungen/min.

Feinstaub

Partikel mit einem sehr kleinen Durchmesser werden als „Feinstaub“ bezeichnet. Die Zusammensetzung kann sowohl aus organischen als auch anorganischen Material bestehen. Da es über ein hohes Absorptionspotential für gasförmige Stoffe verfügt, reichern sich besonders Pestizide und Weichmacher einfach am Feinstaub an. Daher reizen die Partikel die Atemwege und können dort zu entzündlichen Veränderungen führen. Besonders kleiner Feinstaub wie PM2.5 (Feinstaub mit einem Durchmesser von 2.5um) kann daher in den Blutkreislauf gelangen und somit das Herz-Kreislauf-System beeinträchtigen.

Bezeichnung	PM 1	PM 2,5	PM 10
Umweltbundesamt Tagesgrenzwert	-	-	50 µg/m³
Umweltbundesamt Jahresgrenzwert	-	25 µg/m³	40 µg/m³

Abbildung 4: Grenzwerte von Feinstaub [1]

Ozon

Ozon (O_3) ist ein farbloses bis leicht bläuliches und stechend-scharf riechendes Gas. Es ist eines der wichtigsten Gase in der Atmosphäre und bildet sich erst in 20-30 Kilometern höhe. Die Ozonschicht schützt und von schädlichen Ultravioletstrahlung der Sonne. Jedoch ist die Bildung von Ozon in Bodennähe die Folge einer Reaktion von Sauerstoff, Stickoxiden und organischen Verbindungen bei intensiver Sonnenstrahlung. Es ist daher oxidierend und wirkt giftig auf den Menschen und ist auch schwerer als Luft. Die menschliche Wahrnehmung des Gases liegt bei 40µg/m³. Der Alarmwert von Ozon liegt bei einem 1h-Mittelwert von 240µg/m³.

Bezeichnung	Grenzwerte Ozon
Umweltbundesamt maximaler 8h-Wert	120 µg/m³
Umweltbundesamt Informationsschwelle (1h-Mittelwert)	180 µg/m³
Umweltbundesamt Alarmwert (1h-Mittelwert)	240 µg/m³

Abbildung 5: Ozongrenzwerte [2]

NO2

Stickstoffdioxid (NO2) entsteht durch Verbrennung fossiler Energiestoffe, unter anderem wie Kohle, Gas und Öl. NO2 ist ein rotbraunes Gas, welches stark oxidierend und gut in Wasser löslich ist. Es ist zudem giftig und äußerst korrosiv (zerfressend). Stickstoff ist schwerer als Luft und sammelt sich daher auch am Boden ab. Folgen einer zu hohen Konzentration Aussetzung am Menschen kann dazu führen, dass die Bronchien verengt werden. Daher reagieren Personen mit Asthma oder Bronchitis besonders empfindlich auf eine hohe Stickstoffdioxid Belastung.

Bezeichnung	EU-Außenluftgrenzwerte	Arbeitsplatzgrenzwerte	Innenraumrichtwerte
Langzeitwerte	40 µg/m³	-	40 µg/m³
Kurzzeitwerte	Immissionsgrenzwert 1h-Mittelwert 200 µg/m³, Alarmschwelle 3h-Mittelwert 400 µg/m³	-	Vorsorgewert 1h-Mittelwert 80 µg/m³, Gefahrenwert 1h-Mittelwert 250 µg/m³

Abbildung 6: Stickstoffdioxidgrenzwerte [11]

Filter

Als Filter wird ein HEPA-Filter (High Efficiency Particulate Air) verwendet, welcher laut Hersteller bis zu 99.97% der Staubpartikel Filtern kann.

Der Belastungsbereich denn das Produkt damit Filtern kann, liegt bei ein Staubdurchmesser mit größer als 2.5um. Solch ein Staub gelangt bis in die Bronchien.

4.2 Sensorik

Es werden drei Sensoren in diesem System verwendet. Der PM2.5-Sensor, welcher den Feinstaubgehalt misst. Der Ozon-Sensor, der für die Messung des Ozongehalts zuständig ist und der Stickstoffdioxid-Sensor, welcher den Stickstoffdioxidgehalt in der Luft misst.

4.2.1 PM2.5-Sensor

Hierbei handelt es sich um einen optischen Feinstaubsensor (PMS5003). Der PMS5003 Sensor erkennt und zählt Partikel mittels Lichtstreuung. Es strömt Luft durch die Detektionskammer, die Partikel passieren einen Laserstrahl, der an Partikeln gestreut wird. Das Streulicht wird von einer Diode empfangen und in ein elektrisches Signal gewandelt, über welches eine Feinstaubkonzentration berechnet wird. Der Sensor misst verschiedene Feinstaub-Werte. Beispielsweise Partikel mit dem Durchmesser von 1µm, 2.5µm und 10µm und gibt diese dann in µg/m^3 an.

Funktionsblock des Sensors

Wie in Punkt 4.2.1 beschrieben, sieht man in Abbildung 7 die Funktionsblöcke des PM2.5-Sensors

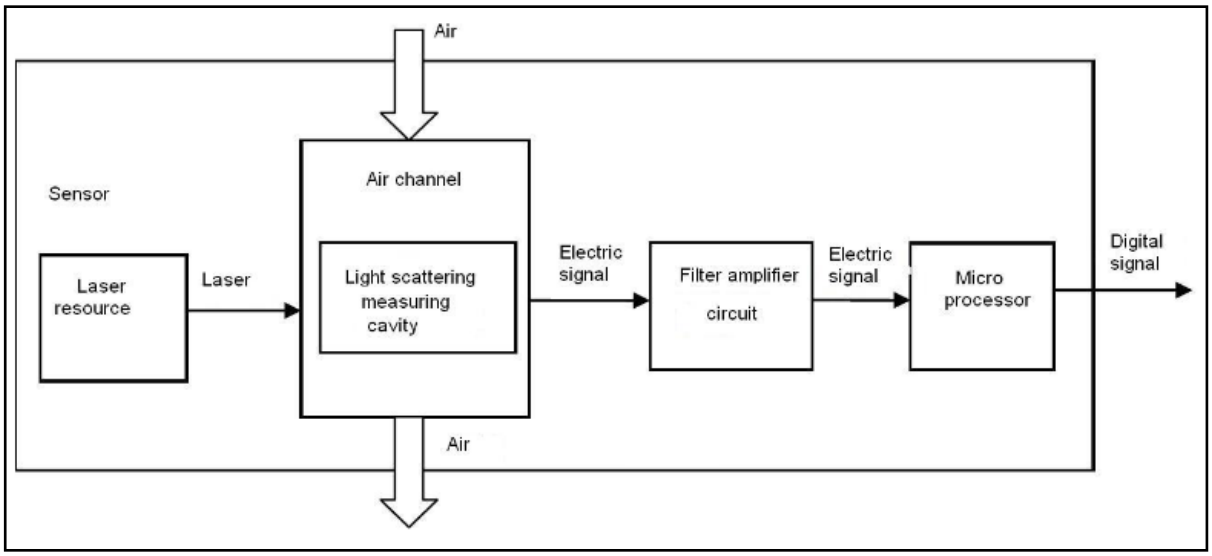


Abbildung 7: Blockschaltbild des PM2.5-Sensors [3]

Pin-Belegung des PMS5003

Bei Abbildung 8 ist der Sensor und seine Pins zu erkennen. Der Feinstaub-Sensor besitzt eine UART-Schnittstelle. Abbildung 9 beschreibt die Pin-Belegung des Sensors genauer.

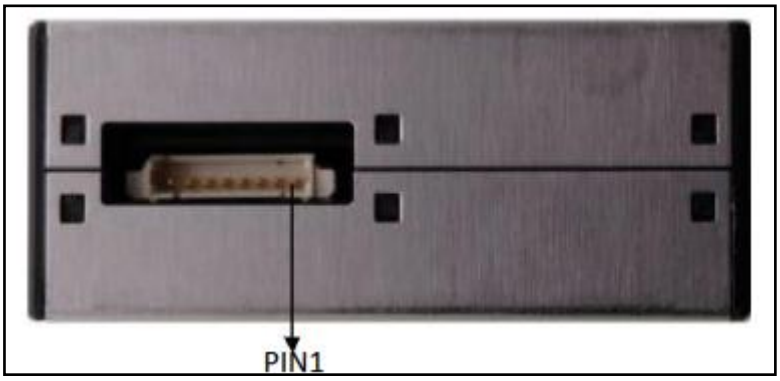


Abbildung 8: Pins des PM2.5-Sensor [12]

PIN1	VCC	Positive power 5V
PIN2	GND	Negative power
PIN3	SET	Set pin /TTL level@3.3V，high level or suspending is normal working status, while low level is sleeping mode.
PIN4	RX	Serial port receiving pin/TTL level@3.3V
PIN5	TX	Serial port sending pin/TTL level@3.3V
PIN6	RESET	Module reset signal /TTL level@3.3V，low reset.
PIN7/8	NC	

Abbildung 9: Pin-Belegung des Sensors [12]

Kommunikation Feinstaubsensor mit Raspberry-Pi

Das Ziel ist, dass ein Python-Programm auf dem Mikrocontroller (Raspberry-Pi) geschrieben wird, um Daten des Sensors zu erhalten. Das heißt, eine UART-Kommunikation muss zwischen Feinstaub-Sensor und Raspberry-Pi aufgebaut werden.

4.2.2 UART-Schnittstelle (Universal Asynchronous Receiver Transmitter)

UART definiert ein Protokoll, das heißt eine Menge von Regeln zum Austausch von seriellen Daten zwischen zwei verschiedenen Geräten.

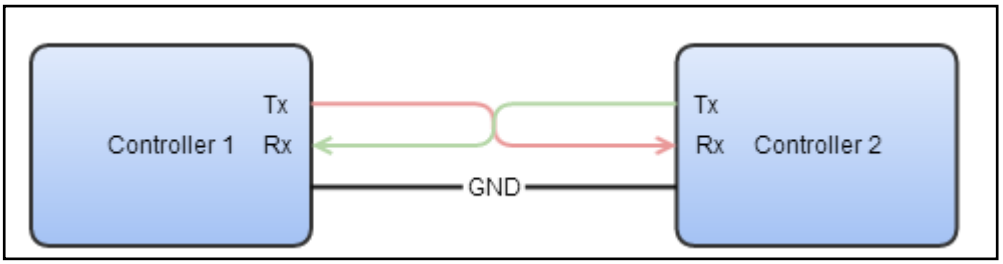


Abbildung 10: Darstellung einer UART-Schnittstelle [4]

In Abbildung 10 wird die elektrische Verbindung beschrieben. Die Verbindung ist sehr einfach, da es nur zwei Leitungsdrähte benötigt. Sie liegen zwischen Sender und Empfänger, um in beide Richtungen Daten senden und empfangen zu können. Das bedeutet das RX mit TX verbunden wird und TX mit RX. Die Kommunikation braucht zusätzlich noch ein gemeinsames Potenzial (GND).

Die Kommunikation kann in mehreren Betriebsarten ausgeführt werden. Im Simplex-Betrieb werden die Daten nur in eine Richtung gesendet. Der Halbduplex-Betrieb, wo beide Seiten Daten senden können, aber nicht zur gleichen Zeit. Und der Vollduplex-Betrieb, bei dem beide Seiten gleichzeitig senden können.

Der Vorteil von UART ist, dass es asynchron arbeitet. Somit verwenden Sender und Empfänger kein gemeinsames Taktsignal. Daher müssen beide Geräte mit derselben Baudrate (Schrittgeschwindigkeit) übertragen, um das Bit-Zeitverhalten sicherzustellen. Ein paar der bekanntesten UART-Baudraten sind 4800, 9600 oder 115200. Zusätzlich zur selben Baudrate, müssen sie auch dieselbe Rahmenstruktur besitzen, um die gleichen Parameter zu nutzen. Eine Datenübertragung besteht aus einem UART-Rahmen, welcher Start-, Stopp-, Datenbits und Paritätsbit beinhaltet. Da asynchron gearbeitet wird, muss dem Sender signalisiert werden, dass ein Datenpaket kommt. Dies passiert mithilfe des Startbits. Durch das Startbit wird ein Übergang vom Ruhezustand High in den Low-Zustand gemacht (siehe Abbildung 11). Nachdem Startbits erfolgen die Datenbits. Es können zwischen 5 und 9 Datenbits gesendet werden. Nach den Datenbits erfolgen die Stoppbits, wobei es ein bis zwei Stoppbits geben kann. Die Stoppbits gehen dann entweder zurück in den High-Zustand oder in den Ruhezustand. Ein UART-Rahmen kann auch über ein optionales Paritätsbit verfügen, das für die Fehlererkennung zuständig ist. Der Wert der Parität kann gerade oder ungerade sein.

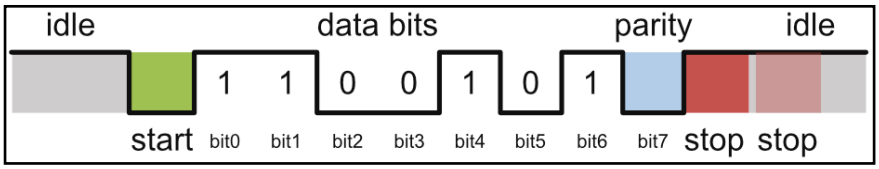


Abbildung 11: Rahmenstruktur einer UART-Übertragung [13]

4.2.3 Verbindung des Sensors mit dem Raspberry-Pi

In Abbildung 12 sind die vordefinierten GPIO-Pins des Mikrocontrollers zu sehen, welche für das UART-Interface wichtig sind. Daher ist zu erkennen, dass die UART-Pins auf Pin 8 und 10 liegen.

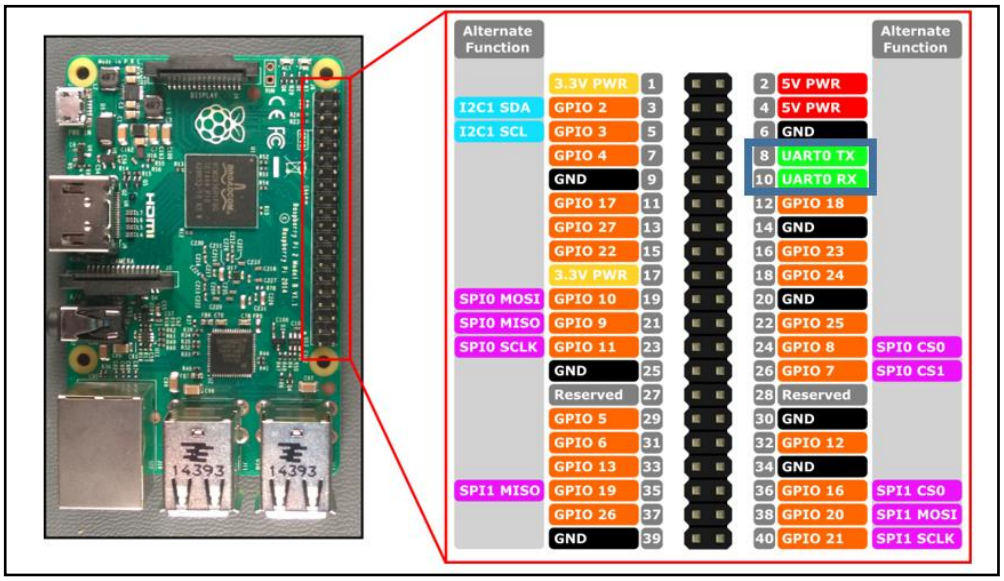


Abbildung 12: Raspberry-Pi Pin-out [5]

Da nur Daten vom Sensor empfangen werden sollen und keine vom Raspberry-Pi gesendet werden, ist TX vom Sensor (Pin 5) mit RX des Raspberry-Pis (Pin 10), Ground und VCC miteinander verbunden.

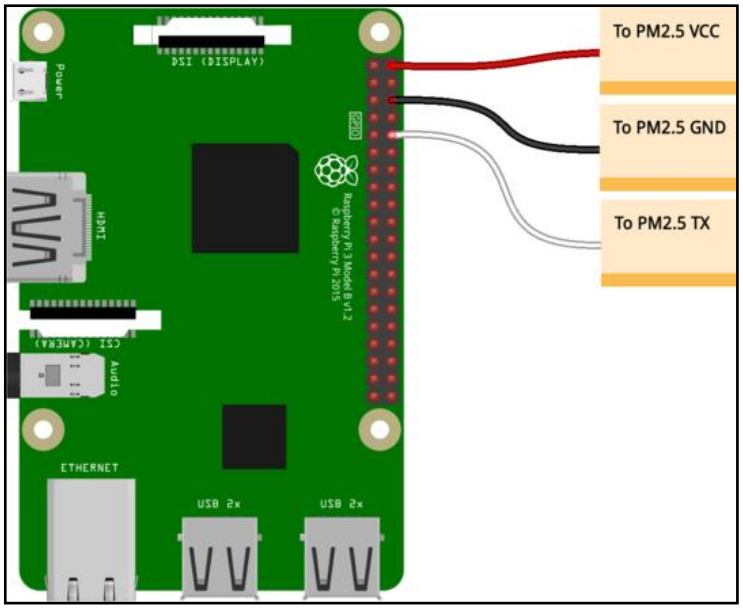


Abbildung 13: Raspberry-Pi (links) angeschlossen an den PM2.5-Sensor (rechts)

4.2.4 Python Code

In Abbildung 14 ist die Übertragungskette des Sensors zu MQTT zusehen. Abbildung 14 gibt Auskunft, welche Funktionen implementiert werden sollen.

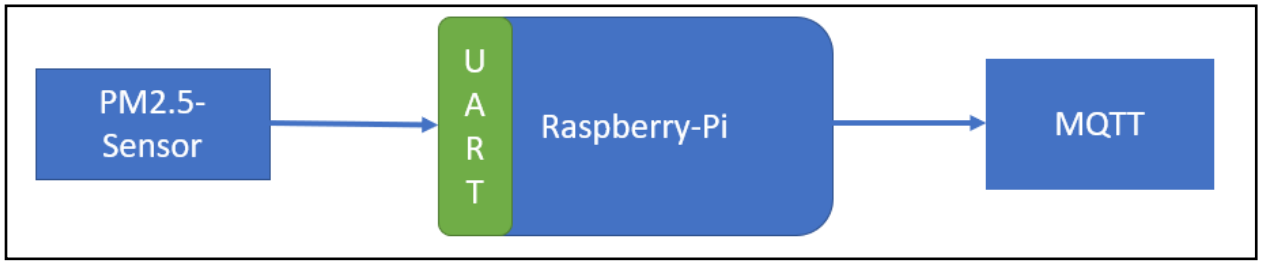


Abbildung 14: Blockschaltbild der Übertragungskette

Der erste Schritt ist es, die benötigten Funktionen und Librarys, wie in Zeile 1-11 (siehe Abbildung 15), zu importieren. Denn diese Zeilen verhelfen dem Programm UART und MQTT Funktionen zu nutzen. Für das Empfangen der Feinstaub Daten ist die Libary *adafruit_pm25.uart (Zeile 11)* am wichtigsten. Diese Libary ist von Adafruit öffentlich zur Verfügung gestellt worden und wird in Abbildung 18 genauer beschrieben. Da ein ganzer Rahmen mit mehreren Feinstaubwerten geschickt wird, aber nur der ein Bestimmter Wert (PM2.5-Wert) von Nutzen ist, wird das Datenpaket geteilt. Dies passiert in Zeile 35-39 (siehe Abbildung 17). Die restlichen Zeilen des Codes sind für die Annahme und Ausgabe von Daten gedacht. Diese sind mit Kommentaren versehen, um das Verständnis der einzelnen Zeilen zu erleichtern.

Loop des Python-Codes

```
1  import paho.mqtt.publish as publish
2  import time
3  import random
4  #pylint: disable=unused-import
5  import time
6  import board
7  import busio
8  import re
9  import serial # Serial wird für UART Importiert
10
11 from adafruit_pm25.uart import PM25_UART   # Importieren der Libary für den Datenframe des PM2.5-Sensors,
12
13 # Einstellung für MQTT-Server
14 MQTT_SERVER = "localhost"
15 MQTT_PATH = "test_channel"
16
17 # UART-Schnittstelle
18 uart = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=0.25)
19
20
21 pm25 = PM25_UART(uart)   # Feinstaub Daten werden in pm25 hinein geschrieben
22
23 print("PM2.5-Sensor gefunden, lese Daten...")
24 while True:
25     time.sleep(1)
26     try:
27         aqdata = pm25.read()   #
28         print("Erkenne keine Daten des Sensors")
29         continue
```

Abbildung 15: Loop des Codes zum Auslesen der Daten (Zeile 1-29)

```
30
31     print()
32     print("PM2.5 Konzentration:")
33
34     # Nur die PM2.5 Daten bekommen
35     str_daten = str(aqdata)                # Daten werden in einen String gefasst
36     teilen_strichp=str_daten.split(":")    # Der String str_daten wird  nach : geteilt
37     pos=teilen_strichp[2]                  # Die dritte Position von teile_strichp wird in pos geschrieben
38     teilen_beistrich=pos.split(",")        # String pos wird nach , aufgeteilt
39     pm25_wert = teilen_beistrich[0]        # Es wird die erste Stelle von teilen_bestreich hergenommen,
40                                            # da der Feinstaub Wert für 2.5um an dieser Stelle liegt
41
42
43     publish.single(MQTT_PATH, pm25_wert, hostname=MQTT_SERVER)   # Daten werden an MQTT geschickt
44     print(pm25_wert)   # Gibt die PM2.5 Konzentration in der Konsole aus
45     time.sleep(2)
46
47
```

Abbildung 16: Loop des Codes zum Auslesen (Zeile 30-47)

Libary für das Datenpacket

```
1  import time
2  from digitalio import Direction, DigitalInOut
3  from . import PM25
4
5  try:
6      # Used only for typing
7      import typing  # pylint: disable=unused-import
8      from busio import UART
9  except ImportError:
10     pass
11
12
13 class PM25_UART(PM25):
14
15     def __init__(self, uart: UART, reset_pin: DigitalInOut = None):
16         if reset_pin:
17             # Reset device
18             reset_pin.direction = Direction.OUTPUT
19             reset_pin.value = False
20             time.sleep(0.01)
21             reset_pin.value = True
22             # it takes at least a second to start up
23             time.sleep(1)
24
25         self._uart = uart
26         super().__init__()
27
28     def _read_into_buffer(self) -> None:
29         while True:
30             b = self._uart.read(1)
31             if not b:
32                 raise RuntimeError("Unable to read from PM2.5 (no start of frame)")
33             if b[0] == 0x42:
34                 break
35         self._buffer[0] = b[0]  # first byte and start of frame
36
37         remain = self._uart.read(31)
38         if not remain or len(remain) != 31:
39             raise RuntimeError("Unable to read from PM2.5 (incomplete frame)")
40         self._buffer[1:] = remain
```

Abbildung 17: Libary Code des PM2.5-Sensors, für den Daten-Rahmen

In Abbildung 18 ist die Adafruit PM2.5 Libary zu sehen. Diese Libary beschreibt die nötigen Befehle und Informationen, um den richtigen UART-Rahmen des Sensors zu erhalten. Abbildung 18 beschreibt die nötigen Konfigurationen, für eine UART Übertragung.

Zeile 28: Initiiert die Funktion *_read_into_buffer*, welche für die Annahme der Daten-Pakete zuständig ist. In der Funktion wird geprüft, ob das richtige Start-bit (0x42, siehe

Abbildung 19) geschickt wurde und ob auch die richtige Rahmenlänge (32 Bytes, aus dem Datenblatt) versendet wurde.

Das bedeutet, dass diese Libary die empfangenen Daten-Pakete überprüft und sie in die Variable *PM25* speichert.

4.2.5 Kommunikation Protokoll Feinstaub-Sensor

Communication Protocol - UART

UART Serial Configuration	
Baud Rate	9600 bps
Data Bits	8
Parity	None
Stop Bits	1

Communication Protocol - Default Output UART Frame Format

Head 1	0x42	
Head 2	0x4D	
H_Length	...	Length = 2 * 13 + 2 (Data+CS)
L_Length	...	
H_D1	...	PM1 = H_D1 * 256 + L_D1 ug/m3 (Standard Smoke, Calculated Value
L_D1	...	
H_D2	...	PM2.5 = H_D2 * 256 + L_D2 ug/m3 (Standard Smoke, Calibrated Value)
L_D2	...	
H_D3	...	PM10 = H_D3 * 256 + L_D3 ug/m3 (Standard Smoke, Calculated Value)
L_D3	...	
H_D4	...	PM1 = H_D4 * 256 + L_D4 ug/m3 (Environment, Calculated Value)
L_D4	...	
H_D5	...	PM2.5 = H_D5 * 256 + L_D5 ug/m3 (Environment, Calculated Value)
L_D5	...	
H_D6	...	PM10 = H_D6 * 256 + L_D6 ug/m3 (Environment, Calculated Value)
L_D6	...	

Abbildung 18: Ausschnitt des Kommunikation-Protokolls [6]

Die obere Abbildung ist ein Ausschnitt aus dem *SM-UART-04L Particulate Dust Sensor* Datenblatt. Diese Spezifikation beschreibt nicht nur die nötigen Rahmenformat Informationen, sondern auch die serielle Konfiguration. Diese Konfiguration ist wichtig für das Testen mittels Logic-Analyzer.

4.2.6 Testen des Sensors mittels Logic-Analyzer

Beim Ausführen der Loop für den PM2.5-Sensor, werden Daten in der Python-IDE Thonny ausgegeben. Um zu wissen, ob die Daten auch richtig gesendet werden, wird ein Logic-Analyzer (*LA*) mit der UART Schnittstelle verbunden. Es wird das GND des Analyzers an den des Sensors und CH0 des *LA* an die Datenleitung des Sensors angeschlossen. Der Analyzer zeigt dann das empfangene Daten-Packet in der Software des Logic-Analyzers an.

Messung des PM2.5-Sensors mit Logic-Analyzer

Nachdem Anschließen des Logic-Analyzer müssen die Einstellungen zum korrekten Protokoll getroffen werden.

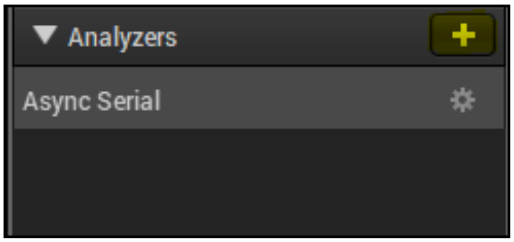


Abbildung 20: Wählen des richtigen Analyzers

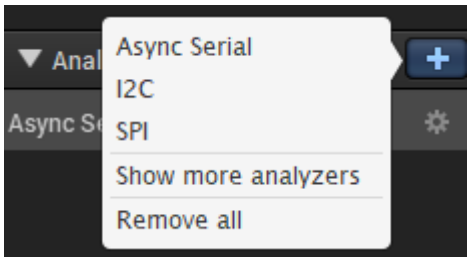


Abbildung 19: Hinzufügen einer Übertragungsprotokoll

Zuerst muss das richtige Übertragungsprotokoll gewählt werden. Das heißt es muss, wie bei Abbildung 20 und 21, ein Analyzer hinzugefügt werden. Da der Sensor mit UART betrieben wird, muss „Async Serial“ gewählt werden.

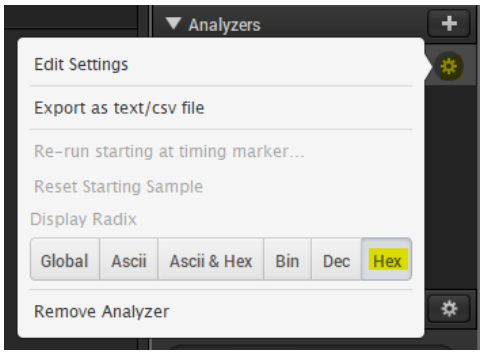


Abbildung 21: Settings Logic-Analyzers

Dann wird wie in Abbildung 18 auf Hex eingestellt, umso die Daten im Hexadezimal Format zu erhalten. Somit erkennt man einfacher, ob das die Start-Bits mit dem Datenblatt übereinstimmen.

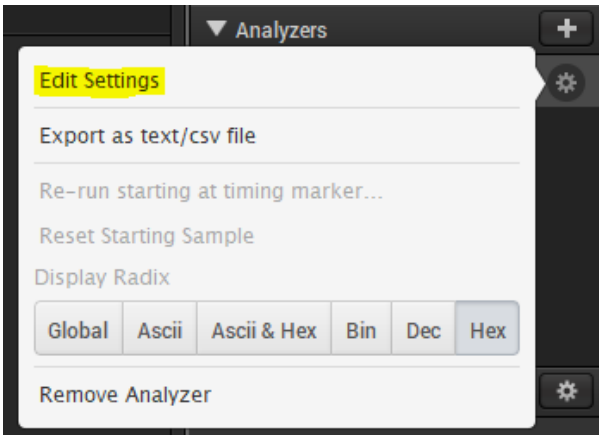


Abbildung 23: Genauere Einstellungen für UART treffen

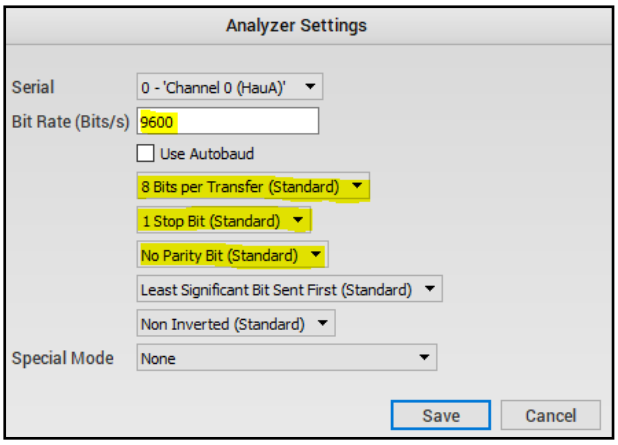


Abbildung 22: UART Einstellungen laut Datenblatt

Als nächste sind die genaueren UART-Einstellungen zu treffen, welche durch „Edit Settings“ bestimmt werden können. Die Einstellungen können durch Abbildung 18 bestimmt werden, indem man auf das „Communication Protocol – UART“ blickt. Die Einstellungen müssen dann wie in Abbildung 22 getroffen und eingestellt werden. Das heißt: Baudrate = 9600, 8 Daten Bits per Transfer, 1 Stopp Bit und kein Parity Bit.

Nachdem die richtigen Einstellungen getroffen worden sind, kann das Programm gestartet werden.

Messergebnisse

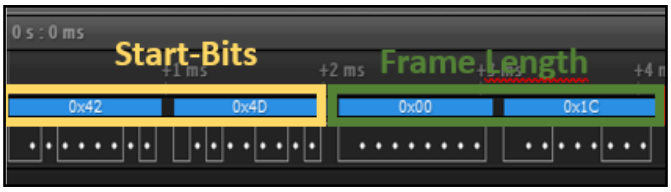


Abbildung 24: Messergebnis mit Logic-Analyzer (Teil 1)

Abbildung 25: Messergebnis mit Logic-Analyzer (Teil 2)

In Abbildung 21 ist das Messergebnis des empfangenen Daten-Pakets zu sehen. Die Daten-Pakete sind wie Abbildung 24, 25 und 26 farblich abgestimmt. Es ist zu erkennen,

dass die Start-Bits mit dem Datenblatt übereinstimmen und auch die Rahmenlänge (Frame length) mit 0x1C, dezimal ≙ 28, stimmt.

Start character 1	0x42	(Fixed)
Start character2	0x4d	(Fixed)
Frame length high 8 bits	……	Frame length=2x13+2(data+check bytes)
Frame length low 8 bits	……	
Data 1 high 8 bits	……	Data1 refers to PM1.0 concentration unit μ g/m3（CF=1，standard particle）*
Data 1 low 8 bits	……	
Data2 high 8 bits	……	Data2 refers to PM2.5 concentration unit μ g/m3（CF=1，standard particle）
Data2 low 8 bits	……	
Data3 high 8 bits	……	Data3 refers to PM10 concentration unit μ g/m3（CF=1， standard particle）
Data3 low 8 bits	……	

Abbildung 26: UART-Rahmen Format (Datenblatt https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf)

Ausgabe des Python Programm

```
PM 1.0: 1ug/m3          PM 2.5: 4ug/m3          PM 10: 4ug/m3
```

Abbildung 27: Ausschnitt des Daten-Outputs zum gleichen Zeitpunkt, wie der der Logic-Analyzer Messung

Das Python-Programm wird zum gleichen Zeitpunkt, wie die der Logic-Analyzer Messung gemacht. Damit die Werte des Analyzers sowie des Programms, auf Einstimmigkeit überprüft werden können.

Berechnung mittels Datasheet

Man bekommt den Wert des Feinstaubgehalt durch diese Formel (aus dem Datenblatt):

H_D*x* * 256 + L_D*x* = [ug/m^3]

Man muss die Datenbits auf Dezimal für die Rechnung umformen.

Hex	Dezimal
0x00	0
0x01	1
0x04	4

Abbildung 28: Hex zu Dezimal

PM1.0 = 0 * 256 + 1 = 1ug/m^3

PM2.5 = 0 * 256 + 4 = 4ug/m^3

PM10 = 0 * 256 + 4 = 4ug/m^3

Die Errechneten werden sind die gleichen wie die in Abbildung 27 (Python Programm). Daher ist zu schließen, dass das Datenpaket richtig gesendet und empfangen wurde. Jedoch kann man nicht sagen, ob dieser Werte auch wirklich stimmen. Dies würde nur gehen, wenn ein anderes Messgerät auch zum gleichen Zeitpunkt misst und dieselben Werte ausgibt.

Wie in den physiologischen Grundlagen beschrieben ist, liegt der Grenzwert bei 25ug/m^3. Wenn von diesem Grenzwert ausgegangen wird, ist zu sagen, dass die gemessenen Feinstaubgehälter in Relation zum Grenzwert stimmen können.

4.2.7 Ozon-Sensor

Es wird ein Ozon-Sensor der Firma Winsen-Sensor mit der Typbezeichnung MQ-131 verwendet. Dieser ist auf einem Modulboard, Namens Wei Sheng drauf montiert. Der Ozongehalt in der Luft wird anhand dem Widerstandsverhältnis des Sensors überprüft. Das heißt, wenn die Sensor-Leitfähigkeit geringer wird, desto größer ist die Gas Konzentration. Der MQ-131 hat eine hohe Empfindlichkeit für Ozon, ist aber auch anderen Oxiden, wie elementares Chlor (Cl2) sehr empfindlich gegenüber.

Erfassungsbereich

Der Erfassungsbereich des Sensors liegt zwischen 10 und 1000ppb (ppb = Parts per Billion). Umgerechnet sind das ungefähr 20-2000ug/m^3 (Abbildung 29).

O_3	1 µg/m^3 = 0,50115 ppb	1 ppb =1,9954 µg/m^3

Abbildung 29: Umrechnung ppb in ug/m^3 laut dem Umweltbundesamt [14]

Wie in Kapitel 5.5 beschrieben, liegen die Umweltbundesamt maximal, -Schwellen und -Alarmwerte zwischen 120 bis 240ug/m^3. Diese Werte gelten aber nur Büro und den Innenraum, da noch kein Arbeitsplatzgrenzwert (AGW) festgelegt wurde. Daher ist zu sagen, dass der hohe Ozon-Erfassungsbereich des Sensors nicht vollständig ausgenutzt

wird, denn nur rund 6% des Bereichs sind realistische Werte im Alltag. Dennoch kann der Sensor in der Industrie eingesetzt werden, da sein Messbereich sich sehr weit erstreckt.

Technische Details des Ozonsensors (MQ-131)

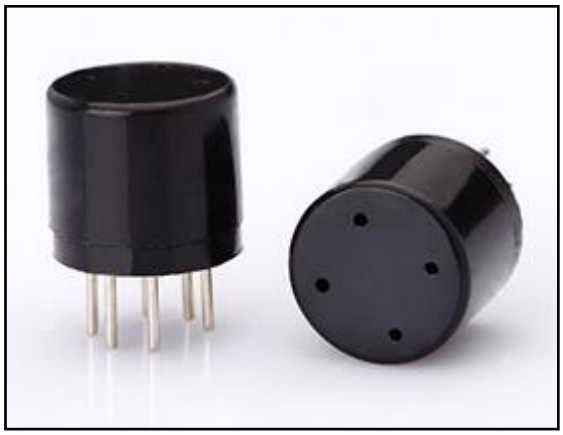


Abbildung 31: MQ131-Sensor von Winsen-Sensor [24]

Abbildung 30: Modulplatine mit aufgesetztem Sensor [25]

Wie schon in 4.2.7 erklärt sieht man in den oberen zwei Abbildungen (31 und 30) denn Sensor und die Modulplatine, auf die der Sensor drauf platziert ist. Im linken Bild ist der Sensor an sich zu sehen. Er hat 6 Stifte (Pins), vier davon werden zum Abrufen von Signalen verwendet, während die anderen beiden für die Bereitstellung der Spannung zuständig sind.

Die Modulplatine:

- besitzt einen TTL Ausgang
- kann an die Microcontroller IO-Ports (Input-Output) angesteckt werden
- Besitzt 5V Betriebsspannung

4.2.8 Pin-Belegung des Ozonsensors (MQ-131)

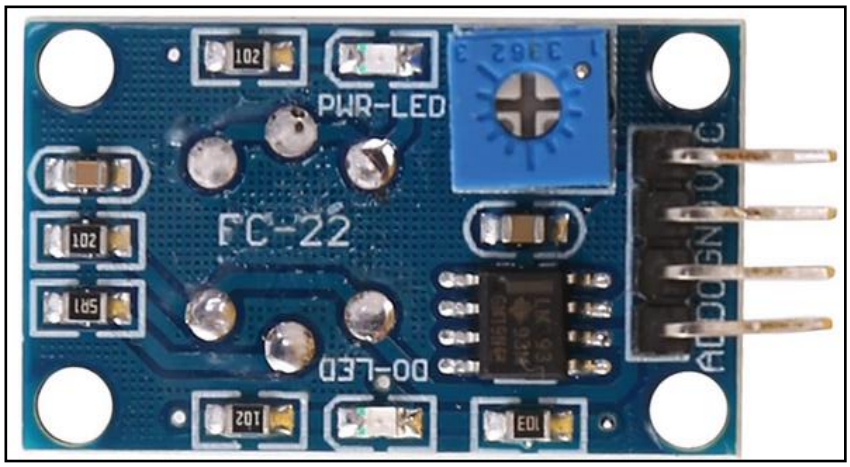


Abbildung 32: Pin-Belegung des Sensors [15]

	Pin-Bezeichnung	Beschreibung
	Vcc	5V Versorgungsspannung
	Digital out	Digitaler Ausgangspin
	Analog out	Analoger Ausgangspin. Spannung basierend auf der Konzentration des Gases
	Ground	Gemeinsame Masse des Moduls

Abbildung 33: Beschreibung der Pins

4.2.9 Kommunikationsaufbau mit Microcontroller

Der Sensor gibt analoge-Signale basierend auf der Ozonkonzentration aus. Das Raspberry-Pi kann nur digitale-Signale einlesen, daher muss man die analogen-Signale auf digitale umwandeln. Diesen Zweck erfüllt der MCP3008, welcher ein häufig verwendeter ADC für den Raspberry-Pi ist. Das heißt der Ozonsensor wird durch den Microcontroller versorgt und sein analoger-Ausgang geht in den ADC. Die Pins des ADC gehen dann wie in Abbildung 34 über SPI zum Microcontroller. Die Pinbelegung des MCP3008 wird in Punkt 4.2.10 Genauer beschrieben.

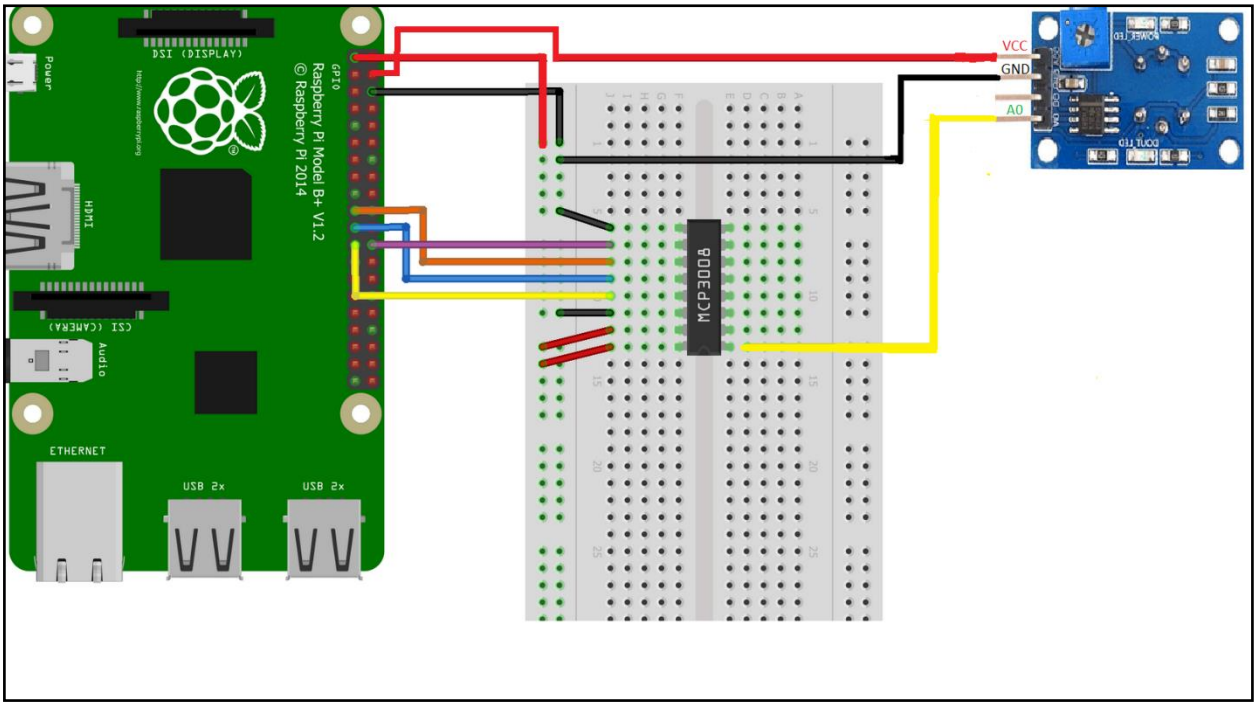


Abbildung 34: Kommunikation des Ozonsensor zu Raspberry-Pi

4.2.10 Pinbelegung des MCP3008

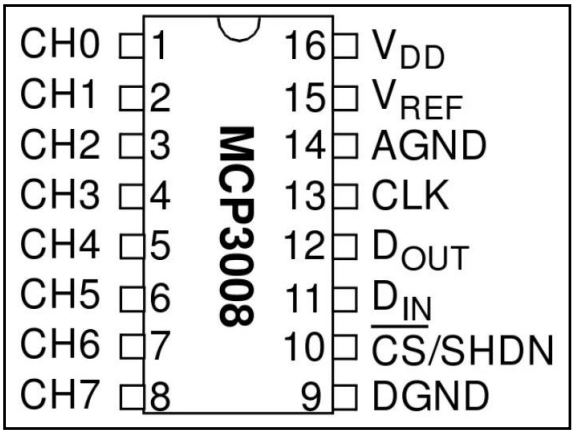


Abbildung 35: PIn Belegung des MCP3088 [16]

Der MCP3008, AD-Wandler, von Microchip verfügt über 8 Analoge-Eingänge (CH0-CH7) und wandelt mit 10 Bit Genauigkeit um. Das heißt, es werden Werte von 0-1023 (2^{10}) über die SPI-Schnittstelle übertragen. Seine Versorgungsspannung liegt bei 2.7 bis 5V.

Die Pins 10 bis 13 sind für die SPI-Schnittstelle zuständig:

MCP3008-Pins	Raspberry-Pi Pins	Bezeichnung
13	23	CLK
12	21	MISO
11	19	MOSI
10	24	CS

4.2.11 SPI-Schnittstelle

SPI steht für *Serial Peripheral Interface*, welches ein serielles Interface mit 4 Leitungen darstellt. SPI ist voll-duplexfähig. Die Daten können ohne Unterbrechung, hintereinander gesendet werden. Es gibt nur einen Master geben, dafür ist die Anzahl der Slaves nur auf die GPIO-Pins vom Microcontroller begrenzt.

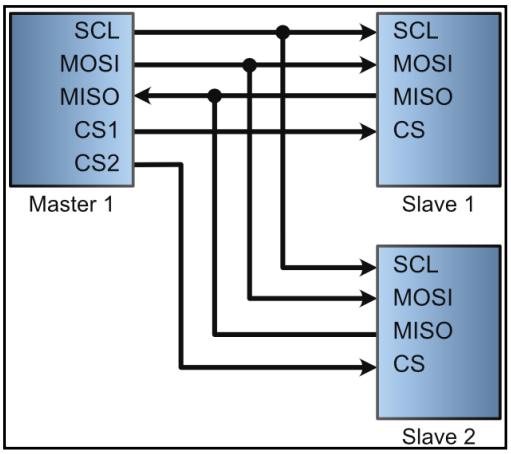


Abbildung 36: Beispiel einer SPI Kommunikation [17]

In Abbildung 36 ist eine Verdrahtung zwischen einem Master und zwei Slaves zu sehen. Man sieht, dass ein Master mehrere Slaves haben kann, dafür aber auch mehrere Chip Select Leitungen (CS) benötigt. SCL ist die Serial Clock Leitung, welche vom Master bedient wird.

MOSI (Master Output, Slave Input) und MISO (Master Input, Slave Output):
Wie der Name MOSI besagt, schickt der Master Daten an den Slave und bei MISO schickt der Slave Daten an den Master. Die Daten werden Bit für Bit gesendet. Die Daten des Masters werden meist mit MSB (most significant bit) first gesendet oder LSB (least significant bit) first.

CS (Chip select): Die CS-Leitung, auch SS-Leitung genannt, wird vom Master (Microcontroller) auf LOW gezogen, um mit dem entsprechenden Slaven zu kommunizieren.

Die SPI-Schnittstelle besitzt auch verschiedene Operation Modes. Denn die Kommunikation zwischen Master und Slave muss bestimmte Tuningbedingungen erfüllen. Diese Bedingungen nimmt das Protokoll zur Synchronisation her. Dies wird durch die Phase (CPHA) und der Polarität (CPOL) des Clock Signals bestimmt.

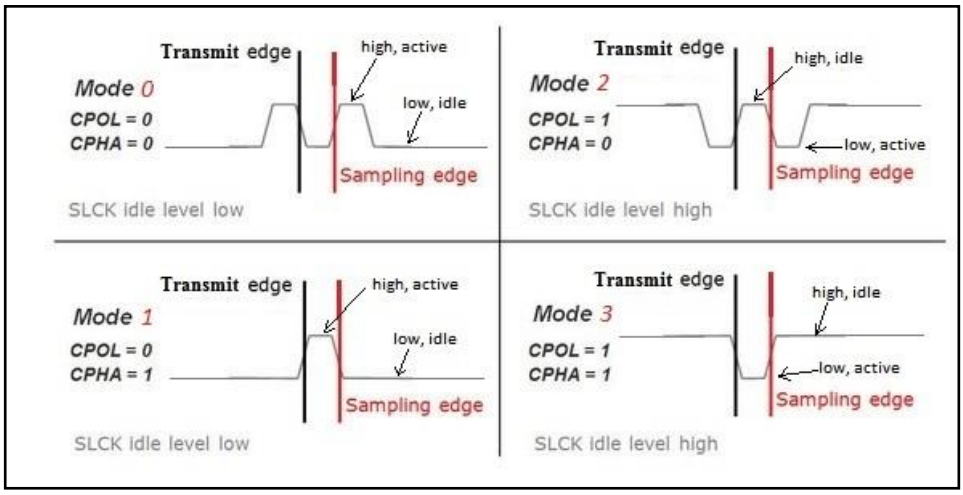


Abbildung 39: SPI-Operation Modes [19]

In Abbildung 39 sind die 4 verschiedenen Operation Modes zu sehen, welche bei einer SPI Übertragung auftreten können. CPOL ist im Leerlauf Modus (idle) entweder High oder Low und CPHA legt fest, ob die Daten am Anfang von der Clock (CPHA = 0) gelesen werden oder am Ende der Phase (CPHA = 1).

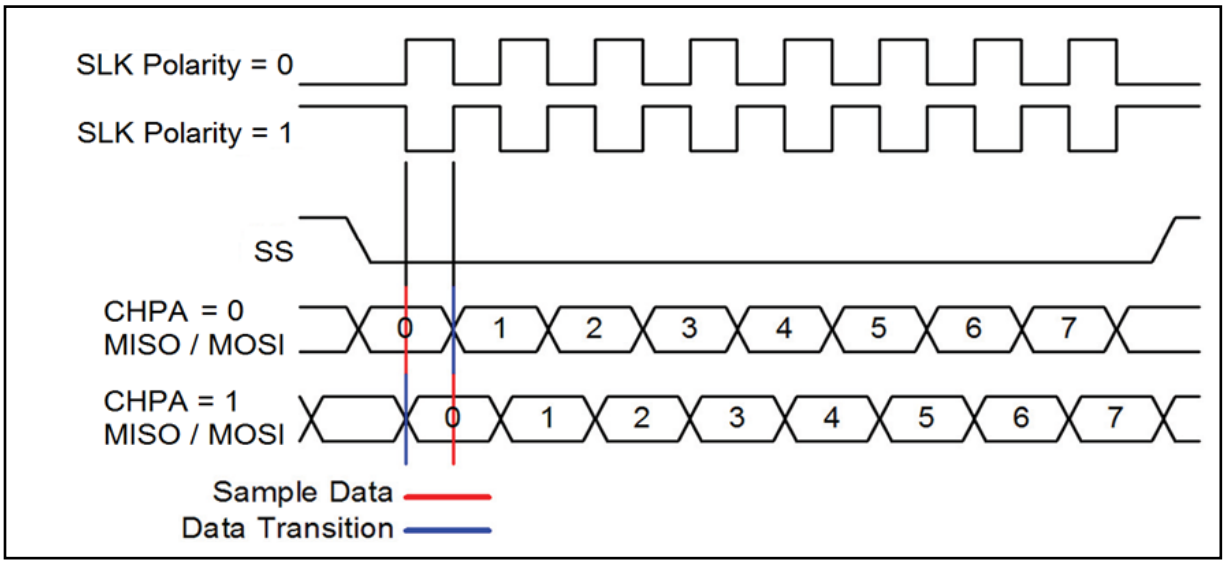


Abbildung 40: SPI-Datenübertragung [18]

Eine SPI Datenübertragung kann dann wie in Abbildung 40 ausschauen. Das heißt, der Slave wird über die CS/SS aktiviert, CPOL wird auf 1 oder 0 gesetzt und die Daten werden, je nach gewählter CPHA, gesendet.

4.2.12 Python-Code

In Abbildung 41 wird gezeigt, wie die Übertragungskette des Ozonsensors zum Raspberry-Pi aussieht.

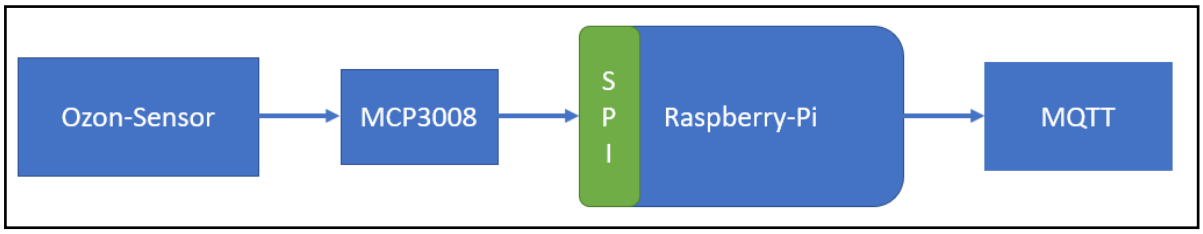
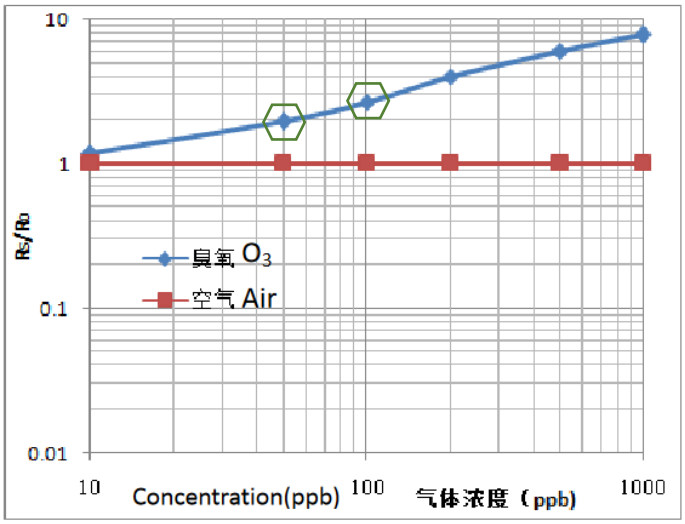


Abbildung 41: Übertragungskette des Ozonsensors zu MQTT

Die Python-Code Grundlage basiert auf einer Libary des MQ2 Sensors. Der Code muss so verändert werden, dass die Parameter und Formel mit dem MQ131 übereinstimmen.

```
1  # Adaptiert von sandboxelectronics für MQ2
2
3  import time
4  import math
5  from MCP3008 import MCP3008
6
7  class MQ():
8
9
10     MQ_PIN                       = 0        # Input Channel des MCP3008
11     RL_VALUE                     = 1000     # Lastwiderstand auf dem Board in kohm
12     RO_CLEAN_AIR_FACTOR          = 1        # RO_CLEAR_AIR_FACTOR=(Sensor resistance in clean air)/RO,
13                                             # Aus dem Datenblatt
14
15     CALIBARAION_SAMPLE_TIMES     = 50       # Wie viele Samples in der Kalibrationsphase genommen werden
16     CALIBRATION_SAMPLE_INTERVAL  = 500      # Abtastzeit in ms zwischen jedem Sample (Kalibrationsphase)
17
18     READ_SAMPLE_TIMES            = 5        # Wie viele Sample Normal genommen werden
19     READ_SAMPLE_INTERVAL         = 50       # Abtastzeit in ms zwischen jedem Sample
20
21     GAS_OZONE                    = 0
22
23
24     def __init__(self, Ro=10, analogPin=0):
25         self.Ro = Ro
26         self.MQ_PIN = analogPin
27         self.adc = MCP3008()
28
29
30         # zwei Punkte werden von der Kurve genommen.
31         # Mit diesen beiden Punkten wird eine Linie gebildet, die "ungefähr äquivalent" ist
32         # der ursprünglichen Kurve entspricht.
33         # [x, y, Steigung]
34         #
35
36         self.OZONCurve = [-1.301,0.301,0.46]
37
38
39         print("Calibrating...")
40         self.Ro = self.MQCalibration(self.MQ_PIN)
41         print("Calibration is done...\n")
42         print("Ro=%f kohm" % self.Ro)
43
44
```

Abbildung 42: Libary-Code für MQ131 (Zeile 1-44)



Die Werte in Zeile 36 bekommt man durch die Berechnung mit dem Datenblatt.

Der Wert 0.301 ist der lg(2).

Der Wert 0.46 ist der lg(2.87)

Und die -1.301 ist die Steigung

```
45      def MQPercentage(self):
46          val = {}
47          read = self.MQRead(self.MQ_PIN)
48          val["GAS_OZON"]  = self.MQGetGasPercentage(read/self.Ro, self.GAS_OZON)
49          return val
50
51      # MQResistanceCalculation
52      # Der Sensor und der Lastwiderstand bilden einen Spannungsteiler,
53      # mit dem man den Widestand des Sensor herleiten kann
54      def MQResistanceCalculation(self, raw_adc):
55          return float(self.RL_VALUE*(1023.0-raw_adc)/float(raw_adc));
56
57
58      # MQCalibration
59      # Diese Funktion berechnet den Sensorwiderstand in sauberer Luft, mittels MQResistanceCalculation
60      # Sie Teilt den Wert durch R0_CLEAN_AIR_FACTOR, welcher vom Datenblatt gegeben ist.
61      def MQCalibration(self, mq_pin):
62          val = 0.0
63          for i in range(self.CALIBARAION_SAMPLE_TIMES):          # Mehrere Sample nehmen
64              val += self.MQResistanceCalculation(self.adc.read(mq_pin))
65              time.sleep(self.CALIBRATION_SAMPLE_INTERVAL/1000.0)
66
67          val = val/self.CALIBARAION_SAMPLE_TIMES                 # Durchschnittswert wird berechnet
68          val = val/self.RO_CLEAN_AIR_FACTOR                      # Geteilt durch R0 in sauberer (Normaler) Luft, ergibt sich R0
69                                                                  # laut Datenblatt
70
71          return val;
72
73      # MQRead
74      # Diese Funktion benutzt MQResistanceCalculation um den Sensor Widerstand Rs zu berechnen
75      # Rs ändert sich, wenn eine andere Gas-Konzentration vorhanden ist
76      def MQRead(self, mq_pin):
77          rs = 0.0
78
79          for i in range(self.READ_SAMPLE_TIMES):
80              rs += self.MQResistanceCalculation(self.adc.read(mq_pin))
81              time.sleep(self.READ_SAMPLE_INTERVAL/1000.0)
82
83          rs = rs/self.READ_SAMPLE_TIMES
84
85          return rs
```

Abbildung 43: Libary-Code des MQ131 (44-85)

```
86
87      # MQGetpercentage
88      # Hier wird auf die Steigung und einen Punkt auf der Ozon-Kurve (aus dem Datenblatt) geschaut
89      # Dadurch kann der Wert abgeleitet werden und dann durch math.pow(10... der Wert endlogarithmiert
90
91      def MQGetPercentage(self, rs_ro_ratio, pcurve):
92          return (math.pow(10,( ((math.log(rs_ro_ratio)-pcurve[1])/ pcurve[2]) + pcurve[0])))
93
94
95      # MQGetGasPercentage
96      # Diese Funktion gibt die Ozon Kurve an MQGetGasPercentage und berechnet den ppm Wert des Gases,
97      # welcher in Main dann auf ug/m^3 umgerechnet wird
98      def MQGetGasPercentage(self, rs_ro_ratio, gas_id):
99          if ( gas_id == self.OZON_LPG ):
100             return self.MQGetPercentage(rs_ro_ratio, self.OZONCurve)
101         return 0
```

Abbildung 44: Libary-Code des Ozon-Sensors (86-101)

Die Abbildungen 42-44 erklären, wie der Ozon-Sensor zu seinen Werten kommt und diese berechnet. Die einzelnen Funktionen sind durch Kommentare beschrieben für ein besseres Verständnis des Codes.

```python
from mq import *
import sys, time

try:
    print("Press CTRL+C to abort.")

    mq = MQ();
    while True:
        perc = mq.MQPercentage()
        sys.stdout.write("\r")
        sys.stdout.write("\033[K")
        sys.stdout.write("OZON: %g ug/m^3" % (2000*perc["GAS_OZON"]))  # Wird mit 2000 Multipliziert,
                                                                        # um auf ug/m^3 zu kommen
        sys.stdout.flush()
        time.sleep(0.1)

except:
    print("\nAbort by user")
```

Abbildung 45: Python Loop des Ozon-Sensors

Abbildung 45 zeigt die Loop des Python Programms, welche beim Starten den Ozongehalt ausgibt.

```
Press CTRL+C to abort.
Calibrating...
Calibration is done...

R0= 18333,40 kohm
OZON: 33.05 ug/m^3
```

Abbildung 46: Ausgabe der Konsole

Die Ausgabe sieht dann wie in Abbildung 46 aus. Beim einstündigen Testen des Sensors sind die Ozon-Werte zwischen 1ug/m^3 und 80ug/m^3 gelegen. Ob diese Werte stimmen, ist nicht genau zu sagen, da es kein zweites Messgerät zum Überprüfen gibt. Jedoch liegen diese Werte unter den Ozongrenzwerten, wie in den physiologischen Grundlagen beschrieben. Daher könnte man schließen, dass die Werte nicht unbedingt falsch sind und in Betracht gezogen werden können.

4.2.13 Stickstoffdioxid-Sensor

Für die Stickstoffdioxid Messung wird ein Stickstoffdioxid-Sensor Clickboard der Firma Mikroe verwendet. Das Board verfügt über einen MICS2714-Sensor (Stickstoffdioxid-Sensor) der Firma SGX Sensortech. Dieser Sensor besteht aus einer mikrobearbeiteten Metalloxid-Halbleiter-Membran mit einem integrierten Heizwiderstand. Der Widerstand erzeugt Wärme, die die Reaktion katalysiert, die wiederum den elektrischen Widerstand der Oxidschicht selbst beeinflusst. Die Temperatur des Heizwiderstands ist recht hoch: Sie liegt im Bereich von 350 °C bis 550 °C. Nach der anfänglichen Vorwärmphase kann der Sensor Gasänderungen in Zeitintervallen von weniger als zwei Sekunden feststellen.

Der Sensor MiCS-2714 hat vier Anschlüsse. Zwei Stifte sind die Anschlüsse des internen Heizelements, während die anderen beiden Stifte die Anschlüsse des MOS-Sensors sind. Die Anwendung beschränkt sich auf die Berechnung eines geeigneten Widerstandes für den Spannungsteiler. Der mittlere Abgriff zwischen dem Sensor (als Widerstand) und dem Festwiderstand wird zur Bereitstellung einer Ausgangsspannung verwendet. Sie hängt direkt vom Widerstand des Sensors ab, sodass sie als Eingang für den MCP3201, einen stromsparenden 12-Bit-A/D-Wandler von Microchip, verwendet werden kann. Mit diesem ADC kann die Ausgangsspannung in ein digitales Signal umgewandelt werden.

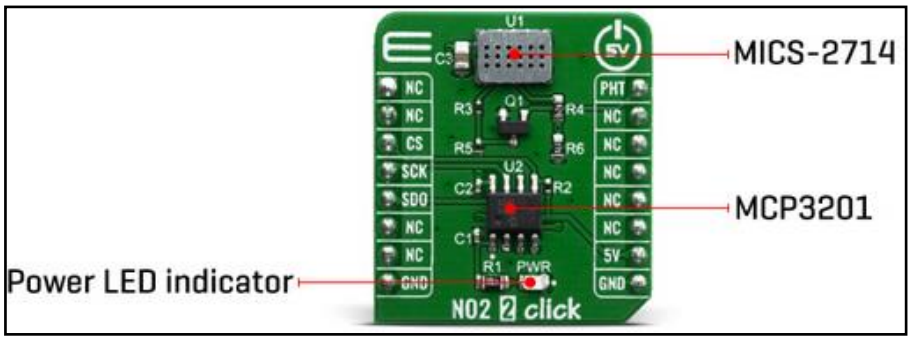


Abbildung 47: NO2-Clickboard [20]

Spezifikationen

Dieses Click Board ist nur für den Betrieb mit 5 V Logikpegel ausgelegt.

Bezeichnung	Min	Typ	Max	Einheit
Messbereich	0	-	10	ppm
Response Time	-	200	-	s
Betriebstemperatur	-30	-	85	˚C
Betriebsfeuchtigkeit	5	-	95	% RH

Abbildung 48: NO2-Spezifikationen

Pin-out des Stickstoffdioxidsensors

Notes	Pin	mikro BUS				Pin	Notes
	NC	1	AN	PWM	16	**PHT**	Preheating
	NC	2	RST	INT	15	NC	
Chip Select	**CS**	3	CS	RX	14	NC	
SPI Clock	**SCK**	4	SCK	TX	13	NC	
SPI Data Out	**SDO**	5	MISO	SCL	12	NC	
	NC	6	MOSI	SDA	11	NC	
	NC	7	3.3V	5V	10	**5V**	Power Supply
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Abbildung 49: Pinout des NO2-Sensors von Mikroe [21]

Wie in Abbildung 49 zu sehen ist, besitzt der NO2-Sensor ein SPI-Interface. Die SPI-Schnittstelle wurde schon in beim Ozonsensor genauer behandelt und wird daher nicht weiters behandelt.

4.2.14 Stickstoffdioxid-Sensor mit Arduino-Uno

Beim Beginn des Projekts wurde mit einem Arduino Uno gearbeitet und daher auch ein C-Programm geschrieben. Für das C-Programm wurde eine Libary von DF-Robots verwendet. Diese Libary ist für MICS-Gassensoren geschrieben. Der Sensor wurde mittels SPI and denn Arduino verbunden.

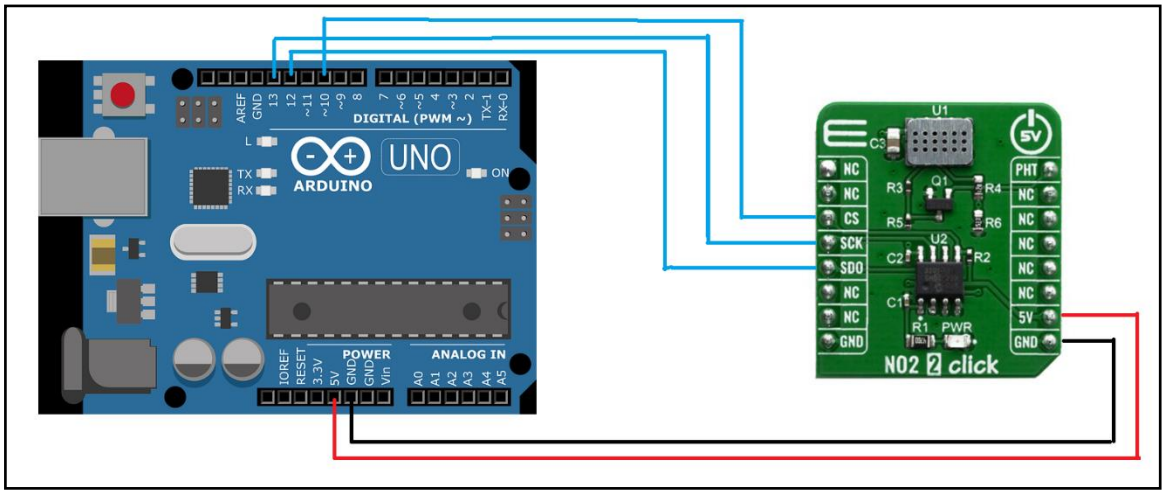


Abbildung 50: SPI-Kommunikation Sensor zu Microcontroller

```
#define SCLK 13
#define CS 10
#define DATA 12
#define CALIBRATION_TIME 3                    // Kalibrationszeit/Aufwärmzeit = 3min
DFRobot_MICS_ADC mics(DATA, SCLK);              // MICS_ADC library von dfrobots

void setup()
{
  Serial.begin(9600);
  while(!Serial);
  while(!mics.begin()){
    Serial.println("Kein NO2-Sensor erkannt");
    delay(1000);
  } Serial.println("Sensor erfolgreich angeschlossen");

// Sensor starten
  uint8_t mode = mics.getPowerState();
  if(mode == SLEEP_MODE){
    mics.wakeUpMode();
    Serial.println("Sensor erfolgreich gestartet");
  }else{
    Serial.println("Sensor ist noch im SAMUEL_MODE"); //
  }
  while(!mics.warmUpTime(CALIBRATION_TIME)){ // Aufwärmen des MICS-2714 Sensors
    Serial.println("Aufwärm Zeit");
    delay(1000);
  }
}

//======Daten bekommen=======
void loop()
{
  //Stickstoffdioxid (NO2)     (0.1  - 10)PPM
  float gasdata = mics.getGasData(NO2);
  Serial.print("NO2: ");
  Serial.print(gasdata,1);
  Serial.println("PPM");
  delay(1000);
  //mics.sleepMode();
}
```

Abbildung 51: Arduino-Code

Wenn das Programm gestartet wird, muss der Sensor erstmal drei Minuten kalibrieren, um auf Betriebstemperatur zu gelangen. Wenn die Kalibrierungsphase zu Ende ist, gibt der Sensor die Stickstoffdioxid Werte in ppm aus. Die ausgegebenen Werte lagen im Bereich von 0.1ppm = 100ppb.

NO_2	1 ppb = 1.88 µg/m^3

Abbildung 52: Siickstoffdioxid von ppb auf ug/m^3 [14]

Die gemessenen Werte lagen im Bereich von 0.1ppm (0.1*1000*1.88=188ug/m^3. Wie in Punkt 4.1 beschrieben, liegen die Kurzzeitwerte zwischen 80-250ug/m^3. Wie schon bei den anderen Sensoren beschrieben, kann nicht gesagt werden, ob die Werte stimmen. Da die Werte unter den Grenzwerten liegen. Es kann aber interpretiert werden, dass sie eine gewisse Richtigkeit besitzen.

4.2.15 Umstellung Arduino-Uno zu Raspberry-Pi

Damit das Projekt für die Online-Visualisierung kompatibel wird, ist das Projekt auf dem Raspberry-Pi umgestellt worden. Demnach musste auch der Stickstoffdioxid-Sensor auf Raspberry-Pi umgestellt werden. Der Sensor wird, wie bei Arduino-Uno, auch mit dem SPI-Interface verbunden.

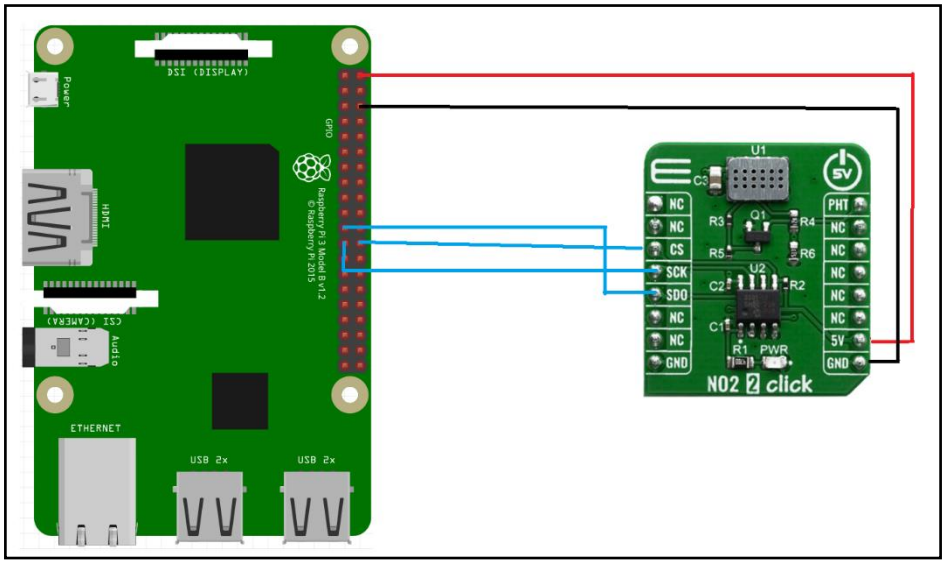


Abbildung 53: Stickstoffdioxidsensor mit Raspberry-Pi

Für den Python-Code wurde auch eine DFRobots-MICS Libary verwendet. Diese Libary funktioniert auch für andere Sensoren beziehungsweise andere Gase, wie Methan, Propan oder Ammonium. Beim Verwenden mit dieser Libary ist eine wichtige Sache übersehen worden. Diese Libary funktioniert nur für das I2C-Übertragungsprotokoll und wurde erst bei der Fehleruntersuchung entdeckt. Nach genauerer Analyse der Libary wurde trotzdem die verwendete Formel für die Gas-Umrechnung verwendet. Ein neuer Code wurde daher geschrieben. Dieser Code erhält vom Sensor 4 Werte. Diese Werte werden im Datenblatt des Stickstoffdioxid nicht genauer beschrieben und können daher auch nicht genauer interpretiert werden. Um die Werte in eine Relation zu setzten, wurden diese in die Formel der DFRobot-Libary gesetzt. Die Werte wurden in die Formel des DF_Robots gegeben, um zu sehen, ob diese Werte auch eine Aussagekräftigkeit haben. Jedoch konnte nichts mit dem Datenblatt in Relations gesetzt werden.

```python
import time
import spidev

bus = 0 # SPI bus 0
device = 0 # CS0 wird verwendet

spi = spidev.SpiDev() # Enable SPI

# Open a connection to a specific bus and device (chip select pin)
spi.open(bus, device)

# Set SPI speed and mode
spi.max_speed_hz = 500000
spi.mode = 0

receive=spi.readbytes(4)

while True:
    time.sleep(2)
    try:

        print(receive) # Es werden 4 Werte ausgegeben

    except RuntimeError:
        print("Unable to read from sensor, retrying...")
        continue



    # Um den 1 Wert der Ausgabe zu bekommen
    x = str(receive)
    y=x.split("[")
    z=y[1]
    a=z.split(",")
    b=a[0]

    print(b)
    data=int(b)
    no_wert= (data - 0.045) / 6.13  # Formel aus der DF-Robots Libary
                                    # Diese Formel wurde benutzt, um den Wert
                                    # In ppm umzurechnen
    print(no_wert)

    time.sleep(2)
```

Abbildung 54: Python-Code nach der Entdeckung der Falschen Libary

4.2.16 Kommunikation LCD-Display zu Microcontroller

Zur Veranschaulichung der Sensordaten wird ein 4-Zeiliges LCD-Display verwendet. Das LCD-Display verfügt über einen I2C-Adapter. Durch den Adapter kann das Display einfach mittels I2C an den Microcontroller verbunden werden.

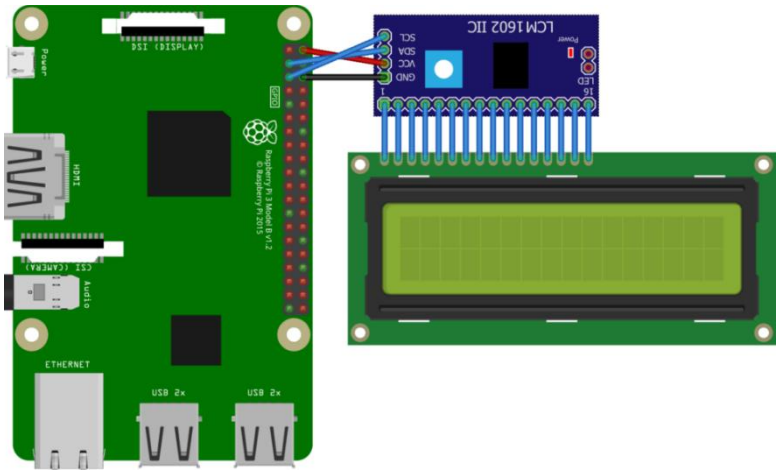


Abbildung 55: LCD-DIsplay zu Mikrocontroller [22]

4.2.17 I2C-Schnittstelle

Bei I2C handelt es ich um einen Datenbus. Es ist ein Verfahren, wie zwischen verschiedenen Teilnehmern kommuniziert wird. Beispielsweise, wie ein Microcontroller mit Sensoren oder Displays kommuniziert. I2C braucht nur zwei Leitungen und kann bis zu 128 Teilnehmern kommunizieren. Die Leitungen bestehen aus SDA (serial data line) über diese die Daten gesendet werden und SCL (serial clock line), die denn Takt vorgibt. Der Unterschied zu UART ist, dass eine synchrone Kommunikation erfolgt. Die Übertragung wird seriell gesendet. Das heißt, die Daten werden also hintereinander über die SDA-Leitung geschickt. Der I2C-Bus definiert auch den Ablauf der Übertragung.

Es können mehrere Master und Slaves auf einem Bus gleichzeitig betrieben werden. Wenn mehrere Master den Bus verwenden, dann darf ein Master, bei einer bestehenden Kommunikation nicht eingreifen. Die Master müssen dann als Multimaster definiert werden, um keine Kommunikation zu stören. Der Master darf nur dann ein Signal senden, wenn die SDA-Leitung High ist.

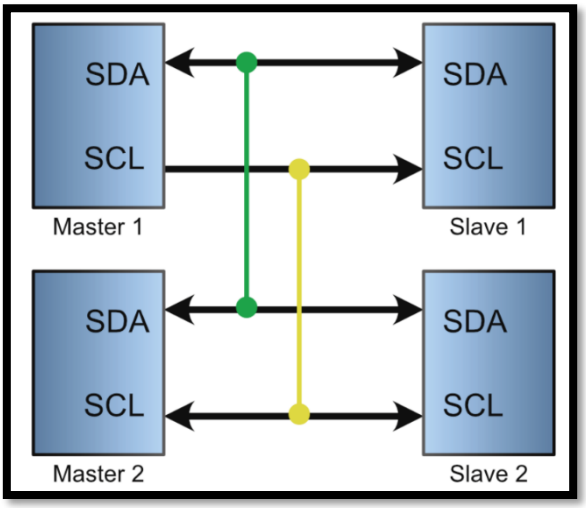


Abbildung 56: I2C-Kommunkation [23]

Das Clock Signal synchronisiert die Ausgabe von Datenbits vom Master mit dem Abtasten von Bits durch den Slave. Um sicherzustellen, dass das Datensignal genug Zeit hat den Pegel zu wechseln, werden die Daten genau in der Mitte von jedem Clock High abgetastet.

I2C besitzt auch ein Start und Stoppbit. Beim Startbit wird das Signal von High auf Low gesetzt, bevor das Clock Signal von High auf Low geht. Beim Stoppbit passiert das genau andersherum, also von Low auf High. Nach dem Startbit sendet der Master die Adresse vom Slave, mit dem er kommunizieren möchte. Die Adresse kann 7 oder 10 Bit groß sein. Bei einer 7 Bit Adresse sendet der Master als 8tes Bit 1 oder 0 zum Slave. Er sagt dem Slave, ob er lesen oder schreiben will (read/write). Beim Schreiben vom Slave wird RW (read/write) auf 0 gesetzt und beim Lesen auf 1.

Nach jedem Byte Daten wird vom Empfänger eine Bestätigungsbit (ACK = Acknowledge) gesendet. Wenn die Übertragung in Ordnung ist, dann ist das Bit Low (ACK). Ist ein Fehler aufgetreten, dann sendet der Slave ein High Signal (NACK = Not Acknowledge).

Nachdem der Master das ACK-Bit vom Slave erhalten hat, kann die eigentliche Datenübertragung erfolgen. Ein Block von Daten ist immer 8bit (1 Byte) groß. Nach jedem Byte, das gesendet wird, bestätigt der Slave mit einem ACK-Bit, dass er die Daten erhalten hat.

4.2.18 Display Implementierung in Python

```
import RPi.GPIO as GPIO
import I2C_LCD_driver    # LCD-I2C Libary

from time import *

from example, pm25_verbessert_v4 import *  # Impotieren der Sensor Dateien

mylcd = I2C_LCD_driver.lcd()

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.cleanup()

while True:

  pm25 = pm25_wert.read()
  ozon = perc.read()

  if result.is_valid():  # Ausgabe auf dem Display
    mylcd.lcd_display_string("Feinstaub: %g ug/m^3" % pm25, 1)
    mylcd.lcd_display_string("Ozon: %g ug/m^3" % ozon, 2)
```

Abbildung 57: Python Implementierung des LCD-Displays

Durch die Verwendung eines I2C-Adapters ist die Implementation in Python, um einiges einfacher. Denn ohne I2C-Adapter müssen die einzelnen GPIO Pins des Displays-angesteuert werden und mit dem Adapter kann eine Libary importiert werde. Dadurch spart man sich eine Menge an Code-Zeilen.

4.3 Lüftersystem

Übersicht

Das Lüftersystem besteht aus drei Hardwareelementen. Dem Lüfter selbst, einem Potentiometer zur Veränderung der Geschwindigkeit und einem Schalter, mit dem man den Lüfter von der Spannungsversorgung trennen kann. Weiteres besteht das Lüftersystem aus zwei Softwareelementen. Erstens der PWM-Steuerung, die die eingestellte Referenzspannung des Potentiometers auswertet, mit SPI zum Raspberry Pi überträgt, und anschließend über einen PWM Pin den Lüfter mit entsprechendem DutyCycle (DC) mit Spannung versorgt. Das zweite Softwareelement ist der Code zum Auslesen des Tachometer-Signals, mit dem man die Umdrehungen pro Minute errechnen kann.

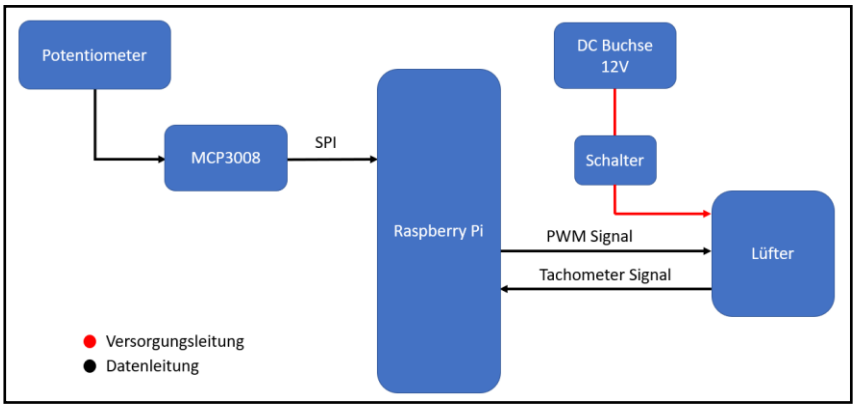


Abbildung 58: Stromlaufplan des Lüftersystem

4.3.1 Einstellung der Lüftergeschwindigkeit

Für die Einstellung der Lüftergeschwindigkeit wurde nach einer einfachen und intuitiven Lösung gesucht. Dabei kamen drei verschiedene Methoden infrage:

1. Druckschalter
2. Schieberegler
3. Potentiometer

Die Nachteile des Druckschalters sind, dass man pro Klick die Umdrehungen pro Minute immer nur um zum Beispiel um 1, 10 oder 100 erhöhen kann. Leider muss die Schrittweite pro Klick bereits im Vorhinein im Code bestimmt werden und der User kann sich die Genauigkeit nicht selbst aussuchen. Weiteres ist auch das Senken der Lüftergeschwindigkeit ein Problem.

Ein Schieberegler ist eine bessere Lösung, da man als User die Schrittweite leichter einstellen kann. Das Rauf- und Runterdrehen der Geschwindigkeit ist im Gegensatz zum Druckschalter auch kein Problem.

Schlussendlich fiel die Entscheidung auf einen Potentiometer, da er die Vorteile eines Schiebereglers besitzt und intuitiver ist, da man Drehknöpfe bereits zur Einstellung der Lautstärke von Radios kennt.

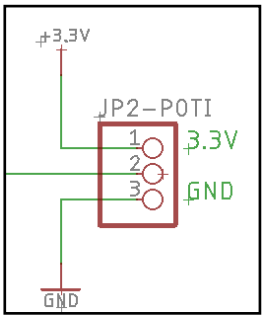


Abbildung 59 Beschaltung des Potentiometers

Der Potentiometer wird statt VCC = 5V mit VCC = 3.3V versorgt. Diese Spannung wurde gewählt, um den Raspberry Pi nicht zu überlasten. Dieser verträgt maximal 3.3V auf den GPIO Pins. Doch hier ergibt sich das nächste Problem: Der Ausgang des Potentiometers liefert analoge Spannungswerte und der Raspberry Pi kann nur digitale Signale verarbeiten.

Dieses Problem wird mit dem IC Baustein MCP3008 gelöst. Dieser wandelt analoge Spannungswerte in digitale um.

4.3.2 ADC Baustein

In Abbildung 60 ist die Pinbelegung des ADC Bausteins „MCP3008“ zu sehen. Durch die Verwendung des ADC-Bausteins kann das Raspberry Pi problemlos die digitalen Spannungswerte des Potentiometers verarbeiten.

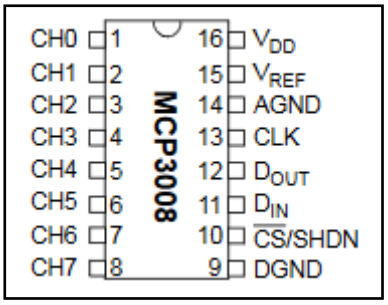


Abbildung 60: Pinbelegung des MCP3008

Der Analogausgang der Referenz wird an den Pin 1 (Channel 0) des ADC angeschlossen. Die Pins 9–16 des ADC Bausteins werden wie in Abbildung 61 zu sehen mit den entsprechenden Raspberry Pi Pins verbunden. Weitere Informationen zu dem IC Baustein MCP3008 wurden in dem Kapitel 4.2.10 beschrieben.

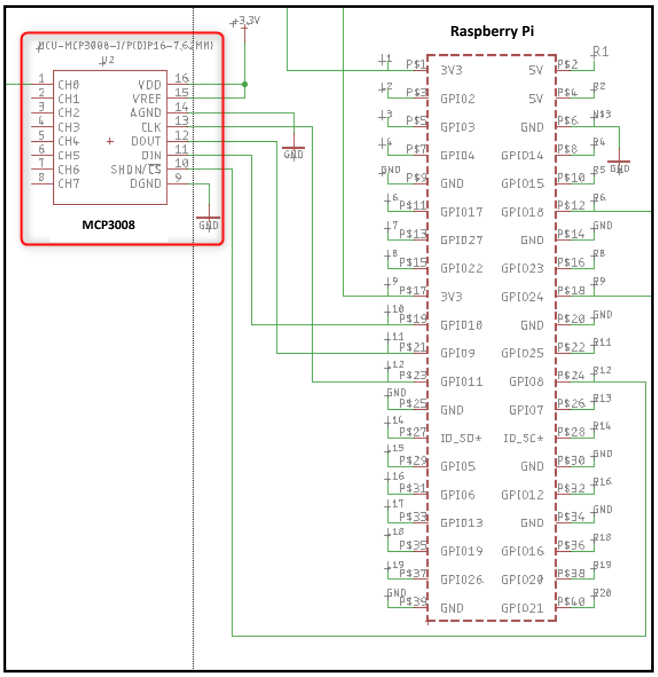


Abbildung 61: Stromlaufplan des ADC Bausteins MCP3008 und Raspberry Pi

Der Python Code für das Auslesen des Potentiometers ist sehr simpel, da Python den Vorteil von includierbaren librarys bringt. Die library „gpiozero“ hat für den ADC Baustein eine bereits fertige library, die nur inkludiert werden muss.

```
from gpiozero import MCP3008

pot = MCP3008(0)

while True:
    print(pot.value)
```

Abbildung 62: Inkludieren der „gpiozero“ library und Ausgabe des Potentiometer Wertes

4.3.3 PWM

Der PWM (Pulsweitenmodulation) Code ist dafür zuständig, dass er die Einstellung des Potentiometers in Prozent umrechnet und anschließend über den PWM Pin den Lüfter entsprechend steuert.

Es muss die library RPi.GPIO importiert werden. Außerdem werden der PWM-Pin, die Wartezeit und die PWM-Frequenz festgelegt. In der Funktion „setFanSpeed“ wird der Prozentsatz, welcher in der variable PWM gespeichert wird, an den PWM-Pin übergeben. Die Funktion setFanSpeed benötigt Zahlen zwischen 0 und 100. In der Schleife „while True“ wird der Potentiometerwert (pot.value) mit 100 multipliziert, da dieser nur von 0 bis 1 geht.

```python
from gpiozero import MCP3008
import RPi.GPIO as GPIO
import time

# Configuration
FAN_PIN = 18            # Lüfter Pin für PWM
WAIT_TIME = 0.1         # [s] Time to wait between each refresh
PWM_FREQ = 25           # [kHz]

# Lüfter an und aus
FAN_HIGH = 100
FAN_OFF = 0

# Set fan speed
def setFanSpeed(speed):
    fan.start(speed)
    return()

try:
    # Setup des GPIO pin
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(FAN_PIN, GPIO.OUT, initial=GPIO.LOW)

    fan = GPIO.PWM(FAN_PIN,PWM_FREQ)
    setFanSpeed(FAN_OFF)

    while True:
        pot = MCP3008(0)
        pwm = pot.value * 100
        setFanSpeed(pwm)
        print(pwm)
        time.sleep(WAIT_TIME)

except KeyboardInterrupt:
    setFanSpeed(FAN_HIGH)
```

Abbildung 63: Python Code (PWM)

4.3.4 Tacho Signal

Das Messen des Tachosignales ist von Vorteil, da man dadurch die Drehzahl des Lüfters überprüfen und visualisieren kann. Die Lüftergeschwindigkeit wird in Umdrehungen pro Minute umgerechnet und für den User angegeben.

Allgemein

Bei einem sogenannten Tachometersignal, auch Tacho genannt, handelt es sich um ein Signal, das von Lüftern ausgegeben wird. Die meisten Lüfter geben jede halbe Umdrehung einen elektrischen Impuls aus. Mithilfe dieses Impulses kann die Umdrehungsgeschwindigkeit des Lüfters gemessen und errechnet werden. Dadurch kann man sicherstellen, dass der Lüfter sich auch tatsächlich dreht und nicht defekt ist. Weiters kann man die Lüftergeschwindigkeit messen und dem User anzeigen. Da die Lüftergeschwindigkeit in Echtzeit angegeben wird, fallen abnormale Situationen sofort auf.

Generierung des Tacho-Signals

Lüfter besitzen einen Drehzahlsensor. Mit diesem Sensor wird, wie bereits in 4.3.4 erklärt, ein Tachometersignal erzeugt, mit dem man die Drehzahl und die Funktionalität des Lüfters überwachen kann. Laut dem Datenblatt wird ein Tachometersignal mit den folgenden Charakteristiken erzeugt:

All Noctua fans provide a tachometer output signal of the following characteristics:
- two cycles per revolution
- open collector output
- Maximum current is 5mA for 5V and 12V fans and 2mA for 24V fans, so for example, when using a 12V fan a resistor value of 2.7K Ohm or larger is suitable.

Abbildung 64: Datasheet Auszug: Generierung Tachometer Signal [7]

- Zwei Zyklen pro Umdrehung
- Open Collector Output
- Maximaler Strom ist 5mA für 5V und 12V Lüfter und 2mA für 24V Lüfter

Es wird ein 12V Lüfter verwendet, da dieser von der Schule bereitgestellt wurde. Somit sind die 12V und die 4 Pins des Lüfters (GND, VCC, Tacho und PWM) schon festgelegt.

In unserem Fall wird der Sensor des Lüfters zweimal pro Umdrehung ausgelöst. Um das Signal mit einem Oszilloskop bzw. mit dem Raspberry Pi auswerten zu können, muss eine positive Spannung an dem Pin des Lüfters anliegen. Beim Auslösen des Sensors wird die Spannung auf GND gezogen und somit eine fallende Flanke erzeugt. Wird der Sensor nicht ausgelöst, bleibt die Leitung dauerhaft auf „High“.

Da der Raspberry Pi nur Spannungen von maximal 3.3V verarbeiten kann wird die Leitung wird mit einem 10kOhm Pull-Up Widerstand auf 3.3V gehalten.

Signalaufbereitung des Tacho-Signals

An den Ausgang des Tacho Pins muss ein Pull-Up Widerstand (Abbildung 65) angeschlossen werden. Wenn der Lüfter sich dreht, wird ein Rechtecksignal erzeugt.

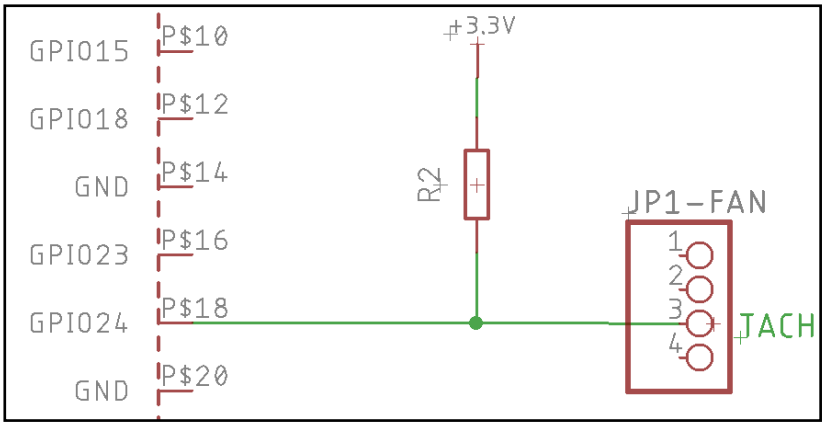


Abbildung 65: Pull-Up Widerstand an der Tachometerleitung

Dieses Rechtecksignal muss jetzt per Micro Controller erhalten und in Umdrehungen pro Minute umgerechnet werden.

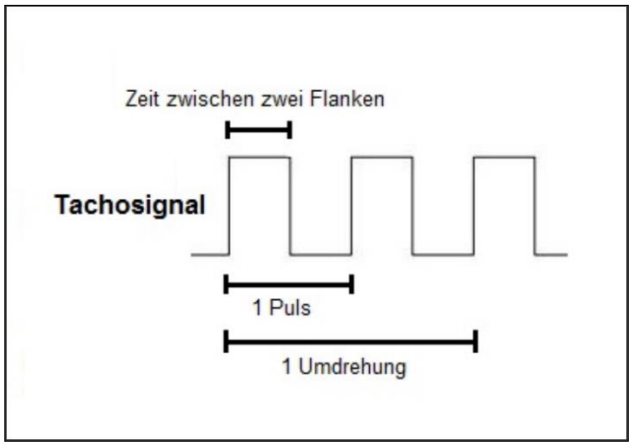


Abbildung 67: ideales Rechtecksignal mit Beschriftung (angepasst an Lüfter)

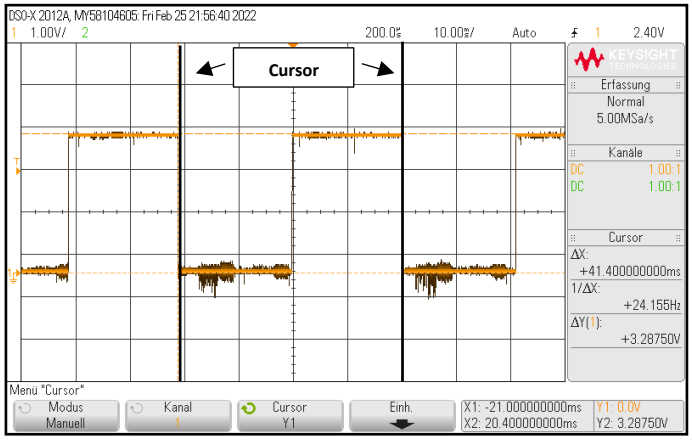


Abbildung 66: gemessenes Tachometersignal mit Zeitmessung zwischen zwei fallenden Flanken mithilfe der Cursor Funktion des Oszilloskops

Die Zeit zwischen zwei fallenden Flanken bzw. die Periodendauer des Signals entspricht einer halben Umdrehung des Lüfters.

T … Periodendauer

$f = \frac{1}{T}$ … Frequenz

PULSE … Pulse pro Umdrehung des Lüfters

rpm … Umdrehungen pro Minute

Es werden in Abbildung 66 für T = 44,1ms gemessen.

Beispielrechnung, um die Umdrehungen pro Minute aus der Periodendauer zu errechnen:

$$PULSE = 2\frac{Puls}{Umdrehung}$$

$$f = \frac{1}{T} = \frac{1}{41.4\frac{ms}{U}} = 24.15Hz$$

$$rpm = \frac{f}{PULSE/U} * 60 = \frac{48.31}{2} * 60 = 724{,}64\frac{\mathrm{U}}{\mathrm{m}}$$

Python Code zur Tacho Auswertung

Folgender Code erkennt die Fallenden am Tachometersignal und berechnet sich die Frequenz. Anschließend werden die Umdrehungen pro Minute errechnet. Als Erstes werden die librarys „RPi.GPIO“ und „time“ importiert.

```
2  import RPi.GPIO as GPIO
3  import time
```

Abbildung 68: Importieren der Rpi.GPIO und time library

Anschließend werden der Tachometer-Pin sowie die Anzahl der Pulse festgelegt.

```
7  # Pin configuration
8  TACH = 24        # Lüfter tachometer Pin Fan's tachometer output pin
9  PULSE = 2        # Pulse pro Umdrehung
```

Abbildung 69: Deklaration des Tachometerpins und den Pulsen pro Umdrehung

Folgend werden Setup Einstellungen der GPIO-Pins getätigt. Zeile 22 bestimmt die Nummerierung der GPIO Pins. Durch diese Einstellung kann man den GPIO 17 mit „17“ ansprechen und muss nicht die physikalische Pin-Nummer angeben. Ebenso wird der Tachometer-Pin in Zeile 24 als Input Pin deklariert, da ein Signal empfangen wird.

```
21 # Setup GPIO
22 GPIO.setmode(GPIO.BCM)
23 GPIO.setwarnings(False)
24 GPIO.setup(TACH, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Pull up to 3.3V
```

Abbildung 70: Setup Einstellung der GPIO Pins

In Zeile 39 wird, durch die Funktion GPIO.add_event_detect(), jede fallende Flanke erkannt und die Funktion „fell“ aufgerufen.

```
39 GPIO.add_event_detect(TACH, GPIO.FALLING, fell)
```

Abbildung 71: Erkennung der Fallenden Flanken

Die anschließende Funktion berechnet die Umdrehungen pro Minute (rpm) durch Errechnung der Frequenz.

```
27 def fell(n):
28     global t
29     global rpm
30
31     dt = time.time() - t
32     if dt < 0.005: return # um kurze Impulse zu ignorieren
33
34     freq = 1 / dt
35     rpm = (freq / PULSE) * 60
36     t = time.time()
```

Abbildung 72: Funktion zur Errechnung der Umdrehungen pro Minute (rpm)

```
50         print( "%.f RPM" % rpm)
51         rpm = 0
52         time.sleep(1)    # Eine Sekunde warten
```

Abbildung 73: Ausgabe der Umdrehungen pro Minute

4.4 Onlinevisualisierung

Übersicht

Um die Sensordaten für den Nutzer leicht anzuzeigen, wird ein System benötigt, welches die Daten sammelt, in eine Datenbank speichert und anschließend verständlich darstellt.

Die Daten der Sensoren werden zur Veranschaulichung in Grafen dargestellt. Um dies zu erreichen, werden die Sensorwerte mit MQTT, unter verschiedenen topics, an das Programm Node-RED geschickt. Dort können die Daten bei Bedarf verarbeitet werden.

Anschließend werden die Sensorwerte mit einem Zeitstempel versehen und in eine Datenbank gespeichert. Um schlussendlich einen Grafen zu erstellen, wird das Visualisierungsprogramm Grafana verwendet. Dieses greift lokal auf die Datenbank zu und erstellt einen Grafen für jeden Sensor.

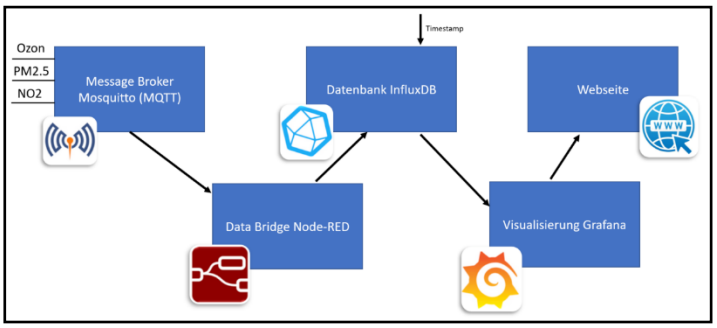


Abbildung 74: Blockschaltbild: Verarbeitung der Gesamtdaten

4.4.1 MQTT-Grundlagen

Allgemeines

MQTT oder auch „Message Queuing Telemetry Transport“ ist ein Nachrichtenprotokoll, das für Geräte mit hoher Latenz, geringer Bandbreite oder unzuverlässigen Netzwerken entwickelt wurde. Es wird oft für M2M (Machine-to-Machine) Kommunikation oder bei Verbindungsarten wie Internet of Things (IOT) verwendet, da die Akkuleistung und die Bandbreite hier oft entscheidend sind.

Es funktioniert mit einem sogenannten Publisher- / Subscriber-Prinzip, welches über einen zentralen Broker betrieben wird. Die Daten werden über ein publish unter einem gewissen topic gesendet. Empfangen werden sie, wenn der Empfänger sich auf das, vom Sender vorbestimmte topic, einschreibt (subscribe). [8]

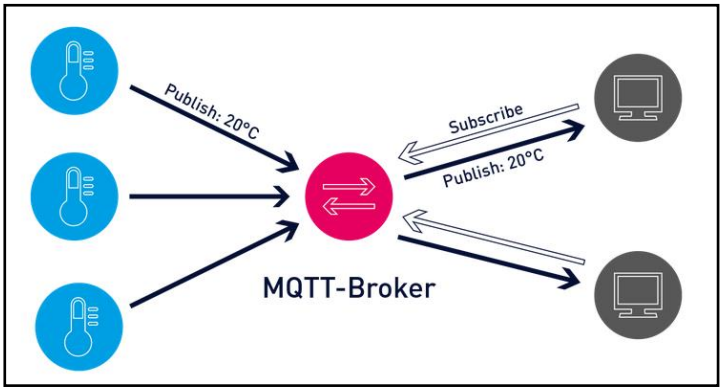


Abbildung 75 Genereller Aufbau wischen Sensor, MQTT Broker und Client [8]

MQTT-Topic

Geräte oder Sensoren können die Daten, die sie gerne senden möchten, unter einem bestimmten topic veröffentlichen (publish). Der MQTT-Broker überträgt anschließend die Informationen an diejenigen Clients, die dieses topic subscribed oder abonniert haben. Dies wird in den folgenden Kapiteln genauer erläutert.

Topics kann man sich wie eine Ordnerstruktur vorstellen. Ein topic könnte zum Beispiel so aussehen:

Sensoren/Ozon

Sensoren/PM2.5

Sensoren/NO2

Abbildung 76: MQTT-Topics

Ein Client, der die Ozon-Werte des Sensors bekommen möchte, kann sich nun auf das topic „Zuhause/Sensoren/Ozon“ einschreiben (subscribe) und die Daten empfangen.

Genauso kann ein Client mit der Nutzung eines # mehrere Unterthemen gleichzeitig empfangen. Um dies zu erreichen, müsste er dem topic „Zuhause/Sensoren/#“ subscriben. Jetzt erhält er die Werte aller Unterthemen. Hier wären es die Ozon-, PM2.5- und NO2 Werte.

Installation MQTT Protokoll

Folgendes muss gemacht werden:

Der Raspberry Pi wird upgedatet und upgegradet, um das OS auf den neuesten Stand zu bringen.

```
pi@raspberry:~ $ sudo apt update && sudo apt upgrade
```

Abbildung 77: update & upgrade

Um über das Kommandozeilen-Programm Advanced Packaging Tool „apt“ den Mosquitto-Broker zu installieren, wird folgender Befehl benutzt:

```
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
```

Abbildung 78: Installtion Mosquitto-Broker

Mosquitto soll bei jedem Start des Raspberry Pi automatisch starten.

```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

Abbildung 79: Autostart von MQTT

Der Befehl „systemctl enable“ sagt dem System, dass folgendes Programm ab nun automatisch starten soll.

Senden der Daten über verschiedene MQTT Topics

Es wurde ein Python Code zum Senden der Sensordaten geschrieben. Dieser importiert die Library paho-mqtt und die library Zeit (time).

```
import paho.mqtt.publish as publish
import time
```

Abbildung 80:

Um Sensordaten veröffentlichen (publish) zu können, muss ein MQTT-Server und ein MQTT-Path vorbestimmt werden. Als MQTT-Server wird der localhost benutzt.

```
MQTT_SERVER = "localhost"
```

Abbildung 81:

Da drei Sensoren verwendet werden, werden drei MQTT-Paths gebraucht. Dies sind die folgenden Pfade:

1. „Sensoren/Ozon“
2. „Sensoren/PM25“
3. „Sensoren/NO2“

```
MQTT_PATH1 = "Sensoren/Ozon"

MQTT_PATH2 = "Sensoren/PM25"

MQTT_PATH3 = "Sensoren/NO2"
```

Abbildung 82: verschiedene MQTT-Paths

Bei der Funktion zur Veröffentlichung (publish) der Daten werden die jeweiligen Variablen unter dem vorbestimmten MQTT-Path an den MQTT-Server geschickt.

```
publish.single(MQTT_PATH1, Werte_Ozon, hostname=MQTT_SERVER)

publish.single(MQTT_PATH2, Werte_PM25, hostname=MQTT_SERVER)

publish.single(MQTT_PATH3, Werte_NO2, hostname=MQTT_SERVER)
```

Abbildung 83: publish Funktionen der drei Sensoren

Dieser publish Vorgang wird alle 60 Sekunden wiederholt, um den neuen Sensorwert zu schicken.

```
time.sleep(60)
```

Abbildung 84: Sleep Funktion

Mit der Konfiguration des MQTT Brokers ist es nun möglich Sensor-Daten, unter den verschiedenen MQTT-Paths zu veröffentlichen (publish). Auf diese kann sich das Programm Node-RED nun einschreiben (subscribe), um die Sensordaten zu erhalten.

4.4.2 Node-RED

Allgemein

Node-RED ist ein grafisches Entwicklungswerkzeug. Mit der Software lassen sich Anwendungsfälle im Bereich Internet der Dinge (IOT) umsetzen. Funktionsbausteine, sogenannte Nodes, lassen sich durch Ziehen von Verbindungen, miteinander verknüpfen. Es gibt Eingabe-, Ausgabe- und Processing Nodes. Wenn man diese verbindet, kann man Daten verarbeiten, verschiedene Dinge kontrollieren und steuern. Im vorliegenden Projekt wird Node-RED verwendet, um die Sensor Daten über MQTT zu empfangen, diese zu kontrollieren umzuwandeln und anschließend in eine Datenbank zu speichern. [9]

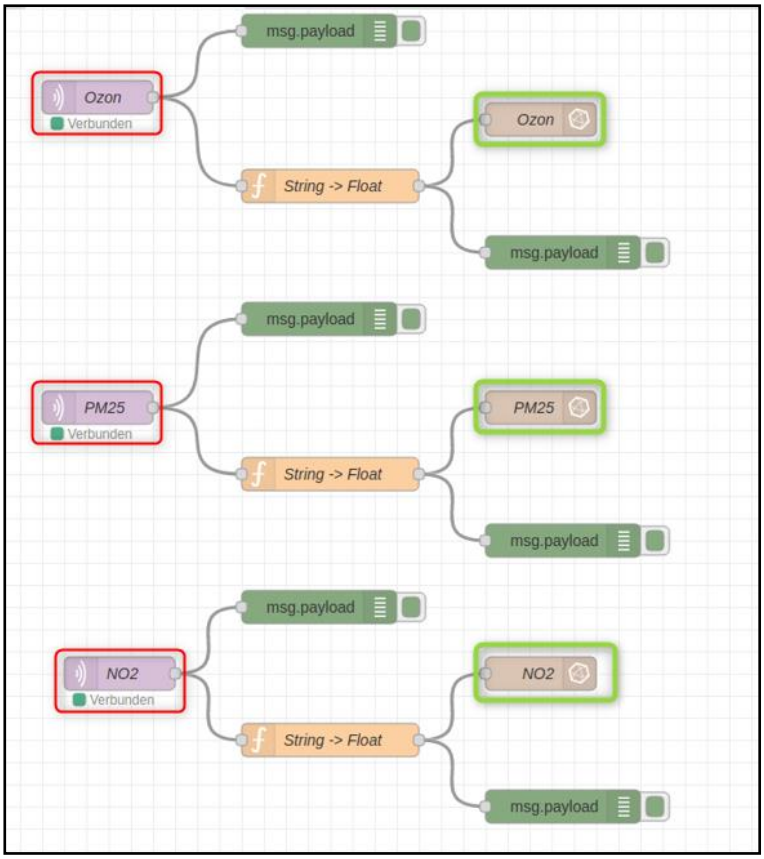


Abbildung 85: Ablaufplan der in dem Programm Node-RED gebaut wurde

Der Ablaufplan in Node-RED (Abbildung 85) besteht aus Funktionsblöcken oder auch Nodes, die Sensordaten über MQTT empfangen (rot) und anschließend in eine Datenbank schreiben (grün). Die einzelnen Nodes, unter anderem auch die Nodes „String->Float“ und „msg.payload“ werden in den folgenden Kapiteln genauer erklärt.

Empfangen der Daten über MQTT

Um nun die MQTT-Daten in Node-RED zu empfangen, benutzen wir den Node „mqtt in“, welcher sich unter dem Reiter „Netzwerk“ befindet. Dieser bewirkt, dass die Daten unter den entsprechenden MQTT-Paths eingelesen und weiterverarbeitet werden können.

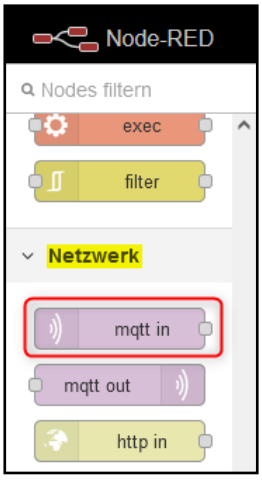


Abbildung 86: „mqtt in“ Node

Bevor die Daten empfangen werden können, müssen noch einige Voreinstellungen getroffen werden. Mit Doppelklick wird der Node bearbeitet und folgendes Fenster erscheint:

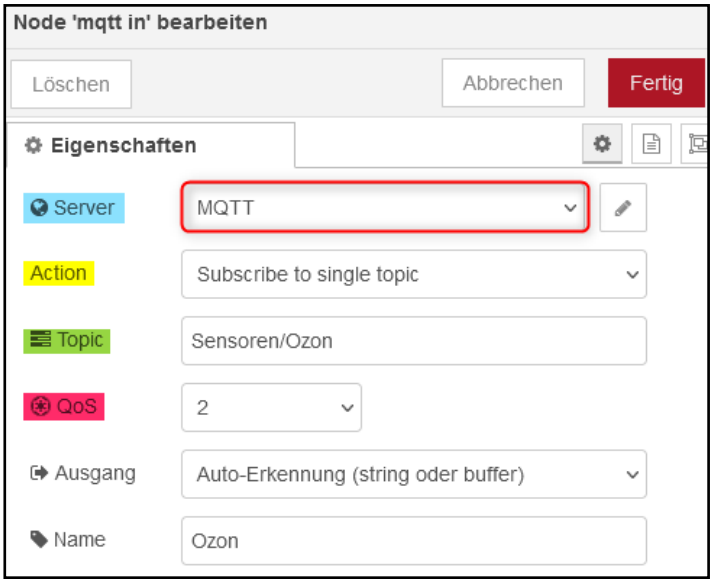


Abbildung 87: Einstellungen in der „mqtt in“ Node

Im vorliegenden Projekt läuft der MQTT-Broker lokal auf dem Raspberry Pi, einem „single-board computer“. Das Fenster Server, hier in Blau hervorgehoben, führt zu weiteren Einstellungen, die weiter unten, erklärt werden.

Unter dem Reiter Action stellen wir das Node auf „Subscribe to single topic“, da dieses Node nur die Ozon-Werte bekommen soll.

In das Fenster topic wird das jeweilige topic geschrieben, hier „Sensoren/Ozon“.

QoS oder auch Quality of Service uns zwischen verschiedenen Levels unterscheiden. Damit kann man die Nachrichtenlänge entweder minimieren oder die Verlässlichkeit der Daten erhöhen.

QoS 0: Diese Option ist für minimale Datenübertragung. Jede Nachricht wird nur einmal gesendet, ohne dass eine Bestätigung des Subscribers. Es gibt keine Möglichkeit zu wissen, ob der Subscriber die Daten erhalten hat oder nicht. Oft wird diese Option als „fire and forget“ oder „at most once delivery” beschrieben.

QoS 1: Der Broker schickt die Daten und wartet dann auf eine bestätigende Antwort des Subscribers, dass die Daten erhalten wurden. Wenn der Broker, in einer festgelegten Zeit, keine Antwort erhält, sendet er die Daten erneut. Bei dieser Option kann es dazu kommen, dass der Subscriber die Daten doppelt erhält. Deshalb wird QoS 1 auch als „at least once delivery“ beschrieben.

QoS 2: Der Client und der Broker unterhalten sich mithilfe eines vierstufigen Handschlages und gehen somit sicher, dass die Daten genau einmal angekommen sind. Deswegen wird dieses Level auch „exactly once delivery“ genannt.

Der Header jedes Datenpaketes ist limitiert auf 2 Byte und die Message-Payload ist limitiert auf 256 MB. Da wir in unserem Anwendungsfall nur jeweils einen float-Wert schicken, wurde bei der Auswahl der Quality of Service (QoS) das höchste Level, also Level 2 gewählt. [8]

Wie bereits erwähnt, öffnet sich (in Abbildung 87) unter dem Reiter „Server“ ein neues Fenster:

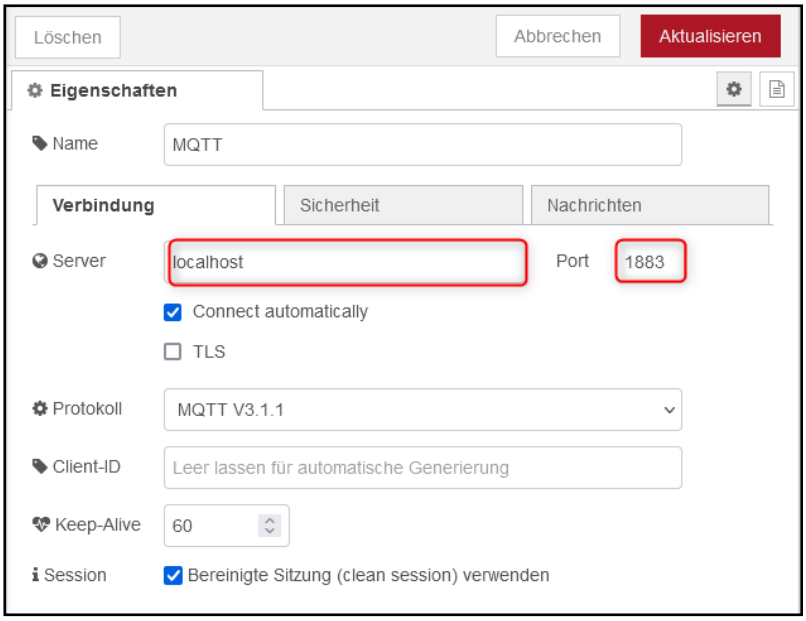


Abbildung 88: MQTT erweiterte Einstellungen

Hier kann man die Server IP eingeben. Da im vorliegenden Projekt der MQTT-Broker lokal auf dem Raspberry Pi läuft, wird „localhost“ mit dem Port 1883 eingetragen. Unter dem Reiter Sicherheit lassen sich, falls dies gewünscht ist, noch Benutzername und Passwort eingeben. Als Protokoll wird MQTT V3.11 benutzt und die Keep-Alive Zeit beträgt 60 Sekunden. Im Nächsten Schritt wird ein Debug-Node genutzt. Durch diesen lassen sich die Empfangen-Daten im Debug-Fenser von Node-RED ansehen.

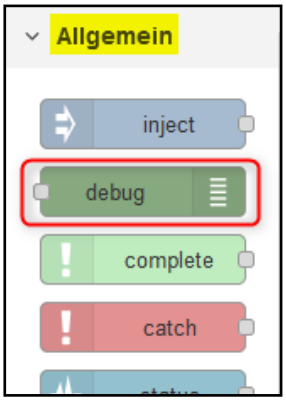


Abbildung 89: Debug Node

Die MQTT-In Node wird mit der Debug-Node verbunden und mit anschließend werden die Änderungen in der MQTT-Node gespeichert.

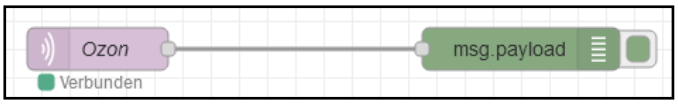


Abbildung 90: Verbindung „mqtt in“ Node mit „msg.payload“ Node

Der grüne Punkt unter der Node zeigt, dass die Verbindung erfolgreich ist. Überprüfung der Werte erfolgt im Debug-Fenster von Node-RED.

Abbildung 91: Ausgabe in dem Debug-Fenster

In Abbildung 91 ist zu erkennen, zu welcher Zeit, welche Werte als String geschickt wurden. Der nächste Schritt ist die Umwandlung der Werte in den Datentyp Float und das Speichern der Daten, mit einem Zeitstempel, in die Datenbank. Dies wird in folgenden Kapiteln beschrieben.

Umwandlung in von String in Float

Die Typenumwandlung der Messwerte ist not wendig, damit die Sensor Werte in dem Visualisierungsprogramm Grafana darstellbar sind. Die Umwandlung gestaltet sich dank Node-RED sehr einfach. Es wird ein Funktionsblock benutzt, dieser wandelt jedes geschickte Datenpaket von String in Float um. Der Funktion-Node befindet sich unter dem Reiter Funktion.

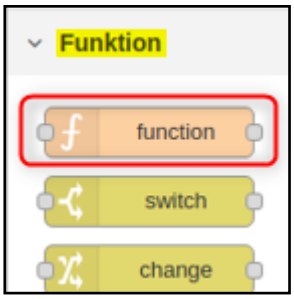


Abbildung 92: „function“ Node

Im Funktionsblock wird die payload Nachricht mit folgendem Befehl in den Datentyp Float umgewandelt:

```
msg.payload = Number(msg.payload)
```

Abbildung 93: Umwandlung payload zu Float

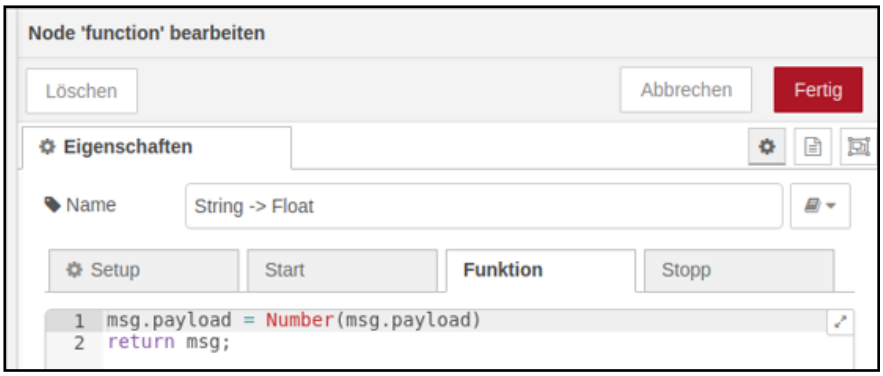


Abbildung 94: Inhalt des Funktionsblocks

Im Debug-Fenster von Node-RED kann man die Änderungen überprüfen:

Abbildung 95: Datentyp des Messwertes, nachdem die Zahl den Funktionsblock durchlaufen hat

Es ist erkennbar, dass die Daten nun als „Number“ erkannt werden.

Speichern in die Datenbank

Die Messwerte aus Node-RED werden in eine Datenbank gespeichert. Da der zeitliche Verlauf von Sensordaten dargestellt werden soll, ist es wichtig den Zeitpunkt, zu dem die Daten aufgenommen wurden, mitzuspeichern und somit die Daten mit einem Zeitstempel zu versehen.

Dazu wurde das Datenbank Tool InfluxDB verwendet. Dieses speichert bei jedem neuen Messwert automatisch den Zeitpunkt, zu dem diese erhalten wurde, in eine eigene Spalte.

Der InfluxDB Node ist nicht standardmäßig in Node-RED enthalten. Deshalb muss diese erst installiert werden.

Unter dem Pfad „Einstellungen → Palette verwalten → Installation“ lassen sich neue Nodes installieren.

Abbildung 96: Installation neuer Nodes

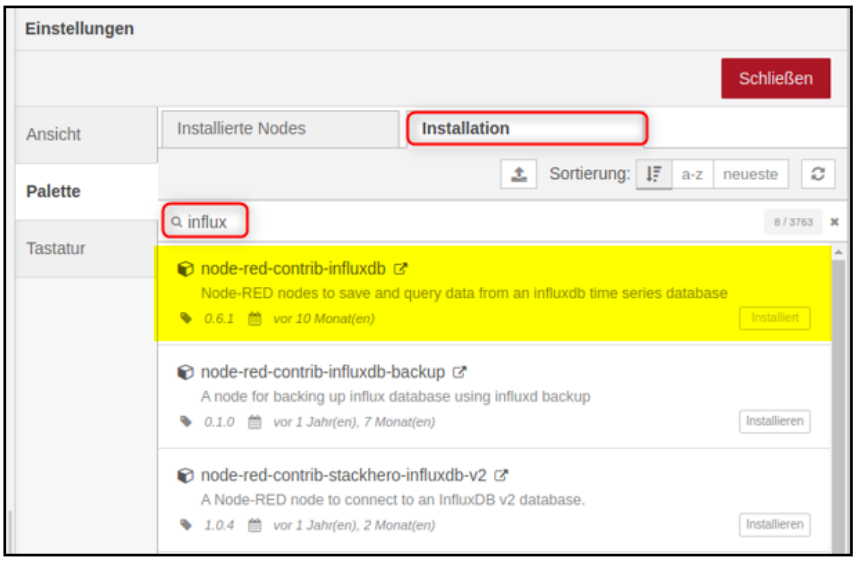


Abbildung 97: Installation InfluxDB Node

Hier wählt man „node-red-contrib-influxdb“ und lädt diese Anwendung. Nach der Installation findet man die neuen Nodes unter dem Reiter „Speicher“

Abbildung 98: „influxdb out“ Node unter dem Reiter „Speicher“

„Influxdb out“ ist der Node, der gebraucht wird, um die Daten in die Datenbank zu schreiben.

Dieser wird mit Doppelklick geöffnet, da noch einige Einstellungen getroffen werden müssen.

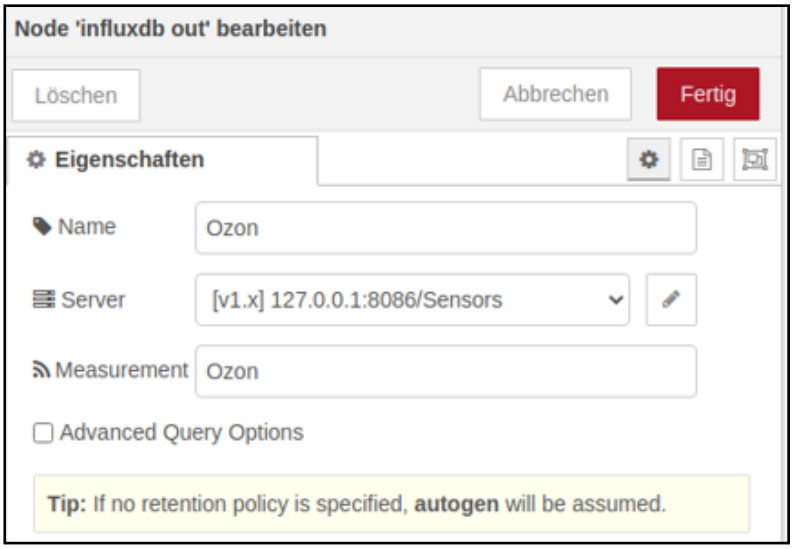


Abbildung 99: Einstellung der „influxdb out“ Node

Der Name dieses Node ist Ozon. Mit dem gleichen Namen wird auch das „Measurement“ Fenster gefüllt. Ein „Measurement“ kann man wie eine Unterteilung der Datenbank beschreiben. Eine Datenbank kann zum Beispiel Sensors heißen, hat aber drei Unterteilungs-„Measurements“ mit den Namen Ozon, PM25 und NO2 haben (Abbildung 100). In jedem Measurement gibt es eine eigene Tabelle mit Zeilen und Spalten.

Ozon

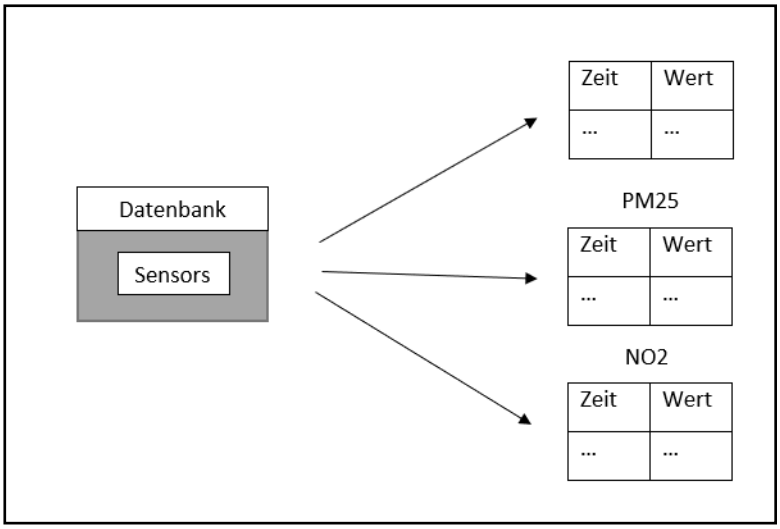


Abbildung 100: Datenbank „Sensors“ und die unter „Measurements“

In den erweiterten Einstellungen des Servers erscheint folgendes Fenster:

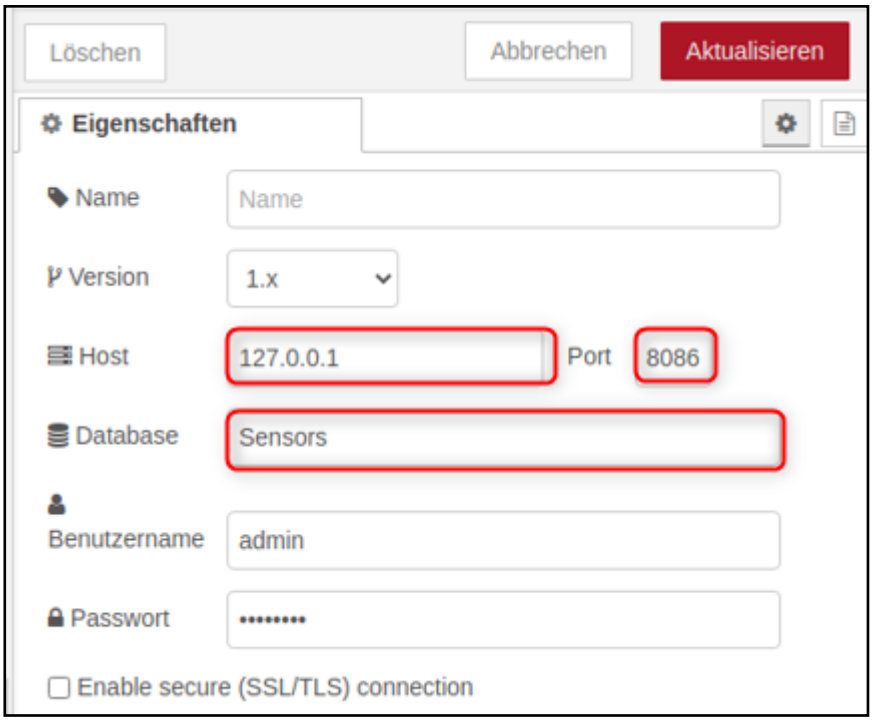


Abbildung 101: Erweiterte Einstellungen der „influxdb out“ Node

Hier legen wir den Host der Datenbank sowie den Port fest. Als Host wird „localhost“ oder eben die IP 127.0.0.1. Es wird der Port 8086 verwendet.

Ebenso muss der Name der Datenbank angegeben werden. Hier „Sensors“. Benutzername und Passwort müssen eingegeben werden, falls die Datenbank gesperrt werden soll.

4.4.3 Influx-DB

Allgemeines

Wenn man Sensordaten in einem Grafen zeitlich auftragen will, benötigt man zu jedem Datenwert einen genauen Zeitwert. Dies ist der Hauptgrund, warum InfluxDB benutzt wird. Es bringt den Vorteil, dass bei jedem neuen Datenwert automatisch ein Zeitstempel im UNIX Format erzeugt und in einer eigenen Spalte gespeichert wird.

Installation von InlfuxDB

Zuerst wird das InfluxDB Repository zu hinzugefügt:

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/os-release
echo "deb https://repos.influxdata.com/debian $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
```

Abbildung 102: Befehl, um das InfluxDB Repository hinzuzufügen

Als nächstes wird der Raspberry Pi upgedatet und InfluxDB installiert:

```
sudo apt update && sudo apt install -y influxdb
```

Abbildung 103: Installation InfluxDB

Abschließend wird InfluxDB gestartet und so eingestellt, dass es beim Boot des Raspberry Pi von selbst startet:

```
sudo systemctl unmask influxdb.service
sudo systemctl start influxdb
sudo systemctl enable influxdb.service
```

Abbildung 104: Autostart InlfuxDB

Erstellen der Datenbank Einträge

Mit dem Befehl „CREATE DATABASE Sensors“ wird die Datenbank „Sensors“ erstellt.

Die Sensordaten werden von Node-RED automatisch in die verschiedenen Measurements Ozon, PM25 und NO2 gespeichert.

Mit folgendem Befehl können alle Ozon-Werte mit den jeweiligen Datenwerten angesehen werden:

```
select * from Ozon
```

Abbildung 105: Befehl, um alle Datenwerte des „Measurements“ Ozon anzuzeigen

```
time                value
----                -----
1646328536465027003 8
1646328596527270384 10
1646328656563959913 2
1646328716615453102 7
1646328776673098891 2
1646328836707078355 6
1646328896770244694 1
1646328956818229686 1
1646329016856524317 5
1646329076920934615 4
1646329136970489288 10
1646329197035997230 2
1646329257046909943 7
1646329317063989537 6
1646329377094080376 7
1646329437103566958 6
1646329497171874197 5
1646329557190209077 6
1646329617202259153 7
1646329677222146342 8
1646329737280621336 9
1646329797291944021 1
1646329857314189018 2
1646329917355650017 7
```

Abbildung 106: der Sensorwerte mit den jeweiligen Zeitstempeln, die überprüft wurden

Die Zeitstempel lassen sich mit einem UNIX Timstamp Umrechner überprüfen. Auf die Umwandlung wird nicht speziell eingegangen. Es gibt online Umrechner die diesen Job problemlos und einfach erledigen.

4.4.4 Grafana

Allgemeines

Grafana ist eine Software, mit der man Daten in Dashboards visualisieren kann. Sie lässt sich mit vielen verschiedenen Datenquellen, unter anderem auch InfluxDB, speisen. Grafana wird oft für Monitoring-Aufgaben oder Visualisierung von Messdaten benutzt. [10]

Erstellung einer Datenquelle

Es wird eine Datenquelle (data source) hinzugefügt. Es wird die Datenbank InfluxDB benutzt, also wird diese hinzugefügt.

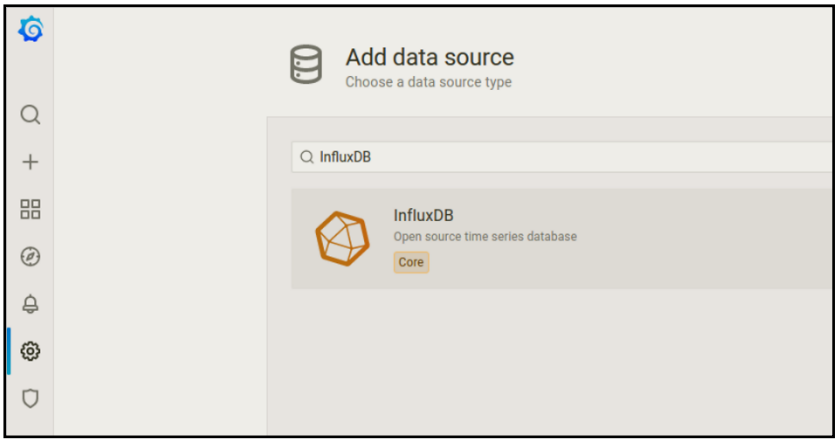


Abbildung 107: Einfügen der Datenquelle (InfluxDB)

Weitere Einstellungen der Datenquelle werden in Abbildung 108 getroffen. Hier muss die URL von InfluxDB eingegeben werden. Es wird der localhost mit dem Port 8086 verwendet.

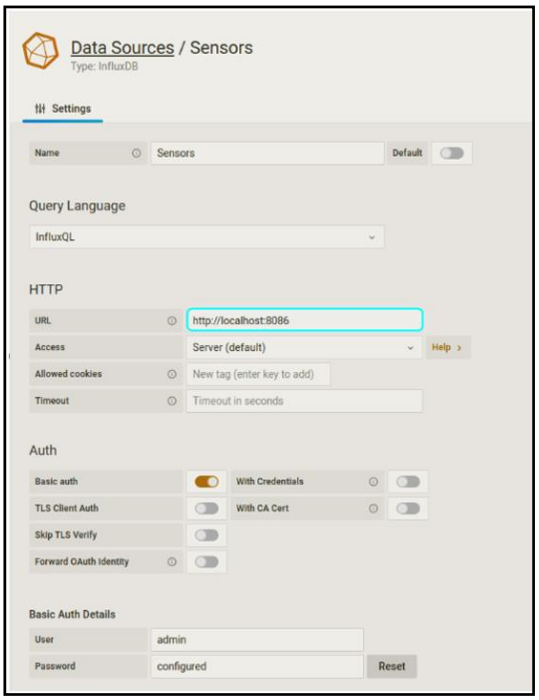


Abbildung 108: Einfügen der InfluxDB URL

Anschließend wird die verwendete Datenbank (Sensors) eingestellt. Weiteres sollte der Zugriff nur mit einloggen durch Usernamen und Passwort möglich sein.

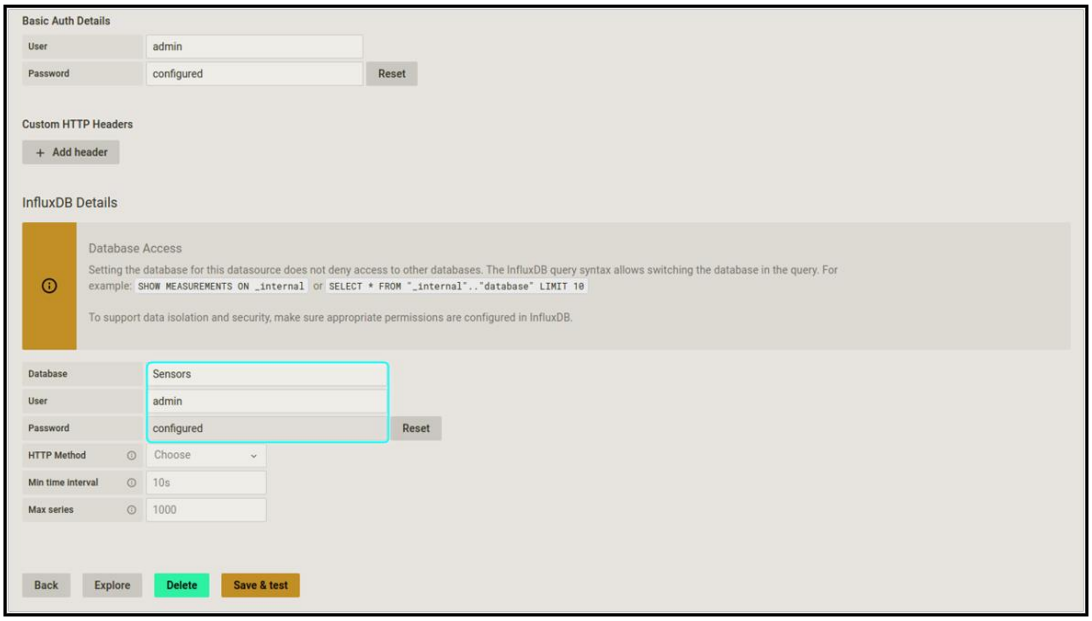


Abbildung 109: Festlegung der verwendeten Datenbank und Sicherung und einloggen mit Username und Passwort

Erstellung eines Panels

Unter dem Tab „Create new Dashboard“ kann ein neues Panel erzeugt werden. Ein Panel kann zum Beispiel ein Linien-, Kreis- oder Balkendiagramm sein.

Abbildung 110 Erstellungsfenster eines neuen Panels (Grafana)

Als erster Schritt wird die Datenquelle bzw. Datenbank ausgewählt (blau). Anschließend wird das entsprechende Measurement (rot) z.B. „Ozon“, „PM25“ oder „NO2“ festgelegt. Unter dem in lila hervorgehobenen Dropdown-Menü kann die Art der Visualisierung gewählt werden. Es

können, falls gewünscht, Balkendiagramme, Liniendiagramme, Tabellen, Kreisdiagramme, Histogramme und vieles mehr erzeugt werden. Dieser Vorgang wird für alle drei Sensoren wiederholt. Die entstandenen Panels lassen sich über einen Link in jede gewünschte Webseite einbauen, um auf die Daten von überall zugreifen zu können.

5 Ergebnisse – Abnahme

Der jetzige Projektstand ist ein System, welches die Luft filtert und den Feinstaubgehalt reduziert. Sowohl der Ozon also auch der Feinstaubsensor messen Daten. Diese werden über MQTT an Node-RED geschickt. In Node Red werden die Daten verarbeitet und anschließend in eine Datenbank gespeichert. Die Daten in der Datenbank werden als Grafen mit dem Visualisierungsprogramm Grafana dargestellt.

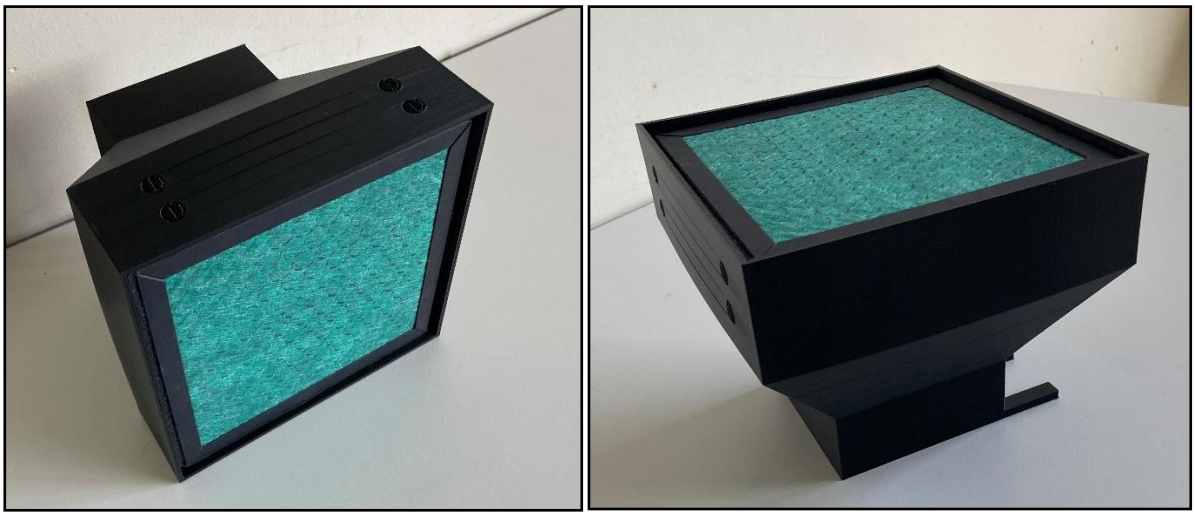


Abbildung 111: 3D-Druck des Gehäuses mit Feinstaub-Filter

Das Filter-Gehäuse ist voll funktional. Die Platienen des Lüftersystems sind noch in Produktion.

Die Sensorwerte werden empfangen, verarbeitet und sind in einem Grafen ersichtlich.

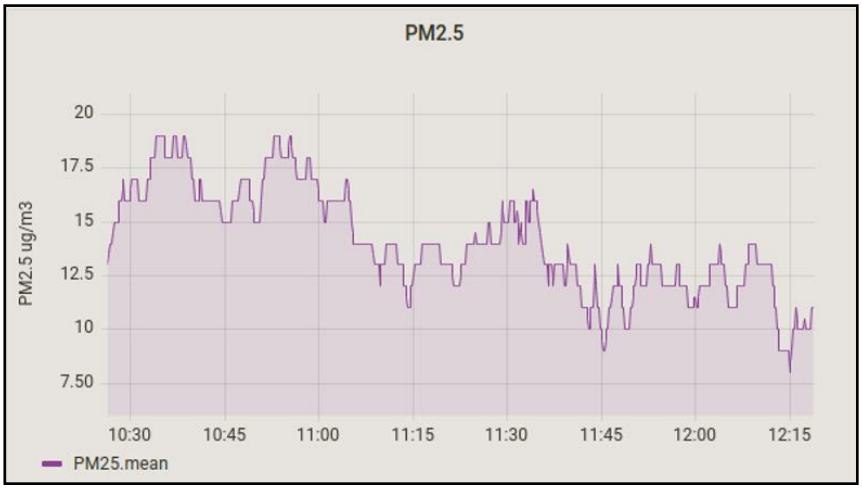


Abbildung 112: Sensor Werte des PM2.5 Sensors zeitlich dargestellt

Anforderung	Testergebnis	Freigabe (J/N)?
Die Lüftergeschwindigkeit kann geändert werden.	Die Lüftergeschwindigkeit kann durch ein Potentiometer geändert werden.	J
Der Feinstaubgehalt verbessert sich im Innenraum, während das Lüftersystem angeschaltet ist.	Da die Platinen noch in Fertigung sind, konnte dies noch nicht getestet werden.	N
Der Feinstaubgehalt wird gemessen.	Der Feinstaubgehalt wird gemessen und in der Konsole ausgegeben.	J
Der Ozongehalt wird gemessen.	Der Ozongehalt wird gemessen und in der Konsole ausgegeben.	J
Der Stickstoffdioxidgehalt wird gemessen.	Der Stickstoffdioxidgehalt gibt Werte aus, welche aber nicht interpretiert werden können.	N
Die Messwerte sind nicht immer die gleichen.	Die Messwerte ändern sich und bleiben nicht gleich.	J
Die Messdaten werden am Display dargestellt.	Die Daten werden auf dem Display angezeigt.	J
Die Messdaten werden auf Grafana dargestellt.	Die erhaltenen Messdaten werden graphisch dargestellt.	J

6 Literaturverzeichnis

[1] „Feinstaub Grenzwerte," [Online]. Available: (https://www.air-q.com/messwerte/feinstaub. [Zugriff am 18 12 2022].

[2] Ozongrenzwerte.

[3] Digikey Electronics, PMS5003 Series Manual Datasheet by Adafruit Industries LLC.

[4] „UART-Kommunikation," [Online]. Available: https://robotfreak.de/elab-wiki/images/5/5b/Uart-verbindung.png. [Zugriff am 17 2 2022].

[5] „Raspberry-Pi Pinout," [Online]. Available: https://bit.ly/3tnyq4P. [Zugriff am 28 12 2021].

[6] „PM2.5 Datasheet (2)," [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SEN0177_Web.pdf. [Zugriff am 20 2 2022].

[7] Noctua, „Noctua PWM specifications," [Online]. Available: https://noctua.at/pub/media/wysiwyg/Noctua_PWM_specifications_white_paper.pdf. [Zugriff am 04 04 2022].

[8] „What is MQTT? Definition and Details," [Online]. Available: https://www.paessler.com/it-explained/mqtt. [Zugriff am 04 04 2022].

[9] E. Bartmann, „IoT-Programmierung mit Node-RED: Visuell programmieren," [Online]. Available: https://www.isbn.de/buch/9783895763281_iot-programmierung-mit-node-red.htm. [Zugriff am 04 04 2022].

[10] „Was ist Grafana?," [Online]. Available: https://www.bigdata-insider.de/was-ist-grafana-a-1016619/. [Zugriff am 04 04 2022].

[11] „NO2-Grenzwerte," [Online]. Available: https://www.air-q.com/messwerte/stickstoffdioxid. [Zugriff am 18 12 2022].

[12] „PM2.5 Datasheet," [Online]. Available: https://bit.ly/3qjX6Jj. [Zugriff am 9 1 2022].

[13] „UART-Rahmen," [Online]. Available: https://i0.wp.com/edistechlab.com/wp-content/uploads/2020/11/Uart-Grafiken.png?fit=959%2C450&ssl=1. [Zugriff am 17 2 2022].

[14] „PPB in ug/m^3 Umrechnung (Bundesumweltamt)," [Online]. Available: https://www.umweltbundesamt.at/fileadmin/site/publikationen/REP0276.pdf. [Zugriff am 20 3 2022].

[15] „MQ131-Pinbelegung," [Online]. Available: shorturl.at/orOTU. [Zugriff am 21 2 2022].

[16] „MCP3008-Pinout," [Online]. Available: https://www.raspberry-pi-geek.de/wp-content/uploads/2016/08/Abbildung-71-4.jpg. [Zugriff am 29 2 2022].

[17] „SPI-Kommunikation," [Online]. Available: https://i0.wp.com/edistechlab.com/wp-content/uploads/2020/11/Screenshot-2020-11-22-at-09.32.01.png?fit=629%2C557&ssl=1. [Zugriff am 25 3 2022].

[18] „SPI-Datenübertragung," [Online]. Available: https://www.marvintest.com/downloads/KnowledgeBase/Q200275/Clock%20Polarity%20and%20Phase%20Timing%20Diagram%202.png. [Zugriff am 27 3 2022].

[19] „SPI Operation modes,“ [Online]. Available: . [Zugriff am 28 3 2022].

[20] „MICS2714-Sensor,“ [Online]. Available: https://download.mikroe.com/images/click_for_ide/no22_click.png. [Zugriff am 30 2 2022].

[21] „Pinout des MICS2714,“ [Online]. Available: https://www.mikroe.com/no2-2-click. [Zugriff am 30 3 2022].

[22] „LCD-Display Kommunikation zu Raspberry-Pi,“ [Online]. Available: https://www.blog.berrybase.de/wp-content/uploads/2020/11/Schaltung-589x1024.png. [Zugriff am 8 1 2022].

[23] „I2C-Kommunikation,“ [Online]. Available: https://i0.wp.com/edistechlab.com/wp-content/uploads/2020/11/I2C-Aufbau.png?fit=698%2C620&ssl=1. [Zugriff am 10 1 2022].

[24] „MQ131-Sensor,“ [Online]. Available: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSMqLIoOwkjMwH2oI4Q7oB6zmvO_z1U62AN4jeuTIQu8r7jZwI0YpqQ1t3HtNHTz9q3k2Q&usqp=CAU. [Zugriff am 19 1 2022].

[25] „MQ131 mit Modulplatine,“ [Online]. Available: https://www.okystar.com/wp-content/uploads/2018/10/mq-131-ozone-sensor-module-1.jpg. [Zugriff am 19 1 2022].

7 Verzeichnis der Abbildungen, Tabellen und Abkürzungen

7.1 Abbildungen

Abbildung 1: Grober Entwurf des Projektes 12
Abbildung 2: GANTT-Diagramm KurS 20
Abbildung 3: GANTT-Diagramm HauA 21
Abbildung 4: Grenzwerte von Feinstaub [1] 22
Abbildung 5: Ozongrenzwerte [2] 23
Abbildung 6: Stickstoffdioxidgrenzwerte [11] 23
Abbildung 7: Blockschaltbild des PM2.5-Sensors [3] 25
Abbildung 8: Pins des PM2.5-Sensor [12] 26
Abbildung 9: Pin-Belegung des Sensors [12] 26
Abbildung 10: Darstellung einer UART-Schnittstelle [4] 27
Abbildung 11: Rahmenstruktur einer UART-Übertragung [13] 28
Abbildung 12: Raspberry-Pi Pin-out [5] 29
Abbildung 13: Raspberry-Pi (links) angeschlossen an den PM2.5-Sensor (rechts) 29
Abbildung 14: Blockschaltbild der Übertragungskette 30
Abbildung 15: Loop des Codes zum Auslesen der Daten (Zeile 1-29) 31
Abbildung 16: Loop des Codes zum Auslesen (Zeile 30-47) 31
Abbildung 17: Libary Code des PM2.5-Sensors, für den Daten-Rahmen 32
Abbildung 18: Ausschnitt des Kommunikation-Protokolls [6] 33
Abbildung 19: Hinzufügen einer Übertragungsprotokoll 34
Abbildung 20: Wählen des richtigen Analyzers 34
Abbildung 21: Settings Logic-Analyzers 34
Abbildung 22: UART Einstellungen laut Datenblatt 35
Abbildung 23: Genauere Einstellungen für UART treffen 35
Abbildung 24: Messergebnis mit Logic-Analyzer (Teil 1) 35
Abbildung 25: Messergebnis mit Logic-Analyzer (Teil 2) 35
Abbildung 26: UART-Rahmen Format (Datenblatt https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf) 36

Abbildung 27: Ausschnitt des Daten-Outputs zum gleichen Zeitpunkt, wie der der Logic-Analyzer Messung 36
Abbildung 28: Hex zu Dezimal 36
Abbildung 29: Umrechnung ppb in ug/m^3 laut dem Umweltbundesamt [14] 37
Abbildung 30: Modulplatine mit aufgesetztem Sensor [25] 38
Abbildung 31: MQ131-Sensor von Winsen-Sensor [24] 38
Abbildung 32: Pin-Belegung des Sensors [15] 39
Abbildung 33: Beschreibung der Pins 39
Abbildung 34: Kommunikation des Ozonsensor zu Raspberry-Pi 40
Abbildung 35: PIn Belegung des MCP3088 [16] 40
Abbildung 36: Beispiel einer SPI Kommunikation [17] 41
Abbildung 37 SPI Operation Modes 42
Abbildung 38 SPI Operation Mode 42
Abbildung 39: SPI-Operation Modes [19] 42
Abbildung 40: SPI-Datenübertragung [18] 43
Abbildung 41: Übertragungskette des Ozonsensors zu MQTT 43
Abbildung 42: Libary-Code für MQ131 (Zeile 1-44) 44
Abbildung 43: Libary-Code des MQ131 (44-85) 45
Abbildung 44: Libary-Code des Ozon-Sensors (86-101) 45
Abbildung 45: Python Loop des Ozon-Sensors 46
Abbildung 46: Ausgabe der Konsole 46
Abbildung 47: NO2-Clickboard [20] 47
Abbildung 48: NO2-Spezifikationen 48
Abbildung 49: Pinout des NO2-Sensors von Mikroe [21] 48
Abbildung 50: SPI-Kommunikation Sensor zu Microcontroller 49
Abbildung 51: Arduino-Code 50
Abbildung 52: Siickstoffdioxid von ppb auf ug/m^3 [14] 50
Abbildung 53: Stickstoffdioxidsensor mit Raspberry-Pi 51
Abbildung 54: Python-Code nach der Entdeckung der Falschen Libary 53
Abbildung 55: LCD-DIsplay zu Mikrocontroller [22] 54
Abbildung 56: I2C-Kommunkation [23] 55
Abbildung 57: Python Implementierung des LCD-Displays 56

Abbildung 58: Stromlaufplan des Lüftersystem 57
Abbildung 59 Beschaltung des Potentiometers 58
Abbildung 60: Pinbelegung des MCP3008 59
Abbildung 61: Stromlaufplan des ADC Bausteins MCP3008 und Raspberry Pi 59
Abbildung 62: Inkludieren der „gpiozero“ library und Ausgabe des Potentiometer Wertes 60
Abbildung 63: Python Code (PWM) 61
Abbildung 64: Datasheet Auszug: Generierung Tachometer Signal [7] 62
Abbildung 65: Pull-Up Widerstand an der Tachometerleitung 63
Abbildung 66: gemessenes Tachometersignal mit Zeitmessung zwischen zwei fallenden Flanken mithilfe der Cursor Funktion des Oszilloskops 63
Abbildung 67: ideales Rechtecksignal mit Beschriftung (angepasst an Lüfter) 63
Abbildung 68: Importieren der Rpi.GPIO und time library 64
Abbildung 69: Deklaration des Tachometerpins und den Pulsen pro Umdrehung 64
Abbildung 70: Setup Einstellung der GPIO Pins 64
Abbildung 71: Erkennung der Fallenden Flanken 64
Abbildung 72: Funktion zur Errechnung der Umdrehungen pro Minute (rpm) 65
Abbildung 73: Ausgabe der Umdrehungen pro Minute 65
Abbildung 74: Blockschaltbild: Verarbeitung der Gesamtdaten 65
Abbildung 75 Genereller Aufbau wischen Sensor, MQTT Broker und Client [8] 66
Abbildung 76: MQTT-Topics 67
Abbildung 77: update & upgrade 67
Abbildung 78: Installtion Mosquitto-Broker 67
Abbildung 79: Autostart von MQTT 67
Abbildung 80: 68
Abbildung 81: 68
Abbildung 82: verschiedene MQTT-Paths 68
Abbildung 83: publish Funktionen der drei Sensoren 68
Abbildung 84: Sleep Funktion 69
Abbildung 85: Ablaufplan der in dem Programm Node-RED gebaut wurde 70
Abbildung 86: „mqtt in“ Node 71
Abbildung 87: Einstellungen in der „mqtt in“ Node 71
Abbildung 88: MQTT erweiterte Einstellungen 73

Abbildung 89: Debug Node 73
Abbildung 90: Verbindung „mqtt in“ Node mit „msg.payload“ Node 74
Abbildung 91: Ausgabe in dem Debug-Fenster 74
Abbildung 92: „function“ Node 74
Abbildung 93: Umwandlung payload zu Float 74
Abbildung 94: Inhalt des Funktionsblocks 75
Abbildung 95: Datentyp des Messwertes, nachdem die Zahl den Funktionsblock durchlaufen hat 75
Abbildung 96: Installation neuer Nodes 76
Abbildung 97: Installation InfluxDB Node 76
Abbildung 98: „influxdb out“ Node unter dem Reiter „Speicher“ 77
Abbildung 99: Einstellung der „influxdb out“ Node 77
Abbildung 100: Datenbank „Sensors“ und die unter „Measurements“ 78
Abbildung 101: Erweiterte Einstellungen der „influxdb out“ Node 78
Abbildung 102: Befehl, um das InfluxDB Repository hinzuzufügen 79
Abbildung 103: Installation InfluxDB 79
Abbildung 104: Autostart InlfuxDB 79
Abbildung 105: Befehl, um alle Datenwerte des „Measurements“ Ozon anzuzeigen 80
Abbildung 106: der Sensorwerte mit den jeweiligen Zeitstempeln, die überprüft wurden 80
Abbildung 107: Einfügen der Datenquelle (InfluxDB) 81
Abbildung 108: Einfügen der InfluxDB URL 81
Abbildung 109: Festlegung der verwendeten Datenbank und Sicherung und einloggen mit Username und Passwort 82
Abbildung 110 Erstellungsfenster eines neuen Panels (Grafana) 82
Abbildung 111: 3D-Druck des Gehäuses mit Feinstaub-Filter 84
Abbildung 112: Sensor Werte des PM2.5 Sensors zeitlich dargestellt 84

8 Begleitprotokoll gemäß § 9 Abs. 2 PrO

8.1 Begleitprotokoll Kurzmann

Name: Hr. Samuel Kurzmann
Diplomarbeitstitel: LRS

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen	3,5h
39	Diplomarbeitsantrag einreichen	1h
40	Projektplanung, Aufgabenverfeinerung, Ganttdiagramm	2h
41	Bauteilbeschaffung	4,5h
42	Arbeiten am Lüfterprototyp	10h
43	Arbeiten am Lüfterprototyp	10h
44	Arbeiten des Lüfterprototyps mit anderen Lüftern	10h
45	Testung des Lüfterprototyps	10h
46	Mehrstündige Testung des Lüfters (PWM und Tacho) / Arbeiten an den Sensoren	10h
47	Laborübung zu PM2.5-Sensor	5h
48	Protokoll zu Laborübung und Beschaffung des Raspberry pi´s	10h
49	Inbetriebnahme des Raspberry pi´s	10h
50	Arbeiten an der MQTT Verbindung	10h
51	Arbeiten an der MQTT Verbindung	10h
52	Arbeiten an der MQTT Verbindung	10h
01	Node-RED subscribe to MQTT	10h
02	Node-RED Datenverarbeitung	10h
03	Node-RED Datenverarbeitung	10h
04	Datenbankrecherche (Timestamp)	10h
05	Datenbankerstellung	10h
06	Daten von Node-RED in Datenbank schreiben	10h
07	Daten von Node-RED in Datenbank schreiben	10h
08	Zugriff von Grafana auf Datenbank	10h
09	Zugriff von Grafana auf Datenbank	10h
10	Visualisierung der Daten auf Grafana	10h
11	Beheben von Fehlern bei der Datenübertragung	10h
12	Diplomarbeit schreiben	10h
13	Diplomarbeit schreiben	10h

KW …Kalenderwoche
----- **…**Semesterabschnitt

8.2 Begleitprotokoll Haumtratz

Name: Hr. Andre Haumtratz
Diplomarbeitstitel: LRS

KW	Beschreibung	Zeitaufwand
38	Diplomarbeitsantrag erstellen	3,5h
39	Diplomarbeitsantrag einreichen	1h
40	Projektplanung, Aufgabenverfeinerung, Ganttdiagramm	2h
41	Bauteilbeschaffung	4,5h
42	Arbeiten an den Sensoren	10h
43	Arbeiten an den Sensoren	10h
44	Arbeiten an den Sensoren	10h
45	Arbeiten an den Sensoren	10h
46	Arbeiten an den Sensoren	10h
47	Laborübung zu PM2.5-Sensor	5h
48	Protokoll zu Laborübung und Beschaffung des Raspberry PIs	10h
49	Inbetriebnahme des Raspberry Pis	10h
50	Einarbeiten in den Raspberry PI	10h
51	Einarbeiten in den Raspberry PI	10h
52	Sensor Umstellung auf Python	10h
01	Sensor Umstellung auf Python	10h
02	Sensor Umstellung auf Python	10h
03	Ansteuerung des LCD-Displays	10h
04	Beheben von Fehlern bei der Datenübertragung	10h
05	Implementierung des PM2.5-Sensors auf MQTT	10h
06	Arbeiten an den Sensoren	10h
07	Ansteuerung des LCD-Displays	10h
08	Arbeiten an den Sensoren	10h
09	Arbeiten an den Sensoren	10h
10	Visualisierung der Daten auf Grafana	10h
11	Beheben von Fehler im Python-Code	10h
12	Diplomarbeit schreiben	10h
13	Diplomarbeit schreiben	10h

KW …Kalenderwoche
----- **…**Semesterabschnitt