BACHELORARBEIT

Möglichkeiten des Monitorings durch 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 des Monitorings durch Einbindung von Single Board Computer in ein bestehendes Fertigungsumfeld
Schlagwörter:	I^2C, 1-Wire, SBC, RRDtool, Raspberry Pi
Betreuer an der FH:	FH-Ass. Prof. Dipl. Phys. Judith Schwarzer

Kurzzusammenfassung

Single Board Computer (SBC) sind Einplatinenrechner, bei denen die verschiedenen Hardware Komponenten auf einer Platine zusammengefasst sind.
Diese Arbeit beschäftigt sich mit dem Aufbau verschiedener Schaltungen zum Einsatz in einem bestehenden Fertigungsumfeld. Zunächst wird auf die unterschiedlichen Möglichkeiten zur Datenübertragung von SBCs und Sensoren eingegangen und einige verschiedenen Übertragungssysteme näher erörtert.
Im praktischen Teil der Arbeit werden drei Testszenarien für den Einsatze von SBCs beschrieben und auf die unterschiedlichen Möglichkeiten zur Datenspeicherung und Visualisierung eingegangen. Die Unterschiede, sowie die Vor- und Nachteile der einzelnen Varianten zur Datenspeicherung werden hier dargelegt.

Abstract

Single Board Computers are computers where the different hardware is placed on one platin.
This paper concerned with the construction of several circuits in relation to an application in existing production lines. First of all the capabilities of communication from SBCs and sensors as well as different telecommunication systems will be explained.
The practical part of this paper showes three test settings for the application of SBCs and discribes the various possibilities respectively to data storage and visualization. The variaties, advantages and disadvantages of the different versions will be presented.

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

Abkürzungsverzeichnis

SPI Serial Peripheral Interface

SBC Single Board Computer

I²C 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

Tabellenverzeichnis

Listings

1 Einleitung

Ein heutzutage häufig verwendetes Schlagwort im Bereich der industriellen Produktion ist *Industrie 4.0*. Damit ist vornehmlich gemeint, dass viele Abläufe vollautomatisiert vonstattengehen, egal ob in der Produktion selbst, in der Logistik, bei der Materialbestellung oder auch beim Versand. Um die entsprechenden Anforderungen zu bewerkstelligen, werden immer neuere Technologien eingesetzt. In dem 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 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 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 Schaltschränken oder Serverräumen um etwaige zu hohe Temperaturen frühzeitig erkennen zu können und diesen entgegenzuwirken. Weiterhin sollten auch noch Möglichkeiten für den Einsatz von Feuchtigkeits- 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 die in der Arbeit verwendeten Fachbegriffe erklärt und einige der auf dem Markt verfügbaren SBCs miteinander verglichen um deren Vor- und Nachteile darzulegen und die bestmögliche Variante für die gegebenen Anforderungen auswählen zu können. Weiterhin werden die im späteren Verlauf verwendeten Bussysteme zur Übertragung der Daten vom Sensor an den Single Board Computer erläutert, sowie deren Übertragungsprotokolle. Im dritten Kapitel, dem praktischen Teil werden die verschiedenen Schaltungen von unterschiedlichen Sensoren, sowie den verschiedenen Möglichkeiten zur Speicherung und Visualisierung der Daten dargestellt. Dabei wird vor allem auf die Datenspeicherung mittels *mySQL* und mit dem *RRDTool* eingegangen. Im letzten, dem 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 verschiedener 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^2C Bus werden in Abschnitt 2.4 erklärt und deren Funktionsweise erläutert.

2.1 Begriffsdefinitionen

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 im Hinblick auf die Hardwareausstattung (z.B. Größe des Speichers, Taktfrequenz der CPU etc.) um einiges unterlegen.

Bussysteme

Bussysteme sind Systeme, die zur seriellen Datenübertragung zwischen einen oder mehreren Komponenten verwendet werden. Beispiele hierfür sind der I^2C 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. Temperatur- 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

General Purpose Input / Output (GPIO) sind elektrische Kontakte auf einem SBC, die zur Realisierung verschiedener Funktionen für elektronische Geräte verwendet werden [12]. Eine genauere Erklärung erfolgt in Abschnitt 2.3.2 .

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. Der Strombedarf bei BLE ist um einiges geringer als bei herkömmlichen Bluetooth [13].

Windows 10 IoT

Windows 10 IoT ist eine „abgespeckte“ Version von Windows 10, die auf mobilen Geräten wie dem Raspberry Pi 3 lauffähig ist. IoT steht dabei für *Internet of Things* [12].

2.2 Vergleich von verschiedenen SBCs

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. Vergleichskriterium ist die Unterstützung von verschiedenen Kommunikationsschnittstellen (I^2C, SPI, 1-Wire), um eine

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

SBC	*Operating System*	*RAM*	*CPU*
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 [6] [7]

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

[1] mit WLAN on Board

[2] nur für Android

Tabelle 2.2: Vergleich Schnittstellen, Netzwerkverbindung, Anzahl GPIO Pins [6] [7]

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, um sich

im späteren Verlauf des Projektes nicht auf ein Betriebssystem einzuschränken. 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, was für das vorgesehene Projekt von großer Bedeutung ist.

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 folgenden Information wurden, soweit nicht anders angegeben von der offiziellen Homepage der **Raspberry Pi Foundation** entnommen [7].
Der RPI 3 ist ein kreditkartengroßer Einplatinenrechner, welcher aktuell an die *sieben Million* 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* zu betreiben, wodurch sich eine Erweiterung der Einsatzmöglichkeiten ergibt. Auch wurde im Gegensatz zu den vorherigen Modellen ein WLAN Modul (2,4 GHz) gleich auf der Platine verbaut und muss nicht mehr 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

Im Gegensatz zu den Vorgängermodellen des RPI 3, welche nur 26 GPIO Pins besaßen, besitzt der RPI 3 40 GPIO Pins. Diese sind in Abbildung 2.1 ersichtlich am linken oberen Ende der Platine zu 2 x 20 Kontakten angeordnet. Der Rasterabstand von einem Pin zum nächsten beträgt 2,54 mm. 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 V* und *5 V*) und Masse Anschlüsse (*0 V*). Weiterhin beinhaltet die Steckerleiste Kontakte für den I²C-, SPI- und 1-Wire-Bus. Bei der Verwendung der GPIO-Kontakten für verschiedene Projekte muss darauf geachtet werden, welche Bezeichnung verwendet wird. Die drei Möglichkeiten sind:

- 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 [12].

In Abbildung 2.2 ist die Pin Belegung des RPI 3 grafisch dargestellt.

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	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 (I²C ID EEPROM)	(I²C 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-Kontakte [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^2C und der *SPI* Bus.

2.4.1 1-Wire

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

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 verwendet, 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. Während der Kommunikation wird der Sensor aus einem internen Kondensator gespeist. Allerdings kann es notwendig sein, bei Sensoren bei denen die interne Spannungsversorgung nicht ausreichend ist, eine extra Spannungsversorgung für den jeweiligen Sensor mitzuführen.
Der 1-Wire Bus ist ein *One-Master-Multi-Slave* Bussystem, was bedeutet, dass es einen Master (z.B. RPI 3) und mehrere Slaves (z.B. Sensoren) gibt. Die Aufgabe des Masters ist es, die Kommunikation zu steuern. Die maximale Anzahl der Sensoren kann bis zu 100 betragen, welche parallel zueinander an den Master angeschlossen werden. Dies ist möglich, da jeder Sensor ein eindeutige 64 Bit lange ID besitzt [8].

Übertragungsprotokoll

Der 1-Wire Bus wird dadurch, dass er kein 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[2] und der zuvor erwähnte interne Kondensator geladen. Wenn eine Übertragung stattfindet liegt die Datenleitung auf Masse und der

[1] 2001 von Maxim Integrated übernommen
[2] bedeutet, dass durch den Pullup Widerstand exakt die Versorgungsspannung z.B. 3,3 V anliegt

Kondensator liefert in diesem Fall die Spannungsversorgung für den Sensor (abhängig vom Sensor). Dadurch, dass keine Taktleitung vorhanden ist, muss für die Kommunikation ein bestimmter Ablauf eingehalten werden (siehe Tabelle 2.3).
Die Zeitspanne für die Übertragung von 1 Bit beträgt immer 60 μs. 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].

Befehl	*Beschreibung*
Write 1	Der Master zieht für $1 - 15$ μs auf Low. Der Rest des Slots bleibt ungenutzt.
Write 0	Der Master zieht den Bus für mindestens 60 μs bis maximal 120 μs auf Low[3]
Read	Der Master zieht für $1 - 15$ μs auf Low. Der Slave, der kommunizieren möchte, hält für eine 0 den Bus weiter auf Low[3]. Will der Slave eine 1 senden, gibt er direkt den Bus wieder frei. Wie man leicht erkennt, ist der Status Write 1 oder Read für den Master gleich. Alleine der Status des Sensors bestimmt, ob ein Read oder Write 1 ausgeführt wird.
Reset / Presence	Der Master zieht den Bus für min. 480 μs auf Low. Wenn ein Slave am Bus vorhanden ist, zieht er max. 60 μs (also einen Slot) die Leitung auf Low. Somit weiß der Master, dass mindestens ein Slave angeschlossen ist.

[3] auf Low ziehen = es liegen exakt 0 V an

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

2.4.2 I²C-Bus

Der I²C 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[4] entwickelt und wird vornehmlich zur internen Kommunikation von Geräten benutzt [8].

[4] heute NPX

Allgemeine Informationen

Technisch gesehen sind der I²C und TWI Bus identisch, die Unterscheidung wird lediglich aus lizenzrechtlichen Gründen getroffen. Bei dem Bus handelt es sich um einen *Master-Slave-Bus*, allerdings ist auch ein *Multi-Master*[5] 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 zur Datenübertragung notwendig, eine Datenleitung (SDA) und eine Taktleitung (SCL). Da der I²C Bus eine Taktleitung besitzt, spricht man hier von einem synchronen seriellen Bussystem. Der Systemtakt wird bei diesem System immer vom Master vorgegeben [8].

Definierte Zustände und Adressierung

Bei der Kommunikation mittels I²C müssen bestimmte Zustände eingehalten werden um die korrekte Funktion sicherzustellen. Diese werden im Anschluss 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 Taktleitung nicht ändern. Dieser Zustand ist in Abbildung 2.3 dargestellt.

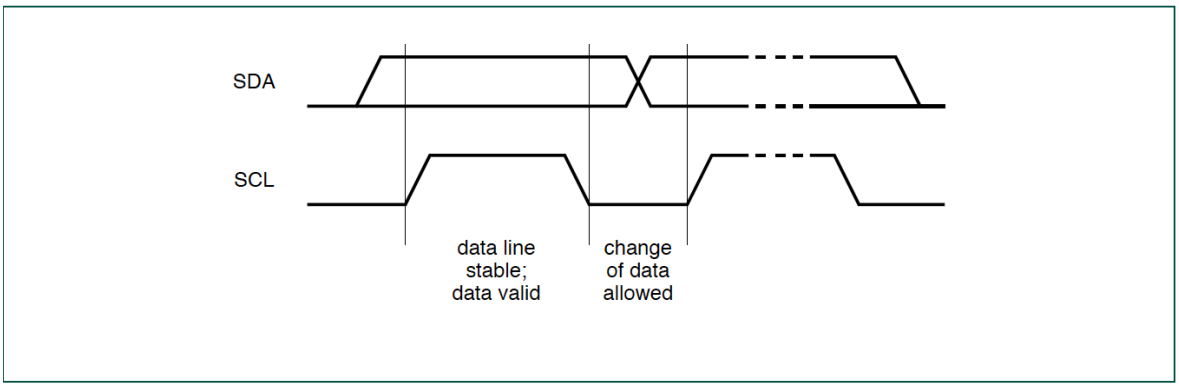

Abbildung 2.3: Bedingung für gültiges Bit auf Datenleitung [3, S. 9]

[5] es gibt mehr als einen Master

START Bedingung

Damit eine Kommunikation stattfinden kann, muss diese als Erstes durch eine Start Bedingung eingeleitet werden. Realisiert wird die Startbedingung, indem der Master die SDA Leitung auf Ground zieht, während die SCL Leitung auf HIGH gesetzt ist. Abbildung 2.4 zeigt diesen Zustand.

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 Potential der Taktleitung die Datenleitung ebenfalls auf HIGH zieht. Grafisch ist dies in Abbildung 2.4 dargestellt.

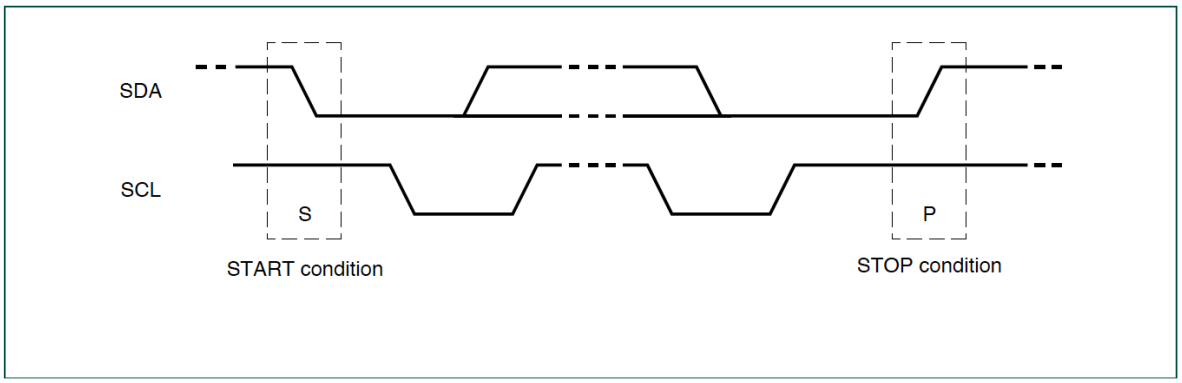

Abbildung 2.4: START, STOP Bedingung I^2C [3, S. 9]

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 maximal 112 Teilnehmer an den Bus betrieben werden.

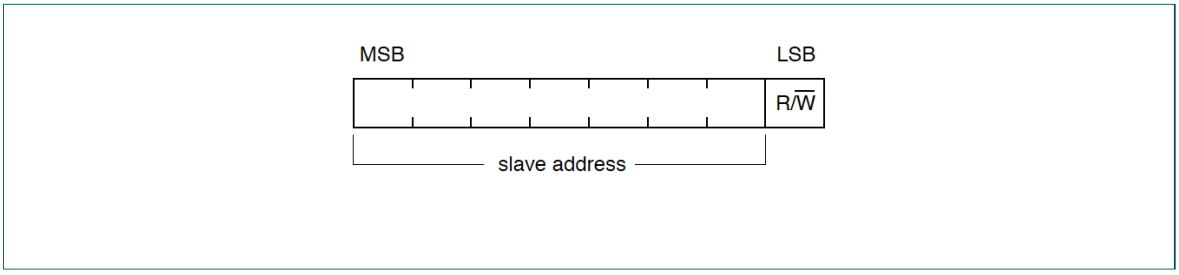

Abbildung 2.5: Erstes Byte nach Start Bedingung I^{2}C [3, S. 13]

Übertragungsprotokoll

Das Übertragungsprotokoll beim I^{2}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 2.6 ist der Ablauf noch einmal grafisch dargestellt.

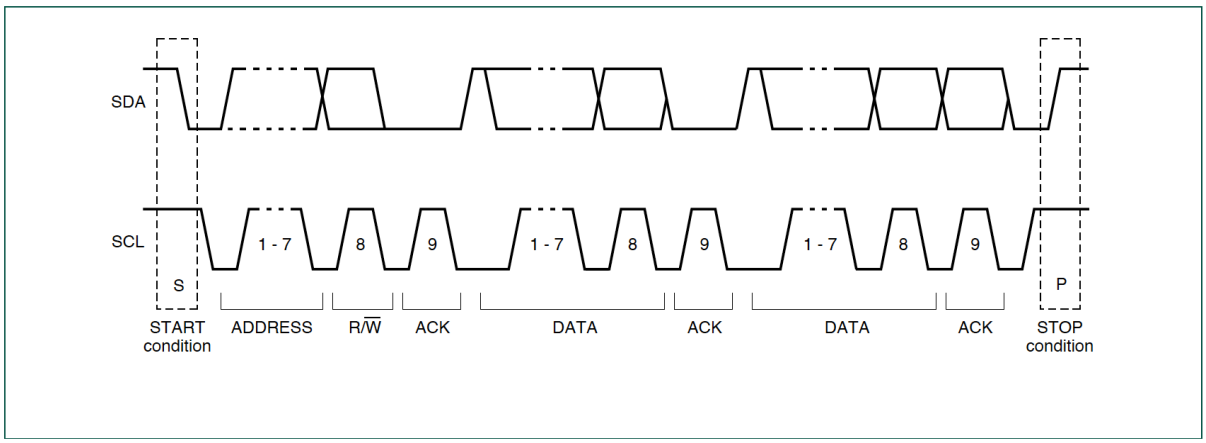

Abbildung 2.6: Datentransfer I^{2}C Bus [3, S. 13]

3 Praktischer Teil

Die Aufgabe war es, verschiedene Möglichkeiten des Monitorings durch den Einsatz des RPI 3 mit unterschiedlichen Sensoren zu erarbeiten. Dafür wurden für die Bereiche Temperatur- , Feuchtigkeits- und Vibrationsermittlung entsprechende Schaltungen erarbeitet. Es wurde ein Schaltplan gezeichnet und entsprechend diesem, die Schaltungen auf einem Steckbrett aufgebaut und die benötigte Software zum Betrieb entwickelt.
Das Ziel dieser Aufgabenstellung war es, mit einfachen Schaltungen die Möglichkeit zu schaffen kritische Entwicklungen im Bereich der Produktion schnell erfassen und darstellen zu können. Dadurch sollte sichergestellt werden, dass eine Fehlerhafte Produktion z.B. durch zu hohe Vibrationen vermieden wird und dadurch Kosteneinsparungen getätigt werden können.
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. Am Anfang dieses Kapitels werden die zur Realisierung der verschiedenen Aufgaben benötigten Bauteile kurz 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 Temperatur- und Luftfeuchtigkeitsmessung 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 nachfolgenden Versuche verwendet. In den jeweiligen Abschnitten, werden die einzelnen Sensoren mit deren wichtigsten Technischen Daten noch genauer erklärt.

Bezeichnung	*Anzahl*
Raspberry Pi 3	1
Temperatursensor DS18S20	2
Sensor HYT 221	1
3-Achsen-Beschleunigungssensor	1
Drahtbrücken	mehrere
elektrischer Widerstand 4.7 $k\Omega$	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. Hier wird auf die Möglichkeit der Datenspeicherung und Visualisierung der Daten mit dem RRDtool eingegangen.

3.2.1 DS18S20

Der DS18S20 (siehe Abbildung **??**) ist ein digitaler 1-Wire Temperatursensor. Dieser ist von der Firma *Maxim Integrated* entwickelt worden. Folgend werden die wichtigsten technischen Daten des DS18S20 aufgeführt.

Parameter	*Minimum*	*Maximum*
Supply Voltage	3 V	5,5 V
Temperature Range	-55 °C	125 °C

Tabelle 3.2: elektische Daten DS18S20 [9]

3.2.2 Schaltungsaufbau

Abbildung 3.1 zeigt den schematischen Schaltungsaufbau mit dem Temperatursensor DS18S20. Anzumerken ist, dass die in der Schaltung in Abbildung 3.1 dargestellten Bauteile optisch nicht immer den realen Bauteilen entsprechen (z.B. Form, Farbe oder Aufdruck). Die Beschaltung der Bauteile ist jedoch korrekt dargestellt.

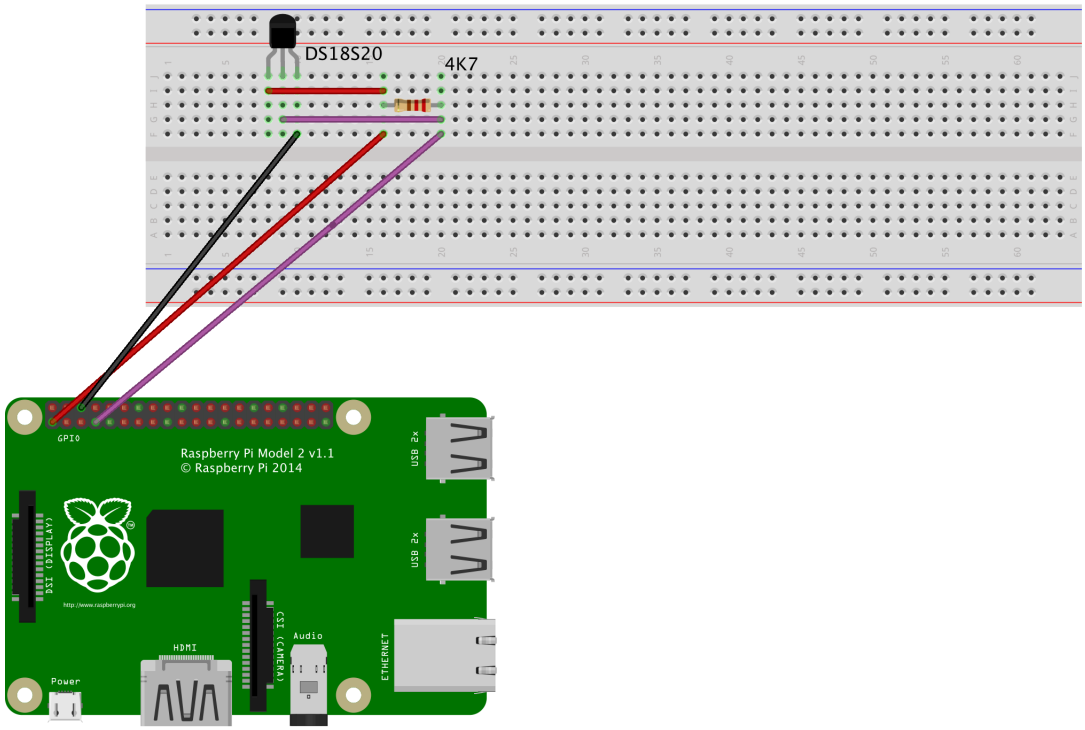

Abbildung 3.1: Schaltungsaufbau DS18S20[1]

RPI 3 Pin	*Sensor Pin*	*Verbindung*
Pin 1 (3,3 V)	V_{DD}	Rot
Pin 6 (GND)	GND	Schwarz
Pin 7 (1-Wire)	DQ	Violett

Tabelle 3.3: Verbindung der einzelnen Pins

Parallel zu dem V_{DD} und DQ Pin ist ein Pullup Widerstand geschalten.

[1] gezeichnet mit Fritzing

3.2.3 Auslesen des Sensors

Um die vom Sensor erfassten Daten auf dem RPI 3 auszulesen, wird ein Python Script verwendet. Dieses wird in Listing 3.1 anhand eines kurzen Ausschnittes erklärt. Der komplette Quellcode ist im Anhang A.1 zu finden.

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

Listing 3.1: Auslesen der gemessenen Temperatur

Um die Temperatur des Sensors auslesen zu können wird in Zeile 2 des Codes das File geöffnet und dessen Inhalt in eine Variable gespeichert. Dieses File enthält die IDs der an den RPI 3 angeschlossenen 1-Wire Sensoren. Jeder dieser Sensoren besitzt im Pfad `/sys/bus/w1/devices/` einen Ordner mit seiner ID. In diesem liegt eine Datei mit der Bezeichnung `w1_slave`, die die gemessenen Daten enthält. Um diese auszulesen, werden in dem Python Script in den Zeilen $8-15$ verschiedene Schritte durchgeführt, auf die nicht näher eingegangen wird[2]. Da die Temperatur in der Datei im Format von z.B. 17687 ($\approx$ 17,7 °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.
Wichtig beim Auslesen der Dateien ist es, darauf zu achten, dass die Temperaturwerte

[2]kann unter `http://www.python-kurs.eu/kurs.php` nachgelesen werden

im richtigen Format ausgelesen werden, da es sonst im späteren Verlauf zu Problemen kommen kann (unübersichtliche Darstellung bei der Visualisierung).

3.2.4 Datenspeicherung

Wie schon in Abschnitt 3.2 angesprochen, wird in diesem Teil der Arbeit auf die Möglichkeit der Datenspeicherung mittels 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

Um eine Datenbank zu erstellen, wurde in Zeile 2 des in 3.2 dargestellten Source Co-

des der Name des zu erstellenden Datenbank Files festgelegt. Weiterhin wurde die Schrittweite[3] auf *60 Sekunden* festgelegt. Die nächsten Zeile definiert die Datenreihe, die Zahlen am Ende der Zeile legen fest, dass wenn innerhalb von 120 Sekunden kein gültiger Wert geliefert wird, die Datenbank an dieser Stelle *NAN* (Not A Number) einträgt. Die Begründung liegt darin, dass wenn im weiteren Verlauf Mittelwerte etc. gebildet werden keine Verfälschung der Daten erfolgt. In Zeile 4 wird der Zeitbereich der Speicherung von den gelieferten Werten definiert. Festgelegt wurde, dass jede Minute ein Wert gespeichert wird 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 ältere Datenwerte zusammengefasst werden, zu definieren. Genauere Informationen bezüglich der Erstellung der Datenbank können unter [15] bezogener werden.
Das in Quellcode 3.2 dargestellte Script musste nach dem Speichern noch ausführbar gemacht werden. Mit dem erfolgreichen Ausführen des Scripts wurde die Datenbank im Verzeichnis `/home/pi/Temperatur/TemperaturSensor/` angelegt.

Datenspeicherung

Damit die über das Python Script ausgelesenen Daten in die Datenbank geschrieben werden, wurde am Ende des in Listing 3.1 dargestellten 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.

[3]Datenbank erwartet alle 60 Sekunden einen Wert

3.2.5 Visualisierung 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 Shell Script verwendet, das den entsprechenden Code enthält. Der Quellcode zur Erstellung der Grafen ist im Anhang dargestellt. Die einzelnen Befehle können auf folgender Quelle nachgeschlagen werden [15].
Abbildung 3.2 und 3.3 zeigen die mittels RRDtool erzeugten Grafen.

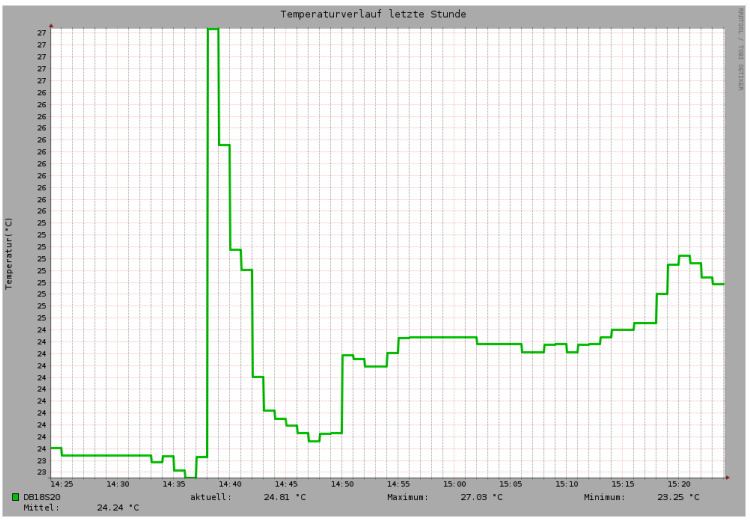

Abbildung 3.2: Temperaturverlauf der letzten Stunde

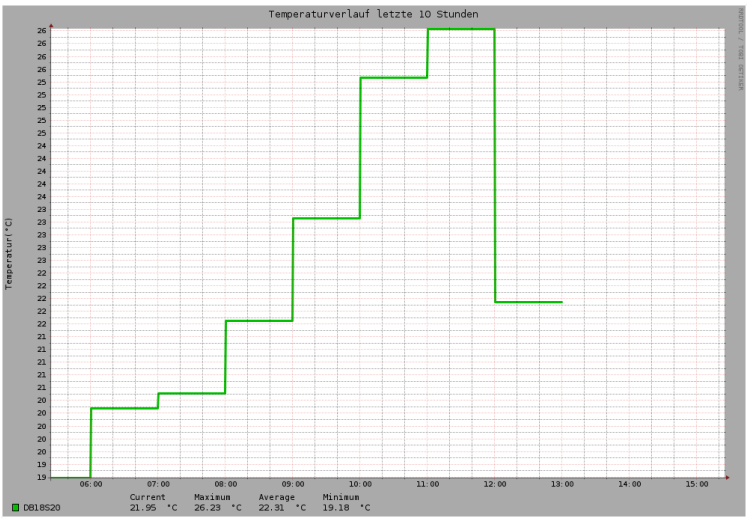

Abbildung 3.3: Temperaturverlauf der letzten 10 Stunden

Um die in diesem Versuch verwendeten Scripte zum Auslesen der Daten, Speichern der Daten oder Erstellung der Grafen für den Temperaturverlauf nicht immer von

„Hand“ausführen zu müssen, wurde für jedes dieser Scripte ein extra Eintrag in der Datei `crontab`[4] vorgenommen. Die Zeit wurde jeweils auf 1 Minute eingestellt.
Durch die Erstellung der oben aufgelisteten Grafen war es möglich den Temperaturverlauf im gemessenen Zeitraum nachzuverfolgen. Der Zeitraum der Messungen wurde über mehrere Tage ausgedehnt, damit eine realistische Temperaturaufzeichnung ausgewertet werden konnte. Anhand der erhaltenen Grafen konnten diverse Rückschlüsse gezogen werden (z.B. sank die Temperatur morgens um sechs Uhr jeweils um zwei Grad ab was über mehrere Tage nachzuverfolgen war. Dies lag daran, dass um diese Uhrzeit immer die Fenster geöffnet wurden).

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^2C Schnittstelle und besitzt eine hohe Messgenauigkeit. In Tabelle 3.4 werden die wichtigsten Daten des Sensors aufgelistet. Diese wurden dem Sensor zugehörigen Datenblatt entnommen [10].

[4]Cron-Daemon = Dienst der automatisch Scripte zu vorgegebenen Zeiten starten kann

Feuchtemessung		**Temperaturmessung**	
Bezeichnung	*Werte*	*Bezeichnung*	*Werte*
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 3.4: Technische Daten HYT-221 [10]

Abbildung 3.4: HYT-221 [4]

Für einen sicheren Betrieb des Sensors muss die angelegte Versorgungsspannung zwischen 2,7...5,5 V liegen.

3.3.2 Schaltungsaufbau

Der in Abbildung 3.5 dargestellte Schaltungsaufbau beinhaltet zwei Sensoren 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^2C Bus betrieben. Für diesen wurde eine $3,3\,V$ Versorgungsspannung (rote Verbindung), ein GROUND Anschluss (schwarze Verbindung), sowie zur Datenübertragung eine Taktleitung (SCL = rote Verbindung) und eine Datenleitung (SDA = blaue Verbindung) benötigt. Beim Aufbau dieser Schaltung war es wichtig, eine einheitliche Verkabelung der einzelnen Komponenten sicherzustellen (rote Verbindungen für 3,3 V, schwarz für Ground etc.), um die Übersichtlichkeit zu bewahren. Ebenso musste darauf geachtet werden, die richtigen Pins miteinander zu verbinden.

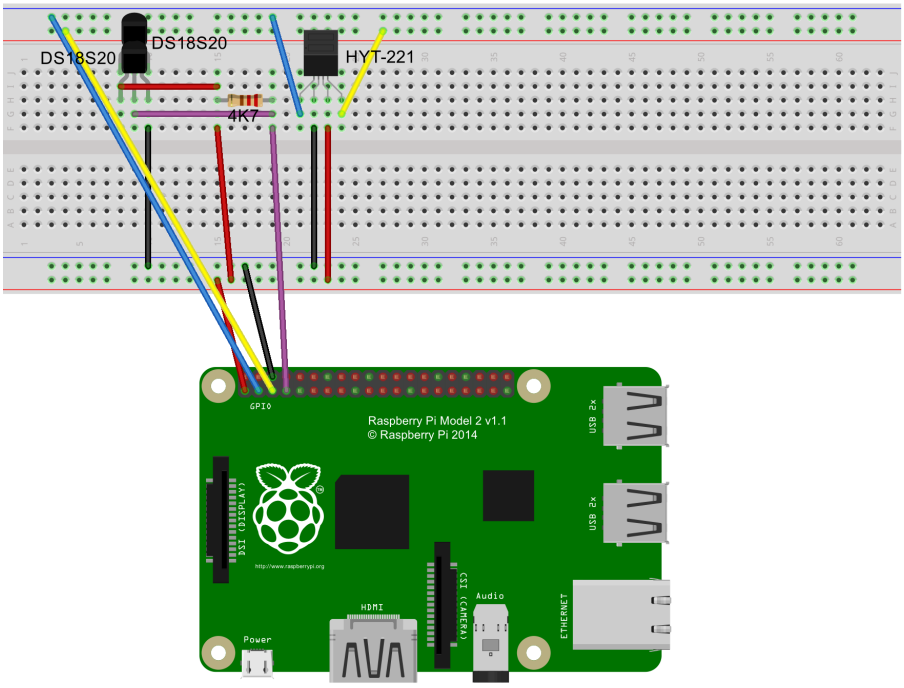

Abbildung 3.5: Schaltungsaufbau mit drei Sensoren

RPI 3 Pin	*DS18S20 Pin*	*HYT-221 Pin*	*Verbindung*
Pin 1 (3,3 V)	V_{DD}	V_{DD}	Rot
Pin 6 (GND)	GND	GND	Schwarz
Pin 7 (1-Wire)	DQ		Violett
Pin 3 (SDA)		SDA	Blau
Pin 5 (SCL)		SCL	Gelb

Tabelle 3.5: Verbindung der einzelnen Pins

3.3.3 Auslesen der Daten

Im folgenden Code wurde das Auslesen der Daten des HYT-221 über den I^2C Bus realisiert. 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 für 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^2C 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 RPI 3 ausgelesen werden. Der Parameter -y bewirkt, dass der Befehl sofort ausgeführt wird, der Parameter 1 legt fest, dass es sich um den I^2C Bus Nummer 1 handelt (andere Hardware kann auch mehrere I^2C Busse besitzen).
Um den smbus verwenden zu können musste ein Objekt dessen erzeugt werden (Zeile 13), die 1 in der Funktion steht wie schon beschrieben für Bus Nummer 1. Im Anschluss wurde das erzeugte Array zur Speicherung der Daten noch initialisiert. Die Funktion in Zeile 22 erzeugt die 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 nacheinander ausgelesen werden. Dies kann in der Protokollbeschreibung für den Sensor in folgender Quelle nachgelesen werden [16]. Die übertragenen Werte, mussten im Anschluss anhand bestimmter Formel in die richtige Form umgerechnet werden. Die Formeln wurden aus dem Datenblatt des Sensors entnommen. Erforderlich war dies, um die Werte 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 konkrete Aussagen über mögliche Ursachen tätigen zu können.
Der Source Code zur Realisierung der Datenspeicherung mittels mySQL DB ist im Anschluss aufgeführt. Die entsprechende DB wurde zuvor auf dem RPI 3 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 zu sehen. 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
        ", 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). Damit die Änderungen in der Tabelle gespeichert werden, musste in Zeile 16 noch ein commit durchgeführt werden. Am Ende wurde die Verbindung zu Datenbank wieder geschlossen.

```
1  ########################## Daten in mySQL speichern
       ###################
2
3  #Cursor zum Eintragen erzeugen
4  cur = db.cursor()
5
6  #SQL-Statment erzeugen
7  sql = "INSERT INTO Temperatur (DB18S20, DB18S20K, HYT22)
       VALUES (%s,%s,%s)"
8  cur.execute(sql,(Temperature_W1[1],Temperature_W1[0],
       HYT_Temperature))
9
10 sql1 = "INSERT INTO Luftfeuchtigkeit (HYT22) VALUES (%s)"
11 cur.execute(sql1,(HYT_Humidity,))
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. Hier wurden die Werte aller drei Sensoren in den Graphen abgebildet, was einen Vergleich der Temperaturwerte zwischen den verschiedenen Sensoren recht leicht machte. Als Ergebnis konnte festgehalten werden, dass sich die Sensoren im Durchschnitt um maximal 0,5 Grad unterschieden haben. Dies spricht wiederum für die Messgenauigkeit der einzelnen verwendeten Sensoren. Auch wurde noch ein weiterer Graph für den Zeitraum von den letzten drei Tagen hinzugefügt. Die Graphen mit den Temperatur und Feuchtigkeitswerten sind 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^2C Bus zur Verfügung. Die wichtigsten technischen Daten wurden dem Datenblatt [11] entnommen und in der Tabelle 3.6 dargestellt.

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

Tabelle 3.6: Technische Daten BMA020 [11]

Abbildung zeigt die fertig bestückte Platine mit dem BMA020.

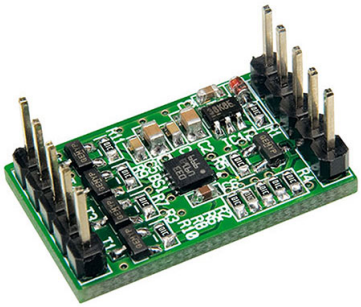

Abbildung 3.6: Platine mit BMA020 [5]

3.4.2 Schaltungsaufbau

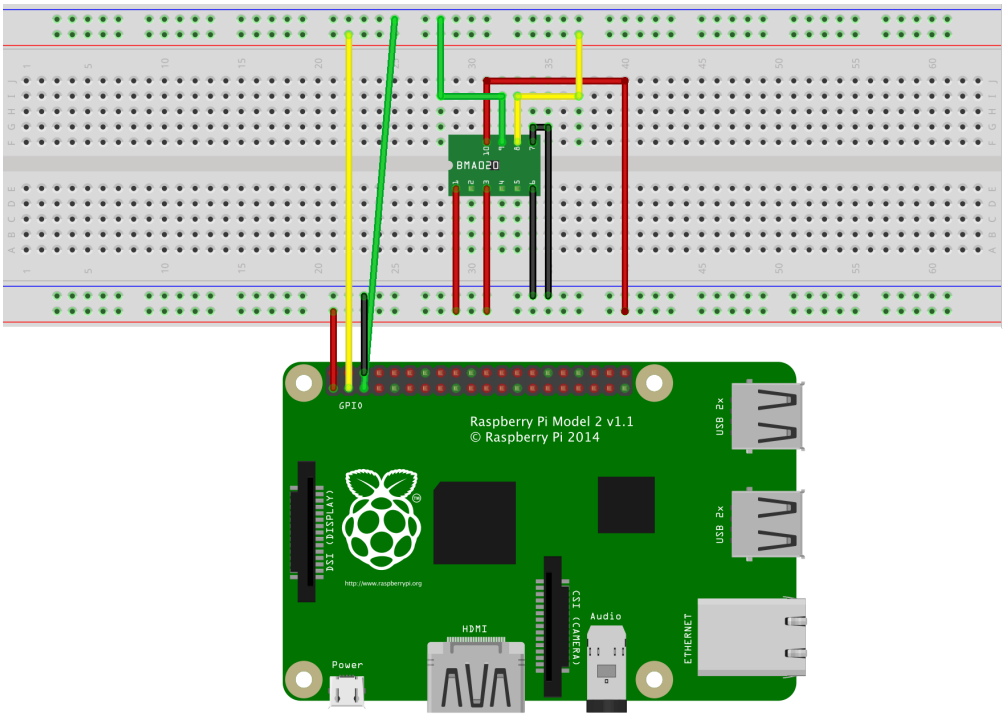

Abbildung 3.7: Schaltungsaufbau Vibrationsmessung mit BMA020

Die in Abbildung 3.7 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 BMA 020 liegt sonst bei max. 3,6 V).
Die Schaltung wurde an Hand des Datenblattes der Platine für einen Betrieb mittles I²C 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 I²C 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^2C 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[5] (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 entsprechnden g-Werte umgerechnet um eine aussagekräftige Darstellung zu ermöglichen. Der Ausschnitt für den eben beschrieben 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
```

[5] 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

Im Gegensatz zu den vorherigen Beispielen, wurde in diesem Fall auf eine Visualisierung mittels Graphen verzichtet, da damit eine Vibrationsmessung in nahezu Echtzeit schwer zu realisieren war. Es hätte hierfür jede Sekunde ein Graph mit dem RDDtool erstellt werden müssen, was eine sehr große Systemlast zur Folge gehabt hätte.
Daher wurde die Auswertung über ein HTML File realisiert, welches mittels AJAX (Asynchronous JavaScript and XML) die aktuellen Beschleunigungswerte mit dem sich am Server befindenden PHP-Script austauscht. Das PHP-Script, das sich auf dem WebServer (wurde auf dem RPI installiert) befindet, ließt die aktuellen Beschleunigungswerte aus einem Textfile aus. Die Daten in diesem Textfile wurden von dem Python Script in dieses geschrieben (Zeile 73 - 75 im Python Script). Die Darstellung der HTML Seite ist im Anhang B.2 ersichtlich. Bei dieser Darstellung wurde noch darauf geachtet, dass bei nicht bedenklichen Beschleunigungswerten die Angaben in grüner Schrift, bei erhöhten Beschleunigungswerten in oranger Schrift und bei kritischen Beschleunigungswerten in roter Schrift dargestellt werden. Wenn nur ein Beschleunigungswert einer Richtung im kritischen Bereich liegt, so wird zusätzlich im unteren Bereich der Anzeige eine Warnmeldung eingeblendet. Dies Logik wurde über

ein JavaScript File realisiert. Die verschiedenen Quelltexte der einzelnen Files sind im Anhang A.7 A.8 A.9 aufgelistet.

4 Zusammenfassung und Ausblick

Diese Arbeit befasste sich damit, die Einsatzmöglichkeiten von SBCs in einem vorhandenen Fertigungsumfeld genauer zu betrachten. Dabei wurde sich speziell auf die Bereiche Temperatur- und Vibrationserfassung fokussiert und den damit verbundenen Möglichkeiten zur Speicherung und Visualisierung der Sensordaten. Eine der ersten Aufgabe war es, die bestmöglichen Komponenten bezüglich SBC und Sensoren auszuwählen um ein möglichst effektives und genaues Messergebnis zu erlangen. Die dafür notwendigen Informationen wurden durch eine Vielzahl von Recherchen erzielt, da es in diesem Bereich wenig bis gar keine Erfahrung gibt.
Nach der Auswahl der Komponenten an Hand bestimmter Faktoren wie Preis, Schnittstellen, Einsatzmöglichkeiten etc. sollte der Betrieb durch verschiedene Prototypen praktisch auf einem Steckbrett ausgetestet werden. Hierfür wurden drei verschiedene Schaltungen wie in Abschnitt 3.2, 3.3 und 3.4 beschrieben aufgebaut und betrieben.

Nachdem das Lesen der Daten von den verschiedenen Sensoren möglich war, wurde sich mit der Datenspeicherung befasst. Die zwei dabei vorgestellten Methoden wurden für die verschiedenen Zwecke getestet und ausgewertet. Am Ende kann der Schluss gezogen werden, dass sich das RRDtool sehr gut zur Erfassung von Temperatur- und Feuchtigkeitsdaten über einen längeren Zeitraum eignet, da es gleichzeitig die Möglichkeit bietet aus diesen Daten die entsprechenden Grafiken zu erstellen. Allerdings stößt es an seine Grenzen, wenn es darum geht Daten in nahezu Echtzeit wiederzugeben.

Die zweite Möglichkeit der Datenspeicherung mittels einer mySQL Datenbank erweist sich dann als sinnvoll, wenn die Daten zu einem späteren Zeitpunkt weiterverarbeitet werden sollen (z.B. in externen Programmen wie MatLab) um andere mögliche äußere Einflüsse mit in Betracht zu ziehen. Möchte man mit der mySQL Datenbank Graphen darstellen wie mit dem RRDtool, so müsste man den Umweg gehen und die Daten als CSV-File exportieren und an Hand dieser dann mit einer anderen Software die Grafik erstellen (z.B. mit Excel).

Als Erkenntnisse zu den durchgeführten Versuchen ergeben sich, dass es kein allzu großes Problem darstellt mit den ausgewählten Komponenten funktionierende Schaltungen und die dazugehörige Software zu realisieren. Auch waren die erhaltenen Messergebnisse zufriedenstellend und ermöglichten einen sehr guten Überblick über

die Gegebenheiten in den Messphasen (Temperaturen, Luftfeuchtigkeit etc.). Daher steht im weiteren Verlauf einem ersten Testeinsatz der einzelnen Schaltungen in dem Produktionsumfeld nichts im Wege. Weiterhin wäre noch angedacht, die einzelnen Schaltungen soweit zu erweitern, dass auch die Möglichkeiten zur Datenauswertung über verschiedenen Wege (z.B. über ein Netzwerk, Internet) möglich wäre.

Die Zukunftsaussichten der in dieser Arbeit getesteten Komponenten, sind durchwegs positiv zu sehen, da es immer mehr große Betrieb gibt, die den Einsatz von SBCs in ihrem Unternehmen testen. Weiterhin kann festgehalten werden, dass SBCs mit den dazugehörigen Sensoren, eine sehr gute Möglichkeit zur Datenerfassung in verschiedenen Bereichen bieten und dies auch noch zu einem Bruchteil der Kosten von professionellen Komponenten. Aus diesem Grund wird sich der Bereich der SBCs in der Zukunft sehr stark weiterentwickeln.

Literaturverzeichnis

[1] E. Upton, „RASPBERRY PI 3 ON SALE NOW AT $35," https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/ (17.04.2016), Februar 2016.

[2] Premier Farnell, „Raspberry Pi 3 Model B GPIO 40 Pin Block Pinout," https://www.element14.com/community/docs/DOC-73950/l/raspberry-pi-3-model-b-gpio-40-pin-block-pinout (18.04.2016).

[3] *I2C-bus specification and user manual*, NXP Semiconductor, April 2014.

[4] Re-In Retail International GmbH, „Feuchte- und Temperatursensor digital Hyt221," http://www.voelkner.de/products/207755/Feuchte-und-Temperaturesensor-digital-Hyt-221.html (21.04.2016).

[5] ELV-/eQ-3-Gruppe, „3-Achsen-Beschleunigungssensor," http://www.elv.de/3-achsen-beschleunigungssensor-3d-bs-komplettbausatz.html (28.04.2016).

[6] D. G. Calin, „Compare 10 Linux single-board computers to find the right one for you," http://www.intorobotics.com/compare-10-linux-single-board-computers-to-find-the-right-one-for-you/ (18.05.2016).

[7] Raspberry PI Foundation, „RASPBERRY PI 3 MODEL B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (16.04.2016).

[8] W. Klaas, *Bussysteme in der Praxis*. Franzis Verlag GmbH, 2015.

[9] *DS18S20 High-Precision 1-Wire Digital Thermometer*, Maxim Integrated Products, 2015.

[10] *HYGROCHIP DIGITALER FEUCHTESENSOR HYT-221*, INNOVATIVE SENSOR TECHNOLOGY, September 2011.

[11] *BMA020 Digital, triaxial acceleration sensor*, 1. Aufl., Bosch Sensortec GmbH, Mai 2008.

[12] Kofler, Kühnast, und Scherbeck, *Raspberry Pi Das umfassende Handbuch*. Rheinwerk Verlag GmbH, Bonn, 2015.

[13] Bluetooth SIG, „Bluetooth Low Energy," https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy (18.04.2016).

[14] L. Upton, „SENIOR PI," https://www.raspberrypi.org/blog/senior-pi/ (17.04.2016), Raspberry Pi Foundation, Oktober 2015.

[15] T. Oetiker, „About RRDtool," http://oss.oetiker.ch/rrdtool/ (25.04.2016), OETIKER+PARTNER AG, September 2014.

[16] *DIGITALER FEUCHTESENSOR PROTOKOLLBESCHREIBUNG I2C*, HYGROSENS INSTRUMENTS GmbH, May 2010.

A Quellcode

A.1 Python Script DS18S20

```
#!/usr/bin/python3
# -*-coding: utf-8 -*-

import sys
import os

#1-Wire Slave-Liste lesen
file = open('/sys/devices/w1_bus_master1/w1_master_slaves')
w1_slaves = file.readlines()
file.close()


#Fuer jeden 1-Wire Slave aktuelle Temperatur ausgeben
for line in w1_slaves:
    w1_slave = line.split("\n")[0]
    file = open('/sys/bus/w1/devices/'+str(w1_slave) + '/
       w1_slave')
    filecontent = file.read()
    file.close()

    stringvalue = filecontent.split("\n")[1].split(" ")[9]
    temperature = float(stringvalue[2:])/1000
    temperature = round(temperature,2)
    print(str(w1_slave) + ': %6.2f C' % temperature)

################ Sensor-Daten speichern #############

ret = rrd_update('/home/pi/Temperatur/TemperaturSensor/
   TemperaturAufzeichnungDB18S20.rrd','N:%s'%(temperature));

sys.exit(0)
```

Listing A.1: Quellcode zum Auslesen des DS18S20

A.2 Erstellung Graphen DS18S20

```
#!/bin/sh

#Grafik erstellen
sudo rrdtool graph /var/www/html/TemperaturTag.png \
-w 900 -h 600 -t "Temperaturverlauf letzte 10 Stunden" \
--end now --start end-10h -v "Temperatur(°C)" --slope-mode \
--x-grid MINUTE:20:HOUR:1:MINUTE:60:0:%R \
-c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#
   ffffff \
--alt-autoscale --rigid \
DEF:Temperatur1=/home/pi/Temperatur/TemperaturSensor/
   TemperaturAufzeichnungDB18S20.rrd:Temperatur1:AVERAGE \
DEF:Temperatur2=/home/pi/Temperatur/TemperaturSensor/
   TemperaturAufzeichnungDB18S20.rrd:Temperatur2:AVERAGE \
VDEF:Temp1Cur=Temperatur1,LAST \
VDEF:Temp1Max=Temperatur1,MAXIMUM \
VDEF:Temp1Avg=Temperatur1,AVERAGE \
VDEF:Temp1Min=Temperatur1,MINIMUM \
VDEF:Temp2Cur=Temperatur2,LAST \
VDEF:Temp2Max=Temperatur2,MAXIMUM \
VDEF:Temp2Avg=Temperatur2,AVERAGE \
VDEF:Temp2Min=Temperatur2,MINIMUM \
COMMENT:"                      " \
COMMENT:"Current    " \
COMMENT:"Maximum    " \
COMMENT:"Average    " \
COMMENT:"Minimum \l" \
LINE3:Temperatur1#00C000:"DB18S20" \
GPRINT:Temp1Cur:"          %6.2lf %S°C" \
GPRINT:Temp1Max:"%6.2lf %S°C" \
GPRINT:Temp1Avg:"%6.2lf %S°C" \
GPRINT:Temp1Min:"%6.2lf %S°C\l" \
LINE2:Temperatur2#0000FF:"DB18S20 mit Kabel" \
GPRINT:Temp2Cur:"%6.2lf %S°C" \
GPRINT:Temp2Max:"%6.2lf %S°C" \
GPRINT:Temp2Avg:"%6.2lf %S°C" \
GPRINT:Temp2Min:"%6.2lf %S°C\l" \

sudo rrdtool graph /var/www/html/TemperaturStunde.png \
-w 900 -h 600 -t "Temperaturverlauf letzte Stunde" \
--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

```
#!/usr/bin/python
# *-* coding: utf-8 *-*
import smbus
import time
import rrdtool
import sys
import os
from rrdtool import update as rrd_update
from time import sleep


############################### HYT 221
   ###################################

#I2C-Adresse

SensorAdresse = 0x28;


#Array fuer die Sensordaten

SensorDaten = bytearray()


#Sensor ID für HYT221

ID_HYT = "HYT_221"

#smbus Objekt fuer I2C bus #1

Sensor = smbus.SMBus(1)


#Initialisierung Array

SensorDaten.append(0x30)
SensorDaten.append(0x31)
SensorDaten.append(0x32)
SensorDaten.append(0x33)


#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
71 ################################# DB18S20
      ################################
72
73 #1-Wire Slave-Liste lesen
74
75 file = open('/sys/devices/w1_bus_master1/w1_master_slaves')
76 w1_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 w1_slaves:
84
85     w1_slave = line.split("\n")[0]
```

```
86      file = open('/sys/bus/w1/devices/'+str(w1_slave) + '/
           w1_slave')
87      filecontent = file.read()
88      file.close()
89
90      stringvalue = filecontent.split("\n")[1]
91
92      stringvalue = stringvalue.split(" ")[9]
93
94      Temperature_W1.append(float(stringvalue[2:])/1000)
95
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', '
       N:%s:%s:%s:%s' %(HYTTemp, DB18S20KabelTemp, DB18S20Temp,
       HYTFeucht));
```

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

A.4 Python Script HYT-221 mit mySQL

```
#!/usr/bin/python3
# *-* coding: utf-8 *-*
import smbus
import time
import sys
import os
import mysql.connector
from time import sleep

############################## Datenbank
   ##############################

#Datenbank Verbindung herstellen
try:
    db = mysql.connector.connect(host="localhost", user="root
       ", passwd="test123", db="RPI-Projekt")
except :
    print("No connection to database")
    exit(0)


################ HYT 221 ############################

#I2C-Adresse
SensorAdresse = 0x28;

#Array fuer die Sensordaten
SensorDaten = bytearray()

#Sensor ID für HYT221
ID_HYT = "HYT_221"

#smbus Objekt fuer I2C bus #1
Sensor = smbus.SMBus(1)

#Initialisierung Array
SensorDaten.append(0x30)
SensorDaten.append(0x31)
SensorDaten.append(0x32)
SensorDaten.append(0x33)

#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 w1_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 w1_slaves:
81     Slave_W1.append(line.split("\n")[0])
82     w1_slave = line.split("\n")[0]
83     file = open('/sys/bus/w1/devices/'+str(w1_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)
           ,2))
92
93
94
95  ############################ Daten in mySQL speichern
       ####################
96
97  #Cursor zum Eintragen erzeugen
98  cur = db.cursor()
99
100 #SQL-Statment erzeugen
101 sql = "INSERT INTO Temperatur (DB18S20, DB18S20K, HYT22)
       VALUES (%s,%s,%s)"
102 cur.execute(sql,(Temperature_W1[1],Temperature_W1[0],
       HYT_Temperature))
103
104 sql1 = "INSERT INTO Luftfeuchtigkeit (HYT22) VALUES (%s)"
105 cur.execute(sql1,(HYT_Humidity,))
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

```
#!/bin/sh

#Grafik erstellen
sudo rrdtool graph /var/www/html/Messung/Bilder/TemperaturStunde.png \
-w 900 -h 600 -t "Temperaturverlauf letzte Stunde" \
--end now --start end-1h -v "Temperatur(°C)" --y-grid=0.2:5 --slope-mode --no-gridfit \
--grid-dash 1:0 \
--x-grid MINUTE:1:MINUTE:15:MINUTE:5:0:%R \
-c GRID#000000 -c BACK#aaaaaa -c SHADEA#ffffff -c SHADEB#ffffff \
DEF:HYT_Temp=/home/pi/Temperatur/Messung/Messung.rrd:HYT_Temp:AVERAGE \
DEF:DB18S20_Temp=/home/pi/Temperatur/Messung/Messung.rrd:DB18S20_Temp:AVERAGE \
DEF:DB18S20_Kabel_Temp=/home/pi/Temperatur/Messung/Messung.rrd:DB18S20_Kabel_Temp:AVERAGE \
VDEF:Temp1Cur=HYT_Temp,LAST \
VDEF:Temp1Max=HYT_Temp,MAXIMUM \
VDEF:Temp1Avg=HYT_Temp,AVERAGE \
VDEF:Temp1Min=HYT_Temp,MINIMUM \
VDEF:Temp2Cur=DB18S20_Temp,LAST \
VDEF:Temp2Max=DB18S20_Temp,MAXIMUM \
VDEF:Temp2Avg=DB18S20_Temp,AVERAGE \
VDEF:Temp2Min=DB18S20_Temp,MINIMUM \
VDEF:Temp3Cur=DB18S20_Kabel_Temp,LAST \
VDEF:Temp3Max=DB18S20_Kabel_Temp,MAXIMUM \
VDEF:Temp3Avg=DB18S20_Kabel_Temp,AVERAGE \
VDEF:Temp3Min=DB18S20_Kabel_Temp,MINIMUM \
COMMENT:"                      " \
COMMENT:"Current    " \
COMMENT:"Maximum    " \
COMMENT:"Average    " \
COMMENT:"Minimum \l" \
LINE2:HYT_Temp#4CCDFF:"HYT 221" \
GPRINT:Temp1Cur:"           %2.2lf %S°C" \
GPRINT:Temp1Max:" %2.2lf %S°C" \
GPRINT:Temp1Avg:" %2.2lf %S°C" \
GPRINT:Temp1Min:" %2.2lf %S°C\l" \
LINE2:DB18S20_Temp#22BA32:"DB18S20 mit Kabel" \
GPRINT:Temp2Cur:" %2.2lf %S°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 %S°C" \
77 GPRINT:Temp1Avg:" %2.2lf %S°C" \
78 GPRINT:Temp1Min:" %2.2lf %S°C\l" \
79 LINE2:DB18S20_Temp#22BA32:"DB18S20 mit Kabel" \
80 GPRINT:Temp2Cur:" %2.2lf %S°C" \
81 GPRINT:Temp2Max:" %2.2lf %S°C" \
82 GPRINT:Temp2Avg:" %2.2lf %S°C" \
83 GPRINT:Temp2Min:" %2.2lf %S°C\l" \
84 LINE2:DB18S20_Kabel_Temp#AF2834:"DB18S20" \
85 GPRINT:Temp3Cur:"           %2.2lf %S°C" \
86 GPRINT:Temp3Max:" %2.2lf %S°C" \
87 GPRINT:Temp3Avg:" %2.2lf %S°C" \
88 GPRINT:Temp3Min:" %2.2lf %S°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

```
#!/usr/bin/python3
# *-* coding: utf-8 *-*
import smbus
import sys
import os
import time
import math
import mysql.connector

from math import sqrt
from time import sleep
from smbus import SMBus

# Verbindung zur Datenbank

try:
    db = mysql.connector.connect(host="localhost", user="root
       ", passwd="test123", db="RPI-Projekt")
except:
    print("No connection to database")
    exit(0)

sql = "INSERT INTO Vibration (XWERT, YWERT, ZWERT) VALUES(%s
   ,%s,%s)"

#I2C Adresse
add = 0x38

#smbus Objekt für I2C erstellen
BMA020 = smbus.SMBus(1)

#Funktionsdefinition zum Auslesen
def read_word(reg):
    LSB = BMA020.read_byte_data(add, reg)
    LSB = LSB - 1
    MSB = BMA020.read_byte_data(add, reg+1)
    value = (MSB<<2)+(LSB>>6)
    return value

def read_word_2c(reg):
    val = read_word(reg)
    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")
86      sleep(1)
```

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

A.7 PHP-Script zum Auslesen der Beschleunigungswerte

```
<?php
    $file = '/home/pi/Projekt/Vibration/Daten.txt';
    $lines = file($file);
    $Daten = explode(" ",$lines[0]);
?>

<?php echo $Daten[0],' ',$Daten[1],' ',$Daten[2];?>
```

Listing A.7: PHP File zum Auslesen der Beschleunigungswerte aus Textfile

A.8 JavaScript File zur Visualisierung der Beschleunigungswerte

```
1  //Script Vibration
2  $(document).ready(function()
3  {
4      //AJAX Funktion
5      function setVib()
6      {
7          $.get("BMA.php", function(data)
8          {
9              var Daten = data.split(" ");
10             $("#Warnung").css("visibility", 'hidden');
11             $("#xWert").css("color", "green");
12             $("#yWert").css("color", "green");
13             $("#zWert").css("color", "green");
14
15             if(Daten[0] > 0.200 || Daten[0] < -0.200)
16             {
17                 $("#xWert").css("color", "orange");
18             }
19             if(Daten[1] > 0.200 || Daten[1] < -0.200)
20             {
21                 $("#yWert").css("color", "orange");
22             }
23             if(Daten[2] > -0.800 || Daten[2] < -1.200)
24             {
25                 $("#zWert").css("color", "orange");
26             }
27             if(Daten[0] > 0.500 || Daten[0] < -0.500)
28             {
29                 $("#xWert").css("color", "red");
30             }
31             if(Daten[1] > 0.500 || Daten[1] < -0.500)
32             {
33                 $("#yWert").css("color", "red");
34             }
35             if(Daten[2] > -0.500 || Daten[2] < -1.500)
36             {
37                 $("#zWert").css("color", "red");
38             }
39
40             if(Daten[0] > 0.500 || Daten[0] < -0.500 || Daten
                   [1] > 0.500 || Daten[1] < -0.500 || Daten[2] >
```

```
                -0.500 || Daten[2] < -1.500)
41              {
42                  $("#Warnung").css("visibility", 'visible');
43                  blinker();
44              }
45
46              document.getElementById('xWert').innerHTML =
                   Daten[0] + " g";
47              document.getElementById('yWert').innerHTML =
                   Daten[1] + " g";
48              document.getElementById('zWert').innerHTML =
                   Daten[2] + " g";
49          });
50      };
51      setInterval(setVib, 100);
52
53      //Warnmeldung Blinken
54      function blinker()
55      {
56              $('#Warnung').fadeOut(500);
57              $('#Warnung').fadeIn(500);
58      }
59  });
```

Listing A.8: Quellcode JavaScript zur Darstellung der Vibrationswerte

A.9 HTML File zur Darstellung der Beschleunigungswerte

```
<!DOCTYPE html>
<html>

<head>
    <meta name="viewport" content="width=device-width,
       initial-scale=1">
    <meta charset="utf-8">
    <link href="styleVib.css" rel="stylesheet" type="text/css
       " />
    <link rel="stylesheet" href="bootstrap/css/bootstrap.min.
       css"/>
    <script type="text/javascript" src="jquery/jquery-1.12.3.
       min.js"></script>
    <script src="bootstrap/js/bootstrap.min.js"></script>
    <script src="script.js"></script>
</head>

<body>

<div class="container-fluid text-center">

    <!--Navigation Bar-->
    <nav class="navbar navbar-inverse">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand" href="#">
                   Vibrationsmessung</a>
            </div>
            <ul class="nav navbar-nav">
                <li class="action">
                    <a href="#">
                        Home
                    </a>
                </li>
                <li><a href="http://localhost/Messung/index.
                   html">Temperaturauswertung</a></li>
            </ul>
            <ul class="nav navbar-nav navbar-right">
                <li>
                    <a href="http://localhost/phpmyadmin/">
                        <span class="glyphicon glyphicon-log-
                           in" ></span> SQL Login
```

```
36                      </a>
37                  </li>
38              </ul>
39          </div>
40      </nav><br>
41      <!--Ende Navigation Bar-->
42
43      <!--Beschriftung Achsen-->
44      <div class="row" >
45
46          <div class="col-sm-4 div_text">
47              <p><strong>X-ACHSE</strong></p>
48          </div>
49
50          <div class="col-sm-4 div_text">
51              <p><strong>Y-ACHSE</strong></p>
52          </div>
53
54          <div class="col-sm-4 div_text">
55              <p><strong>Z-ACHSE</strong></p>
56          </div>
57      <!--Ende Beschriftung-->
58
59      <!--Achsen-Werte-->
60      </div><br>
61
62      <div class="row">
63          <div class="col-sm-4">
64              <div class="div_kreis img-circle">
65                  <span id="xWert">X</span>
66              </div>
67          </div>
68
69          <div class="col-sm-4">
70              <div class="div_kreis img-circle">
71                  <span id="yWert">Y</span>
72              </div>
73          </div>
74
75          <div class="col-sm-4">
76              <div class="div_kreis img-circle">
77                  <span id="zWert">Z</span>
78              </div>
79          </div>
80      </div>
```

```
81      <!--Ende Achsen-Werte-->
82
83
84      <div class="row">
85          <div class="col-sm-4 text-left" style="margin-top:
               300px">
86              <p class="text_service"><strong>Service</strong><
                   /p>
87              <p class="text_service" id="Name"><span class="
                   glyphicon glyphicon-user"></span> Christian
                   Auer</p>
88              <p class="text_service"><span class="glyphicon
                   glyphicon-map-marker"></span> (HlP/TEF1)
                   Container 207</p>
89              <p class="text_service"><span class="glyphicon
                   glyphicon-phone"></span> +43(6245)792-6014</p>
90              <p class="text_service"><span class="glyphicon
                   glyphicon-envelope"></span> Christian.
                   Auer2@bosch.com</p>
91          </div>
92
93          <div class="col-sm-4">
94
95          </div>
96
97          <div class="col-sm-4">
98
99          </div>
100     </div>
101
102     <!--Warnmeldung-->
103     <p align="center" id="Warnung">Achtung Vibration!!!</p>
104     <!--Ende Warnmeldung-->
105
106 </div>
107 <!--div Container -->
108
109 </body>
110
111 </html>
```

Listing A.9: Quellcode HTML File zur Darstellung der Vibrationswerte

B Abbildungen

B.1 Graphen Temperatur und Luftfeuchtigkeit

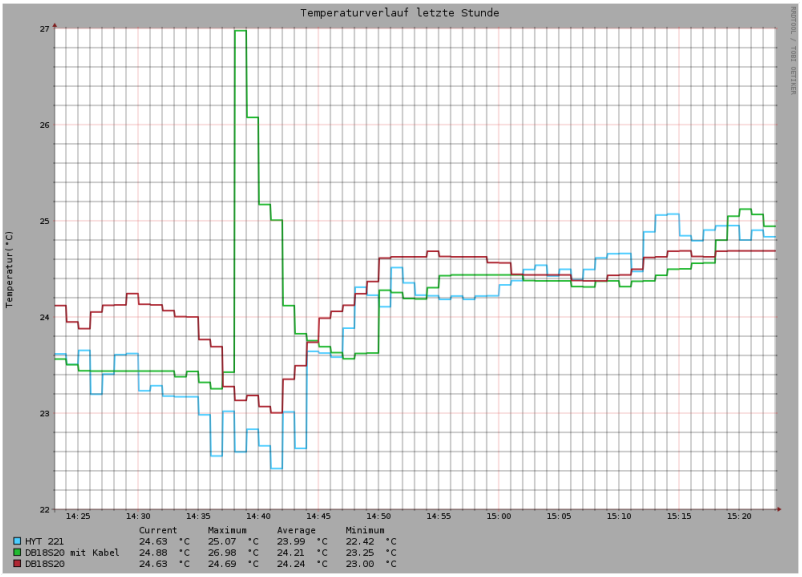

Abbildung B.1: Temperaturverlauf der letzten Stunde

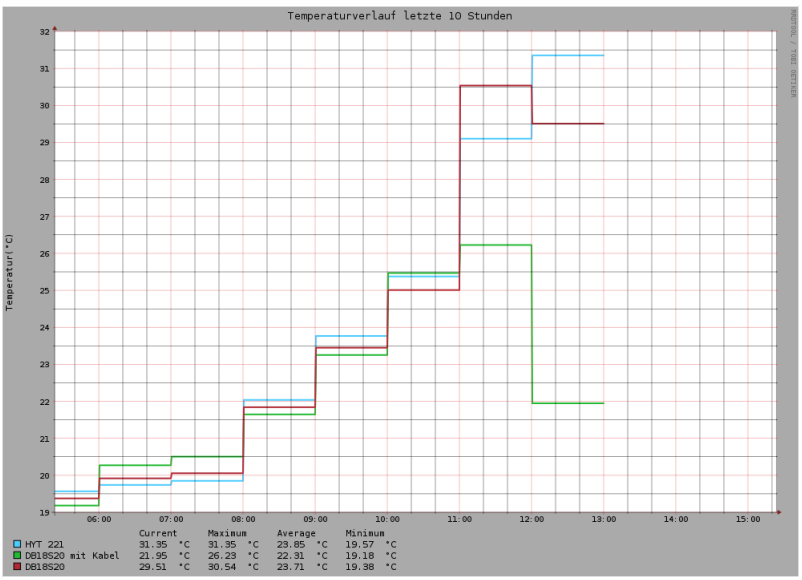

Abbildung B.2: Temperaturverlauf der letzten 10 Stunden

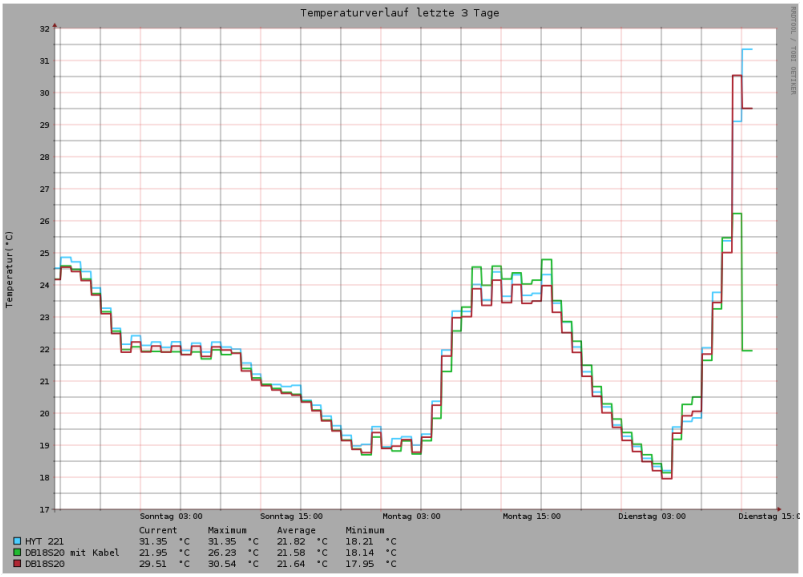

Abbildung B.3: Temperaturverlauf der letzten 3 Tage

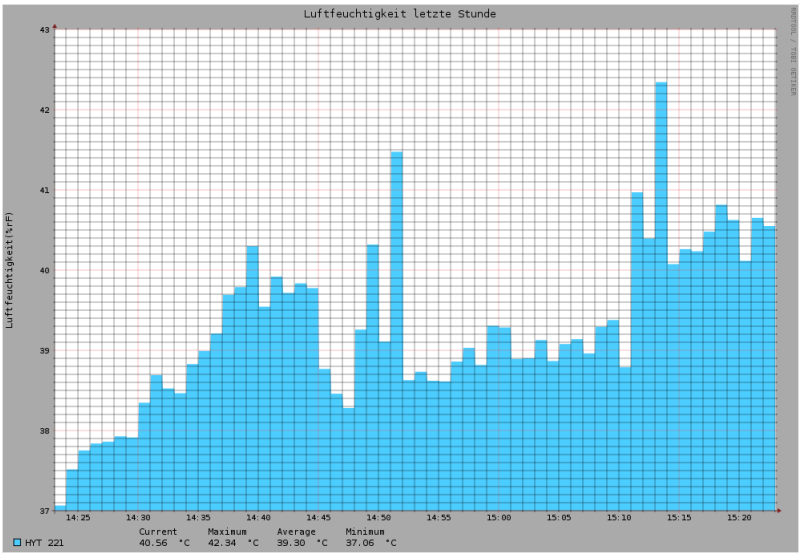

Abbildung B.4: Luftfeuchtigkeit der letzten Stunde

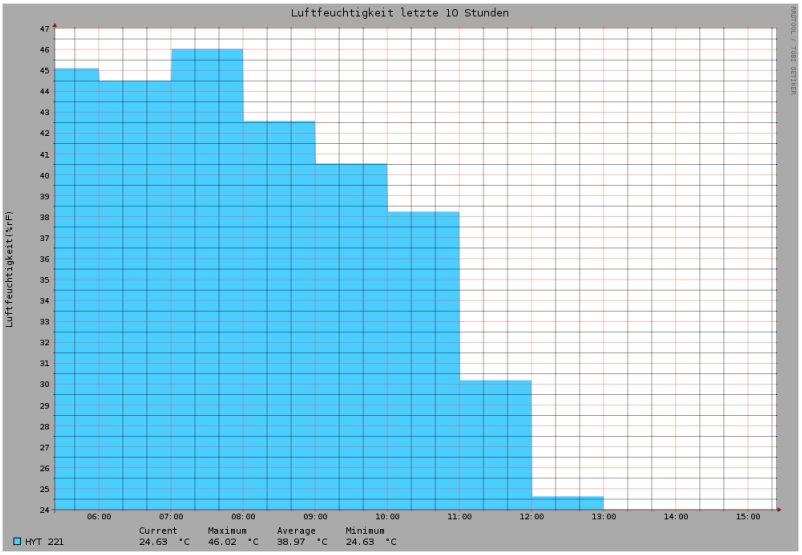

Abbildung B.5: Luftfeuchtigkeit der letzten 10 Stunden

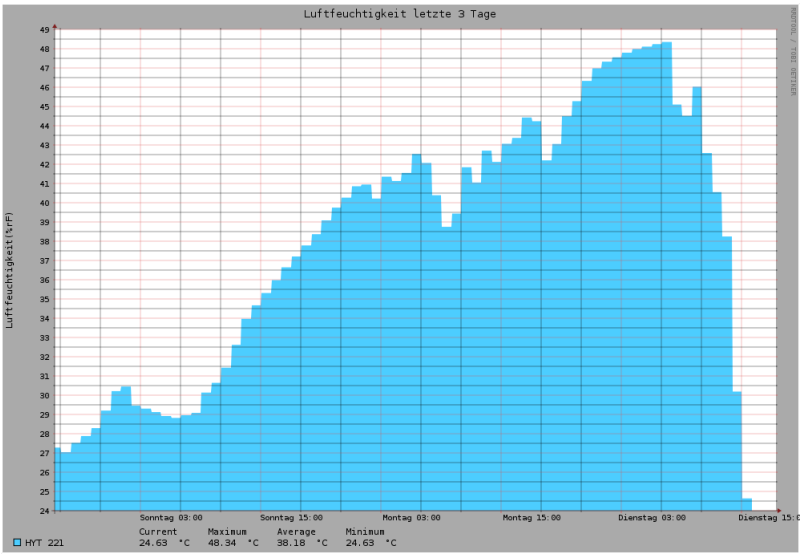

Abbildung B.6: Luftfeuchtigkeit der letzten 3 Tage

B.2 Visualisierung der Vibrationswerte

Abbildung B.7: Vibration im normalen Bereich

Abbildung B.8: Vibration im kritischen Bereich