



**Hochschule für Technik  
und Wirtschaft Berlin**

*University of Applied Sciences*

# **Positionsbestimmung drathafter mobiler eingebetteter Systeme mittels Time Difference of Arrival**

## **Abschlussarbeit**

zur Erlangung des akademischen Grades

**Bachelor of Engineering**

an der

Hochschule für Technik und Wirtschaft Berlin  
Fachbereich 1: Ingenieurwissenschaften - Energie und Information  
Studiengang Computer Engineering

**Erstprüfer** Prof. Dr.-Ing. habil. Carsten Gremzow

**Zweitprüfer** Prof. Dr. rer. nat. Sebastian Bauer

Eingereicht von: Oliver Koepf

Matrikelnummer: s0559122

Abgabe: XX.XX.2019

# Inhaltsverzeichnis

1	Einleitung . . . . .	1
1.1	Aufbau der Arbeit . . . . .	1
2	Grundlagen . . . . .	2
2.1	Verwendete Hardware . . . . .	2
2.2	Verwendete Software . . . . .	8
2.3	Time Difference of Arrival - TDOA . . . . .	9
2.4	User Datagram Protocol - UDP . . . . .	9
2.5	Netzwerk-Socket . . . . .	10
2.6	Indoor/Outdoor Positionsbestimmung . . . . .	10
2.7	Zeitsynchronisation . . . . .	10
2.8	Mathematischer Hintergrund - Positionsbestimmung . . . . .	11
3	Entwurf . . . . .	15
3.1	Hardware . . . . .	15
3.2	Struktur . . . . .	15
4	Implementierung . . . . .	17
4.1	SparkFun Sound Detector . . . . .	17
4.2	Modul A . . . . .	20
4.3	Modul B . . . . .	21
4.4	Modul C . . . . .	22
4.5	Modul D . . . . .	22
4.6	Systemzeit . . . . .	24
4.7	Zeitsynchronisation . . . . .	25
4.8	433 Mhz Funk Transmitter-Receiver . . . . .	27
4.9	Software . . . . .	28
5	Auswertung . . . . .	31
5.1	Hardware . . . . .	31
5.2	Messergebnis . . . . .	31
5.3	Software . . . . .	31
6	Unit Test . . . . .	32
6.1	Quadratische Gleichung . . . . .	32
6.2	Abstand zweier Punkte . . . . .	32
6.3	Positionsbestimmung . . . . .	33
7	Gelöste Probleme . . . . .	35
7.1	Kommunikation Master – Slave . . . . .	35

7.2	Genauigkeit Zeitsynchronisation . . . . .	35
7.3	Messgenauigkeit . . . . .	35
8	Ausblick . . . . .	36
	Eidesstattliche Erklärung . . . . .	38
	Literaturverzeichnis . . . . .	39
9	Anhang . . . . .	41
1	Mathematische Herleitung . . . . .	41
2	Steuercodes . . . . .	43

# Abbildungsverzeichnis

1	Vorderseite des SAM R21 Xplained Pro Evaluation KitBoard . . . . .	2
2	Vorderseite des HC-SR04 Sensor . . . . .	3
3	Funktionsweise von Ultraschallsensoren . . . . .	4
4	Sparkfun Sound Detector . . . . .	4
5	Pegelwechsel des GATE-Ausgang . . . . .	5
6	433 Mhz Funk Transmitter-Receiver . . . . .	6
7	Ansteuerung des Lautsprechers . . . . .	8
8	Vergleich von RIOT mit drei anderen Betriebssystemen . . . . .	9
9	Nachrichtenaustausch in PTP . . . . .	11
10	Prinzip der Positionsbestimmung . . . . .	12
11	Bereich indem sich der Zielknoten befinden muss . . . . .	13
12	Versuchsaufbau, unterteilt in Module . . . . .	15
13	Verdrahtungsplan Master . . . . .	16
14	Verdrahtungsplan Slave . . . . .	16
15	Versuchsaufbau als Blockschaltbild . . . . .	17
16	Versuchsaufbau . . . . .	17
17	Verzögerung der Schallausbreitung . . . . .	18
18	Messwerte von $t_{prop}$ bei hundert Messungen . . . . .	26
19	DATA-Pin des 433 Mhz Funk Transmitter-Receiver Empfänger . . . . .	28
20	Programmablaufplan der Software . . . . .	29
21	Zwei Möglichkeiten der Positiosbestimmung . . . . .	33

# Tabellenverzeichnis

1	Messwerte bei 1 m Entfernung . . . . .	19
2	Messwerte bei 2 m Entfernung . . . . .	19
3	Messwerte bei 3 m Entfernung . . . . .	20
4	ISR-Verzögerung Mikrofon-SAM R21 Xplained Pro Evaluation Kit bei unterschiedlicher Kabellänge . . . . .	21
5	Verzögerung der Slave ISR bei unterschiedlicher Kabellänge . . . . .	22
6	Abweichung zwischen Mikrofon und Lautsprecher bei 1 m . . . . .	23
7	Abweichung zwischen Mikrofon und Lautsprecher bei 2 m . . . . .	23
8	Abweichung zwischen Mikrofon und Lautsprecher bei 3 m . . . . .	24
9	Messwerte für die Funktion <i>getSystemTime()</i> . . . . .	25
10	Messwerte für das Setzen und Löschen eines GPIO . . . . .	25
11	Eingaben und Ausgaben der pq-Funktion . . . . .	32
12	Alle Steuercodes . . . . .	43

## **Abkürzungsverzeichnis**

<b>MCU</b>	Microcontroller Unit
<b>FPU</b>	Floating Point Unit
<b>RISC</b>	Reduced Instruction Set Computer
<b>ARM</b>	Advanced RISC Machine
<i>I<sup>2</sup>C</i>	Inter-Integrated Circuit
<b>SPI</b>	Serial Peripheral Interface
<b>GPIO</b>	General-purpose input/output
<b>MAC</b>	Media-Access-Control
<b>IPv4</b>	Internet Protocol Version 4
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>UDP</b>	User Datagram Protocol
<b>TCP</b>	Transmission Control Protocol

## 1 Einleitung

Diese Arbeit hat das Ziel, eine Positionsbestimmung auf Basis von Schalllaufzeitmessungen zu entwickeln und aufzubauen. Dies soll die Grundlage sein für eine fehlerfreie Fahrt in Gebäuden von dem Hausinternen CE-Car [1]. Die Positionsgenauigkeit spielt dabei eine untergeordnete Rolle. Positionierungssysteme gibt es viele, allerdings sind diese nicht immer frei verfügbar, kostenintensiv und oft nur für den Outdoor Bereich entwickelt worden [2]. Ziel ist es, ein mobiles eingebettetes System für den Indoorbereich zu entwickeln, mit dem Fokus auf geringe Kosten, sodass Modellautos, Drohnen oder Roboter damit ausgestattet werden können. Darüber hinaus muss das System ohne externe Dienste oder Netzanbindung funktionstüchtig sein. Zudem muss das System einfach erweiterbar sein, um der zukünftigen Entwicklung Stand zu halten. Die Validierung, erfolgt durch eine prototypische Anwendung.

### 1.1 Aufbau der Arbeit

Die Bachelorarbeit ist folgendermaßen gegliedert: Zuerst werden in Kapitel 2 die nötigen theoretischen Grundlagen erläutert, sowie die verwendete Hardware und Software beschrieben. Darauf aufbauend kommt Kapitel 3, indem die Umsetzung beschrieben wird. Im Anschluss wird das System evaluiert. Zum Schluss werden Probleme aufgezeigt, sowie ein Ausblick für mögliche Erweiterungen des Systems gegeben.

## 2 Grundlagen

### 2.1 Verwendete Hardware

#### Controller

Diese Arbeit verwendet das von Atmel entwickelte Board "SAM R21 Xplained Pro Evaluation Kit" [3]. Das Board besitzt neben einem ARM Cortex-M0+ Prozessor noch ein Energiesparenden ISM-Bandsender-Empfänger auf einem Chip. Dieser Sender nutzt die Frequenz 2.4GHz zur Datenübertragung. Mit den vorhandenen GPIOs kann das Board verschiedene Sensoren und Aktoren angesteuert werden. Das alles zusammen macht das SAM R21 Xplained Pro Evaluation Kit zu einem universell einsetzbaren Board für die drahtlose Kommunikation, speziell geeignet für den Embedded Bereich. Die folgende Abbildung 1 zeigt das SAM R21 Xplained Pro Evaluation KitBoard von der Vorderseite.

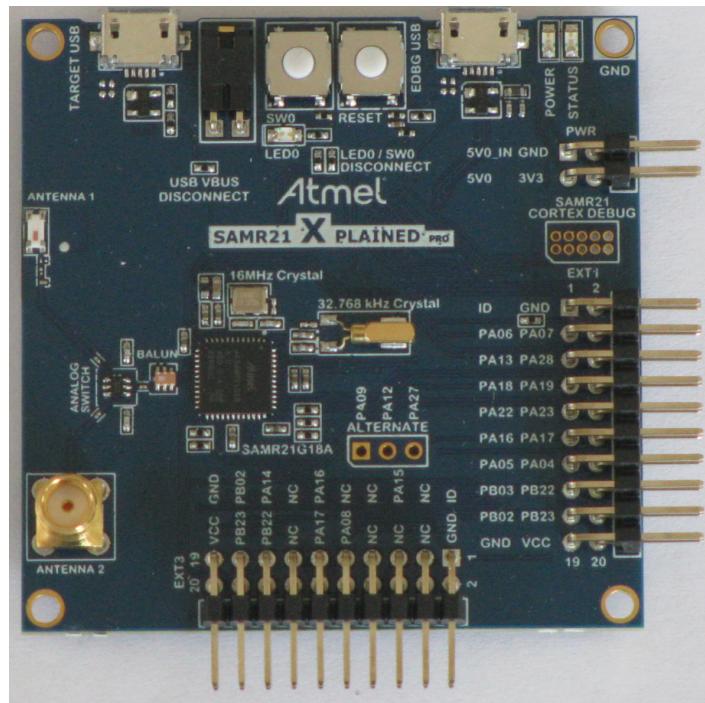


Abbildung 1: Vorderseite des SAM R21 Xplained Pro Evaluation KitBoard

#### Sensoren

Für die Laufzeitmessung können verschiedene Sensoren verwendet werden. Folgend ein Vergleich zwischen dem Ultraschallsensor HC-SR04 und dem SparkFun Sound Detector Sensor. Dieser Vergleich wird durchgeführt, weil beide Sensoren in der Entwicklung zum Einsatz kamen, aber nur einer die nötigen Performance für die Positionsbestimmung leistet.

### Ultraschallsensor

Für die Positionsbestimmung ist der Ultraschallsensor HC-SR04 (Abbildung 2) zum Einsatz gekommen [4]. Ich habe mich für den HC-SR04 Sensor aufgrund seiner weiten Verbreitung im Arduino/ Raspberry Pi Bereich, seiner kompakten Bauweise, der geringen Kosten und der einfachen Ansteuerung entschieden. Ultraschallsensoren werden verwendet, um Distanzen zu einem Gegenstand zu ermitteln. Dabei wird ein Ultraschallsignal ausgesendet und von dem Gegenstand zurück reflektiert und wieder empfangen. Über die Zeitdifferenz zwischen Aussenden und Empfangen des Ultraschallsignals kann die Distanz berechnet werden. Aus der Abbildung 3 wird das Prinzip deutlich. Der HC-SR04 hat eine Reichweite von 3 cm bis 4 m. Der Vorteil von diesem Ultraschallsensor ist, dass es ohne weitere Sensoren auskommt. Allerdings weiß der Ultraschallsensor einige Nachteile auf, weswegen er von einem leistungsfähigeren Schallmikrofon (SparkFun Sound Detector) abgelöst worden ist. Ultraschallsensoren funktionieren nur, wenn sie direkten Sichtkontakt zum Objekt haben. Dies ist nicht immer gegeben. Des weiteren können nur Objekte, die in dem Sendekegel des Ultraschallsensors liegen, detektiert werden. Der Sendekegel für den HC-SR04 liegt bei  $15^\circ$ . Aufgrund dieser Nachteile habe ich mich in der Endanwendung gegen diesen Sensor entschieden, denn die Positionsbestimmung wird dadurch stark beeinträchtigt.

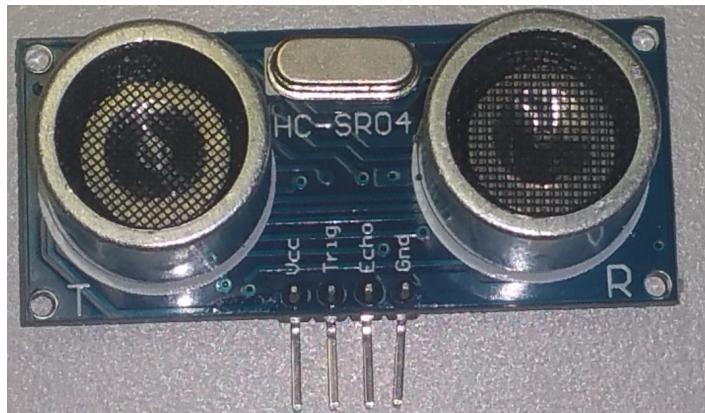


Abbildung 2: Vorderseite des HC-SR04 Sensor

### Sound Detector

Im Verlauf der Umsetzung für die Positionsbestimmung hat sich gezeigt, dass sich der Ultraschallsensor HC-SR04 nicht eignet. Abhilfe schafft der SparkFun Sound Detector. Abbildung 4 zeigt den SparkFun Sound Detector. Der Sensor kann hörbaren Schall detektieren. Darüber hinaus kann die Empfindlichkeit durch das Einlöten eines Widerstandes erhöht oder verringert werden. Der Vorteil gegenüber dem Ultraschallsensor ist, dass der Schall sich kugelförmig ausbreitet. Somit muss der Sensor nicht auf eine Richtung ausgerichtet werden. Allerdings stellen Hindernisse für einen hörbaren Schall ein Problem da. Deswegen sollten keine oder nur wenige Hindernisse auf der Ebene vorhanden sein. Die Reichweite kann über eine Amplitudenveränderung des Tongebbers verändert werden. Für die Ansteuerung des SparkFun Sound Detector

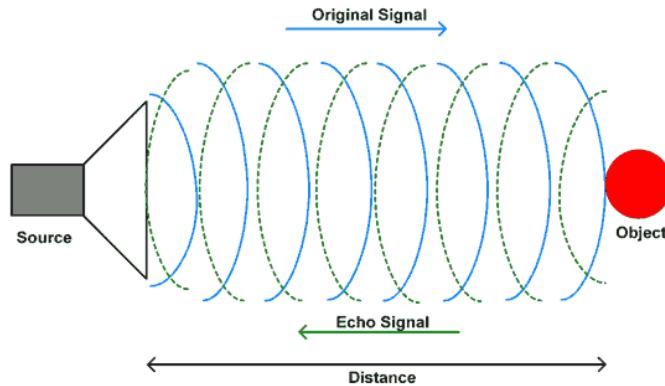


Abbildung 3: Funktionsweise von Ultraschallsensoren

kann der digitale Ausgang GATE verwendet werden. Sobald ein Signal die eingestellte Schallschwelle überschreitet, wird der GATE Ausgang des Sound Detectors auf HIGH gesetzt. Wird die Schallschwelle unterschritten, fällt der GATE Ausgang zurück auf LOW. Es gibt noch weitere Ausgänge, allerdings werden diese nicht verwendet [5]. Die Abbildung 5 zeigt ein Überschreiten des Schallpegels.



Abbildung 4: Sparkfun Sound Detector

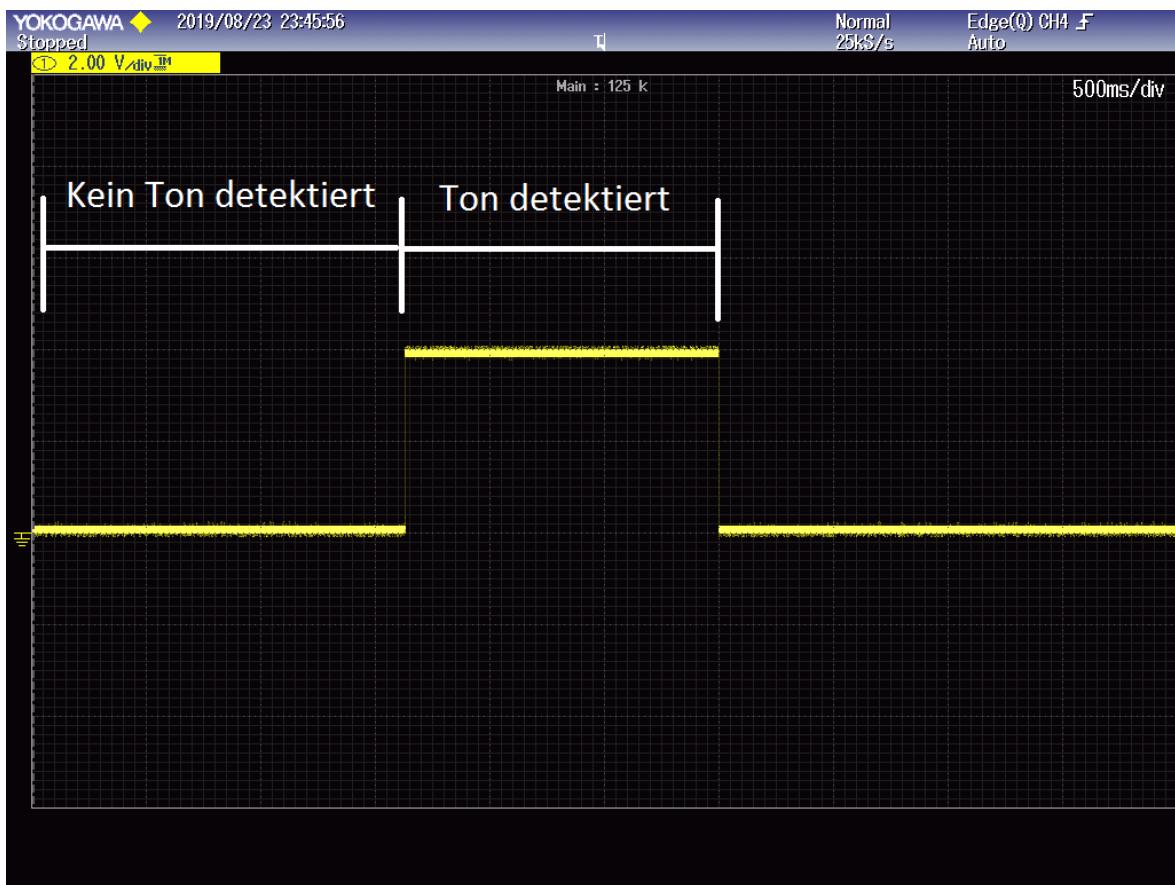
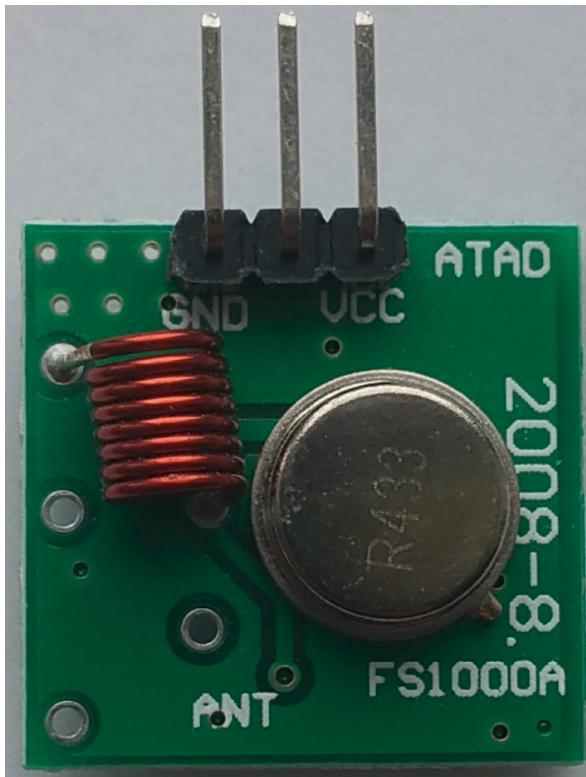


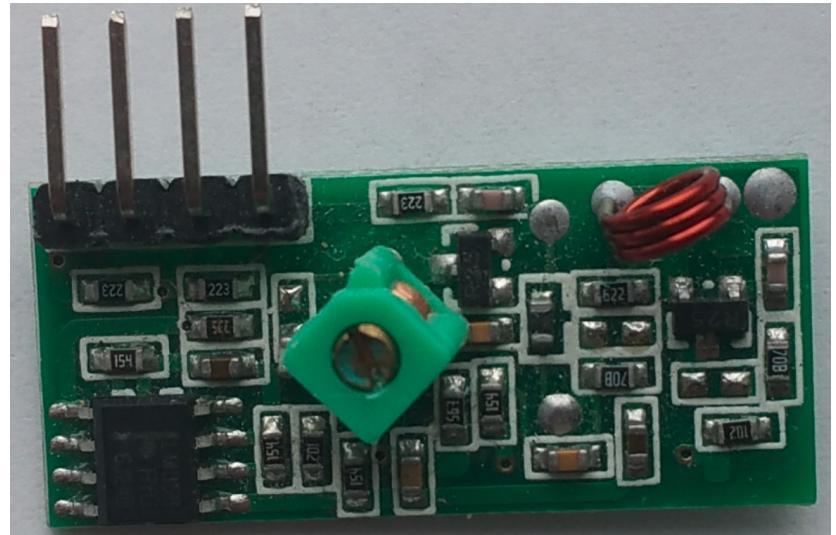
Abbildung 5: Pegelwechsel des GATE-Ausgang

### 433 Mhz Funk Transmitter-Receiver

Dieser Funksender sendet auf der 433 MHz Frequenz [6]. Er ist im Arduino Umfeld weit verbreitet und aufgrund seiner schlichten Ansteuerung einfach zu bedienen. Der Funksender besitzt keine Fehlerkorrektur für verlorene Nachrichten, sowie keine Signalkodierung. Deswegen eignet er sich gut für geringe Bandbreiten. Meine Arbeit verwendet diesen Sensor, um ein Startsignal zu senden. Im Laufe der Arbeit hat sich allerdings gezeigt, dass diese Variante nicht fehlerfrei funktioniert. Es folgt ein Foto 6 des Sensors.



(a) Funksender



(b) Empfänger

Abbildung 6: 433 Mhz Funk Transmitter-Receiver

Es folgt eine Auflistung der Anschlüsse des Funksender von links.

<b>GND</b>	Masse
<b>DATA</b>	Payload
<b>VCC</b>	Versorgungsspannung

Der Empfänger besitzt vier Anschlüsse. Diese sind wie folgt:

<b>GND</b>	Masse
<b>DATA</b>	Payload
<b>DATA</b>	Payload
<b>VCC</b>	Versorgungsspannung

Mit einem Pegelwechsel des Funksender bei dem Anschluss DATA kann eine Nachricht übertragen werden. Der Empfänger gibt die empfangenen Daten über die DATA Anschlüsse wieder aus. Mithilfe eines Schmitt-Triggers kann dieses Signal geglättet werden.

### **Lautsprecher**

Als Tongeber wird ein handelsüberlicher aktiver Lautsprecher verwendet [7]. Im Vergleich zu passiven Lautsprechern, muss die Frequenz nicht selbst erzeugt werden. Mit dem Anlegen der Betriebsspannung wird die Membran in Schwingung versetzt. Die Lautstärke wird über die Betriebsspannung reguliert. Da der Lautsprecher mit 24 V betrieben wird, benötigt man eine externe Schaltung. Diese besteht aus einem n-dotierten Mosfet und dem Lautsprecher. Abbildung 7 zeigt die Schaltung. Wenn an dem GATE Eingang des Mosfet eine Spannung von 2 - 4 V anliegt, schaltet der Mosfet durch, und der Lautsprecher wird mit Strom versorgt.

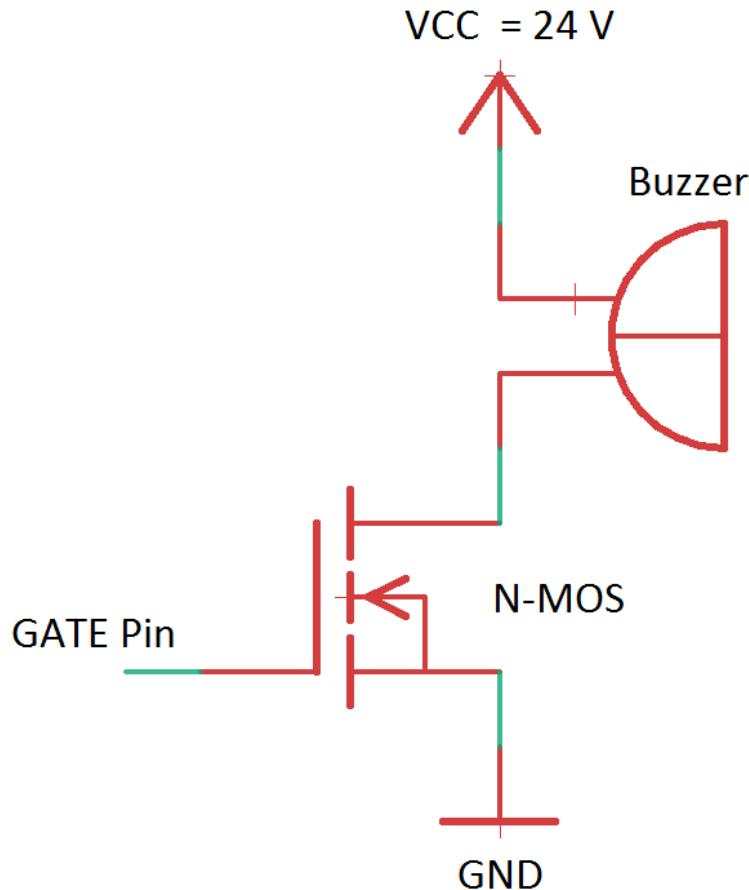


Abbildung 7: Ansteuerung des Lautsprechers

## 2.2 Verwendete Software

### RIOT OS

Um das SAM R21 Xplained Pro Evaluation Kit in Betrieb zu nehmen, kommt das echtzeitfähige Betriebssystem RIOT zum Einsatz. RIOT steht für: "The friendly Operating System for the Internet of Things" und wurde von der Freien Universität Berlin, der INRIA (Institut national de recherche en informatique et en automatique, Le Chesnay, Frankreich) und der Hochschule für Angewandte Wissenschaften, Hamburg entwickelt. Es entstand aus dem "FeuerWhere" Projekt, bei dem Feuerwehrleute im Einsatz überwacht werden sollten. 2010 kam es zu einer Abspaltung des Projekts - das war die Geburtsstunde von RIOT. RIOT ist ein Betriebssystem für Internet of Things Anwendungen. RIOT hat den Fokus auf drahtlose Sensornetzwerke gelegt. Protokolle wie 6LoWPAN, RPL, UDP und TCP wurden mit der Zeit implementiert. Des weiteren unterstützt RIOT echtes Multithreading. Vergleicht man RIOT mit anderen Embedded Betriebssystemen, erkennt man, dass RIOT, die steigenden Anforderungen an Embedded Betriebssystemen unterstützt. Weiterhin ist RIOT mit dem verwendeten Board SAM R21 Xplained Pro Evaluation Kit kompatibel, weswegen es sich perfekt als Betriebssystem für diese Arbeit eignet [8]. Die folgende

Abbildung 8 vergleicht RIOT mit drei anderen Betriebssystemen.

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	○	✗	○	✓	○	○
Tiny OS	<1kB	<4kB	✗	✗	○	✓	✗	✗
Linux	~1MB	~1MB	✓	✓	✓	✗	○	○
RIOT	~1.5kB	~5kB	✓	✓	✓	✓	✓	✓

TABLE I

KEY CHARACTERISTICS OF CONTIKI, TINYOS, LINUX, AND RIOT. (✓) FULL SUPPORT, (○) PARTIAL SUPPORT, (✗) NO SUPPORT. THE TABLE COMPARES THE OS IN MINIMUM REQUIREMENTS IN TERMS OF RAM AND ROM USAGE FOR A BASIC APPLICATION, SUPPORT FOR PROGRAMMING LANGUAGES, MULTI-THREADING, MCUs WITHOUT MEMORY MANAGEMENT UNIT (MMU), MODULARITY, AND REAL-TIME BEHAVIOR.

Abbildung 8: Vergleich von RIOT mit drei anderen Betriebssystemen

Source: <https://www.riot-os.org/docs/riot-infocom2013-abstract.pdf>

### 2.3 Time Difference of Arrival - TDOA

Das TDOA ist ein Verfahren zur Laufzeitmessung, welches den Laufzeitunterschied eines Zeitstempels misst. Damit können Endgeräte über mindestens drei Basisstationen geortet werden. Für die Laufzeitmessung kann jede Art von Signal verwendet werden [9].

### 2.4 User Datagram Protocol - UDP

UDP ist ein Netzwerkprotokoll, welches im OSI-Modell in Schicht vier zu finden ist. UDP ist 1977 für die Sprachübertragung in Rechnernetzwerken entwickelt worden. Es ist einfach aufgebaut. Es arbeitet verbindungslos, d.h. der Sender bekommt keine automatische Meldung ob das gesendete Paket angekommen ist. Im Gegensatz dazu arbeitet TCP verbindungsorientiert. TCP steht für: Transmission Control Protocol. Es ist wie UDP ein Netzwerkprotokoll zur Datenübertragung. Des weiteren hat UDP den Vorteil, dass vorher keine Verbindung mit dem Empfänger aufgebaut werden muss. Das ist besonders im IoT- Bereich wichtig, denn dort sind die Systeme meistens batteriebetrieben. Allerdings kann nicht ausgeschlossen werden, dass die Daten unverfälscht beim Empfänger ankommen.

Ein UDP Paket wird unterteilt in ein Headerfeld und ein Datenfeld. Die Größe des Headers sind immer 8 Byte. Es folgt eine Auflistung der Komponenten aus denen das UDP Paket besteht [10]:

<b>Quellport</b>	Port des Quellrechners
<b>Zielport</b>	Port des Zielrechners
<b>Länge</b>	Gibt die Länge des Datensegmentes in Byte an
<b>Prüfsumme</b>	Ein Wert, der aus dem Datensegment errechnet wird, um Manipulationen zu erkennen
<b>Daten</b>	Nutzlast

## 2.5 Netzwerk-Socket

Ein Socket ist eine Schnittstelle, die vom Betriebssystem bereitgestellt wird. Ein Socket verbindet einen Kommunikationsendpunkt mit dem Betriebssystem. Über ein Socket kann ein Programm, welches eine Datei ist, auf den Kommunikationsendpunkt zugreifen. Wenn Netzwerddaten empfangen werden, liegen diese zur Abholung im Socket bereit. Sockets können bidirektional<sup>1</sup> betrieben werden. Sockets sind immer an einen Port gebunden. Dadurch weiß das Betriebssystem, welche Pakete zu welchem Socket gehören. Im Gegenzug gibt es noch die RAW-Sockets. Diese erlauben den Zugriff auf das ganze Paket. Dort werden keine Daten vorher aus dem Paket gefiltert [11].

## 2.6 Indoor/Outdoor Positionsbestimmung

Indoor-Positionsbestimmungssysteme sind nicht so weit verbreitet wie Outdoor-Systeme. Für Outdoor-positionsbestimmungen wird häufig das globale Navigationssatellitensystem GPS (Global Positioning System) verwendet. Es kann aber auch der Mobilfunk verwendet werden. Für die Positionsbestimmung im Indoorbereich existieren mehrere Verfahren. Dazu zählen Messungen des Einfalls winkels, Signalstärkemessungen oder Laufzeitmessungen. Da wir bereits entschieden haben, dass wir Schall für die Positionsbestimmung nehmen, verwenden wir das Verfahren der Laufzeitmessung. Dabei wird die Zeitdifferenz zwischen Sende- und Empfangszeit ermittelt, sodass man die Signallaufzeit erhält [12]. Zusammen mit der Ausbreitungsgeschwindigkeit von Schall ( $343,2 \frac{m}{s}$  bei  $20^{\circ}\text{C}$ ) kann die Distanz über die folgende Formel 2.1 berechnet werden:

$$\text{Distanz } [m] = (\text{Empfangszeit} - \text{Sendezzeit}) [\text{sec}] \cdot \text{Ausbreitungsgeschwindigkeit } [m/\text{sec}] \quad (2.1)$$

## 2.7 Zeitsynchronisation

Für eine Laufzeitmessung ist es wichtig, dass Sender und Empfänger die gleiche Uhrzeit haben. Damit eine hohe Genauigkeit bei der Laufzeitmessung erreicht wird kann, muss der Zielknoten (Master) wissen, zu welchem Zeitpunkt der Accesspoint (Slave) den Schall, aussendet. Ist dies nicht gewährleistet, muss der Master raten. Damit nicht geraten werden muss, ist es notwendig, dass beide Knoten die gleiche Zeitbasis haben. Für dieses Problem wird eine Zeitsynchronisation für drahtlose Netzwerke verwendet, das Precision Time Protocol (PTP). PTP ist für drahtlose Sensornetzwerke entwickelt worden - es gibt hierbei keine Hierarchien wie beim Network Time Protocol. Der Vorteil liegt darin, dass PTP nicht mit jeder Hierarchie Genauigkeit verliert wie NTP. Es spezialisiert sich auf kleine Netzwerke ohne Hierarchien. Eine Genauigkeit im Nanosekundenbereich kann erreicht werden, wenn die aktuelle Systemzeit kurz vor dem Absenden des Pakets hinzugefügt wird. Desto geringer die Verzögerung zwischen dem Funktionsaufruf `getSystemTime()` und dem Absenden des Pakets, desto genauer wird die Zeitsynchronisation.

---

<sup>1</sup>Datenübertragung in beide Richtungen

Die folgende Abbildung 9 zeigt, welchen Nachrichtenaustausch für die Zeitsynchronisation nötig ist [13]. Der Master ist der Sender und der Slave ist der Empfänger.

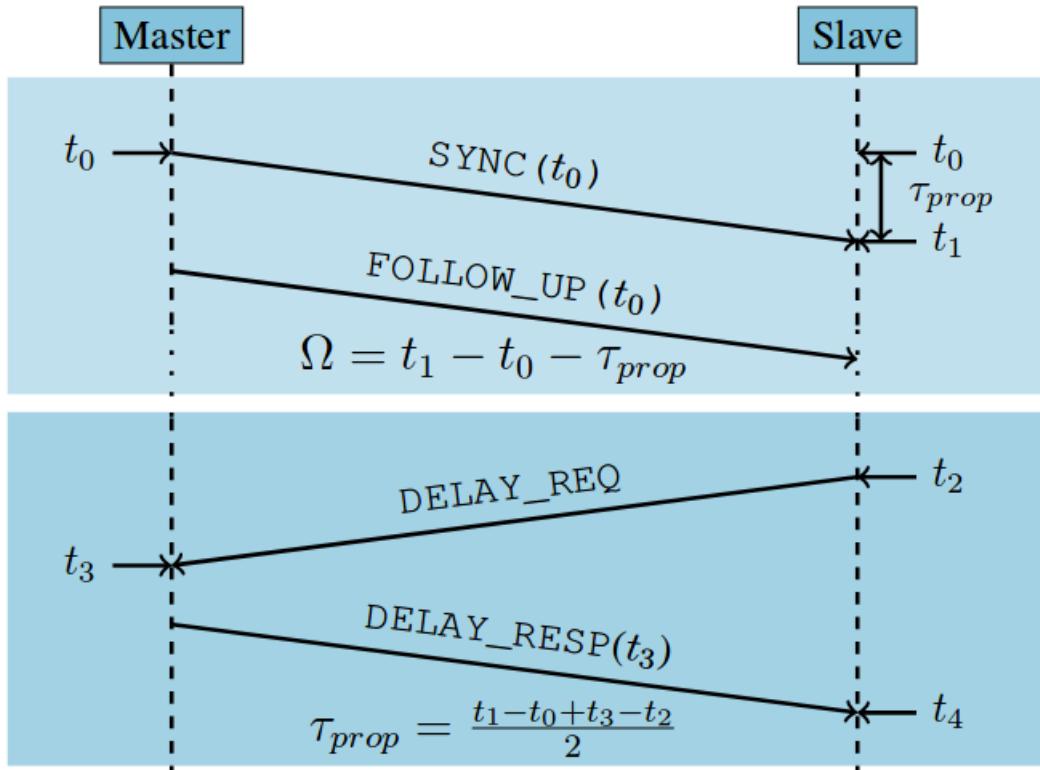


Abbildung 9: Nachrichtenaustausch in PTP

Source: [https://www.ibr.cs.tu-bs.de/oa/vonzengen\\_ICIT2017.pdf](https://www.ibr.cs.tu-bs.de/oa/vonzengen_ICIT2017.pdf)

Zuerst sendet der Master eine SYNC Nachricht mit seinem Zeitstempel  $t_0$  an den Slave. Aufgrund der Verarbeitungszeit, der Laufzeitverzögerung und der Zugriffszeit, ist der Zeitstempel  $t_0$  für den Slave nicht präzise. Mit einer FOLLOW\_UP Nachricht werden diese Probleme gemildert. Dabei enthält die FOLLOW\_UP Nachricht den Zeitstempel  $t_0$ . Nachdem beide Nachrichten beim Slave angekommen sind, antwortet der Slave mit einer DELAY\_REQ Nachricht - ohne Inhalt. Der Master sendet darauf hin ein DELAY RESP mit dem Zeitstempel  $t_3$ . Nun kann mit den folgenden zwei Formeln 2.2 die Zeitdifferenz berechnet werden. Dabei entspricht  $\tau_{prop}$  den Zeitunterschied zwischen beiden Systemen und Omega ( $\Omega$ ) ist der Offset zwischen Master und Slave.

$$\begin{aligned}\tau_{prop} &= \frac{t_1 - t_0 + t_3 - t_2}{2} \\ \Omega &= t_1 - t_0 - \tau_{prop}\end{aligned}\tag{2.2}$$

## 2.8 Mathematischer Hintergrund - Positionsbestimmung

Damit auf einer Ebene die Position bestimmt werden kann, muss der Master von mindestens drei Slaves die Distanz messen. Da sich der Schall auf einer Ebene kreisförmig ausbreitet, vereinfacht

sich das Problem auf den Schnittpunkt von drei Kreisen (Abbildung 10). Des weiteren müssen die Koordinaten der Slaves bekannt sein.

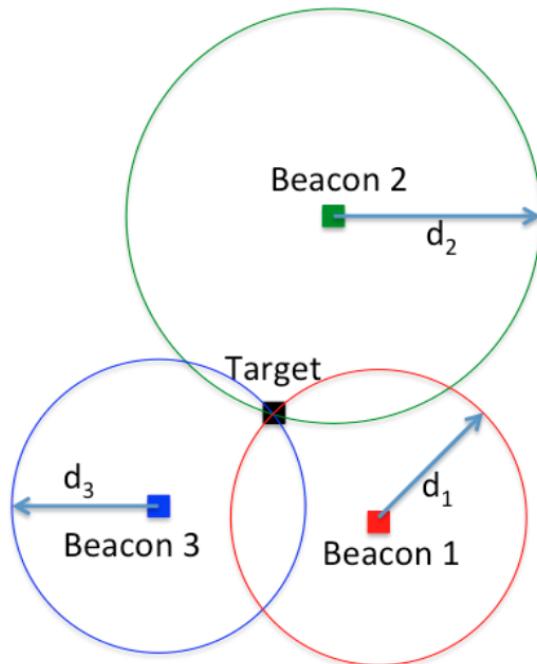


Abbildung 10: Prinzip der Positionsbestimmung

Source: [https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick\\_OKeefe\\_F1.pdf](https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick_OKeefe_F1.pdf)

Aufgrund von Schwankungen kann es sein, dass es keinen gemeinsamen Schnittpunkt der drei Kreise gibt, wie Abbildung 10 vermittelt. Deswegen wird ein Bereich angegeben, wo sich der Master befinden muss. Je größer die Schwankungen bei der Distanzmessung sind, desto größer ist der Zielbereich [14]. Abbildung 11 verdeutlicht das Problem. Dort befindet sich der Master zwischen den folgenden Punkten:

$$\begin{aligned}
 A &: (2, 7671 \mid 3, 3700) \\
 B &: (3, 1775 \mid 2, 5081) \\
 C &: (2, 2727 \mid 1, 4454)
 \end{aligned} \tag{2.3}$$

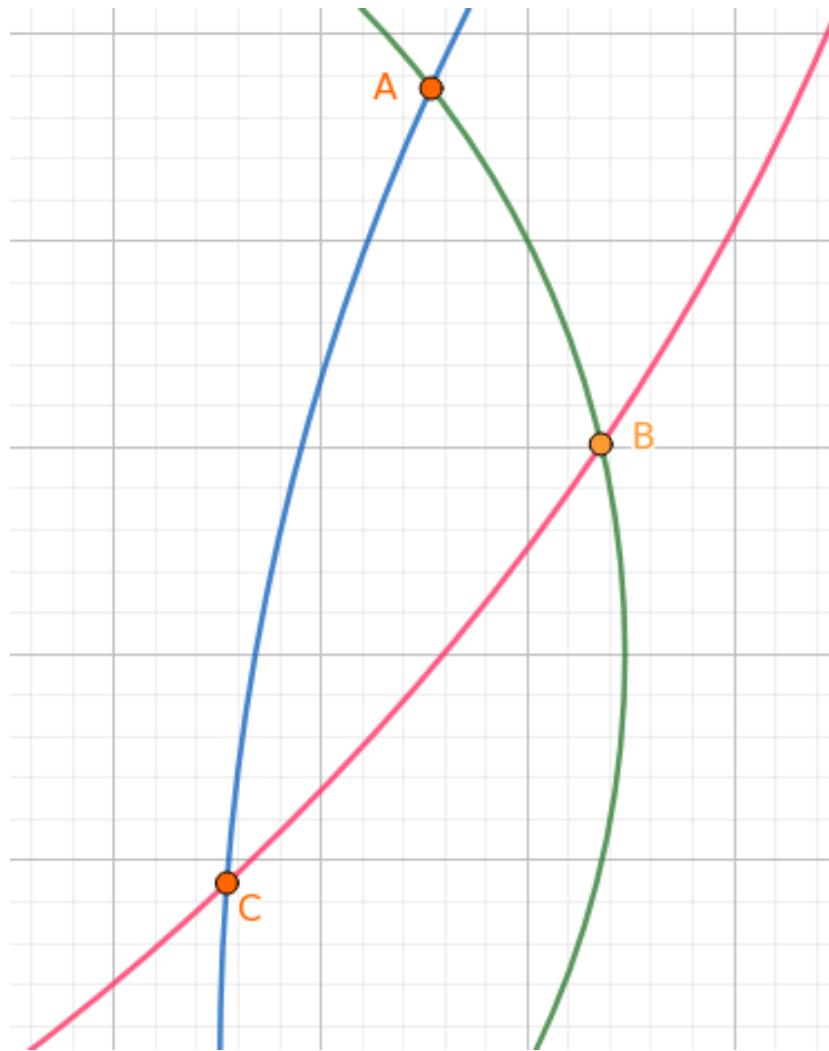


Abbildung 11: Bereich indem sich der Zielknoten befinden muss

Die Normalform für eine Kreisgleichung mit Mittelpunkt  $(x_0|y_0)$  und Radius  $r$  lautet:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \quad (2.4)$$

Mit Hilfe der drei Kreise (A, B, C) (Abbildung 10) kann nun der Schnittpunkt berechnet werden.  
- siehe folgendes Gleichungssystem 2.5. Dabei entsprechen  $x_A, y_A$  den Mittelpunkt von Kreis A mit Radius  $r_A$ . Diese Notation wird auch auf Kreis B und C angewendet.

$$\begin{aligned} I. \quad & (x - x_A)^2 + (y - y_A)^2 = r_A^2 \\ II. \quad & (x - x_B)^2 + (y - y_B)^2 = r_B^2 \\ III. \quad & (x - x_C)^2 + (y - y_C)^2 = r_C^2 \end{aligned} \quad (2.5)$$

Um den Schnittpunkt zwischen allen drei Kreisen zu erhalten, muss das Gleichungssystem 2.5 nach  $x$  und  $y$  aufgelöst werden. Aufgrund von Schwankungen ist dies nicht immer möglich. Punkt 1 berechnet sich durch das Auflösen nach  $x$  von I. und II. . Für Punkt 2 werden die

Gleichungen *I.* und *III.* verwendet, für den dritten Punkt *II.* und *III..* Die Herleitung ist im Anhang aufgeführt.

### 3 Entwurf

Dieses Kapitel befasst sich mit dem Entwurf der Implementierung. Damit wird die Herangehensweise beschrieben und es erläutert, wieso bestimmte Entscheidungen getroffen worden sind.

#### 3.1 Hardware

Um die Hardwarekosten bei einer Skalierung für die Positionsbestimmung niedrig zu halten, wurde der SparkFun Sound Detector auf dem Master-Knoten verbaut und dementsprechend die Lautsprecher auf dem Slave-Knoten. Diese Variante hat allerdings den Nachteil, dass nur ein Slave abgefragt werden kann. Somit braucht es bei drei Slaves mindestens drei Schallmessungen, gegenüber einer Messung. Die verwendete Variante hat jedoch den Vorteil, dass der Master entscheidet, mit welchem Slave eine Messung vorgenommen wird. Weiterhin ist der Softwareaufwand für eine Wiederholung der Messung, aufgrund von Fehlern geringer.

#### 3.2 Struktur

Zuerst wird der SparkFun Sound Detector dahingehend untersucht, ob er überhaupt schnell genug die Positionsbestimmung ist. Danach wird ein Versuchsaufbau nach dem folgenden Blockschaltbild 12 eingerichtet. Das Blockschaltbild teilt den Versuchsaufbau in verschiedene Module auf - damit lassen sich Abweichungen besser zuordnen.

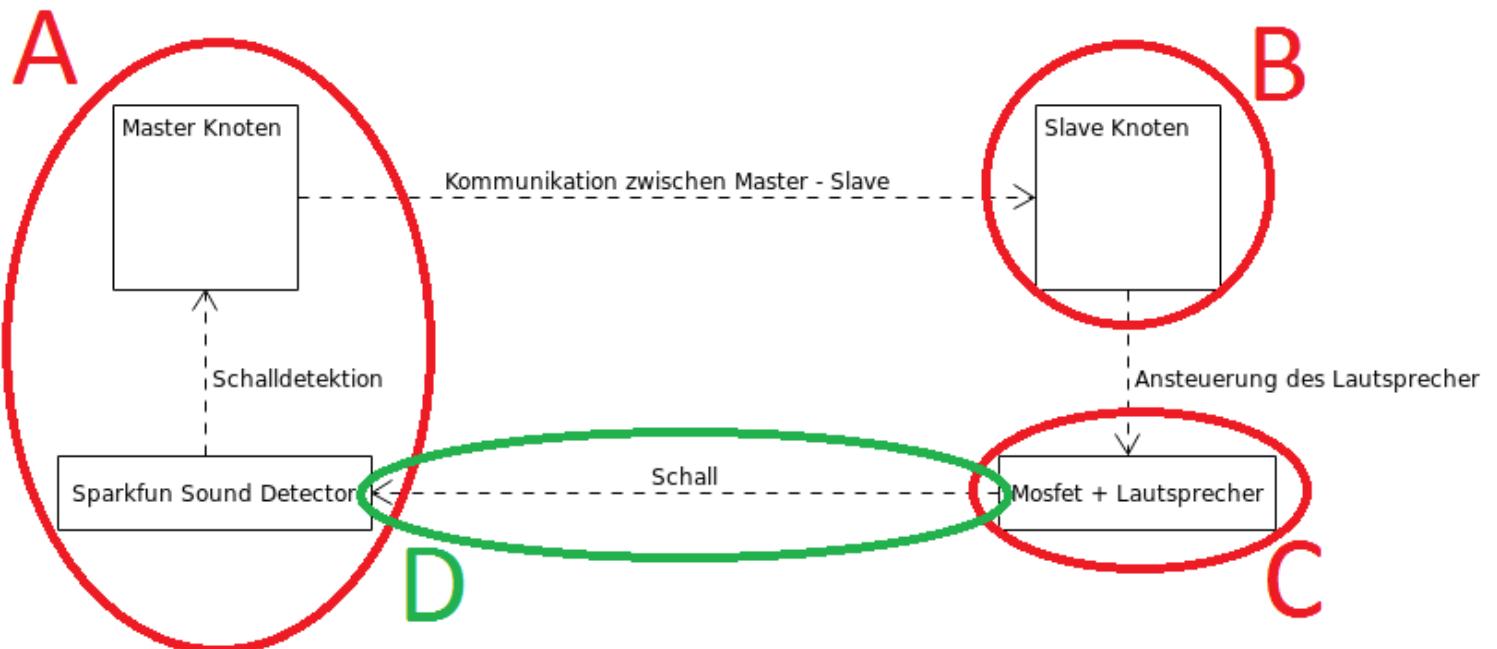


Abbildung 12: Versuchsaufbau, unterteilt in Module

Nachdem die einzelnen Module getestet worden sind, wird der Aufruf der RIOT Funktion `getSystemTime()` geprüft. Die Prüfung dieser Funktion ist wichtig, weil sie die aktuelle Sys-

temzeit ausliest. Sobald der Ton das Mikrofon passiert, wird diese Funktion aufgerufen. Diese Abweichung hat eine hohe Relevanz, da ein Funktionsaufruf mehrere CPU-Zyklen benötigt, bevor die Systemzeit in der Variable gespeichert werden kann. Anschließend wird die Zeitsynchronisation überprüft. Danach folgt ein Versuch, bei dem anstatt der Funkkommunikation vom SAM R21 Xplained Pro Evaluation Kit, das 433 Mhz Funk Transmitter-Receiver Modul verwendet wird. Die verwendete Software für die einzelnen Module ist auf dem mitgelieferten USB-Stick gespeichert. Bei den einzelnen Modulen ist der GATE-Pin bei dem Master an dem Pin PB23 angeschlossen. Für den Slave wird immer Pin PA18 verwendet. Die folgende Abbildungen (13, 14) zeigt die Verdrahtung.

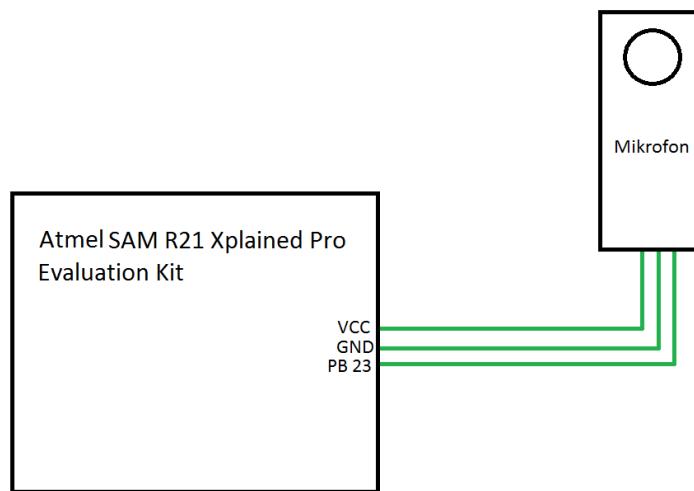


Abbildung 13: Verdrahtungsplan Master

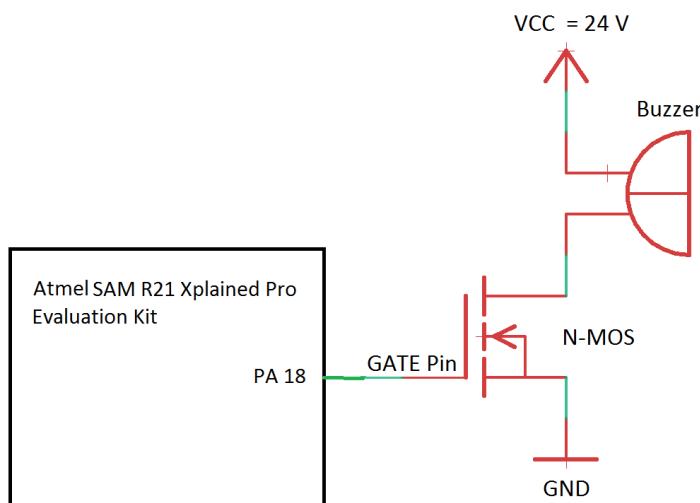


Abbildung 14: Verdrahtungsplan Slave

## 4 Implementierung

Nachdem die theroretische Einleitung abgeschlossen ist, widmen wir uns in diesem Kapitel der konkreten Umsetzung der Positionsbestimmung.

### 4.1 SparkFun Sound Detector

Bevor eine Messung mit dem Board durchgeführt wird, wird zuerst eine Plausibilitätscheck durchgeführt. Dabei wird überprüft ob das SparkFun Sound Detector Mikrofon korrekte Ergebnisse liefert. Der Aufbau ist folgend durch das Blockschaltbild 15 abgebildet. Die Abbildung 16 zeigt den Aufbau im Laborraum.

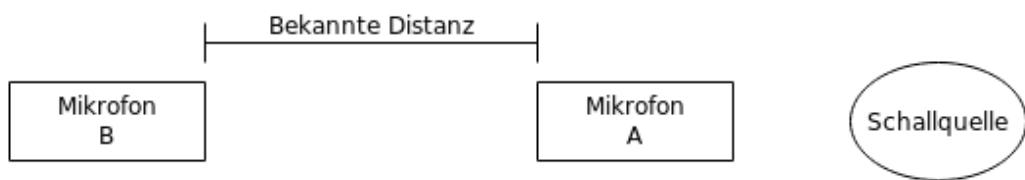


Abbildung 15: Versuchsaufbau als Blockschaltbild

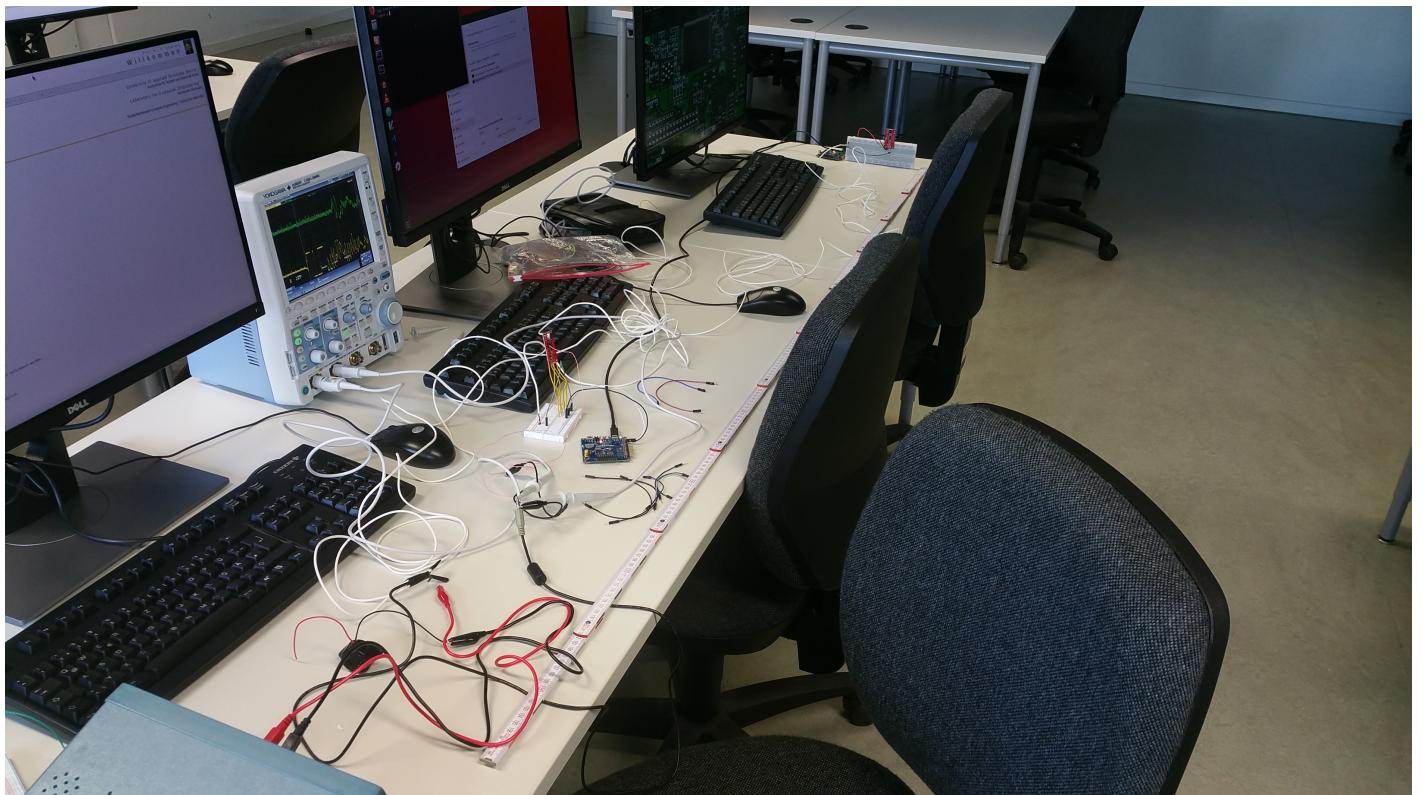


Abbildung 16: Versuchsaufbau

Die beiden Mikrofone sind an einem Oszilloskop angeschlossen. Es werden die AUDIO und

*GATE* Ausgänge untersucht. Wenn die Schallquelle einen Ton aussendet (durch Klatschen oder ähnliches), passiert es zuerst das Mikrofon A und mit einer Verzögerung Mikrofon B. Mithilfe des bekannten Abstandes der beiden Mikrofone wird untersucht ob das Ergebnis auf dem Oszilloskop mit dem Abstand der Mikrofone übereinstimmt. In der folgenden Abbildung 17 sind vier verschiedene Signale abgebildet. Dabei entspricht Signal *gelb* dem *GATE* Ausgang von Mikrofon A und *grün* dem *AUDIO* Ausgang. Signal *violett* und *blau* entsprechen dem *GATE* und *AUDIO* Ausgang von dem zweiten Mikrofon.



Abbildung 17: Verzögerung der Schallausbreitung

Es ist deutlich erkennbar, dass eine Verzögerung vorhanden ist. Die beiden *GATE* Ausgänge liegen ungefähr  $2863 \mu s$  auseinander. Mit einer Ausbreitungsgeschwindigkeit von  $0.034 \frac{cm}{\mu s}$  entspricht das ungefähr  $97,342 \text{ cm}$ . Der gemessene Abstand ist  $100 \text{ cm}$ . Diese Auswertung zeigt, dass die Verzögerung nur minimal mit dem gemessenen Abstand abweicht. Es folgt weitere Messungen bei dem anstatt eines Klatschen der verwendete Tongeber verwendet wird. Dabei sitzt der Lautsprecher direkt neben einem der Mikrofon. Das zweite Mikrofon sitzt  $1\text{m}$ ,  $2\text{m}$  und  $3\text{m}$

entfernt. Es folgen drei Tabellen (1 - 3) mit den Messwerten.

Tabelle 1: Messwerte bei 1 m Entfernung

<b>1 m Entfernung</b>		
<b>Messwert von Mikrofon 1 [μs]</b>	<b>Messwert von Mikrofon 2 [μs]</b>	<b>Differenz [μs]</b>
595,00	4345,00	3750,00
603,00	4340,00	3737,00
461,00	4440,00	3979,00
608,00	4560,00	3952,00
641,00	4670,00	4029,00
443,00	4400,00	3957,00
450,00	4415,00	3965,00
634,00	4335,00	3701,00
638,00	4425,00	3787,00
640,00	4850,00	4210,00
<b>Durchschnitt</b>		
571,30	4036,50	3511,00

Tabelle 2: Messwerte bei 2 m Entfernung

<b>2 m Entfernung</b>		
<b>Messwert von Mikrofon 1 [μs]</b>	<b>Messwert von Mikrofon 2 [μs]</b>	<b>Differenz [μs]</b>
476,00	20150,00	19674,00
458,00	37900,00	37442,00
471,00	36150,00	35679,00
681,00	38750,00	38079,00
452,00	38150,00	37698,00
466,00	34150,00	33684,00
675,00	38750,00	38075,00
450,00	33550,00	33100,00
447,00	32600,00	32153,00
460,00	35800,00	35340,00
<b>Durchschnitt</b>		
503,60	34595,00	34072,40

Tabelle 3: Messwerte bei 3 m Entfernung

3 m Entfernung		
Messwert von Mikrofon 1 [ $\mu\text{s}$ ]	Messwert von Mikrofon 2 [ $\mu\text{s}$ ]	Differenz [ $\mu\text{s}$ ]
454,00	33450,00	32996,00
428,50	35800,00	35371,50
474,00	37650,00	37176,00
429,00	33500,00	33071,00
437,50	37700,00	37262,50
439,50	35550,00	35110,50
435,50	25200,00	24764,50
437,50	23800,00	23362,50
430,00	24950,00	24520,00
427,50	25800,00	25372,50
Durchschnitt		
439,30	31340,00	30900,70

Aus den obigen Tabellen ist zu erkennen, dass Mikrofon 1 mindestens  $439,30 \mu\text{s}$  braucht um den Ton wahrzunehmen. Allerdings steht das Mikrofon 1 direkt neben dem Tongeber. Dieser Messwert müsste null sein. Weitherin ist zu erkennen, dass es eine deutliche Abweichung gibt obwohl der Versuch mit Klatschen als Tongeber ein brauchbares Ergebnis produzierte. Dieses Messergebnis zeigt, dass der Lautsprecher und das Mikrofon zusammen keine brauchbaren Messergebnisse erzielen.

## 4.2 Modul A

Das Modul A untersucht die Verzögerung des Masters vom detektieren eines Mikrofon und des Aufrufs der Interruptroutine. Das Mikrofon liefert bei erkennen eines Tons einen Pegelwechsel von LOW - HIGH. In der Interruptroutine des Masters wird ein Pin auf HIGH gesetzt. Der Slave besitzt dabei den Lautsprecher. Es wird ein PIEP-Ton erzeugt. Master und Slave agieren unabhängig voneinander. Das Oszilloskop wurde an den GATE-Ausgang des Mikrofon angehängt und an dem Pin der in der Interruptroutine auf HIGH gesetzt wird. Darüber hinaus wurde die Schallquelle dreimal versetzt um Abweichungen abhängig von der Distanz machen zu können. Es folgt eine Tabelle 4 mit den Messwerten. Die Messwerte wurden von dem Oszilloskop abgelesen.

Tabelle 4: ISR-Verzögerung Mikrofon-SAM R21 Xplained Pro Evaluation Kit bei unterschiedlicher Kabellänge

Messwert [ $\mu\text{s}$ ]		
1 m	2 m	3 m
67,00	91,00	66,00
78,00	83,00	88,50
94,00	86,00	72,50
81,50	59,00	75,50
78,50	82,00	83,00
87,50	76,00	82,50
74,00	87,50	92,50
87,00	80,00	84,50
88,50	88,00	88,00
81,50	70,50	64,50
Durchschnitt		
74,45	81,30	79,75

Die resultierenden Durchschnittswerte weichen nur minimal voneinander ab. Somit hat die Distanz zwischen der Schallquelle und dem Mikrofon keinen Einfluss auf die Interruptzeit. Da die Interruptzeit nur minimal abweicht, kann diese Verzögerung weggerechnet werden.

### 4.3 Modul B

Dieses Modul untersucht die Abweichungen vom aussenden eines HIGH-Signals des Master bis zum durchschalten des Mosfets beim Slave. Der Slave nimmt das HIGH-Signal vom Master durch eine Interruptroutine entgegen und setzt den GATE-Pin des Mosfets sofort auf HIGH. Der Master ist per Kabel mit dem Slave verbunden. Auch hier wird die Kabellänge verändert. Für dieses Modul wird kein Lautsprecher und Mikrofon benötigt.

Zu erwarten ist, dass die Interruptzeit nicht von der Kabellänge zwischen Master und Slave abhängig ist. Die Messwerte für eine Kabellänge von 5 m und einer direkten Verbindung sind wie folgt.

Tabelle 5: Verzögerung der Slave ISR bei unterschiedlicher Kabellänge

<b>Messwert [μs]</b>	
<b>Direkte Verbindung</b>	5 m
78,10	92,00
70,80	74,00
68,30	87,00
78,10	91,80
74,70	84,80
97,20	89,60
81,20	70,40
94,60	90,80
94,50	76,80
88,80	67,40
<b>Durchschnitt</b>	
82,46	82,63

Auch hier zeigt sich, dass die Interruptzeit nicht von der Kabellänge abhängig ist. Weiterhin ist die Interruptzeit des Slaves nahezu konstant, dadurch kann auch diese Verzögerung weggerechnet werden.

#### 4.4 Modul C

Hierbei soll die Anstiegszeit des N-Mosfet ermittelt werden. Die Anstiegszeit liegt laut dem Datenblatt bei 40 ns. Das verwendete Oszilloskop schafft es nicht, die Angabe zu überprüfen, da die Zeitauflösung nicht genau genug ist. Die Anstiegszeit verursacht eine Abweichung von  $0,000034[\frac{cm}{ns}] \cdot 40[ns] = 0,00136[cm]$ . Diese Abweichung ist zu gering, um Einfluss auf das Endergebnis auszuüben, da sie erst ab der dritten Nachkommastelle Einfluss ausübt. Aufgrund dessen wird diese Abweichung vernachlässigt.

#### 4.5 Modul D

Dieses Modul untersucht die Abweichungen vom Lautsprecher und dem Mikrofon. Dabei wird die Zeit zwischen dem Lautsprecher und dem GATE-Signal des SparkFun Sound Detector gemessen. Der Masterknoten wird nicht benötigt. Der Slaveknoten besitzt den Lautsprecher und den dazugehörigen Mosfet. Er macht den Lautsprecher für  $40000\mu s$  an. Theoretisch muss errechnete Wert aus der Distanz und der Schallgeschwindigkeit dem gemessenen Werten auf dem Oszilloskop entsprechen. Messungen werden mit den Distanzen 1 m, 2 m und 3 m gemacht. Es folgen drei Tabellen (6 - 8) mit den Messwerten.

Tabelle 6: Abweichung zwischen Mikrofon und Lautsprecher bei 1 m

1 m Entfernung			
Theorie [ms]	Messwert [ms]	Differenz [ms]	Distanz [cm]
2,94	5,14	2,19	74,76
2,94	5,14	2,19	74,76
2,94	4,87	1,92	65,58
2,94	4,87	1,92	65,58
2,94	5,15	2,20	75,10
2,94	4,88	1,93	65,92
2,94	4,87	1,92	65,58
2,94	4,90	1,95	66,60
2,94	4,89	1,94	66,26
2,94	4,90	1,95	66,60
Durchschnitt			
2,94	4,96	2,01	68,67

Tabelle 7: Abweichung zwischen Mikrofon und Lautsprecher bei 2 m

2 m Entfernung			
Theorie [ms]	Messwert [ms]	Differenz [ms]	Distanz [cm]
5,88	11,38	5,49	186,92
5,88	11,08	5,19	176,81
5,88	11,66	5,77	196,44
5,88	11,72	5,83	198,48
5,88	12,00	6,11	208,00
5,88	11,12	5,23	178,00
5,88	11,48	5,59	190,32
5,88	11,14	5,25	178,76
5,88	11,72	5,83	198,48
5,88	11,44	5,55	188,96
Durchschnitt			
5,88	11,47	5,59	190,10

Tabelle 8: Abweichung zwischen Mikrofon und Lautsprecher bei 3 m

3 m Entfernung			
Theorie [ms]	Messwert [ms]	Differenz [ms]	Distanz [cm]
8,82	23,95	15,12	514,30
8,82	35,70	36,87	913,80
8,82	31,75	22,92	779,50
8,82	24,25	15,42	524,50
8,82	30,75	21,92	745,50
8,82	22,30	13,47	458,20
8,82	22,30	13,47	458,20
8,82	34,00	25,17	856,00
8,82	23,60	14,77	502,40
8,82	26,15	17,32	589,10
Durchschnitt			
8,82	27,47	18,65	634,15

Aus den Messwerten geht hervor, dass mit steigender Distanz der Fehler größer wird. Der Fehler steigt nicht linear an. Da kein Muster erkennbar ist, kann diese Abweichung nicht verrechnet werden. Da bei den anderen Modulen ein konstanter Fehler zu erkennen war, und dieses Modul das nicht aufweist, muss die Abweichung des Endergebnis, zwischen dem Lautsprecher und dem Mikrofon liegen. Eine solche Abweichung macht die Positionsbestimmung im Zentimeterbereich unmöglich. Eine Vermutung für diesen größer werdenden Fehler ist, dass der Schall durch die Gegenstände im Raum abgelenkt wird und deswegen später beim Mikrofon ankommt. Weiterhin kann vielleicht durch eine Änderung der Tonfrequenz ein genaueres Ergebnis mit weniger Fehler produziert werden.

## 4.6 Systemzeit

Das RIOT OS stellt die Funktion `getSystemTime()` zur Verfügung. Als Rückgabewert wird ein 32Bitunsignedint zurückgegeben. Die Analyse soll verdeutlichen wie lange ein Aufruf kostet, um die Systemzeit auszulesen. Dabei wird vor dem Aufruf ein GPIO-Pin gesetzt und danach wieder zurückgesetzt. Die Differenz entspricht dabei ungefähr dem Systemaufruf. Es folgt eine Tabelle 9 mit den gemessenen Messwerten.

Tabelle 9: Messwerte für die Funktion *getSystemTime()*

<b>Messwert [μs]</b>
7,26
7,09
7,43
7,10
7,42
6,93
7,26
7,10
7,27
7,25
<b>Durchschnitt</b>
7,21

Aus den Messwerten erkennt man deutlich, dass der Aufruf nahezu konstant ist. Auch diese Abweichung kann im Endergebnis ausgeglichen werden. Der Durchschnittswert ist ein Maximalwert des Aufrufes, denn der Durchschnittswert beinhaltet das setzen und löschen des GPIO. Die Messwerte für das setzen und löschen des GPIO wird durch die folgende Tabelle verdeutlicht.

Tabelle 10: Messwerte für das Setzen und Löschen eines GPIO

<b>Messwert [ns]</b>
418,00
417,50
417,50
416,50
417,00
417,00
417,00
417,50
417,50
417,50
<b>Durchschnitt</b>
417,25

Aus dem Durchschnittswert ergibt sich eine Abweichung von 0,014178 cm. Dieser ist zu gering um auf die Messwerte der Funktion *getSystemTime()* Einfluss zu nehmen. Somit können die Messwerte aus Tabelle 10 ohne Fehler betrachtet werden.

#### 4.7 Zeitsynchronisation

Bei den bisherigen Messungen war der Master kabelgebunden mit dem Slave verbunden. Allerdings ist in der Endanwendung der Master drahtlos mit dem Slave verbunden. Bevor die Zeitsynchronisation in die Software eingefügt werden kann, wird geprüft welche Genauigkeit die Zeitsyn-

chronisation erreicht. Dabei werden zwei SAM R21 Xplained Pro Evaluation Kit nebeneinander aufgebaut. Es werden insgesamt hundert Zeitsynchronisation (PTP) nacheinander durchgeführt. Danach werden die Ergebnisse graphisch aufbereitet und ausgewertet. Es wird nur auf die Variable  $t_{prop}$  eingegangen, da diese die Differenz zwischen Slave und Master beinhaltet.

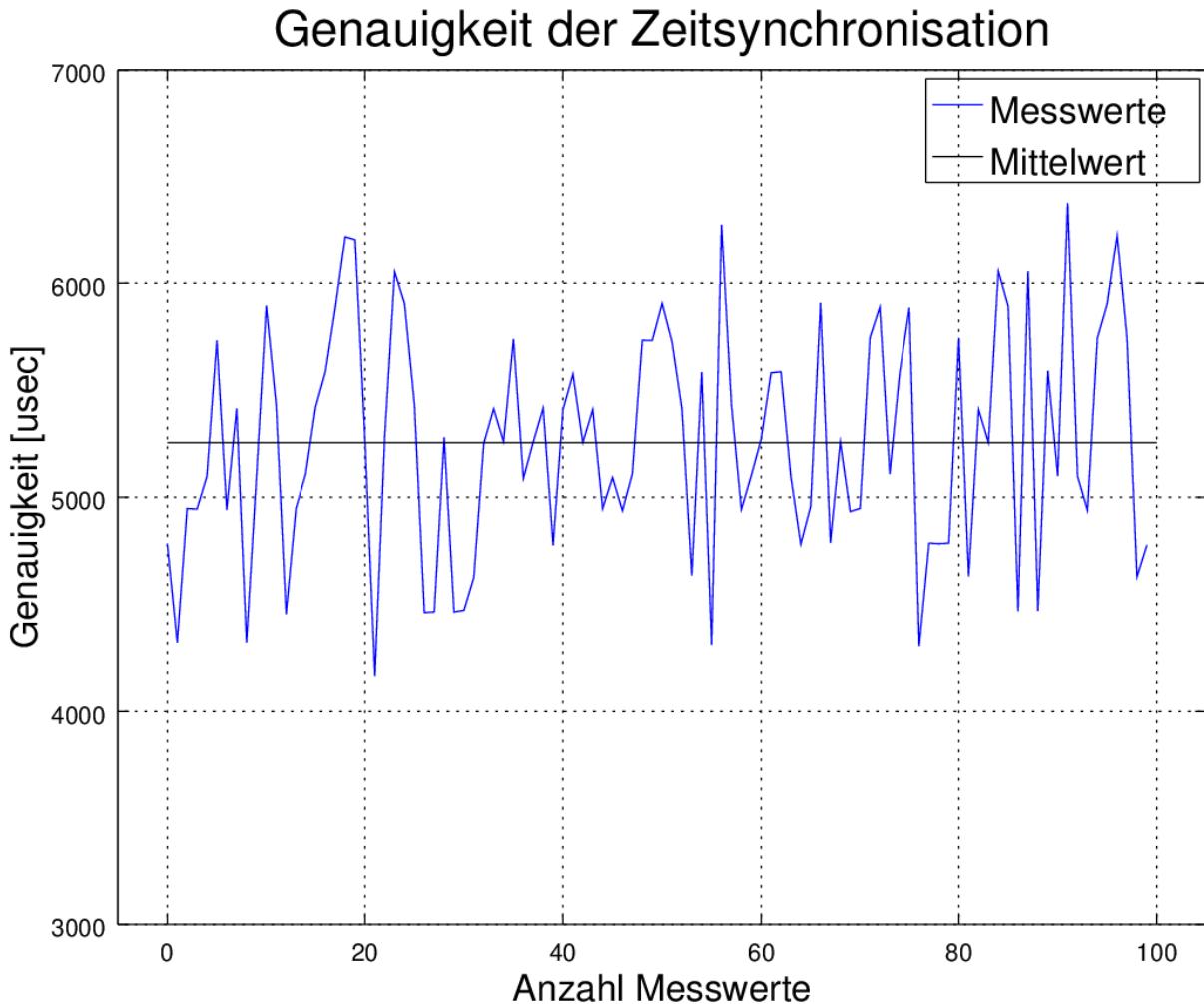


Abbildung 18: Messwerte von  $t_{prop}$  bei hundert Messungen

Die obige Abbildung 18 zeigt die erreichte Zeitsynchronisation. Da die Zeitsynchronisation sich im Millisekundenbereich bewegt, bedeutet das für die Positionsbestimmung eine durchschnittliche Abweichung von 178,6394 cm. Das Ergebnis kommt durch die folgende Formel 18 zustande.

$$5254,1 \text{ } [\mu\text{s}] \cdot 0,034 \left[ \frac{\text{cm}}{\mu\text{s}} \right] = 178,6394 \text{ } [\text{cm}] \quad (4.6)$$

Diese Abweichung kann nur durch die Abweichung des Aufrufes `getSystemTime()` zustande kommen. Der Funktionsaufruf hat eine Abweichung von 0,49028[cm]. Dies wird durch die folgen-

de Formel erklärt.

$$7,21[\mu\text{s}] \cdot 2 \cdot 0,034[\frac{\mu\text{s}}{\text{cm}}] = 0,49028[\text{cm}] \quad (4.7)$$

Allerdings erklärt das nicht die Abweichung von 178 cm. Eine Vermutung ist, dass die Funktion `udp_send_packet()`, die für das Aussenden der Pakete verantwortlich ist. In dieser Funktion wird das UDP-Paket noch zusammengebaut, welches widerrum Zeit kostet. Weitherin fragt der Empfänger durch Polling die Funktion `udp_receive_packet()` ab. RIOT bietet keine Interruptroutine für das Empfangen von Paketen an. Somit können Pakete verloren gehen, welches sich kritisch auf die Zeitsynchronisation auswirkt. Allerdings ist eine kleine Abweichung von Master und Slave nicht zu verhindern. Da jeder Frequenzgeber schwingt, kommt nicht jeder Taktpegel exakt genau an. Der Jitter gibt an, um wieviele Nanosekunden der Frequenzgeber abweicht kann. Dieser ist beim Master und Slave minimal unterschiedlich.

## 4.8 433 Mhz Funk Transmitter-Receiver

Um die Abweichung der Zeitsynchronisation aufzufangen, wird der 433 Mhz Funk Transmitter-Receiver für das Startsignal einer Messung verwendet. Dafür muss allerdings der Empfänger ein eindeutiges Signal erhalten. Dies ist ein LOW-Signal.

Da eine solche Abweichung für eine Zentimetergenaue Positionsbestimmung nicht vertretbar ist, wird für den Start der Messung ein 433 Mhz Funk Transmitter-Receiver verwendet. Für den Testaufbau wurde der Empfänger an das Oszilloskop angeschlossen. Der DATA-Eingang des Senders wurde mit dem SAM R21 Xplained Pro Evaluation Kit verbunden. Das Board produziert eine 1 Hz Frequenz auf dem verbunden Pin. Das Oszilloskop sollte nun diese Frequenz wieder-spiegeln. Das Messergebnis ist in Abbildung 19 dargestellt. Es ist zu erkennen das das Mikrofon Rauschen erkennt. Dies belegen die vielen kurzen LOW-Pegel. Als vom Sender ein LOW-Pegel gesendet wurde, kann nicht eindeutig bestimmt werden ob das Signal angekommen ist oder im Rauschen untergeht. Die Pegelwechsel A-E belegen, dass sich kein Pegelwechsel am Ausgang eindeutig vom Rauschen abhebt. Somit kann diese Variante nicht als Startsignal verwendet werden. Der Funkempfänger scheidet damit aus.

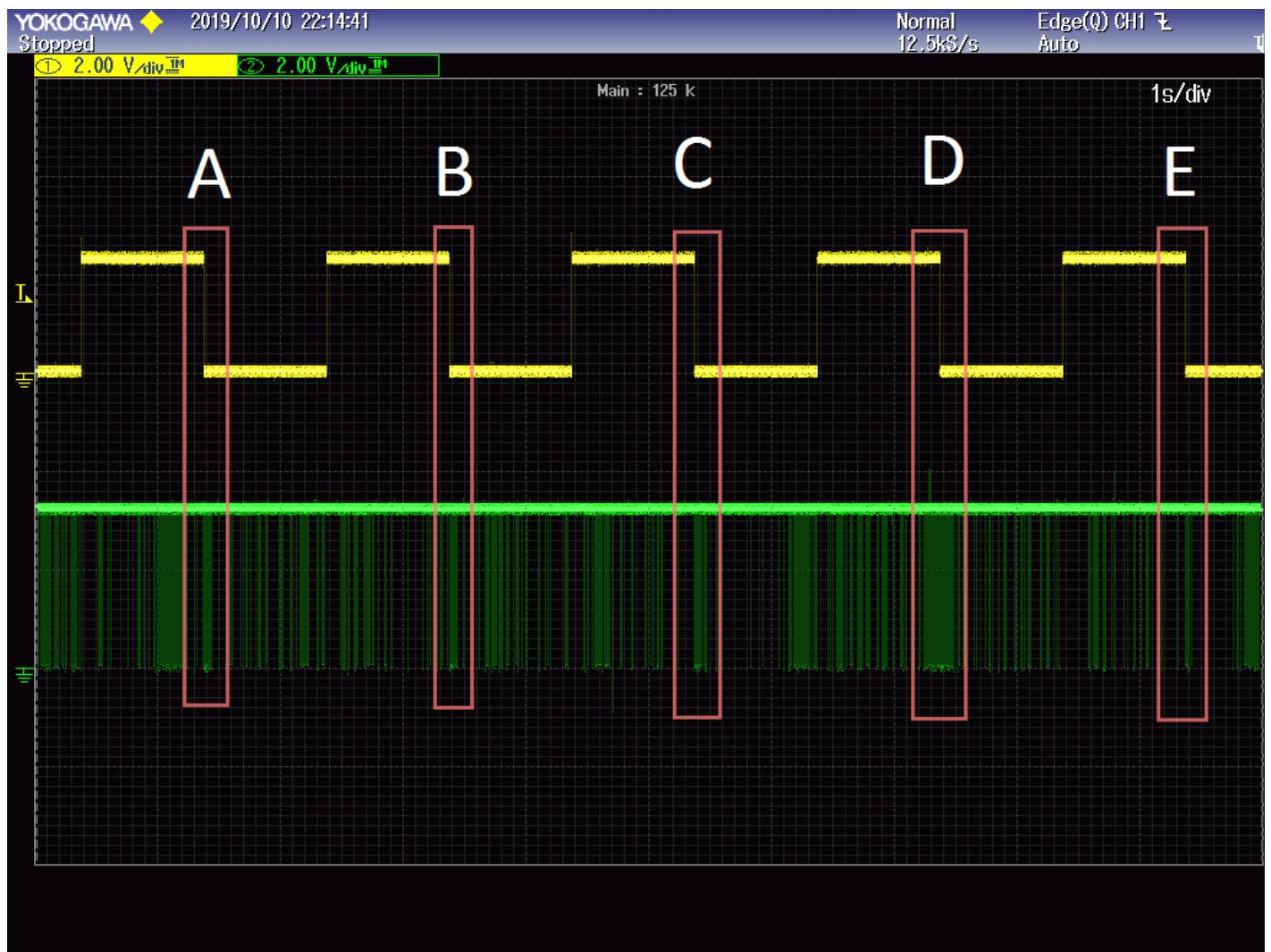


Abbildung 19: DATA-Pin des 433 Mhz Funk Transmitter-Receiver Empfänger

#### 4.9 Software

Um für die Positionsbestimmung, verhält sich die Software nach dem folgenden Programmablaufplan 20. Zuerst wird der Zeitunterschied ausgeglichen der zwischen den Master und Slave besteht. Dafür wird das bereits beschriebene PTP Protokoll verwendet. Erst nachdem die Zeitsynchronisation erfolgreich ist, werden Messungen vorgenommen. Dabei spricht der Master einzeln die Slaves an. Der Master gibt zu einem bestimmten Zeitpunkt vor, wann der entsprechende Slave den Lautsprecher für  $40000 \mu s$  einschaltet. Über das Mikrofon beim Master wird der Ton empfangen und es wird sofort die Systemzeit bestimmt. Über die Differenz vom Aussenden und Empfangen kann die Distanz berechnet werden.

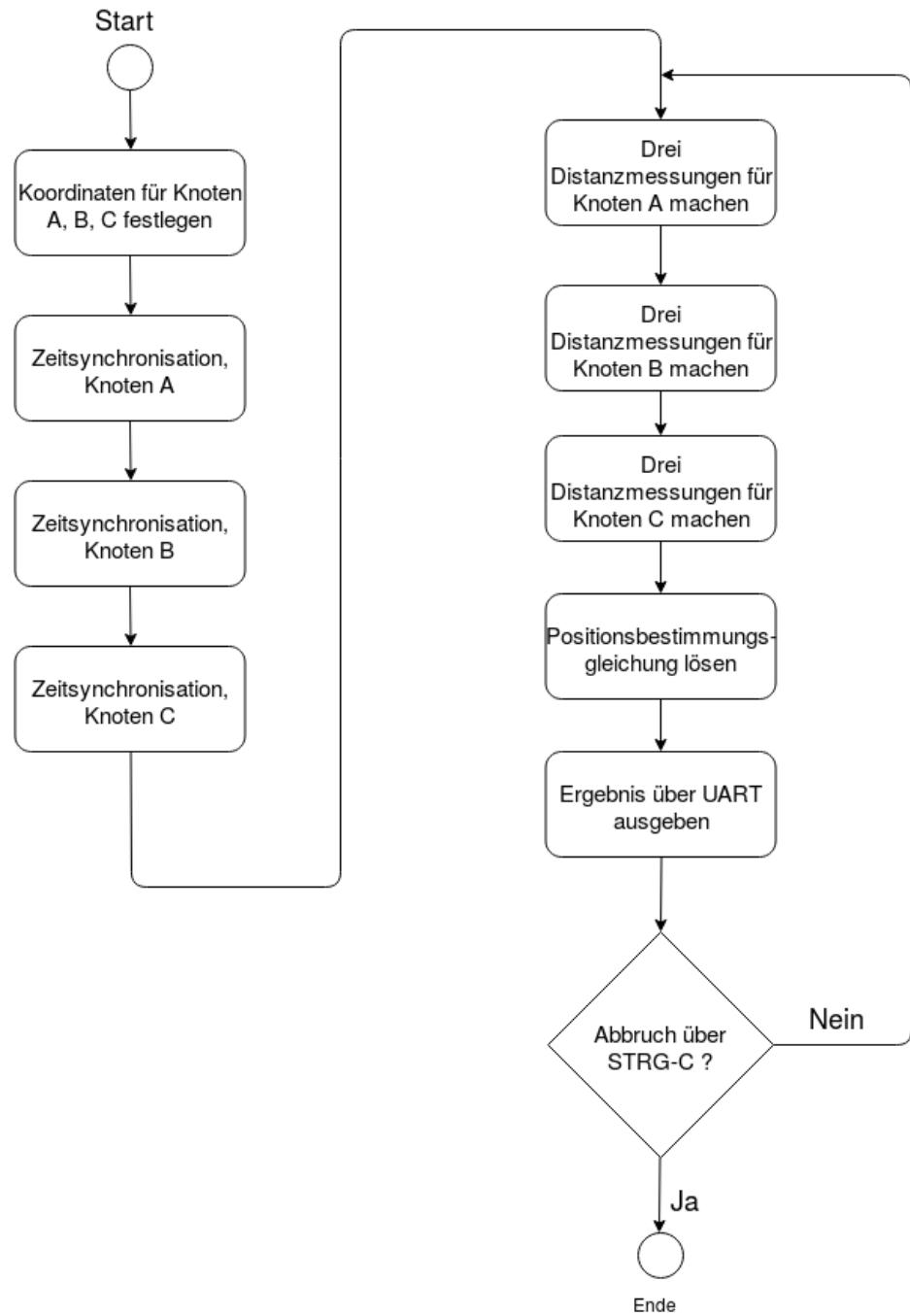


Abbildung 20: Programmablaufplan der Software

Dieses Vorgehen wird für alle drei Slaves wiederholt. Zusammen mit den Koordinaten der Slaves ergeben sich auf einer Ebene drei Kreise, die sich schneiden. Der Schnittpunkt der Kreise ist foglich die Position des Masters. Dabei ist zu achten das der Master sich während der drei Messungen nicht bewegen darf, sonst werden die Ergebnisse verfälscht. Mit den drei Distanzen wird die Position bestimmt. Die Funktion für die Positionsbestimmung wurde von github entnommen [15]. Wird die Software nicht abgebrochen, wird die Messung für alle Slaves wiederholt.

Somit kann der Master sich auf der Ebene bewegen und bekommt seine Position angezeigt. Es muss bedacht werden, dass durch den Jitter des Frequenzgebers die Systemzeit der Slaves mit der Zeit divergieren, d.h. wartet man entsprechend lange, ist die Systemzeit der Slaves nicht mehr übereinstimmend mit dem Master. Deswegen muss regelmäßig eine Zeitsynchronisation erfolgen. Da diese Arbeit unter Laborbedingungen durchgeführt wird, gibt es nur eine Zeitsynchronisation.

## 5 Auswertung

Nachdem die Implementierung abgeschlossen ist, widmen wir uns in diesem Kapitel der Auswertung der Daten.

### 5.1 Hardware

Die aufgetretenen Abweichungen entstanden meistens wenn die Peripherie Daten generiert hat bzw. angesprochen wurde. Da viele dieser Abweichungen konstant sind, kann ein Offset mit in das Ergebnis einfließen für ein korrektes Ergebnis. Allerdings sind einige Abweichungen nicht erklärbar, was die Bestimmung eines Offset erschwert. Weiterhin zeigt sich, dass die verwendete Hardware nicht optimal auf dieses Problem abgestimmt ist. Dazu zählt das Betriebssystem RIOT, der Tongeber und das SparkFun Sound Detector. RIOT sieht sich zwar als Echtzeitbetriebssystem, allerdings sind zu viele Schichten zwischen dem laufenden Programm und der Hardware, denn ein Systemaufruf für die Abfrage der Systemzeit darf nicht  $7,21 \mu\text{s}$  dauern. Dies verfälscht die Systemzeit die eigentlich  $\mu\text{-Sekunden}$  genau sein soll. Des weiteren zeigt sich, desto weniger Komponenten (Peripherie) verwendet wird, desto einfacher ist die Fehlersuche. Eine Laufzeitmessung könnte z.B. im Funkchip enthalten sein. Somit spart man sich aufwendige Peripherie.

### 5.2 Messergebnis

Die Messergebnisse zeigen, dass eine Positionsbestimmung auf diese Weise nicht genau genug ist. Die Abweichung die das Messergebnis hauptsächlich verfälscht, liegt beim Tongeber und dem Mikrofon. Dadurch das kein Muster bei der Abweichung erkennbar ist, kann kein Offset bestimmt werden. Weiterhin liefert das Oszilloskop die gleichen Abweichungen für alle verwendeten Mikrofone, sodass die Verwendung eines fehlerbehafteten Bausteins ausgeschlossen werden kann.

### 5.3 Software

Die Software kann eine Positionsbestimmung durchführen, dabei ist sie allerdings auf korrekte Distanzmesswerte angewiesen. Da diese fehlen, kann die Software nur durch Unit-Tests validiert werden.

## 6 Unit Test

Unitests werden benötigt um die Software auf ihre Korrektheit zu überprüfen. Für diese Software werden nur Teilbereiche getestet, da viele Funktionen von der Peripherie abhängig ist.

### 6.1 Quadratische Gleichung

Für die Positionsbestimmung muss eine quadratisch Gleichung gelöst werden. Die Parameter für diese Funktion sind p und q. Diese entsprechen den Variablen aus der folgenden Gleichung:

$$x^2 + p \cdot x + q = 0 \quad (6.8)$$

Es folgt eine Tabelle mit den Eingaben der Funktion und den dazugehörigen Ausgaben/ Rückgabewerten.

Tabelle 11: Eingaben und Ausgaben der pq-Funktion

Eingaben		Ergebnis	
p	q	x1	x2
3	2	-1	-2
4	1	-0,26	-3,73
-3	-9	4,85	-1,85

Aus der Tabelle 11 wird deutlich das die Funktion korrekte Rückgabewerte liefert. Bei einer fehlerhaften Eingabe gibt die Funktion ein NaN (Not a Number) zurück.

### 6.2 Abstand zweier Punkte

Bei der Positionsbestimmung kann es dazu kommen, dass die drei Kreise sich nicht treffen. Dann haben wir zwei Punkte Bereiche indem der Knoten sich befindet. Um den korrekten Bereich zu bestimmen, wird die Funktion *determine\_point\_radius()* benötigt. Liefert diese Funktion ein falsches Ergebnis führt das dazu das die Positionsbestimmung nicht mehr korrekt ist. Der korrekte Bereich kann bestimmt werden indem von zwei Punkten die Distanz zu einem Kreismittelpunkt bestimmt wird. Ein Punkt liegt innerhalb des Radius und der andere außerhalb. Der Punkt mit der geringsten Distanz ist der korrekte Punkt und markiert einen Eckpunkt des Bereichs indem sich der Knoten befindet. Die folgende Abbildung verdeutlicht das Problem.

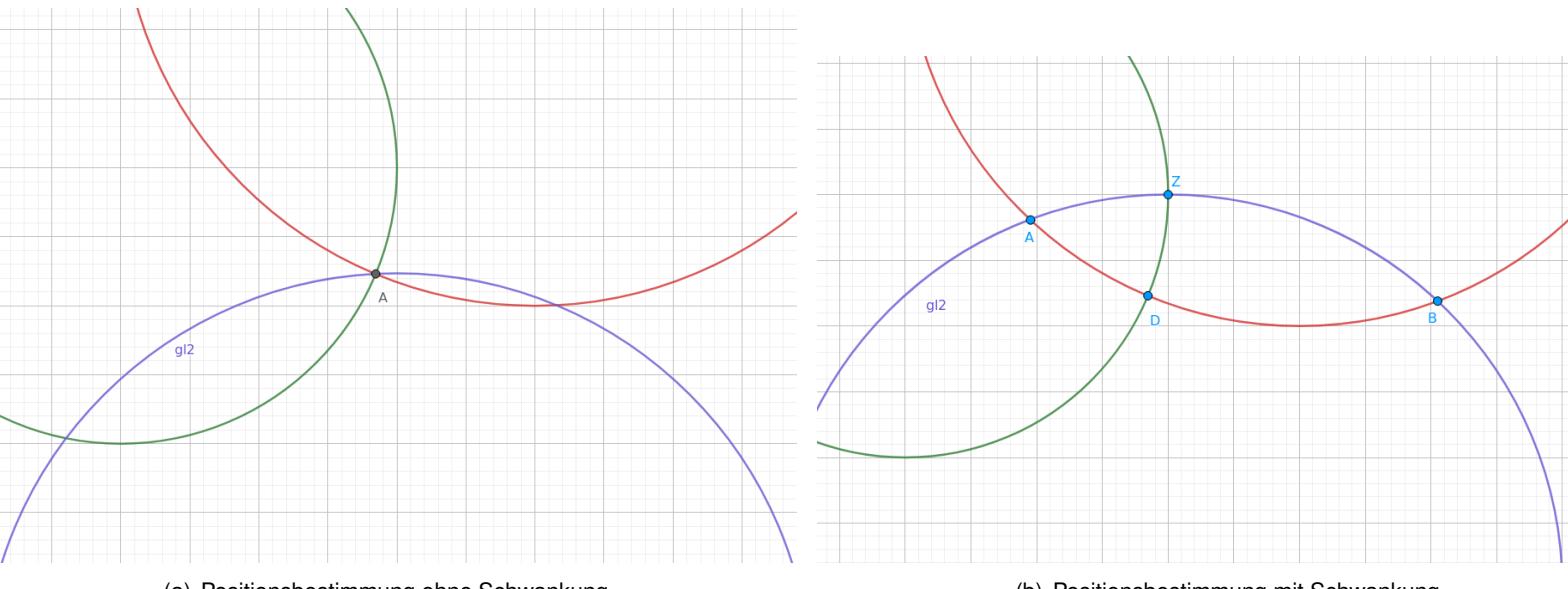


Abbildung 21: Zwei Möglichkeiten der Positionsbestimmung

Aus der Abbildung 21 wird erkennbar, dass es durch die Schwankung zwei Bereiche geben kann. Ein Bereich durch die Eckpunkte A, Z und D und ein zweiter durch B, Z und D abgesteckt. Um nun zu bestimmen ob Punkt A oder B den Bereich ausgegangen von dem grünen Kreis kennzeichnet wird der Abstand zum Kreismittelpunkt vom grünen Kreis berechnet. Die Funktion wird mit den folgenden zwei Punkten und der Kreisgleichung getestet.

$$P_A : (3, 9528 | 2, 8069)$$

$$P_B : (7, 0504 | 2, 1899)$$

$$(x - 3)^2 + (y - 3)^2 = 2^2$$

Punkt A ist der korrekte Punkt weil sein Abstand zum Kreismittelpunkt kleiner ist als von Punkt B ( $0.9721 < 4.1306$ ) und daher sollte er von der Funktion zurückgegeben werden. Die Funktion funktioniert wie erwartet fehlerfrei.

### 6.3 Positionsbestimmung

Zur Bestimmung der Position braucht die Funktion drei Parameter. Diese sind die drei Kreise die sich im einem Schnittpunkt schneiden oder ein Bereich aufspannen. Für den Fall das alle drei Kreise einen gemeinsamen Schnittpunkt haben gibt die Funktion dreimal die X/Y-Koordinaten zurück. Bei einem Bereich werden die X/Y-Koordinaten der Eckpunkte zurückgegeben. Die Funktion wird mit den folgenden Kreisgleichungen getestet. Zuerst ein Test wobei es ein gemeinsamer Schnittpunkt gibt.

$$(x - 3)^2 + (y - 3)^2 = 2^2$$

$$(x - 6)^2 + (y - 2)^2 = 2^2$$

$$(x - 4)^2 + (y - 1,8693)^2 = 2^2$$

Der Schnittpunkt der drei Kreise ist  $(4, 8872 \mid 3, 6618)$ . Dies gibt auch die Funktion zurück. Als nächstes wird der Rückgabewert bei keinem gemeinsamen Schittpunkt untersucht.

$$(x - 3)^2 + (y - 3)^2 = 2^2$$

$$(x - 6)^2 + (y - 2)^2 = 2^2$$

$$(x - 4)^2 + (y - 5)^2 = 2^2$$

Die obigen drei Kreise haben keinen gemeinsamen Schittpunkt. Die Funktion gibt die folgenden Koordinaten der Eckpunkte zurück.

$$P_A : (4, 8872 \mid 3, 6618)$$

$$P_B : (4, 9832 \mid 3, 2583)$$

$$P_C : (4, 2794 \mid 3, 0196)$$

In beiden Fällen ist der Rückgabewert der Funktion korrekt. Somit funktioniert die Funktion wie erwartet.

## 7 Gelöste Probleme

Dieser Abschnitt widmet sich den Problemen, die erst bei der Durchführung dieser Arbeit aufgetreten sind und vorher nicht abzuschätzen waren.

### 7.1 Kommunikation Master – Slave

Die Idee der drahtlosen Kommunikation im Rahmen dieser Bachelorarbeit bestand darin, dass Steuerkommandos und Daten immer getrennt gesendet und immer auf 32-Bit aligned werden. Das hat zur Folge, dass Daten immer vom vorherigen Steuercode abhängig sind. Kommt der Steuercode nicht beim Empfänger an, können danach folgende Daten dem nicht zugeordnet bzw. korrekt interpretiert werden. Da es häufig zu Verbindungsabbrüchen während der Kommunikation kam, wurde diese Kommunikationsvariante durch Structs abgelöst. Hierbei enthält das Struct den Steuercode und die dazugehörigen Daten. Dies hat den Vorteil, dass auch der Datenoverhead verringert wird.

### 7.2 Genauigkeit Zeitsynchronisation

Bei der Zeitsynchronisation kam es immer wieder zu dem Problem das die Genauigkeit bei wiederholten mal, nicht besser wird. Somit ist es egal ob eine Zeitsynchronisation einmal oder mehrmals stattfindet. Dadurch das die Zeitsynchronisation nicht im  $\mu$ -Sekundenbereich auflöst, kann keine Messung ohne große Abweichung erfolgen. Die Abweichung kommt hauptsätzlich durch die Aufrufe `getSystemTime()` und `udp_send_packet()` zustande. Denn die Systemzeit wird vor dem Aussenden vom Programm selber in das Datenpaket eingefügt und nicht von der Firmware des Funksenders. Weitherhin baut die Funktion `udp_send_packet()` zuerst das UDP Paket zusammen, welches wiederrum Zeit kostet bis es gesendet wird. Eine weitere Vermutung ist, dass aufgrund von Störsignalen keine  $\mu$ -Sekundenauflösung erreicht werden kann für die Zeitsynchronisation.

### 7.3 Messgenauigkeit

Theoretisch ist nach einer Zeitsynchronisation bekannt, um welche Zeit der Master dem Slave hinterherhängt, bzw. vorraus ist. Allerdings kam es bei den Distanzmessungen mit dem SAM R21 Xplained Pro Evaluation Kit dabei immer wieder zu großen Distanzschwankungen. Diese haben folgende Ursachen. Eine nicht im  $\mu$ -Sekundenbereich auflösende Zeitsynchronisation, Verzögerung der Messinstrumente und der Ableitung des Schalls vom Aussenden bis zum Empfangen des Mikrofons.

## 8 Ausblick

Die Software kann man in verschiedenen Module einteilen. Dadurch ist es möglich, die Software zu erweitern ohne die anderen Module zu verändern. Folgend werden Verbesserungsvorschläge für die einzelnen Module vorgestellt.

### Kommunikation

Für die Kommunikation zwischen Master und Slave, könnte anstatt des UDP-Protokoll das TCP-Protokoll verwendet werden. Dadurch wird eine fehlerbehaftete Kommunikation verbessert, weil TCP mit ACKs arbeitet und zuerst ein Verbindungsaufbau erfolgt. Darüber hinaus könnte die Identifikation der Knoten im Netzwerk auf IPv4 oder IPv6 umgestellt werden.

### Zeitsynchronisation

Damit eine Uhrensynchronisation im Nanosekundenbereich erreicht wird, muss die aktuelle Uhrzeit kurz vor dem Aussenden in das Datenpaket eingefügt werden. Dies sollte unabhängig vom System geschehen. Dafür muss die Systemzeit von der Firmware des Funkmoduls verwaltet werden. Wird die Systemzeit allerdings vom Betriebssystem verwaltet, gibt es immer eine Verzögerung von dem Funktionsaufruf *getSystemTime()* und dem Aussenden des Pakets. Genau um diese Verzögerungszeit zu eliminieren, sollte bei jedem Aussenden des Paketes die aktuelle Systemzeit angehängt werden.

### Übertragungsmedium

Anstatt von Schall, können auch Radiosignale verwendet werden. Diese haben den Vorteil dass das menschliche Gehör diese Frequenzen nicht wahrnimmt. Des Weiteren breiten sich Radiosignale mit Lichtgeschwindigkeit aus. Zusammen mit einer Frequenzmodulation können mehrere Accesspoints gleichzeitig abgefragt werden. Dies ermöglicht eine schnellere Positionsbestimmung.

### Messung

Für die Positionsbestimmung müssen drei Gleichungen gelöst. Da MCUs nicht immer über eine FPU verfügen, können die gemessenen Daten über UART ausgegeben werden und dann mit Octave oder Matlab weiterverarbeitet werden. Das hat den Vorteil, dass die MCU entlastet wird und sich ganz auf die Messung konzentrieren kann. Darüber hinaus ermöglicht Octave, Statistiken oder graphische Aufbereitung der Daten.

### Hardware

Aktuell ist der Lautsprecher und das Mikrofon über ein Steckbrett mit dem SAM R21 Xplained Pro Evaluation Kit verbunden. Es könnte eine Leiterplatte angefertigt werden, die auf das SAM R21 Xplained Pro Evaluation Kit aufgesteckt wird. Das würde eventuelle Fehler beim Steckbrett vermeiden.

**Signalanalyse**

Um genauer herauszufinden, wann das Signal beim Mikrofon ankommt, kann eine Signalanalysen durchgeführt werden. Dadurch ist es dann möglich, den exakten Zeitpunkt zu bestimmen, wann das Lautsprechersignal das Rauschen am *AUDIO*- Ausgang überlagert. Damit wird weiterhin die Genauigkeit erhöht.

**Betriebssystem**

Anstatt das Betriebssystem RIOT zu verwenden, kann auch ein eigenes Betriebssystem geschrieben und verwendet werden. Dies ermöglicht eine bessere Programmanalyse über das laufende Programm. Darüber hinaus können eventuelle Softwareroutinen die nicht unbedingt essentiell sind, abgeschaltet werden, bzw. gar nicht erst implementiert werden.

## **Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, XX.XX.2019

---

Oliver Koepp

sdfbsbf[? ]

## Literaturverzeichnis

- [1] *CE-Car* - URL <https://ce-master.htw-berlin.de/> - abgerufen am 02.09.2019
- [2] *In- und Outdoor Positionierungssysteme* - URL [https://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/2005-ws/seminar/Vortrag\\_Positionierung-Uebersicht\\_Indoor\\_und\\_Outdoor\\_Positionierungssysteme.pdf](https://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/2005-ws/seminar/Vortrag_Positionierung-Uebersicht_Indoor_und_Outdoor_Positionierungssysteme.pdf) - abgerufen am 02.09.2019
- [3] *Atmel SAM R21* - URL [https://riot-os.org/api/group\\_\\_boards\\_\\_samr21-xpro.html](https://riot-os.org/api/group__boards__samr21-xpro.html) - abgerufen am 02.09.2019
- [4] *Ultraschall Sensor HC-SR04 und kompatible Ultraschall-Module* 2016 - URL <https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc-sr04/> - abgerufen am 02.09.2019
- [5] *Sound Detector Hookup Guide* - URL <https://learn.sparkfun.com/tutorials/sound-detector-hookup-guide> - abgerufen am 02.09.2019
- [6] *433 Mhz Funk Transmitter-Receiver* - URL <https://draeger-it.blog/arduino-tutorial-433mhz-sender-empfaenger/> - abgerufen am 02.09.2019
- [7] *Goliton 5X Alarm 95dB Hohe Dezibel 6-24V 12V elektronischer Summer kontinuierlicher Piep Arduino MFC-27* - URL [https://www.amazon.de/Goliton-Decibel-elektronischer-kontinuierlicher-Arduino/dp/B01MT5V0FM/ref=sr\\_1\\_1?\\_\\_mk\\_de\\_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=Goliton+5X+Alarm+95dB+Hohe+Dezibel+6-24V&qid=1572779993&sr=8-1](https://www.amazon.de/Goliton-Decibel-elektronischer-kontinuierlicher-Arduino/dp/B01MT5V0FM/ref=sr_1_1?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=Goliton+5X+Alarm+95dB+Hohe+Dezibel+6-24V&qid=1572779993&sr=8-1) - abgerufen am 01.08.2019
- [8] *RIOT-The friendly Operating System for the Internet of Things* - URL <https://riot-os.org/> 2013 - abgerufen am 02.09.2019
- [9] *TIME DIFFERENCE OF ARRIVAL* - URL <https://www.sewio.net/uwb-technology/time-difference-of-arrival/> - abgerufen am 02.09.2019
- [10] *UDP - User Datagram Protocol* - URL <https://www.elektronik-kompendium.de/sites/net/0812281.htm> - abgerufen am 02.09.2019
- [11] *Introduction to RAW-sockets* - URL <https://tuprints.ulb.tu-darmstadt.de/6243/1/TR-18.pdf> - abgerufen am 14.08.2019
- [12] *In- und Outdoor Positionierungssysteme* - URL [https://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/2005-ws/seminar/Vortrag\\_Positionierung-Uebersicht\\_Indoor\\_und\\_Outdoor\\_Positionierungssysteme.pdf](https://pi4.informatik.uni-mannheim.de/pi4.data/content/courses/2005-ws/seminar/Vortrag_Positionierung-Uebersicht_Indoor_und_Outdoor_Positionierungssysteme.pdf)

- [13] *A Sub-Microsecond Clock Synchronization Protocol for Wireless Industrial Monitoring and Control Networks* 2017 - URL [https://www.ibr.cs.tu-bs.de/oa/vonzengen\\_ICIT2017.pdf](https://www.ibr.cs.tu-bs.de/oa/vonzengen_ICIT2017.pdf) - abgerufen am 26.08.2019
- [14] *Finding Location with Time of Arrival and Time Difference of Arrival Techniques* - URL [https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick\\_OKeefe\\_F1.pdf](https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick_OKeefe_F1.pdf) 2017 - abgerufen am 11.07.2019
- [15] *Software zur Positionsbestimmung* 2019 - URL <https://github.com/pepebecker/circle-intersection> - abgerufen am 28.10.2019

## 9 Anhang

### 1 Mathematische Herleitung

Diese Arbeit zeigt die Herleitung, nur für einen Punkt aus der Abbildung 11, denn die Herleitungen der anderen beiden Punkte unterscheiden sich nur in den Variablen  $x_A$ ,  $y_A$  und  $r_A$ . Zuerst wird eine Geradengleichung bestimmt, die durch die beiden Schnittpunkte von Kreis A und B gehen.

I.

$$(x - x_A)^2 + (y - y_A)^2 = r_A^2$$

II.

$$(x - x_B)^2 + (y - y_B)^2 = r_B^2$$

$$I. - II. \quad x \cdot (2 \cdot x_B - 2 \cdot x_A) + y \cdot (2 \cdot y_B - 2 \cdot y_A) + x_A^2 - x_B^2 + y_A^2 - y_B^2 = r_A^2 - r_B^2$$

$$y = \frac{x \cdot (2 \cdot x_A - 2 \cdot x_B) + r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2}{2 \cdot (y_B - y_A)} \quad (1.9)$$

Um nun die X-Koordinaten zu bekommen, setzen wir die Gleichung 1.9 in Gleichung I. ein und lösen nach  $x$  auf.

$$\begin{aligned} P &= r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2 \\ (x - x_A)^2 + (y - y_A)^2 &= r_A^2 \\ (x - x_A)^2 + \left( \frac{x \cdot (2 \cdot x_A - 2 \cdot x_B) + P}{2 \cdot (y_B - y_A)} - y_A \right)^2 &= r_A^2 \\ x^2 \left(1 + \left(\frac{x_B - x_A}{y_B - y_A}\right)^2\right) + x \left(\frac{-P \cdot (x_B - x_A)}{(y_B - y_A)^2} + \frac{2 \cdot y_A \cdot (x_B - x_A)}{y_B - y_A} - 2 \cdot x_A\right) &= r_A^2 - y_A^2 + \frac{y_A \cdot P}{y_B - y_A} - \frac{P^2}{4 \cdot (y_B - y_A)^2} - x_A^2 \\ (1 + \left(\frac{x_B - x_A}{y_B - y_A}\right)^2) &= R \\ \left(\frac{-P \cdot (x_B - x_A)}{(y_B - y_A)^2} + \frac{2 \cdot y_A \cdot (x_B - x_A)}{y_B - y_A}\right) &= S \\ r_A^2 - y_A^2 + \frac{y_A \cdot P}{y_B - y_A} - \frac{P^2}{4 \cdot (y_B - y_A)^2} - x_A^2 &= T \\ x^2 + x \cdot \frac{S - 2 \cdot x_A}{R} - \frac{T}{R} &= 0 \end{aligned} \quad (1.10)$$

Zum lösen einer quadratischen Gleichung wird die pq-Formel verwendet.

$$p = \frac{S - 2 \cdot x_A}{R}$$

$$q = \frac{T}{R}$$

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q} \quad (1.11)$$

$$x_1 = -\frac{p}{2} + \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

$$x_2 = -\frac{p}{2} - \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

Nachdem die X-Koordinaten ermittelt worden sind, muss noch die Y-Koordinate für  $x_1$  und  $x_2$  bestimmt werden. Dafür werden die X-Koordinaten in die Geradengleichung 1.9 eingesetzt.

$$I. \quad (x - x_A)^2 + (y - y_A)^2 = r_A^2$$

$$y_1 = \frac{x_1 \cdot (2 \cdot x_A - 2 \cdot x_B) + r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2}{2 \cdot (y_B - y_A)} \quad (1.12)$$

$$y_2 = \frac{x_2 \cdot (2 \cdot x_A - 2 \cdot x_B) + r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2}{2 \cdot (y_B - y_A)}$$

Die Schnittpunkte von Kreis A und B sind:

$$(x_1 | y_1) \quad (1.13)$$

$$(x_2 | y_2)$$

Da nur ein Schnittpunkt auch in Kreis C liegt, muss dieser bestimmt werden, sonst erhalten wir nicht minimalste Fläche die drei Kreise erzeugen. Um zu prüfen welcher Schnittpunkt in Kreis C liegt, wird der Abstand vom Mittelpunkt des Kreises C zu den Schnittpunkten aus Gleichung 1.13 berechnet.

$$d_1 = \sqrt{(x_C - x_1)^2 + (y_C - y_1)^2} \quad (1.14)$$

$$d_2 = \sqrt{(x_C - x_2)^2 + (y_C - y_2)^2}$$

Wenn  $d_1 < d_2$  ist, dann ist der gesuchte Punkt  $(x_1 | y_1)$ . Falls die Ungleichung  $d_1 > d_2$  wahr ist, dann heißt der gesuchte Punkt  $(x_2 | y_2)$ .

Diese Mathematische Vorgehen wird für Kreis A – C, und B – C wiederholt.

## 2 Steuercodes

Die folgende Tabelle listet alle vorhandenen Steuercodes auf.

Tabelle 12: Alle Steuercodes

Steurcode	<i>unsigned int</i> Wert	Beschreibung
CODE_MESSUNG	65	Messung durchführen
CODE_NOP	66	ACK anfordern
CODE_ZEIT_SYNC	67	SYNC-MSG senden
CODE_ZEIT_FOLLOW_UP	68	FOLLOW_UP-MSG senden
CODE_ZEIT_DELAY_REQ	69	DELAY_REQ-MSG senden
CODE_ZEIT_DELAY_RESP	70	DELAY_RESP-MSG senden
CODE_READ_T1_T2_T4	71	Zurücksenden der Werte $t_1, t_2, t_4$
CODE_SERVER_RESPONSE	72	ACK