



**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 Koepp

Matrikelnummer: s0559122

Abgabe: 17.12.2019

Inhaltsverzeichnis

1	Einleitung	1
1.1	Aufbau der Arbeit	1
2	Grundlagen	2
2.1	Verwendete Hardware	2
2.2	Verwendete Software	7
2.3	Time Difference of Arrival – TDOA	8
2.4	User Datagram Protocol – UDP	8
2.5	Netzwerk-Socket	9
2.6	Indoor/Outdoor Positionsbestimmung	9
2.7	Zeitsynchronisation	9
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	26
4.8	433 MHz Funk Transmitter-Receiver	27
4.9	Software	28
5	Unit Test	31
5.1	Quadratische Gleichung	31
5.2	Abstand zweier Punkte	31
5.3	Positionsbestimmung	33
6	Auswertung	34
6.1	Hardware	34
6.2	Messergebnis	34
6.3	Software	34
6.4	RIOT	34
6.5	Zeitsynchronisation	35

6.6	Mathematischer Hintergrund	35
7	Probleme	36
7.1	Kommunikation Master – Slave	36
7.2	Genauigkeit – Zeitsynchronisation	36
7.3	Genauigkeit – Messungen	36
8	Ausblick	37
9	Zusammenfassung	39
	Eidesstattliche Erklärung	40
	Literaturverzeichnis	41
10	Anhang	43
10.1	Mathematische Herleitung	43
10.2	Steuercodes	45

Abbildungsverzeichnis

1	Vorderseite des SAM R21 Xplained Pro Evaluation KitBoard	2
2	Vorderseite des HC-SR04 Sensors	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	7
8	Vergleich von RIOT mit drei anderen Betriebssystemen	8
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	27
19	DATA-Pin des 433 MHz Funk Transmitter-Receiver Empfänger	28
20	Programmablaufplan der Software	30
21	Positionsbestimmung ohne Schwankung	32
22	Positionsbestimmung mit Schwankung	32

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	31
12	Alle Steuercodes	45

Abkürzungsverzeichnis

MCU	Microcontroller Unit
FPU	Floating Point Unit
RISC	Reduced Instruction Set Computer
ARM	Advanced RISC Machine
<i>I²C</i>	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
GPIO	General-Purpose Input/Output
MAC	Media-Access-Control
IPv4	Internet Protocol Version 4
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

1 Einleitung

Diese Arbeit hat das Ziel, eine Positionsbestimmung auf Basis von Schalllaufzeitmessungen zu entwickeln und aufzubauen. Sie soll die Grundlage sein für eine fehlerfreie Fahrt in Gebäuden mit dem hausinternen CE-Car, wobei die Positionsgenauigkeit dabei eine untergeordnete Rolle spielt [1]. 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 Indoor-Bereich zu entwickeln – mit dem Fokus auf geringe Kosten. Damit können Modellautos, Drohnen oder Roboter ausgestattet werden. Darüber hinaus muss das System ohne externe Dienste oder Netzanbindung funktionstüchtig sein. Zudem sollte es einfach erweiterbar sein, um der zukünftigen Entwicklung Schritt zu halten. Die Validierung erfolgt durch eine prototypische Anwendung.

1.1 Aufbau der Arbeit

Die Bachelorarbeit ist folgendermaßen gegliedert: Neben der Einleitung in Kapitel 1 werden in Kapitel 2 die theoretischen Grundlagen erläutert, sowie die verwendete Hardware und Software beschrieben. Darauf aufbauend beschreibt Kapitel 3 den Entwurf und danach in Kapitel 4 die Implementierung. Im Anschluss wird das System ausgewertet und 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 einen energie-sparenden ISM-Bandsender-Empfänger auf einem Chip. Dieser Sender nutzt die Frequenz 2,4 GHz zur Datenübertragung. Mit den vorhandenen GPIOs kann das Board verschiedene Sensoren und Aktoren ansteuern. Damit wird 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.

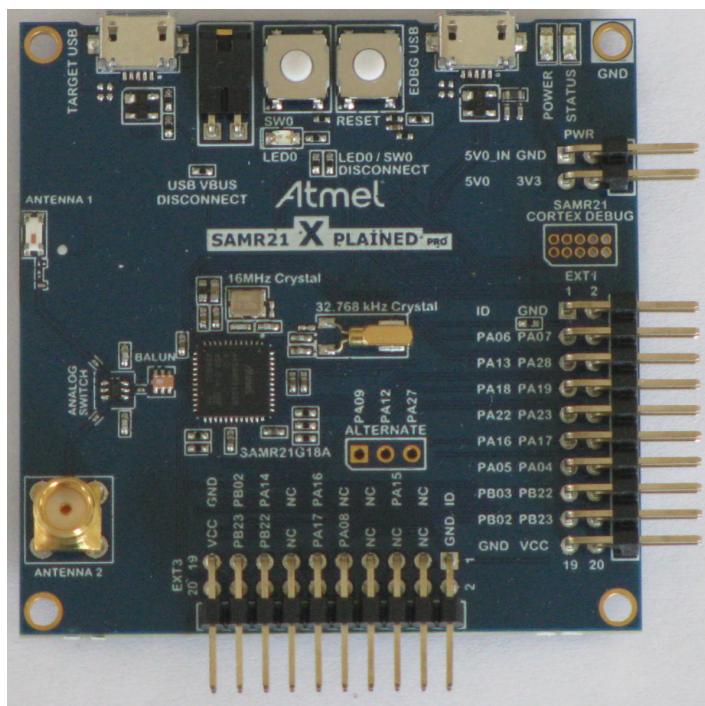


Abbildung 1: Vorderseite des SAM R21 Xplained Pro Evaluation KitBoard

Sensoren

Für die Laufzeitmessung können verschiedene Sensoren verwendet werden. Im Folgenden wird ein Vergleich zwischen dem Ultraschallsensor HC-SR04 und dem SparkFun Sound Detector Sensor vorgenommen. Ein Vergleich ist notwendig, weil beide Sensoren in der Entwicklung zum Einsatz kamen, aber nur einer die nötige Performance für die Positionsbestimmung leistet.



Abbildung 2: Vorderseite des HC-SR04 Sensors

Ultraschallsensor

Für die Positionsbestimmung ist der Ultraschallsensor HC-SR04 (Abbildung 2) zum Einsatz gekommen [4]. Aus den folgenden Gründen wurde sich für den HC-SR04 Sensor entschieden: weite Verbreitung im Arduino/Raspberry-Pi-Bereich, kompakte Bauweise, geringe Anschaffungskosten und einfache Ansteuerung.

In der Regel werden Ultraschallsensoren verwendet, um Distanzen zu einem Gegenstand zu ermitteln. Dabei wird ein Ultraschallsignal ausgesendet, welches von dem Gegenstand zurück reflektiert und wieder empfangen wird. 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 400 cm. Der Vorteil von diesem Ultraschallsensor ist, dass er ohne weitere Sensoren auskommt. Allerdings weißt er auch 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 °C, was eine Positionsbestimmung stark beeinträchtigt. Aufgrund der oben genannten Nachteile wurde sich in der Endanwendung gegen diesen Sensor entschieden.

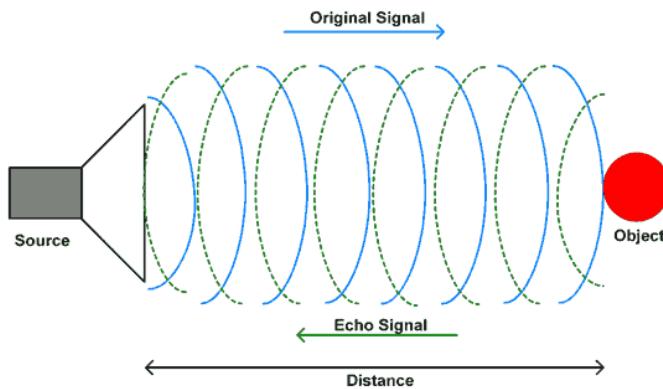


Abbildung 3: Funktionsweise von Ultraschallsensoren

Source: http://ceg.annauniv.edu/internship/2018/intern_one/ECE/ECE5.pdf

Sound Detector

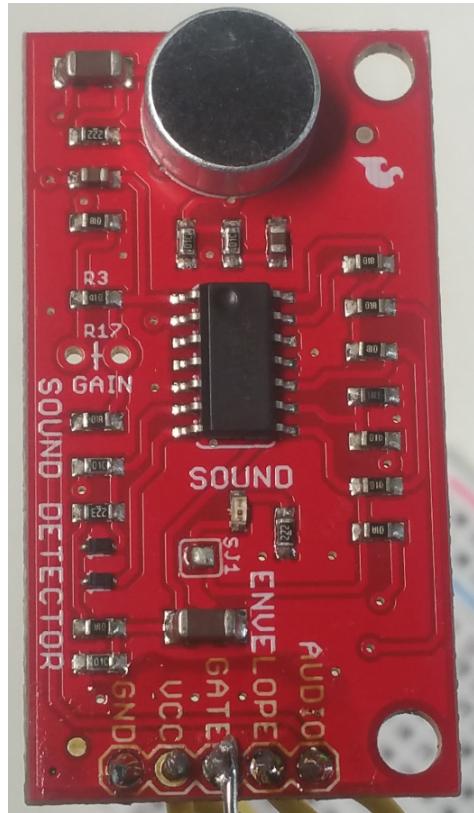


Abbildung 4: Sparkfun Sound Detector

Abbildung 4 zeigt den SparkFun Sound Detector. Der Sensor kann hörbaren Schall detektieren und darüber hinaus die Empfindlichkeit durch das Einlöten eines Widerstandes erhöhen oder verringern. Der Vorteil gegenüber dem Ultraschallsensor liegt darin, dass der Schall sich

kugelförmig ausbreiten kann. Somit muss der Sensor nicht auf eine Richtung ausgerichtet werden. Allerdings stellen Hindernisse für einen hörbaren Schall ein Problem dar. Deswegen sollten keine oder nur wenige Hindernisse auf der Ebene vorhanden sein. Die Reichweite kann über eine Amplitudenveränderung des Tongebers variiert werden. Für die Ansteuerung des SparkFun Sound Detector gibt es die Möglichkeit, den digitalen Ausgang GATE zu verwenden. Neben diesem Ausgang gibt es noch weitere Ausgänge, allerdings werden diese nicht verwendet [5]. 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. Abbildung 5 zeigt ein Überschreiten des Schallpegels (Pegelwechsel).

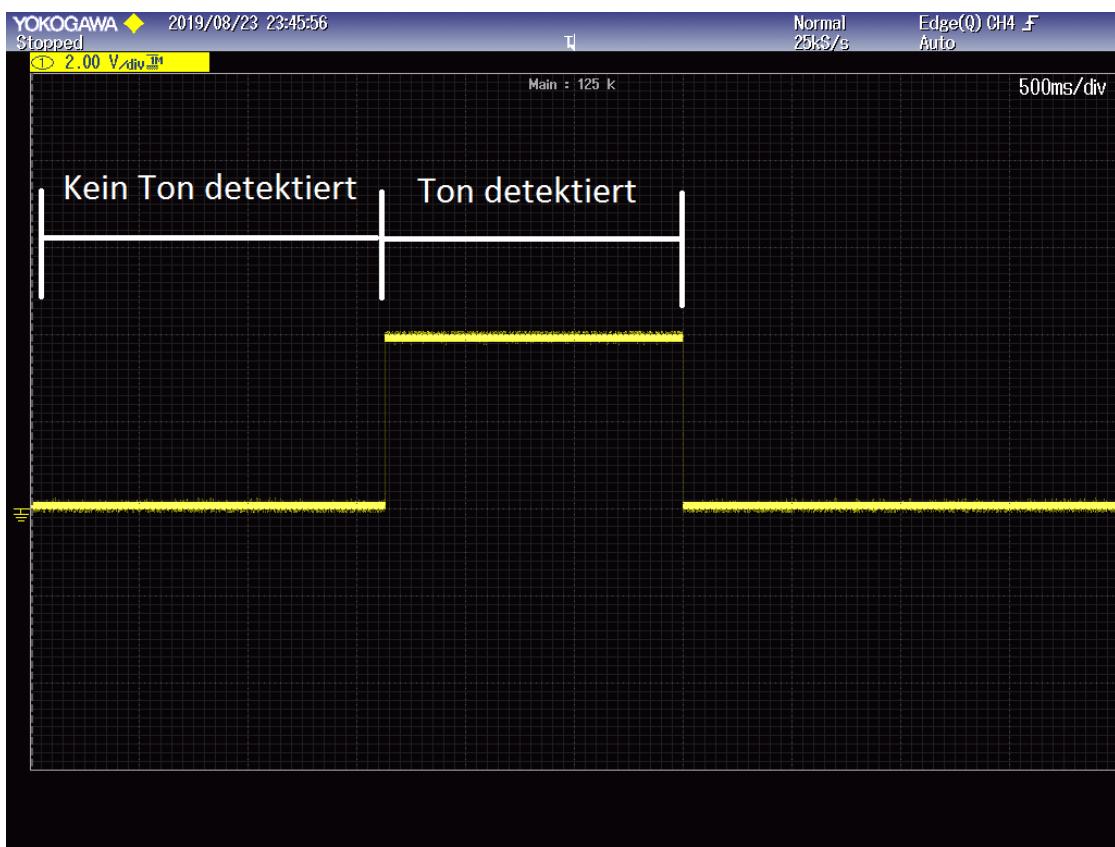
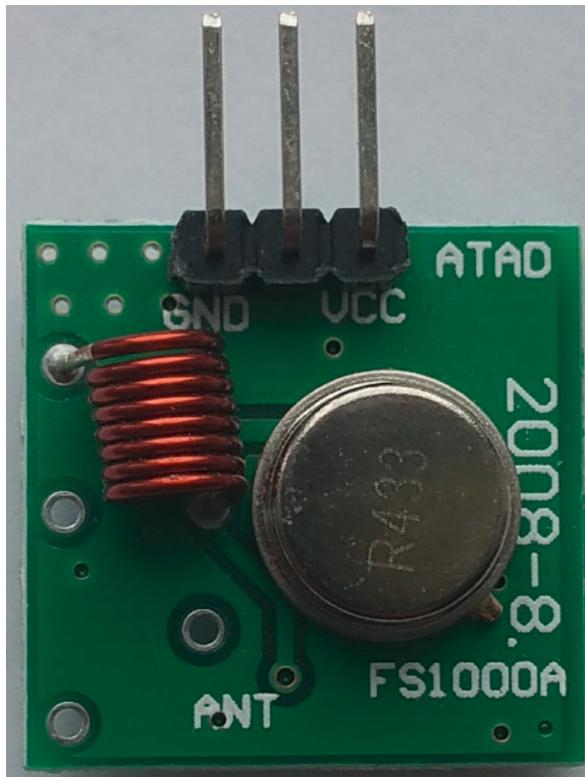


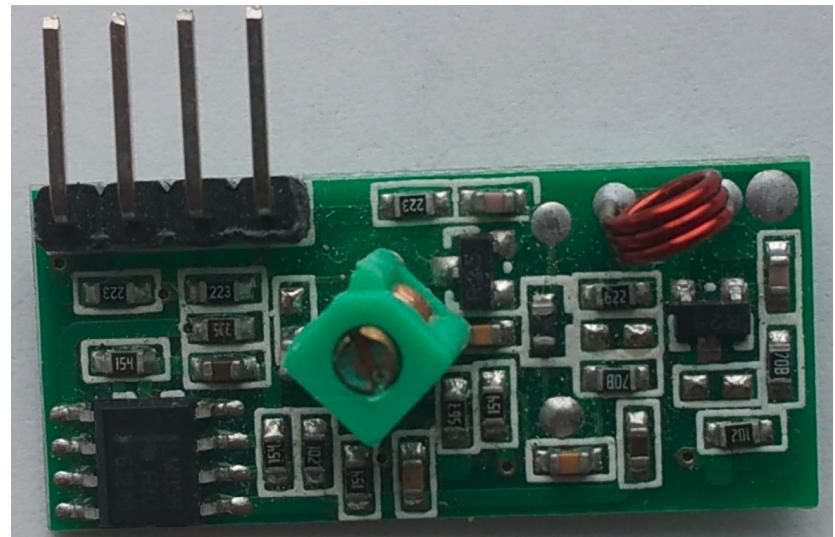
Abbildung 5: Pegelwechsel des GATE–Ausgang

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. Die Arbeit verwendet diesen Sensor, um ein Startsignal zu senden. Im Verlauf der Anwendung hat sich allerdings gezeigt, dass diese Variante nicht fehlerfrei funktionierte. Abbildung 6 zeigt ein Foto des 433 MHz Funk Transmitter-Receiver.



(a) Funksender



(b) Empfänger

Abbildung 6: 433 MHz Funk Transmitter-Receiver

Es folgt eine Auflistung der Anschlüsse des Funksenders (a) (von links aus gezählt):

GND	Masse
DATA	Payload
VCC	Versorgungsspannung

Der Empfänger (b) besitzt vier Anschlüsse. Diese sind wie folgt (von links aus gezählt):

GND	Masse
DATA	Payload
DATA	Payload
VCC	Versorgungsspannung

Mit einem Pegelwechsel des Funksenders 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 handelsüblicher 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 Mosfets und dem Lautsprecher. Abbildung 7 zeigt die Schaltung. Wenn an dem GATE-Eingang des Mosfet eine Spannung von 2 bis 4 V anliegt, schaltet der Mosfet durch, und der Lautsprecher wird mit Strom versorgt.

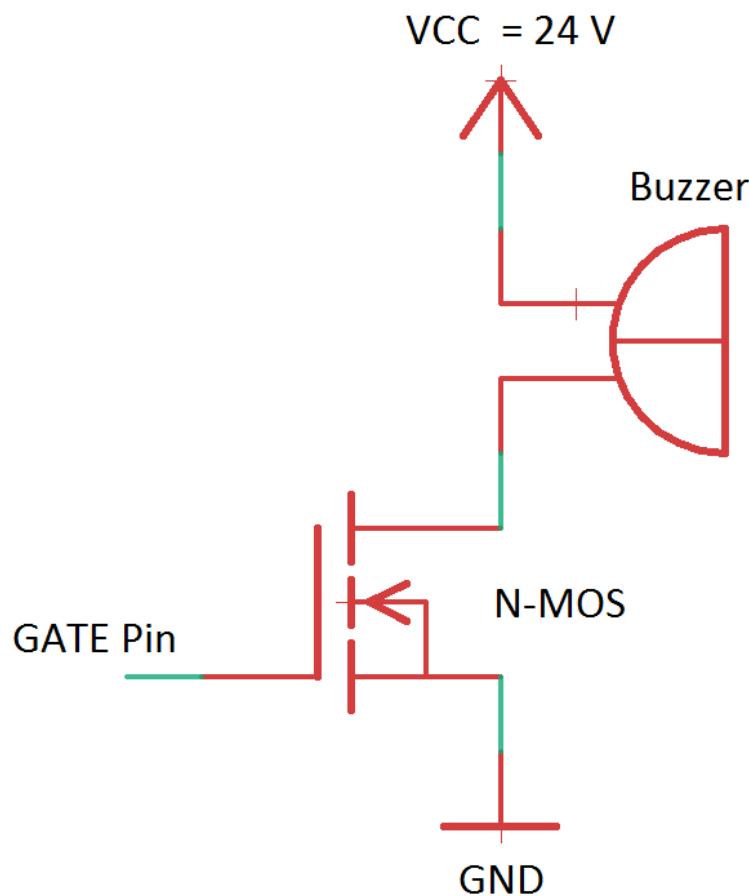


Abbildung 7: Ansteuerung des Lautsprechers

2.2 Verwendete Software

RIOT OS

Um das SAM R21 Xplained Pro Evaluation Kit in Betrieb nehmen zu können, 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 – dies war die Geburtsstunde von RIOT. RIOT ist ein Betriebssystem für "Internet of Things"- Anwendungen. Es 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, weshalb es sich perfekt als Betriebssystem für diese Arbeit eignet [8]. 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 gegenüber anderen Netzwerkprotokollen einfach aufgebaut. Es arbeitet verbindungslos, d.h. der Sender bekommt keine automatische Meldung, ob das gesendete Paket angekommen ist. Ein weiteres Netzwerkprotokoll zur Datenübertragung ist TCP. TCP steht für "Transmission Control Protocol" und es arbeitet im Gegensatz zu UDP verbindungsorientiert.

UDP hat 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 in ein Headerfeld und ein Datenfeld unterteilt. Die Größe des Headers

sind immer 8 Byte. Es folgt eine Auflistung der Komponenten, aus denen das UDP-Paket besteht [10]:

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

2.5 Netzwerk-Socket

Ein Socket ist eine Schnittstelle, die vom Betriebssystem bereitgestellt wird. Es verbindet einen Kommunikationsendpunkt mit dem Betriebssystem. Über ein Socket kann ein Programm, welches eine Datei ist, auf den Kommunikationsendpunkt zugreifen. Wenn Netzwerkdaten empfangen werden, liegen diese zur Abholung im Socket bereit. UDP-Sockets sind immer an einen Port gebunden. Dadurch weiß das Betriebssystem, welche Pakete zu welchem Socket gehören. Neben UDP-Sockets gibt es noch RAW-Sockets. RAW-Sockets 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 genutzt werden. Für die Positionsbestimmung im Indoorbereich existieren mehrere Verfahren. Dazu zählen Messungen des Einfalls winkels, Signalstärkemessungen oder Laufzeitmessungen. Da bereits entschieden worden ist, dass wir Schall für die Positionsbestimmung nehmen, wird das Verfahren der Laufzeitmessung verwendet. Dabei wird die Zeitdifferenz zwischen Sende- und Empfangszeit ermittelt, sodass man die Signallaufzeit erhält [12]. Zusammen mit der Ausbreitungsgeschwindigkeit von Schall ($343,2 \frac{\text{m}}{\text{s}}$ bei 20°C), kann somit die Distanz über die folgende Formel 2.1 berechnet werden:

$$\text{Distanz [m]} = (\text{Empfangszeit} - \text{Sendezzeit}) [\text{s}] \cdot \text{Ausbreitungsgeschwindigkeit} \left[\frac{\text{m}}{\text{s}} \right] \quad (2.1)$$

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 werden kann, muss der

Zielknoten (Master) wissen, zu welchem Zeitpunkt der Accesspoint (Slave) den Schall aussendet. Ist dies nicht gewährleistet, müsste der Master raten. Damit dies nicht geschehen 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 kleine hierarchielose Netzwerke entwickelt worden – es gibt hierbei keine Hierarchien wie beim Network Time Protocol (NTP). Der Vorteil liegt darin, dass PTP nicht wie NTP mit jeder Hierarchie Genauigkeit verliert. Es spezialisiert sich auf kleine Netzwerke ohne Hierarchien. Um eine Genauigkeit im Nanosekundenbereich zu erreichen, muss die aktuelle Systemzeit kurz vor dem Absenden des Pakets hinzugefügt werden. Je geringer die Verzögerung zwischen dem Funktionsaufruf `getSystemTime()` und dem Absenden des Pakets ist, desto genauer wird die Zeitsynchronisation. Abbildung 9 zeigt, welchen Nachrichtenaustausch für die Zeitsynchronisation nötig ist. Der Master ist der Zeitgeber und der Slave synchronisiert seine Zeit. Zuerst wird bei PTP die Laufzeitverzögerung ermittelt. Danach kommt es zur eigentlichen Synchronisation der Zeit. Für die Laufzeitverzögerung sendet zuerst 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 nicht präzise. Mit einer FOLLOW_UP-Nachricht werden diese Probleme gemildert. Dabei enthält die FOLLOW_UP-Nachricht den Zeitstempel t_0 . Nach der FOLLOW_UP-Nachricht weiß der Slave, wie groß die Laufzeitverzögerung ist. Die FOLLOW_UP-Nachricht kann auch weggelassen werden, dafür muss allerdings der Master den Zeitstempel t_0 weit genug in der Zukunft bestimmen, sodass die Vorverarbeitung der SYNC-Nachricht abgeschlossen werden kann. Dann ist der Zeitpunkt des Aussenden der SYNC-Nachricht exakt gleich mit dem Zeitstempel t_0 – diese Arbeit verwendet dies allerdings nicht, da die Genauigkeit der Systemzeit nicht exakt ist.

Die FOLLOW_UP-Nachricht wird mit dem Zeitstempel t_0 versehen und von dem Master ausgesendet. Somit ist nun die Laufzeitverzögerung bekannt. Um nun den Offset zu bestimmen, sendet der Slave eine DELAY_REQ-Nachricht – ohne Inhalt. Dabei wird der Zeitstempel t_2 beim Absenden des Slave festgehalten. Der Master empfängt die DELAY_REQ-Nachricht und speichert seinen Zeitstempel in der Variable t_3 . Darauf wird mit einer DELAY_RESP-Nachricht mit dem Inhalt des Zeitstempels t_3 geantwortet. Der Slave empfängt die Nachricht und speichert bei Empfang in t_4 seinen Zeitstempel. Nun kann mit den folgenden zwei Formeln (2.2) die Zeitdifferenz berechnet werden. Dabei entspricht τ_{prop} der Laufzeitverzögerung und Omega (Ω) ist der Zeitunterschied zwischen Master und Slave [13].

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

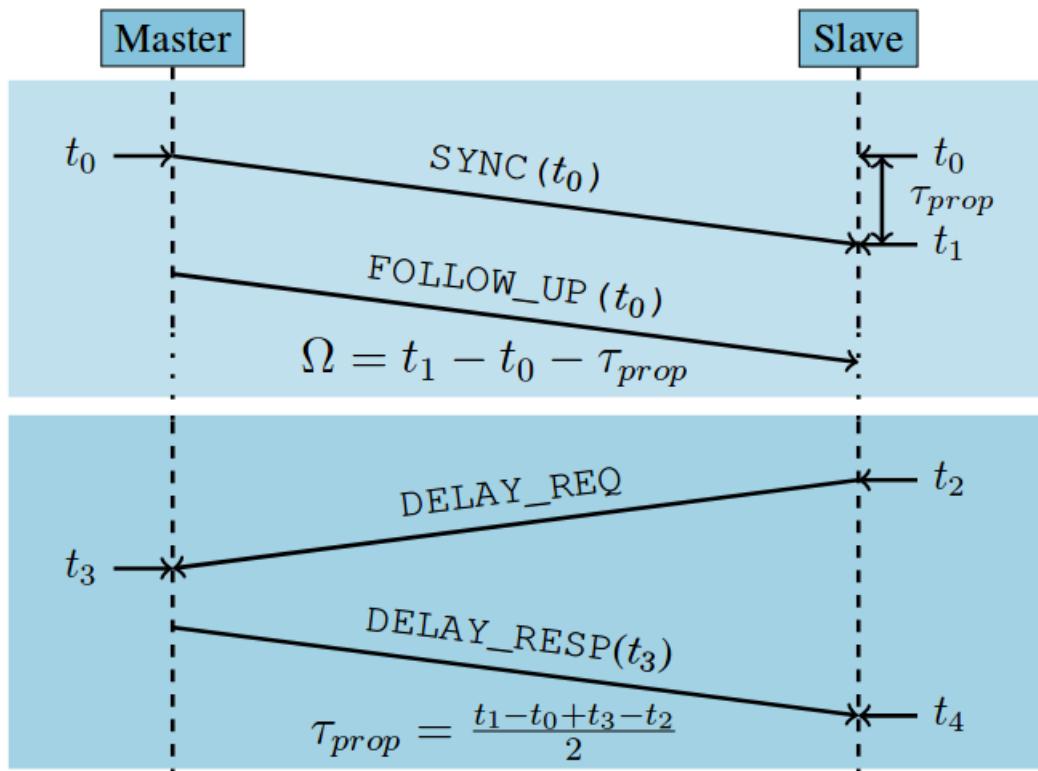


Abbildung 9: Nachrichtenaustausch in PTP

Source: https://www.ibr.cs.tu-bs.de/oa/vonzengen_ICIT2017.pdf

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

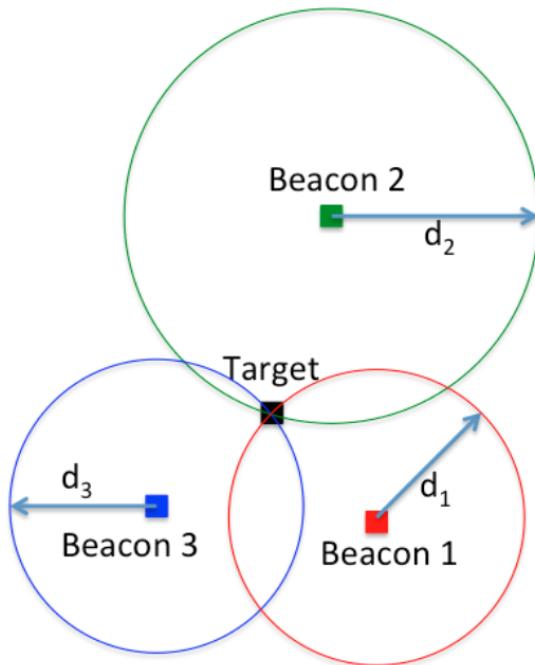


Abbildung 10: Prinzip der Positionsbestimmung

Source:

https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick_OKeefe_F1.pdf

Aufgrund von Schwankungen kann es vorkommen, 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:

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

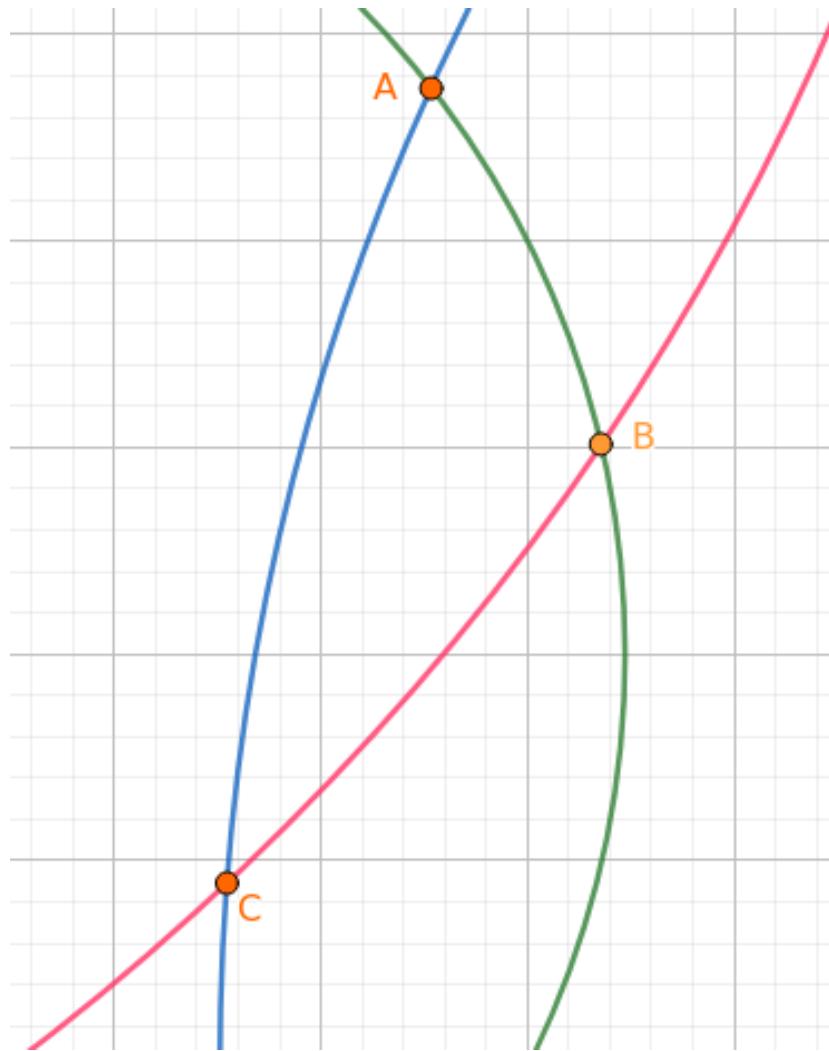


Abbildung 11: Bereich indem sich der Zielknoten befinden muss

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 (Seite 43) aufgeführt.

3 Entwurf

Dieses Kapitel befasst sich mit der Struktur der Implementierung. Damit wird die Herangehensweise beschrieben und es wird erläutert, wieso bestimmte Entscheidungen getroffen wurden.

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 für 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.

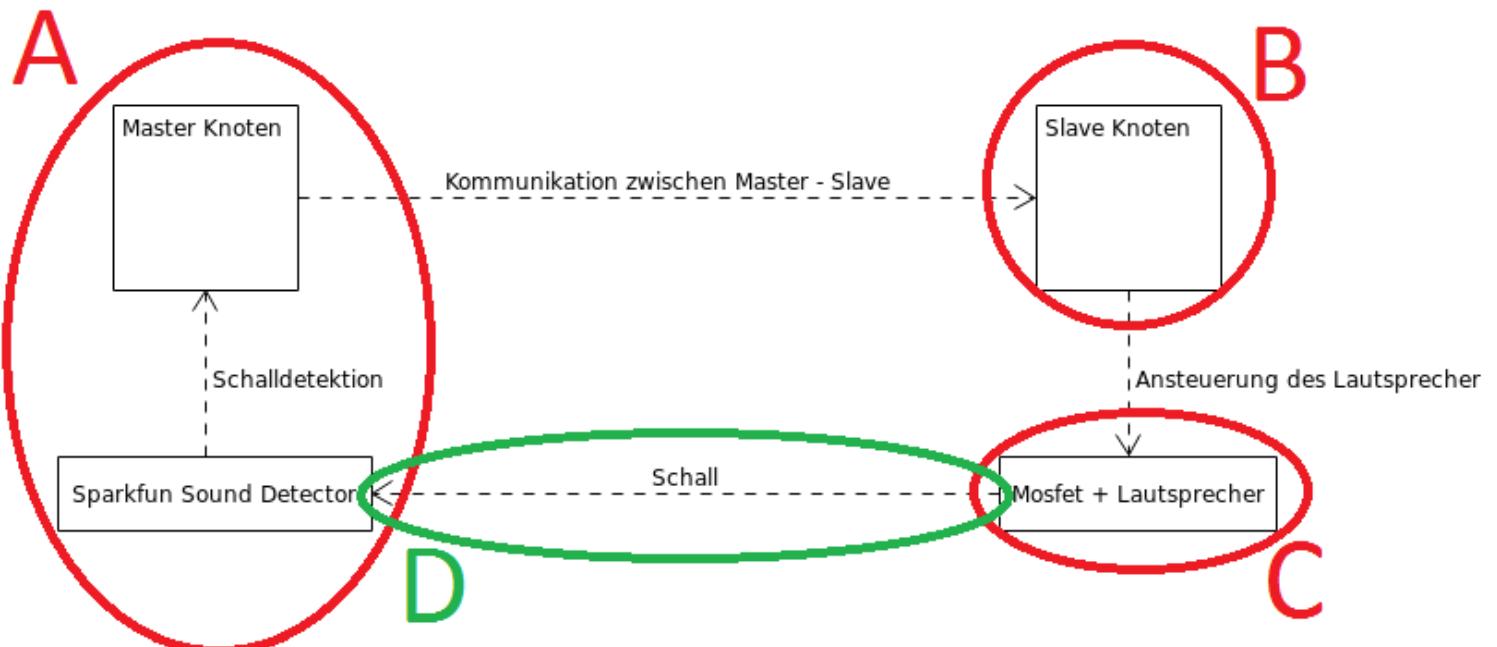


Abbildung 12: Versuchsaufbau, unterteilt in Module

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 Systemzeit ausliest. Sobald der Ton das Mikrofon passiert, wird diese Funktion aufgerufen. Im Rahmen des Funktionsaufrufes entsteht eine zeitliche Abweichung mit einer hohen Relevanz, da mehrere CPU-Zyklen benötigt werden, bevor die Systemzeit in der Variable gespeichert wird. 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 jeweiligen Modulen ist der GATE-Pin bei dem Master an dem Pin PB23 angeschlossen. Für den Slave wird immer Pin PA18 verwendet. Abbildungen 13 und 14 zeigen die Verdrahtungen.

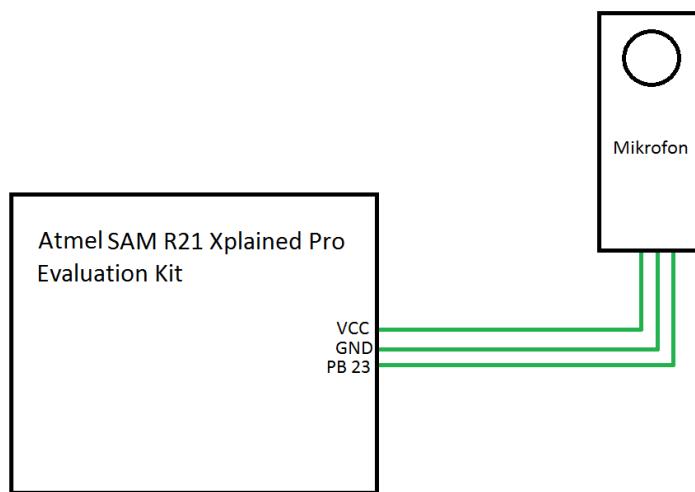


Abbildung 13: Verdrahtungsplan Master

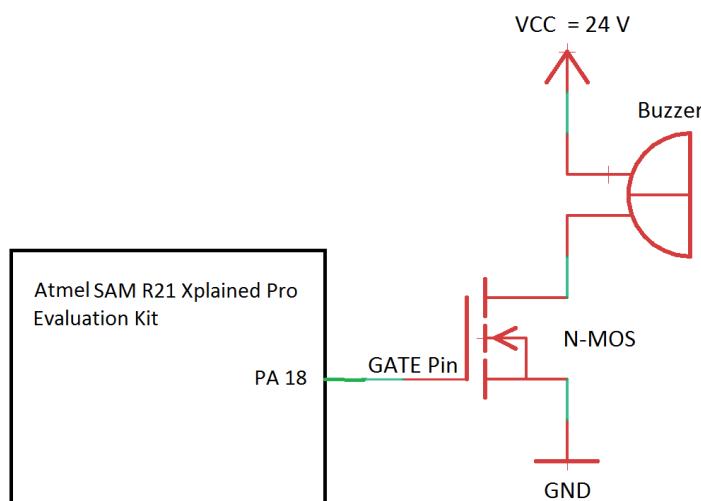


Abbildung 14: Verdrahtungsplan Slave

4 Implementierung

Dieses Kapitel widmet sich der konkreten Implementierung.

4.1 SparkFun Sound Detector

Bevor eine Messung mit dem Board durchgeführt worden ist, wurde zuerst ein Plausibilitätscheck vorgenommen. Dabei überprüft man, ob die beiden SparkFun Sound Detector Mikrofone korrekte Ergebnisse liefern. Der Aufbau ist durch das folgende Blockschaltbild 15 abgebildet. Abbildung 16 zeigt den Aufbau im Laborraum.

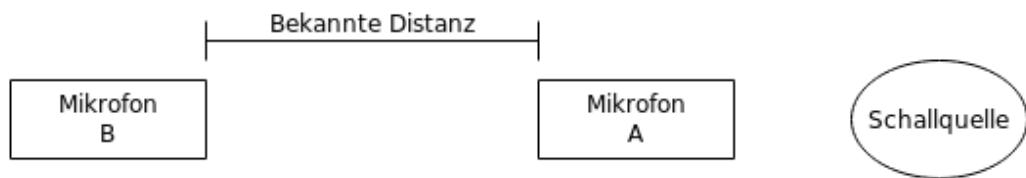


Abbildung 15: Versuchsaufbau als Blockschaltbild

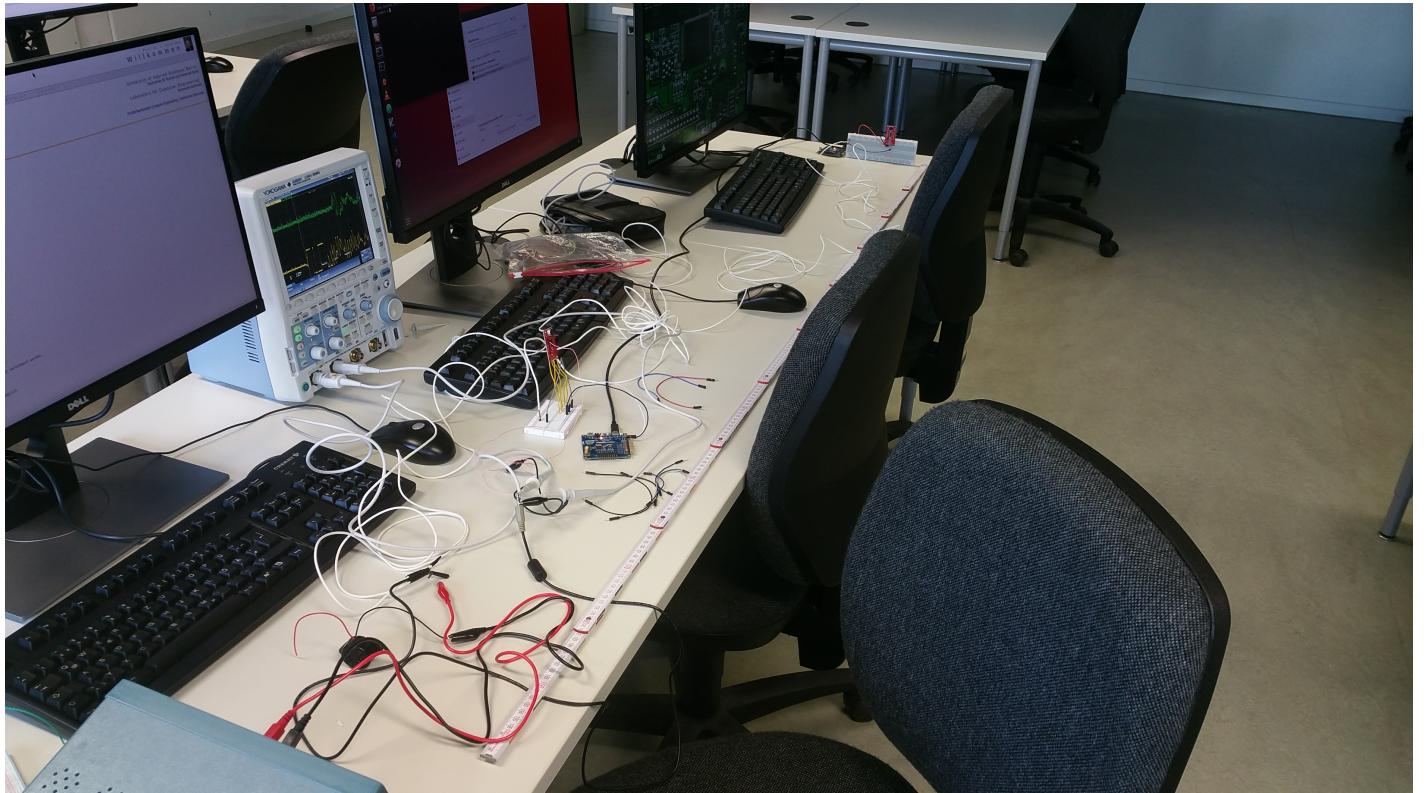


Abbildung 16: Versuchsaufbau

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 er zuerst das Mikrofon A und dann mit einer Verzögerung Mikrofon B. Der Abstand der beiden Mikrofone wurde daraufhin mit dem Oszilloskop überprüft. In 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 Mikrofon B. Es ist deutlich erkennbar, dass eine Verzögerung vorhanden ist. Die beiden GATE-Ausgänge liegen ungefähr 2863 μ s auseinander. Mit einer Ausbreitungsgeschwindigkeit von $0,034 \frac{\text{cm}}{\mu\text{s}}$ entspricht das ungefähr 97,342 cm. Der gemessene Abstand beträgt 100 cm. Diese Auswertung zeigt, dass die Verzögerung nur minimal von dem gemessenen Abstand abweicht. Es folgen weitere Messungen, bei dem anstatt eines Klatschens der verwendete Tongeber genommen wird. Dabei sitzt der Lautsprecher direkt neben einem der beiden Mikrofone. Das zweite Mikrofon wird in Abständen von 1 m, 2 m und 3 m aufgestellt. Es folgen drei Tabellen (1 - 3) mit den Messwerten (Arithmetisches Mittel).



Abbildung 17: Verzögerung der Schallausbreitung

Tabelle 1: Messwerte bei 1 m Entfernung

1 m Entfernung			
Messwert von Mikrofon 1 [μs]	Messwert von Mikrofon 2 [μs]	Differenz [μs]	Distanz [cm]
595,00	4345,00	3750,00	127,50
603,00	4340,00	3737,00	125,05
461,00	4440,00	3979,00	135,28
608,00	4560,00	3952,00	134,36
641,00	4670,00	4029,00	136,98
443,00	4400,00	3957,00	134,53
450,00	4415,00	3965,00	134,81
634,00	4335,00	3701,00	125,83
638,00	4425,00	3787,00	128,75
640,00	4850,00	4210,00	143,14
Durchschnitt			
571,30	4036,50	3511,00	132,62

Tabelle 2: Messwerte bei 2 m Entfernung

2 m Entfernung			
Messwert von Mikrofon 1 [μs]	Messwert von Mikrofon 2 [μs]	Differenz [μs]	Distanz [cm]
476,00	20150,00	19674,00	668,91
458,00	37900,00	37442,00	1273,02
471,00	36150,00	35679,00	1213,08
681,00	38750,00	38079,00	1294,68
452,00	38150,00	37698,00	1281,73
466,00	34150,00	33684,00	1145,25
675,00	38750,00	38075,00	1294,55
450,00	33550,00	33100,00	1125,40
447,00	32600,00	32153,00	1093,20
460,00	35800,00	35340,00	1201,56
Durchschnitt			
503,60	34595,00	34072,40	1159,13

Tabelle 3: Messwerte bei 3 m Entfernung

3 m Entfernung			
Messwert von Mikrofon 1 [μs]	Messwert von Mikrofon 2 [μs]	Differenz [μs]	Distanz [cm]
454,00	33450,00	32996,00	1121,86
428,50	35800,00	35371,50	1202,63
474,00	37650,00	37176,00	1263,98
429,00	33500,00	33071,00	1124,41
437,50	37700,00	37262,50	1266,92
439,50	35550,00	35110,50	1193,75
435,50	25200,00	24764,50	841,99
437,50	23800,00	23362,50	794,32
430,00	24950,00	24520,00	833,68
427,50	25800,00	25372,50	862,66
Durchschnitt			
439,30	31340,00	30900,70	1050,62

Aus den obigen Tabellen ist zu erkennen, dass Mikrofon 1 mindestens 439,30 μs benötigt, um den Ton wahrzunehmen. Allerdings steht das Mikrofon 1 direkt neben dem Tongeber. Durch Latenzen beim Mikrofon ist dieser Messwert nicht null. Allerdings nimmt die Latenz mit der Entfernung ab, woraus keine konstante Abweichung entnommen werden kann. Weiterhin 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, da die Abweichungen irregulär ansteigen.

4.2 Modul A

Das Modul A untersucht die Verzögerung des Masters vom Detektieren eines Mikrofons zu dem Aufruf 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. Dabei besitzt der Slave den Lautsprecher. Es wird ein PIEP-Ton erzeugt. Master und Slave agieren unabhängig voneinander. Das Oszilloskop wurde an dem GATE-Ausgang des Mikrofons sowie an dem Pin angehängt, der in der Interruptroutine auf HIGH gesetzt worden ist. Darüber hinaus ist die Schallquelle dreimal versetzt worden, um Abweichungen abhängig von der Distanz vornehmen 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 [μs]		
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 (arithmetisches Mittel) 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 vernachlässigt werden.

4.3 Modul B

Dieses Modul untersucht die Abweichungen vom Aussenden eines HIGH-Signals des Masters 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. Für dieses Modul wird kein Lautsprecher und kein Mikrofon benötigt.

Es wird angenommen, 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

Messwert [μs]	
Direkte Verbindung	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
Durchschnitt	
82, 46	82, 63

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 weg gerechnet werden.

4.4 Modul C

Das Modul C soll die Anstiegszeit des N-Mosfets ermitteln. Die Anstiegszeit liegt laut 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 $0,000034 \frac{\text{cm}}{\text{ns}} \cdot 40 \text{[ns]} = 0,00136 \text{[cm]}$. Diese Abweichung ist zu gering, um Einfluss auf das Ergebnis 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 Detectors gemessen. Der Master-knoten wird nicht benötigt. Der Slaveknoten besitzt den Lautsprecher und den dazugehörigen Mosfet. Er schaltet den Lautsprecher für $40000 \mu\text{s}$ an. Theoretisch muss der errechnete Wert aus der Distanz und der Schallgeschwindigkeit den gemessenen Werten auf dem Oszilloskop entsprechen. Messungen werden mit den Distanzen 1 m, 2 m und 3 m vorgenommen. 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 vorherigen Modulen (A-C) ein konstanter Fehler zu erkennen war und das Modul D ihn nicht aufweist, muss die Abweichung des Endergebnisses zwischen dem Lautsprecher und dem Mikrofon liegen. Eine Abweichung, wie in Modul D macht die Positionsbestimmung im Zentimeterbereich unmöglich. Die Ergebnisse aus Modul A lassen sich bei diesen Messungen bestätigen. Dadurch verfestigt sich der Verdacht, dass der Fehler beim Mikrofon oder beim Lautsprecher und dem Raum dazwischen liegen muss. Eine Vermutung für diesen größer werdenden Fehler ist, dass der Schall durch die Gegenstände im Raum abgelenkt wird und deshalb später beim Mikrofon ankommt. Weiterhin kann vielleicht durch eine Änderung der Tonfrequenz ein genauereres Ergebnis mit weniger Fehlern produziert werden.

4.6 Systemzeit

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

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

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

Aus den Messwerten erkennt man deutlich, dass der Aufruf nahezu konstant ist. Auch diese Abweichung kann im Endergebnis ausgeglichen werden. Bei diesem Aufruf werden verschiedene Funktionen ausgeführt. Eine Vermutung, warum dieser Funktionsaufruf $7,21\ \mu\text{s}$ andauert, ist, dass der Timer-Wert in Ticks zurückgegeben und danach in $\mu\text{-Sekunden}$ umgewandelt wird. Die Umwandlung von Ticks in $\mu\text{-Sekunden}$, zusammen mit dem Aufruf verschiedener Funktionen, sind für die $7,21\ \mu\text{s}$ verantwortlich. Der Durchschnittswert ist ein Maximalwert des Aufrufs, denn der Durchschnittswert beinhaltet das Setzen und Löschen des GPIOs. Die Messwerte für das Setzen und Löschen des GPIO werden durch Tabelle 10 verdeutlicht.

Tabelle 10: Messwerte für das Setzen und Löschen eines GPIO

Messwert [ns]
418, 00
417, 50
417, 50
416, 50
417, 00
417, 00
417, 00
417, 50
417, 50
417, 50
Durchschnitt
417, 25

Aus dem Durchschnittswert ergibt sich eine Abweichung von $0,014178\ cm$. Der Durchschnittswert 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 mithilfe eines Kabels mit dem Slave verbunden. In der Endanwendung jedoch ist der Master drahtlos mit dem Slave verbunden worden. Bevor die Zeitsynchronisation in die Software eingefügt werden kann, wird geprüft, welche Genauigkeit die Zeitsynchronisation erreicht. Dabei baut man zwei SAM R21 Xplained Pro Evaluation Kit nebeneinander auf. Insgesamt werden hundert Zeitsynchronisationen (PTP) nacheinander durchgeführt. Danach bereitet man die Ergebnisse graphisch auf und wertet sie aus. Es wird nur auf die Variable t_{prop} (Propagation Delay) eingegangen, da diese die Laufzeitverzögerung zwischen Master und Slave entspricht. Die Abbildung 18 zeigt die Laufzeitverzögerung bei 100 Messungen. Da die Messwerte sich im Millisekundenbereich bewegen, führt dies für die Zeitsynchronisation zu falschen Ergebnissen. PTP verlangt für Zeitsynchronisation gleiche Laufzeitverzögerungen. Da diese im Millisekundenbereich schwanken, ist es zu erheblichen Abweichungen bei der Synchronisation gekommen. Neben der Laufzeitverzögerung spielt noch das Hinzufügen der Systemzeit in das Paket eine Rolle. Je später die Systemzeit in das UDP-Paket eingefügt wird, desto weniger Fehler gibt es bei der Zeitsynchronisation. Bei PTP muss die Systemzeit genau dem Aussenden entsprechen; deswegen ist die Systemzeit so spät wie möglich zum Paket hinzuzufügen. Passiert dies nicht, geht Genauigkeit verloren, weil das Zusammenbauen des UDP-Pakets Zeit kostet. Eine Vermutung für die Abweichungen bei der Laufzeitverzögerung ist, dass die Funktion `udp_receive_packet()` durch Polling abgefragt wird. Somit gehen Pakete verloren und werden nur zu einem bestimmten Zeitpunkt entgegengenommen. RIOT bietet keine Interruptroutine für das Empfangen von Paketen an. Somit können Pakete verloren gehen, welches sich kritisch auf die Zeitsynchronisation auswirkt. Weiterhin ist eine kleine Abweichung von Master und Slave nicht zu verhindern. Da jeder Frequenzgeber schwingt, kommt nicht jeder Taktpiegel exakt genau an. Der Jitter gibt an, um wie viele Nanosekunden der Frequenzgeber abweichen kann. Dieser ist beim Master und Slave minimal unterschiedlich.

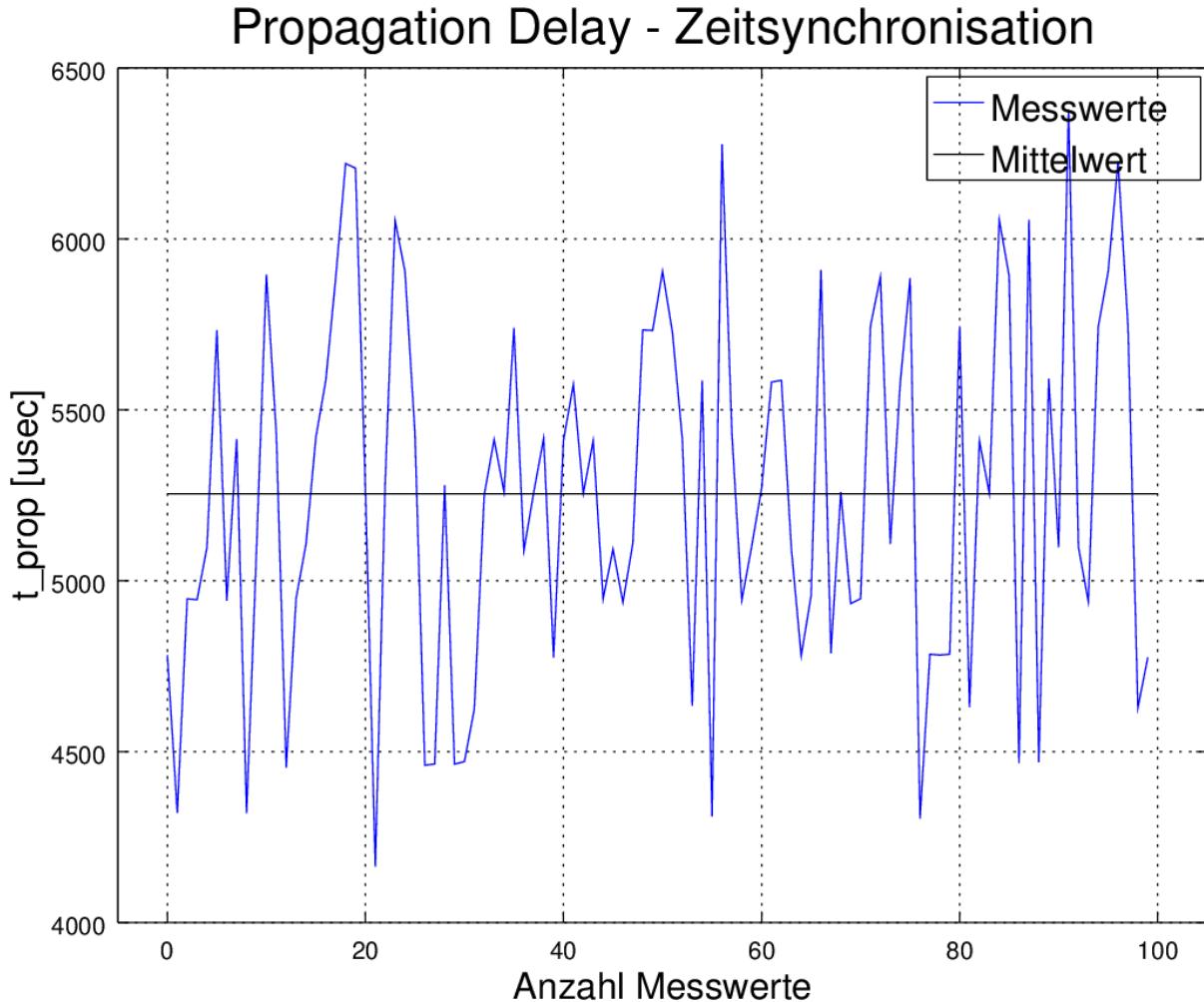


Abbildung 18: Messwerte von t_{prop} bei hundert Messungen

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 verbundenen Pin. Das Oszilloskop sollte nun diese Frequenz widerspiegeln. Das Messergebnis ist in Abbildung 19 dargestellt. Das Mikrofon erkennt nur ein Rauschen. Dies bestätigen auch die vielen kurzen LOW-Pegel. Als vom Sender ein LOW-Pegel übertragen wurde, konnte nicht eindeutig bestimmt werden, ob das Signal angekommen war oder durch das Rauschen unterging. 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.

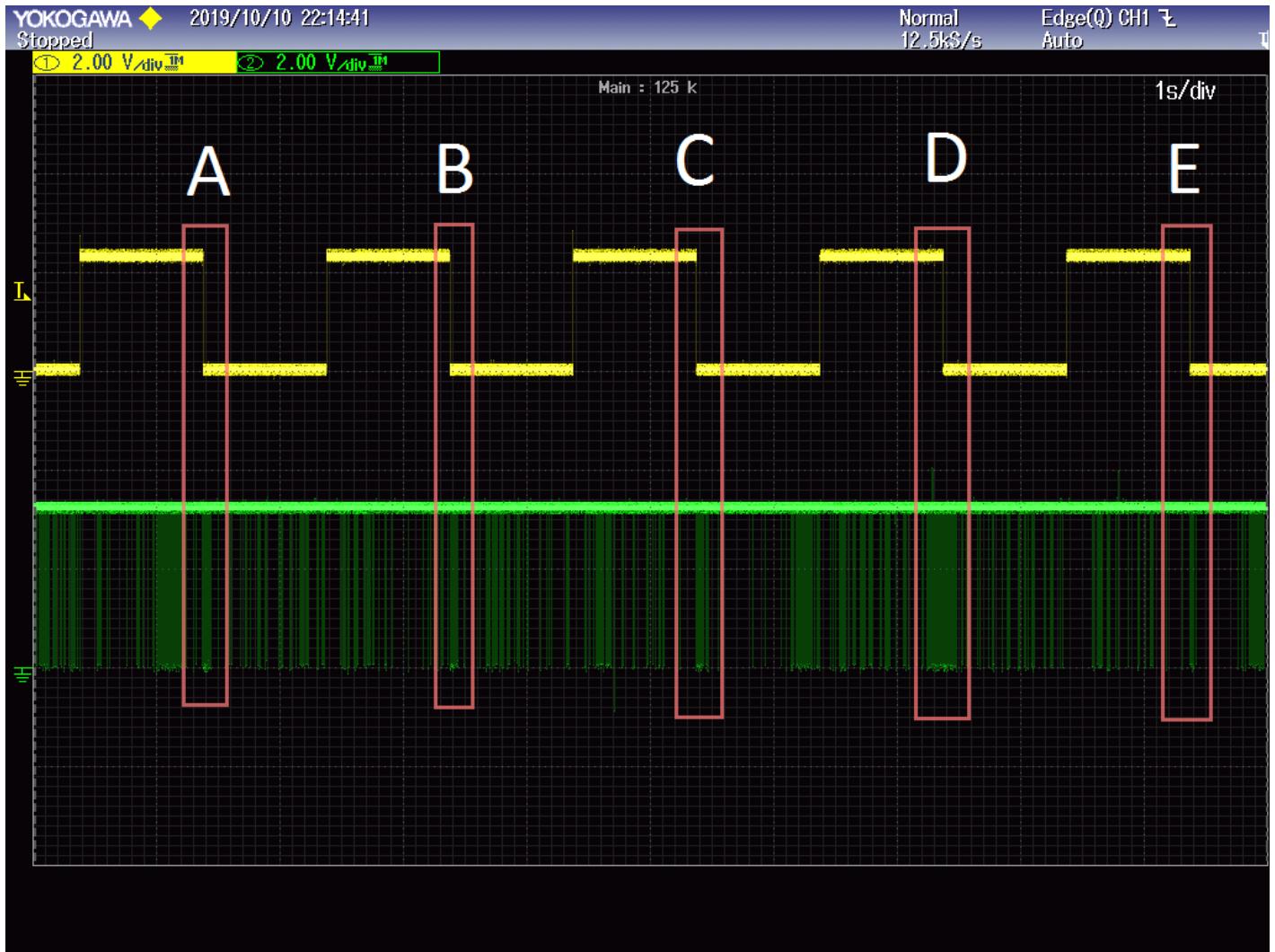


Abbildung 19: DATA-Pin des 433 MHz Funk Transmitter-Receiver Empfänger

4.9 Software

Für die Positionsbestimmung verhält sich die Software nach dem Programmablaufplan (Abbildung 20). Zuerst wird der Zeitunterschied ausgeglichen, der zwischen dem Master und dem 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 $40000 \mu\text{s}$ einschaltet. Über das Mikrofon beim Master wird der Ton empfangen und sofort die Systemzeit bestimmt. Über die Differenz vom Aussenden und Empfangen kann die Distanz berechnet werden. 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 folglich die Position des Masters. Dabei ist zu beachten, dass 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 (pepebecker/circle-intersection) entnommen [15]. Wird die Software nicht abgebrochen, wiederholt sich die Messung für alle Slaves. Somit kann sich der Master auf der Ebene bewegen und bekommt seine Postion angezeigt. Es muss berücksichtigt werden, dass durch den Jitter des Frequenzgebers die Systemzeit der Slaves mit der Zeit divergiert, d.h, wartet man entsprechend lange, ist die Systemzeit der Slaves nicht mehr übereinstimmend mit dem Master. Deswegen muss regelmäßig eine Zeitsynchronisation erfolgen.

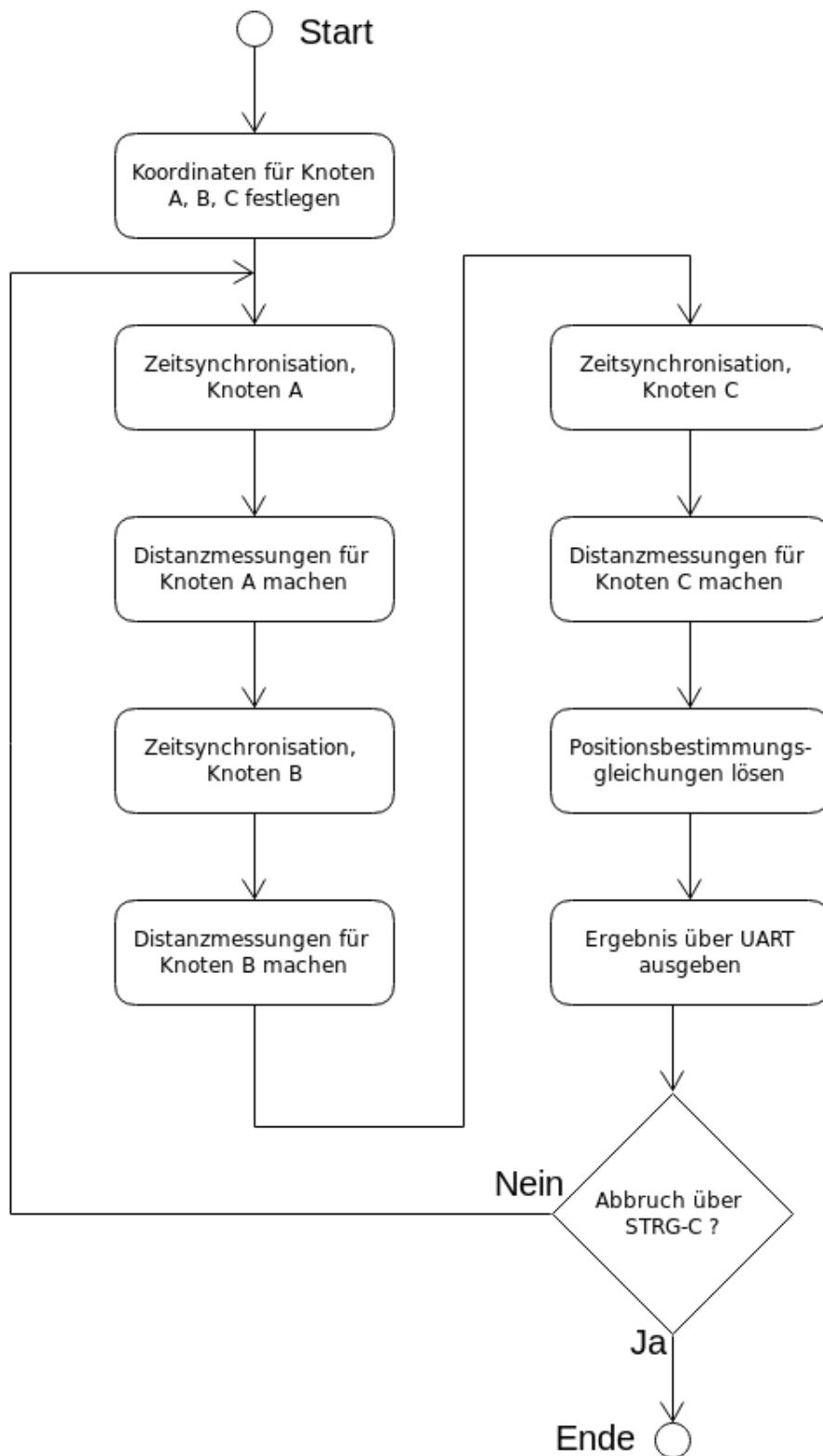


Abbildung 20: Programmablaufplan der Software

5 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 sind.

5.1 Quadratische Gleichung

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

$$x^2 + p \cdot x + q = 0 \quad (5.6)$$

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 lässt sich erkennen, dass die Funktion korrekte Rückgabewerte liefert. Bei einer fehlerhaften Eingabe gibt die Funktion ein NaN (Not a Number) zurück.

5.2 Abstand zweier Punkte

Bei der Positionsbestimmung kann es dazu kommen, dass die drei Kreise sich nicht treffen. Dann haben wir zwei Bereiche, indem sich der Knoten befindet. Um den korrekten Bereich zu bestimmen, wird die Funktion *determine_point_radius()* benötigt. Liefert diese Funktion ein falsches Ergebnis, führt das dazu, dass die Positionsbestimmung nicht mehr korrekt ist. Der korrekte Bereich kann jedoch dann bestimmt werden, wenn von zwei Punkten die Distanz zu einem Kreismittelpunkt ermittelt 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. Abbildung 22 verdeutlicht das Problem.

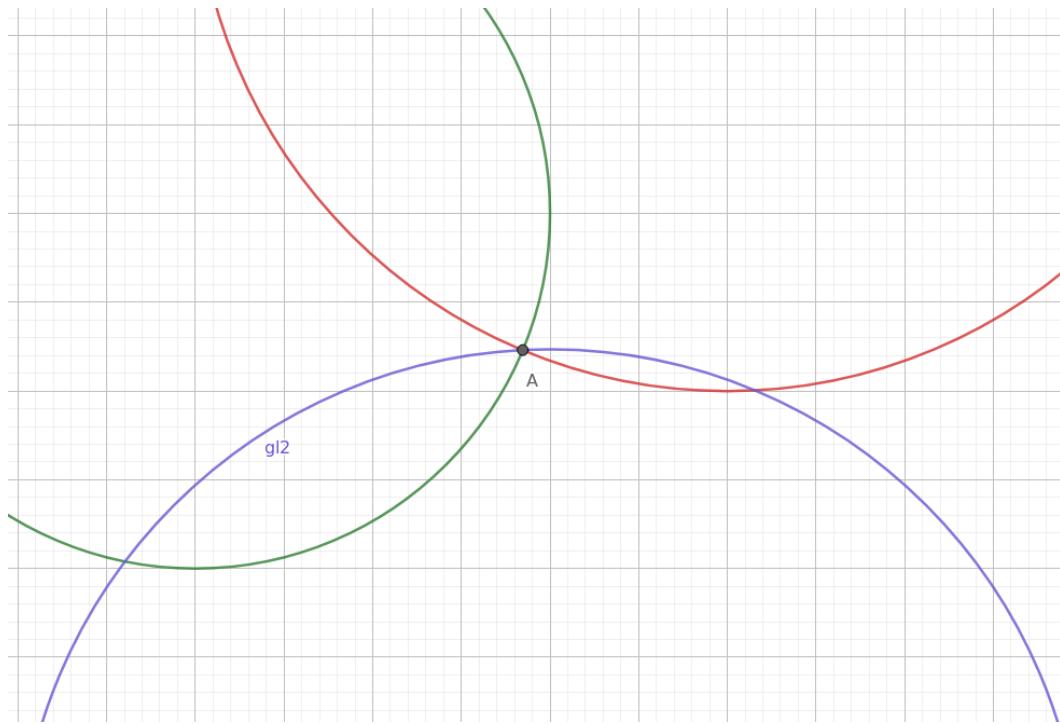


Abbildung 21: Positionsbestimmung ohne Schwankung

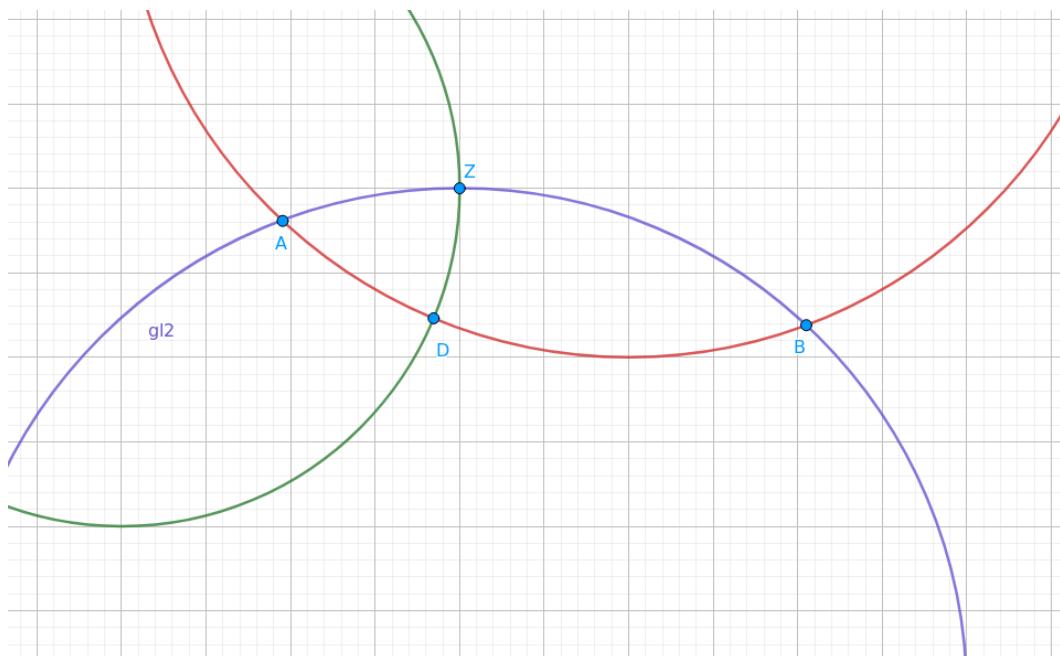


Abbildung 22: Positionsbestimmung mit Schwankung

In der Abbildung 22 ist erkennbar, dass es durch die Schwankung zwei Bereiche gibt. Ein Bereich ist 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 aus berechnet. Die

Funktion wird mit den folgenden zwei Punkten und der Kreisgleichung getestet.

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

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

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

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

5.3 Positionsbestimmung

Zur Bestimmung der Position benötigt die Funktion *calc_position()* drei Parameter. Diese sind die drei Kreise, die sich in einem Schnittpunkt schneiden oder einen Bereich aufspannen. Für den Fall, dass 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 von der Funktion zurückgegeben. Die Funktion testet man mit den folgenden Kreisgleichungen: zuerst wird ein Test unternommen, 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 $(4, 8872 | 3, 6618)$. Das Ergebnis gibt auch die Funktion zurück. Als Nächstes wird der Rückgabewert bei keinem gemeinsamen Schnittpunkt 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 Funktion gibt die folgenden Koordinaten der Eckpunkte zurück.

$$P_A : (4, 8872 | 3, 6618)$$

$$P_B : (4, 9832 | 3, 2583)$$

$$P_C : (4, 2794 | 3, 0196)$$

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

6 Auswertung

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

6.1 Hardware

Die aufgetretenen Abweichungen entstanden meistens, wenn die Peripherie Daten generiert hatte bzw. angesprochen wurde. Da viele dieser Abweichungen konstant sind, kann ein Offset in das Endergebnis einfließen. 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 der Tongeber und der SparkFun Sound Detector. RIOT sieht sich zwar als Echtzeitbetriebssystem, allerdings sind zu viele Schichten zwischen dem laufenden Programm und der Hardware vorhanden. Ein Systemaufruf für die Abfrage der Systemzeit dauert $7,21\ \mu\text{s}$. Dies ist zu lange und verfälscht die Systemzeit, die eigentlich $\mu\text{-Sekunden}$ genau sein soll. Des Weiteren zeigt sich, je weniger Komponenten (Peripherie) verwendet werden, umso einfacher ist die Fehlersuche.

6.2 Messergebnis

Die Messergebnisse zeigen, dass eine Positionsbestimmung wie sie durchgeführt wurde, nicht genau genug ist. Die Abweichung, die das Messergebnis hauptsächlich verfälscht, liegt beim Tongeber und dem Mikrofon. Weil kein Muster bei der Abweichung erkennbar ist, kann kein Offset bestimmt werden. Die Verwendung von fehlerbehafteten Bausteinen kann ausgeschlossen werden, da das Oszilloskop für alle verwendeten Mikrofone die gleichen Abweichungen aufzeichnet. Allerdings ist RIOT bei dem Aufruf der Interruptroutinen nahezu konstant.

6.3 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. Allerdings zeigt sich, dass die einzelnen Module perfekt zusammen agieren, auch wenn die Distanzmesswerte fehlerbehaftet sind.

6.4 RIOT

Im Verlauf dieser Arbeit hat sich herausgestellt, dass RIOT eher hinderlich als hilfreich ist. Da zwischen der Hardware und dem ausführbaren Code ein Betriebssystem eingeführt wurde, kommt es zwingend zu weiteren Verzögerungen. Darüber hinaus bietet RIOT nicht die Möglichkeit einer Interruptroutine beim Eintreffen von drahtlosen Nachrichten. Ebenfalls ist das Mitsenden der Systemzeit bei drahtlosen Nachrichten nicht vorhanden. Somit wird wertvolle Zeit verschwendet. Das SAM R21 Xplained Pro Evaluation Kit wird von RIOT

nur rudimentär unterstützt. Somit können nicht alle Funktionen des SAM R21 Xplained Pro Evaluation Kit verwendet werden.

6.5 Zeitsynchronisation

Die vorhandenen Abweichungen bei der Zeitsynchronisation (siehe Kapitel 4.7) kommen dadurch zustande, dass die Zeitsynchronisation in Software implementiert wurde und kein Hardwarebeschleuniger verwendet wurde. Somit kommt es immer zu einer Verzögerung von mehreren Millisekunden. Mithilfe eines Hardwarebeschleunigers können auch diese Verzögerungen eliminiert werden. Weiterhin zeigt sich, dass diese Arbeit alle Anforderungen für den Einsatz des PTP erfüllt.

6.6 Mathematischer Hintergrund

Die Gleichungen, die für eine Positionsbestimmung eines zwei-dimensionalen Raumes verwendet werden, können auch für N-Dimensionen erweiterbar. Pro Dimension kommt eine Gleichung und eine Potenz hinzu und diese müssen gleichgesetzt werden.

7 Probleme

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

7.1 Kommunikation Master – Slave

Der Forschungsansatz 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 nachfolgende Daten dem nicht zugeordnet bzw. korrekt interpretiert werden. Die Kommunikation war serialisiert – die nächsten Daten konnten nur interpretiert werden, wenn die vorherigen Daten fehlerfrei ankamen. Da es häufig zu nicht erklärbaren Verbindungsabbrüchen während der Kommunikation kam, wurde die serialisierte Kommunikationsvariante durch Structs abgelöst. Dabei enthalten die Structs den Steuercode und die dazugehörigen Daten. Somit liegen Steuercodes und Daten zusammen. Sollten nun Daten verloren gehen, können durch den dazugehörigen Steuercode nachfolgende Daten weiterhin interpretiert werden. Die Kombination von Daten und Steuercodes ermöglicht eine variable Kommunikation. Dadurch ist man nicht mehr abhängig von den vorherigen Daten.

7.2 Genauigkeit – Zeitsynchronisation

Im Rahmen der Zeitsynchronisation kam es immer wieder zu dem Problem, dass die Genauigkeit bei mehrfachen Wiederholungen nicht besser wurde. Dadurch, dass sich die Zeitsynchronisation nicht im μ -Sekundenbereich auflöste, konnten auch nur Messungen mit größeren Abweichungen erfolgen. Die Abweichung kam insbesondere durch die Aufrufe `getSystemTime()` und `udp_send_packet()` zustande. Die Systemzeit wurde vor dem Aussenden vom Programm selbst in das Datenpaket eingefügt und nicht von der Firmware des Funksenders. Weiterhin baute die Funktion `udp_send_packet()` zuerst das UDP-Paket zusammen, welches wiederum Zeit kostete, bis es gesendet werden konnte.

7.3 Genauigkeit – Messungen

Theoretisch ist nach einer Zeitsynchronisation bekannt, um wie viel Zeit der Master dem Slave hinterher hängt, bzw. voraus ist. Bei den praktischen Distanzmessungen mit dem Slave kam es dabei immer wieder zu großen Distanzschwankungen. Diese haben vermutlich drei Ursachen: eine nicht im μ -Sekundenbereich auflösende Zeitsynchronisation, eine Verzögerung der Messinstrumente sowie eine Ablenkung des Schalls vom Aussenden bis zum Empfangen des Mikrofons.

8 Ausblick

Die Software kann in verschiedene Module eingeteilt werden. Damit ist es möglich, die Software zu erweitern, ohne die anderen Module zu verändern. Im Folgenden werden Verbesserungsvorschläge für die einzelnen Module vorgestellt.

Kommunikation

Für die Kommunikation zwischen Master und Slave könnte anstatt des UDP-Protokolls 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 mit einer IP-Adresse erfolgen, anstatt auf Portnummern, wie es aktuell der Fall ist.

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 zwischen dem Funktionsaufruf *getSystemTime()* und dem Aussenden des Pakets. Um diese Verzögerungszeit zu eliminieren, sollte bei jedem Aussenden des Paketes die aktuelle Systemzeit angehängt werden.

Übertragungsmedium

Für zukünftige Anwendungen könnte man neben dem Schall auch Radiosignale verwenden. Sie haben den Vorteil, dass das menschliche Gehör diese Frequenzen nicht wahrnimmt. Des Weiteren breiten sich Radiosignale mit Lichtgeschwindigkeit aus.

Darüber hinaus könnte eine Frequenzmodulationen durchgeführt werden, wodurch eine gleichzeitige Abfrage von mehreren Accesspoints möglich wäre. Dadurch ist eine schnellere Positionsbestimmung durchführbar. Radiosignale haben allerdings auch den Nachteil, dass eine Dämpfung z.B. bei Wänden stattfindet. Weiterhin muss bedacht werden, dass es zu Reflexionen, Streuung und Absorption kommen kann [16].

Messung

Für die Positionsbestimmung müssen drei Gleichungen gelöst werden. Da MCUs nicht immer über eine FPU verfügen, kann eine MCU mit FPU verwendet werden. Als Alternative kann auch ein Raspberry-Pi genommen werden. Dieser ermöglicht die gemessenen Daten mit Octave oder Matlab weiterzuverarbeiten. Darüber hinaus ermöglicht Octave Statistiken oder graphische Aufbereitung von Daten.

Hardware

Aktuell sind der Lautsprecher und das Mikrofon über ein Steckbrett mit dem SAM R21

Xplained Pro Evaluation Kit verbunden. Es könnte auch eine Leiterplatte angefertigt werden, die auf das SAM R21 Xplained Pro Evaluation Kit aufgesteckt wird. Dies würde eventuelle Fehler beim Steckbrett vermeiden.

Signalanalyse

Um genauer herauszufinden, wann das Signal beim Mikrofon ankommt, könnte eine Signalanalyse durchgeführt werden. Dies hätte den Vorteil, dass fortlaufend die Zeit gestoppt wird, sofort nachdem die erste Halbwelle der Sinusschwingung erkannt wird bzw. eine Periode vorbei ist. Dadurch wäre es dann möglich, einen definierten Zeitpunkt zu bestimmen, wann das Lautsprechersignal das Rauschen am AUDIO-Ausgang überlagert. Eine Voraussetzung dafür wäre allerdings, dass ein Lautsprecher ein Signal dafür zurückgeben müsste, sofort nachdem die erste Halbwelle erreicht wurde.

Betriebssystem

Anstatt das Betriebssystem RIOT zu verwenden, könnte auch ein eigenes Betriebssystem zur Anwendung kommen. Dies würde eine bessere Programmanalyse über das laufende Programm ermöglichen. Darauf hinaus könnten eventuelle Softwareroutinen, die nicht unbedingt essenziell sind, abgeschaltet bzw. gar nicht erst implementiert werden. Weiterhin käme auch in Betracht, das RIOT OS anzupassen bzw. zu erweitern.

9 Zusammenfassung

Das Ziel dieser Arbeit, eine Positionsbestimmung im Zentimeterbereich durchzuführen, konnte nicht vollständig erreicht werden. Ein Grund war die mir nur begrenzt zur Verfügung stehende Zeit sowie die Einschränkung bei der Leistungsfähigkeit der Hardware. Die Arbeit hat verschiedene Aspekte der Positionsbestimmung untersucht: das Betriebssystem RIOT, die Hardware SparkFun Sound Detector und den HC-SR04. Dabei wurde die Verzögerung von Eingangs- und Ausgangssignalen untersucht.

Die Ergebnisse haben gezeigt, dass RIOT sich aufgrund der starken Verzögerung des Funktionsaufrufs *getSystemTime()* nicht für eine Positionsbestimmung eignet. Weiterhin wurde festgestellt, dass der verwendete Ultraschallsensor nicht die erforderlichen Anforderungen erfüllte, da er direkten Sichtkontakt benötigt und nur einen Sendekegel von 15 °C hat. Eine Vermutung für die nicht vollständige Positionsbestimmung ist, dass der Schall zwischen dem Raum des Sound Detectors und dem Lautsprecher abgelenkt und nicht auf den direkten Weg zum Mikrofon transportiert wird. Für die Positionsbestimmung müssen Master und Slave eine synchrone Zeit haben. Um dies zu gewährleisten, wurde mit PTP eine Zeitsynchronisation implementiert. Dabei konnte nachgewiesen werden, dass ohne Hardwareunterstützung keine genaue Zeitsynchronisation im Nanosekundenbereich möglich ist, die für eine Positionsbestimmung notwendig ist. Dies spiegelt auch mein Ergebnis wider. Um eventuelle neue Abweichungen diagnostizieren zu können, wäre es hilfreich, die beschriebenen Module für zukünftige anwendungsorientierte Forschungen zu wiederholen. Die Software liegt der Arbeit bei, und wurde auf Github [17] veröffentlicht.

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, 17.12.2019

Oliver Koepp

Literaturverzeichnis

- [1] *CE-Car* - URL <https://ce-master.htw-berlin.de/> - abgerufen am 12.09.2019
- [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 - abgerufen am 7.7.2019
- [3] *Atmel SAM R21* - URL https://riot-os.org/api/group__boards__samr21-xpro.html - abgerufen am 02.07.2019
- [4] *Ultraschall Sensor HC-SR04 und kompatible Ultraschall-Module* 2016 - URL <https://www.mikrocontroller-elektronik.de/ultraschallsensor-hc-sr04/> - abgerufen am 03.09.2019
- [5] *Sound Detector Hookup Guide* - URL <https://learn.sparkfun.com/tutorials/sound-detector-hookup-guide> - abgerufen am 09.09.2019
- [6] *433 Mhz Funk Transmitter-Receiver* - URL <https://draeger-it.blog/arduino-tutorial-433mhz-sender-empfaenger/> - abgerufen am 18.10.2019
- [7] *Goliton 5X Alarm 95dB Hohe Dezibel 6-24V 12V elektronischer Summer kontinuierlicher Piep Arduino MFC-27* - URL https://www.amazon.de/Goliton-Dezibel-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 30.07.2019
- [9] *TIME DIFFERENCE OF ARRIVAL* - URL <https://www.sewio.net/uwb-technology/time-difference-of-arrival/> - abgerufen am 04.07.2019
- [10] *UDP - User Datagram Protocol* - URL <https://www.elektronik-kompendium.de/sites/net/0812281.htm> - abgerufen am 25.08.2019
- [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
- [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 - abgerufen am 26.08.2019

- [14] *Finding Location with Time of Arrival and Time Difference of Arrival Techniques* - URL https://sites.tufts.edu/eesenior/designhandbook/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
- [16] *Funktechnik (Grundlagen)* 2019 - URL <https://www.elektronik-kompendium.de/sites/kom/0810301.htm> - abgerufen am 06.11.2019
- [17] *Bachelorarbeit* 2019 - URL <https://github.com/git-oliver/Bachelorarbeit> - abgerufen am 09.11.2019
- [18] *Understanding and Applying Precision Time Protocol* 2019 - URL https://cdn.selinc.com/assets/Literature/Publications/Technical%20Papers/6650_UnderstandingApplying_SA_20140206_Web3.pdf - abgerufen am 13.11.2019

10 Anhang

10.1 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.

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

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

$$y = \frac{x \cdot (2 \cdot x_A - 2 \cdot x_B) + r_A^2 - r_B^2 - x_A^2 + x_B^2 + y_A^2 - y_B^2}{2 \cdot (y_B - y_A)} \quad (10.7)$$

Um nun die X-Koordinaten zu bekommen, setzen wir die Gleichung 10.7 in Gleichung *I.* ein und lösen nach x auf.

$$\begin{aligned} P &= r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2 \\ (x - x_A)^2 + (y - y_A)^2 &= r_A^2 \\ (x - x_A)^2 + \left(\frac{x \cdot (2 \cdot x_A - 2 \cdot x_B) + P}{2 \cdot (y_B - y_A)} - y_A \right)^2 &= r_A^2 \\ x^2 \left(1 + \left(\frac{x_B - x_A}{y_B - y_A}\right)^2\right) + x \left(\frac{-P \cdot (x_B - x_A)}{(y_B - y_A)^2} + \frac{2 \cdot y_A \cdot (x_B - x_A)}{y_B - y_A} - 2 \cdot x_A\right) &= r_A^2 - y_A^2 + \frac{y_A \cdot P}{y_B - y_A} - \frac{P^2}{4 \cdot (y_B - y_A)^2} - x_A^2 \\ \left(1 + \left(\frac{x_B - x_A}{y_B - y_A}\right)^2\right) &= R \\ \left(\frac{-P \cdot (x_B - x_A)}{(y_B - y_A)^2} + \frac{2 \cdot y_A \cdot (x_B - x_A)}{y_B - y_A}\right) &= S \\ r_A^2 - y_A^2 + \frac{y_A \cdot P}{y_B - y_A} - \frac{P^2}{4 \cdot (y_B - y_A)^2} - x_A^2 &= T \end{aligned} \quad (10.8)$$

$$x^2 + x \cdot \frac{S - 2 \cdot x_A}{R} - \frac{T}{R} = 0$$

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

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

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

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q} \quad (10.9)$$

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

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

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 10.7 eingesetzt.

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

$$y_1 = \frac{x_1 \cdot (2 \cdot x_A - 2 \cdot x_B) + r_A^2 - r_B^2 - x_A^2 + x_B^2 - y_A^2 + y_B^2}{2 \cdot (y_B - y_A)} \quad (10.10)$$

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

Die Schnittpunkte von Kreis A und B sind:

$$(x_1|y_1) \quad (10.11)$$

$$(x_2|y_2)$$

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 10.11 berechnet.

$$d_1 = \sqrt{(x_C - x_1)^2 + (y_C - y_1)^2} \quad (10.12)$$

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

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 $(x_2|y_2)$.

Dieses mathematische Vorgehen wird für Kreis $A - C$ und $B - C$ wiederholt.

10.2 Steuercodes

Die folgende Tabelle listet alle vorhandenen Steuercodes auf.

Tabelle 12: Alle Steuercodes

Steuercode	<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