

# **BACHELORARBEIT**

## **Möglichkeiten zur Einbindung von Single Board Computer in ein bestehendes Fertigungsumfeld**

durchgeführt am  
Studiengang Informationstechnik & System–Management  
an der  
Fachhochschule Salzburg GmbH

vorgelegt von

**Michael Pfnür**



Studiengangsleiter: FH-Prof. DI Dr. Gerhard Jöchtl

Betreuer: FH-Ass. Prof. Dipl. Phys. Judith Schwarzer

Salzburg, Mai 2016

## Eidesstattliche Erklärung

Hiermit versichere ich, Michael Pfnür, geboren am 22.Mai 1981, dass die vorliegende Bachelorarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Salzburg, am 15.05.2016



Michael Pfnür

1310555048

\_\_\_\_\_  
Matrikelnummer

## Allgemeine Informationen

Vor- und Zuname: Michael Pfnür  
Institution: Fachhochschule Salzburg GmbH  
Studiengang: Informationstechnik & System-Management  
Titel der Bachelorarbeit: Möglichkeiten zur Einbindung von Single Board Computer in ein bestehendes Fertigungsumfeld  
Schlagwörter: KEYWORD 1, KEYWORD 2, KEYWORD 3, KEYWORD 4, KEYWORD 5  
Betreuer an der FH: FH-Ass. Prof. Dipl. Phys. Judith Schwarzer

## Abstract

Abstract

## Danksagung

Zunächst möchten ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt haben.

Ganz besonders danken möchten ich in erster Linie meiner Betreuerin, Frau FH-Ass. Prof. Dipl. Phys. Judith Schwarzer, für ihre ausgiebige Unterstützung. Durch stetiges Hinterfragen und konstruktive Kritik verhalf sie mir zu einer durchdachten Herangehensweise und Umsetzung. Dank ihrer Erfahrung konnte sie mich immer wieder bei meinen Recherchen und bei meinen Fragen unterstützen. Vielen Dank für Zeit und Mühen, die Sie in meine Arbeit investiert haben.

Auch möchten ich mich bei der Robert Bosch AG für die gegebene Möglichkeit dieses Projekt durchzuführen, sowie für das dazu benötigte Equipment, welches zur Verfügung gestellt wurde bedanken.

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>5</b>
<b>Abkürzungsverzeichnis</b>	<b>6</b>
<b>Abbildungsverzeichnis</b>	<b>7</b>
<b>Tabellenverzeichnis</b>	<b>8</b>
<b>Listingverzeichnis</b>	<b>9</b>
<b>1 Einleitung</b>	<b>10</b>
<b>2 Theoretischer Teil</b>	<b>11</b>
2.1 Begriffsdefinition . . . . .	11
2.2 Vergleich von verschiedenen SBC . . . . .	12
2.3 Raspberry Pi 3 . . . . .	14
2.3.1 Allgemeine technische Daten . . . . .	14
2.3.2 GPIO-Kontakte . . . . .	15
2.4 Bussysteme . . . . .	17
2.4.1 1-Wire . . . . .	17
2.4.1.1 Allgemeine Informationen . . . . .	17
2.4.1.2 Übertragungsprotokoll . . . . .	18
2.4.2 I <sup>2</sup> C-Bus . . . . .	19
2.4.2.1 Allgemeine Informationen . . . . .	19
2.4.2.2 Definierte Zustände und Adressierung . . . . .	19
2.4.2.3 Übertragungsprotokoll . . . . .	21
<b>3 Praktischer Teil</b>	<b>23</b>
3.1 Benötigte Materialien . . . . .	23
3.2 Temperaturmessung mit Sensor DS18S20 . . . . .	24
3.2.1 DS18S20 . . . . .	24
3.2.2 Schaltungsaufbau . . . . .	25
3.2.3 Auslesen des Sensors . . . . .	26
3.2.4 Datenspeicherung . . . . .	27
3.2.5 Visualisieren der Daten . . . . .	29
3.3 Temperatur- und Luftfeuchtigkeitsmessung mit HYT-221 . . . . .	30

3.3.1	HYT-221	30
3.3.2	Schaltungsaufbau	31
3.3.3	Auslesen der Daten	32
3.3.4	Datenspeicherung	33
3.3.5	Visualisierung der Daten	34
3.4	Vibrationsmessung mit Sensor BMA020	35
3.4.1	BMA 020	35
3.4.2	Schaltungsaufbau	36
3.4.3	Auslesen der Daten	36
3.4.4	Datenspeicherung	38
3.4.5	Visualisierung der Vibrationswerte	38
<b>4</b>	<b>Zusammenfassung und Ausblick</b>	<b>39</b>
<b>Literatur</b>		<b>40</b>
<b>A</b>	<b>Quellcode</b>	<b>42</b>
A.1	Python Script DS18S20	42
A.2	Erstellung Graphen DS18S20	43
A.3	Python Script HYT-221 mit RRD	45
A.4	Python Script HYT-221 mit mySQL	48
A.5	Erstellung Graphen HYT-221	51
A.6	Python Script Vibrationsmessung	57
<b>B</b>	<b>Abbildungen</b>	<b>60</b>
B.1	Graphen Temperatur und Luftfeuchtigkeit	60
B.2	Visualisierung der Vibrationswerte	60

## Abkürzungsverzeichnis

**SPI** Serial Peripheral Interface

**SBC** Single Board Computer

**I<sup>2</sup>C** Inter-Integrated Circuit

**CAN** Controller Area Network

**RPI** Raspberry Pi

**BLE** Bluetooth Low Energy

**CSI** Camera Serial Interface

**DSI** Display Serial Interface

**GPIO** General Purpose Input / Output

**TWI** Two-Wire-Interface

**IST** INNOVATIVE SENSOR TECHNOLOGY

## Abbildungsverzeichnis

2.1	Raspberry Pi 3 [1] . . . . .	15
2.2	GPIO-Header [2] . . . . .	16
2.3	Bedingung für gültiges Bit auf Datenleitung [3, S. 9] . . . . .	20
2.4	START, STOP Bedingung Inter-Integrated Circuit (I <sup>2</sup> C) [3, S. 9] . . .	20
2.5	Erstes Byte nach Start Bedingung I <sup>2</sup> C [3, S. 13] . . . . .	21
2.6	Datentransfer I <sup>2</sup> C Bus [3, S. 13] . . . . .	22
3.1	elektrische Daten DB18S20 [4, S. 2] . . . . .	24
3.2	DS18S20 [5] . . . . .	24
3.3	Schaltungsaufbau DS18S20 . . . . .	25
3.4	Temperaturverlauf der letzten Stunde . . . . .	29
3.5	Temperaturverlauf der letzten 10 Stunden . . . . .	29
3.6	HYT-221 [6] . . . . .	31
3.7	Schaltungsaufbau mit drei Sensoren . . . . .	31
3.8	Platine mit BMA020 [7] . . . . .	35
3.9	Schaltungsaufbau Vibrationsmessung mit BMA020 . . . . .	36
B.1	Temperaturverlauf der letzten Stunde . . . . .	60
B.2	Temperaturverlauf der letzten 10 Stunden . . . . .	60
B.3	Temperaturverlauf der letzten 3 Tage . . . . .	61
B.4	Luftfeuchtigkeit der letzten Stunde . . . . .	61
B.5	Luftfeuchtigkeit der letzten 10 Stunden . . . . .	62
B.6	Luftfeuchtigkeit der letzten 3 Tage . . . . .	62
B.7	Vibration im normalen Bereich . . . . .	63
B.8	Vibration im kritischen Bereich . . . . .	63

## Tabellenverzeichnis

2.1	Vergleich OS, RAM, CPU verschiedener SBCs . . . . .	13
2.2	Vergleich Schnittstellen, Netzwerkverbindung, Anzahl GPIO Pins . . . . .	13
2.3	Befehle bei 1-Wire [8, S. 35] . . . . .	18
3.1	benötigte Materialien . . . . .	23
3.2	Technische Daten HYT-221 [9] . . . . .	30
3.3	Technische Daten BMA020 [10] . . . . .	35

## Listings

3.1	Auslesen der gemessenen Temperatur . . . . .	26
3.2	Erstellung RRD Datenbank . . . . .	27
3.3	Datenspeicherung in DB . . . . .	28
3.4	Auslesen der Temperatur und Luftfeuchtigkeit . . . . .	32
3.5	Verbindung zur DB herstellen . . . . .	33
3.6	Speicherung der Daten in mySQL DB . . . . .	34
3.7	Auslesen der Vibrationswerte . . . . .	37
A.1	Quellcode zum Auslesen des DS18S20 . . . . .	42
A.2	Quellcode Erstellung Temperaturgraphen DS18S20 . . . . .	43
A.3	Quellcode zum Auslesen und Speichern der Daten von drei Sensoren mit RRDtool . . . . .	45
A.4	Quellcode zum Auslesen und Speichern der Daten von drei Sensoren mit mySQL DB . . . . .	48
A.5	Quellcode zum Erstellen der Graphen für die drei Sensoren . . . . .	51
A.6	Quellcode zum Auslesen und Speichern der Vibrationswerte . . . . .	57

# 1 Einleitung

Ein häufig verwendetes Schlagwort heutzutage wenn über die industrielle Produktion gesprochen wird ist **Industrie 4.0**. Damit ist vornehmlich gemeint, dass viele Abläufe vollautomatisiert stattfinden gehen. Egal ob dies in der Produktion selbst, in der Logistik, bei der Materialbestellung oder auch beim Versand der Fall ist. Um diese Anforderungen zu bewerkstelligen werden immer neuere Technologien eingesetzt, in diesen Zusammenhang werden Produkte wie Kleinstrechner, sogenannte **Single Board Computer (SBC)** von immer größerem Interesse. Beispiele hierfür sind der „**Raspberry Pi**“ oder auch der „**Banana Pi**“, um nur einmal zwei der bekanntesten zu nennen. Es gibt allerdings auch noch eine Vielzahl anderer Produkte von SBCs auf dem Markt.

Die Aufgabe dieser Arbeit bestand darin, die Möglichkeiten für einen Einsatz von solchen SBCs in einem bestehenden Produktionsumfeld zu erproben. Der Bereich für den Einsatz erstreckt sich von der Temperaturmessung in den einzelnen Maschinen einer Produktionslinie bis hin zur Temperaturmessung in Schaltschränke oder Serverräumen um etwaige zu hohe Temperaturen frühzeitig erkennen zu können und diesen entgegenzuwirken. Weiterhin sollten auch noch Möglichkeiten für den Einsatz von Feuchtigkeitssensoren oder Vibrationssensoren erarbeitet werden, um z.B. Aussagen über die Schwingungsbelastung von nahegelegenen vielbefahrenen Zugstrecken und deren eventuelle Auswirkung auf die Produktion tätigen zu können. Ein weiterer zu erarbeitender Punkt war unterschiedliche Möglichkeiten zu testen, um die von den Sensoren gelieferten Daten effektiv zu speichern und aufzubereiten.

Die Arbeit ist folgendermaßen gegliedert. In Kapitel zwei werden einige der sich auf dem Markt befindenden SBCs miteinander verglichen um deren Vor- und Nachteile darzulegen und die bestmögliche Variante für die gegebenen Anforderungen auswählen zu können. Weiterhin werden die in den folgenden Kapiteln verwendeten Fachbegriffe erklärt, um diese zu verstehen. Auch werden die im späteren Verlauf verwendeten Bussysteme zur Übertragung der Daten vom Sensor an den Single Board Computer erläutert, sowie dessen Übertragungsprotokolle. Im dritten Kapitel dem praktischen Teil werden die verschiedenen Schaltungen von unterschiedlichen Sensoren, sowie den unterschiedlichen Möglichkeiten zur Speicherung und Visualisierung der Daten dargestellt. Dabei wird vor allem auf die Datenspeicherung mittels **mySQL** und mit dem **RRDTool** eingegangen. Im letzten, den vierten Kapitel werden die zuvor erlangten Ergebnisse noch einmal zusammengefasst und ein Ausblick auf die Verwendung von SBCs in den verschiedenen Einsatzbereichen für die Zukunft gegeben.

## 2 Theoretischer Teil

Dieses Kapitel befasst sich mit den theoretischen Grundlagen, die für den Einsatz von SBCs zur Datenerfassung mittels verschiedenen Sensoren nötig sind. In Abschnitt 2.1 werden die im weiteren Verlauf der Arbeit verwendeten Fachbegriffe erläutert, um die beschriebenen Zusammenhänge gut zu verstehen. Der Abschnitt 2.2 behandelt die unterschiedlichen sich auf dem Markt befindenden SBCs mit ihren jeweiligen Vor- bzw. Nachteilen. Die verschiedenen Bussysteme wie z.B. der I<sup>2</sup>C Bus werden in Abschnitt 2.4 erklärt und deren Funktionsweise erläutert.

### 2.1 Begriffsdefinition

Die in diesem Abschnitt beschriebenen Definitionen wurden, soweit nicht anders angegeben aus folgendem Dokument entnommen [8].

#### Single Board Computer

Unter einem *Single Board Computer* (SBC) versteht man ein Computersystem, welches sich komplett auf einer einzigen Platine befindet. SBCs können fast die gleichen Aufgaben erledigen wie gewöhnliche Computer, allerdings sind die Einplatinen Rechner diesen in Hinblick auf die Hardwareausstattung doch um einiges unterlegen.

#### Bussysteme

*Bussysteme* sind Systeme, die verwendet werden zur seriellen Datenübertragung zwischen einen oder mehreren Komponenten. Beispiele hierfür sind der I<sup>2</sup>C Bus, der Serial Peripheral Interface (SPI) Bus oder auch der Controller Area Network (CAN) Bus. Eine genauere Beschreibung der Bussysteme folgt in Abschnitt 2.4.

#### Raspberry Pi

Der *Raspberry Pi (RPI)* ist ein SBC, der von der britischen *Raspberry Pi Foundation* aus Komponenten von Android-Smartphones entwickelt wurde.

#### Raspbian

*Raspbian* ist ein Betriebssystem, welches auf der Linux Distribution Debian basiert und speziell auf den Raspberry Pi angepasst wurde.

## RRDTool

Das *RRDTool* ist ein Programm, mit dem man Round-Robin Datenbanken erstellen kann. Diese Datenbanken eignen sich besonders gut für die Aufzeichnung von zeitlich fortlaufenden Datenreihen wie z.B. Temperaturmessungen oder Strommessungen. Die Datenbank liegt dabei in einem einzigen File auf dem Datenträger und hat ab dem Erstellen eine feste Größe, die sich auch bei vielen Messungen über einen längeren Zeitraum nicht vergrößert.

## General Purpose Input/Output

Unter *General Purpose Input / Output (GPIO)* versteht man elektrische Kontakte, die zur Realisierung verschiedener Funktionen für elektronische Geräte verwendet werden. Eine genauere Erklärung erfolgt in Abschnitt 2.3.2 [11].

## Python

*Python* ist eine Programmiersprache, die vor allem auf dem Raspberry Pi bevorzugt verwendet wird.

## Bluetooth Low Energy

*Bluetooth Low Energy (BLE)* ist eine Funktechnologie, die es ermöglicht, dass sich Geräte in unmittelbarer Entfernung zueinander vernetzen lassen. Die Stromkosten bei BLE sind um einiges geringe als bei herkömmlichen Bluetooth [12].

## Windows 10 IoT

*Windows 10 IoT* ist eine „abgespeckte“ Version von Windows 10, die auf Mobilen Geräten wie dem Raspberry Pi lauffähig ist. IoT steht dabei für *Internet of Things* [11].

## 2.2 Vergleich von verschiedenen SBC

In diesem Abschnitt werden einige der bekanntesten SBCs, die sich auf dem Markt befinden miteinander verglichen, um den bestmöglichen für die vorgegebenen Anforderungen auswählen zu können. Wichtige Kriterien für die Auswahl sind, dass die Möglichkeit besteht verschiedene Betriebssysteme (Windows und Linux) mit dem jeweiligen Einplatinenrechner betreiben zu können, sowie die verbaute Hardware(Taktfrequenz des Chips, RAM-Speicher). Ein weiterer Punkt welcher von Bedeutung ist, ist die Unterstützung von verschiedenen Kommunikationsschnittstellen (I<sup>2</sup>C, SPI, 1-Wire), um eine

große Anzahl von Sensoren nutzen zu können. Eine Übersicht der einzelnen Komponenten der verschiedenen SBCs sind in den Tabellen 2.1 und 2.2 dargestellt.

<i>SBC</i>	<i>Operating System</i>	<i>RAM</i>	<i>CPU</i>
Banana Pi	Linux, Android	1 GB	ARM Cortex-A7, 1 GHz
Raspberry Pi3	Windows, Linux	1 GB	ARM Cortex-A53 1,2 GHz
BeagleBone Black	Linux	512 MB	ARM Cortex-A8 1 GHz
HummingBoard i2eX	Linux, Android	1 GB	ARM Cortex-A9 1 GHz
Intel Galileo Gen 2	Windows, Linux	256 MB	x86 Quark 400 MHz
Radxa Rock	Linux	2 GB	ARM Cortex-A9 1,6 GHz

Tabelle 2.1: Vergleich OS, RAM, CPU verschiedener SBCs

<i>SBC</i>	<i>Communication</i>	<i>Networking</i>	<i>GPIO</i>
Banana Pi	I <sup>2</sup> C, SPI	1 GigE	80
Raspberry Pi3	I <sup>2</sup> C, SPI	10/100 Mbps <sup>1</sup>	40
BeagleBone Black	I <sup>2</sup> C, SPI	10/100 Mbps	66
HummingBoard i2eX	I <sup>2</sup> C, SPI	1 GigE	8
Intel Galileo Gen 2	I <sup>2</sup> C, SPI	10/100 Mbps	20
Radxa Rock	I <sup>2</sup> C, SPI <sup>2</sup>	10/100 Mbps	80

<sup>1</sup> mit WLAN on Board

<sup>2</sup> nur für Android

Tabelle 2.2: Vergleich Schnittstellen, Netzwerkverbindung, Anzahl GPIO Pins

Wie aus den Tabellen ersichtlich ist, unterstützen die meisten aktuellen SBCs das Betriebssystem Linux. Eine Anforderung für diese Projekt war allerdings, dass sowohl ein Linux System, wie auch ein Windows System auf dem Board lauffähig ist. Aus diesem

Grund fiel die Wahl auf den RPI 3, da dieser beide Betriebssysteme unterstützt und auch bei den anderen betrachteten Aspekten wie *RAM*, *CPU* etc. den meisten Boards ebenbürtig oder sogar überlegen ist. Ein weiterer wichtiger Entscheidungsgrund für den RPI 3 war, dass es für diesen eine sehr große Anzahl an unterstützten Sensoren gibt (Sensoren die mit einer elektrischen Spannung von 3,3 V - 5 V) betrieben werden. Dies ermöglicht einen sehr weit gefächerten Einsatz des RPI 3 und ist für das vorgesehene Projekt von großer Bedeutung.

## 2.3 Raspberry Pi 3

Das folgende Kapitel beschreibt den Raspberry Pi 3 und befasst sich genauer mit den verbauten Komponenten, welche im weiteren Verlauf der Arbeit benötigt werden.

### 2.3.1 Allgemeine technische Daten

Die im folgenden Information wurden, soweit nicht anders angegeben von der offiziellen Homepage der **Raspberry Pi Foundation** entnommen [13].

Der RPI 3 ist ein Kreditkarten großer Einplatinenrechner, welcher aktuell an die *sieben Millionen* Mal verkauft wurde [14]. Dieser besitzt...

- einen 1.2 GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- BLE
- 4 USB Ports
- 40 GPIO Pins
- Full HDMI Port
- Ethernet Port
- Camera Serial Interface (CSI)
- Display Serial Interface (DSI)
- Micro SD Karten Slot

Die aktuelle Version des Raspberry Pi, der RPI 3 ist in Abbildung 2.1 dargestellt.



Abbildung 2.1: Raspberry Pi 3 [1]

Durch die im Gegensatz zu den Vorgänger Modellen leistungsstärkere CPU mit einem 64-Bit quad-core Prozessor, ist es beim RPI 3 möglich das Windows Betriebssystem *Windows 10 IoT* auf dem Gerät zu betreiben, wodurch sich eine Erweiterung der Einsatzmöglichkeiten (nicht mehr nur auf Linux Betriebssysteme beschränkt) ergibt. Auch wurde im Gegensatz zu den vorherigen Modellen beim aktuellen ein WLAN Modul (2,4 GHz) gleich auf der Platine verbaut und muss nicht mehr extra durch ein externes USB-WLAN Modul realisiert werden. Der RPI3 bietet weiterhin einen 10/100 MBit Ethernet Anschluss sowie einen CSI und DSI Anschluss zur direkten Anbindung einer Kamera oder Displays. Die verschiedenen GPIO-Pins werden in Kapitel 2.3.2 genauer beschrieben.

### 2.3.2 GPIO-Kontakte

Wie bereits in Abschnitt 2.3.1 angesprochen besitzt der RPI3 40 GPIO-Kontakte, welche in einer Ecke der Platine zu 2 x 20 Kontakten angeordnet sind. Diese haben einen Rasterabstand von 2,54 mm zueinander und stellen die Grundlage für viele Projekte dar. Die ersten Modelle des RPI besaßen dagegen nur 26 Pins die zur Verfügung standen. Die GPIO-Pins sind elektrische Kontakte, die zur Messung und Steuerung von elektronischen Geräten wie z.B. Sensoren, Analog Digital Wandlern, LEDs etc. verwendet werden.

Die Steckerleiste beinhaltet einige allgemein verwendbare Pins (= *General Purpose Input / Output*), sowie zwei verschiedene Spannungsversorgungen ( $3,3\text{ V}$  und  $5\text{ V}$ ) und Masse Anschlüsse ( $0\text{ V}$ ). Weiterhin beinhaltet die Steckerleiste Kontakte für den I<sup>2</sup>C-, SPI- und 1-Wire-Bus. Bei der Verwendung der GPIO-Kontakten für verschiedene Projekte, muss darauf geachtet werden, welche Bezeichnung verwendet wird. Diese ist in vielen Literaturen verschieden angegeben, da es drei verschiedene Möglichkeiten der Bezeichnung gibt. Die Benennung kann durch

- die physikalische Pin-Nummer, anhand seine Position auf dem Board (von oben gesehen, Pin 1 besitzt eine quadratische Lötstelle)
- die BCM-Pin-Nummer, welche sich auf die Nummerierung der offiziellen Dokumentation des BCM2836-Chips bezieht
- den Pin Namen, welcher von den RPI Entwicklern vergeben wurde vorgenommen werden [11].

In Abbildung ist die Pin Belegung des RPI grafisch dargestellt.

Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I <sup>2</sup> C ID EEPROM)		(I <sup>2</sup> C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Abbildung 2.2: GPIO-Header [2]

## 2.4 Bussysteme

Im folgenden Kapitel werden die am häufigsten verwendeten Bussysteme für die verschiedenen Sensoren erläutert. Zu diesen Bussystemen gehören der *1-Wire*, *I<sup>2</sup>C* und der *SPI* Bus. Auch wird auf die Funktion, sowie die Eigenheiten der Datenübertragung des jeweiligen Busses eingegangen.

### 2.4.1 1-Wire

Der 1-Wire Bus ist ein serielles Bussystem von der Firma *Dallas*<sup>1</sup>, bei dem die Daten seriell (nacheinander) über eine Datenleitung übertragen werden.

#### 2.4.1.1 Allgemeine Informationen

Für dieses Bussystem wird nur eine Datenleitung benötigt, die auch als Spannungsversorgung für den / die jeweiligen Sensoren benutzt wird. Physikalisch werden allerdings zwei Leitungen benötigt, da die Masse auch mitgeführt werden muss. Es gibt für diesen Bus eine große Anzahl an Sensoren wie z.B. Temperatursensoren, die sich durch einen sehr geringen Stromverbrauch auszeichnen. Dies kommt daher, da für die Datenübertragung und die Stromversorgung die gleiche Leitung genutzt wird, wird während der Kommunikation der Sensor aus einem internen Kondensator gespeist. Allerdings kann es notwendig sein, bei Sensoren wo die interne Spannungsversorgung nicht ausreichend ist eine extra Spannungsversorgung für den jeweiligen Sensor mitzuführen.

Diese System ist ein *One-Master-Multi-Slave* Bussystem, was bedeutet, dass es nur einen Master (z.B. RPI) gibt aber mehrere Slaves (z.B. Sensoren). Die Aufgabe des Masters in diesem System ist es, die Kommunikation zu steuern. Die Anzahl der Sensoren kann bis zu 100 betragen, die parallel an den Master angeschlossen werden. Dies ist möglich, da jeder Sensor ein eindeutige 64 Bit lange ID besitzt. Diese gliedert sich in [8]

- 8 Bit *Family Code*
- 48 Bit *Seriennummer*
- 8 Bit *CRC-Prüfsumme*

---

<sup>1</sup>2001 von Maxim Integrated übernommen

### 2.4.1.2 Übertragungsprotokoll

Der 1-Wire Bus wird dadurch, dass er keine Taktsignal benötigt als *asynchroner* Bus bezeichnet. Dieser kommuniziert im *Halbduplex* Verfahren, was bedeutet, dass immer nur ein Teilnehmer auf dem Bus senden oder empfangen kann (entweder Master oder ein Slave). Wenn keine Kommunikation stattfindet, wird die Datenleitung über einen Pullup-Widerstand auf *high* gezogen und der in Abschnitt 2.4.1.1 erwähnte interne Kondensator geladen. In dem andern Fall wenn eine Übertragung stattfindet liegt die Datenleitung auf Masse und der Kondensator liefert in diesem Fall die Spannungsversorgung für den Sensor (abhängig vom Sensor siehe 2.4.1.1). Da wie schon angesprochen keine Taktleitung vorhanden ist, muss für die Kommunikation ein bestimmter Ablauf eingehalten werden. Dafür gibt es zwei Übertragungsmöglichkeiten, den *normalen Modus* wo ca.  $16,3 \text{ kBit/s}$  übertragen werden und den *Overdrive Modus* mit bis zu  $142 \text{ kBit/s}$ . Die Zeitspanne für die Übertragung von 1 Bit beträgt immer  $60 \mu\text{s}$ . Diese Steuerung, egal in welche Richtung die Übertragung stattfindet werden durch den Master initiiert. Die Befehle die dazu notwendig sind, können aus der Tabelle 2.3 entnommen werden [8].

<i>Befehl</i>	<i>Beschreibung</i>
<b>Write 1</b>	Der Master zieht für $1 - 15 \mu\text{s}$ auf Low. Der Rest des Slots bleibt ungenutzt.
<b>Write 0</b>	Der Master zieht den Bus für mindestens $60 \mu\text{s}$ bis maximal $120 \mu\text{s}$ auf Low.
<b>Read</b>	Der Master zieht für $1 - 15 \mu\text{s}$ auf Low. Der Slave, der kommunizieren möchte, hält für eine 0 den Bus weiter auf Low. Will der Slave eine 1 senden, gibt er direkt den Bus wieder frei. Wie man leicht erkennt, ist der Status <i>Write 1</i> oder <i>Read</i> für den Master gleich. Alleine der Status des Sensors bestimmt, ob ein <i>Read</i> oder <i>Write 1</i> ausgeführt wird.
<b>Reset / Presence</b>	Der Master zieht den Bus für min. $480 \mu\text{s}$ auf Low. Wenn ein Slave am Bus vorhanden ist, zieht er max. $60 \mu\text{s}$ (also einen Slot) die Leitung auf Low. Somit weiß der Master, dass mindestens ein Slave angeschlossen ist.

Tabelle 2.3: Befehle bei 1-Wire [8, S. 35]

## 2.4.2 I<sup>2</sup>C-Bus

Der I<sup>2</sup>C Bus ist auch noch unter einem anderen Namen als Two-Wire-Interface (TWI) Bus (z.B. bei Atmel) bekannt. Er wurde 1982 von der Firma Philips Semiconductors<sup>2</sup> entwickelt und wird vornehmlich zur internen Kommunikation von Geräten benutzt [8].

### 2.4.2.1 Allgemeine Informationen

Technisch gesehen sind der I<sup>2</sup>C und TWI Bus identisch, die Unterscheidung wird lediglich aus Lizenz rechtlichen Gründen getroffen. Bei dem Bus handelt es sich um einen *Master-Slave-Bus*, allerdings ist auch ein *Multi-Master*<sup>3</sup> Betrieb möglich. Der Beginn einer Kommunikation wird immer vom Master initiiert, bei dem angesprochenen Multi-Master Betrieb, arbeitet dann der vom Initiator angesprochene Master wie ein Slave. Im Gegensatz zum 1-Wire Bus sind zwei Leitungen notwendig, eine Datenleitung (SDA) und eine Takteleitung (SCL). Da der I<sup>2</sup>C Bus eine Takteleitung besitzt, spricht man hier von einem synchronen seriellen Bussystem. Der Systemtakt wird bei diesem System immer vom Master vorgegeben [8].

### 2.4.2.2 Definierte Zustände und Adressierung

Bei der Kommunikation mittels I<sup>2</sup>C müssen bestimmte Zustände eingehalten werden um die korrekte Funktion sicherzustellen. Diese werden im folgenden dargestellt. Für die folgenden Definitionen dient falls nicht anders angegeben folgende Quelle [3].

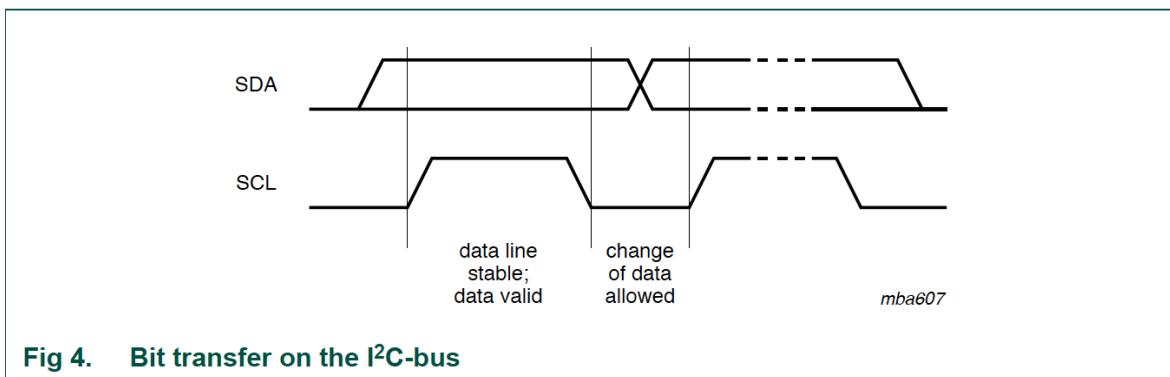
#### Gültigkeit Datenbit

Damit ein Bit auf der Datenleitung als gültig betrachtet wird, darf sich dessen Pegel während der High Phase der Takteleitung nicht ändern. Dieser Zustand ist in Abbildung 2.3 dargestellt.

---

<sup>2</sup>heute NXP

<sup>3</sup>es gibt mehr als einen Master



**Fig 4. Bit transfer on the I<sup>2</sup>C-bus**

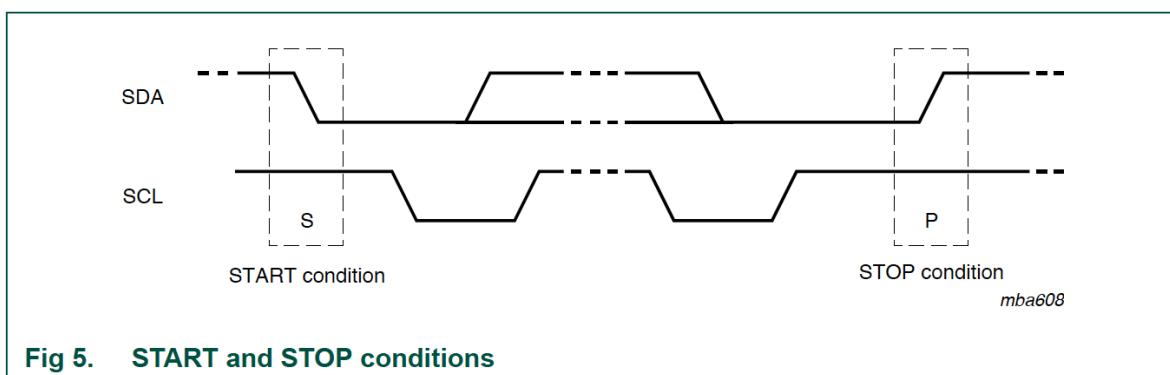
Abbildung 2.3: Bedingung für gültiges Bit auf Datenleitung [3, S. 9]

### START Bedingung

Damit eine Kommunikation stattfinden kann muss diese als erstes einmal gestartet werden. Dies geschieht, indem der Master die SDA Leitung auf Ground zieht, während die SCL Leitung auf HIGH gesetzt ist. Dieser Zustand wird in Abbildung noch einmal grafisch dargestellt

### STOP Bedingung

Um die Übertragung zu beenden, ist die STOP Bedingung definiert. Diese unterscheidet sich von der START Bedingung dadurch, dass der Master bei HIGH Zustand der Taktleitung die Datenleitung ebenfalls auf HIGH zieht. Grafisch ist dies in Abbildung dargestellt.



**Fig 5. START and STOP conditions**

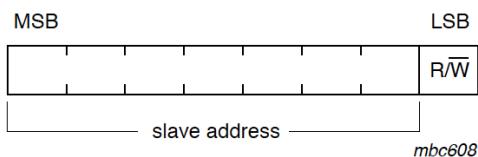
Abbildung 2.4: START, STOP Bedingung I<sup>2</sup>C [3, S. 9]

## Adressierung

Bei der Adressierung wird nach der Start Bedingung 1 Byte vom Master an den Slave gesendet. Bei diesem Byte stellen die ersten sieben Bits die Adresse des Slaves dar und das achte Bit das Read / Write Bit. Die Kennzeichnung des R / W Bits ist immer aus Sicht des Masters zu deuten. Dies bedeutet, dass bei WRITE Daten vom Master an den Slave übertragen werden und bei READ Daten vom Slave an den Master. Abbildung 2.6 verdeutlicht noch einmal den Aufbau des ersten Bytes.

Dadurch, dass sieben Bits für die Adresse des Slaves verwendet werden können und 16 Adressen davon für Erweiterungen reserviert sind, können bis zu maximal 112 Teilnehmer (Berechnung entsprechend Formel 2.1 ) an den Bus angeschlossen werden.

$$\text{max. Teilnehmer} = 2^7 - 16 = 112 \quad (2.1)$$



**Fig 10. The first byte after the START procedure**

Abbildung 2.5: Erstes Byte nach Start Bedingung I<sup>2</sup>C [3, S. 13]

### 2.4.2.3 Übertragungsprotokoll

Das Übertragungsprotokoll beim I<sup>2</sup>C Bus ist immer so aufgebaut, dass die Kommunikation mit einem Start Signal beginnt und als nächstes das Byte mit der Slave Adresse und dem R / W Bit folgt. Jedes Byte das gesendet wird, wird durch den Slave mit einem ACK-Bit quittiert. Je nachdem wie das R / W Bit gesetzt ist (READ = 1, WRITE = 0) werden die Daten byteweise gelesen oder geschrieben. Das ACK Bit wird dann entweder vom Master oder vom Slave gesendet. Ist das letzte Byte der Übertragung gesendet, wird dieses vom Master mit einem NACK quittiert. Im Anschluss wird die Kommunikation durch die STOP Bedingung beendet. In Abbildung ist der Ablauf noch einmal grafisch dargestellt.

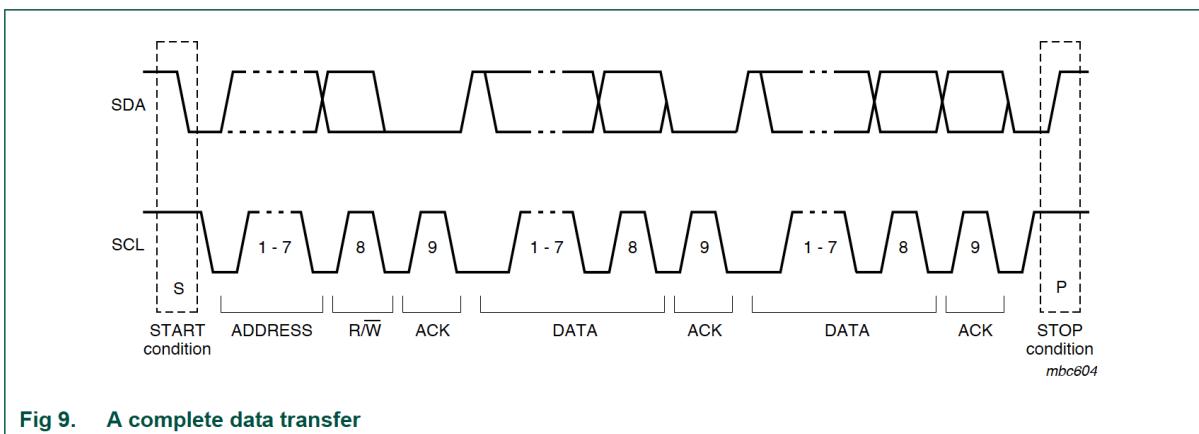


Fig 9. A complete data transfer

Abbildung 2.6: Datentransfer I<sup>2</sup>C Bus [3, S. 13]

## 3 Praktischer Teil

In diesem Kapitel werden die verschiedenen Sensorschaltungen und deren Konfiguration, die Problem die sich bei der Realisierung ergaben, sowie die Ergebnisse und Auswertungen beschrieben. Außerdem werden die verschiedenen Möglichkeiten zur Datenspeicherung und Visualisierung dargestellt und erklärt. Am Anfang dieses Kapitels werden die zur Realisierung der verschiedenen Aufgaben benötigten Bauteile, kurz etwas näher erklärt und aufgelistet. In Abschnitt 3.2 wird eine Schaltung mit einem einzigen 1-Wire Temperatursensor aufgebaut, bei der zweiten Schaltung in Abschnitt 3.3 kommt zu dem Sensor aus 3.2 ein zweiter Temperatursensor und ein Sensor zur Temperaturmessung und Bestimmung der Luftfeuchtigkeit hinzu. Der letzte Abschnitt (3.4) befasst sich mit der Realisierung einer Vibrationsmessung mittels Beschleunigungssensors.

### 3.1 Benötigte Materialien

Die in Tabelle aufgelisteten Materialien wurden für die in Kapitel 3 beschriebenen Versuche verwendet. In den jeweiligen Abschnitten, werden die einzelnen Sensoren mit deren wichtigsten Technischen Daten noch genauer erklärt.

<i>Bezeichnung</i>	<i>Anzahl</i>
Raspberry Pi 3	1
Temperatursensor DS18S20	2
Sensor HYT 221	1
3-Achsen-Beschleunigungssensor	1
Drahtbrücken	mehrere
elektrischer Widerstand $4.7\text{ k}\Omega$	1
Elektronik Steckbrett	1

Tabelle 3.1: benötigte Materialien

## 3.2 Temperaturmessung mit Sensor DS18S20

Abschnitt 3.2 befasst sich mit dem Schaltungsaufbau zur Temperaturmessung mittels DS18S20. Weiterhin werden die verschiedenen Möglichkeiten zur Datenspeicherung (RRD Tool, MySQL), sowie eine Möglichkeit zur Visualisierung der Daten erläutert.

### 3.2.1 DS18S20

Der DS18S20 (siehe Abbildung 3.2) ist ein digitaler 1-Wire Temperatursensor, der eine 9 Bit Temperaturmessung ermöglicht. Dieser ist von der Firma *Maxim Integrated* entwickelt worden. Folgend werden die wichtigsten technischen Daten des DB18S20 aufgeführt.

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS	
Supply Voltage	$V_{DD}$	Local Power (Note 1)		+3.0		+5.5	V	
Pullup Supply Voltage	$V_{PU}$	Parasite Power	(Note 1, 2)	+3.0		+5.5	V	
		Local Power		+3.0		$V_{DD}$		
Thermometer Error	$t_{ERR}$	-10°C to +85°C	(Note 3)			$\pm 0.5$	°C	
		-55°C to +125°C				$\pm 2$		
Input Logic-Low	$V_{IL}$	(Note 1, 4, 5)		-0.3		+0.8	V	
Input Logic-High	$V_{IH}$	Local Power	(Note 1, 6)	+2.2	The lower of 5.5 or $V_{DD}$ + 0.3		V	
		Parasite Power		+3.0				
Sink Current	$I_L$	$V_{I/O} = 0.4V$ (Note 1)		4.0			mA	
Standby Current	$I_{DDS}$	(Note 7, 8)		750		1000	nA	
Active Current	$I_{DD}$	$V_{DD} = 5V$ (Note 9)		1		1.5	mA	
DQ Input Current	$I_{DQ}$	(Note 10)		5			μA	
Drift		(Note 11)				$\pm 0.2$	°C	

**Note 1:** All voltages are referenced to ground.

**Note 2:** The Pullup Supply Voltage specification assumes that the pullup device is ideal, and therefore the high level of the pullup is equal to  $V_{PU}$ . In order to meet the  $V_{IH}$  spec of the DS18S20, the actual supply rail for the strong pullup transistor must include margin for the voltage drop across the transistor when it is turned on; thus:  $V_{PU\_ACTUAL} = V_{PU\_IDEAL} + V_{TRANSISTOR}$ .

**Note 3:** See typical performance curve in [Figure 1](#).

**Note 4:** Logic-low voltages are specified at a sink current of 4mA.

**Note 5:** To guarantee a presence pulse under low voltage parasite power conditions,  $V_{ILMAX}$  may have to be reduced to as low as 0.5V.

**Note 6:** Logic-high voltages are specified at a source current of 1mA.

**Note 7:** Standby current specified up to +70°C. Standby current typically is 3μA at +125°C.

**Note 8:** To minimize  $I_{DDS}$ , DQ should be within the following ranges: GND ≤ DQ ≤ GND + 0.3V or  $V_{DD} - 0.3V \leq DQ \leq V_{DD}$ .

**Note 9:** Active current refers to supply current during active temperature conversions or EEPROM writes.

**Note 10:** DQ line is high ("high-Z" state).

**Note 11:** Drift data is based on a 1000-hour stress test at +125°C with  $V_{DD} = 5.5V$ .

Abbildung 3.1: elektische Daten DB18S20 [4, S. 2]



Abbildung 3.2: DS18S20 [5]

### 3.2.2 Schaltungsaufbau

Abbildung 3.3 zeigt den schematischen Schaltungsaufbau mit dem Temperatursensor DS18S20. Anzumerken ist, dass die in der Schaltung in Abbildung 3.3 dargestellten Bauteile optisch nicht immer den realen Bauteilen entsprechen (z.B. Aufdruck auf dem Sensor). Die Beschaltung der Bauteile wurde in der praktischen Erprobung ganz genau so durchgeführt.

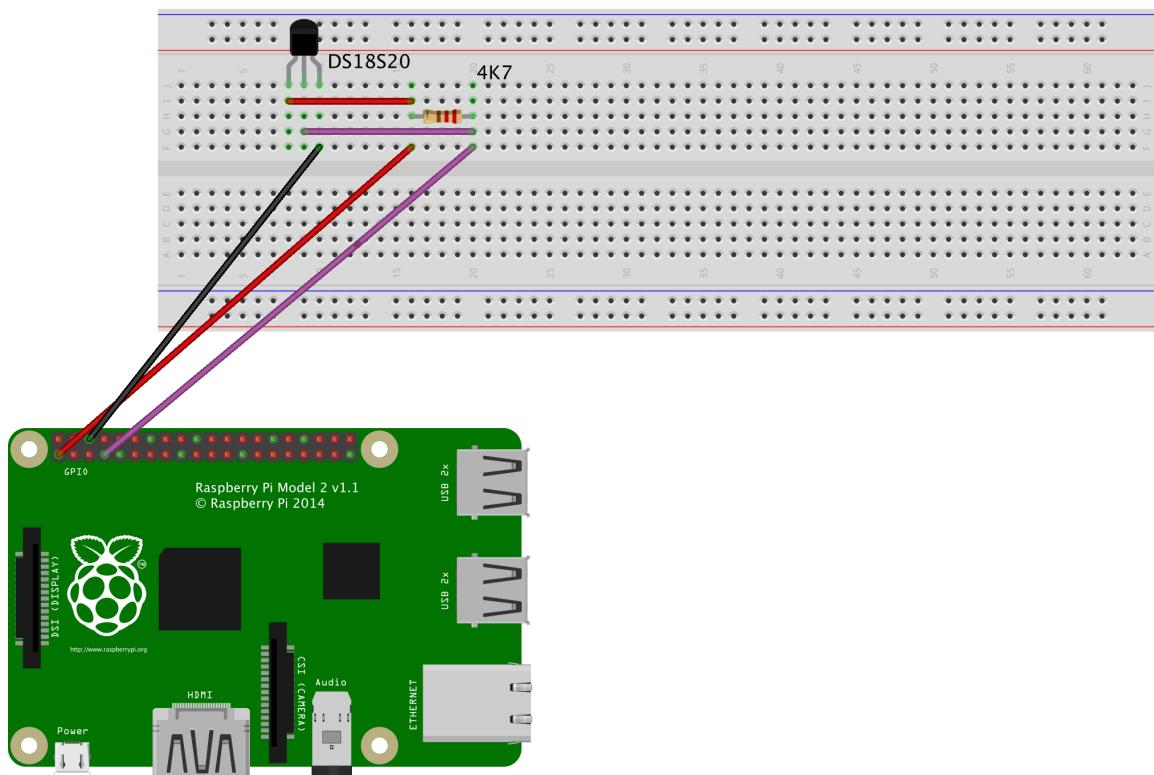


Abbildung 3.3: Schaltungsaufbau DS18S20

Die oben dargestellte Schaltung<sup>1</sup> zur Temperaturmessung wird folgendermaßen realisiert, der Pin  $V_{DD}$  des Sensors ist mit Pin 1 (3,3 V) des RPI 3 verbunden (rote Verbindung). Die schwarze Verbindung stellt die GROUND Verbindung des Pin 6 des RPI mit dem GND Pin des Sensors dar. Weiterhin ist der er DQ Pin des Sensors (1-Wire Interface) mit dem Pin 7 des RPI verbunden (violette Verbindung) welcher für den 1-Wire Bus verwendet wird. Zu dem  $V_{DD}$  Pin und dem DQ Pin ein Pullup Widerstand parallel geschalten.

---

<sup>1</sup>gezeichnet mit Fritzing

### 3.2.3 Auslesen des Sensors

Um die vom Sensor erfassten Daten auf dem RPI3 auszulesen, wird ein Python Script verwendet. Dieses wird im Anschluss an Hand kurzer Ausschnitte aus dem Quellcodes erklärt. Der komplette Quellcode ist im Anhand dieser Arbeit zu finden.

```

1   file = open('/sys/bus/w1/devices/' + str(w1_slave) + '/
2     w1_slave')
3   filecontent = file.read()
4   file.close()
5
6   stringvalue = filecontent.split("\n")[1].split(" ")[9]
7   temperature = float(stringvalue[2:]) / 1000
8   temperature = round(temperature, 2)
9   print(str(w1_slave) + ': %6.2f C' % temperature)
10 ###### Sensor-Daten speichern #####
11
12 ret = rrd_update('/home/pi/Temperatur/TemperaturSensor/
13   TemperaturAufzeichnungDB18S20.rrd', 'N:%s' % (temperature));
14 sys.exit(0)
```

Listing 3.1: Auslesen der gemessenen Temperatur

Um die Temperatur des Sensors auslesen zu können wird als erstes das File wie in Zeile 2 des Codes ersichtlich ist geöffnet und dessen Inhalt in eine Variable gespeichert. Dieses File enthält die gesamten an den RPI3 angeschlossenen IDs der 1-Wire Sensoren. Jeder angeschlossenen 1-Wire Sensor besitzt im Pfad `/sys/bus/w1/devices/` einen Ordner mit seiner ID. In diesem gibt es wiederum eine Datei mit der Bezeichnung `w1_slave`, die die gemessenen Daten enthält. Um diese auszulesen, werden in unseren Python Script in den Zeilen 8 – 15 verschiedene Schritte durchgeführt, auf die nicht näher eingegangen wird<sup>2</sup>. Da die Temperatur in dieser Datei vom Sensor im Format von z.B. 17687 ( $\approx 17,7^{\circ}\text{C}$ ) angegeben wird, muss dieser Wert noch durch 1000 dividiert werden um den genauen Temperaturwert zu erhalten(Zeile 15 im Quellcode). In Zeile 16 wird der Wert noch auf zwei Stellen nach dem Komma gerundet und ist in der Variable `temperature` gespeichert. Mit dem Wert dieser Variable kann nun weiter gearbeitet werden.

---

<sup>2</sup>kann in verschiedenen Python Tutorials nachgelesen werden

### 3.2.4 Datenspeicherung

Wie schon in Abschnitt 3.2 angesprochen, wird in dieser Arbeit auf zwei verschiedene Möglichkeiten der Datenspeicherung eingegangen. Zum Einen mit dem RRDtool und zum Andern mittels mySQL Datenbank. Die Möglichkeit der Datenspeicherung in einer mySQL Datenbank wird im Abschnitt 3.3 beschrieben. In diesem Abschnitt wird näher auf die Möglichkeiten mit dem RRDtool eingegangen.

#### RRDtool

Das RRDtool ist ein Programm, welches es ermöglicht, Messdaten für einen beim Erstellen festgelegten Zeitbereich zu speichern. RRD steht für *Round-Robin-Database*, diese wird durch das Tool in einer Datei erzeugt, die von der Erstellung weg eine feste Größe besitzt. Dies ist ein großer Unterschied zu herkömmlichen mySQL Datenbanken, da bei diesen mit zunehmender Zeit und Datenmenge auch die Größe der Datenbank anwächst. Das Anwachsen der Größe der Datei wird bei der RRD Datenbank dadurch verhindert, dass sie wie ein Ringbuffer arbeitet, was bedeutet, dass wenn der Speicherplatz belegt ist, Werte zusammengefasst (z.B. durch Bildung eines Mittelwertes für einen Tag aus den einzelnen stündlichen Werten) und die ältesten überschrieben werden. Somit kann sichergestellt werden, dass die Größe der Datenbank schon beim Erstellen dieser festgelegt ist.

Wie eine solche Datenbank definiert und erstellt wird, wird im Quellcode 3.2 beschrieben. Zum Erstellen der Datenbank wurde ein Shell-Script verwendet um nicht jedes Mal wenn die Datenbank neu erstellt wird alle Befehle einzeln eingeben zu müssen.

#### Erstellen der Datenbank

```

1 #!/bin/sh
2 sudo rrdtool create TemperaturAufzeichnungDB18S20.rrd --step
   60 \
3 DS:Temperatur1:GAUGE:120:-30:120 \
4 RRA:AVERAGE:0.5:1:600 \
5 RRA:MAX:0.5:1440:365 \
6 RRA:MIN:0.5:1440:365 \
7 RRA:AVERAGE:0.5:60:720 \
8 RRA:AVERAGE:0.5:240:720 \
9 RRA:AVERAGE:0.5:1440:365

```

Listing 3.2: Erstellung RRD Datenbank

Zeile 2 des in 3.2 dargestellten Source Codes erstellt das File, welches als Datenbank

fungiert. Weiterhin wird die Schrittweite<sup>3</sup> auf *60 Sekunden* festgelegt. In der nächsten Zeile wird die Datenreihe festgelegt, die Zahlen am Ende der Zeile legen fest, dass wenn innerhalb von 120 Sekunden kein gültiger Wert geliefert wird, die Datenbank an dessen Stelle *NAN* (Not A Number) einträgt. Die Begründung liegt darin, dass wenn im weiteren Verlauf Mittelwerte etc. gebildet werden keine Verfälschung der Daten erfolgt (würde passieren wenn z.B. 0 eingetragen wird). In Zeile 4 wird der Zeitbereich der Speicherung von den gelieferten Werten festgelegt. In dem dargestellten Beispiel werden jede Minute ein Wert gespeichert und dies 600 Minuten ( $\approx 10$  Stunden) lang. Die Zeilen 5-9 des Quellcodes dienen dazu um die Bereiche von Maximl- bzw. Minimalwerten, sowie die Zeitspannen wann älteren Datenwerte zusammengefasst werden zu definieren. Genauere Informationen bezüglich der Erstellung der Datenbank können von folgender Adresse bezogen werden <http://oss.oetiker.ch/rrdtool/> [15].

Das in Quellcode 3.2 dargestellte Script musste nach dem speichern noch ausführbar gemacht werden. Dabei ergab sich das Problem, dass dieses beim Aufrufen die Datenbank wegen einer fehlenden Berechtigung nicht erstellen konnte. Dies wurde dadurch behoben, dass in Zeile 1 des Scripts der Aufruf des rrdtool mit *sudo* realisiert wurde. Mit dem erfolgreichen Ausführen des Scripts wurde die Datenbank im Verzeichnis /home/pi/Temperatur/TemperaturSensor/ erstellt.

## Datenspeicherung

Damit die über das Python Script wie vorher dargestellt ausgelesenen Daten in die Datenbank geschrieben werden, wurde am Ende des Scriptes folgende Zeile hinzugefügt.

```

1 ##### Sensor-Daten speichern #####
2
3 ret = rrd_update('/home/pi/Temperatur/TemperaturSensor/
    TemperaturAufzeichnungDB18S20.rrd','N:%s'%(temperature));

```

Listing 3.3: Datenspeicherung in DB

Durch diese Codezeile, wird ein Update der Datenbank durchgeführt und der Wert, der in der Variable `temperature` gespeichert ist in diese geschrieben.

---

<sup>3</sup>Datenbank erwartet alle 60 Sekunden einen Wert

### 3.2.5 Visualisieren der Daten

Die grafische Darstellung der Daten wurde wieder mit dem RRDtool realisiert. Hier wurden zwei Grafen erstellt die die gemessenen Temperaturwerte für verschiedene Zeiträume darstellen. Zur Erstellung dieser Grafen wurde ein Script verwendet, das den entsprechenden Code enthält. Der Quellcode zur Erstellung der Graphen ist im Anhang dargestellt. Die einzelnen Befehle können auf folgender Quelle nachgeschlagen werden [http://oss.oetiker.ch/rrdtool/doc/rrdgraph\\_data.en.html](http://oss.oetiker.ch/rrdtool/doc/rrdgraph_data.en.html).

Abbildung 3.4 und 3.5 zeigen die mittels RRDtool erzeugten Graphen.

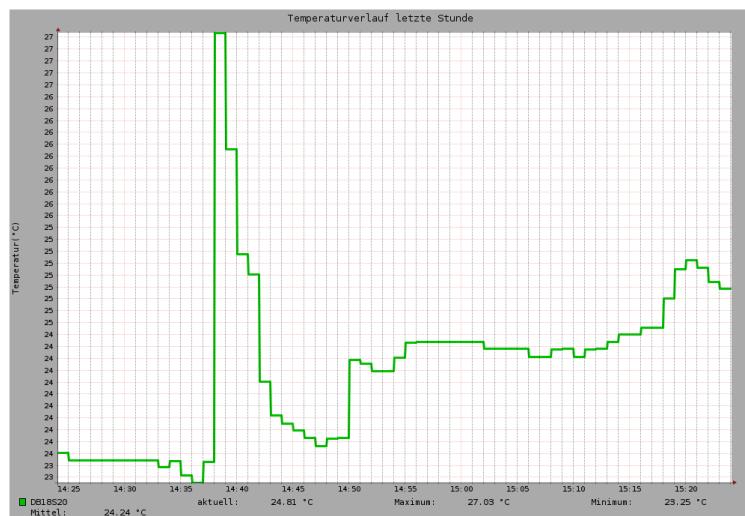


Abbildung 3.4: Temperaturverlauf der letzten Stunde

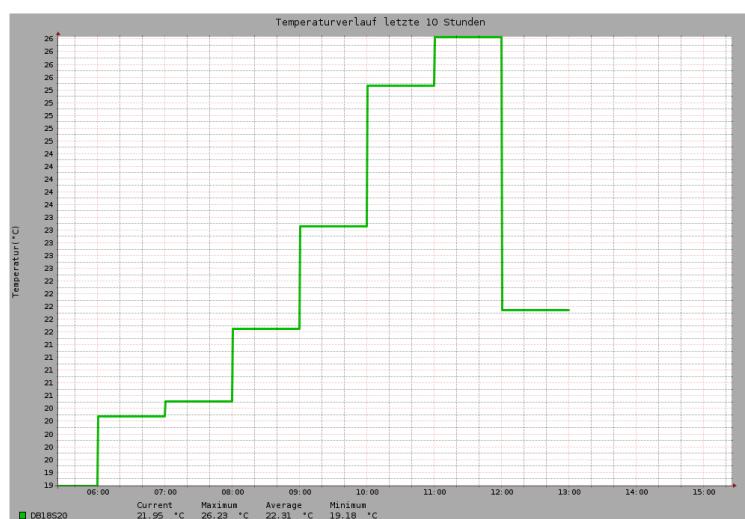


Abbildung 3.5: Temperaturverlauf der letzten 10 Stunden

Um die in diesem Versuch verwendeten Scripte zum Auslesen der Daten, Speichern

der Daten oder Erstellung der Graphen für den Temperaturverlauf nicht immer von „Hand“ ausführen zu müssen, wurde für jedes dieser Scripte ein extra Eintrag in der Datei `crontab`<sup>4</sup> vorgenommen. Die Zeit wurde jeweils auf 1 Minute eingestellt. Als Erkenntnis konnte aus diesem Versuchsaufbau gewonnen werden, dass der Einsatz eines 1-Wire Sensors an dem RPI 3 zur Temperaturaufzeichnung, -speicherung und -visualisierung relativ schnell und problemlos möglich ist. Der Temperaturverlauf konnte an Hand der Graphen gut und übersichtlich nachvollzogen werden.

### 3.3 Temperatur- und Luftfeuchtigkeitsmessung mit HYT-221

Bei diesem Versuchsaufbau, kommen im Gegensatz zu dem Aufbau in Abschnitt 3.2 zwei weitere Sensoren hinzu. Zum Einen wird ein weiterer DS18S20 verbaut und zum Anderen ein HYT-221. Diese drei Sensoren werden gleichzeitig betrieben und liefern so eine gute Möglichkeit die Temperaturwerte zu einer bestimmten Zeit an Hand von unterschiedlichen Sensoren zu vergleichen.

#### 3.3.1 HYT-221

Der HYT-221 ist ein Sensor zur Messung der Temperatur und Luftfeuchtigkeit. Er wurde von der Schweizer Firma INNOVATIVE SENSOR TECHNOLOGY (IST) für die Anwendung in den Bereichen Meteorologie, Industrielle Trocknungstechnik, Medizinische Geräte, Luftfahrt und Extremsport entwickelt. Der Sensor kommuniziert über eine I<sup>2</sup>C Schnittstelle und besitzt eine hohe Messgenauigkeit. In Tabelle 3.2 werden die wichtigsten Daten des Sensors aufgelistet. Diese wurden dem Sensor zugehörigen Datenblatt entnommen [9].

<b>Feuchtemessung</b>		<b>Temperaturmessung</b>	
<i>Bezeichnung</i>	<i>Werte</i>	<i>Bezeichnung</i>	<i>Werte</i>
Messbereich	0...100 % rF	Messbereich	-40...+125 °C
Genauigkeit	± 1,8 % rF	Genauigkeit	± 0,2 °C
Auflösung	0,02 % rF	Auflösung	0.015 °C

Tabelle 3.2: Technische Daten HYT-221 [9]

<sup>4</sup>Cron-Daemon = Dienst der automatisch Scripte zu vorgegebenen Zeiten starten kann



Abbildung 3.6: HYT-221 [6]

Für einen sicheren Betrieb des Sensors muss die angelegte Versorgungsspannung zwischen  $2,7 \dots 5,5\text{ V}$  liegen.

### 3.3.2 Schaltungsaufbau

Der in Abbildung dargestellte Schaltungsaufbau beinhaltet zwei Sensoren der des Typs DS18S20, welche mit dem 1-Wire Bus betrieben werden. Der in Kapitel 3.2.2 erwähnte Pull-Up Widerstand wird dafür allerdings nur einmal benötigt, da beide Sensoren parallel zueinander geschalten sind. Als dritter Sensor ist ein HYT-221 verbaut, welcher Temperaturwerte und die Luftfeuchtigkeit misst. Der HYT-221 wird über den I<sup>2</sup>C Bus betrieben. Dieser benötigt eine  $3,3\text{ V}$  Versorgungsspannung (rote Verbindung), einen GROUND Anschluss (schwarze Verbindung), sowie zur Datenübertragung eine Taktleitung (SCL = rote Verbindung) und eine Datenleitung (SDA = blaue Verbindung).

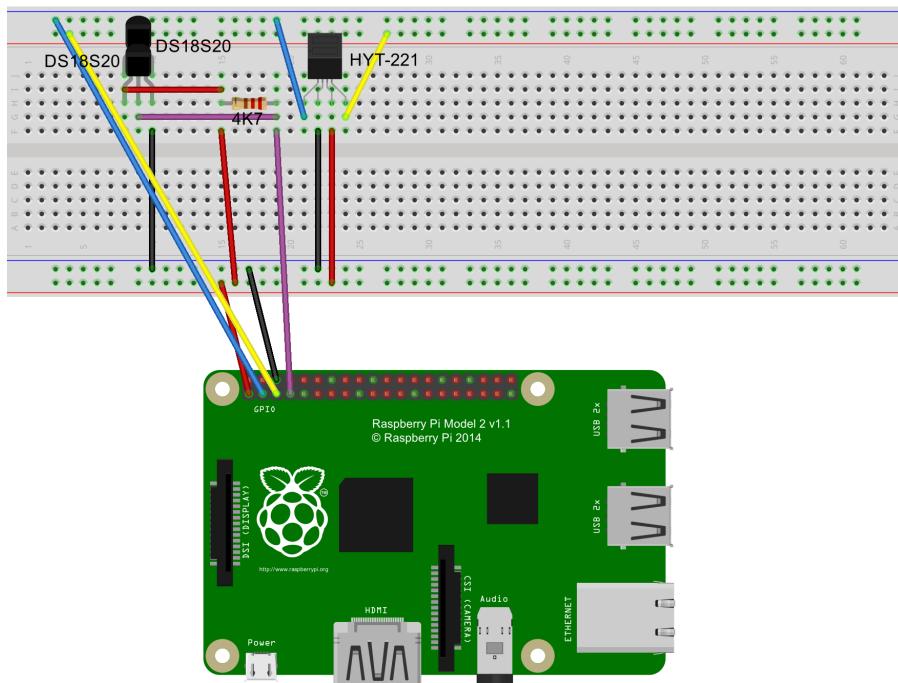


Abbildung 3.7: Schaltungsaufbau mit drei Sensoren

### 3.3.3 Auslesen der Daten

Im folgenden wird der Code zum Auslesen der Daten des HYT-221 über den I<sup>2</sup>C Bus erklärt. Das Auslesen der Daten der beiden anderen Sensoren erfolgt analog zu der in Abschnitt 3.2.3 dargestellten Möglichkeit, daher wird darauf nicht mehr explizit eingegangen. Der komplette Source Code ist in Anhang dargestellt.

```

1 ##### HYT 221 #####
2
3 #I2C-Adresse
4 SensorAdresse = 0x28;
5
6 #Array fuer die Sensordaten
7 SensorDaten = bytearray()
8
9 #Sensor ID fuer HYT221
10 ID_HYT = "HYT_221"
11
12 #smbus Objekt fuer I2C bus #1
13 Sensor = smbus.SMBus(1)
14
15 #Initialisierung Array
16 SensorDaten.append(0x30)
17 SensorDaten.append(0x31)
18 SensorDaten.append(0x32)
19 SensorDaten.append(0x33)
20
21 #Sensor zum lesen initialisieren, Antwort ignorieren
22 ans = Sensor.read_byte_data(SensorAdresse, 0)
23 sleep(0.1)
24
25 #4 Byte der Daten lesen
26 SensorDaten = Sensor.read_i2c_block_data(SensorAdresse, 4)

```

Listing 3.4: Auslesen der Temperatur und Luftfeuchtigkeit

Um das Auslesen der Daten einfach zu gestalten, wurde im Script die Bibliothek *smbus* importiert, welche fertige Funktionen zum Auslesen eines I<sup>2</sup>C Busses bereitstellt.

Als erstes wurde eine Variable mit der Adresse des Sensors angelegt. Diese Adresse kann dem Datenblatt des Sensors entnommen werden oder über den Befehl `i2cdetect -y 1` in der Konsole des RPI3 ausgelesen werden. Der Parameter -y bewirkt, dass der Befehl sofort ausgeführt wird, der Parameter 1 legt fest das es sich um den I<sup>2</sup>C Bus Nummer 1 handelt (andere Hardware kann auch mehrere I<sup>2</sup>C Busse besitzen).

Um den smbus verwenden zu können muss ein Objekt dessen erzeugt werden (Zeile 13), die 1 in der Funktion steht wie schon beschrieben für den Bus Nummer 1. Im Anschluss wird das erzeugte Array zur Speicherung der Daten noch initialisiert. Die Funktion in Zeile 22 erzeugt die wie im Theorieteil beschriebene Startbedingung. Im Anschluss an die Startbedingung werden mit der Funktion `read_i2c_block_data()` die Register, in denen die Werte für die Temperatur und Luftfeuchtigkeit gespeichert sind ausgelesen und in das Array gespeichert. Die Temperatur- und Feuchtigkeitswerte sind in vier Byte aufgeteilt und müssen dementsprechend ausgelesen werden. Dies kann in der Protokollbeschreibung für den Sensor in folgender Quelle nachgelesen werden [16]. Die übertragenen Werte, mussten im Anschluss an Hand von bestimmten Formeln, die dem Datenblatt des Sensors entnommen wurden in die jeweils richtige Form umgerechnet werden, um diese korrekt weiterverarbeiten zu können.

### 3.3.4 Datenspeicherung

Wie schon in Abschnitt 3.2.4 beschrieben, wurden die Daten wieder in eine RRD Datenbank gespeichert. Der Source Code hierfür ist in Anhang A.3 abgebildet.

Weiterhin wurden die Sensordaten in einer mySQL Datenbank abgelegt, um die Möglichkeit zu schaffen, zu einem späteren Zeitpunkt durch exportieren dieser Daten (z.B. als CSV-File) eine weitere Bearbeitung zu ermöglichen. Möglichkeiten hierfür wären z.B. bei einer Fehlproduktion zu einem bestimmten Zeitpunkt (der die Aufzeichnungszeit des RRDtools übersteigt) eine Aussage über eventuelle Faktoren die dafür verantwortlich sein könnten tätigen zu können.

Der Source Code zur Realisierung der Datenspeicherung mittels mySQL DB ist im folgenden ersichtlich. Die entsprechende DB wurde zuvor auf dem RPI3 erstellt. Zur einfacheren Nutzung der mySQL Datenbank in dem Python Script wurde die Bibliothek `mysql.connector` importiert. Diese stellt vordefinierte Funktionen für die Verbindung mit der DB zur Verfügung, sowie Funktionen zum Lesen und Schreiben in bzw. aus der DB. Die Herstellung einer Verbindung zur Datenbank ist in Listing 3.5 ersichtlich. Wichtig dabei war, die korrekten Anmeldedaten für die Datenbank zu verwenden, da ansonsten die Verbindung nicht hergestellt werden konnte und eine Fehlermeldung ausgegeben wurde.

```

1 #Datenbank Verbindung herstellen
2 try:
3     db = mysql.connector.connect(host="localhost", user="root",
4                                 passwd="test123", db="RPI-Projekt")
4 except :
```

---

```

5     print("No connection to database")
6     exit(0)

```

Listing 3.5: Verbindung zur DB herstellen

Um auf die Tabelle in der Datenbank zugreifen zu können musste noch ein Cursor erzeugt werden (Zeile 4). An Hand von SQL Statements wurden dann die Daten in die entsprechenden Spalten der Tabelle eingefügt (Zeile 9-10 und Zeile 13-14). Um die Änderungen in der Tabelle wirksam zu machen, wurde in Zeile 16 noch ein commit durchgeführt und im Anschluss die Verbindung zu Datenbank wieder geschlossen.

```

1 ##### Daten in mySQL speichern
2 #####
3 #Cursor zum Eintragen erzeugen
4 cur = db.cursor()
5
6 #SQL-Statement erzeugen
7 sql = "INSERT INTO Temperatur (DB18S20, DB18S20K, HYT22)
8     VALUES (%s,%s,%s)"
9 cur.execute(sql, (Temperature_W1[1],Temperature_W1[0],
10                 HYT_Temperature))
11
12
13 db.commit()
14 db.close()

```

Listing 3.6: Speicherung der Daten in mySQL DB

### 3.3.5 Visualisierung der Daten

Die Messdaten wurden wie im Abschnitt 3.2.5 mit dem RRDtool visualisiert. Die Graphen mit den Temperatur und Feuchtigkeitswerten sind für die verschiedenen Zeitspannen im Anhang B.1 aufgelistet. Ebenso ist der Quellcode für die Erstellung der Graphen in A.5 ersichtlich.

### 3.4 Vibrationsmessung mit Sensor BMA020

Im dritten Versuchsaufbau sollte mit dem Beschleunigungssensor BMA020 die Vibration gemessen werden. Verwendet wurde hier eine schon fertig bestückte Platine die den BMA020 enthält.

#### 3.4.1 BMA 020

Der BMA 020 ist ein von Bosch entwickelter Sensor, der die Beschleunigung in drei Richtungen (x-, y- und z-Achse) misst. Die Messdaten werden im Format eines 2er Komplements mit 10 Bit ausgegeben. Als Schnittstellen stellt der BMA 020 einen SPI- und I<sup>2</sup>C Bus zur Verfügung. Die wichtigsten technischen Daten wurden dem Datenblatt [10] entnommen und in der Tabelle 3.3 dargestellt.

<i>Parameter</i>	<i>Minimum</i>	<i>Maximum</i>
Acceleration range	-2 g	2 g
	-4 g	4 g
	-8 g	8 g
Supply voltage analogue	2 V	3,6 V
Acceleration output resolution	10 Bit	

Tabelle 3.3: Technische Daten BMA020 [10]

Abbildung zeigt die fertig bestückte Platine mit dem BMA020.

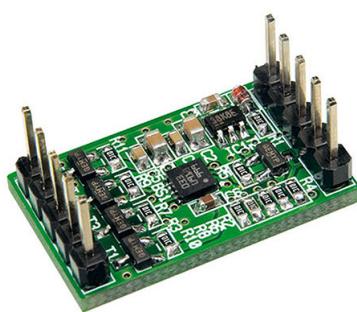


Abbildung 3.8: Platine mit BMA020 [7]

### 3.4.2 Schaltungsaufbau

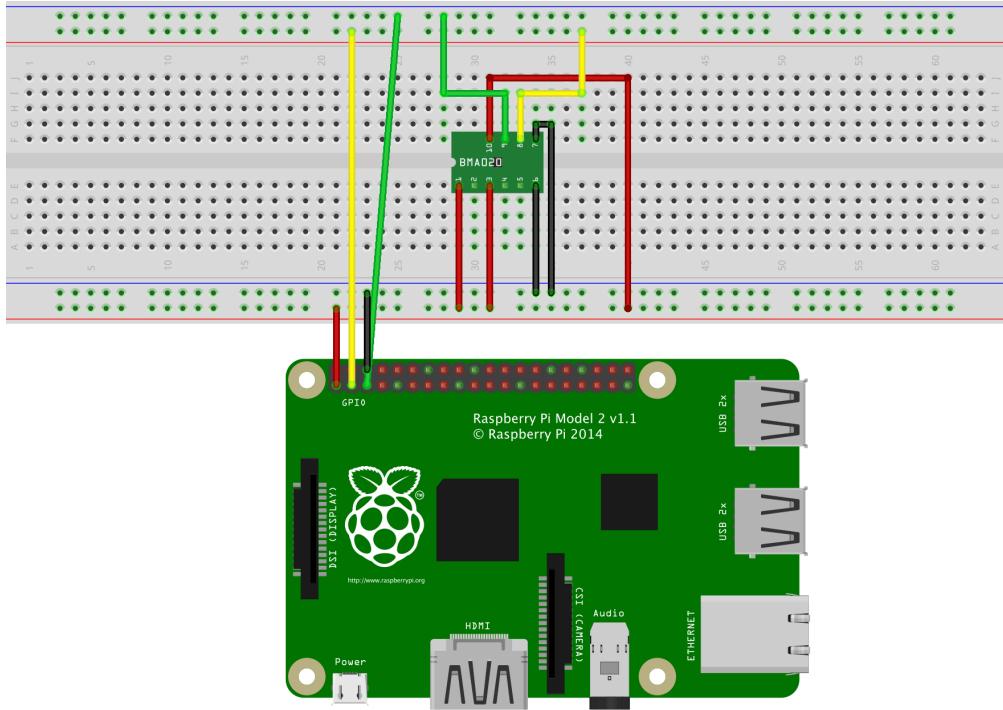


Abbildung 3.9: Schaltungsaufbau Vibrationsmessung mit BMA020

Die in Abbildung 3.9 dargestellte Schaltung beinhaltet die fertig bestückte Platine mit dem BMA020. Weiterhin ist auf der Platine auch ein Pullup Widerstand mit verbaut, sowie weitere Widerstände, um eine Spannungsversorgung der Platine mit 2,5 V - 6 V zu gewährleisten ohne dass der BMA020 Schaden nimmt (Spannungsversorgung des BMA020 liegt sonst bei max. 3,6 V).

Die Schaltung wurde an Hand des Datenblattes der Platine für einen Betrieb mittels I<sup>2</sup>C Schnittstelle verdrahtet. Dafür wurde die Pins *UIN*, *CSB* und *UPULLUP* mit einer 3,3 V Versorgungsspannung belegt (rote Verbindungen). Die Pins *SDO* und *GND* wurde auf GROUND gelegt (schwarze Verbindungen). Für die Datenübertragung wurde der Pin *SCK* mit dem *SCL* Pin des RPI (grüne Verbindung), sowie der *SDI* Pin mit dem *SDA* Pin des RPI (gelbe Verbindung) verbunden.

### 3.4.3 Auslesen der Daten

Um die Daten über den I<sup>2</sup>C Bus auszulesen, wurde ähnlich vorgegangen wie bei der Temperatur- und Luftfeuchtigkeitsmessung. Die Änderung dazu war allerdings, dass das Python Script nicht wie zuvor über den Crown-Deamon jede Minute aufgerufen wurde sondern die Messungen in einer while Schleife jede Sekunde ausgewertet wur-

den. Dies war für die Aussage über eventuelle Vibrationen nötig, da ansonsten keine aussagekräftigen Werte ermittelt werden können.

Als erstes wurden die Funktionen zum Auslesen der Register und für die Umwandlung des 2er Komplements definiert (Zeile 30 - 43). In Zeile 45 - 47 ist die Funktion für die Startbedingung des I<sup>2</sup>C Busses definiert. In der folgenden while Schleife wurden dann mittels der vorher definierten Funktion die Register für die verschiedenen Beschleunigungsrichtungen ausgelesen. Hier mussten für jeden Beschleunigungswert zwei verschiedene Register gelesen werden, da die Werte in diesen gespeichert waren<sup>5</sup> (für x-Achse musste z.B. das Register 0x02, das die LSB-Werte enthält und das Register 0x03 mit den MSB-Werten gelesen und entsprechend zusammengesetzt werden). Im Anschluss wurden noch die digitalen Werte in die entsprechenden g-Werte umgerechnet um eine aussagekräftige Darstellung zu ermöglichen. Der Ausschnitt für den eben beschriebenen Teil des Source Codes ist in Listing 3.7 ersichtlich. Der gesamte Quellcode ist in Anhang A.6 aufgelistet.

```

1 #Funktionsdefinition zum Auslesen
2 def read_word(reg):
3     LSB = BMA020.read_byte_data(add, reg)
4     LSB = LSB - 1
5     MSB = BMA020.read_byte_data(add, reg+1)
6     value = (MSB<<2)+(LSB>>6)
7     return value
8
9 def read_word_2c(reg):
10    val = read_word(reg)
11    if(val >= 0x1FF):
12        return -((1024 - val) + 1)
13    else:
14        return val
15
16 def Initialisierung_BMA020(SensorAdd):
17     BMA020.write_quick(SensorAdd)
18     sleep(0.5)
19
20 # Initialisierung BMA020
21 Initialisierung_BMA020(add)
22
23 # Cursor Erstellung für DB-Zugriff
24 cur = db.cursor()
25
```

---

<sup>5</sup>im Datenblatt des Sensors ersichtlich

```
26 # Dauerschleife für Ausgabe der Beschleunigungswerte
27
28 while(0==0):
29     x = read_word_2c(0x02)
30     y = read_word_2c(0x04)
31     z = read_word_2c(0x06)
32
33 # Berechnung für +/- 2g
34     x = x / 256.0
35     y = y / 256.0
36     z = z / 256.0
```

Listing 3.7: Auslesen der Vibrationswerte

#### 3.4.4 Datenspeicherung

Die Vibrationswerte der verschiedenen Bewegungsrichtungen wurden in diesem Testaufbau im Sekundentakt in einer mySQL Datenbank gespeichert. Die Werte wurden wie schon im vorherigen Beispiel über SQL Statements in die entsprechende Tabelle der Datenbank geschrieben. Zuvor wurde wieder die Verbindung zur DB über das Python Script hergestellt. Der genaue Ablauf ist im Quelltext im Anhang A.6 ersichtlich.

#### 3.4.5 Visualisierung der Vibrationswerte

## 4 Zusammenfassung und Ausblick

## Literaturverzeichnis

- [1] E. Upton. (2016, Februar) RASPBERRY PI 3 ON SALE NOW AT \$35. [Online]. Available: <https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/>
- [2] Raspberry Pi 3 Model B GPIO 40 Pin Block Pinout. [Online]. Available: <https://www.element14.com/community/docs/DOC-73950/l/raspberry-pi-3-model-b-gpio-40-pin-block-pinout>
- [3] *I2C-bus specification and user manual*, NXP Semiconductor, April 2014.
- [4] *DS18S20 High-Precision 1-Wire Digital Thermometer*, Maxim Integrated Products, 2015.
- [5] DS18S20 Temperature Sensor. [Online]. Available: <http://www.smartliving.com.au/ds18b20-temperature-sensor.html>
- [6] Feuchte- und Temperatursensor digital Hyt221. [Online]. Available: <http://www.voelkner.de/products/207755/Feuchte-und-Temperaturesensor-digital-Hyt-221.html>
- [7] 3-Achsen-Beschleunigungssensor. ELV-/eQ-3-Gruppe. [Online]. Available: <http://www.elv.de/3-achsen-beschleunigungssensor-3d-bs-komplettbausatz.html>
- [8] W. Klaas, *Bussysteme in der Praxis*. Franzis Verlag GmbH, 2015.
- [9] *HYGROCHIP DIGITALER FEUCHTESENSOR HYT-221*, INNOVATIVE SENSOR TECHNOLOGY, September 2011.
- [10] *BMA020 Digital, triaxial acceleration sensor*, 1. Aufl., Bosch Sensortec GmbH, Mai 2008.
- [11] Kofler, Kühnast, und Scherbeck, *Raspberry Pi Das umfassende Handbuch*. Rheinwerk Verlag GmbH, Bonn, 2015.
- [12] Bluetooth Low Energy. [Online]. Available: <https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy>
- [13] RASPBERRY PI 3 MODEL B. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- [14] L. Upton. (2015, Oktober) SENIOR PI. Raspberry Pi Foundation. [Online]. Available: <https://www.raspberrypi.org/blog/senior-pi/>
- [15] T. Oetiker. (2014, September) About RRDtool. OETIKER+PARTNER AG. [Online]. Available: <http://oss.oetiker.ch/rrdtool/>

- [16] (2010, May) DIGITALER FEUCHTESENSOR PROTOKOLLBESCHREIBUNG I2C. HYGROSENS INSTRUMENTS GmbH. [Online]. Available: [http://www.hydrochip.com/fileadmin/user\\_upload/Produkte/Sensorelemente/](http://www.hydrochip.com/fileadmin/user_upload/Produkte/Sensorelemente/Digitale_Feuchtesensoren/HYT_I2C_Protokollbeschreibung_D.pdf)  
Digitale\_Feuchtesensoren/HYT\_I2C\_Protokollbeschreibung\_D.pdf

## A Quellcode

### A.1 Python Script DS18S20

```

1  #!/usr/bin/python3
2  # -*-coding: utf-8 -*-
3
4  import sys
5  import os
6
7  #1-Wire Slave-Liste lesen
8  file = open('/sys/devices/w1_bus_master1/w1_master_slaves')
9  wl_slaves = file.readlines()
10 file.close()
11
12
13 #Fuer jeden 1-Wire Slave aktuelle Temperatur ausgeben
14 for line in wl_slaves:
15     wl_slave = line.split("\n")[0]
16     file = open('/sys/bus/w1/devices/' + str(wl_slave) + '/w1_slave')
17     filecontent = file.read()
18     file.close()
19
20     stringvalue = filecontent.split("\n")[1].split(" ")[9]
21     temperature = float(stringvalue[2:]) / 1000
22     temperature = round(temperature, 2)
23     print(str(wl_slave) + ': %6.2f C' % temperature)
24
25 ##### Sensor-Daten speichern #####
26
27 ret = rrd_update('/home/pi/Temperatur/TemperaturSensor/
28                   TemperaturAufzeichnungDB18S20.rrd', 'N:%s' % (temperature));
29 sys.exit(0)

```

Listing A.1: Quellcode zum Auslesen des DS18S20

## A.2 Erstellung Graphen DS18S20

```

1 #!/bin/sh
2
3 #Grafik erstellen
4 sudo rrdtool graph /var/www/html/TemperaturTag.png \
5 -w 900 -h 600 -t "Temperaturverlauf letzte 10 Stunden" \
6 --end now --start end-10h -v "Temperatur(°C)" --slope-mode \
7 --x-grid MINUTE:20: HOUR:1:MINUTE:60:0:%R \
8 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
     ffffff \
9 --alt-autoscale --rigid \
10 DEF:Temperatur1=/home/pi/Temperatur/TemperaturSensor/
      TemperaturAufzeichnungDB18S20.rrd:Temperatur1:AVERAGE \
11 DEF:Temperatur2=/home/pi/Temperatur/TemperaturSensor/
      TemperaturAufzeichnungDB18S20.rrd:Temperatur2:AVERAGE \
12 VDEF:Temp1Cur=Temperatur1, LAST \
13 VDEF:Temp1Max=Temperatur1, MAXIMUM \
14 VDEF:Temp1Avg=Temperatur1, AVERAGE \
15 VDEF:Temp1Min=Temperatur1, MINIMUM \
16 VDEF:Temp2Cur=Temperatur2, LAST \
17 VDEF:Temp2Max=Temperatur2, MAXIMUM \
18 VDEF:Temp2Avg=Temperatur2, AVERAGE \
19 VDEF:Temp2Min=Temperatur2, MINIMUM \
20 COMMENT:"" \
21 COMMENT:"Current" \
22 COMMENT:"Maximum" \
23 COMMENT:"Average" \
24 COMMENT:"Minimum \l" \
25 LINE3:Temperatur1#00C000:"DB18S20" \
26 GPRINT:Temp1Cur:"%6.2lf %S°C" \
27 GPRINT:Temp1Max:"%6.2lf %S°C" \
28 GPRINT:Temp1Avg:"%6.2lf %S°C" \
29 GPRINT:Temp1Min:"%6.2lf %S°C\l" \
30 LINE2:Temperatur2#0000FF:"DB18S20 mit Kabel" \
31 GPRINT:Temp2Cur:"%6.2lf %S°C" \
32 GPRINT:Temp2Max:"%6.2lf %S°C" \
33 GPRINT:Temp2Avg:"%6.2lf %S°C" \
34 GPRINT:Temp2Min:"%6.2lf %S°C\l" \
35
36 sudo rrdtool graph /var/www/html/TemperaturStunde.png \
37 -w 900 -h 600 -t "Temperaturverlauf letzte Stunde" \
38 --end now --start end-1h -v "Temperatur(°C)" --slope-mode -c
      GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#

```

```
ffffff --x-grid MINUTE:1:MINUTE:15:MINUTE:5:0:%R --alt-
autoscale --rigid \
39 DEF:Temperatur1=/home/pi/Temperatur/TemperaturSensor/
    TemperaturAufzeichnungDB18S20.rrd:Temperatur1:AVERAGE \
40 DEF:Temperatur2=/home/pi/Temperatur/TemperaturSensor/
    TemperaturAufzeichnungDB18S20.rrd:Temperatur2:AVERAGE \
41 LINE3:Temperatur1#00C000:"DB18S20 mit Kabel\t\t\t" \
42 LINE2:Temperatur2#0000FF:"DB18S20\n" \
43 GPRINT:Temperatur1:LAST:" aktuell\: %10.2lf °C\t\t" \
44 GPRINT:Temperatur2:LAST:" aktuell\: %10.2lf °C\n" \
45 GPRINT:Temperatur1:MAX:" Maximum\: %10.2lf °C\t\t" \
46 GPRINT:Temperatur2:MAX:" Maximum\: %10.2lf °C\n" \
47 GPRINT:Temperatur1:MIN:" Minimum\: %10.2lf °C\t\t" \
```

Listing A.2: Quellcode Erstellung Temperaturgraphen DS18S20

### A.3 Python Script HYT-221 mit RRD

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3 import smbus
4 import time
5 import rrdtool
6 import sys
7 import os
8 from rrdtool import update as rrd_update
9 from time import sleep
10
11
12 ##### HYT 221 #####
13
14 #I2C-Adresse
15
16 SensorAdresse = 0x28;
17
18
19 #Array fuer die Sensordaten
20
21 SensorDaten = bytearray()
22
23
24 #Sensor ID fuer HYT221
25
26 ID_HYT = "HYT_221"
27
28 #smbus Objekt fuer I2C bus #1
29
30 Sensor = smbus.SMBus(1)
31
32
33 #Initialisierung Array
34
35 SensorDaten.append(0x30)
36 SensorDaten.append(0x31)
37 SensorDaten.append(0x32)
38 SensorDaten.append(0x33)
39
40
41 #Sensor zum lesen initialisieren, Antwort ignorieren
```

```
42
43 ans = Sensor.read_byte_data(SensorAdresse,0)
44 sleep(0.1)
45
46
47 #4 Byte der Daten lesen
48
49 SensorDaten = Sensor.read_i2c_block_data(SensorAdresse,4)
50
51
52 #Luftfeuchtigkeit berechnen
53
54 DataHumidity = SensorDaten[0]<<8 | SensorDaten[1]
55 DataHumidity = DataHumidity & 0x3FFF
56
57 HYT_Humidity = 100.0*DataHumidity/(2**14)
58
59
60 #Temperatur berechnen
61
62 SensorDaten[3] = SensorDaten[3] & 0x3F
63 Temp = SensorDaten[2] << 6 | SensorDaten[3]
64
65 HYT_Temperature = 165.0*Temp/(2**14)-40
66
67 HYT_Temperature = round(HYT_Temperature,2)
68 HYT_Humidity = round(HYT_Humidity,2)
69
70 ##### DB18S20
71 #####
72
73 #1-Wire Slave-Liste lesen
74
75 file = open('/sys/devices/w1_bus_master1/w1_master_slaves')
76 wl_slaves = file.readlines()
77 file.close()
78 Temperature_W1 = []
79
80
81 #Fuer jeden 1-Wire Slave aktuelle Temperatur ausgeben
82
83 for line in wl_slaves:
84     wl_slave = line.split("\n")[0]
```

```

86     file = open('/sys/bus/w1/devices/' + str(w1_slave) + '/
87         w1_slave')
88     filecontent = file.read()
89     file.close()
90
91     stringvalue = filecontent.split("\n") [1]
92
93     stringvalue = stringvalue.split(" ") [9]
94
95     Temperature_W1.append(float(stringvalue[2:])/1000)
96
96 HYTTemp = float(HYT_Temperature)
97 HYTFeucht = float(HYT_Humidity)
98 DB18S20Temp = float(Temperature_W1[1])
99 DB18S20KabelTemp = float(Temperature_W1[0])
100
101 print ("HYT-Temperatur : "), str(HYTTemp)
102 print ("DB-Temperatur : "), str(DB18S20Temp)
103 print ("DB-Temperatur : "), str(DB18S20KabelTemp)
104 print ("HYT-Feuchtigkeit: "), str(HYTFeucht)
105
106 ##### RRD-Daten speichern
107 #####
108
109 ret = rrd_update('/home/pi/Temperatur/Messung/Messung.rrd', '
110     N:%s:%s:%s' %(HYTTemp, DB18S20KabelTemp, DB18S20Temp,
111     HYTFeucht));

```

Listing A.3: Quellcode zum Auslesen und Speichern der Daten von drei Sensoren mit RRDtool

## A.4 Python Script HYT-221 mit mySQL

```

1 #!/usr/bin/python3
2 # -*- coding: utf-8 -*-
3 import smbus
4 import time
5 import sys
6 import os
7 import mysql.connector
8 from time import sleep
9
10 ##### Datenbank #####
11
12 #Datenbank Verbindung herstellen
13 try:
14     db = mysql.connector.connect(host="localhost", user="root",
15                                   passwd="test123", db="RPI-Projekt")
16 except :
17     print("No connection to database")
18     exit(0)
19
20 #### HYT 221 #####
21
22 #I2C-Adresse
23 SensorAdresse = 0x28;
24
25 #Array fuer die Sensordaten
26 SensorDaten = bytearray()
27
28 #Sensor ID fuer HYT221
29 ID_HYT = "HYT_221"
30
31 #smbus Objekt fuer I2C bus #1
32 Sensor = smbus.SMBus(1)
33
34 #Initialisierung Array
35 SensorDaten.append(0x30)
36 SensorDaten.append(0x31)
37 SensorDaten.append(0x32)
38 SensorDaten.append(0x33)
39
40 #Sensor zum lesen initialisieren, Antwort ignorieren

```

```
41 ans = Sensor.read_byte_data(SensorAdresse, 0)
42 sleep(0.1)
43
44 #4 Byte der Daten lesen
45 SensorDaten = Sensor.read_i2c_block_data(SensorAdresse, 4)
46
47
48 #Luftfeuchtigkeit berechnen
49
50 DataHumidity = SensorDaten[0]<<8 | SensorDaten[1]
51 DataHumidity = DataHumidity & 0x3FFF
52
53 HYT_Humidity = 100.0*DataHumidity/(2**14)
54
55
56 #Temperatur berechnen
57
58 SensorDaten[3] = SensorDaten[3] & 0x3F
59 Temp = SensorDaten[2] << 6 | SensorDaten[3]
60
61 HYT_Temperature = 165.0*Temp/(2**14)-40
62
63 HYT_Temperature = round(HYT_Temperature, 2)
64 HYT_Humidity = round(HYT_Humidity, 2)
65
66
67 ##### DB18S20
68 #####
69 #1-Wire Slave-Liste lesen
70
71 file = open('/sys/devices/w1_bus_master1/w1_master_slaves')
72 wl_slaves = file.readlines()
73 file.close()
74 Temperature_W1 = []
75 Slave_W1 = []
76
77
78 #Fuer jeden 1-Wire Slave aktuelle Temperatur ausgeben
79
80 for line in wl_slaves:
81     Slave_W1.append(line.split("\n")[0])
82     wl_slave = line.split("\n")[0]
83     file = open('/sys/bus/w1/devices/' + str(wl_slave) + '/w1_slave')
```

```
84     filecontent = file.read()
85     file.close()
86
87     stringvalue = filecontent.split("\n") [1]
88
89     stringvalue = stringvalue.split(" ") [9]
90
91     Temperature_W1.append(round((float(stringvalue[2:])/1000)
92                               ,2))
93
94
95 ##### Daten in mySQL speichern
96 ######
97 #Cursor zum Eintragen erzeugen
98 cur = db.cursor()
99
100 #SQL-Statement erzeugen
101 sql = "INSERT INTO Temperatur (DB18S20, DB18S20K, HYT22)
102           VALUES (%s,%s,%s)"
103 cur.execute(sql, (Temperature_W1[1],Temperature_W1[0],
104                 HYT_Temperature))
105
106
107 db.commit()
108 db.close()
```

Listing A.4: Quellcode zum Auslesen und Speichern der Daten von drei Sensoren mit mySQL DB

## A.5 Erstellung Graphen HYT-221

```

1 #!/bin/sh
2
3 #Grafik erstellen
4 sudo rrdtool graph /var/www/html/Messung/Bilder/
    TemperaturStunde.png \
5 -w 900 -h 600 -t "Temperaturverlauf letzte Stunde" \
6 --end now --start end-1h -v "Temperatur(°C)" --y-grid=0.2:5
    --slope-mode --no-gridfit \
7 --grid-dash 1:0 \
8 --x-grid MINUTE:1:MINUTE:15:MINUTE:5:0:%R \
9 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
10 DEF:HYT_Temp=/home/pi/Temperatur/Messung/Messung.rrd:HYT_Temp
    :AVERAGE \
11 DEF:DB18S20_Temp=/home/pi/Temperatur/Messung/Messung.rrd:
    DB18S20_Temp:AVERAGE \
12 DEF:DB18S20_Kabel_Temp=/home/pi/Temperatur/Messung/Messung.
    rrd:DB18S20_Kabel_Temp:AVERAGE \
13 VDEF:Temp1Cur=HYT_Temp,LAST \
14 VDEF:Temp1Max=HYT_Temp,MAXIMUM \
15 VDEF:Temp1Avg=HYT_Temp,AVERAGE \
16 VDEF:Temp1Min=HYT_Temp,MINIMUM \
17 VDEF:Temp2Cur=DB18S20_Temp,LAST \
18 VDEF:Temp2Max=DB18S20_Temp,MAXIMUM \
19 VDEF:Temp2Avg=DB18S20_Temp,AVERAGE \
20 VDEF:Temp2Min=DB18S20_Temp,MINIMUM \
21 VDEF:Temp3Cur=DB18S20_Kabel_Temp,LAST \
22 VDEF:Temp3Max=DB18S20_Kabel_Temp,MAXIMUM \
23 VDEF:Temp3Avg=DB18S20_Kabel_Temp,AVERAGE \
24 VDEF:Temp3Min=DB18S20_Kabel_Temp,MINIMUM \
25 COMMENT:"" \
26 COMMENT:"Current" \
27 COMMENT:"Maximum" \
28 COMMENT:"Average" \
29 COMMENT:"Minimum \l" \
30 LINE2:HYT_Temp#4CCDFF:"HYT 221" \
31 GPRINT:Temp1Cur:" %2.2lf °C" \
32 GPRINT:Temp1Max:" %2.2lf °C" \
33 GPRINT:Temp1Avg:" %2.2lf °C" \
34 GPRINT:Temp1Min:" %2.2lf °C\l" \
35 LINE2:DB18S20_Temp#22BA32:"DB18S20 mit Kabel" \
36 GPRINT:Temp2Cur:" %2.2lf °C" \

```

```

37 GPRINT:Temp2Max:" %2.2lf %S°C" \
38 GPRINT:Temp2Avg:" %2.2lf %S°C" \
39 GPRINT:Temp2Min:" %2.2lf %S°C\l" \
40 LINE2:DB18S20_Kabel_Temp#AF2834:"DB18S20" \
41 GPRINT:Temp3Cur:" %2.2lf %S°C" \
42 GPRINT:Temp3Max:" %2.2lf %S°C" \
43 GPRINT:Temp3Avg:" %2.2lf %S°C" \
44 GPRINT:Temp3Min:" %2.2lf %S°C\l" \
45
46
47
48 sudo rrdtool graph /var/www/html/Messung/Bilder/TemperaturTag
    .png \
49 -w 900 -h 600 -t "Temperaturverlauf letzte 10 Stunden" \
50 --end now --start end-10h -v "Temperatur(°C)" --y-grid=0.5:2
    --slope-mode --no-gridfit \
51 --grid-dash 1:0 \
52 --x-grid MINUTE:20:HOUR:1:MINUTE:60:0:%R \
53 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
54 DEF:HYT_Temp=/home/pi/Temperatur/Messung/Messung.rrd:HYT_Temp
    :AVERAGE \
55 DEF:DB18S20_Temp=/home/pi/Temperatur/Messung/Messung.rrd:
    DB18S20_Temp:AVERAGE \
56 DEF:DB18S20_Kabel_Temp=/home/pi/Temperatur/Messung/Messung.
    rrd:DB18S20_Kabel_Temp:AVERAGE \
57 VDEF:Temp1Cur=HYT_Temp,LAST \
58 VDEF:Temp1Max=HYT_Temp,MAXIMUM \
59 VDEF:Temp1Avg=HYT_Temp,AVERAGE \
60 VDEF:Temp1Min=HYT_Temp,MINIMUM \
61 VDEF:Temp2Cur=DB18S20_Temp,LAST \
62 VDEF:Temp2Max=DB18S20_Temp,MAXIMUM \
63 VDEF:Temp2Avg=DB18S20_Temp,AVERAGE \
64 VDEF:Temp2Min=DB18S20_Temp,MINIMUM \
65 VDEF:Temp3Cur=DB18S20_Kabel_Temp,LAST \
66 VDEF:Temp3Max=DB18S20_Kabel_Temp,MAXIMUM \
67 VDEF:Temp3Avg=DB18S20_Kabel_Temp,AVERAGE \
68 VDEF:Temp3Min=DB18S20_Kabel_Temp,MINIMUM \
69 COMMENT:"" \
70 COMMENT:"Current" \
71 COMMENT:"Maximum" \
72 COMMENT:"Average" \
73 COMMENT:"Minimum \l" \
74 LINE2:HYT_Temp#4CCDFF:"HYT 221" \
75 GPRINT:Temp1Cur:" %2.2lf %S°C" \

```

```

76 GPRINT:Temp1Max:" %2.2lf °C" \
77 GPRINT:Temp1Avg:" %2.2lf °C" \
78 GPRINT:Temp1Min:" %2.2lf °C\l" \
79 LINE2:DB18S20_Temp#22BA32:"DB18S20 mit Kabel" \
80 GPRINT:Temp2Cur:" %2.2lf °C" \
81 GPRINT:Temp2Max:" %2.2lf °C" \
82 GPRINT:Temp2Avg:" %2.2lf °C" \
83 GPRINT:Temp2Min:" %2.2lf °C\l" \
84 LINE2:DB18S20_Kabel_Temp#AF2834:"DB18S20" \
85 GPRINT:Temp3Cur:" %2.2lf °C" \
86 GPRINT:Temp3Max:" %2.2lf °C" \
87 GPRINT:Temp3Avg:" %2.2lf °C" \
88 GPRINT:Temp3Min:" %2.2lf °C\l" \
89
90
91
92 sudo rrdtool graph /var/www/html/Messung/Bilder/
    Temperatur3Tage.png \
93 -w 900 -h 600 -t "Temperaturverlauf letzte 3 Tage" \
94 --end now --start end-3d -v "Temperatur(°C)" --y-grid=0.5:2
    --slope-mode --no-gridfit \
95 --grid-dash 1:0 \
96 --x-grid HOUR:4:HOUR:12:MINUTE:720:0:%A" "%R \
97 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
98 DEF:HYT_Temp=/home/pi/Temperatur/Messung/Messung.rrd:HYT_Temp
    :AVERAGE \
99 DEF:DB18S20_Temp=/home/pi/Temperatur/Messung/Messung.rrd:
    DB18S20_Temp:AVERAGE \
100 DEF:DB18S20_Kabel_Temp=/home/pi/Temperatur/Messung/Messung.
    rrd:DB18S20_Kabel_Temp:AVERAGE \
101 VDEF:Temp1Cur=HYT_Temp, LAST \
102 VDEF:Temp1Max=HYT_Temp, MAXIMUM \
103 VDEF:Temp1Avg=HYT_Temp, AVERAGE \
104 VDEF:Temp1Min=HYT_Temp, MINIMUM \
105 VDEF:Temp2Cur=DB18S20_Temp, LAST \
106 VDEF:Temp2Max=DB18S20_Temp, MAXIMUM \
107 VDEF:Temp2Avg=DB18S20_Temp, AVERAGE \
108 VDEF:Temp2Min=DB18S20_Temp, MINIMUM \
109 VDEF:Temp3Cur=DB18S20_Kabel_Temp, LAST \
110 VDEF:Temp3Max=DB18S20_Kabel_Temp, MAXIMUM \
111 VDEF:Temp3Avg=DB18S20_Kabel_Temp, AVERAGE \
112 VDEF:Temp3Min=DB18S20_Kabel_Temp, MINIMUM \
113 COMMENT:"" \
114 COMMENT:"Current" \

```

```

115 COMMENT:"Maximum      " \
116 COMMENT:"Average      " \
117 COMMENT:"Minimum \l" \
118 LINE2:HYT_Temp#4CCDFF:"HYT 221" \
119 GPRINT:Temp1Cur:"           %2.2lf %S°C" \
120 GPRINT:Temp1Max:" %2.2lf %S°C" \
121 GPRINT:Temp1Avg:" %2.2lf %S°C" \
122 GPRINT:Temp1Min:" %2.2lf %S°C\l" \
123 LINE2:DB18S20_Temp#22BA32:"DB18S20 mit Kabel" \
124 GPRINT:Temp2Cur:" %2.2lf %S°C" \
125 GPRINT:Temp2Max:" %2.2lf %S°C" \
126 GPRINT:Temp2Avg:" %2.2lf %S°C" \
127 GPRINT:Temp2Min:" %2.2lf %S°C\l" \
128 LINE2:DB18S20_Kabel_Temp#AF2834:"DB18S20" \
129 GPRINT:Temp3Cur:"           %2.2lf %S°C" \
130 GPRINT:Temp3Max:" %2.2lf %S°C" \
131 GPRINT:Temp3Avg:" %2.2lf %S°C" \
132 GPRINT:Temp3Min:" %2.2lf %S°C\l" \
133
134
135
136 sudo rrdtool graph /var/www/html/Messung/Bilder/
    FeuchtigkeitStunde.png \
137 -w 900 -h 600 -t "Luftfeuchtigkeit letzte Stunde" \
138 --end now --start end-1h -v "Luftfeuchtigkeit(%rF)" --y-grid
    =0.1:10 --slope-mode --no-gridfit \
139 --grid-dash 1:0 \
140 --x-grid MINUTE:1:MINUTE:15:MINUTE:5:0:%R \
141 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
142 DEF:HYT_Feucht=/home/pi/Temperatur/Messung/Messung.rrd:
    HYT_Feucht:AVERAGE \
143 VDEF:FeuchtCur=HYT_Feucht,LAST \
144 VDEF:FeuchtMax=HYT_Feucht,MAXIMUM \
145 VDEF:FeuchtAvg=HYT_Feucht,AVERAGE \
146 VDEF:FeuchtMin=HYT_Feucht,MINIMUM \
147 COMMENT:"           " \
148 COMMENT:"Current      " \
149 COMMENT:"Maximum      " \
150 COMMENT:"Average      " \
151 COMMENT:"Minimum \l" \
152 AREA:HYT_Feucht#4CCDFF:"HYT 221" \
153 GPRINT:FeuchtCur:"           %2.2lf %S°C" \
154 GPRINT:FeuchtMax:" %2.2lf %S°C" \
155 GPRINT:FeuchtAvg:" %2.2lf %S°C" \

```

```

156 GPRINT:FeuchtMin:" %2.2lf %S°C\l" \
157
158
159
160 sudo rrdtool graph /var/www/html/Messung/Bilder/
    FeuchtigkeitTag.png \
161 -w 900 -h 600 -t "Luftfeuchtigkeit letzte 10 Stunden" \
162 --end now --start end-10h -v "Luftfeuchtigkeit(%rF)" --y-grid
    =0.5:2 --slope-mode --no-gridfit \
163 --grid-dash 1:0 \
164 --x-grid MINUTE:20: HOUR:1:MINUTE:60:0:%R \
165 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
166 DEF:HYT_Feucht=/home/pi/Temperatur/Messung/Messung.rrd:
    HYT_Feucht:AVERAGE \
167 VDEF:FeuchtCur=HYT_Feucht, LAST \
168 VDEF:FeuchtMax=HYT_Feucht, MAXIMUM \
169 VDEF:FeuchtAvg=HYT_Feucht, AVERAGE \
170 VDEF:FeuchtMin=HYT_Feucht, MINIMUM \
171 COMMENT: "                                " \
172 COMMENT: "Current      " \
173 COMMENT: "Maximum      " \
174 COMMENT: "Average      " \
175 COMMENT: "Minimum \l" \
176 AREA:HYT_Feucht#4CCDFF:"HYT 221" \
177 GPRINT:FeuchtCur:"          %2.2lf %S°C" \
178 GPRINT:FeuchtMax:" %2.2lf %S°C" \
179 GPRINT:FeuchtAvg:" %2.2lf %S°C" \
180 GPRINT:FeuchtMin:" %2.2lf %S°C\l" \
181
182
183
184 sudo rrdtool graph /var/www/html/Messung/Bilder/
    Feuchtigkeit3Tage.png \
185 -w 900 -h 600 -t "Luftfeuchtigkeit letzte 3 Tage" \
186 --end now --start end-3d -v "Luftfeuchtigkeit(%rF)" --y-grid
    =0.5:2 --slope-mode --no-gridfit \
187 --grid-dash 1:0 \
188 --x-grid HOUR:4:HOUR:12:MINUTE:720:0:%A" "%R \
189 -c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
    ffffff \
190 DEF:HYT_Feucht=/home/pi/Temperatur/Messung/Messung.rrd:
    HYT_Feucht:AVERAGE \
191 VDEF:FeuchtCur=HYT_Feucht, LAST \
192 VDEF:FeuchtMax=HYT_Feucht, MAXIMUM \

```

```
193 VDEF:FeuchtAvg=HYT_Feucht,AVERAGE \
194 VDEF:FeuchtMin=HYT_Feucht,MINIMUM \
195 COMMENT:"" \
196 COMMENT:"Current" \
197 COMMENT:"Maximum" \
198 COMMENT:"Average" \
199 COMMENT:"Minimum \l" \
200 AREA:HYT_Feucht#4CCDFF:"HYT 221" \
201 GPRINT:FeuchtCur:"%2.2lf %S°C" \
202 GPRINT:FeuchtMax:"%2.2lf %S°C" \
203 GPRINT:FeuchtAvg:"%2.2lf %S°C" \
204 GPRINT:FeuchtMin:"%2.2lf %S°C\l" \
```

Listing A.5: Quellcode zum Erstellen der Graphen für die drei Sensoren

## A.6 Python Script Vibrationsmessung

```

1  #!/usr/bin/python3
2  # -*- coding: utf-8 -*-
3  import smbus
4  import sys
5  import os
6  import time
7  import math
8  import mysql.connector
9
10 from math import sqrt
11 from time import sleep
12 from smbus import SMBus
13
14 # Verbindung zur Datenbank
15
16 try:
17     db = mysql.connector.connect(host="localhost", user="root",
18                                   passwd="test123", db="RPI-Projekt")
19 except:
20     print("No connection to database")
21     exit(0)
22
23 sql = "INSERT INTO Vibration (XWERT, YWERT, ZWERT) VALUES (%s
24 ,%s,%s)"
25
26 #I2C Adresse
27 add = 0x38
28
29 #smbus Objekt für I2C erstellen
30 BMA020 = smbus.SMBus(1)
31
32 #Funktionsdefinition zum Auslesen
33 def read_word(reg):
34     LSB = BMA020.read_byte_data(add, reg)
35     LSB = LSB - 1
36     MSB = BMA020.read_byte_data(add, reg+1)
37     value = (MSB<<2)+(LSB>>6)
38     return value
39
40 def read_word_2c(reg):
41     val = read_word(reg)
42     if(val >= 0x1FF):

```

```

41         return -( (1024 - val) + 1)
42     else:
43         return val
44
45 def Initialisierung_BMA020(SensorAdd):
46     BMA020.write_quick(SensorAdd)
47     sleep(0.5)
48
49 # Initialisierung BMA020
50 Initialisierung_BMA020(add)
51
52 # Cursor Erstellung für DB-Zugriff
53 cur = db.cursor()
54
55 # Dauerschleife für Ausgabe der Beschleunigungswerte
56
57 while(0==0):
58     x = read_word_2c(0x02)
59     y = read_word_2c(0x04)
60     z = read_word_2c(0x06)
61
62 # Berechnung für +/- 2g
63     x = x / 256.0
64     y = y / 256.0
65     z = z / 256.0
66
67 # Runden der Ergebnisse auf drei Nachkommastellen
68     x = round(x,3)
69     y = round(y,3)
70     z = round(z,3)
71
72 #daten_in = open("Daten.txt")
73 daten_out = open("Daten.txt", "w")
74 daten_out.write(str(x) + " " + str(y) + " " + str(z))
75 daten_out.close()
76 # Daten in DB eintragen
77 cur.execute(sql,(x, y, z,))
78 db.commit()
79
80 # Gesamtbeschleunigung berechnen
81 total = (sqrt(x**2 + y**2 + z**2))
82 total = round(total, 3)
83
84 # Ausgabe der Ergebnisse
85 print("x: " + str(x) + "      " + "y: " + str(y) + "      "

```

```
+ "z: " + str(z) + "      " + "Beschleunigung: " + str  
(total) + " g")  
sleep(1)
```

Listing A.6: Quellcode zum Auslesen und Speichen der Vibrationswerte

## B Abbildungen

### B.1 Graphen Temperatur und Luftfeuchtigkeit

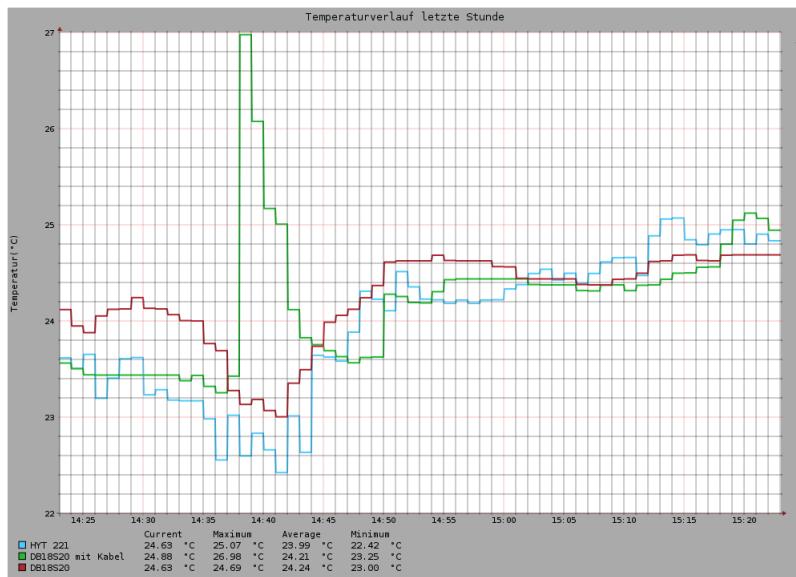


Abbildung B.1: Temperaturverlauf der letzten Stunde

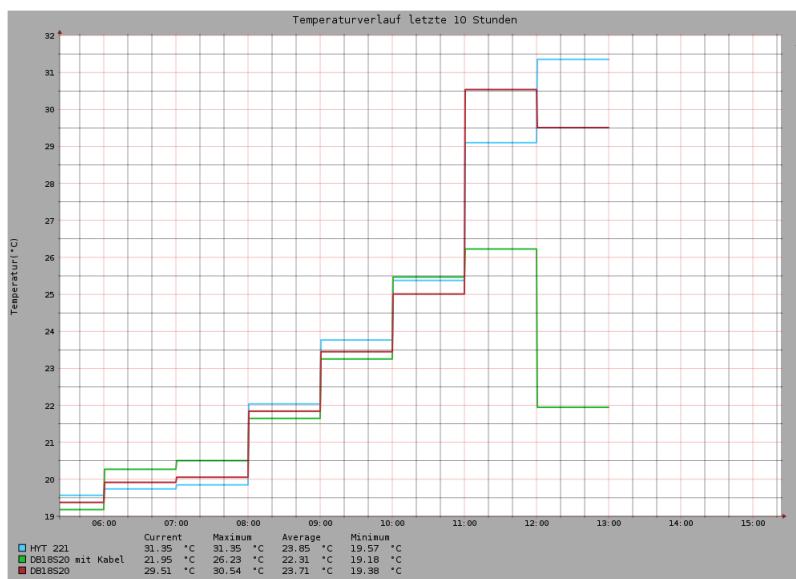


Abbildung B.2: Temperaturverlauf der letzten 10 Stunden

### B.2 Visualisierung der Vibrationswerte

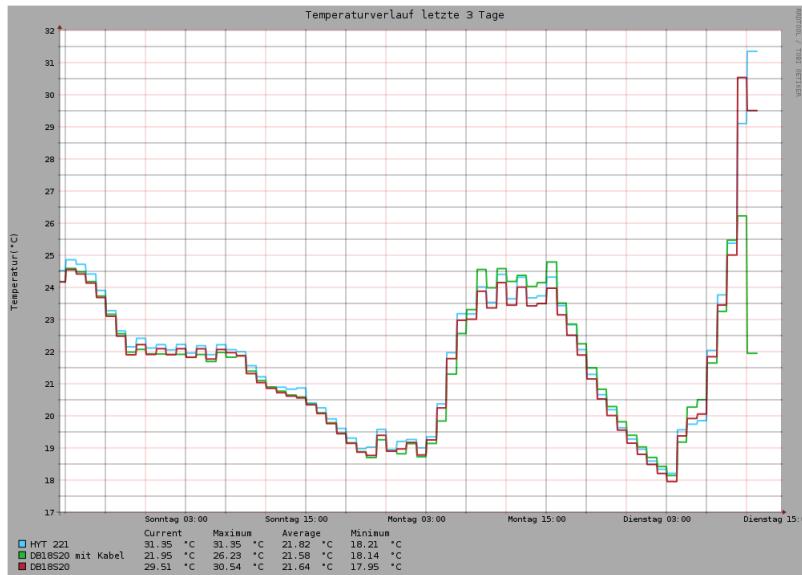


Abbildung B.3: Temperaturverlauf der letzten 3 Tage

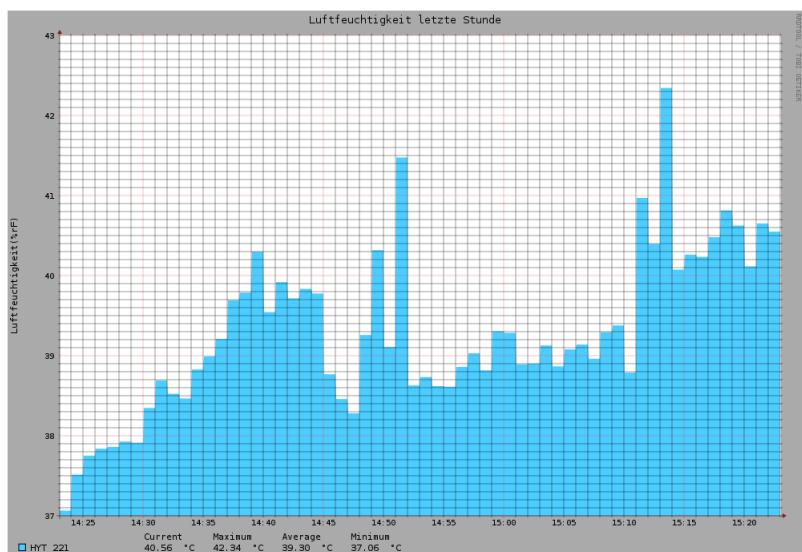


Abbildung B.4: Luftfeuchtigkeit der letzten Stunde

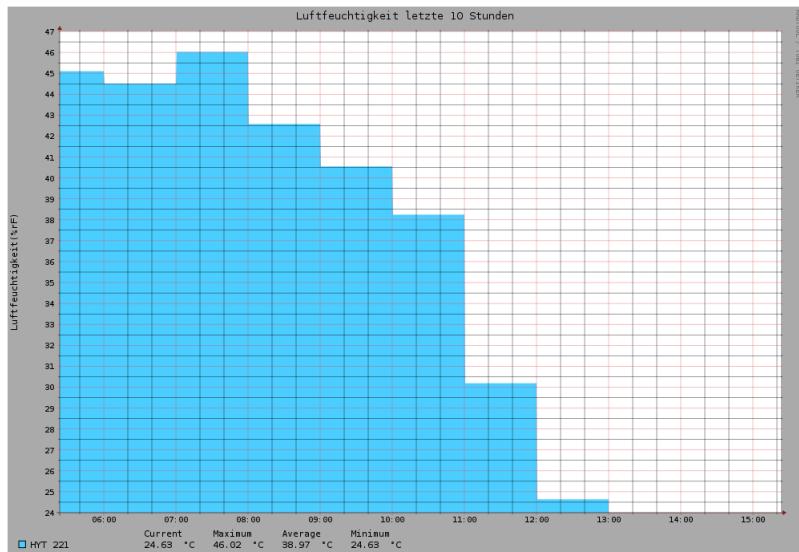


Abbildung B.5: Luftfeuchtigkeit der letzten 10 Stunden

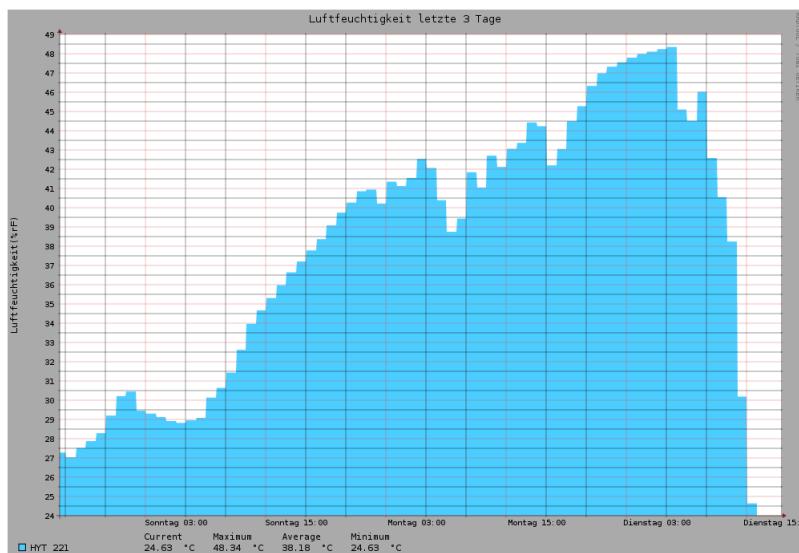


Abbildung B.6: Luftfeuchtigkeit der letzten 3 Tage



Abbildung B.7: Vibration im normalen Bereich

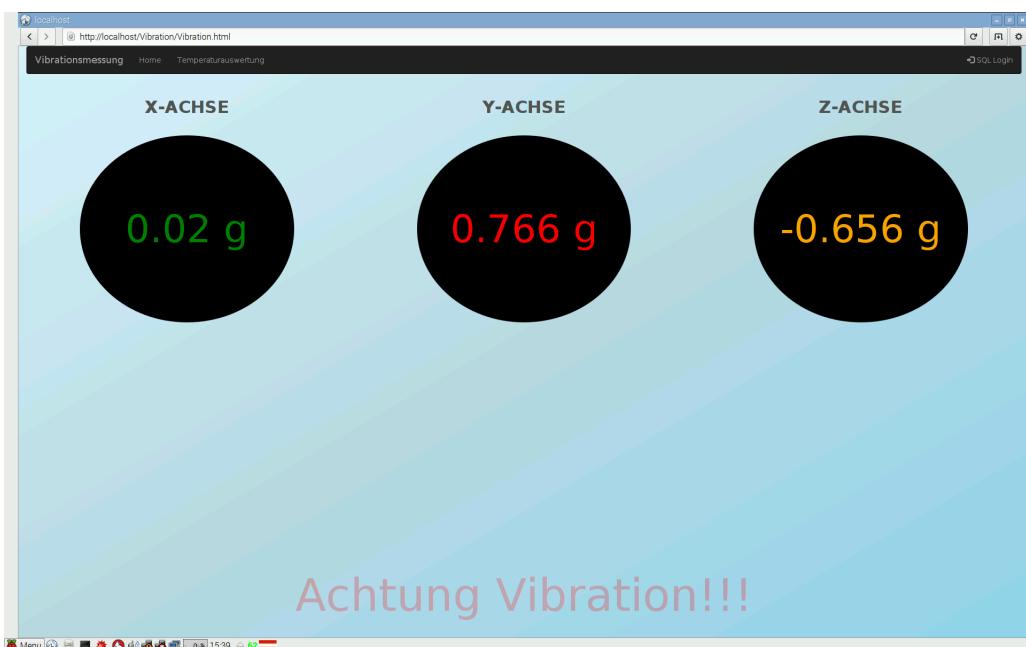


Abbildung B.8: Vibration im kritischen Bereich