

# Sensorknoten - Internet der Dinge

## Studienarbeit

für die Prüfung zum

**Bachelor of Engineering**

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Jan Gerber und David Preiß**

Abgabedatum 15. Mai 2017

Bearbeitungszeitraum

Matrikelnummer

Kurs

Gutachter der Dualen Hochschule

Theoriephase 5. & 6. Semester

5757291 (Jan Gerber) 3199578(David Preiß)

TINF14B3

Prof. Hans-Jörg Haubner

## **Erklärung**

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

---

Ort      Datum

---

Unterschrift

---

Ort      Datum

---

Unterschrift

## **Zusammenfassung**

Diese Studienarbeit thematisiert sich mit einem Sensorknoten für das Internet der Dinge. Hierzu werden die theoretischen und technischen Grundlagen des Raspberry Pis und des Arduinos vorgestellt. Die Umsetzung beschäftigt sich mit der Planung und Entwicklung des Knotens und der dafür benötigter Komponenten.

Im Vordergrund der Arbeit steht die Umsetzung der Sensorknoten mit Hilfe eines Funkmoduls. Sie kommunizieren über eine Art Mesh-Netz untereinander und mit dem Raspberry Pi.

Motivation der Studienarbeit ist der Wunsch nach einem skalierbaren und energiesparendem Sensorknoten. Er soll die gesammelten Messdaten in einer Datenbank speichern und diese anschließend visualisiert darstellt. Ferner sollen die Bedürfnisse des Nutzers nach einem *Plug and Play* System nicht vernachlässigt werden.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Ziel dieser Arbeit . . . . .	1
1.2 Aufgabenstellung . . . . .	1
1.3 Stand der Technik . . . . .	1
1.4 Motivation und Vorausblick . . . . .	2
<b>2 Problemstellung</b>	<b>3</b>
2.1 Ist-Zustand . . . . .	3
2.2 Soll-Zustand . . . . .	4
<b>3 Theoretische und technische Grundlagen</b>	<b>6</b>
3.1 Internet of Things . . . . .	6
3.2 Arduino . . . . .	8
3.2.1 AVR Microcontroller . . . . .	9
3.2.2 Arduino Nano / Pro Mini . . . . .	13
3.2.3 Software . . . . .	14
3.3 Raspberry Pi . . . . .	15
3.3.1 Hardware . . . . .	15
3.3.2 Software . . . . .	24
3.4 Maschine-zu-Maschine Kommunikation . . . . .	26
3.5 Sensorik . . . . .	27
3.5.1 Allgemein . . . . .	28
3.5.2 Temperatur . . . . .	30
3.5.3 Luftfeuchtigkeit . . . . .	30
3.5.4 Beleuchtungsstärke . . . . .	31
3.5.5 Bewegungsmelder . . . . .	31
3.5.6 Luftdruck . . . . .	32
3.5.7 Bodenfeuchte . . . . .	32
3.6 Mesh-Netze . . . . .	32
3.7 Webservice mit REST . . . . .	34
<b>4 Konzeption</b>	<b>37</b>
4.1 Mesh-Netz . . . . .	37
4.2 Anbindung Pi/Arduino . . . . .	38
4.3 Messwerterfassung mit Arduino . . . . .	41
<b>5 Realisierung/Implementierung</b>	<b>43</b>
5.1 Topologie . . . . .	43
5.1.1 Allgemeine Beschreibung . . . . .	43

5.1.2	Umsetzung des Mesh-Algorithmus . . . . .	45
5.2	Verwendete Sensoren . . . . .	47
5.3	Hardware Design Sensorknoten . . . . .	50
5.3.1	Arduino Pro Mini . . . . .	50
5.3.2	Arduino Nano . . . . .	51
5.3.3	Produktionsprozess der Sensorknoten . . . . .	52
5.4	Energiesparmodus Arduino Pro Mini . . . . .	56
5.5	Raspberry Pi . . . . .	57
5.5.1	Verkabelung . . . . .	57
5.5.2	Receiver . . . . .	57
5.5.3	Kodierung . . . . .	58
5.5.4	Hashing . . . . .	59
5.5.5	Datenbankanbindung . . . . .	60
5.5.6	Scheduling . . . . .	60
5.5.7	Logging . . . . .	61
5.6	Datenbankschema . . . . .	61
5.7	Webservice . . . . .	62
5.8	Datenvizualisierung . . . . .	64
<b>6</b>	<b>Lösungsbewertung und Fazit</b>	<b>67</b>
6.1	Vergleich Ist-/Soll-Zustand . . . . .	67
6.2	Bewertung des Sensorknotens . . . . .	68
6.3	Zusammenfassung . . . . .	69
6.4	Ausblick . . . . .	70
	<b>Literaturverzeichnis</b>	<b>70</b>

# Abbildungsverzeichnis

3.1	Verfügbarkeit von IPv4 Adressen bei den beiden IP-Vergabeorganisationen IANA und RIR . . . . .	7
3.2	Aufbau eines AVR Mikrocontrollers . . . . .	10
3.3	Arduino IDE 1.8.1 . . . . .	15
3.4	Überblick über die Register . . . . .	17
3.5	Das Statusregister im Detail . . . . .	17
3.6	Virtuelle Adressierung . . . . .	20
3.7	I2C im freien Zustand . . . . .	21
3.8	Startbedingung I2C . . . . .	22
3.9	Logische 0 bei I2C . . . . .	22
3.10	Logische 1 bei I2C . . . . .	22
3.11	I2C Acknowledgement . . . . .	23
3.12	I2C No-Acknowledgement . . . . .	23
3.13	SPI Schema . . . . .	23
3.14	PyCharm IDE . . . . .	25
3.15	nRF24L01 Funkmodule . . . . .	29
3.16	Schema einer Regelung . . . . .	29
3.17	PT100 Temperatursensor . . . . .	30
3.18	Photoelektrischer Effekt . . . . .	31
3.19	Schaltplan eines IR-Bewegungsmelders . . . . .	32
3.20	Drucksensor . . . . .	32
3.21	Mesh Netzwerk Topologie . . . . .	33
4.1	Topologie Mesh-Netz . . . . .	37
4.2	Programmablaufplan - Mesh- Algorithmus . . . . .	39
4.3	Eagle Autodesk – Layout Editor und Schaltplan Editor . . . . .	42
5.1	Topologie Sensorknoten . . . . .	44
5.2	Verwendete Sensoren . . . . .	47
5.3	Energiesparender Sensorknoten . . . . .	50
5.4	Normaler Sensorknoten . . . . .	51
5.5	Prototyp Sensorknoten . . . . .	53
5.6	Gerber-Viewer – Seeedstudio.com . . . . .	54
5.7	Platine für normaler Sensorknoten . . . . .	55
5.8	Platine für energiesparender Sensorknoten . . . . .	55
5.9	Schema der Datenbank . . . . .	61
5.10	Webseite: Übersicht aller Sensorknoten . . . . .	65
5.11	Webseite: Aktuelle Übersicht eines Sensorknotens . . . . .	65
5.12	Webseite: Grafik eines Messwertedatensatz . . . . .	66

# **Tabellenverzeichnis**

3.1	ARM-Modi . . . . .	16
3.2	Modi im ARM Statusregister . . . . .	18
5.1	Stromstärken bei den verschiedenen Energiespartechniken . . . . .	57

# **Listingverzeichnis**

5.1	Auschnitt Mesh Algorithmus . . . . .	45
5.2	Anpassung RF24 Bibliothek . . . . .	46
5.3	DHT22 Sensorauswertung . . . . .	47
5.4	BMP180 Sensorauswertung . . . . .	48
5.5	Bodenfeuchtigkeit Sensorauswertung . . . . .	48
5.6	Reed-Kontakt Sensorauswertung . . . . .	49
5.7	1-Wire Sensorauswertung . . . . .	49

# **Abkürzungsverzeichnis**

<b>CSI</b>	Camera Serial Interface
<b>DSI</b>	Display Serial Interface
<b>DSP</b>	Digitaler Signalprozessor
<b>GPIO</b>	General Purpose Input Output
<b>HATEOAS</b>	Hypermedia As The Engine Of Application State
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	HyperText Transfer Protocol Secure
<b>IDE</b>	Integrated Development Environment
<b>IoT</b>	Internet of Things
<b>I<sup>2</sup>C</b>	Inter-Integrated Circuit
<b>JSON</b>	JavaScript Object Notation
<b>LXDE</b>	Lightweight X11 Desktop Environment
<b>MMU</b>	Memory Management Unit
<b>M2M</b>	Machine-to-Machine
<b>NFC</b>	Near Field Communication
<b>PCB</b>	Printed Circuit Board
<b>PIXEL</b>	Pi Improved Xwindows Environment, Lightweight
<b>REST</b>	Representational State Transfer
<b>RFID</b>	radio-frequency identification
<b>RISC</b>	Reduced Instruction Set Computer
<b>ROA</b>	Resource-oriented Architecture
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure Sockets Layer
<b>SPI</b>	Serial Peripheral Interface Bus
<b>URI</b>	Uniform Resource Identifier
<b>USART</b>	Universal synchronous/asynchronous Receiver-Transmitter
<b>WDT</b>	Watchdog Timer

# **Kapitel 1**

## **Einleitung**

Dies ist die Studienarbeit der Studenten der Informationstechnik Jan Gerber und David Preiß. Die Studenten realisierten das Projekt im fünften und sechsten Theoriesemester an der DHBW in Karlsruhe. Sie wurden betreut von Herrn Professor Hans-Jörg Haubner.

### **1.1 Ziel dieser Arbeit**

Die vorliegende Arbeit zielt darauf ab, Sensorknoten für das Internet of Things zu realisieren. Die Sensorknoten sollen möglichst selbstkonfigurierend sein. Die Teilnahme an dem zu entwickelnden Mesh-Netz soll keine Vorkonfiguration erfordern.

Ferner ist das Ziel, dass die Sensorknoten energiesparend sind, sodass sie sich mittels Batterie betreiben lassen. Dadurch ergibt sich ein breites Anwendungsgebiet für die Sensorknoten.

### **1.2 Aufgabenstellung**

Für die eigentlichen Sensorknoten sollen Arduinos zum Einsatz kommen. Sie lesen die entsprechenden Sensoren aus. Die erfassten Daten senden sie über ein Funkmodul an einen Raspberry Pi. Dieser hat eine Internetanbindung. Der Raspberry Pi verarbeitet die Daten, bringt sie in einen standardisiertes Format und gibt sie an einen externen Server weiter. Der externe Server kann dann eine Auswertung der Daten vornehmen.

### **1.3 Stand der Technik**

Für die Umsetzung der gestellten Aufgabe eignen sich Arduino-Mikrocontroller und der Raspberry Pi besonders. Sie lösen die generell bestehenden technischen Probleme relativ gut. Zu diesen Problemen gehört unter anderem der Energieverbrauch und die Kommunikation. Für den Bau von Prototypen sind diese Mikrocontroller sehr beliebt.

Fertige Produkte auf dem Markt nutzen dagegen meist proprietäre eingebettete Systeme. Diese lassen sich allerdings schlecht oder gar nicht modifizieren.

## 1.4 Motivation und Vorausblick

Hardwareentwicklung, Softwareentwicklung sowie die Webentwicklung sind die relevanten Bereiche dieser Arbeit.

Nach der Einleitung betrachten die Autoren die gestellte Problemstellung. Im dritten Kapitel werden relevante technische und theoretische Grundlagen vermittelt.

Anschließend folgt die Konzeption. Hier sind die grundsätzlichen Überlegungen zur Realisierung des Projekts niedergeschrieben und graphisch veranschaulicht.

In dem Kapitel Implementierung sind wesentliche Schritte zur Realisierung beschrieben. Ebenso sind dort relevante Quellcodeausschnitte erklärt.

Das Fazit enthält einen Vergleich zwischen der gestellten Aufgabe und dem Ergebnis sowie einen Ausblick inwiefern dieses Projekt fortsetzbar ist.

# Kapitel 2

## Problemstellung

In diesem Kapitel soll die Problemstellung genauer erläutert werden. Hierfür wird der Ist-Zustand genauer betrachtet, sowie der Soll-Zustand der nach der Fertigstellung der Studienarbeit erreicht werden sollte.

### 2.1 Ist-Zustand

Für die Durchführung der Studienarbeit sind bereits zwei Hardwarekomponenten gegeben. Weshalb zunächst auf beide eingesetzten Systeme kurz eingegangen wird.

**Raspberry Pi** Bei einem Raspberry Pi handelt es sich um einen kleinen Einplatinencomputer. Dieser kann vor allem für kleinere Hard-/Software Projekte eingesetzt werden. Der Raspberry Pi überzeugt insbesondere durch seine Kompaktheit und seine Erweiterungsfähigkeit. An den GPIO<sup>1</sup> Schnittstellen können zusätzliche Module für Sensoren, LEDs und Funkmodule angeschlossen werden. Durch ein integriertes WLAN Modul und einem LAN Modul bietet der Raspberry Pi alle benötigten Komponenten um als Gateway für das Internet der Dinge zu dienen. Der Raspberry Pi kann mit vielen verschiedenen Programmiersprachen programmiert werden, wie zum Beispiel Java, C , C++ oder auch Python. In Kapitel 3.3 wird genauer auf den Aufbau des Raspberry Pi's eingegangen.

**Arduino** Bei einem Arduino handelt es sich um ein Board mit einem Mikrocontroller. Der Arduino basiert auf einem Atmel-AVR-Mikrocontroller der megaAVR-Serie. Dieser wird mit 5V betrieben und bietet mehrere digitale und analoge Input- / Output-Pins. Diese Pins können genutzt werden um verschiedene Sensoren, Aktoren und Funkmodule anzuschließen. Der Arduino kann mit Hilfe des eigens entwickeltem Arduino-IDE per USB Schnittstelle programmiert werden. Die Programmierung erfolgt in C bzw. C++. Der Arduino zeichnet sich vor allem durch seine Kompaktheit und seinem niedrigem Energieverbrauch aus. Für

---

<sup>1</sup>General Purpose Input Output

Projekte bezüglich des Internets der Dinge sind dies optimale Eigenschaften. In Kapitel 3.2 wird genauer auf die Eigenschaften und Aufbau des Arduinos eingegangen.

Für die Fertigstellung des Projekts stehen die beiden Theoriesemester des 5. und 6. Semesters des Studiums zum Bachelor of Engineering Informationstechnik bereit. Außer diesen beiden Systemen sind keine zu verwendete Hardware und Software gegeben mit denen die Studienarbeit bearbeitet werden muss. Diese Studienarbeit baut auf keiner Studienarbeit oder anderem Projekt auf.

## 2.2 Soll-Zustand

Das Ziel der Studienarbeit ist es ein Sensorknoten für das Internet der Dinge zu entwickeln. Hierfür sollen mehrere Arduinos Sensordaten sammeln und diese an einen Raspberry Pi senden. Der Raspberry Pi sendet die gesammelten Daten dann zu einem externen Server. Der externe Server speichert die aufkommenden Daten in einer Datenbank und stellt sie über eine Webschnittstelle bereit. Die Präsentation der Daten erfolgt über ein Webinterface, auf das der Endbenutzer Zugriff hat.

**Arduino** Der Arduino ist für das Sammeln der Daten zuständig. Er wertet hierfür mehrere Sensoren aus. Die Sensoren sind über verschiedene Bus-Systeme und analogen, sowie digitale Pins angeschlossen. Die Sensoren werten mehrere Eigenschaften aus ihrer Umwelt aus, wie zum Beispiel Temperatur, Luftfeuchtigkeit, Luftdruck, Bewegung und Lichtintensität. Die Arduinos nutzen ein Funkmodul um untereinander und mit dem Gateway (Raspberry Pi) zu kommunizieren. Um eine ständige Kommunikation mit dem Raspberry Pi sicherzustellen soll eine Art von Mesh-Netzwerk eingesetzt werden. Dieses erlaubt eine unterbrechungsfreie Kommunikation mit dem Raspberry Pi, falls ein Sensorknoten ausfallen würde. Um ein praktikablen Einsatz in verschiedenen Umgebungen zu gewährleisten sollten die Arduinos auch teilweise mit Batterien betrieben werden können. Die Arduinos sollten über mehrere Monate hinweg, mit nur einer Batterieladung, die Daten an den Raspberry senden können.

**Raspberry Pi** Der Raspberry stellt das Gateway zum Internet dar. Er empfängt alle Sensordaten die mit Hilfe der Arduinos gesammelt wurden. Hierfür nutzt der Raspberry Pi das gleiche Funkmodul, das auch bei den Arduinos verwendet wurde. Beim Starten des Raspberry Pi führt dieser automatisch die benötigten Programme aus um die Daten zu empfangen. Nachdem der Raspberry Pi die Daten enkodiert hat sendet er diese weiter an einen externen Server.

**Webservice** Der externe Server besitzt eine SQL<sup>2</sup> Datenbank in der die Sensordaten gespeichert werden. Die Datenbank enthält nur Einträge von zuvor eingetragenen Statio-

---

<sup>2</sup>Structured Query Language

nen. Zusätzlich zu den Sensordaten sind zusätzliche Tabellen mit Informationen zu den einzelnen Sensorknoten vorhanden. Mit Hilfe einer REST<sup>3</sup>-Schnittstelle können die Daten als Webservice abgerufen werden. Die Daten sollten zur einfachen Weiterverarbeitung im JSON<sup>4</sup> Datenformat kodiert werden. Die Schnittstelle sollte mehrere Methoden zur Verfügung stellen:

- Abruf aller verfügbaren Sensorknoten mit zusätzlichen Informationen, wie zum Beispiel Name und Aufstellungsart
- Abruf der neusten Sensordaten eines Sensorknotens
- Abruf aller Daten eines Sensors innerhalb eines bestimmten Zeitraumes

**Datenrepräsentation** Die Webseite dient zur Präsentation der gesammelten Daten. Der Nutzer soll sich mit Hilfe einer Benutzerkennung und eines Passworts authentifizieren können. Nach einer erfolgreichen Authentifizierung erhält der Nutzer eine Übersicht über alle Stationen die entweder jetzt in Betrieb sind oder gegebenenfalls nicht im Betrieb sind. Es sollte zusätzlich die Möglichkeit bestehen, alle aktuellsten Sensorwerte eines einzelnen Sensorknoten anzusehen. Mit Hilfe einer Grafik sollten die Sensordaten über einen Zeitraum anschaubar sein. Der Zeitraum sollte vom Nutzer festgelegt werden können.

---

<sup>3</sup>Representational State Transfer

<sup>4</sup>JavaScript Object Notation

# Kapitel 3

## Theoretische und technische Grundlagen

### 3.1 Internet of Things

IoT<sup>1</sup> steht für die Vernetzung von Gegenständen, die primär keine Rechner sind. IoT ist in der Literatur nicht scharf definiert. Der wissenschaftliche Mitarbeiter des MITs Kevin Ashton hat sich 1999 wie folgt zum Thema IoT geäußert:

If we had computers that knew everything there was to know about things – using data they gathered without any help from us – we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory. RFID and sensor technology enable computers to observe, identify and understand the world—without the limitations of human-entered data.

Der Hintergedanke bei den meisten Definitionen von *IoT* ist, Abläufe zu automatisieren. Damit die *Things*, die IoT-Geräte, nützlich sein können, müssen sie ihre Umwelt wahrnehmen können. Dazu kommen Technologien wie

- RFID<sup>2</sup>
- NFC<sup>3</sup>
- und Bluetooth

zum Einsatz. Die eigentliche Idee dazu, entwickelte sich nicht erst in den letzten Jahren. Die Umsetzung der Idee ist jedoch begünstigt durch Fortschritte bei der Entwicklung von

---

<sup>1</sup>Internet of Things

<sup>2</sup>radio-frequency identification

<sup>3</sup>Near Field Communication

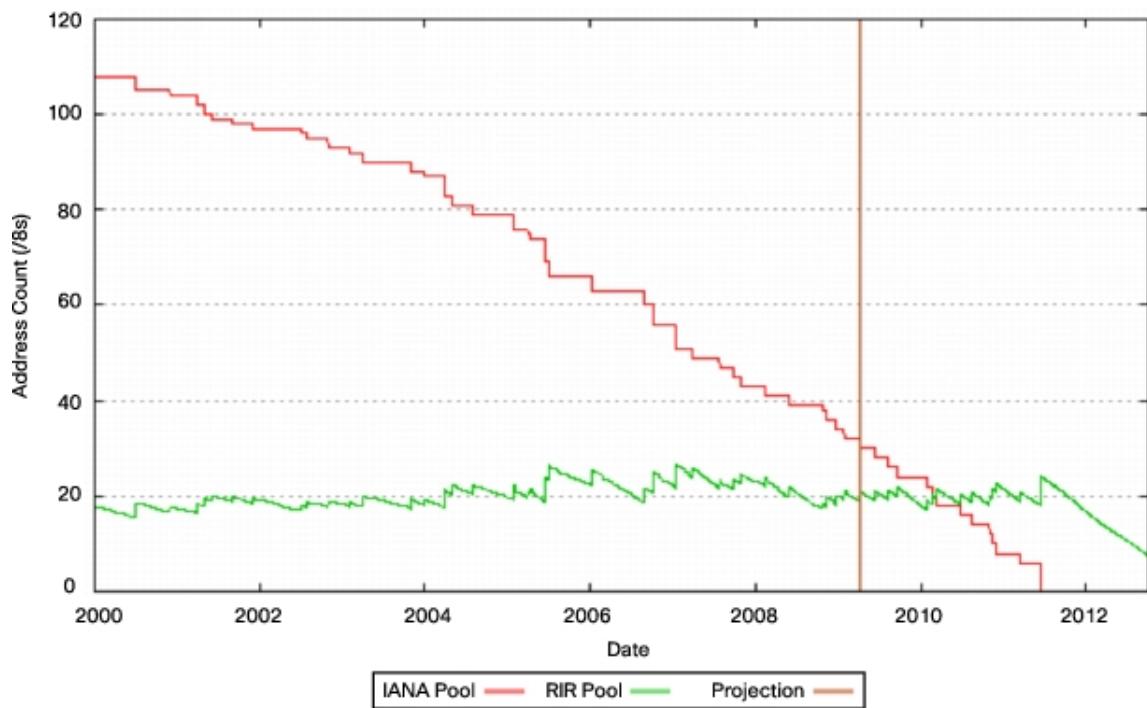


Abbildung 3.1: Verfügbarkeit von IPv4 Adressen bei den beiden IP-Vergabeorganisationen IANA und RIR [Cis16]

Batterien und Akkumulatoren, kompakten Photovoltaikelementen und mobilen Datennetzen (LTE, UMTS). Damit in Zukunft viele Geräte im Internet kommunizieren können, ist man auf lange Sicht wahrscheinlich dazu gezwungen, das Internetprotokoll Version sechs zu verwenden. Das Internetprotokoll in der Version vier unterstützt in der Theorie nur zirka 4,3 Milliarden ( $2^{32}$ ) Geräte. Durch die Nutzung von Verfahren wie Network Address Translation (NAT) geht man momentan noch Enpassen bei der Vergabe von Adressen aus dem Weg.

Um nun aber mehr als zirka 4,3 Milliarden Geräte direkt zu adressieren, muss man ein anderes Protokoll verwenden. Hier bietet sich das Internetprotokoll Version 6 an. Die Adressen sind nun nicht mehr nur 32 Bit lang, sondern 128 Bit. Das ermöglicht eine Adressierung von  $2^{128}$  Geräten. So könnte man ohne Engpässe alle IoT-Geräte direkt adressieren. Eine wichtige Rolle im Hinblick auf IoT ist die Maschine-zu-Maschine-Kommunikation. Für die Realisierung dieser ist es essentiell, dass jede Maschine (IoT-Gerät) erreichbar und eindeutig adressierbar ist. Die Nutzung des Internetprotokolls in der Version sechs kann dies sicherstellen. Die Maschine-zu-Maschine-Kommunikation wird in diesem Kapitel noch genauer betrachtet. Ferner hat IoT der Definition der Firma Lopez Research nach folgende Paradigmen:

- Kommunikation
- Kontrolle
- Kostensenkung

*Kommunikation* meint, was oben schon beschrieben ist. Die Kommunikation ist für IoT-Geräte essentiell. *Kontrolle* meint, dass IoT die Möglichkeit bieten soll, dass ein Benutzer jeder Zeit ein IoT-Gerät kontrollieren kann. Ein Beispiel könnte sein, dass man die Waschmaschine von unterwegs mit Hilfe des Smartphones aktivieren kann.

[Cis16]

## 3.2 Arduino

Der erste Arduino wurde 2005 von Hernando Barragan, Massimo Banzi und David Curatielles erfunden. Das Ziel war es einen Mikrocontroller-Board zu erfinden das sich einfach mit verschiedenen Dingen verbinden ließ und einfach zu Programmieren war. Da das Board von Studenten und Künstlern benutzt werden sollte stand auch der Preis des Boards im Vordergrund. Die Erfinder entschieden sich einen 8-bit Mikrocontroller der Atmel aus der AVR-Familie. Die Platine auf der der Mikrocontroller aufgebracht war erhielt einfache Ansteckmöglichkeiten, um etwa Sensoren, Relais und Motoren anschließen zu können. Die Erfinder schrieben zusätzlich einen Bootloader für den Mikrocontroller, der mit einfach mit der eigens entwickelten IDE<sup>4</sup> bespielt werden kann. Die Programmierung des Mikrocontrollers erfolgt mit C bzw. C++.

Über die Jahre wurden verschiedene Boards entwickelt die verschiedene Typen des Atmel Prozessoren nutzten. 2012 wurde sogar ein Arduino mit einem 32-Bit ARM Cortex-M3 Prozessor entwickelt. Die neueren Versionen der Arduino Boards haben bessere Prozessoren und mehr bzw. verbesserte Input / Output Pins Features. Die verschiedenen Boards unterscheiden sich hinsichtlich Größe, Form und zusätzliche Features wie zum Beispiel USB Anschluss und externer Stromversorgung. Bei der Hardware, sowie der Software von Arduinos handelt es sich um ein Open-Source Projekt. Die Community von Arduino ist in den letzten Jahren ständig gewachsen, was zu einer Fehlerbehandlung und die Suche nach neuen Ideen stark vereinfacht. Es gibt im Internet viele Do-It-Yourself Projekte, die die Nutzer mit detailreichen Anleitungen zum Ziel führen.

Trotz der vielen Möglichkeiten und der einfachen Programmierung mit Hilfe der Arduino IDE, dennoch gibt es einige kleine Einschränkungen die man bei Projekten beachten muss:

- Der Speicher ist einer der größten Einschränkungen, da dieser meist nicht erweiterbar ist. Die verwendeten Mikrocontroller haben eine feste Größe für den Programm- und Variablen Speicher. Der Arduino war nie als Ersatz für ein komplettes Computersystem mit viel Arbeitsspeicher und einem großen dauerhaften Speichersystem gedacht.
- Die Geschwindigkeit des Arduinos ist eine weitere Einschränkung. Die typische Geschwindigkeit der CPU liegt zwischen 8 und 20 MHz. Dennoch ist diese Geschwindigkeit meist ausreichend. Da der Arduino meist nur eine dedizierte Aufgabe hat

---

<sup>4</sup>Integrated Development Environment

und diese relativ langsam geschehen. Zum Beispiel müssen die Sensorwerte bei einer Hausüberwachung nur jede Minute ausgewertet werden.

- Eine weitere Einschränkung ist der elektrische Strom. Da die Input / Output Pins direkt mit dem Mikrocontroller verbunden sind, ist Vorsicht geboten, welche Spannungen an diese angeschlossen werden. Sollte die Spannung zu hoch sein kann der Mikrocontroller zerstört werden. Die verschiedenen Mikrocontroller die in Arduinos verbaut sind arbeiten meist entweder mit 3,3 V oder mit 5 V.

### 3.2.1 AVR Microcontroller

Der AVR Mikrocontroller wurde in den frühen 1990er als Studienprojekt und später von Atmel aufgekauft. Bei AVR Mikrocontroller handelt es sich um eine Harvard Architektur, bei der der Programmcode in einem Read-Only Speicher gespeichert wird und die modifizierbaren Variablen in einem separaten Speicherbaustein sind. Ein Mikrocontroller besteht dabei aus einer AVR CPU und zusätzlich Peripherikomponenten wie Timers, Counters, Serielle Interface Logik, Analog Digital Wandler, Analog Comperators und diskrete digitale Input-Output Ports.

**Architektur** Im Bild 3.2 ist der allgemeine Aufbau des AVR Mikrocontrollers zu sehen. Der Mikrocontroller besteht aus dem AVR Core, Speicherbausteinen (Flash Speicher und EEPROM) und verschiedenen peripheralen Funktionen wie Input/Output, Analog-Digital Wandler, Zähler, Timer und ein Serielles Interface. Dabei sind in der folgende Liste die Grundspezifikationen des Mikrocontrollers zusammengefasst:

- RISC Architektur
  - 131 Instruktionen
  - 32 8-bit General-Purpose-Register
  - Bis zu 20 MHz CPU Geschwindigkeit
- Speicher
  - Flash-Speicher (bis zu 256 Kilobytes)
  - EEPROM (bis zu 4 Kilobytes)
  - Interner SRAM (bis zu 32 Kilobytes)
- Betriebsspannung
  - Spannung zwischen 1,8 V und 5V

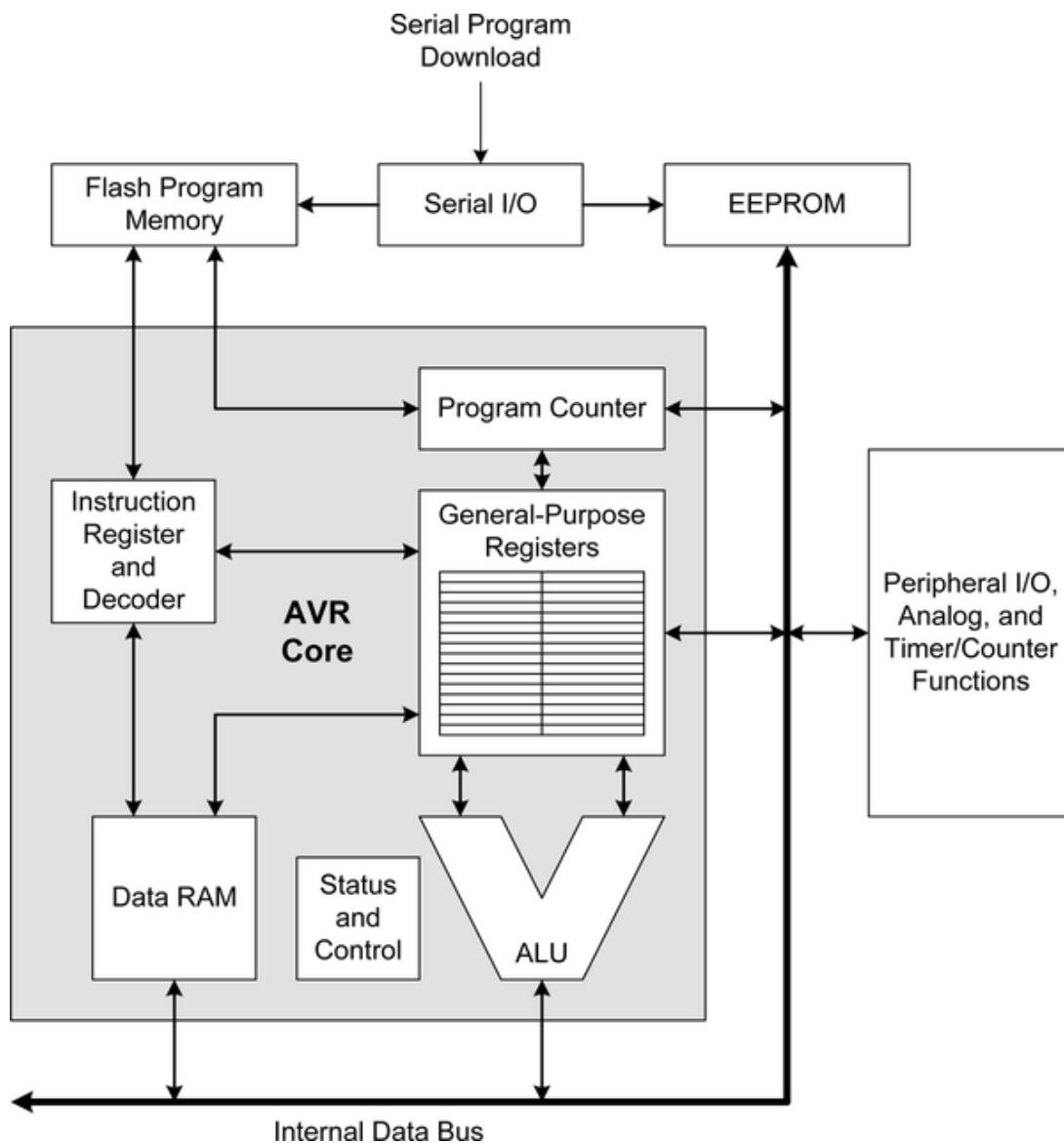


Abbildung 3.2: Aufbau eines AVR Mikrocontrollers. Quelle:[Hug16]

Der Mikrocontroller nutzt drei verschiedene Typen von Speicher, dem Flash Speicher, SRAM und dem EEPROM. Der Flash Speicher wird für die Speicherung des Programmcodes genutzt und ist nicht flüchtig. Beim SRAM handelt es sich um einen flüchtigen Speicher. Dieser speichert die Variablen des laufenden Programmes und enthält den Stack. Der EEPROM ist ein nicht flüchtiger Speicher, der Variablen dauerhaft speichern kann. Dieser speichert die Daten auch dann, wenn der Arduino keine Spannung anliegen hat. Der Nachteil eines EEPROMs ist jedoch die Zugriffsgeschwindigkeit im Gegensatz zum SRAM.

**Peripherie** Der folgende Abschnitt beschreibt die Peripherie des AVR-Mikrocontrollers, die an den internen Datenbus angeschlossen sind.

Innerhalb des Kontroll-Registers wird bestimmt wie die I/O Ports, der Timer, die Kommunikationsinterfaces und andere Funktionen funktionieren und reagieren. Das Kontroll-Register ist notwendig, da der AVR-Mikrocontroller deutlich mehr Funktionen hat wie Anschlüsse (Input/Output Pins). Zusätzlich können verschiedene Funktionen durch bestimmte Werte im Kontroll-Register verändert oder angepasst werden. Da das Kontroll-Register dynamisch konfigurierbar ist können Pins verschiedene Pins zu unterschiedlichen Zeiten und verschiedenen Funktionen haben. Ein Beispiel hierfür ist, dass an einem Pin sowohl der Analog Vergleicher angeschlossen ist und der Pin als Interrupt Quelle genutzt werden kann.

Der AVR-Mikrocontroller besitzt mehrere digitale Input/Output Ports um mit der externen Umgebung zu kommunizieren. Die Ports sind bidirektional, was bedeutet, dass sie sowohl als Input als auch als Output genutzt werden können. Jeder Pin besitzt eine interne Logik die festlegt ob es sich um einen Input oder Output handelt und ob ein Pull-Up Widerstand benutzt wird. Die interne Logik verfügt über viele Funktionen, die Einzel konfiguriert werden können.

In einem AVR-Mikrocontroller ist sowohl ein 8-Bit als auch ein 16-Bit Timer/Counter verbaut. Es kann zwischen zwei Modis unterschieden werden. Zu einem das Hochzählen mit Hilfe des internen Taktes und zum anderen mit Hilfe einer externen Quelle. Bei einem Überlaufen(Overflow) des Registers wird das Register wieder auf Null gesetzt, da das Register nur eine bestimmte Speicherbreite hat (8 bzw. 16 Bit). Bei einem Überlauf besteht die Möglichkeit einen Interrupt auszulösen.

Mit Hilfe des Prescalers können die Zählraten des Timers bzw. des Counters angepasst werden. Der Prescaler nutzt die interne Uhr der CPU um nach einem bestimmten Wert diese zu dividieren. Dies bedeutet, dass der Timer/Counter nur nach jedem X. Takt hochgezählt wird. Als Divisor zur Verlangsamung steht 8, 64, 256 oder 1024 zur Verfügung. So kann der Timer/Counter besser an die reale Umgebung angepasst werden, da wiederkehrende Events meist erst nach Sekunden und nicht nach Mikrosekunden passieren sollen.

Fast alle AVR-Mikrocontroller haben direkt einen Analog Komparator integriert. Der Komparator hat zwei Eingänge, einen normalen und einen invertierten. Der Analog Komparator vergleicht nun beide Spannungen miteinander und gibt aus welche der beiden Spannungen größer ist.

Mit Hilfe eines Analog zu Digital Wandler können sehr einfach analoge Größen in einen Digitalen Wert gewandelt werden. Ein Analog-Digital Wandler besitzt meist zwischen 4 und 28 Inputs (sogenannte Kanäle). Da der Mikrocontroller nur einen Analog-Digital Wandler ist oder hat, besitzt dieser zusätzlich einen Multiplexer, der die verschiedenen Kanäle auf den Analog-Digital Wandler schaltet. Für eine Umwandlung werden ca. 65 Mikrosekunden benötigt.

Der AVR-Mikrocontroller bietet drei verschiedene Formen eines seriellen Interfaces:

- USART<sup>5</sup> : Diese Schnittstelle erlaubt eine synchrone oder asynchrone serielle Datenübertragung. Die häufigsten Anwendungsgebiete sind die Kommunikation mit dem PC, Datenübertragung mit einer IR oder einem Bootloader.
- SPI<sup>6</sup> (siehe Kapitel 3.3)
- Two-Wire Interface: Diese Schnittstelle ist kompatibel mit dem  $I^2C$  Protokoll(siehe Kapitel 3.3 )

**Interrupts** Bei einem Interrupt handelt es sich um eine Unterbrechungsanforderung des normalen Programmablaufs. Ein Interrupt kann durch verschiedene Szenarien ausgelöst werden:

- Interrupt durch einen externen Pin
- Interrupt durch den Timer/Counter
- Interrupt durch die Fertigstellung einer Seriellen Übertragung
- Interrupt wenn der EEPROM mit dem Speichern bzw. Lesen von Daten fertig ist

Wenn ein Interrupt detektiert wurde wird das gerade ausgeführte Programm unterbrochen und die sogenannte Interrupt Service Routine aufgerufen. Wenn diese fertig ausgeführt wurden wird das Programm an der unterbrochenen Stelle wieder ausgeführt. Die Interrupts können im Kontroll-Register zu einem zum Global aktiviert werden und zum anderen können einzelne Interrupts an und ausgeschalten werden. Wenn ein Interrupt passiert wird in der Interrupt Vektor Tabelle nachgeschaut an welcher Stelle sich die Interrupt Service Routine befindet.

---

<sup>5</sup>Universal synchronous/asynchronous Receiver-Transmitter

<sup>6</sup>Serial Peripheral Interface Bus

**Watchdog Timer** Der AVR-Mikrocontroller besitzt einen Watchdog Timer (WDT<sup>7</sup>). Der WDT besitzt eine einstellbare Zeit zwischen 16 ms und 8s. Wenn es beim WDT zu einem Overflow kommt wird entweder der Programmcounter auf 0 zurückgesetzt (Reset-Modus), ein Interrupt generiert oder eine Kombination aus beidem. Da der Watchdog Timer einen separaten Oszillator besitzt, wird der WDT auch im Sleep Modus weiter hochgezählt. Dies kann genutzt werden um den Mikrocontroller nach einer bestimmten Zeit wieder zu aktivieren. Sollte der WDT im Reset Modus sein, so können damit Deadlocks verhindert werden, wenn zum Beispiel der WDT nicht mehr softwareseitig zurückgesetzt wird.

### 3.2.2 Arduino Nano / Pro Mini

In dieser Studienarbeit werden zwei unterschiedliche Arduino Boards eingesetzt, weshalb der Arduino Nano und Arduino Pro Mini in diesem Abschnitt genauer betrachtet wird.

In beiden Arduinos ist der ATmega328 verbaut. Dieser ist aus der Produktfamilie der AVR-Mikrocontroller. Im Folgenden werden die gemeinsamen Eigenschaften der Arduinos aufgelistet:

- Speicherkapazität
  - Programmspeicher/Flash Speicher: 32 Kilobytes
  - EEPROM 1 Kilobytes
  - SRAM 2 Kilobytes
- Taktfrequenz: 16 MHz
- Externe Interrupts: 2
- Analoge I/O Pins: 8
- Digitale I/O Pins: 14
- Betriebsspannung: 5 V
- Versorgungsspannung: 5V – 12V

Die beiden Boards unterscheiden sich vor allem in der Größe und zusätzlichen Ausstattung. Der Arduino Nano hat eine Größe von 18 x 45mm, wohingegen der Arduino Pro Mini nur eine Größe von 18 x 33mm besitzt.

Hinsichtlich der Ausstattung unterscheiden sich die zwei Modelle sehr deutlich. Der Arduino Nano besitzt einen Mini USB Anschluss. Dieser kann zu einem zur Stromversorgung genutzt werden, zum anderen kann der Arduino Nano über den USB Anschluss programmiert werden. Der Arduino Pro Mini hingegen hat keine USB Anschluss. Um den

---

<sup>7</sup>Watchdog Timer

Pro Mini zu programmieren wird ein USB-RS232-Interface-Chips benötigt. Mit diesem ist es möglich eine serielle Schnittstelle über USB verfügbar zu machen. Eines der bekanntesten Programmern ist von der schottischen Firma Future Technology Devices International (FTDI). Der Arduino Nano besitzt im Gegensatz zum Pro Mini bereits auf dem Board einen 5V zu 3,3 V Wandler. Zusammenfassend kann gesagt werden, dass der Arduino Pro Mini eine reduzierte Version des Arduino Nano's ist.

### 3.2.3 Software

Dieses Unterkapitel betrachtet die Softwareumgebung, sowie die genutzte Entwicklungsumgebung auf dem Arduino.

**C/C++** Das Arduino Board kann mit Hilfe von C oder C++ oder mit einer reduzierten Version von C, der sogenannten „Arduino Programming Language“, programmiert werden. Diese enthält die Grundoperatoren, Variablen und Funktionen. Zur Kompilation des Programmcodes wird der GNU avr-gcc Compiler verwendet.

C und C++ werden vor allem für eine effiziente und maschinennahe Programmierung eingesetzt. C++, sowie C bestehen nur aus sehr wenigen Schlüsselwörtern. Sie können jedoch mit Bibliotheken erweitert werden. Bei C++ handelt es sich um eine Erweiterung von dem C Standard von 1990. C++ bietet gegenüber von C mehr Datentypen, Ausnahmehandlung, das Überladen von Operationen und weitere Funktionen, wie zum Beispiel virtuelle Funktionen.

**Arduino IDE** Die ebenfalls von Arduino erhältliche IDE ermöglicht das einfache Schreiben von Code und das Uploaden auf die Arduino Boards. Die Arduino IDE ist Open-Source und für Windows Mac OS X und für Linux erhältlich. Während der Studienarbeit wurde die Version 1.8.2 eingesetzt. Die Entwicklungsumgebung bietet dabei viele Features.

- Kompilieren und Hochladen des Programmcodes auf den Arduino
- Einbinden von externen Bibliotheken
- Übersichtliches Bedienkonzept
- Serieller Monitor zur Ausgabe von Daten des Arduinos

Die Arduino IDE zeichnet sich vor allem durch seine Einfachheit aus. Dies ermöglicht einen schnellen Einstieg in die Programmierung des Arduinos(siehe Grafik 3.3).

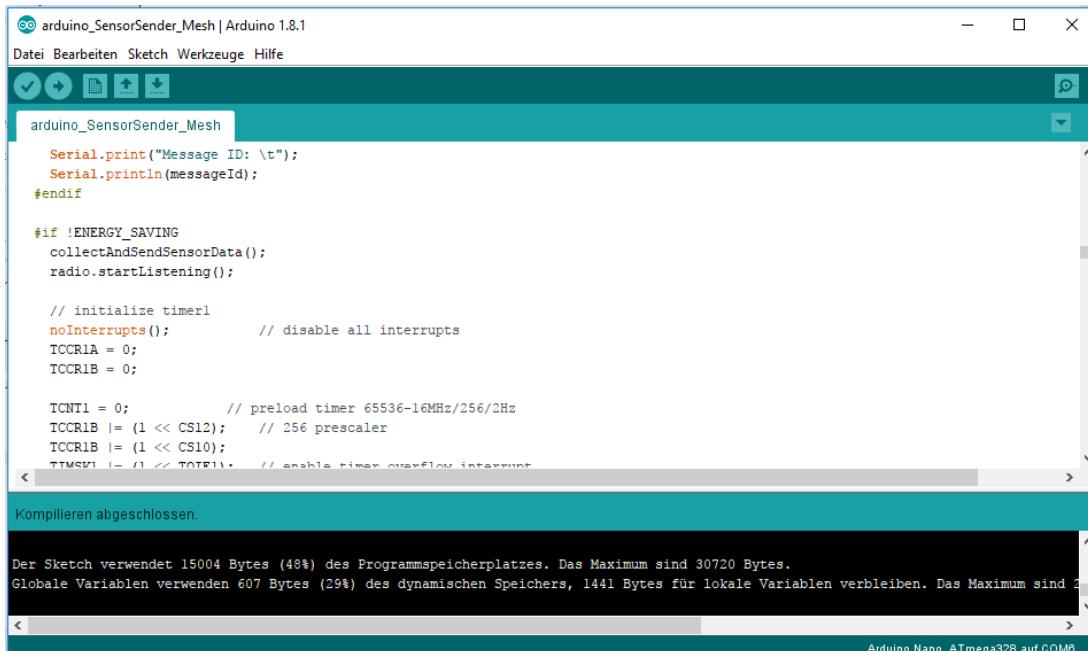


Abbildung 3.3: Arduino IDE 1.8.1

### 3.3 Raspberry Pi

Der Raspberry Pi stellt in diesem Projekt unter anderem die Schnittstelle zum Web oder Internet dar. Für den Raspberry Pi sprachen einige Argumente.

- Es kann das gleiche Funkmodul wie für die Arduinos verwendet werden
- Es ist eine große Community vorhanden

Ferner handelt es sich beim Raspberry Pi um einen Ein-Platinen-Rechner.

#### 3.3.1 Hardware

Bei dem Hauptprozessor des Raspberry Pi handelt es sich um einen Broadcom BCM2835. Die Architektur des Prozessors nennt sich *ARM*. ARM hat einige Vorteile gegenüber x86 Prozessoren:

- kleiner Energiebedarf
- geringe Wärmeentwicklung  
normalerweise keine Kühlung notwendig
- kleine Bauweise

Modus	Beschreibung	Ausnahme-Modus?
User	Ausführung normaler Programme	nein
System	privilegierter Modus	nein
Supervisor	Modus für Systemcalls	ja
Abort	Memory Abort	ja
Undefined	Udefinierte Anweisung (Fehler)	ja
Interrupt	Modus für Interrupts	ja
Fast Interrupt	Modus für schnellen Interrupt	ja

Tabelle 3.1: ARM-Modi

## Allgemein

Die ARM-Architektur besitzt folgende Eigenschaften:

- RISC<sup>8</sup>
  - 32-Bit Daten- und Adressbus
  - 7 Prozessor Modi
  - 37 Register
  - 5 Hauptadressierungsmodi
  - Datentypen
    - 32 Bit Wort
    - 16 Bit Halbwort
    - Byte (8 Bit)
  - Instruktionen
    - 54 ARM-Instruktionen, sie sind jeweils ein Wort
    - 38 Thumb-Instruktionen
- Datenverarbeitungen werden auf Wörter angewandt

## Modi

Ein Interrupt und ein Fast Interrupt unterscheiden sich in ihrer Priorität. Ein Fast Interrupt kann also einen Interrupt unterbrechen. Es darf jedoch nur ein Fast Interrupt zu einem Zeitpunkt auftreten. Der User-Modus ist der einzige Modus ohne Privilegien. Die anderen Modi haben mehr Rechte wie der User-Modus. In User-Modus werden standardmäßig Programme ausgeführt. Einige Register sind Modussensitiv, soll heißen, dass sie nur in einem bestimmten Modus nutzbar sind. Wenn der Rechner startet, befindet sich der

---

<sup>8</sup>Reduced Instruction Set Computer

Abbildung 3.4: Überblick über die Register [Wer09]



Abbildung 3.5: Das Statusregister im Detail [Wer09]

Prozessor im Supervisor-Modus. Später kann von jedem Modus aus gewechselt werden, außer aus dem User-Modus. Man kann ihn nur über einen Softwareinterrupt verlassen.

## Register

Im User-Modus kann man über die Register R0 bis R12 frei verfügen. Register R13 ist der Stackpointer, Register R14 das Linkregister. Das Linkregister merkt sich die Return-Adresse um nach einem Funktionsaufruf zurückzukehren. Das Register R15 ist wird als Programm-Counter genutzt.

Das Statusregister ist im Detail wie folgt aufgebaut:

Das hochwertigste Bit fungiert als Vorzeichen. Das 30. Bit ist bei Null gesetzt, das 29. Bit  $C$  ist gesetzt, wenn das Ergebnis einer Operation eine Übertrag hat. Bit  $V$  ist bei einem Overflow (Überlauf) gesetzt. Bit  $Q$  fungiert bei ARM-Prozessoren, die als DSP<sup>9</sup>

## <sup>9</sup>Digitaler Signalprozessor

Bit M <sub>4</sub> , ..., M <sub>0</sub>	Modus
10000	User
10001	Fast Interrupt
10010	Interrupt
10011	Supervisor
10111	Abort
11011	Undefined
11111	System

Tabelle 3.2: Modi im ARM Statusregister

eingesetzt werden, als Überlaufbit für erweiterte Instruktionen. Die Bits 26 bis 8 sind undefiniert, Bit *I* (7) ist das Interruptbit. Es ist bei einem Interrupt gesetzt. Bit *F* (6) ist das Fast Interrupt-Bit. Es ist dementsprechend gesetzt. Das Bit *T* (5) gibt den zeigt an, ob der Thumb-Mode aktiv ist. Im Thumb-Mode sind die Instruktionen nicht 32 Bit lang, sondern nur 16 Bit. Instruktionen im Thumb-Mode sind bietet demnach weniger Optionen. Sie arbeiten häufig implizit. Der Vorteil von Thumb-Instruktionen ist, dass die Instruktionen weniger Speicher benötigen. Die Abarbeitungsgeschwindigkeit von 32 Bit Instruktionen und Thumb-Instruktionen ist identisch. Die Bits 4 bis 0 kodieren den Modus der CPU: [Wer09]

## Exceptions

Der ARM-Prozessoren kennen typischer sieben verschiedene Exceptions (Ausnahmen). Sie haben folgende Priorität:

1. Reset
2. Data Abort
3. Fast Interrupt Request
4. Interrupt Request
5. Prefetch Abort
6. Undefined Instruction
6. Software Interrupt

Kommt es zur Exception *Reset*, bricht der Prozessor seine Arbeit an der momentanen Stelle ab und springt sofort zur Adresse, die der Reset-Exception zugeteilt ist. Diese Adresse der Speicherstelle an die der Prozessor nach dem Start springt, ist mit der Reset-Adresse identisch. Greift ein Programm unberechtigterweise auf einen Speicherbereich zu, schmeißt der schmeißt das System die *Data Abort*-Exception. Anschließend springt der Prozessor an eine definierte Speicherstelle. Hier kann der Programmierer die Exception

durch Implementieren einer Routine abfangen. Ein *Fast Interrupt Request* führt zur Abarbeitung der entsprechenden Methode wenn die Anfrage gültig ist, das heißt wenn der Prozessor gerade keine andere Exception und der Fast Interrupt Request eindeutig ist. Gleiches wie für den Fast Interrupt Request gilt auch für den Interrupt Request. Allerdings hat der Interrupt Request eine niedrigere Priorität. Der Prozessor kann ihn nur behandeln, wenn keine Exception höherer Priorität auftritt. Dies kann beispielsweise ein Fast Interrupt Request sein. Externe Hardware kann ein (Fast) Interrupt Request auslösen. Will der Prozessor beim Fetch auf eine Adresse zugreifen, auf die er kein Zugriff hat, so schmeißt er eine *Prefetch Abort*-Exception. Kennt der Prozessor die geforderte Instruktion nicht, so kommt es zu einer *Undefined Instruction*-Exception. Sie hat die gleiche Priorität wie die *Software Interrupt*-Exception. Dies hat den Hintergrund, dass es nicht möglich ist, dass sie gleichzeitig auftreten. In einem Programm kann man explizit eine *Software Interrupt* Exception auslösen. Dies kann nur geschehen, wenn der Prozessor die Instruktion interpretieren kann. Man kann die Software Interrupt-Exception beispielsweise nutzen um in den Supervisor-Modus zu wechseln. Generell muss der Assemblerprogrammierer die entsprechenden Exceptionroutinen implementieren.

[Wer09]

## **Memory Management Unit**

ARM-Prozessoren verfügen wie auch x86-Prozessoren über eine MMU<sup>10</sup>. Dieser Hardware-Baustein hat die Aufgabe virtuelle Speicheradressen in reale Speicheradressen zu übersetzen. Dies hat mehrere Vorteile:

- einfacherer Umgang mit Adressen
- Erhöhung der Sicherheit
- keine großen, ungenutzten Speicherstellen am Stück

Die Virtuelle Adresse (V-Address) besteht (vereinfacht) aus einer Adresse für das Page Directory, der Adresse für die eigentliche Page und dem Offset innerhalb der Page.  $P$  und  $W$  in der Abbildung sind Bits. Ist das  $P$ -Bit gesetzt, so ist die jeweilige Seite im Hauptspeicher verfügbar. Ist das  $W$ -Bit gesetzt, so darf auf die Page geschrieben werden. Das  $W$ -Bit ist für weitere Mechanismen notwendig. Diese sind jedoch nicht Gegenstand dieser Arbeit.

Dadurch, dass der Prozessor nicht mit realen Adressen arbeiten muss, muss der Programmierer oder Compiler die konkreten realen Adressen nicht kennen. Nun ist es möglich zu realisieren, dass es für Programme so aussieht, als wäre der zugeteilte Speicherbereich am Stück. Physikalisch ist der zugewiesene Speicher jedoch über den ganzen RAM verteilt.

---

<sup>10</sup>Memory Management Unit

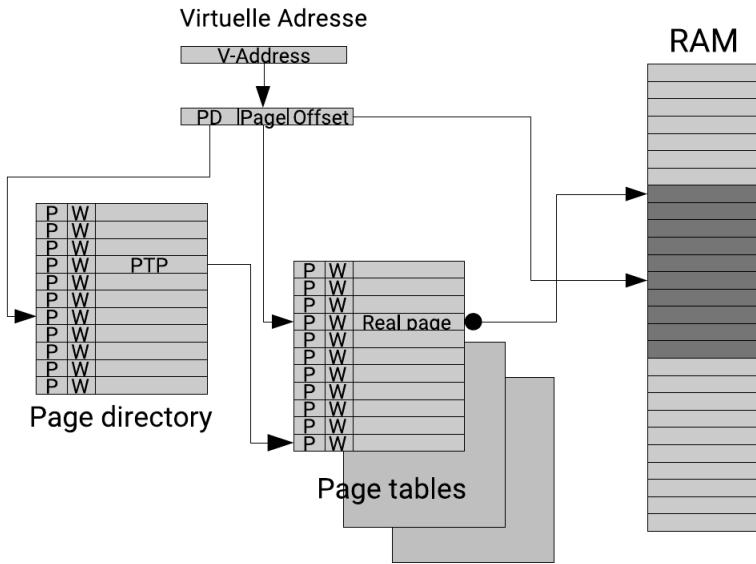


Abbildung 3.6: Virtuelle Adressierung [Ric]

Ferner ist es auch der Fall, dass ein Betriebssystem ständig Programme startet und beendet. Endet ein Programm, so muss man den allokierten Speicherbereich wieder deallocieren. Die virtuelle Adressierung verhindert in diesem Zusammenhang, dass groß Speicherbereiche "brach" liegen. Würde man kein virtuelle Adressierung verwenden, so ist nach dem Ende eines Programmes, dass 1 Gb Speicher benutzt hat, eine 1 Gb große Lücke im Speicher. Dieser Umstand kann zu Problemen führen, wenn ein neues Programm startet und viel Speicher benötigt. Im Falle der virtuellen Adressierung ist es irrelevant, ob der angeforderte Speicher am Stück ist oder ob die benutzen Speicherstellen über den RAM verteilt sind.

[Ric]

### Externe Anschlüsse

Der Raspberry Pi verfügt über mehrere physikalische Schnittstellen.

- GPIO
- USB
- 3,5 mm Klinke für Audio
- CSI<sup>11</sup>
- RJ45 für Ethernet
- HDMI

<sup>11</sup>Camera Serial Interface

- DSI<sup>12</sup>

Die GPIO-Pins ermöglichen unter anderem eine Kommunikation über I<sup>2</sup>C<sup>13</sup> und den SPI.

## I<sup>2</sup>C

Bei I<sup>2</sup>C handelt es sich um einen seriellen Bus mit zwei Leitungen. Es gibt die Leitung SCL (Signal Clock) und SDA (Signal Data). Da mit SCL eine Takteleitung vorhanden ist, redet man von einem synchronen Bus. Der Bus arbeitet nach dem Master/Slave-Prinzip. Er ist des Weiteren auch Multi-Master-fähig. Es können mehrere Teilnehmer die Rolle des Masters einnehmen. Prinzipiell können Master zwei Operationen durchführen:

- Lesen
- Schreiben

Um einen reibungslosen Ablauf zu gewährleisten, organisiert sich der Bus durch

- Startbedingung
- Stoppbedingung
- Acknowledge
- No-Acknowledge

Bevor ein Master mit einer Operation beginnen kann, muss er prüfen, ob dieser frei ist. Ist der Bus frei, so sind beide Leitungen auf HIGH (fünf Volt).



Abbildung 3.7: I<sup>2</sup>C im freien Zustand [Com08]

Nun kann der Master seine Transaktion mit der Startbedingung ankündigen. Dazu lässt er die Takteleitung SCL auf HIGH und legt die Datenleitung über einen Pull-down-Widerstand auf Masse (LOW). Diese Vorgehensweise ermöglicht die Multi-Master-Fähigkeit. Möchte jetzt ein weiterer Master eine Übertragung an diesem Bus starteten, so fällt ihm bei Prüfen des Busses auf, dass SDA LOW ist. An dieser Tatsache kann er auch nichts ändern. Der Strom fließt über den Pull-down-Widerstand des aktiven Masters auf Masse

<sup>12</sup>Display Serial Interface

<sup>13</sup>Inter-Integrated Circuit

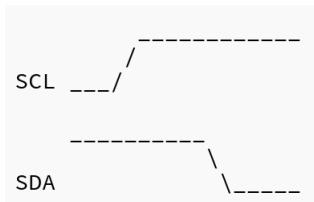


Abbildung 3.8: Startbedingung bei I<sup>2</sup>C [Com08]

ab. Nur der aktive Master kann SDA wieder auf HIGH legen und den Bus damit wieder frei geben.

Daten sind wie folgt kodiert: Für eine logische Null zieht der entsprechende Teilnehmer die Datenleitung (SDA) auf LOW. In der Zeit, in der die Takteleitung HIGH ist, gilt die logische Null.

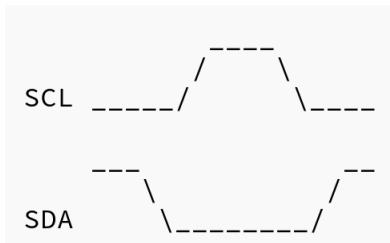


Abbildung 3.9: Logische 0 bei I<sup>2</sup>C [Com08]

Für eine logische Eins lässt der Busteilnehmer die Datenleitung auf HIGH. Auch die logische Eins gilt so lange, bis SCL HIGH ist.

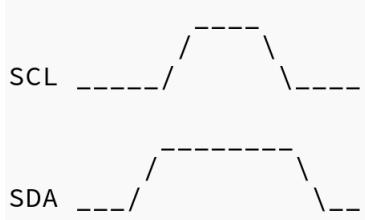


Abbildung 3.10: Logische 1 bei I<sup>2</sup>C [Com08]

Nur ein Master kann eine Operation starten mittels Startbedingung (sofern der Bus frei ist). Nachdem ein Master einen Bus erfolgreich reserviert hat, schreibt er ein Byte auf die Leitung. Die ersten 7 Bit des Bytes sind eine Adresse. Sie gehört dem Slave, mit dem der Master kommunizieren möchte. Bei I<sup>2</sup>C kann man somit 128 verschiedene Adressen vergeben. Das letzte Bit von dem Byte gibt die Richtung des Informationsflusses an. Der Master kann entweder lesen (1) oder schreiben (0). Diesem ersten Byte lauschen alle Geräte. Erkennt ein Slave seine Adresse auf dem Bus und er ist verfügbar, so sendet er ein Acknowledge.

Erkennt ein Gerät seine Adresse, ist aber nicht verfügbar, so sendet es ein No-Acknowledge.

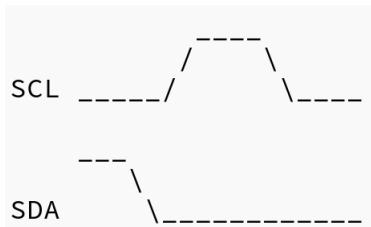


Abbildung 3.11: Acknowledgement bei I<sup>2</sup>C [Com08]

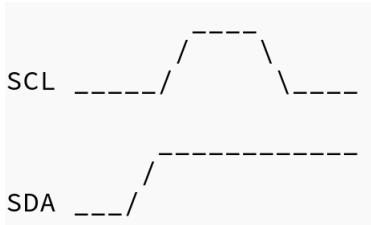


Abbildung 3.12: No-Acknowledgement bei I<sup>2</sup>C [Com08]

Um eine Übertragung zu beenden, zieht der Master die Datenleitung auf LOW, die Takteleitung bleibt währenddessen auf HIGH.

[Dem15] [Leh]

## SPI

Bei SPI kann man relativ hohe Datenübertragungsraten erreichen. Dazu darf allerdings die Wegstrecke der Leitungen nur relativ klein sein (auf einem Chip). Die Spezifikation von SPI ist weiter gefasst als die von I<sup>2</sup>C. Ferner handelt es sich bei SPI um kein Bussystem im klassischen Sinne. Jedes Gerät benötigt nämlich unter anderem eine eigene Leitung. Auch eine exakte Spezifikation der Geräte sieht SPI nicht vor.

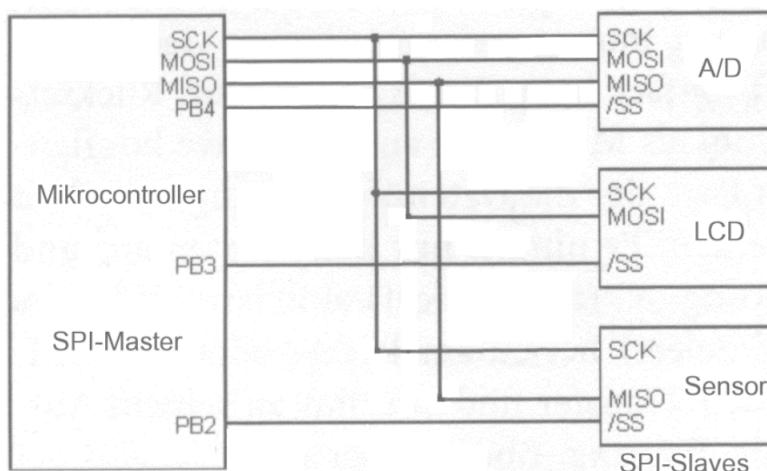


Abbildung 3.13: SPI-Schema [Dem15]

Es folgt eine Beschreibung, wie die SPI-Kommunikation beim Raspberry Pi aussehen kann: Der Master legt einen Takt von 100 kHz auf die Takteleitung SCK. Nun selektiert

er über eine Slave-select-Leitung (SS) einen Slave. Die SS-Leitung ist LOW-aktiv. Das heißt, dass die Leitung im unselektierten Zustand HIGH ist, im selektierten Zustand ist sie LOW.

[Dem15] [Leh]

### 3.3.2 Software

Dieses Unterkapitel betrachtet die Software, die auf dem Raspberry Pi nutzbar ist und im Rahmen dieser Arbeit genutzt wurde.

#### Raspbian

Bei Raspbian handelt es sich um ein Betriebssystem das für den Raspberry Pi entwickelt wurde. Es basiert auf Debian. Debian ist eine weit verbreitete Linux-Distribution. Sie erfreut sich großer Beliebtheit da viel Software (Packages) für sie entwickelt ist. Viele weitere Linux-Distributionen basieren ebenfalls auf Debian. Die Bedienung des Betriebssystems unterscheidet sich nur unwesentlich von der von Debian. Raspbian muss im Gegensatz zu den meisten Linux-Distributionen für ARM kompiliert sein. Raspbian wird zusammen mit PIXEL<sup>14</sup> zum download angeboten. PIXEL ist leitet von LXDE<sup>15</sup> ab. Bei LXDE handelt es sich um eine Desktop-Umgebung für Linuxsysteme. Die Entwickler von LXDE achten besonders auf Performance. Die Desktop-Umgebung soll möglichst wenig Ressourcen benötigen. So verzichten sie beispielsweise auf rechenintensiven Schattenwurf von graphischen Elementen. PIXEL ist wiederum auch für ARM kompiliert und speziell auf den Raspberry Pi angepasst. Das Betriebssystem Raspbian sowie die Desktopumgebungen LXDE und PIXEL sind Open Source.

#### Python

Python ist sowohl eine Programmiersprache als auch der Name des entsprechenden Interpreters. Der Interpreter ist in C implementiert und für viele Plattformen (Linux, Windows, MacOS) verfügbar. Python ist demzufolge eine interpretierte Programmiersprache. Der Interpreter ist ebenfalls Open Source. Des Weiteren weist die Sprache folgende Eigenschaften auf:

- Python ist nicht typisiert, man muss keine für Variablen und Objekte keine expliziten Datentypen angeben
- Variablen muss man nicht initialisieren

---

<sup>14</sup>Pi Improved Xwindows Environment, Lightweight

<sup>15</sup>Lightweight X11 Desktop Environment

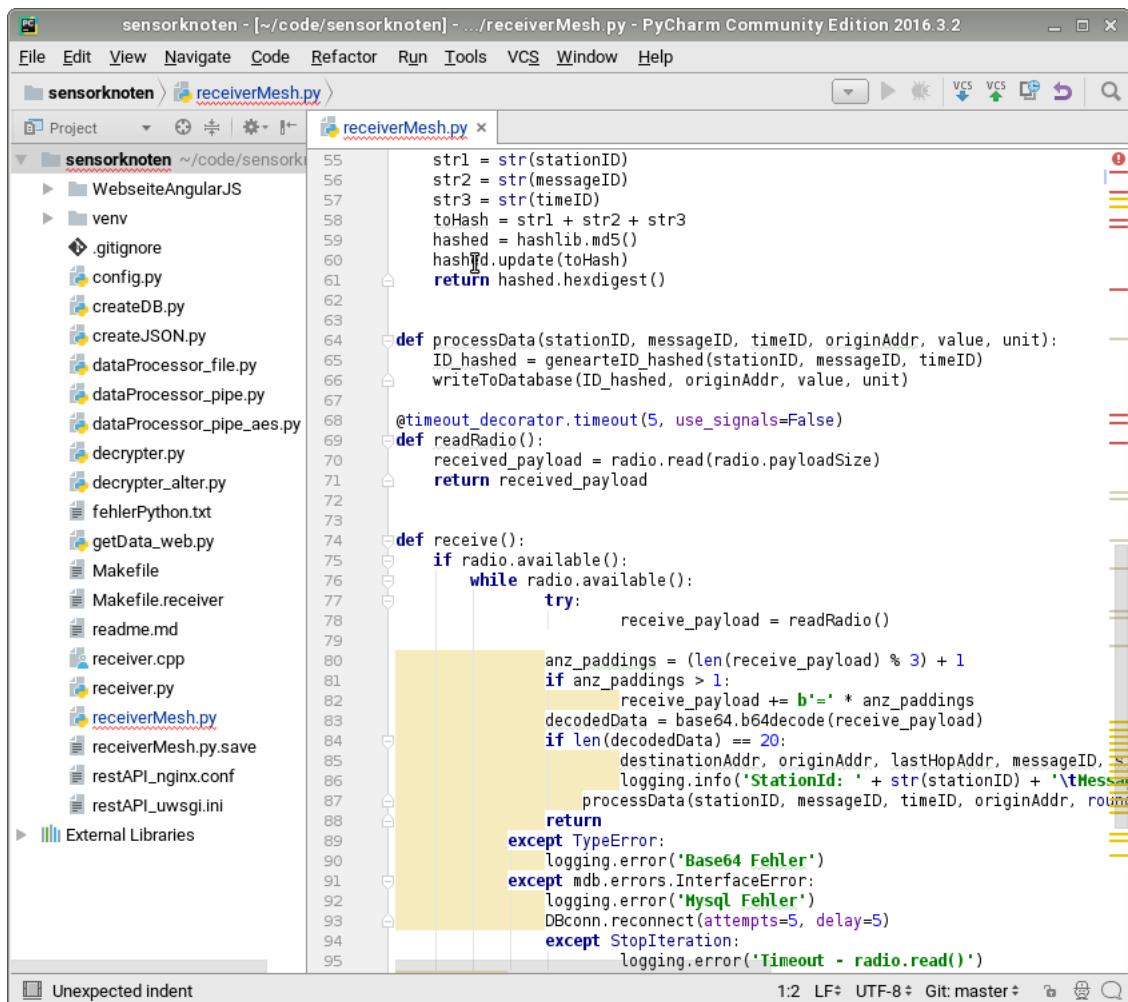


Abbildung 3.14: PyCharm IDE

- Der Code wird strukturiert und formatiert durch Einrückungen. Der Gebrauch von Klammern entfällt
  - Statements benötigen am Ende kein Semikolon
  - Python ist Objektorientiert
  - In Python kann C-Code aufgerufen werden

. Statements benötigen am Ende

## PyCharm

PyCharm ist eine Interierte Entwicklungsumgebung für Python. Sie wird von der Firma JetBrains herausgegeben. PyCharm ist in einer *Community Edition* und in einer *Professional Edition* verfügbar. Die Community Edition ist Open Source. Im Rahmen dieser Arbeit hat man die Community Edition genutzt. Die Entwicklungsumgebung bietet viele Features. Dazu gehören unter anderem:

- Sofortiges Parsen des eingegebenen Codes. Fehler wie Syntaxfehler markiert das Programm sofort
- Die Entwicklungsumgebung ermöglicht einfaches Refactoring des Codes wie zum Beispiel Extrahieren einer Methode in ein andere Quelltextdatei.
- Git: Das Programm bietet eine graphische Benutzeroberfläche für das Versionsverwaltungssystem *Git*

Die Entwicklungsumgebung ist für die meisten Betriebssysteme verfügbar (Linux, Windows, MacOS).

[Dem15]

## 3.4 Maschine-zu-Maschine Kommunikation

Bei der Maschine-zu-Maschine Kommunikation (M2M<sup>16</sup>) geht es darum, dass Maschinen automatisiert Daten miteinander austauschen können. Die Übertragungen von Daten werden dabei nicht durch Menschen ausgelöst, sondern direkt von den Maschinen. Für die Übertragung kann zum Beispiel Bluetooth, WLAN, Ethernet, Mobilfunk, ZigBee, RFID oder ein anderes Funknetzt genutzt werden. Meistens involviert ein solches Netz mehrere Geräte desselben Typs. Es werden oft auch sogenannte „M2M Area Network“ eingesetzt, damit die Geräte miteinander kommunizieren können. Diese Netzwerke haben dann meistens ein Art Router bzw. Gateway, das mit dem Internet verbunden ist.

Das Grundkonzept ist, dass ein Gerät, bei dem es sich meistens nur um einen Mikrocontroller handelt, Daten über ein Kommunikationsnetz an einen externen Server sendet, und dieser die Daten verarbeitet, speichert und gegebenenfalls Aktionen auslöst.

Bei M2M Kommunikation sind jedoch auch einige Hürden durch die verwendeten Geräte gegeben [BEH12, S. 4f]:

- **Limitierte Funktionalität** Die Geräte verfügen über wenig Rechenpower im Gegensatz zu modernen Computern. So kann ein Softwareupdate meist nicht drahtlos durchgeführt werden. Zusätzlich müssen die Geräte günstig produziert werden, was sich meistens durch an der Ausstattung bemerkbar macht.
- **Energiesparend** Viele Geräte die M2M nutzen sind nicht direkt an eine dauerhafte Stromversorgung angeschlossen, sondern werden mit Batterien bzw. Akkus betrieben. Dies schränkt die hohe Frequenz und Datenumfang von Nachrichten ein.
- **Eingebettet** Die Geräte sind oft direkt in ein anderes System eingebaut, was das wechseln erschwert. Die Geräte können nur mit hohen Aufwand und einem starken

---

<sup>16</sup>Machine-to-Machine

Eingriff in das System ausgetauscht werden. Beispielsweise dann, wenn die Geräte direkt in das System gelötet wurden.

- **Langlebigkeit** Die Systeme in denen die Geräte eingesetzt werden sind oft nicht mit der normalen Lebensdauer von IT-Systemen zu vergleichen. Beispielsweise werden diese in Systemen eingesetzt die mehrere Jahrzehnte betrieben werden.

Das Einsatzgebiet von M2M reicht von medizinischer Überwachung, Tracking von Fahrzeugen, Industrielle Anlagenüberwachung, Automatisierung von Gebäudetechnik bis zu dem Internet of Things.

**nRF24L01 Funkmodul** Beim nRF24L01 Funkmodul handelt es sich um ein 2,4 GHz Transceiver, das für den Einsatz in Geräten mit einem sehr niedrigen Energieverbrauch ausgelegt ist. Das Funkmodul kann sowohl als Empfänger als auch Sender fungieren. Das Funkmodul kann über ein SPI angesprochen werden. Mit Hilfe des Interfaces können auch die Register zur Konfiguration des Funkmoduls verändert werden. Das Funkmodul ist in zwei unterschiedlichen Versionen erhältlich(siehe Bild 3.15), einem Modul mit externer Antenne und einem ohne externe Antenne.

Zusätzlich zu den genannten Eigenschaften des Funkmoduls wird das Modul vom Hersteller Nordic Semiconductor mit folgenden Eigenschaften beworben [Sem07]:

- Energieverbrauch
  - 900 nA im Power Down Modus
  - 22  $\mu$ A Standby Modus
- Datenübertragungsrate: 250 Kbps – 2 Mbps
- Übertragungskanäle: 126
- Betriebsspannung 1,9 – 3,6 V
- Größe von Datenpakete: 1-32 Bytes
- Startzeit:
  - Power Down Modus: max. 1,5ms
  - Standby Modus: max 130  $\mu$ s

## 3.5 Sensorik

Da wie oben beschrieben die Sensorik eine große Rolle im Bereich der IoT spielt, sind in diesem Kapitel die Grundlagen der Sensorik aufgeführt.

### 3.5.1 Allgemein

Der Begriff *Sensorik* kommt aus dem Latein. *Sensus* bedeutet "der Sinn". Generell lässt sich sagen, dass ein Sensor ein System ist, das eine physikalische Größe und deren Änderung misst und die Messung in nutzbare Signale wandelt. Regelungen können Sensoren als Messerichtungen nutzen.

Sensoren kann man in folgende Kategorien einteilen:

- Mechanische Sensoren
- Temperatursensoren
- Chemische Sensoren
- Biosensoren
- Optische Sensoren
- Akustische Sensoren
- Magnetische Sensoren
- Gassensoren

Mechanische Sensoren können beispielsweise Kontaktsensoren sein. Sie können zum benutzt werden um zu erkennen, ob eine Tür geschlossen ist. Chemische Sensoren können chemische Stoffe quantifizieren. Der Unterschied zu Biosensoren ist, dass diese meist organische Verbindungen messen können (zum Beispiel Glucose).

Sensoren können unterschiedlich Komplex sein, je nach Anwendungsgebiet. Man unterscheidet zwischen:

- Elementarsensoren
- Integrierten Sensoren
- und Intelligenten Sensoren.

Elementare Sensoren sind man einfachsten aufgebaut. Im Falle eines optischen Sensors kann dieser aus lediglich einer Photozelle bestehen. Das Signal das er zurück gibt, muss unter Umständen ein A/D-Wandler noch digitalisieren. Ein Integrierter Sensor ist da schon komplexer. Er bereitet sein Signal auf. Das kann beispielsweise eine Verstärkung oder eine Filterung sein. Bei einem Intelligenten Sensor redet man schon von einem Rechner. Es kann sich zum Beispiel um eine Kamera oder um einen Laserscanner handeln. Derartige Sensoren arbeiten mit einer umfassenden Vorverarbeitung der Daten. Ferner unterscheidet man zwischen internen und externen Sensoren. Im weiteren betrachten wir die externen Sensoren. Sie sind in der IoT-Welt von hoher Bedeutung. [Str]



Abbildung 3.15: nRF24L01 Funkmodule

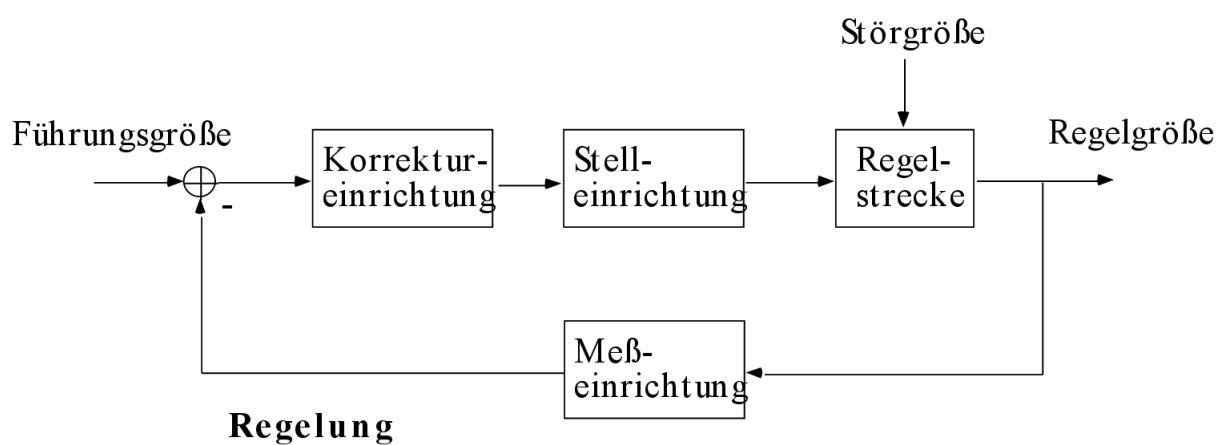


Abbildung 3.16: Schema einer Regelung [Str]

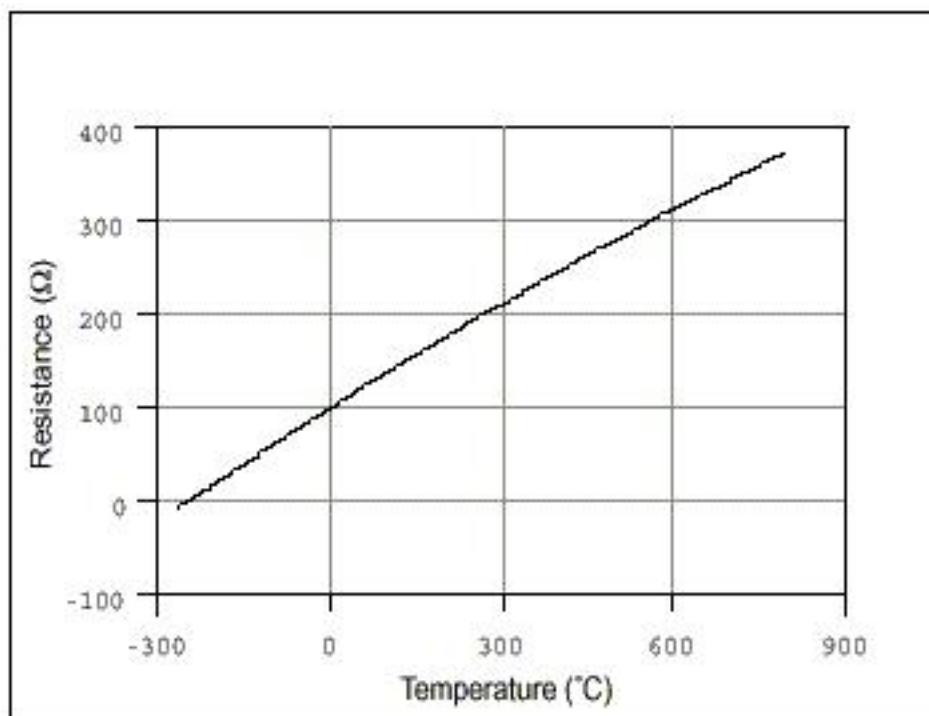


Abbildung 3.17: PT100 Temperatursensor [mik17]

### 3.5.2 Temperatur

Die Temperatur lässt sich auf viel Art und Weisen berechnen. Sensoren für Mikrocontroller kann man in zwei Kategorien einteilen:

- Analoge Sensoren
- Digitale Sensoren

Ein gängiger analoger Temperatursensor ist der *PT100*. Es handelt sich hierbei um einen Platinwiderstand. Er hat bei null Grad Celsius einen Widerstand von 100 Ohm. Der Widerstand nimmt ungefähr 0,4 Ohm pro einem Grad Temperaturänderung zu. Ein Vorteil dieses Sensors ist es, dass er einen großen Messbereich hat. Generell benötigt man für derartige Temperatursensoren eine relativ komplexe Schaltung. Außerdem kann man mit dem Sensor einen relativ großen Temperaturbereich abdecken.

[Mic17c]

### 3.5.3 Luftfeuchtigkeit

Luftfeuchtigkeitssensoren messen meist die relative Luftfeuchte. Dazu benutzen moderne Messeinrichtungen entweder;

- kapazitive oder
- widerstandsisierte Verfahren.

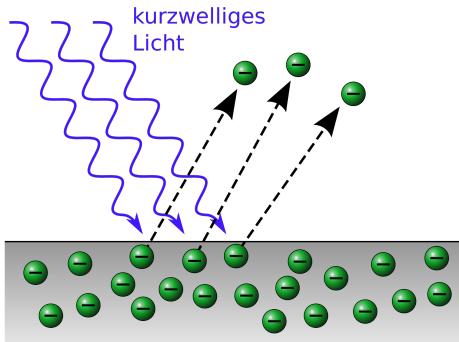


Abbildung 3.18: Photoelektrischer Effekt [Wik]

Bei der kapazitiven Messung ist ein Kondensator vorhanden. Die Dielektrizitätskonstante ändert sich in Abhängigkeit von der Luftfeuchte. Damit ändert sich auch die Kapazität des Kondensators. Über eine Messung der Kapazität kann man auf die Luftfeuchte schließen. Bei einem anderen Verfahren nutzt man Materialien deren Leitfähigkeit abhängig von der Luftfeuchte ist. Mit einer Widerstandsmessung kann man dann auf die Luftfeuchte schließen.

### 3.5.4 Beleuchtungsstärke

Zur Helligkeitsmessung kann man entweder Photowiderstände oder Photohalbleiter verwenden. Der ohmsche Widerstand eines Photowiderstandes ist abhängig vom einfallenden Licht. Über eine Widerstandsmessung kann man dann über auf die Helligkeit schließen. Photohalbleiter induzieren bei einfallendem Licht eine Spannung.

Einfallende Photonen regen Elektronen in der Photozelle an. Die Bewegung der Elektronen kann man als Spannung messen.

[Mic17b]

### 3.5.5 Bewegungsmelder

Bewegungsmelder können mit Infrarotstrahlung arbeiten. Sie messen die Infrarotstrahlung und geben zurück, sobald sich diese ändert. Die Sensoren nutzen dazu den pyroelektrischen Effekt. In Bewegungsmelder kommen zwei dieser Sensoren zum Einsatz.

Ein pyroelektrischer Sensor kann man nun direkt in einen Raum richten, beispielsweise mithilfe einer Linse. Verdeckt ein Objekt den einen Sensor, löschen sich nun die Spannungen der pyroelektrischen Bauteile nicht mehr aus. Der Bewegungsmelder gibt zurück, dass er eine Bewegung erkannt hat.

[Mic17a]

Dualsensor

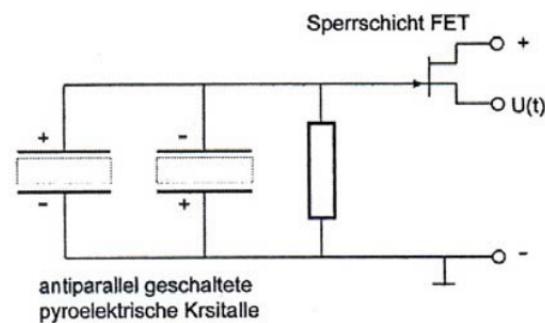


Abbildung 3.19: Schaltplan eines IR-Bewegungsmelders [Mic17a]

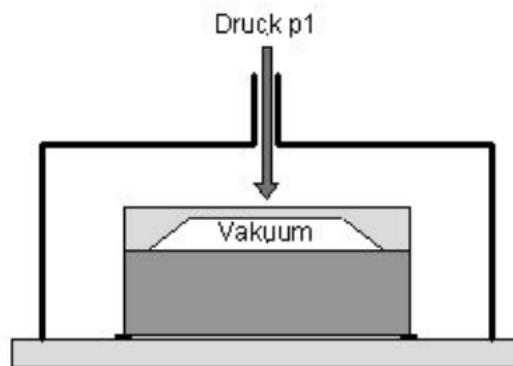


Abbildung 3.20: Drucksensor

### 3.5.6 Luftdruck

Den atmosphärischen Luftdruck kann man mit Absolutdrucksensoren messen. Die Messrichtungen haben ein Vakuum. Das Vakuum trennt man mit einer Membran von der Umgebung. Ändert sich nun der Umgebungsdruck, so bewegt sich die Membran. Die Bewegung nutzt man, um mit Piezokristallen eine Spannung zu messen. Die Spannung ist bei größeren Bewegungen und somit größeren Änderungen des Luftdrucks auch größer.

[Fir15]

### 3.5.7 Bodenfeuchte

Die Messung der Bodenfeuchte kann dem gleichen Prinzip wie die Messung der Luftfeuchte folgen (Kapitel 3.5.3).

## 3.6 Mesh-Netze

Bei einem Mesh-Netz (vermaschten Netz) handelt es sich um eine Netzwerktopologie. Das Netzwerk kann entweder aus drahtgebunden oder drahtlosen Netzwerkgeräten bestehen. In weiteren Verlauf werden nur drahtlose Netzwerkgeräte betrachtet. Bei einem Mesh-Netz

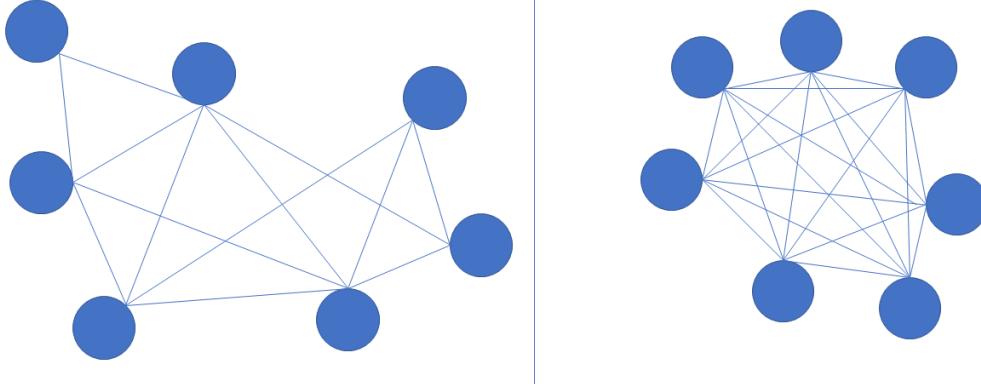


Abbildung 3.21: Mesh Netzwerk Topologie

können alle Netzwerkknoten genutzt werden um Daten von Punkt A zu Punkt B zu übertragen.

Von einem Mesh Netz wird gesprochen wenn jedes Netzwerkgerät mit mindestens einem oder mehreren Netzwerkgeräten verbunden ist (siehe linke Hälfte der Grafik 3.21). In der Literatur (vgl. [Con13]) wird in der Regel erst von einem Mesh Netz gesprochen, wenn das Netzwerkgerät über zwei unabhängige Pfade erreicht werden kann, sowie wenn jedes Netzwerkgerät mit jedem Netzwerkgerät verbunden ist. Erst dann wird von einem vollständig vermaschten Netz gesprochen (siehe rechte Hälfte der Grafik 3.21). Ein Beispiel für ein teilweise vermaschtes Netz ist das Internet. Bei einem Ausfall eines Knotens/Netzwerkgerät im Mesh Netz können die Daten der anderen Netzwerkgeräte trotzdem an ihr Ziel gelangen. Deshalb ist das Netz vor allem hinsichtlich seiner Ausfallwahrscheinlichkeit, im Gegensatz zu anderen Netzwerktopologien (zum Beispiel Stern-Topologie), deutlich erhöht. Ein Mesh Netz bietet die typischen Netzwerkaufgaben:

- Wegsuche (Routing)
- Überlastkontrolle (Congestion Control)
- Flußkontrolle (Flow Control)

Es kann innerhalb eines Mesh Netzwerk zwischen zwei Arten von Netzwerkgeräten unterschieden werden, dem Mesh Router und dem Mesh Clients. Der Mesh Router ist für das gesamte Routing zuständig. Da diese Mesh Router deutlich mehr Funktionen besitzen und deshalb eine höhere Anforderung an die Leistung der Geräte haben, sind diese meist an stationären Punkten aufgestellt. Mesh Clients können jegliche Arten von Geräten

sein. Diese Geräte benötigen deutlich weniger Rechenleistungen und müssen deshalb nicht stationär angebracht sein.

Das Routing innerhalb eines Mesh Netzwerk ist eine besondere Herausforderung und kann mit speziellen Mesh Protokolle gemeistert werden. Diese Protokolle lassen sich in drei Kategorien unterteilt werden. Das sind proaktiv, reaktive und hybride Routing Protokolle [VGA12]. Bei einem proaktiven Routing Protokoll werden alle Pfade zu allen Knoten, egal ob der Router Daten an diese Knoten senden muss oder nicht, bestimmt. Hierfür wird in periodischen Zeitabständen überprüft ob der Pfad zu den einzelnen Knoten noch verfügbar ist und gegebenenfalls ein neuer Pfad bestimmt wird. Der große Vorteil bei einem proaktiven Vorgehen ist, dass Informationen zum Pfad sehr schnell bereitstehen und diese zur Datenkommunikation verwendet werden können. Bei reaktiven Routing hingegen werden die Pfade nur bestimmt wenn ein Knoten Daten an einen anderen senden möchte. Das Finden des Pfades endet wenn ein Pfad gefunden wurde oder nachdem alle möglichen Kombinationen probiert wurden. Reaktive Protokolle skalieren besser in großen Mesh Netzen. Bei hybriden Routing Protokolle werden beide Ansätze kombiniert.

Trotz einem komplizierten Routing bieten ein drahtloses Mesh Netz dennoch einige Vorteile gegenüber einem normalen drahtlosen Netz [VGA12]:

- einfache Breitstellung
- hohe Zuverlässigkeit
- Selbstkonfiguration
- Selbstheilung bei einem Ausfall von Knoten
- Skalierbarkeit

## 3.7 Webservice mit REST

REST ( Representation State Transfer) wird vor allem wegen seiner Wiederverwendbarkeit, Skalierbarkeit, Erweiterbarkeit und eine einfache Integration in Web Services geschätzt. Das zentrale Konzept von REST sind Ressourcen, weshalb in diesem Umfeld auch von einer Ressource Oriented Architecture (ROA<sup>17</sup>) gesprochen wird.

REST wurde erstmals im Jahr 2000 in der Dissertation Architectural Styles and the Design of Network-based Soft-ware Architectures von Roy Fielding vorgestellt. Er hatte das im Jahr 1994 entworfene HTTP<sup>18</sup> Objekt Model weiterentwickelt. Nach [Til11] lässt sich REST auf 5 Kernprinzipien zusammenfassen:

- Eindeutig identifizierbare Ressourcen

---

<sup>17</sup>Resource-oriented Architecture

<sup>18</sup>Hypertext Transfer Protocol

- Verknüpfung von Ressourcen mithilfe von Links/Hypermedia
- Standardmethoden
- Repräsentation der Ressourcen in verschiedenen Formaten
- Zustandslose Kommunikation

Nach dem REST Prinzip sind Ressourcen eindeutig über Uniform Resource Identifier (URI<sup>19</sup>) erreichbar. So können Ressourcen ohne Hintergrundinformationen aufgerufen werden. Ein weiterer Begriff in diesem Umfeld ist Hypermedia, das für Inhalte steht die miteinander verknüpft sind und in dessen sich ein Programm/ Nutzer frei bewegen kann. Die Standartmethoden geben an was mit der Ressource gemacht werden soll. Jede Ressource sollte folgende Methoden GET, POST, PUT, DELETE, HEAD und OPTIONS implementiert haben. Bei GET handelt es sich um eine sichere Methode, da nur eine Ressource beim Server angefordert wird und so kein Nebeneffekt entstehen kann. PUT verändert bereits bestehende Daten, wobei DELETE vorhandene Ressourcen löscht. Bei PUT und DELETE sowie GET handelt es sich um indempotente Methoden, da bei einem erneuten Aufruf der Methoden keine Nebeneffekte auftreten. Nur bei der POST-Methode gibt es bei erneutem Aufrufen der Methode keine Garantie, dass Nebeneffekte auftreten.

Die zustandslose Kommunikation zwischen Client und Server ist ein weiterer wichtiger Punkt von REST. Es werden bei einem Aufruf einer Ressource keinerlei Informationen zwischen dem Aufruf mehreren Ressourcen zwischengespeichert. Der Vorteil von zustandsloser Kommunikation ist, dass ein Webservice leicht Skalierbar ist und so etwa bei großen Services sehr einfach Anfragen an mehrere Server verteilt werden können. Dieses Prinzip wird Lastenverteilung genannt.

Ein weiterer Performancevorteil bietet das Caching. Da jeder GET eine eindeutige Adresse hat kann festgestellt werden, ob eine Ressource bereits angefordert wurde und wenn ja, die Ressource aus dem Cache verwenden. Durch dieses Prinzip können Anfragen beim Server reduziert werden.

Zum Thema der Sicherheit sagt der Erfinder von REST Roy T. Fielding: „RESTful systems perform secure operations in the same way as any messaging protocol: either by encapsulating the message stream (SSL, TLS, SSH, IPsec) or by encrypting the messages (PGP, S/MIME, etc.). There are numerous Examples of that in practice, and more in the future once browsers learn how to implement other authentication mechanisms.“ [Fie08]. Es lässt sich jede Sicherheitstechnologie in ein REST Service einbauen, wie zum Beispiel SSL<sup>20</sup>,

---

<sup>19</sup>Uniform Resource Identifier

<sup>20</sup>Secure Sockets Layer

HTTPS<sup>21</sup> oder auch OAuth. Dies ermöglicht einen Einsatz von REST-Anwendungen im Unternehmensumfeld.

Bei Hypermedia as the Engine of Application State(HATEOAS<sup>22</sup>) handelt es sich um ein Entwurfsmuster. Das Ziel ist es, einer Ressource einen Link mitzugeben der zu einer Folgeoperation führt. Durch diese lose Kopplung kann die Schnittstelle angepasst werden ohne am Client festgelegte Einstellungen zu ändern.

Zusammenfassend lässt sich über REST sagen, dass es sich um ein Ressourcenorientiertes Design handelt das durch die Zustandslosigkeit einer Operation viele Vorteile im Bereich Load-Balancing und Chaching bietet. Durch die einfache Implementation der REST-Schnittstelle sind Services leicht zu warten, erweiterbar und skalierbar.

---

<sup>21</sup>HyperText Transfer Protocol Secure

<sup>22</sup>Hypermedia As The Engine Of Application State

# Kapitel 4

## Konzeption

### 4.1 Mesh-Netz

Für die Entwicklung eines Mesh-Netzes muss zunächst eine Topologie designt werden, die den Eigenschaften der Sensorknoten entsprechen. In das Netzwerk muss zu einem ein Sensorknoten eingebunden werden der extrem energiesparend ist, ein normaler Sensor-knoten der über eine dauerhafte Stromversorgung verfügt und ein Gateway zum Internet. Jeder dieser Geräte hat unterschiedliche Anforderungen und teilweise auch verschiedene Betriebssysteme.

In der Konzeptionsphase wurden die Geräte in drei Gruppen unterteilt. Dies waren die normalen Sensorknoten, energiesparende Sensorknoten und ein Gateway zum Datenbankserver/Internet. In der Grafik 4.1 ist die Topologie des Mesh-Netzes zu sehen.

**Normaler Sensorknoten** Dieser Sensorknoten übernimmt neben der Messwerterfassung noch das Weiterleiten von Nachrichten im Netztwerk. Hierfür ist er mit einer dauerhaften

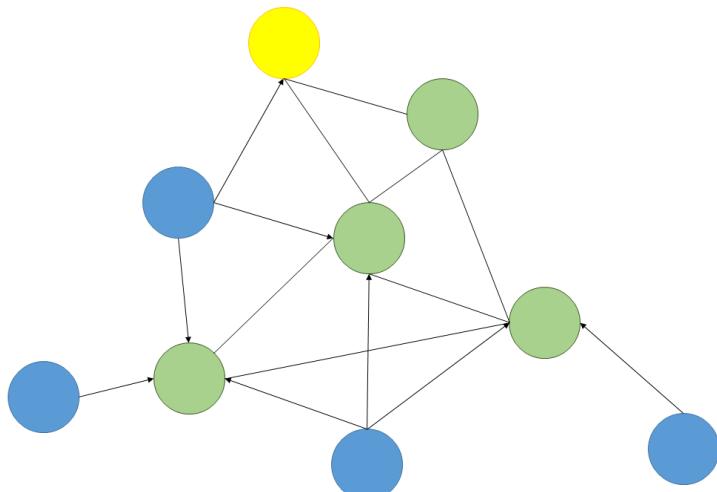


Abbildung 4.1: Topologie Mesh-Netz: blaue Knoten → energiesparende Sensorknoten, grüne Knoten → normale Sensorknoten und gelber Knoten → Gateway

Stromversorgung ausgestattet. Während der Sendephase des Sensorknotens kann dieser keine Nachrichten empfangen und weiterleiten. Wenn der Sensorknoten nicht in der Sendephase ist hört er die gesamte Zeit in das Netzwerk und wartet bis er eine Nachricht empfängt. Der genaue Ablauf wird im darauffolgenden Paragraphen genauer erläutert. Wie in den Grundlagen (3.6 Kapitel) erläutert ist, handelt es sich um einen aktiven Knoten, genauer um einen Mesh-Router. Mit diesen Geräten kann das Netz, bezüglich Reichweite, erweitert werden.

**Energiesparender Sensorknoten** Beim energiesparenden Sensorknoten handelt es sich nur um einen Mesh-Client (siehe 3.6 Kapitel). Das bedeutet, dass er ein passiver Knoten im Mesh-Netz ist. Dieser übernimmt keine Weiterleitungsaufgaben im Netzwerk. Nachdem der Sensorknoten seine Messwerte erfasst und gesendet hat geht dieser Knoten in einen Schlafmodus. Mit diesem Sensorknoten ist es nicht möglich die Reichweite des Netzwerks zu erhöhen. Dieser Knoten ist darauf angewiesen, dass entweder ein normaler Sensorknoten in der Nähe ist oder direkt das Gateway.

**Gateway zum Internet** Beim der letzten Komponente des Mesh Netzes handelt es sich um das Gateway zum Internet. Dieses Modul übernimmt nur das Empfangen von Nachrichten. Das Gateway ist im permanenten Empfangsmodus und wartet auf das Eingehen neuer Nachrichten. Nachdem das Datenpaket enkodiert ist wird sichergestellt, dass diese Nachricht zum ersten Mal angekommen ist. Wenn die Nachricht zum ersten Mal erhalten wurde, wird der Messdatensatz dauerhaft gespeichert.

**Programmablaufplan Mesh-Algorithmus** Nachdem wir die einzelnen Geräte genauer beleuchtet haben wird in diesem Abschnitt auf den selbst entwickelte Art von Mesh-Algorithmus eingegangen. In der Grafik 4.2 ist der Programmablaufplan bei einem normalen Sensorknoten zu sehen. Dieser beschreibt das Vorgehen, beginnend vom Empfang einer Nachricht bis zum Verwurf der Nachricht bzw. Weiterleiten der Nachricht.

## 4.2 Anbindung Pi/Arduino

Das RF24-Modul (Kapitel 3.4) soll der Kommunikation zwischen Arduino und Raspberry Pi dienen.

**RF24** Ein Grund der über die bisher genannten Gründe hinaus geht ist, dass das RF24 Modul mit dem Raspberry Pi als auch mit dem Arduino kompatibel ist. Es existiert eine Bibliothek die man für den Raspberry Pi und auch für den Arduino nutzen kann: <https://github.com/nRF24/RF24>. Diese Bibliothek beinhaltet auch Beispiele, die die Benutzung der Bibliotheksfunctionen veranschaulichen:

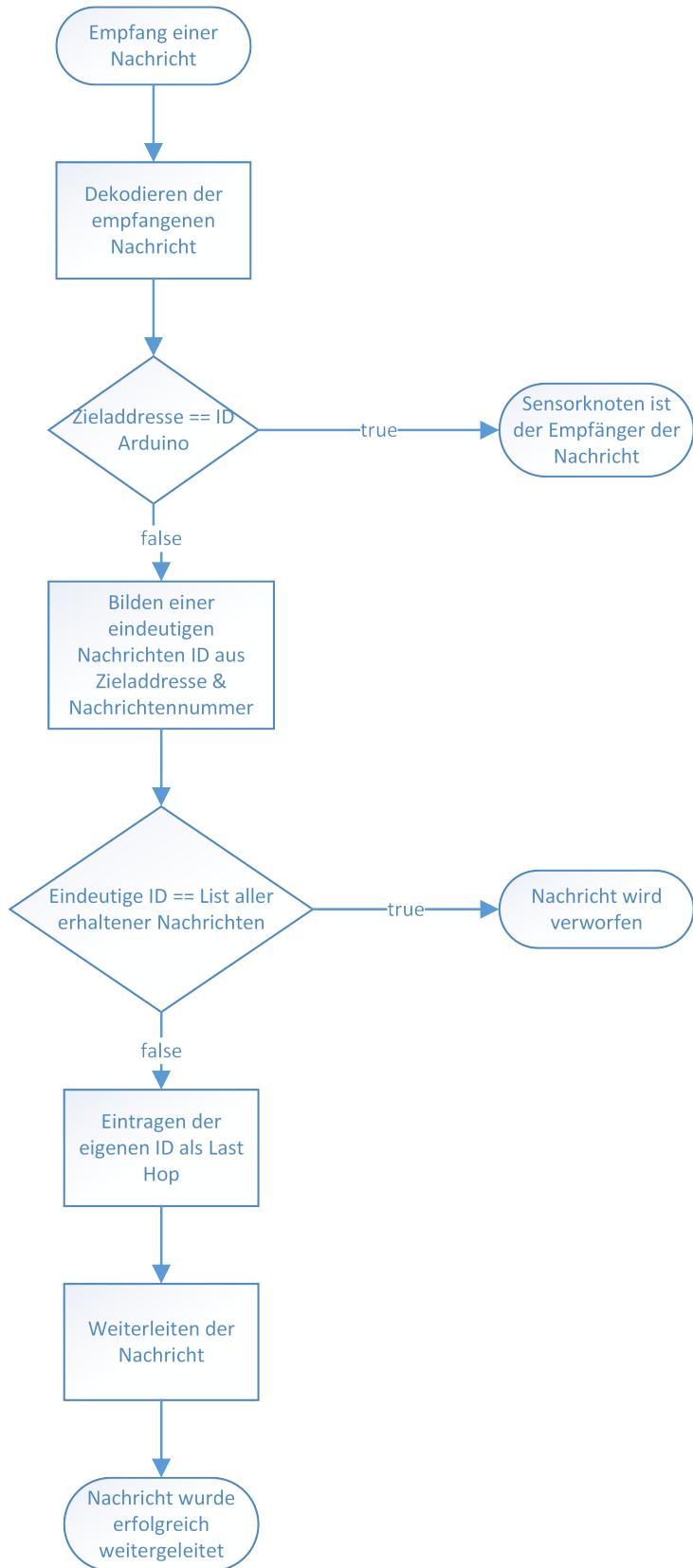


Abbildung 4.2: Programmablaufplan - Mesh- Algorithmus

---

```
if ( radio.available() ) {
    // Dump the payloads until we've gotten everything
    unsigned long got_time;

    // Fetch the payload, and see if this was the last one.
    while(radio.available()){
        radio.read( &got_time, sizeof(unsigned long) );
    }
    radio.stopListening();

    radio.write( &got_time, sizeof(unsigned long) );

    // Now, resume listening so we catch the next packets.
    radio.startListening();
}
```

---

Da auf beiden Plattformen die gleiche Bibliothek nutzbar ist, ist der Code kann der Code für den Raspberry ähnlich dem für den Arduino sein.

**Kodierung** Um sicherzustellen, dass der Raspberry Pi die Nachrichten des Arduinos lesen kann, gibt es unter anderem folgende Möglichkeiten:

1. Die Schnittstelle auf dem Raspberry Pi in C implementieren. Der Arduino sendet ein C-Struct. Beim Einlesen des Structs auf dem Raspberry muss man die Größe der Datentypen beachten (Ein Integer auf dem Arduino entspricht einem Short Integer auf dem Raspberry Pi, sie sind jeweils zwei Bytes groß). Der Vorteil ist, das man keine Rechenzeit zum kodieren auf dem Arduino benötigt.
2. Der Arduino sendet keinen Struct, sondern einen String. Alle Daten die der Arduino sendet, konkateniert man zu einem String. Der Compiler für den Arduinio encodiert Strings mithilfe von ASCII. Der Raspberry Pi muss dann die empfangenen Daten mit ASCII dekodieren. Auch hier benötigt man keine Rechenzeit zum kodieren, jedoch zur Konkatenation.
3. Explizit kodieren auf dem Arduino und dementsprechend auf dem Raspberry Pi dekodieren. Die benutzte Kodierung ist theoretisch nicht relevant, sie muss lediglich auf beiden Systemen identisch implementiert sein. Dies kostet Rechenzeit, die Kodierung ist jedoch frei wählbar.

Die dritte Möglichkeit sähe in Pseudocode auf dem Arduino wie folgt aus (ASCII ist das Kodierungsverfahren):

---

```
encode_ASCII('PAYLOAD');
```

---

Auf dem Raspberry:

---

```
decode_ASCII('PAYLOAD');
```

---

## 4.3 Messwerterfassung mit Arduino

In diesem Unterkapitel werden die verschiedenen Messgrößen und die allgemeine Messwerterfassung dargestellt, sowie das Programm das zur Planung und Gestaltung der Platinen genutzt wurde.

**Messgrößen** Der Sensorknoten soll nach seiner Fertigstellung verschiedene Messgrößen ermitteln können. Aus diesem Grund muss zunächst festgelegt werden, welche Messgrößen bestimmt werden sollen. Für diese Messgrößen müssen dann für die Umsetzung passende Sensoren gefunden werden. Folgende Messgrößen sollte der Sensorknoten bestimmen können:

- Temperatur
  - Lufttemperatur
  - Bodentemperatur
- Luftfeuchtigkeit
- Luftdruck
- Beleuchtungsstärke
- Bodenfeuchtigkeit
- Erkennung von Bewegung um den Sensorknoten
- Erkennung ob ein Fenster/Tür geschlossen oder geöffnet ist

**Ablauf der Messwerterfassung** Die Messwerterfassung soll in zyklischen Abständen erfolgen. Um das Mesh-Netz nicht mit Nachrichten zu überlasten werden nur jede Minute alle Messwerte gemessen und in das Netzwerk gesendet. Jeder Messwert hat eine eindeutige ID. Mit Hilfe dieser ID wird angeben welche Einheit der Messwert hat.

Die verschiedenen Messwerte werden mit mehreren Sensoren ermittelt. In jedem Zyklus werden die Sensorwerte in der gleichen Reihenfolge bestimmt. Zusätzlich zur Ermittlung der Messwert muss überprüft werden ob der Messwert überhaupt bestimmt werden kann. Beispiele dafür, dass der Messwert nicht bestimmt werden kann ist, dass der Sensor nicht angeschlossen ist oder der Sensor nicht mehr Ordnungsgemäß funktioniert.

Nachdem der Sensorwert ermittelt wurde werden die Werte in ein Paket zusammengefasst und versendet.

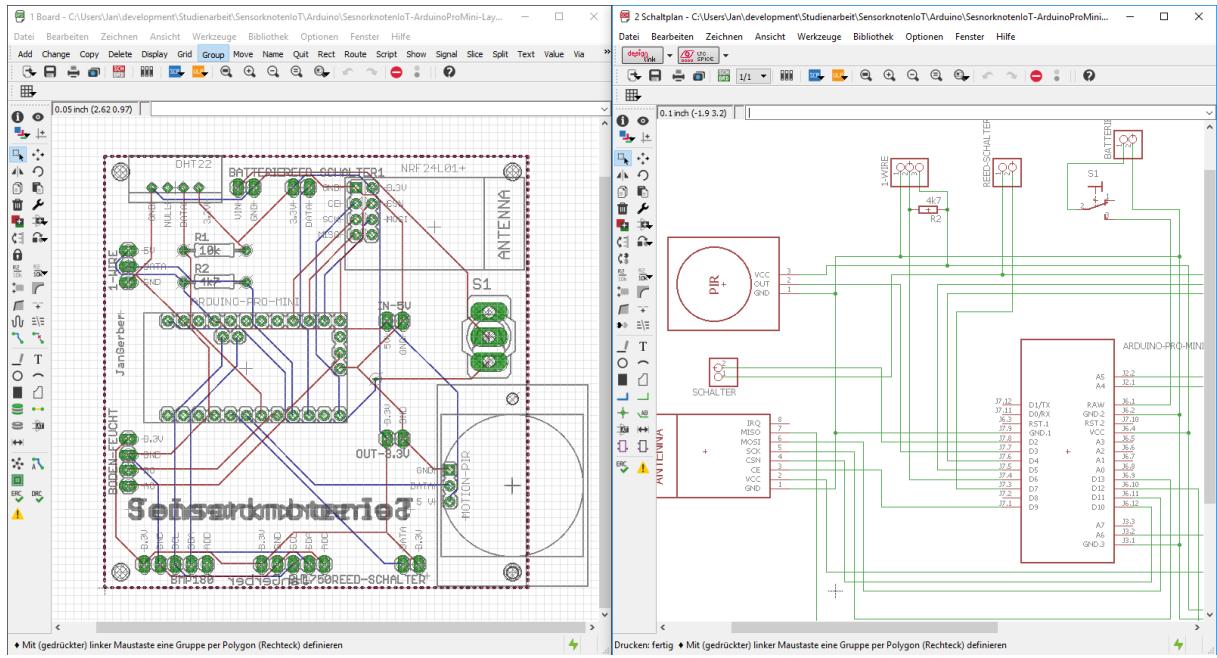


Abbildung 4.3: Eagle Autodesk – Layout Editor (linke Bildhälfte) und Schaltplan Editor (rechte Bildhälfte)

**Eagle Autodesk** Für die Entwicklung des Schaltplans und der Platine, auf der die Sensoren, das Funkmodul und der Arduino aufgebracht werden sollen, wird Eagle von der Firma Autodesk eingesetzt. Bei Eagle handelt es sich um ein PCB<sup>1</sup>-Designer zum Erstellen von Layouts für Platten (siehe Screenshot 4.3). In der Studienarbeit wurde die Version 8.0.2 verwendet. Eagle ist für Windows, Mac und Linux Systeme erhältlich. Eagle besitzt verschiedene Lizenzversionen, die teilweise kostenlos zur Verfügung stehen. Die kostenlose Version ist für die nicht kommerzielle Nutzung lizenziert. Diese enthält einige Einschränkungen. So können maximal zwei Schaltpläne für ein Projekt verwendet werden, es werden maximal zwei Schichten unterstützt und die Größe der Platine ist auf 80 cm<sup>2</sup> beschränkt.

Eagle selbst besteht aus mehreren Komponenten, dazu gehören ein Schaltplan-Editor, ein Layout Editor und PCB-Libraries. Der Schaltplan Editor ermöglicht es einen Schaltplan zu zeichnen. Hierfür können entweder Standardelemente verwendet werden oder über Libraries neue elektronische Bauteile genutzt werden. Zusätzlich bietet Eagle einen ‚Electrical Rule Checking‘ an, mit dessen Hilfe können Fehler im Schaltplan gefunden werden. Durch eine direkte Verknüpfung zwischen Schaltplan und Layout können effizient Platten entwickelt werden. Mit Hilfe des Layouters lassen sich Bauteile einfach platzieren und mit dem Autorouter lassen sich die einzelnen Bauteile mit Leiterbahnen verbinden. Das Design kann anschließend nach definierten Regeln, wie zum Beispiel Abstand zwischen den Leiterbahnen, überprüft werden.

<sup>1</sup>Printed Circuit Board

# Kapitel 5

## Realisierung/Implementierung

### 5.1 Topologie

Nachdem in Kapitel 4.1 ein Konzept für die Topologie entwickelt wurde folgt in diesem Unterkapitel die Beschreibung der Umsetzung des Mesh-Netzes, sowie der allgemeine Aufbau der gesamten Topologie.

#### 5.1.1 Allgemeine Beschreibung

Das gesamte Projekt des Sensorknoten lässt sich in fünf Ebenen unterteilen (siehe Grafik 5.1). Im den folgenden Abschnitten werden diese vier Schichten genauer erläutert.

**Energiesparender Sensorknoten** Der energiesparende Sensorknoten wurde mit einem Arduino Pro Mini realisiert. Dieser ist extrem energiesparend und übernimmt keinerlei Weiterleitungsfunktion im Mesh-Netz. Dieser Sensorknoten sendet seine Daten entweder direkt dem Raspberry Pi oder einem normalen Sensorknoten. Dieser leitet die Nachricht dann zum Raspberry Pi weiter. Auf den genauen Aufbau des energiesparenden Sensorknoten wird in Kapitel 5.3.1 eingegangen.

**Normaler Sensorknoten** Der normale Sensorknoten wird mit Hilfe eines Arduino Nano's realisiert. Der Arduino Nano verfügt über eine permanente Stromversorgung, diese wird über den USB Port realisiert. Dieser Sensorknoten ist das Herz des Mesh-Netzes. Er übernimmt die Weiterleitung von Nachrichten im Netzwerk. Zusätzlich bestimmt er jede Minute seine eigenen Messwerte. Diese werden dann ebenfalls entweder an einen weiteren normalen Sensorknoten weitergeleitet oder wenn der Raspberry Pi in der Nähe ist direkt an ihn.

**Gateway** Das Gateway zur Datenbank bzw. Internet wird mit einem Raspberry Pi realisiert. Dieser verfügt über das gleiche Funkmodul wie die beiden Sensorknoten mit dem

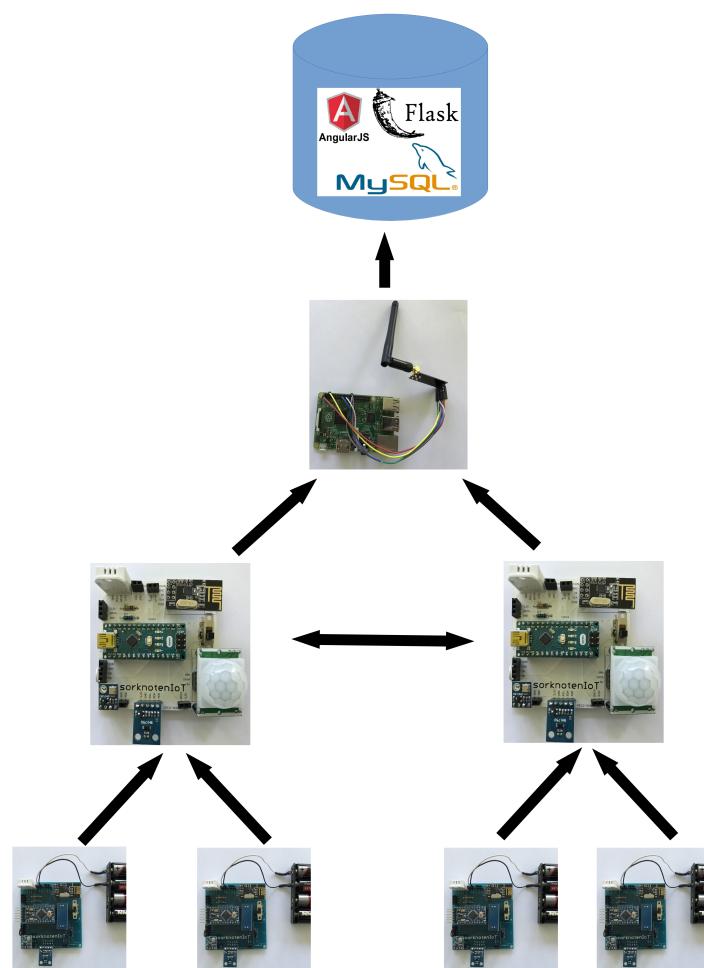


Abbildung 5.1: Topologie Sensorknoten

Unterschied, dass das verwendete Funkmodul mit einer externen Antenne ausgestattet ist. Der Raspberry Pi verfügt zusätzlich über eine Verbindung zum Internet. Diese kann wahlweise mit dem WLAN Modul erfolgen oder über Ethernet. In der DHBW wurde aufgrund von Problemen mit dem verwendeten Authentifizierungsprotokoll Ethernet verwendet. Das genaue Vorgehen des Raspberry Pi's nach dem Erhalten der Nachricht wird in Kapitel 5.5.2 genauer betrachtet.

**Datenhaltungsschicht** Der Raspberry Pi speichert die erhaltenen Daten in einer Datenbank. Diese Datenbank wird auf einem externen Server gehostet. Als relationales Datenbankverwaltungssystem wird MySQL eingesetzt. Der genaue Aufbau der Datenbank wird in Kapitel 5.6 betrachtet.

**Datavisualisierungsschicht** Nachdem die Daten erfolgreich in der Datenbank gespeichert wurden, können Sie mit Hilfe eines REST-Services aus dieser wieder extrahiert werden und auf eine Webseite dargestellt werden. Der REST-Service wurde mit dem Python Microframework Flask (siehe Kapitel 5.7) entwickelt. Der REST Service läuft auf einem NGINX (<https://nginx.org/en/>). Der REST Service kann von vielen verschiedenen Plattformen genutzt werden. So ist es möglich neben einer Webseite noch eine native App für Smartphones zu entwickeln. Die Webseite wird auf einem TomEE (<http://tomee.apache.org/>) gehostet. Die Frontend wurde mit AngularJS, HTML5, CSS3 und Bootstrap entwickelt. Die genaue Beschreibung der Visualisierung erfolgt in Kapitel 5.8.

### 5.1.2 Umsetzung des Mesh-Algorithmus

Die Umsetzung des Mesh-Algorithmus erforderte einige Überlegungen hinsichtlich der begrenzten Systemressourcen. Beide Arduinos verfügen über 2048 Bytes an SRAM für die Speicherung von Variablen. Der entwickelte Algorithmus basierte auf einer Art Blacklist, die die ankommenden Nachrichten überprüft ob sie bereits weitergeleitet wurden. Ein Eintrag in der Blacklist besteht aus einem „unsigned long“, dieser Datentyp hat eine Länge von 4 Bytes. Mit einer zunehmenden Vergrößerung kommt zusätzlich ein erhöhter Suchaufwand hinzu. Aus diesen Gründen heraus wurde die Blacklist auf 50 Einträgen reduziert.

---

Listing 5.1: Auschnitt Mesh Algorithmus

```
void processData(dataPacket t_dataPacket){  
    if(!(t_dataPacket.destinationAddr == arduinoId)){  
        unsigned long uniqueMessageId =  
            (unsigned long) ((unsigned long) t_dataPacket.originAddr << 16)  
            + (unsigned long) t_dataPacket.messageId;
```

```
if(!compareToList(uniqueMessageId)){
    insertInList(uniqueMessageId);
    t_dataPacket.lastHopAddr = arduinoId;
    sendDataPacket(t_dataPacket);
}else{
    #if DEBUG_AUSGABE
        //Message verwefen
        Serial.println("Message verworfen");
    #endif
}
}else{
    #if DEBUG_AUSGABE
        //Arduino ist Empfaenger
        Serial.print("Arduino ist Empfaenger: \t");
        ausgabeDataPacket(t_dataPacket);
    #endif
}
}
```

---

Dieses Listing 5.1 beginnt nachdem die Nachricht empfangen und dekodiert wurde. Zunächst wird überprüft ob der Empfänger der Arduino ist oder nicht. Sollte der Arduino nicht der Empfänger sein, so wird überprüft ob die Nachricht bereits Weitergeleitet wurde. Sollte die Nachricht bereits Weitergeleitet worden sein, so wird sie einfach Verworfen. Ansonsten wird die empfangene Nachricht weitergeleitet und *uniqueMessageId* wird in der Blacklist abgespeichert.

**Anpassung der Bibliothek für den RF24** Der Wechsel zwischen Empfangen und Senden mit der gleichen Hardware-Adresse verursachte mit der RF24 Bibliothek Probleme. Um mit der Bibliothek den Fehler zu beheben hätte das gesamte Funkmodul neu gestartet werden und alle Werte neu initialisiert werden. Diese Lösung war jedoch Aufgrund der langen Startzeit keine Option. Eine Abhilfe konnte geschaffen werden in dem der genutzten Bibliothek eine weitere Funktion *Sensorknoten\_resetRegister()* hinzugefügt wurde (siehe Listing 5.2). Die Funktion setzt einige Register des Funkmoduls auf die Standardwerte zurück. Mit Hilfe dieser Anpassung kann zwischen Empfangen und Senden problemlos gewechselt werden, ohne dass das komplette Funkmodul zurückgesetzt wird.

---

Listing 5.2: Anpassung RF24 Bibliothek

---

```
void RF24::SensorknotenIoT_resetRegister(void){
    write_register(NRF_STATUS, _BV(RX_DR) | _BV(TX_DS) | _BV(MAX_RT));
    flush_rx();
    flush_tx();
```

---

```

write_register(NRF_CONFIG,(read_register(NRF_CONFIG)) & ~_BV(PRIM_RX));
write_register(EN_RXADDR, 0x03 );
}

```

---

## 5.2 Verwendete Sensoren

In diesem Unterkapitel werden auf die verwendeten Sensoren eingegangen. Dabei werden die in Kapitel 4.3 geforderten Messwerte den einzelnen Sensoren zugeordnet. Zusätzlich wird zu jedem Sensor ein kleines Code Listing vorgestellt oder das Vorgehen erklärt mit dem der Sensor ausgelesen werden kann.

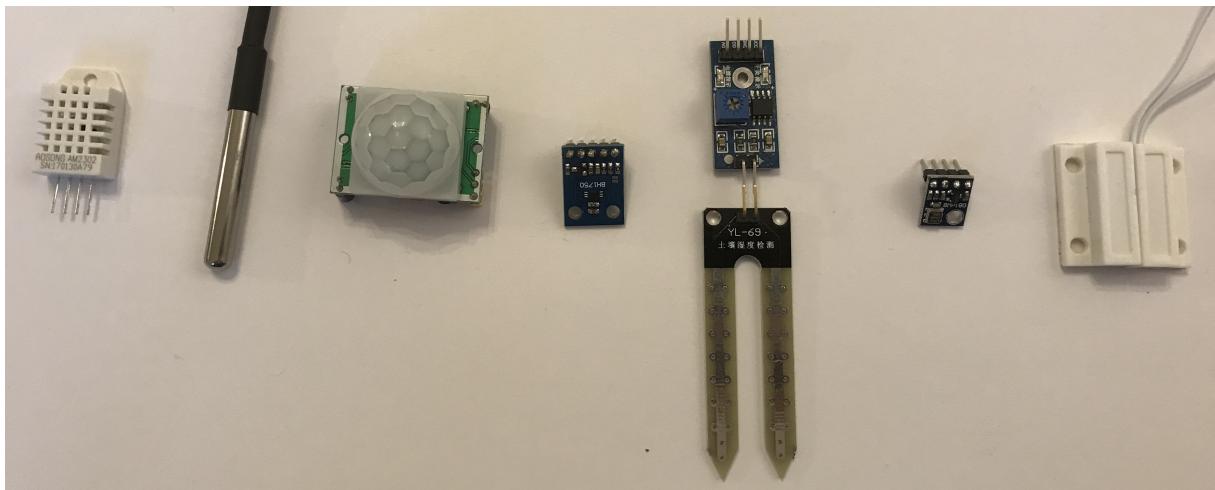


Abbildung 5.2: Verwendete Sensoren(beginnend von Links): DHT22, DS18B20, PIR-Sensor, BH1750, Bodenfeuchtigkeitssensor, BMP180 und Reed-Kontakt

**DHT11 / DHT22** Bei diesem Sensor handelt es sich um zwei Sensoren, die zur Erfassung von Lufttemperatur und Bestimmung der Luftfeuchtigkeit genutzt werden können. Die beiden Sensoren unterscheiden sich nur in der Messgenauigkeit, wobei der DHT22 der genauere der Beiden ist. Die Luftfeuchtigkeit kann bis auf +/- 2% und die Temperatur auf +/- 0,5 Grad bestimmt werden. Der Wert wird digital zurückgeliefert. Für die Auswertung (siehe Listing 5.3) der Sensoren wird eine Bibliothek von Adafruit eingesetzt [Fie16].

Listing 5.3: DHT22 Sensorauswertung

---

```

void getTemperatureHumidity(){
    humidity = dht.readHumidity();
    temperatur = dht.readTemperature();
}

```

---

**BMP180** Beim BMP180 handelt es sich um einen Luftdrucksensor der über die  $I^2C$  Schnittstelle mit dem Arduino verbunden wird. Der Luftdruck kann zwischen 300 und

1100 hPa bestimmt werden [Sen13]. Zusätzlich lässt sich mit diesem Sensor die Temperatur bestimmen, diese Funktion wird allerdings nicht genutzt. Der Sensorknoten bestimmt nur den Luftdruck mit einer Bibliothek von Adafruit (Listing 5.4).

Listing 5.4: BMP180 Sensorauswertung

---

```
void getPressure(){
    sensors_event_t event;
    bmp.getEvent(&event);
    pressure = event.pressure;
}
```

---

**BH1750** Dieser Sensor wird für die Messung der Beleuchtungsstärke eingesetzt, diese wird in Lux gemessen. Der Sensor unterstützt eine Messung im Bereich zwischen 1 und 65535 Lux, bei einer Auflösung von einem Lux. Der Sensor kann wie der BMP180 über die  $I^2C$  Schnittstelle angeschlossen werden. Die Adresse auf der der Sensor erreichbar ist, ist einstellbar. Die Adresse lautet entweder *0x23h* (0 Volt an ADDR- Pin) oder *0x5Ch* (3,3 Volt an ADDR-Pin).

**PIR Sensor** Zur Erkennung von Bewegung wird ein Bewegungsmelder eingesetzt. Der PIR Sensor hat einige Einstellungsmöglichkeiten. Es besteht die Möglichkeit, die Sensitivität und die Dauer des Signals einzustellen. Der PIR Sensor benötigt im Gegensatz zu den vorherigen Sensoren eine Betriebsspannung von 5V. Der Messwert wird einfach mit einem digitalen Eingang gemessen. Eine Nachricht an den Raspberry Pi (Messwertintervall 1min) wird erst nach dem erstmaligen Auslösen des Bewegungsmelders gesendet.

**Bodenfeuchtigkeitssensor** Der Bodenfeuchtigkeitssensor verfügt über zwei Ausgänge. Ein Ausgang ist digital und kann mit einem Drehregler direkt am Sensor eingestellt werden. Der Sensor löst beim digitalen Betrieb ab dem definierten Schwellwert aus. Zusätzlich zum digitalen Ausgang liefert der Sensor noch einen analogen Wert, dieser wird dann in einen prozentualen Wert gewandelt (siehe Listing 5.5).

Listing 5.5: Bodenfeuchtigkeit Sensorauswertung

---

```
void getSoilMoisture(){
    soilMoistureAnalog =(100 - ((100/1023.0)* analogRead(soilMoistureAnalogPin)));
    soilMoistureDigital = digitalRead(soilMoistureDigitalPin);
}
```

---

**Reed-Kontakt** Für die Erkennung ob ein Fenster/Tür geschlossen oder geöffnet ist, wurde ein Reed-Kontakt eingesetzt. Dieser fungiert als eine Art Schalter, der ausgelöst wird sobald ein Magnet außerhalb der Reichweite des Reed-Kontaktes ist. Auf dem Sensorknoten sind

zwei unterschiedliche Reed-Kontakte installiert. Bei einem kann nur zyklisch überprüft werden ob der Kontakt geschlossen oder geöffnet ist. Der zweite ist an einem Pin mit Interrupt Funktion angeschlossen. Wird dieser ausgelöst wird der Interrupt Service Routine abgearbeitet. Die Nachricht ob ein Reed-Kontakt ausgelöst wurde wird allerdings nur zyklisch abgeschickt. Hierfür wird in jedem Zyklus der Alarm zurückgesetzt und der Interrupt wieder aktiviert, sobald der Reed-Kontakt ausgelöst wurde (siehe Listing 5.6). Ein Interrupt wird immer bei wechseln des Signals ausgelöst, das heißt entweder bei einer fallenden oder steigenden Flanke des Signals.

---

Listing 5.6: Reed-Kontakt Sensorauswertung

---

```
boolean getAndResetReedContactSensor(){
    if(interruptHappendReedContact){
        interruptHappendReedContact = false;
        attachInterrupt(digitalPinToInterruption(REED_CONTACT_WITHINT_PIN),
                        interruptContactDetector, CHANGE);
    }
    return reedContactWithInterrupt;
}
```

---

**DS18B20** Beim DS18B20 handelt es sich um einen Temperatur Sensor, der sowohl die Lufttemperatur messen kann als auch die Bodentemperatur bestimmen kann. Der gesamte Sensor ist in einem wasserdichten Gehäuse. Der Sensor kann über die 1-Wire Schnittstelle angesprochen werden. Dadurch können mehrere Sensoren hintereinander gehängt werden. Diese benötigen eine Datenleitung sowie ein GND und 3,3 V bzw. 5V Leitung. Es können Temperaturen zwischen -55 Grad bis zu 125 Grad mit einer Messgenauigkeit von +/- 0,5 Grad gemessen werden. Jeder Sensor besitzt eine eindeutige und einmalige 64-Bit Seriennummer. An den Sensorknoten können beliebig viele 1-Wire Temperatursensoren angeschlossen werden. Es wird nachdem alle Messwerte von den 1-Wire Sensoren bestimmt wurde jeder Messwert einzeln verschickt (siehe Listing 5.7).

---

Listing 5.7: 1-Wire Sensorauswertung

---

```
getOneWireTemperature();
for(int i = 100; i < oneWireDallas.getDeviceCount() + 100 && i <= 199 ;i++){
    sendDataPacket(createSensorDataPacket(oneWireDallas.getTempCByIndex(i-100),
                                         (i) ));
}
```

---

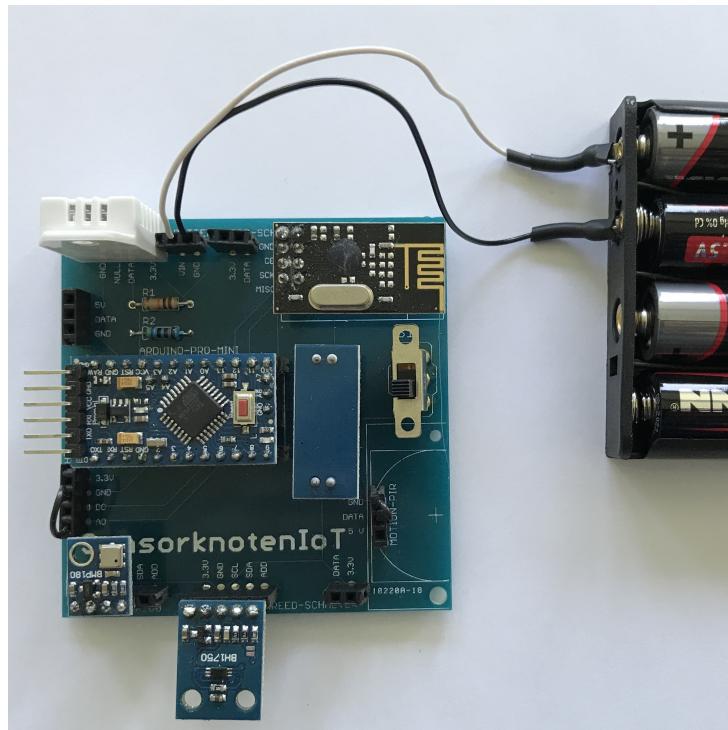


Abbildung 5.3: Energiesparender Sensorknoten: bestückt mit verschiedenen Sensoren und einem Arduino Pro Mini

## 5.3 Hardware Design Sensorknoten

In diesem Unterkapitel werden das Design des Sensorknotens sowie der Produktionsprozess näher betrachtet. Es wird dabei zuerst auf den energiesparenden Sensorknoten, der mit dem Arduino Pro Mini realisiert wurde und anschließend auf den normalen Sensorknoten, der mit dem Arduino Nano eingegangen.

### 5.3.1 Arduino Pro Mini

Der energiesparende Sensorknoten wird dauerhaft mit Batterien betrieben. Dieser ist in Bild 5.3 zu sehen. Der Sensorknoten ist mit einem Arduino Pro Mini bestückt, da dieser energiesparender ist als der Arduino Nano. Sensorknoten lassen sich mit den in Kapitel 5.2 vorgestellten Sensoren bestücken. Die Sensoren können mit Hilfe von Stifteleisten jederzeit getauscht werden. Nur beim DHT22 bzw. DHT11 haben wir uns entschieden diesen direkt aufzulöten. Der Grundgedanke dabei war, dass jeder Sensorknoten als Mindestanforderung die Temperatur und Luftfeuchtigkeit bestimmen kann. Jeder Sensorknoten besitzt ein nRF24L01 Modul zur Kommunikation mit anderen Sensorknoten oder dem Raspberry Pi.

**Spannungsversorgung** Der Arduino Pro Mini verfügt über kein 5V zu 3,3V Wandler, weshalb zusätzlich noch ein Wandler aufgebracht wurde. Dieser 5V zu 3,3V Wandler ist ebenfalls nur mit Hilfe von Stifteleisten aufgebracht.

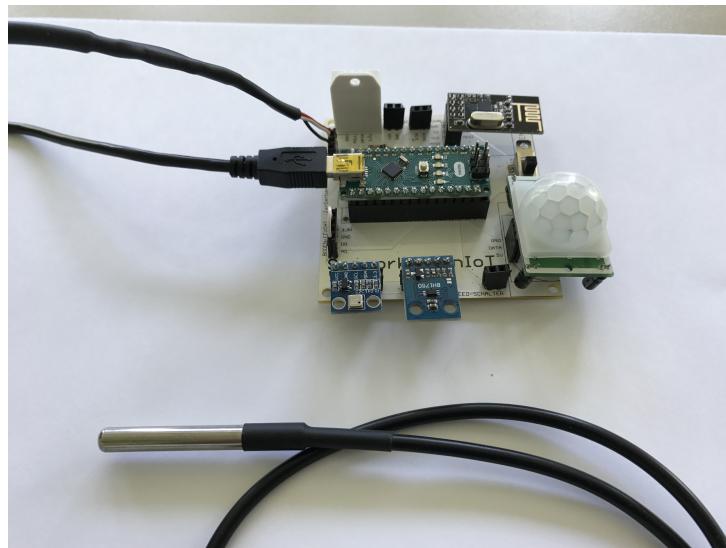


Abbildung 5.4: Normaler Sensorknoten: bestückt mit verschiedenen Sensoren und einem Arduino Nano

Die Spannungsversorgung des Sensorknoten erfolgt mit 4 x 1,5 AA Batterien. Diese liefern gemeinsam eine Spannung von 6V zu Beginn ihrer Betriebszeit. Der Sensorknoten kann bis ca. 4,2V betrieben werden. Die Spannungszufuhr kann mit Hilfe eines Schalters unterbrochen werden.

**Aufgetretene Probleme** Bei der Erstellung des Schaltplans ist ein Fehler bei Sensorschnittstellen für den BMP180 und BH1750 entstanden. Beide Sensoren werden über das  $I^2C$  Schnittstelle angesprochen. Hierbei kam es zu einer Vertauschung der 3,3V Leitung und der Masse Leitung. Aus diesem Grund können diese beiden Sensoren nur mit einem Adapter bzw. mit flexiblen Steckbrücken betrieben werden. Die Funktion der Schnittstelle ist davon nicht betroffen.

Zusätzlich kam es bei der Auswertung der Reed-Kontakte zu zufälligen falschen Werten. Diese konnten behoben werden in dem ein zusätzlicher 10k Ohm Pull Down-Widerstand eingelötet wurde, der das Signal auf Masse zieht.

Um zusätzlich zufällige Werte auszuschließen, wenn kein Bewegungsmelder oder Bodenfeuchtigkeitsmesser angeschlossen ist, wurde eine Steckbrücke genutzt. Diese Steckbrücke verbindet die Datenleitung mit Masse. Durch dieses Verfahren kann einfach überprüft werden ob die Sensoren angeschlossen sind oder nicht.

### 5.3.2 Arduino Nano

Der normale Sensorknoten ist mit einem Arduino Nano bestückt. Dieser verfügt über die gleichen Anschlussmöglichkeiten wie der energiesparende Sensorknoten. In Bild 5.4 ist der normale Sensorknoten zu sehen.

**Spannungsversorgung** Der normale Sensorknoten kann entweder über die vorhandene USB Schnittstelle betrieben werden oder ebenfalls wie der Arduino Pro Mini über eine externe Stromversorgung, wie zum Beispiel Batterien oder Akkus. Da diese Sensorknoten jedoch deutlich mehr Energie benötigt als der energiesparende Sensorknoten, sollten genügend mAh zur Verfügung gestellt werden. Die externe Stromversorgung kann über einen Schalter ausgeschalten werden. Der Arduino Nano verfügt direkt auf dem Board ein Wandler von 5V zu 3,3V.

**Aufgetretene Probleme** Mit dem normalen Board sind die gleichen Probleme hinsichtlich dem Einsatz von Widerständen aufgetreten. Diese Probleme konnten wie beim energiesparende Sensorknoten gelöst werden. Das Sensorknoten Board war allerdings vollständig richtig, was das aufbringen und wechseln von Sensoren deutlich erleichtert.

### 5.3.3 Produktionsprozess der Sensorknoten

In diesem Unterkapitel wird auf den gesamten Produktionsprozess eingegangen. Beginnend mit der Entwicklung eines Prototypens, über die Erstellung genauer Schaltpläne und abschließend mit der Bestellung der Platinen, sowie der Bestückung dieser Platinen.

**Erstellung eines Prototypen** Um sich in das Thema einzuarbeiten, wurde zunächst ein Prototyp entwickelt (siehe Bild 5.5). Dieser wurde auf einer einfachen Lochrasterplatine entwickelt. Die einzelnen Komponenten wurden mit Drahtstücken verbunden. Der Löts- und Bestückungsvorgang ging pro Prototyp 2-3 Stunden. Auf dem Prototyp war das Funkmodul, der DHT22 Sensor und eine Sensor mit einer  $I^2C$  Schnittstelle aufgebracht. Alle Sensoren konnten in Steckleisten befestigt werden. Dies ermöglichte einen schnellen Austausch, falls ein Sensor ausfallen würde. Für die Erstellung der Prototypen wurde ein handschriftlicher Schaltplan entworfen.

**Erstellung eines Schaltplans** Nachdem ein funktionsfähiger Prototyp entworfen wurde, konnte ein Schaltplan für den energiesparenden Sensorknoten und den normalen Sensorknoten entwickelt werden. Die Schaltpläne beider Sensorknoten sind im Anhang zu finden. Der Schaltplan wurde komplett entworfen ohne alle Bauteile getestet zu haben. Zusätzlich wurde nur für den normalen Sensorknoten ein Prototypen entworfen, da zum Zeitpunkt der Erstellung des Prototypen die Arduino Pro Mini nicht vorlagen. Der Schaltplan wurde mit Hilfe von Eagle (siehe Kapitel 4.3) erstellt.

**Erstellung eines Layouts** Eagle bietet die Möglichkeit, nach der Erstellung eines Schaltplans ein Layout für diesen Schaltplan zu entwickeln. Zunächst muss die Größe der Platine festgelegt werden, diese ist bei den Sensorknoten 70 mm \* 69 mm. Im nächsten Schritt werden alle Bauteile auf der Platine platziert und positioniert. Die Bauteile wurden so

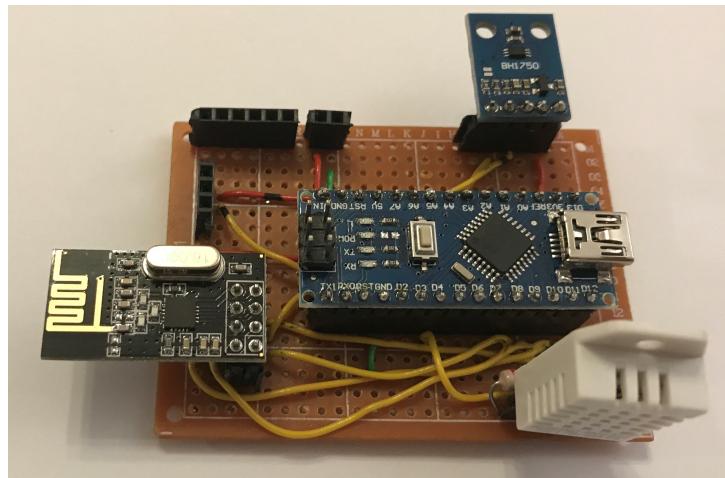


Abbildung 5.5: Prototyp Sensorknoten bestückt mit einem Arduino Nano, DHT22 und dem nRF24L01 Funkmodul und wahlweise mit einem BH1750 oder BMP180

positioniert, dass genügend Platz zwischen den Bauteilen ist. Anschließend können die Leiterbahnen, mit Hilfe des Autorouters auf der Leiterplatte verlegt werden. Die Leiterbahnen werden nach bereits vorher angegebenen Anforderungen (zum Beispiel Abstand und Breite Leiterbahnen) verlegt. Im letzten Schritt wird die Platine noch beschriftet und Bohrlöcher in den Ecken hinzugefügt. So besteht auch die Möglichkeit die Platinen in einem Gehäuse zu befestigen.

**Bestellprozess** Da das Erstellen der Platinen Zeitaufwendig wäre haben wir uns entschieden die Platinen produzieren zu lassen. Nach einem kurzen Vergleich verschiedener Hersteller kamen wir zum Schluss, dass eine Bestellung bei einem chinesischen Hersteller die günstigste Variante ist. Der Hersteller war um den Faktor 10 billiger. Wir haben uns für die Webseite <https://www.seeedstudio.com> entschieden.

Zunächst musste ein das erstellte Layout in das Gerber-Format exportiert werden. Das Gerber Format enthält alle wichtigen Informationen, die zur Produktion der Platine benötigt werden. Es handelt sich bei diesem Format um eine Art Quasi-Standard. Die Gerber Dateien müssen anschließend gezippt werden und können auf die Webseite hochgeladen werden. Die Webseite bietet ein Gerber-Viewer um die Leiterplatte nach dem Hochladen noch einmal auf die Korrektheit zu überprüfen (siehe Bild 5.6). Dieser Schritt war enorm hilfreich, da so festgestellt werden konnte, dass die Beschriftungen nicht richtig skaliert wurden und Sonderzeichen nicht dargestellt wurden. Eine Abhilfe konnte durch eine Umwandlung der Schriften in eine Vektorgrafik erlangt werden.

Nachdem die Dateien im Gerber-Format hochgeladen wurden mussten noch einige Angaben zur Platine selbst gemacht werden. Diese Angaben waren für beide Platinen die gleichen mit einer Ausnahme:

- Material: FR-4 → Verbundwerkstoff aus Epoxidharz und Glasfasergewebe

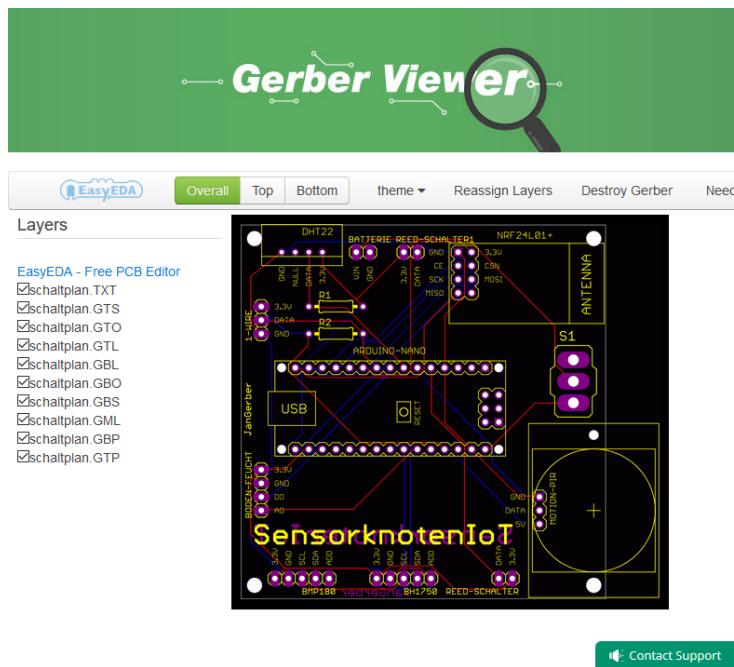


Abbildung 5.6: Gerber-Viewer – Seeedstudio.com

- Anzahl Schichten: 2
- Größe der Platine: 70 mm x 69 mm
- Anzahl an Platinen: 10 Stück
- Dicke der Platine: 1,6 mm
- Farbe der Platine: Weiß (normaler Sensorknoten), Blau (energiesparender Sensor-knoten)
- Oberflächenbehandlung: HASL (Hot Air Solder Leveling mit Zinn/Blei)
- Mindestabstand zwischen zwei Leiterbahnen: 0,4 mm
- Gewicht der Kupferbahnen:  $1oz/ft^2$  entspricht  $300g/m^2$
- Mindestgröße von Bohrlöchern: 0,3 mm

Nachdem die Eigenschaften alle festgelegt wurden, wurde der Bestellvorgang fortgesetzt. Im letzten Schritt musste noch die Versandmethode gewählt werden. Wir entschieden uns für die günstigste, die jedoch auch die längste Versanddauer hatte. Der Endbetrag errechnete sich dann aus 10,90 Euro Versandkosten und zweimal 4,51 Euro für die Platinen.

Die Platinen wurden am 14. November 2016 bestellt. Am 22. November 2016 waren die Platinen fertiggestellt und wurden verschickt. Den Versand wurde von Singapore Post und DHL übernommen. Die Ware wurde per Luftfracht verschickt und wurde am 21. Dezember 2016 geliefert. Es gab bei der Zustellung jedoch auch Probleme, da bei dem Bestellvorgang die falsche PLZ angegeben wurde und sich deshalb das Paket um eine Woche verzögerte.

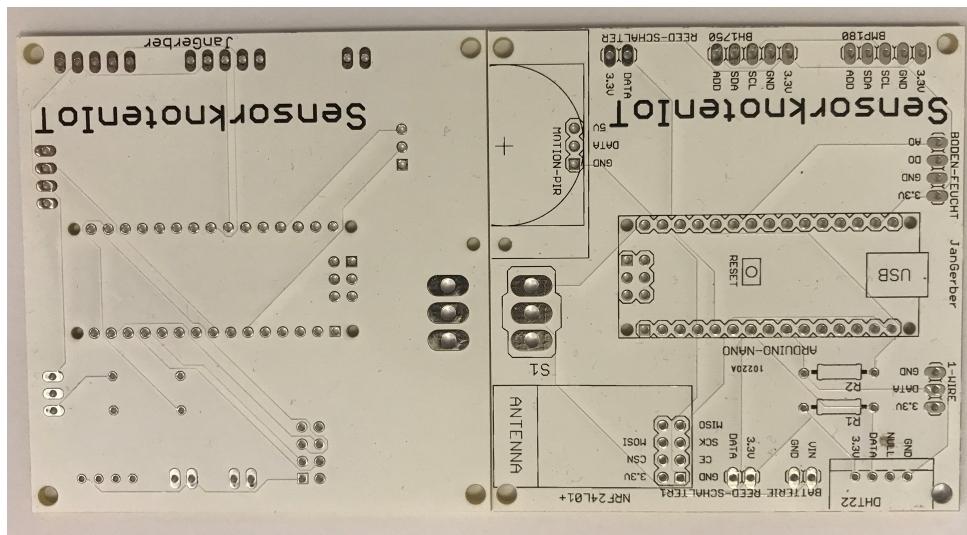


Abbildung 5.7: Platine für normaler Sensorknoten

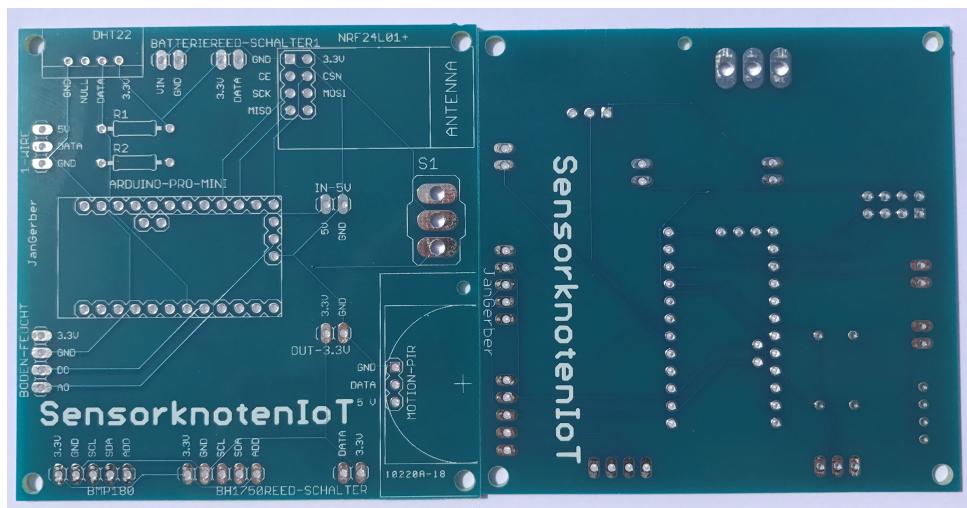


Abbildung 5.8: Platine für energiesparender Sensorknoten

**Bestückung de Platinen** Die Platinen entsprachen vollständig dem erwarteten Ergebnisses (siehe Bild 5.8 und 5.7). Auf die Platinen konnte extrem einfach und schnell die Steckverbindungen, der Schalter, die Widerstände und den DHT22 Sensor aufgebracht werden. Die Produktionszeit einer Platine reduzierte sich so von 3 Stunden auf 10 Minuten. Zusammenfassend kann eine Empfehlung für *Seeedstudio* ausgesprochen werden. Falls das Produkt jedoch kommerziell genutzt werden sollte, darf aufgrund von neuer EU-Richtlinien nur noch bleifreies Lötzinn verwendet werden.

## 5.4 Energiesparmodus Arduino Pro Mini

In diesem Unterkapitel wird auf die eingesetzten Techniken und Anpassungen eingegangen. Diese sind notwendig, um einen sehr energiesparenden Sensorknoten zu entwickeln. Der Sensorknoten sollte wie bereits beschrieben nur mit 4x AA-Batterien betrieben werden. Hierfür wurden drei grundlegende Ideen umgesetzt. Diese werden in den folgenden Abschnitten genauer erläutert.

**Verwendung des Arduino Pro Mini** Der Arduino Pro Mini eignet sich besonders für Projekte bei denen nur wenig Energie verbraucht werden darf. Der Pro Mini verfügt im Gegensatz zum Arduino Nano über keinen USB Anschluss und keinen Spannungswandler von 5V zu 3,3V. Diese beiden Komponenten benötigen zusätzlich Strom. Durch den Verzicht auf diese beiden Komponenten kann ca. 2 - 4mA eingespart werden.

**Entfernen der Status LED** Auf den Arduinos ist eine Status LED aufgebracht. Diese zeigt nur an, ob der Arduino aktiv ist oder nicht. Da diese LED keine weitere Funktion hat kann sie entfernt werden. Das Löten von SMD-LEDs ist extrem schwierig. Aus diesem Grund wird nicht die LED selbst ausgelötet, sondern der verbaute Vorwiderstand. So kann gegebenenfalls die LED einfacher wieder in Betrieb genommen werden. Der Ausbau der LED bringt eine Ersparnis von 3 – 4 mA.

**Verwendung des Watchdog-Timer und Sleep Modus** Um jedoch wirklich Energie zu sparen muss der Mikrocontroller in einen Schlafmodus versetzt werden. Hierfür wird der Watchdog-Timer verwendet (siehe Kapitel 3.2.1). Der Watchdog Timer wird verwendet um den Mikrocontroller 8 Sekunden in den Schlafmodus zu versetzen. In dieser Zeit verbraucht der Mikrocontroller fast keinen Strom mehr, da dieser in der Zeit keine Programmabarbeitungen vornimmt. Die Umsetzung dieser Funktion wurde mit Hilfe einer *Low-Power* Bibliothek (<https://github.com/rocketscream/Low-Power>) umgesetzt.

In Tabelle 5.1 sind die Einsparungen der verschiedenen Techniken zu sehen. Bei der Messung war der DHT22, BH1750 und das Funkmodul angeschlossen. Am besten hat die

Tabelle 5.1: Stromstärken bei den verschiedenen Energiespartechniken

	Normaler Modus	Sleep Modus
Arduino Nano	19,61 - 29,31 mA	7,62 mA
Arduino Pro Mini	17,26 - 28,28 mA	3,24 mA
Arduino Nano ohne LED	15,71 - 26,20 mA	3,09 mA
Arduino Pro Mini ohne LED	14,96 - 24,83 mA	182 $\mu$ A

Kombination aus einem Arduino Pro Mini ohne LEDs im Sleep-Modus mit nur 182  $\mu$ A abgeschnitten. Mit diesen Techniken kann der energiesparende Sensorknoten über mehrere Monate hinweg über Batterie betrieben werden.

## 5.5 Raspberry Pi

Dieses Kapitel beinhaltet die Realisierungen, die den Raspberry Pi betreffen.

### 5.5.1 Verkabelung

Wie eingangs bereits erwähnt, nutzt der Raspberry RF24-Chip. Der Modul existiert in mehreren Ausführungen. Für den Raspberry Pi kommt ein Modul mit externem Antennenanschluss zu Einsatz. Dieses Modul verfügt dank der externen Antenne über eine größere Reichweite. Es hat jedoch auch einen höheren Energiebedarf und die Bauform ist nicht so kompakt wie die Ausführung ohne externe Antenne. Diese zwei Nachteile kommen im vorliegenden Fall beim Raspberry Pi jedoch nicht zum tragen.

**Belegung der GPIO-Pins** Die Belegung der GPIO-Pins kann man dem Handbuch der RF24-Bibliothek entnehmen:

#Pin RF24	Name auf RF24	#Pin Raspberry	Name auf Raspberry
1	GND	6	GND
2	VCC	1	3,3 Volt
3	CE	31	GPIO22
4	CSN	3	GPIO8
5	SCK	23	SCKL(GPIO14)
6	MOSI	19	MOSI(GPIO12)
7	MISO	21	MISO(GPIO13)
8	IRQ		

### 5.5.2 Receiver

*Receiver* meint die Software, die auf dem Raspberry Pi läuft und dort die Nachrichten den Arduinos empfängt. Anfangs war der Receiver in C implementiert. Empfangene Nachrichten

(C-Structs) hat das C-Programm geparsst (den Bitstrom wieder in einen Struct geschrieben) und in eine Datei geschrieben. Um zu verhindern, dass die Datei unnötig groß wird und um Zugriffe auf die SD-Karte zu verhindern, ging man dazu über eine named Pipe zu nutzen. Da named Pipes nur ein Eintrag im Dateisystem haben, die Speicherstelle jedoch auf den RAM gemappt ist, ist die sehr schnell. Ein zweites Programm in Python hat die empfangenen Daten verarbeitet. Da sich die Installation weiterer Bibliotheken für den Raspberry Pi in C als sehr umständlich erwies, entwickelte man einen Receiver in Python. Die Bibliothek stellt hierfür auch Funktionen bereit.

Zu Beginn erfolgt eine Initialisierung:

---

```
1 radio.begin()
2 radio.payloadSize = 28
3 radio.enableDynamicPayloads()
4 radio.setAutoAck(1)
5 radio.setDataRate(RF24_250KBPS)
6 radio.setPALevel(RF24_PA_MAX)
7 radio.setChannel(90)
8 radio.setRetries(15, 15)
```

---

Der Aufruf in Zeile 2 legt fest, wie viel Bytes die eintreffenden Nachrichten haben. Zeile 4 sorgt dafür, dass der Raspberry dem Sender den erfolgreichen Empfang sofort automatisch bestätigt. Ferner kann man auch die Datenrate anpassen (Zeile 5) und auch die Empfangs-/Sendeleistung des RF24-Moduls. Im Falle des Raspberry Pis ist die Leitung auf maximal gesetzt. Da er an das Stromnetz angeschlossen ist, ist die Leitungsaufnahme weniger relevant. Das Funkmodul verfügt über 126 Kanäle. Jeder Kanal hat eine Breite von einem Megaherz. Der Kanal ist wählbar über die Funktion *setChannel(CHANNEL)* in Zeile 7. Außerdem kann man definieren, wie oft das Modul versuchen soll, zu senden bevor es eine Bestätigung (Ack) erhält.

Der eigentliche Empfang der Daten erfolgt über den simplen Funktionsaufruf:

---

```
receive_payload = readRadio()
```

---

[Ele08]

### 5.5.3 Kodierung

Das Parsen des empfangenen Bitstroms in einen C-Struct war in Python weiteres nicht mehr möglich. Eine der in Kapitel 4.2 erwähnten Möglichkeiten der Kodierung ist das Erstellen eines Strings und dessen Versand auf dem Arduino. Ein Problem hierbei ist jedoch, das man beachten muss, dass die Länge des Strings variieren kann. Man muss nach der Konkatenation des Strings sicherstellen, dass der String die erwartete Länge hat. Ansonsten kann es zu Problemen bei der Dekodierung beziehungsweise beim Parsen

der Daten kommen. Diese Gründe sowie die schlechte Erweiterbarkeit motivierten die Kodierung explizit zu realisieren. Eine der in Kapitel 4.2 erwähnten Möglichkeiten der Kodierung ist das Erstellen eines Strings und dessen Versand auf dem Arduino. Ein Problem hierbei ist jedoch, dass man beachten muss, dass die Länge des Strings variieren kann. Man muss nach der Konkatenation des Strings sicherstellen, dass der String die erwartete Länge hat. Ansonsten kann es zu Problemen bei der Dekodierung beziehungsweise beim Parsen der Daten kommen. Diese Gründe sowie die schlechte Erweiterbarkeit motivierten die Kodierung explizit zu realisieren.

---

```
decodedData = base64.b64decode(receive_payload)
```

---

Es kommt Base64 zum Einsatz. Die Kodierung erfordert zwar auf Seite der Arduinos Rechenzeit, die gesendeten Nachrichten sind dadurch jedoch eindeutig zu dekodieren.

Nach der Dekodierung kann das Programm mit Hilfe des *struct*-Moduls den C-Struct rekonstruieren:

---

```
destinationAddr, originAddr, lastHopAddr, messageID, stationID, value, unit,  
timeID = unpack('<hhhhfhL', decodedData)
```

---

Die deklarierten Variablen erhalten den dementsprechenden Rückgabewert der Funktion *unpack*. Als erstes Argument nimmt sie die Byte order. In diesem Fall ist sie *little-endian*. Anschließend folgen die Datentypen. *h* steht für Short, wobei es sich auf dem Arduino um einen Integer handelt (auf einem x86 würde es sich um einen Short handeln). *f* steht für Float, eine Fließkommazahl und *L* für einen unsigned Long, einen long Integer ohne Vorzeichen.

[Fou17]

## 5.5.4 Hashing

Um später einen eindeutigen Schlüssel für die Datenbank zu haben, nutzt das Programm mehrere Attribute. Eine Kombination aus der ID des Arduinos, der Nachrichten-ID und dem hochgezählten Zeitstempel (der Arduino hat keine Echtzeituhr) ist eindeutig, solange keiner dieser Variablen überläuft. Ein Überlauf sollte in ausreichend großer Zukunft liegen.

---

```
def genearteID_hashed(stationID, messageID, timeID):  
    str1 = str(stationID)  
    str2 = str(messageID)  
    str3 = str(timeID)  
    toHash = str1 + str2 + str3  
    hashed = hashlib.md5()  
    hashed.update(toHash)  
    return hashed.hexdigest()
```

---

Es kommt ein MD5-Hash zum Einsatz. Dabei handelt es sich um einen 128 Bit langen Hash.

### 5.5.5 Datenbankanbindung

Das Programm schreibt die empfangenen Daten auch in die Datenbank. Abgesehen von dem Exception-Handling sieht das wie folgt aus:

---

```
1 def writeToDatabase(ID_hashed, originAddr, value, unit):
2     timeStamp = int(time.time())
3     queryCurs.execute(''INSERT IGNORE INTO messwerte (id,originAddr,value,unit,
4                         timestamp)
5                     VALUES (%s,%s,%s,%s)'', (ID_hashed, originAddr, value, unit,
6                         timeStamp))
7     DBconn.commit()
```

---

Das *IGNORE* in Zeile 3 erspart eine Prüfung, ob die Nachricht bereits verarbeitet ist. Ist die Nachricht in der Datenbank vorhanden, so passiert nichts weiter. Die Datenbank bricht stillschweigend die Transaktion ab. Das entfallen der Prüfung erspart Rechenzeit. Skaliert das gesamte System vertikal (mehr Sensorknoten), so könnte es andernfalls hier zu Engpässen kommen.

### 5.5.6 Scheduling

Das Receiverprogramm startet sobald der Raspberry Pi bootet. Außerdem sollte der Receiver den Betrieb automatisch wieder aufnehmen nachdem er ungewollter Weise beendet wurde. Um das zu realisieren, nutzt man Systemd. Systemd ist ein Initialisierungssystem von Linux. Es ist das erste Programm das beim Boot startet und weitere Programme nachlädt. In diesem Fall auch den Receiver (receiverMesh.py).

---

```
1 [Unit]
2 Description=Receiver
3 After=multi-user.target
4
5 [Service]
6 Type=simple
7 ExecStart=/usr/bin/python /home/pi/sensorknoten/receiverMesh.py
8 Restart=on-abort
9
10 [Install]
11 WantedBy=multi-user.target
```

---

Um einen automatischen Start nach einem Fehler zu gewährleisten bedarf es Zeile 8. Systemd loggt derartige Fälle womit man Fehler identifizieren und beheben kann.

### 5.5.7 Logging

Neben Systemd loggt auch das Receiverprogramm. Die Logfile trägt den Namen receiver.log. Das Programm legt sie in dem gleichen Verzeichnis in dem es läuft an. Alle auftretenden Exceptions die nicht zum Programmabsturz führen sollen, finden sich darin wieder.

## 5.6 Datenbankschema

Zu Beginn der Entwicklung kam *SQLite* als Datenbankmanagementsystem zum Einsatz. Es stellte sich jedoch heraus, dass es zu Problemen bei gleichzeitigem Zugriff auf die Datenbank kommt. Daher kommt momentan *MySQL* zum Einsatz. MySQL ist weit verbreitetes Datenbankmanagementsystem, es ist Open Source und frei verfügbar. Mit mehreren Datenbankverbindungen kommt es problemlos aus. Das System wird auf dem gleichen Host wie der Webserver für die Datenrepräsentation und der Webservice für die Datenbereitstellung ausgeführt. Dieser externe Server läuft in einer virtuellen Maschine der DHBW Karlsruhe. Das Betriebssystem des Server ist ein Ubuntu-Server. Die Datenbank beinhaltet drei

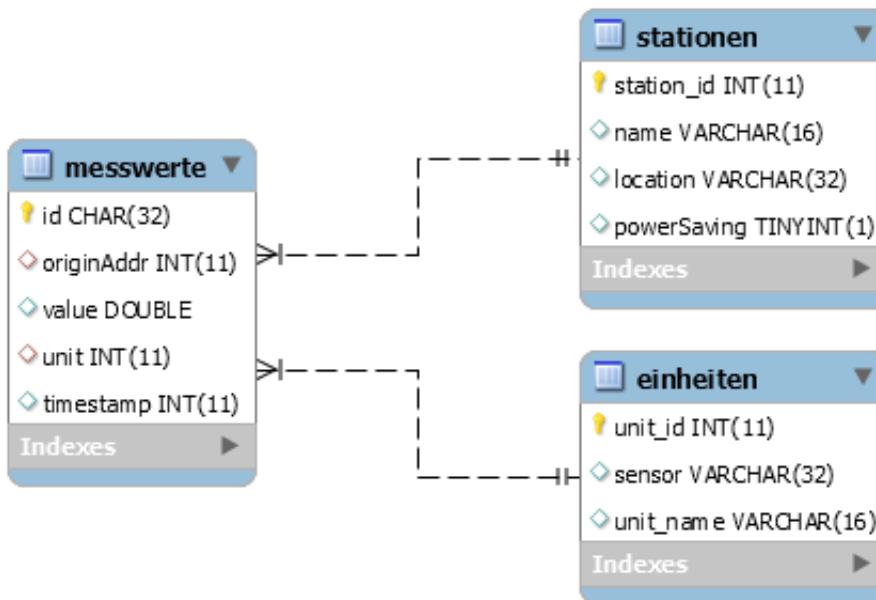


Abbildung 5.9: Schema der Datenbank

Tabellen. Bei dem Attribut *id* vom Typ CHAR(32) in der Tabelle *messwerte* handelt es sich um die gehashte ID. Diese Tabelle enthält die eigentlichen Messdaten. Die ID dient als Primarykey. Das Attribut *originAddr* gibt an, von welchem Sensorknoten (Arduino) das Tupel stammt. *value* gibt den Messwert der entsprechenden Messgröße beziehungsweise Einheit (*unit*) an. Der *timestamp* ist im UNIX-Format. Er zählt die Sekunden seit dem ersten 01.01.1970 hoch. Diese Datumsrepräsentation ist besonders simpel. Da die Arduinos keine Echtzeituhr haben, erstellt der Raspberry Pi beim Empfang den Zeitstempel. Die

Attribute *originAddr* und *unit* sind in *messwerte* jeweils Fremdschlüssel. *originAddr* ist Primärschlüssel in der Tabelle *stationen*. Sie dient der Speicherung aller Sensorknoten. Die Tabelle enthält außerdem noch die Spalte *name*, den Standort (*location*) und ein Flag (*powerSavig*), ob es sich um einen energiesparenden Arduino handelt oder nicht. *einheiten* hat als eindeutigen Primärschlüssel *unit\_id*. In der Tabelle sind alle Messgrößen enthalten. Ein String (VARCHAR(32)) bezeichnet den Sensor (zum Beispiel Luftfeuchte). *unit\_name* repräsentiert die Einheit der Messgröße (zum Beispiel %)

## 5.7 Webservice

Um von der Clientseite für die Datenrepräsentation die Messdaten zu erhalten, kommt ein Webservice zum Einsatz. Der Webservice ist wie in Kapitel 5.1 erwähnt in Python implementiert. Das Microframework *Flask* kam zum Einsatz. Es ermöglicht eine einfache und effektive Implementierung des Webservices. Der Webservice gibt die angefragten Ressourcen im JSON-Format zurück.

**Routen** Der Webservice verfügt über folgende Routen:

1. /mdata/station
2. /mdata/station/<int:station>
3. /mdata/station/<int:station>/<int:unit>
4. /mdata/<string:mdatum\_id>

Frägt man auf der ersten Route an, so erhält man eine Liste mit allen in der Datenbank enthaltenen Stationen, zum Beispiel:

---

```
"location": "Wohnzimmer",
"name": "Station100",
"originAddr": 100,
"powerSaving": 0
```

---

Mit Hilfe der zweiten Route kann man alle Messdaten einer Station abrufen. Hier ein Ausschnitt:

---

```
"originAddr": 400,
"sensor": "Luftdruck",
"timestamp": 1494414190,
"unit": 4,
"unit_name": "hPa",
"uri": "http://193.196.7.13:8080/mdata/f7bc3361937ee46c33de8aff35196572",
"value": 993.0
```

---

Über die URI ist die Ressource direkt adressierbar. Dazu kann man die vierte oben genannte Route nutzen, wobei der String *mdatum* der URI entspricht. Die URI ist in der Datenbank die ID. Sie ist auch der Primärschlüssel. Die Erzeugung dieser ID erfolgt wie in 5.5.4 beschrieben. Die dritte Route kann man nutzen um Messdaten einer Größe für einen definierten Zeitraum zu erhalten:

---

```
1 @app.route('/mdata/station/<int:station>/<int:unit>', methods=['GET'])
2 @auth.login_required
3 def get_mdataUnit(station, unit):
4     begin = request.args.get('begin')
5     end = request.args.get('end')
6     anzahldatenpunkte = request.args.get('anzahl')
7
8     if begin is not None and end is not None:
9         query_result = queryDB_station_interval(station, unit, begin, end)
10        query_result = minimizeData(query_result)
11        if len(query_result) != 0:
12            return jsonify({'Messdaten': [make_public_mdatum(data) for data in
13                                         query_result]})
```

---

```
13
14 abort(404)
```

Die Methode nutzt POST-Parameter. In diesem Fall sind die POST-Parameter nicht optional, Zeile 8 prüft, ob sie gegeben sind. Ist die der Fall und die Abfrage aus der Datenbank nicht leer, so springt Zeile 12 aus der Methode raus, ansonsten wirft Zeile 14 den 404 HTTP-Fehler (not found).

**Virtuelle Umgebung** Um das oben beschriebene Pythonprogramm sicher auf dem Webserver nginx zu veröffentlichen, bedarf es im Idealfall einer virtuellen Pythonumgebung. Diese garantiert unter anderem, dass der User *www-data* (er führt den Webserver aus) nicht unnötig viele Berechtigungen benötigt. Ferner kann die virtuelle Umgebung alle Abhängigkeiten beinhalten. Das Hostsystem muss neben der Installation von Python (ist bei Linux im Regelfall vorinstalliert) keine weiteren Voraussetzungen erfüllen. Im Rahmen dieser Arbeit kommt *virtualenv* zum Einsatz. Mithilfe des Tools *pip* kann man Python-Module (Bibliotheken) in dieser virtuellen Umgebung installieren. Die virtuelle Umgebung enthält auch eine Python-Binary.

**Veröffentlichung auf Webserver** Um dem Webserver nginx zu ermöglichen, ein Pythonprogramm auszuführen benötigt er einen Applikationsserver. *uWSGI* ist einer dieser Server. Um die Pythonapplikation ausführen zu können, ist er unter anderem wie folgt konfiguriert:

---

```
1 [uwsgi]
2 #application's base folder
3 base = /var/www/restAPI
4
5 #python module to import
6 app = getData_web
7 module = %(app)
8
9 home = %(base)/venv
10 pythonpath = %(base)
```

---

Zeile 3 definiert das Wurzelverzeichnis des Servers. In Zeile 6 gibt man den Namen der Pythonanwendung bekannt. In Zeile 9 kann man nun den Pfad nur Python-Binary angeben. Da es sich hier wie oben beschrieben um eine virtuelle Pythonumgebung handelt, ist der Pfad zur Binary %(base)/venv.

## 5.8 Datenvisualisierung

Ein weiteres Ziel der Studienarbeit war es die gesammelten Daten zu präsentieren. Die Entscheidung fiel auf eine Webapplikation. Der Grund für diese Entscheidung war, dass die Anwendung von vielen verschiedenen Geräten genutzt werden kann. Die Webseite kann unter <http://sensorknoteniot.it.dh-karlsruhe.de/> aufgerufen werden, der Benutzername lautet *tester* mit dem Kennwort *python*.

Als Technologie wurde AngularJS gewählt, ein clientseitiges JavaScript Webframework zur Erstellung von Singel-Page-Webanwendungen. Zur Präsentation der Webseite wurde HTML5 und CSS3 eingesetzt. Zur Oberflächengestaltung wurde Bootstrap (CSS-Framework) eingesetzt.

Die Webseite besteht aus drei Seiten insgesamt. Auf diese kann erst nach einer Authentifizierung zugegriffen werden. Die Webseite nutzt den in Kapitel 5.7 vorgestellten REST Webservice, um auf die Daten aus der Datenbank zuzugreifen.

**Übersicht aller Sensorknoten** Zunächst erhält der Nutzer, wie in Bild 5.10, eine Übersicht über alle Sensorknoten. Die Übersicht enthält dabei alle Sensorknoten, die bereits einmal einen Datensatz in der Datenbank gespeichert haben.

Die Daten werden in einer Tabelle dargestellt. Diese enthält dabei Informationen zur Stations Nummer, Name der Station, der Ort an dem sich das Gerät befindet und ob die Station sich im Energiesparmodus befindet oder nicht. Von dieser Übersichtsseite gelangt der Nutzer mit dem Button *Aktuelle Werte* zu den aktuellen Werten des Sensorknotens.

Stations Nr.	Name	Ort	Aktuelle Werte	Energiesparmodus
100	Station100	Wohnzimmer	Aktuelle Werte	OFF
200	Station200	Küche	Aktuelle Werte	OFF
300	Station300	Schlafzimmer	Aktuelle Werte	OFF
400	Station400	Arbeitszimmer	Aktuelle Werte	OFF
500	Station500	Flur	Aktuelle Werte	OFF
600	Station600	Bad	Aktuelle Werte	OFF
1000	Station1000	Keller	Aktuelle Werte	ON
2000	Station2000	Garage	Aktuelle Werte	ON
3000	Station3000	Vogelhaus	Aktuelle Werte	ON
4000	Station4000	Dach	Aktuelle Werte	ON
5000	Station5000	Rasen	Aktuelle Werte	ON

Abbildung 5.10: Webseite: Übersicht aller Sensorknoten

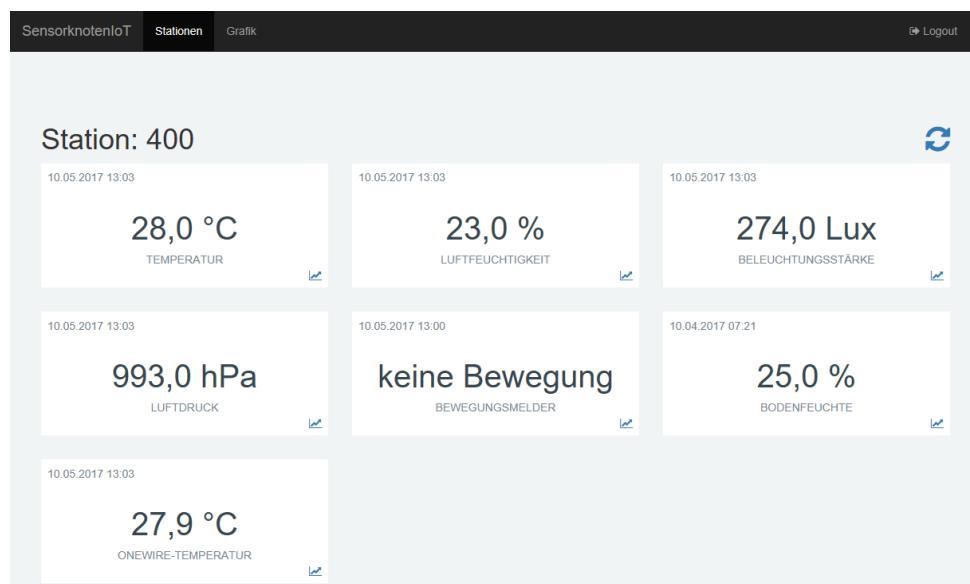


Abbildung 5.11: Webseite: Aktuelle Übersicht eines Sensorknotens

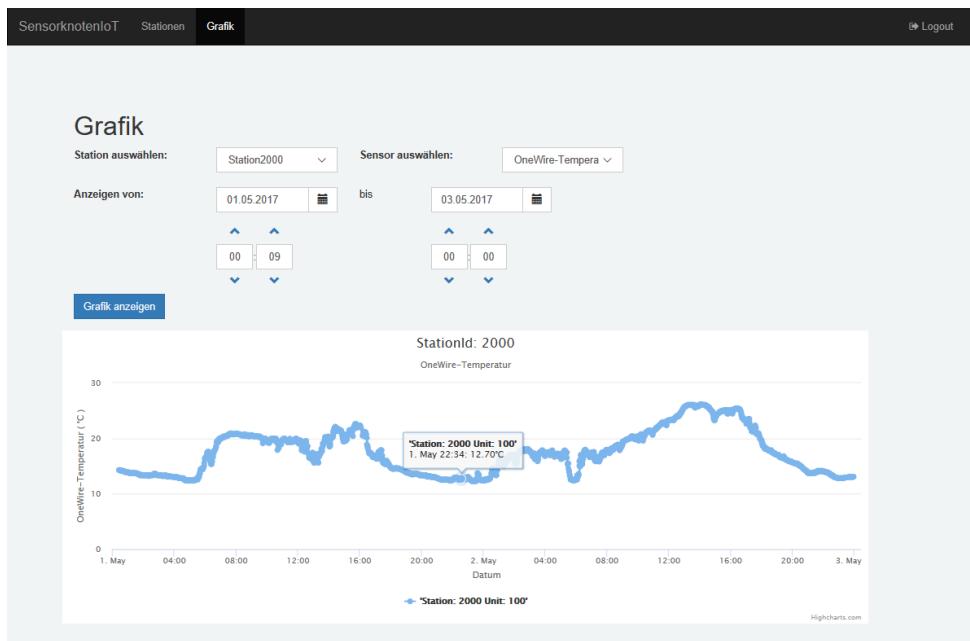


Abbildung 5.12: Webseite: Grafik eines Messwertedatensatz

**Aktuelle Werte eines Sensorknotens** Auf dieser Seite kann sich der Nutzer die aktuellsten Werte des Sensorknotens anzeigen lassen. Die Anzahl der Kacheln wird dynamisch erzeugt, dies hängt davon ab, welche Messwerte der Sensorknoten bereits gesendet hat. In einer Kachel sind die Informationen enthalten, wann der Messwert gemessen wurde, welchen Wert der Messwert hat und welche Einheit dieser hat. Zusätzlich ist noch ein Button auf jeder Kachel bei dem sich der Nutzer zusätzlich die Grafik der vergangene Werte anzeigen lassen kann. Die Webseite aktualisiert die Daten alle 30 Sekunden. Es besteht aber auch die Möglichkeit(Button oben rechts) eine manuelle Aktualisierung anzustoßen.

**Grafik über einen bestimmten Zeitraum** Zur Ansicht älterer Sensorwerte besteht die Möglichkeit sich die gesammelten Werte in einem Kurvendiagramm darzustellen. Der Nutzer muss zunächst die Station und den Sensorwert auswählen den er betrachten möchte. Anschließend muss er angeben in welchem Zeitraum er die Messwerte dargestellt haben möchte. Mit dem Betätigen des „Buttons Grafik anzeigen“ wird die Grafik angezeigt. In der x-Achse befindet sich das Datum, in der y-Achse ausgewählte Wert. Die x- und y-Achse wird entsprechend der gewählten Größen angepasst. Mit der Maus kann über die Kurve gefahren werden und sich mit Hilfe eines Tooltips genauere Informationen anzeigen zu lassen. Für die Grafikanzeige wurde ein JavaScript Plugin verwendet mit dem verschiedene Diagrammtypen dargestellt werden können. Als Framework wurde *Highcharts* verwendet. Momentan werden noch alle Werte angezeigt. Hier könnte bei einer großen Anzahl an Messwerten eine Reduzierung / Ausdünnung vorgenommen werden.

# Kapitel 6

## Lösungsbewertung und Fazit

### 6.1 Vergleich Ist-/Soll-Zustand

In diesem Unterkapitel wird der erreichte Ist-Zustand mit dem geforderten Soll-Zustand verglichen. Hierfür werden die gleichen Gliederungspunkte wie in der Problemstellung (siehe Kapitel 2.2) verwendet und hinsichtlich ihrer Realisierung bewertet.

**Arduino** Das Ziel war es mit Hilfe des Arduinos verschiedene Sensordaten auszuwerten. Dieser Punkt konnte vollständig erfüllt werden. Die Arduinos können von verschiedenen Sensoren die Messwerte bestimmen. Die Sensoren können je nach Anforderungsprofil an das Sensorknotenboard gesteckt werden. Jeder Sensorknoten hat mindestens den DHT22 oder DHT11 installiert, dieser kann die Temperatur und Luftfeuchtigkeit bestimmen. Alle verwendeten Sensoren wurden bereits in Kapitel 5.2 vorgestellt.

Ein weiteres Ziel war die Erstellung eines Mesh-Netzes mit Hilfe eines Funkmoduls. Dies wurde mit dem Funkmodul nRF24l01 (siehe Kapitel 3.4) realisiert. Die normalen Sensorknoten bauen gemeinsam ein Art Mesh-Netz auf, bei dem über mehrere Knoten hinweg Nachrichten geschickt werden können. Die Nachrichten werden per Broadcast an alle umliegenden Stationen weitergeleitet. Der verwendete Algorithmus wird in Kapitel 5.1.2 genauer erklärt.

Als letzter Punkt bei den Arduinos war die Entwicklung eines energiesparenden Sensorknotens. Der entwickelte Sensorknoten verwendet verschiedene Techniken um möglichst wenig Strom zu verbrauchen (siehe Kapitel 5.4). Der energiesparende Sensorknoten benötigt im Sleep-Modus nur noch  $182\mu\text{A}$ . Durch diesen geringen Stromverbrauch kann der Sensorknoten über Monate hinweg autark mit Batterien betrieben werden.

**Raspberry Pi** Der Raspberry sollte als Gateway zum Internet dienen. Dieser Soll-Zustand wurde vollständig erreicht. Der Raspberry Pi empfängt mit demselben Funkmodul wie die Sensorknoten die Nachrichten der Messwerte. Nach dem Empfangen der Nachricht

enkodiert der Raspberry Pi die enthaltene Nachricht und speichert sie in der Datenbank ab. Hierbei werden nur Messwerte von bekannten Sensorknoten und Nachrichten gespeichert die noch nicht in der Datenbank gespeichert wurden. Damit jede Nachricht eindeutig identifizierbar ist wird ein Hash aus der ID des Arduinos, der Nachrichten ID und dem hochgezählten Zeitstempel gebildet (siehe Kapitel 5.5.4).

Das Programm zum Empfang der Messwerte startet sobald der Raspberry fertig gebootet ist (siehe Kapitel 5.5.6).

**Webservice** Das Soll-Zustand dieses Unterpunktes war es die Sensordaten dauerhaft in einer Datenbank zu speichern und diese Daten mit Hilfe eines REST-Webservice wieder zur Verfügung zu stellen. Dieser Soll-Zustand wurde vollständig erfüllt. Die Implementierung wurde mit Hilfe des Microframework Flask in Python vorgenommen. Der REST-Service stellt vier Routen zur Verfügung die ein Ergebnis in der JSON Notation bereitstellen. Es lassen sich alle Stationen ausgeben, die neusten Messwerte einer Station, die Messdaten einer bestimmten Messgröße über einen bestimmten Zeitraum und die Ressource eines bestimmten Messwertes (siehe Kapitel 5.7).

**Datenrepräsentation** Als letzte Anforderung war noch die Präsentation der Daten, hierfür sollte ein Frontend entwickelt werden. Das Frontend wurde mit Hilfe verschiedener Webtechnologien entwickelt. Es wurde eine Webseite entwickelt bei dem sich der Nutzer anmelden kann und nach der Authentifizierung eine Übersicht über alle Sensorknoten erhält. Zusätzlich kann sich der Nutzer die aktuellsten Messwerte und zu jedem Sensor ein Kurvendiagramm über ein vom Benutzer definierten Zeitraum ausgeben lassen. Darstellen (siehe Kapitel 5.8).

## 6.2 Bewertung des Sensorknotens

Dieses Unterkapitel bietet eine Bewertung über das Ergebnis der Studienarbeit.

Die Studienarbeit kann als Erfolg eingestuft werden, da alle geforderten Anforderungen vollständig erfüllt worden sind. Besonders hervorzuheben sind die selbst designeten und entwickelten Sensorknoten. Hier konnte der komplette Entwicklungs- und Produktionsprozess durchlaufen werden. Beginnend mit der Planung und Umsetzung eines Prototypen, bis zur Fertigstellung eines kompletten Sensorknoten der *Plug-and-Play* verwendet werden kann.

Das Endprodukt der Studienarbeit ist ein Sensorknoten auf Basis von Arduinos, ein Gateway auf Basis eines Raspberry Pi's, einer Datenhaltungsschicht in Form einer MySQL Datenbank und einer Datenvisualisierung in Form einer Webseite.

Die Vorteile des Sensorknoten ist die Erweiterbarkeit und einfache Befestigung von Sensoren auf dem Sensorknoten. Die energiesparende Sensorknoten können in verschiedenen Szenarien eingesetzt werden, da sie vollständig mit Batterien betrieben werden können.

Der Raspberry Pi ist mit dem gleichen Funkmodul ausgestattet, dies führt zu einer sehr einfachen Infrastruktur. Sollte Sensoren oder Funkmodule aufgrund von verschiedener Umstände nicht mehr Funktionstüchtig sein können sie einfach ausgetauscht werden.

Die Webseite bietet noch am meisten Erweiterungsmöglichkeiten, da zum Beispiel das hinzufügen neuer Sensorknoten fehlt.

Zusammenfassend lässt sich sagen, dass das Ergebnis das Anforderungsprofil der Studienarbeit übertroffen hat.

## 6.3 Zusammenfassung

Zeitlich gesehen ging das Entwicklungsteam wie folgt vor:

- Verbindung zwischen Raspberry Pi und Arduino herstellen (Kapitel 4.2)
- Bestückung der Arduinos mit den Sensoren auf der eigens designeten Leiterplatine (siehe Kapitel 5.3.3)
- gleichzeitige Realisierung der Verarbeitung der Messdaten auf dem Raspberry Pi (Kapitel 5.5.2)
- Erstellen der Messdatenvisualisierung (Kapitel 5.8) und der Messdatenbereitstellung (Kapitel 5.7)

Bei der Realisierung des Sensorknotens kam es zu einigen Problemen die zu lösen waren. Dazu gehörten unter anderem:

1. Installation weiterer Bibliotheken auf dem Raspberry Pi
2. Die Kodierung der Nachrichten zwischen Arduino und Raspberry Pi
3. Fehlende Widerstände auf der gefertigten Platine
4. Fehler beim gleichzeitigen Zugriff auf die Datenbank
5. Fehlverhalten der Funkmodule beim Wechseln zwischen Senden und Empfangen

Beim ersten Fehler war es notwendig, wie in Kapitel 5.5.2 beschrieben, in eine andere Programmiersprache zu wechseln (Python). Die Installation von weiteren Bibliotheken konnte man dort mittels eines Tools einfach vollziehen. Um Probleme bei der Kodierung zu eliminieren half es, Nachrichten explizit zu kodieren. Dies geschieht zwar zu Ungunsten der Rechenzeit auf dem Arduino, war jedoch effektiv (Kapitel 5.5.3). Weitere Probleme

traten auf bei der Fertigung der Platinen. Aufgrund eines Planungsfehlers fehlte ein Widerstand (siehe Kapitel 5.3.1). Zur Option stand, die Platinen erneut mit den korrigierten Schaltplänen (siehe Anhang) erneut zu fertigen oder den fehlenden Widerstand manuell einzulöten. Aufgrund der relativ langen Lieferzeit entschied man sich dafür, die Widerstände einzulöten. Bei dem gleichzeitigen Zugriff mehrere Programme auf die SQLite-Datenbank zuzugreifen, traten Seiteneffekte auf. Abhilfe schaffte hier der Umstieg auf ein mächtigeres Datenbankmanagementsystem (MySQL, siehe 5.6). In der Bibliothek des RF24-Moduls lauerte ebenfalls ein Fehler. Es war wohl nicht vorgesehen einen Wechseln zwischen Broadcast senden und empfangen zu vollziehen. Die Lösung lag in der Anpassung der Bibliothek, das ist in Kapitel 5.1.2 beschrieben.

## 6.4 Ausblick

Bei der Realisierung dieses Projekts kamen Ideen für die Fortsetzung der Arbeit auf. So wäre eine benutzerfreundliche Möglichkeit weitere Sensorknoten zu registrieren mit Sicherheit ein gefragtes Feature. Man könnte dies mit einer Webapplikation umsetzen. Der vorhanden "Mesh-Algorithmus" ist in zukünftigen Arbeiten ersetzbar durch einen Mesh-Algorithmus wie er in der Literatur definiert ist. Dazu kann man beispielsweise einen entsprechenden RFC implementieren. In diesem Zuge ist es auch möglich eine Routing-Metrik für das neu implementierte Protokoll zu designen und die Datenpakete zu verschlüsseln.

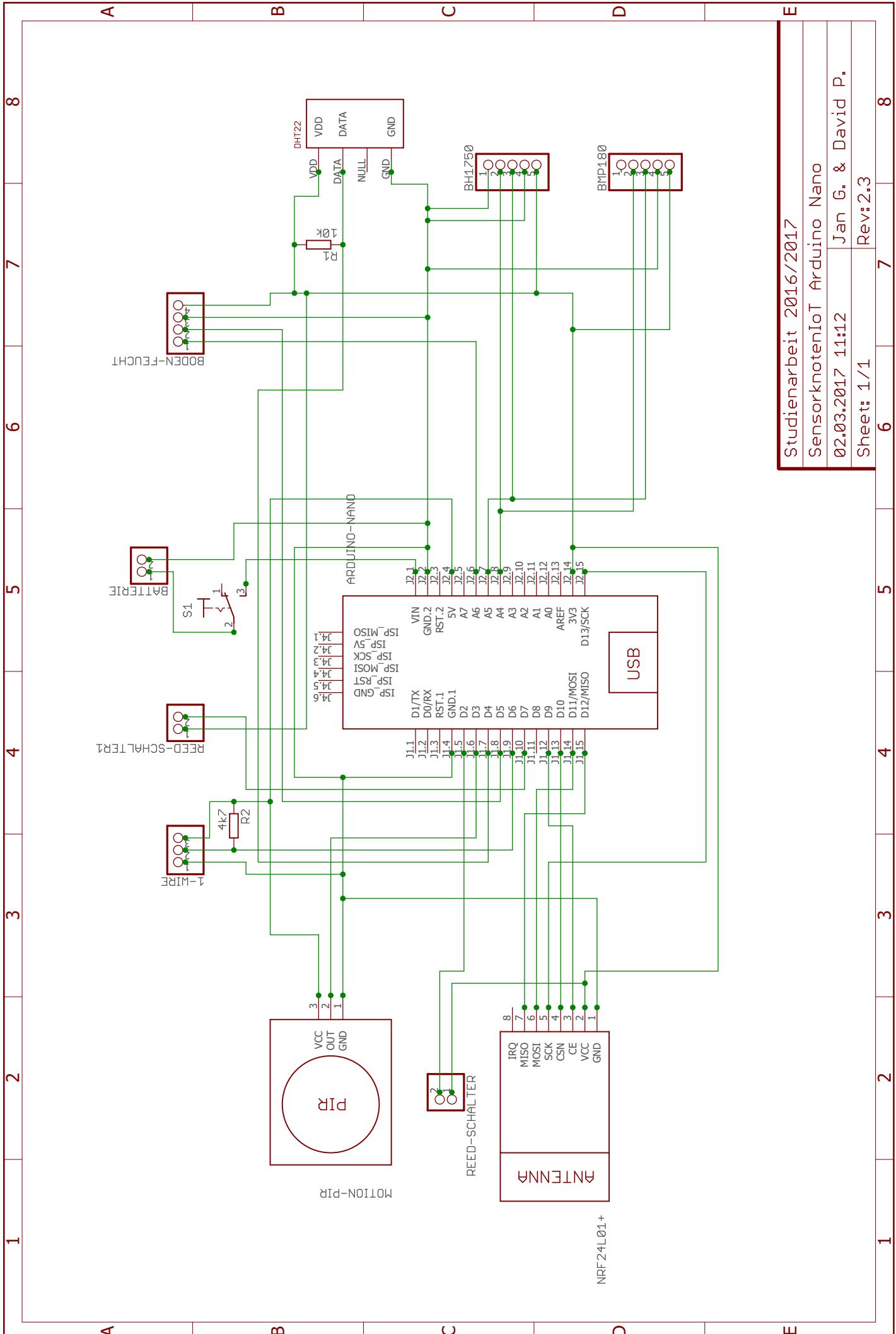
Die gesammelten Sensordaten sind auch als Basis für Dataminig nutzbar.

Ferner kann auch der Gegenspieler zum Sensorknoten, ein Aktorknoten, realisiert werden.

# Literatur

- [BEH12] D. BOSWARTHICK, O. ELLOUMI und O. HERSENT. *M2M Communications: A Systems Approach*. EBL-Schweitzer. Wiley, 2012. ISBN: 9781119994756. URL: <https://books.google.de/books?id=7Xdz3ryx0TIC> [siehe S. 26].
- [Cis16] CISCO. *Solution Overview*. 5. Dez. 2016. URL: [http://www.cisco.com/dam/en/us/products/collateral/switches/mgx-8800-series-switches/solution\\_overview\\_c22-531339.doc/\\_jcr\\_content/renditions/solution\\_overview\\_c22-531339-01.jpg](http://www.cisco.com/dam/en/us/products/collateral/switches/mgx-8800-series-switches/solution_overview_c22-531339.doc/_jcr_content/renditions/solution_overview_c22-531339-01.jpg) [besucht am 10.04.2017] [siehe S. 7, 8].
- [Com08] COMPUTER:CLUB2. *I2C Bus Hintergrundwissen*. 10. Nov. 2008. URL: [http://www.cc-zwei.de/wiki/index.php?title=I2C\\_Bus\\_Hintergrundwissen](http://www.cc-zwei.de/wiki/index.php?title=I2C_Bus_Hintergrundwissen) [besucht am 15.04.2017] [siehe S. 21–23].
- [Con13] D. CONRADS. *Datenkommunikation: Verfahren - Netze - Dienste*. Moderne Kommunikationstechnik. Vieweg+Teubner Verlag, 2013. ISBN: 9783322919731. URL: <https://books.google.de/books?id=9sfJBgAAQBAJ> [siehe S. 33].
- [Dem15] Klaus DEMBOWSKI. *Raspberry Pi-Das technische Handbuch: Konfiguration, Hardware, Applikationserstellung*. Springer-Verlag, 2015 [siehe S. 23, 24, 26].
- [Ele08] Spark fun ELECTRONICS. *nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification v1.0*. 1. März 2008. URL: [https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Plus\\_Preliminary\\_Product\\_Specification\\_v1\\_0.pdf](https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Plus_Preliminary_Product_Specification_v1_0.pdf) [besucht am 10.04.2017] [siehe S. 58].
- [Fie16] Roy FIELDING. *DHTxx Sensors*. 26. Dez. 2016. URL: <https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf> [besucht am 03.05.2017] [siehe S. 47].
- [Fie08] Roy FIELDING. *REST APIs must be hypertext-driven*. 1. Nov. 2008. URL: <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven> [besucht am 11.04.2017] [siehe S. 35].
- [Fir15] FIRSTSENSOR. *Unterschied zwischen Absolut-, Relativ- und Differenzdrucksensoren*. 12. März 2015. URL: <https://www.first-sensor.com/de/produkte/drucksensoren/drucksensoren-und-transmitter/druckarten.html> [besucht am 15.04.2017] [siehe S. 32].
- [Fou97] Python Software FOUNDATION. *Comparing Python to Other Languages*. 1. Jan. 1997. URL: <https://www.python.org/doc/essays/comparisons/> [besucht am 03.05.2017] [siehe S. 25].
- [Fou17] Python Software FOUNDATION. *struct — Interpret strings as packed binary data*. 27. März 2017. URL: <https://docs.python.org/2.7/library/struct>.

- html?highlight=struct#module-struct [besucht am 15.04.2017] [siehe S. 59].
- [Hug16] J.M. HUGHES. *Arduino: A Technical Reference: A Handbook for Technicians, Engineers, and Makers*. O'Reilly Media, 2016. ISBN: 9781491934494. URL: <https://books.google.de/books?id=yYMpDAAAQBAJ> [siehe S. 10].
- [Leh] Herr LEHMANN. „Rechnerarchitetur an der DHBW im Wintersemster 2015“. submitted [siehe S. 23, 24].
- [Mic17a] MICROCONTROLLER.NET. *Funktionsweise eines passiven Infrarot-Bewegungsmelders*. 15. Apr. 2017. URL: [https://www.didaktik.physik.uni-muenchen.de/materialien/sensorik/temp\\_infrarot/bewegungsmelder/pid\\_info1.pdf](https://www.didaktik.physik.uni-muenchen.de/materialien/sensorik/temp_infrarot/bewegungsmelder/pid_info1.pdf) [besucht am 15.04.2017] [siehe S. 31, 32].
- [Mic17b] MICROCONTROLLER.NET. *Lichtsensor / Helligkeitssensor*. 15. Apr. 2017. URL: [https://www.mikrocontroller.net/articles/Lichtsensor/\\_Helligkeitssensor](https://www.mikrocontroller.net/articles/Lichtsensor/_Helligkeitssensor) [besucht am 15.04.2017] [siehe S. 31].
- [Mic17c] MICROCONTROLLER.NET. *Microcontroller.net Temperatursensor*. 15. Apr. 2017. URL: <https://www.mikrocontroller.net/articles/Temperatursensor> [besucht am 15.04.2017] [siehe S. 30].
- [mik17] MIKROCONTROLLER.NET. *Temperatursensoren - Analoge Temperatursensoren - PT100*. 10. Mai 2017. URL: <https://www.mikrocontroller.net/articles/Temperatursensor> [besucht am 10.05.2017] [siehe S. 30].
- [Ric] Prof. Dr. RICHTER. „Betriebssysteme an der DHBW im Sommersemester 2016“. submitted [siehe S. 20].
- [Sem07] Nordic SEMICONDUCTOR. *nRF24L01 product specification*. 2007 [siehe S. 27].
- [Sen13] Bosch SENSORTEC. „Data sheet BMP180 Digital pressure sensor“. In: [2013]. URL: <https://cdn.sparkfun.com/datasheets/Sensors/Pressure/BMP180.pdf> [besucht am 03.05.2017] [siehe S. 48].
- [Str] Prof Dr. Marcus STRAND. „Prozessautomatisierung 1 an der DHBW im Wintersemester 2016“. submitted [siehe S. 28, 29].
- [Til11] Stefan TILKOV. *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*. dpunkt. verlag, 2011 [siehe S. 34].
- [VGA12] KP VIJAYAKUMAR, P GANESHKUMAR und M ANANDARAJ. „Review on routing algorithms in wireless mesh networks“. In: *International Journal of Computer Science and Telecommunications* 3.5 [2012], S. 87–92 [siehe S. 34].
- [Wer09] Prof. Matthias WERNER. *ARM-Architektur*. 1. Okt. 2009. URL: <https://osg.informatik.tu-chemnitz.de/lehre/lab/slides/arm-2009.pdf> [besucht am 01.05.2017] [siehe S. 17–19].
- [Wik] WIKIPEDIA. *Wikipedia*. URL: [https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Fotoelektrischer\\_Effekt.svg/2000px-Fotoelektrischer\\_Effekt.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Fotoelektrischer_Effekt.svg/2000px-Fotoelektrischer_Effekt.svg.png) [besucht am 15.04.2017] [siehe S. 31].



Studienarbeit 2016/2017

SensorknotenLoT Arduino Nano

02.03.2017 11:12 Jan G. & David P.

Sheet: 1/1 Rev: 2.3

6 7 8

5

4

3

2

1

A

B

C

D

E

