Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Artur Heck

Analyse und Test eines zirkularen 6-Mikrofon-Arrays von ReSpeaker

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Artur Heck

Analyse und Test eines zirkularen 6-Mikrofon-Arrays von ReSpeaker

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Mechatronik
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr.-Ing. Hans Peter Kölzer
Zweitprüfer: Prof. Dr. Ulrich Sauvagerd

Abgabedatum: 18.06.2019

Zusammenfassung

Artur Heck

Thema der Bachelorthesis

Analyse und Test eines zirkularen 6-Mikrofon-Arrays von ReSpeaker

Stichworte

ReSpeaker, zirkulares Mikrofonarray, Audioquellen Lokalisierung, DOA, TDOA, GCC-PHAT, Beamforming, Rauschunterdrückung

Kurzzusammenfassung

In dieser Arbeit werden die frei verfügbaren Sprachverbesserungsalgorithmen zur Lokalisation einer Audioquelle, Beamforming und Rauschunterdrückung für ein zirkulares Mikrofonarray von ReSpeaker analysiert. Die notwendigen Schritte zum Einrichten des Gerätes sowie installieren der genannten Sprachverbesserungsalgorithmen werden dargestellt. Weiter werden die Algorithmen analysiert und in Simulationen sowie in Verbindung mit dem Mikrofonarray bezüglich ihrer Leistungsfähigkeit getestet.

Artur Heck

Title of the paper

Analysis and test of a ReSpeaker circular 6 microphone array

Keywords

ReSpeaker, circular microphone array, audio source localization, DOA, TDOA, GCC-PHAT, Beamforming, noise suppression

Abstract

This thesis analyzes the freely available voice enhancement algorithms for audio source localization, beamforming and noise suppression for a ReSpeaker circular microphone array. The necessary steps for setting up the device and installing the voice enhancement algorithms are described. Next, the algorithms are analyzed and tested in simulations and in combination with the microphone array.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung und Motivation

Zur heutigen Zeit gehört die sprachliche Kommunikation mit elektrischen Geräten zur Alltäglichkeit. Wo früher die Interaktion mit einem Gerät durch Schnittstellen wie Tasten und Anzeigeelemente stattfand, können heutzutage viele Geräte mit Sprachbefehlen gesteuert werden. Einen großen Aufschwung in dem Bereich der Sprachsteuerungen haben die Sprachassistenten hingelegt. Diese spielen per Sprachbefehl Musik ab, ermöglichen es mit anderen zu telefonieren oder dienen als Schnittstelle, um andere elektronische Geräte ein- oder auszuschalten.

Bevor die Sprachbefehle verarbeitet werden können, müssen diese erstmal von der Hardware aufgenommen werden. Die Aufnahme der Sprachbefehle stellt dabei eine grundlegende Herausforderung dar. Im Vergleich zu einer Befehlseingabe über ein Terminal ergeben sich über einen akustischen Kanal viele Störgrößen. Diese treten durch die in der Umgebung befindlichen Störgeräusche auf.

Zur Kompensation der störenden Umgebungsgeräusche und der damit verbundenen Verbesserung des Sprachsignals, sind mehrere hard- und softwareseitige Lösungen vorhanden. Die grundlegende Lösung zur Verbesserung des Sprachsignals ist zum einen die Lokalisation des Sprechers und die anschließende Fokussierung auf diesen. Die Fokussierung kann durch das sogenannte Beamforming vollzogen werden und erfordert zusammen mit der Lokalisation mehrere Mikrofone, die in einer bestimmten Geometrie räumlich angeordnet sind. Die Anordnung mehrerer Mikrofone wird dabei als Mikrofonarray bezeichnet. Eine weitere reine Softwareseitige Lösung zur Minderung der Umgebungsgeräusche bietet die Rauschunterdrückung.

Das in dieser Arbeit untersuchte 6-Mikrofonen-Array von ReSpeaker verfügt über die entsprechende Hard- und Software für die erwähnten Verfahren zur Sprachverbesserung. Es handelt sich dabei um einen auf dem freien Markt verfügbaren Einplatinencomputer mit zusätzlich sechs zirkular angeordneten Mikrofonen auf der Platine. Dazu werden frei verfügbare Algorithmen zur Sprachverbesserung angeboten, mit denen verschiedene Sprachapplikationen direkt auf dem Gerät implementiert und durchgeführt werden können.

Die Arbeit beschäftigt sich mit der Untersuchung dieser Algorithmen und die darin verwendeten Verfahren. Zudem wird die Leistungsfähigkeit der Algorithmen in Verbindung mit der Hardware getestet.

1.2 Zielsetzung

Ziel der Arbeit ist es, das zirkulare 6-Mikrofonen-Array von ReSpeaker einzurichten und die dafür verfügbaren Algorithmen zur Lokalisation einer Audioquelle, Beamforming und

Rauschunterdrückung zu analysieren. Die darin genutzten Verfahren sollen einmal erläutert und dargestellt werden. Weiter sollen die jeweiligen Algorithmen sowohl in einer Simulation als auch auf der Hardware bezüglich ihrer Leistungsfähigkeit untersucht werden.

1.3 Aufbau der Arbeit

Im folgenden Kapitel werden die für die Arbeit notwendigen Definitionen und Darstellungen beschrieben. In Kapitel 3 wird das verwendete 6-Mikrofonen-Array von ReSpeaker vorgestellt und auf dessen Spezifikationen eingegangen. Zudem wird das Aufspielen des Betriebssystems, die Installation der notwendigen Sprachverarbeitungspakete und die vorgenommenen Einstellungen dargelegt. Außerdem wird die Anwendung der Sprachverarbeitungspakete dargestellt. In den Kapiteln 4 bis 6 werden die verfügbaren Algorithmen zur Lokalisation einer Audioquelle, zum Beamforming und zur Rauschunterdrückung analysiert und dessen Funktionsprinzip erläutert. Die Algorithmen werden anschließend zur Überprüfung ihrer Leistungsfähigkeit in einer Simulationsumgebung getestet. Weiter werden diese auf der Hardware ausgeführt und in verschiedenen (idealer und / oder realer) Umgebungen getestet. Die simulierten und getesteten Ergebnisse werden miteinander verglichen, um die Praxistauglichkeit der Algorithmen sowie die Leistungsfähigkeit der Hardware zu beurteilen. Im letzten Kapitel wird die Arbeit durch ein Fazit abgerundet.

2 Definitionen und Darstellungen

Für die Arbeit sind einige grundlegende Definitionen und Darstellungen notwendig. Diese werden in diesem Kapitel vorgestellt.

2.1 Definition des verwendeten Koordinatensystems

Das in der Arbeit verwendete Koordinatensystem ist eine Mischung aus einem kartesischen Koordinatensystem und einem Kugelkoordinatensystem. Kugelkoordinaten beschreiben einen Vektor oder einen Punkt im Raum anhand eines Abstands und zwei Winkeln. Es gibt mehrere Konventionen bezüglich der Spezifikation der beiden Winkel. Die hier verwendete ist in Abbildung 2.1 dargestellt und zeigt einen Punkt mit den kartesischen Koordinaten (x, y, z) und den Kugelkoordinaten (r, φ, θ).

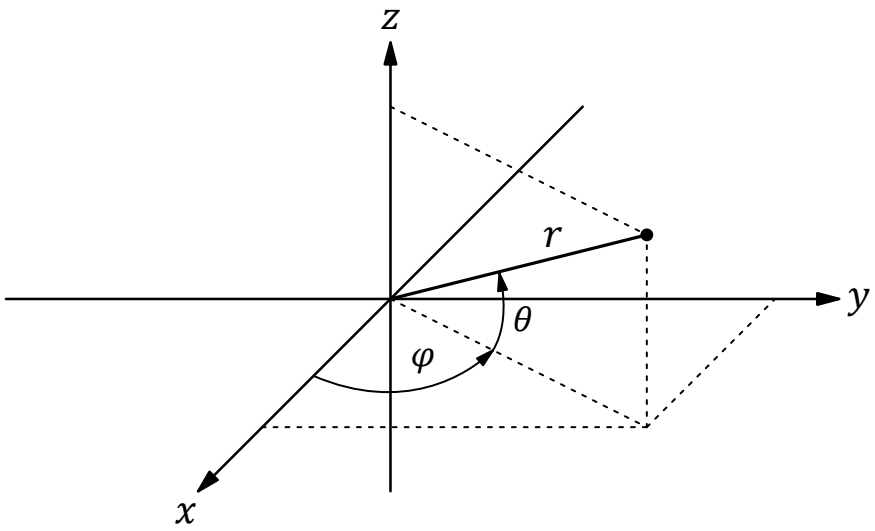


Abbildung 2.1: Darstellung des verwendeten Kugel- und kartesischen Koordinatensystems

Bei den Kugelkoordinaten ist r der Ortsvektor, φ der Azimutwinkel und θ der Elevationswinkel. Diese haben folgende Wertebereiche und Bedeutungen:

- $0 \leq r$, Ortsvektor vom Ursprung zum Punkt
- $0 \leq \varphi < 2\pi$, Winkel zwischen positiver *x*-Achse und der Projektion des Ortsvektors auf die *xy*-Ebene
- $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$, Winkel zwischen der Projektion des Ortsvektors auf die *xy*-Ebene und dem Ortsvektor

Die Transformationsgleichungen von Kugelkoordinaten zu kartesischen Koordinaten lauten:

$$\begin{aligned} x &= r\cos\theta\cos\varphi \\ y &= r\cos\theta\sin\varphi \\ z &= r\sin\theta \end{aligned} \tag{2.1}$$

Die Transformationsgleichungen von kartesischen Koordinaten zu Kugelkoordinaten lauten:

$$\begin{aligned} r &= \sqrt{x^2+y^2+z^2} \\ \varphi &= \tan^{-1}\left(\frac{y}{x}\right) \\ \theta &= \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right) \end{aligned} \tag{2.2}$$

Bei einer Angabe der Position von einer Audioquelle in Bezug auf ein Mikrofonarray, bezieht man sich häufig auf die Entfernung und Richtung der Audioquelle zum Mikrofonarray [1]. Der Abstand zum Array entspricht dem Ortsvektor r in Kugelkoordinaten. Die Richtung entspricht den Azimut- und Elevationswinkel.

2.2 Definition einer ebenen Welle

Schallwellen breiten sich von einer Quelle in alle Richtungen aus. Man spricht von einer omnidirektionalen Ausbreitung. Die Ausbreitungsgeschwindigkeit der Schallwellen hängt von den Eigenschaften des Mediums ab. In Luft beträgt die Schallgeschwindigkeit 343 m/s bei einer Lufttemperatur von 20°C [2]. In Bezug auf ein Mikrofonarray entstehen aufgrund der Position der Mikrofone und der Schallgeschwindigkeit eine Zeitverzögerung zwischen den Mikrofonsignalen. Bei der Berechnung der Laufzeitdifferenz zwischen den Mikrofonen wird zur Vereinfachung das Modell einer ebenen Welle verwendet.

Bei einer ebenen Welle handelt es sich um eine parallel verlaufende Wellenfront, auf dessen Ebene alle Punkte die gleiche Phase haben. Dieses Verhalten kann angenommen werden, wenn die Messung des Wellensignals aus ausreichender Entfernung stattfindet. Es wird dabei zwischen dem Nahfeld und dem Fernfeld unterschieden (siehe Abbildung 2.2). Um die Schallwellen als eine parallel verlaufende Wellenfront und somit als ebene Welle zu interpretieren, muss die folgende Bedingung erfüllt sein [3]:

$$r \gg \frac{d^2}{c} f\,. \tag{2.3}$$

Dabei ist r der Abstand zwischen dem Mikrofonarray und der Quelle, d der Abstand der äußersten Mikrofone des Arrays, c ist die Ausbreitungsgeschwindigkeit der Welle und f die Frequenz der Welle.

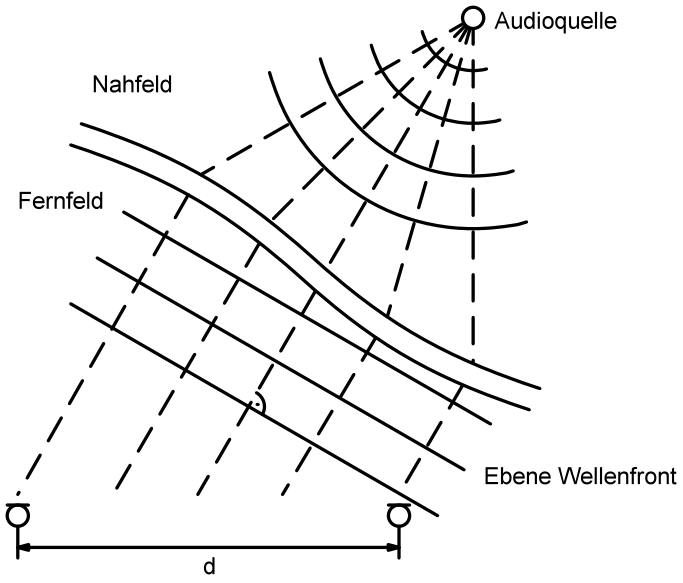


Abbildung 2.2: Darstellung des Nah- und Fernfelds

2.3 Eigenschaften von Sprachsignalen

Ein Sprachsignal kann als ein stochastisches (zufälliges) Signal gesehen werden. Der exakte Verlauf ist nicht genau darstellbar [4]. Es weist jedoch einige Eigenschaften und Merkmale auf, die im Zeitbereich als auch im Frequenzbereich sichtbar sind.

Zeitlich gesehen beinhaltet das Sprachsignal quasiperiodische und rauschartige Signalabschnitte. Dazwischen können beliebige Übergangs- und Mischbereiche auftreten. Die quasiperiodischen Abschnitte weisen eine Grundwelle und Oberwellen auf. Die Frequenz der Grundwelle wird als Grundfrequenz bezeichnet. Diese ist nicht konstant, sondern ändert sich laufend [5]. Die Grundfrequenz liegt bei Männern zwischen 50 Hz und 250 Hz und bei Frauen zwischen 120 Hz und 500 Hz [6].

Ein Sprachsignal ist aus mehreren Komponenten zusammengesetzt, die frequenzmäßig über den gesamten Hörbereich verteilt sind. Über einen längeren Zeitraum gesehen ist die relative Bandbreite von Sprachsignalen groß. Schaut man sich einen kürzeren Zeitraum an, so sind Sprachsignale an gewissen Stellen quasistationär, aber nie exakt stationär. Exakt stationäre (periodische) Signale werden nicht als Sprache, sondern als technische Geräusche wahrgenommen [5]. Als quasistationär kann das Sprachsignal bei einer Dauer zwischen 15 ms bis 20 ms angesehen werden [7].

Eine weitere wichtige Eigenschaft von Sprachsignalen ist, dass ihre spektrale Zusammensetzung sich nicht abrupt ändert [5].

3 Vorstellung des zirkularen 6-Mikrofon-Arrays

Bei der in dieser Arbeit verwendeten Hardware handelt es sich um den ReSpeaker Core v2.0. Dies ist ein Einplatinencomputer mit zusätzlich sechs zirkular angeordneten Mikrofonen, die auf derselben Platine aufgelötet sind. Das Board läuft unter anderem mit dem Betriebssystem Debian und es stehen Software basierte Sprachverbesserungsalgorithmen zur Verfügung [8].

Laut der Wiki Seite des Herstellers [9] sind beispielhaft die folgenden Anwendungen mit dem Gerät realisierbar:

- intelligenter Lautsprecher
- intelligentes Sprachassistenzsystem
- Sprachaufzeichnungsgerät
- Sprachkonferenzsystem
- Meeting-Kommunikationsausrüstung
- Sprachinteraktiver Roboter
- Auto-Sprachassistent
- andere Szenarien mit Sprachschnittstellen

In diesem Kapitel wird auf die Hardware des Gerätes näher eingegangen. Weiter werden die benötigten Schritte zum Aufspielen des Betriebssystems und die vorzunehmenden Einstellungen erläutert. Anschließend wird der Einsatz der in der Arbeit untersuchten Sprachverbesserungsfunktionen dargestellt. Zum Ende des Kapitels wird die Nutzung der Sprachverbesserungsalgorithmen aufgezeigt.

3.1 Hardware Spezifikationen des ReSpeaker Core v2.0

Der ReSpeaker Core v2.0 ist speziell für Sprachschnittstellenanwendungen konzipiert. Als Basis dient ein Rockchip RK3229, ein Quad-Core ARM Cortex A7 mit einer Taktfrequenz von bis zu 1,5 GHz und einer Mali 400MP GPU. Zusätzlich stehen insgesamt 1 GB DDR3 RAM zur Verfügung [9]. Es befinden sich außerdem noch sechs zirkular angeordnete Mikrofone auf dem Board.

Das Board besteht aus zwei Einheiten. Die erste Einheit ist das Kernmodul, bestehend aus der CPU, RAM und PMU. Die zweite Einheit ist die äußere Trägerplatine, die die Peripheriegeräte wie eMMC, diverse Anschlüsse, Komponenten für die drahtlose Verbindung, das Mikrofonarray und im Ring angeordnete RGB LEDs enthält [8]. Die Abbildung 3.1 zeigt die Front des ReSpeaker Core v2.0. Im oberen Bereich der Abbildung sind die Anschlüsse und einige Komponenten der Trägerplatine zu sehen. Im unteren Bereich der Abbildung ist nochmals das Kernmodul mit den darauf befindlichen Komponenten und deren Spezifikationen dargestellt.

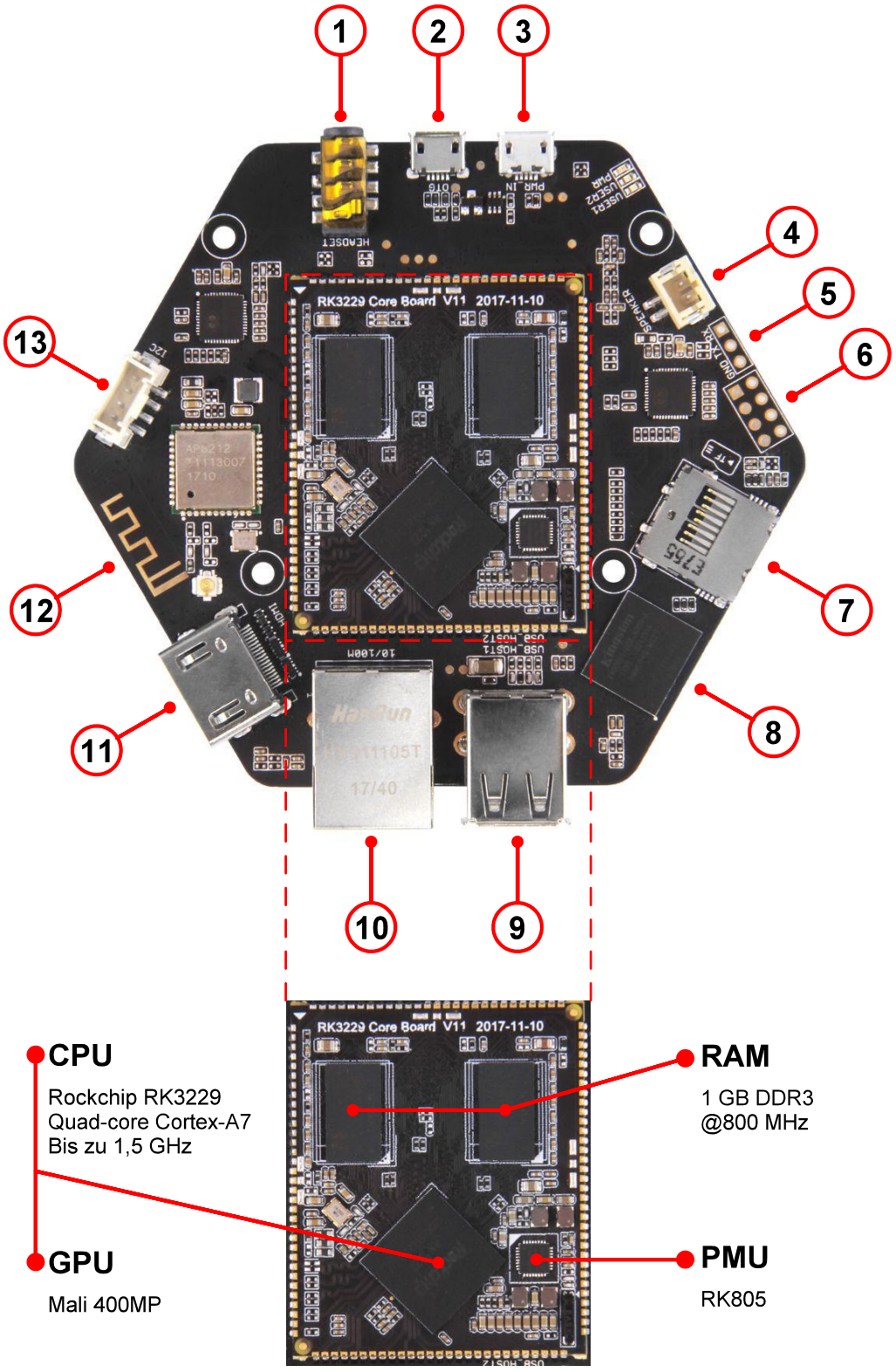


Abbildung 3.1: Front des ReSpeaker Core v2.0, oben Trägerplatine mit Kernmodul, unten Kernmodul [8]

In der folgenden Liste werden die nummerierten Komponenten von Abbildung 3.1 kurz beschrieben [9].

① **3,5 mm Kopfhöreranschluss:**
Es können aktive Lautsprecher oder Kopfhörer zur Audioausgabe angeschlossen werden.

② **USB OTG:**
Über diesen USB Anschluss kann eine serielle Verbindung mit einem Computer hergestellt werden. Anhand eines geeigneten Programms (z.B. *PuTTY*) können beide Geräte miteinander kommunizieren.

③ **USB Stromanschluss:**
Dieser Anschluss dient zur Stromversorgung des ReSpeaker Core v2.0.

④ **Lautsprecheranschluss:**
JST 2.0 Buchse zum Anschluss eines passiven Lautsprechers für die Audioausgabe.

⑤ **Universal Asynchronous Receiver Transmitter (UART):**
Alternativer UART-Anschluss zur Verbindung des ReSpeaker Core v2.0 mit dem Computer zur seriellen Kommunikation.

⑥ **8 Pin General Purpose Input Output (GPIO):**
GPIO Schnittstelle für erweiterte Anwendungen.

⑦ **SD-Kartensteckplatz:**
Zum Einstecken einer Micro-SD-Karte für das Betriebssystem oder erweiterten Speicherplatz.

⑧ **Embedded Multi Media Card (eMMC):**
Fest verbauter 4 GB Speicher für das Aufspielen eines Betriebssystems.

⑨ **USB-Host:**
Zwei USB Schnittstellen zum Anschließen von USB-Peripheriegeräten wie USB-Maus, USB-Tastatur oder USB-Flash-Speicher.

⑩ **Ethernet:**
Zum Anschließen eines Ethernet-Kabels, um Zugang zum Internet zu erlangen.

⑪ **HDMI 2.0:**
HDMI Schnittstelle zur Ausgabe von Videosignalen. Hardwareseitig ist eine Videoausgabe von maximal 2160p@60fps möglich. Das Betriebssystem Debian unterstützt momentan jedoch nur 1080p@60fps [8].

⑫ **Bluetooth- und WLAN-Antenne:**
Integrierte Antenne für WLAN und Bluetooth.

⑬ **Grove Buchse:**
Buchse mit einer I2C Schnittstelle zum Anschluss verschiedener Sensoren, Aktoren oder Anzeigeelementen, die über eine Grove Buchse verfügen [10].

Auf der Rückseite des ReSpeaker Core v2.0 befinden sich sechs Mikrofone, die in jeder Ecke sitzen. Unterhalb der Mikrofone sind insgesamt zwölf RGB LEDs kreisförmig angeordnet. Die Rückseite ist in Abbildung 3.2 dargestellt.

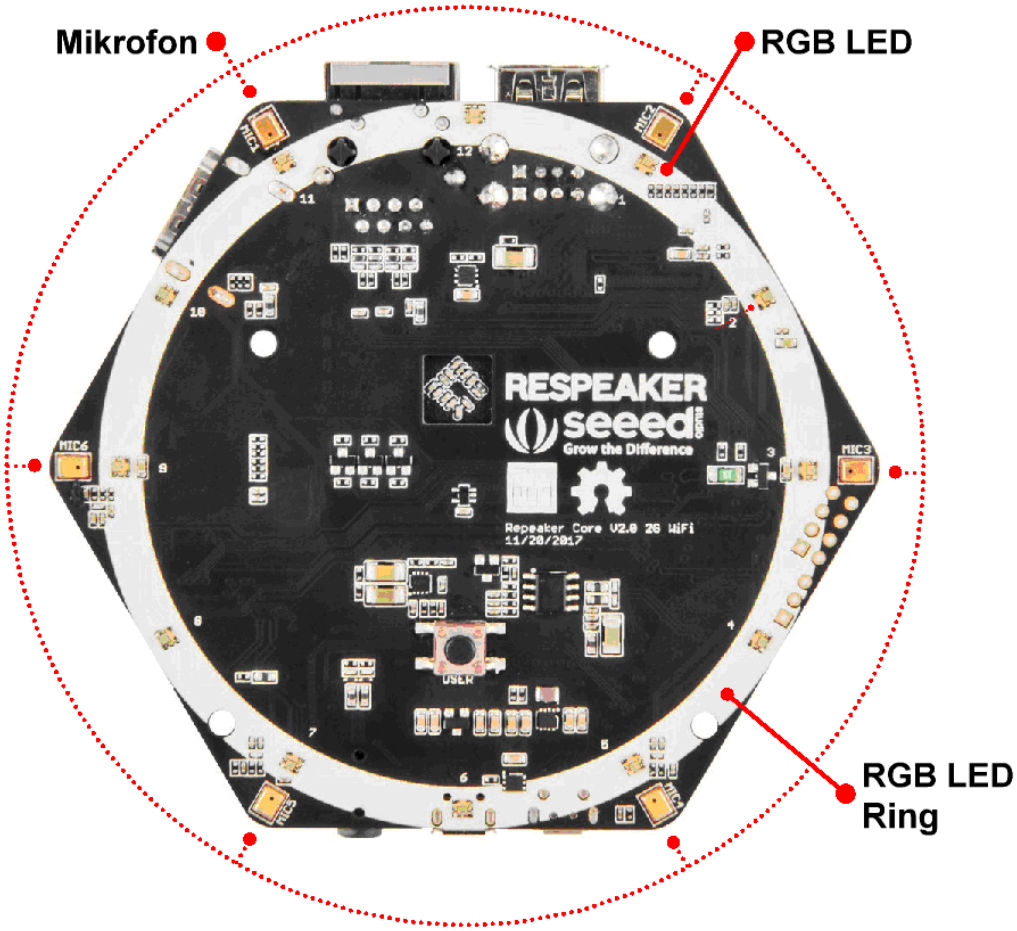


Abbildung 3.2: Rückseite des ReSpeaker Core v2.0 [8]

Die sechs Mikrofone sind an zwei 4-Kanal Analog-Digital-Umsetzer (engl. Analog-Digital-Converter) (ADC) des Typs AC108 angeschlossen. Für die beiden ADCs ergeben sich insgesamt acht analoge Kanäle, wofür jeweils ein Kanal für ein Mikrofon benötigt wird. Die Kanäle eins bis sechs sind mit den Mikrofonen eins bis sechs verbunden. Die beiden verbliebenden Kanäle dienen als sogenannte *loopback-Channels*. Dabei ist der linke Kanal des Kopfhöreranschlusses mit dem siebten Kanal vom ADC verbunden und der Lautsprecheranschluss mit dem achten Kanal vom ADC. Da die beiden Ausgänge über Leiterbahnen mit dem ADC verbunden sind, handelt es sich um Hardware loopbacks. Die Abbildung 3.3 zeigt die Audiostrecke des ReSpeaker Core v2.0 als Blockschaltbild. Das gesamte System als Blockschaltbild ist im Anhang unter A.1 dargestellt.

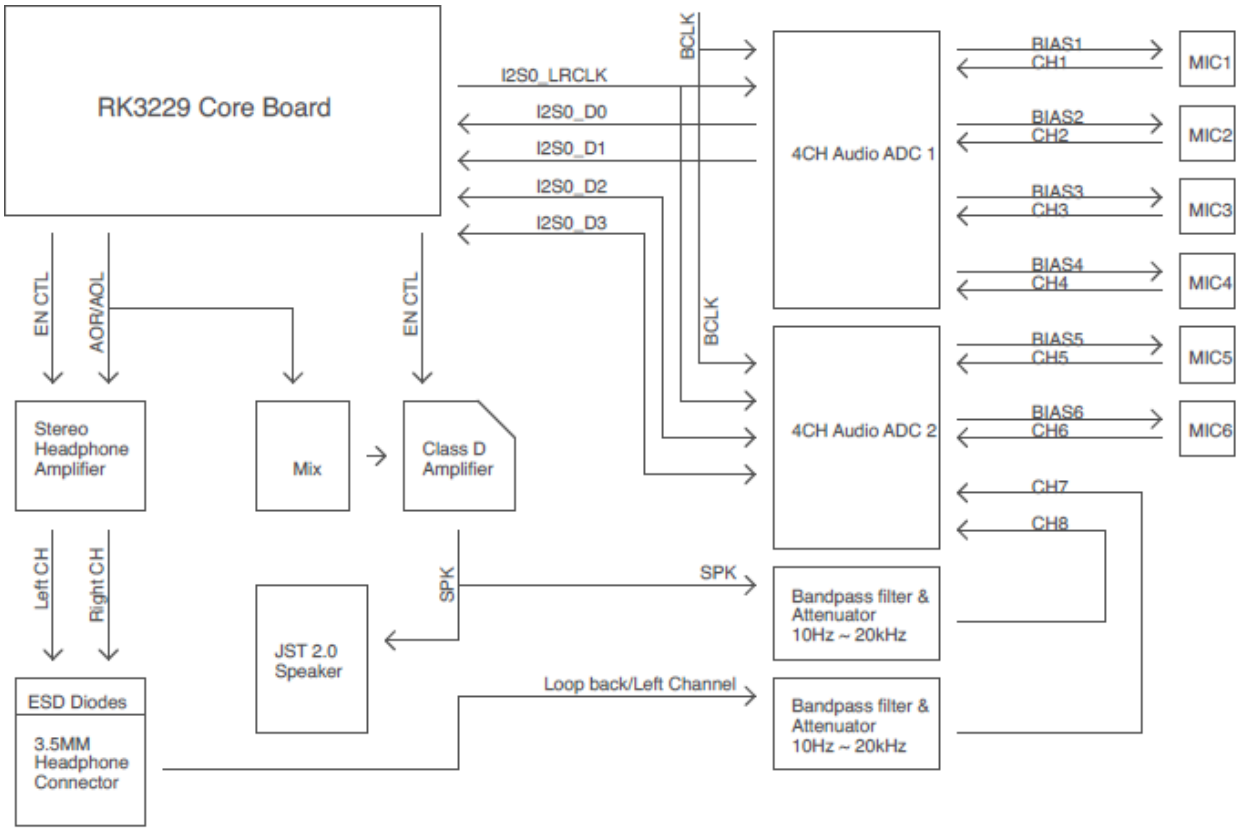


Abbildung 3.3: Audiostrecke des ReSpeaker Core v2.0 als Blockschaltbild [9]

Bei den ADCs handelt es sich um spezielle Audio ADCs zum Anschließen von analogen Mikrofonen für Plattformen zur Sprachaufnahme und -erkennung. Die unterstützten Abtastfrequenzen reichen von 8 kHz bis 96 kHz. Jeder Kanal besitzt einen unabhängigen und frei programmierbaren Verstärker mit einer Verstärkung von 0 dB bis 30 dB in 1 dB Abstufungen. Die Datenübertragung ist über eine I2S Schnittstelle gewährleistet und über das Kernmodul mit dem Rockchip RK3229 verbunden. Weitere Informationen zu dem AC108 können aus dem Datenblatt unter [11] entnommen werden.

3.2 Einrichtung des ReSpeaker Core v2.0

Um den ReSpeaker Core v2.0 verwenden zu können, ist es notwendig ein Betriebssystem auf das Gerät aufzuspielen. Ohne ein Betriebssystem kann das Gerät, ähnlich wie ein Desktopcomputer oder Laptop, nicht vollwertig genutzt werden. Nachdem das Betriebssystem aufgespielt ist, kann der ReSpeaker Core v2.0 eingerichtet und die verschiedenen Module und Pakete für die Sprachverarbeitung installiert werden.

In den zwei folgenden Abschnitten wird einmal das Aufspielen eines Betriebssystems und die Einrichtung und Installation der für diese Arbeit notwendigen Module und Pakete dargestellt. Das beschriebene Vorgehen basiert auf der *ReSpeaker Core v2.0 – Seeed Wiki* und kann unter [9] näher eingesehen werden.

3.2.1 Aufspielen des Betriebssystems

Für den ReSpeaker Core v2.0 stehen die Betriebssysteme Android O und Debian zur Verfügung [8]. In dieser Arbeit wird das Betriebssystem Debian verwendet. Bei Debian handelt es sich um ein freies und offenes Betriebssystem, welches einen Linux-Kern verwendet [12].

Beim ReSpeaker Core v2.0 kann sich das Betriebssystem entweder auf der eMMC befinden oder auf der Micro-SD-Karte (im weiteren Text gekürzt zu SD-Karte). Zum Aufspielen des Betriebssystems wird für beide Varianten eine SD-Karte benötigt. Der Vorteil an der Variante mit dem Betriebssystem auf dem eMMC liegt darin, dass im Nachhinein für den Betrieb keine zusätzliche SD-Karte benötigt wird. Der Vorteil an der Variante mit dem Betriebssystem auf der SD-Karte liegt in der Flexibilität. So ist es möglich verschiedene SD-Karten mit unterschiedlichen Einstellungen und Programmen zu erstellen und diese je nach Bedarf auszutauschen. Zudem kann über bestimmte Software ein Abbild der SD-Karte mit allen Einstellungen und Programmen erzeugt werden. Damit können fehlerhafte Einstellungen oder Software, die das System korrumpieren, wieder zurückgesetzt werden. Dafür kann die SD-Karte formatiert und mit dem zuvor erstellten Abbild neu beschrieben werden.

In dieser Arbeit wird die Variante mit dem Betriebssystem auf der SD-Karte verwendet.

Um das Betriebssystem auf die SD-Karte zu schreiben, sind abgesehen von der SD-Karte, eine Datei erforderlich, die das Betriebssystem enthält. Diese Datei wird als *Image* bezeichnet. Weiter wird eine Software für das Schreiben des Betriebssystems auf die SD-Karte benötigt. Zudem ist eine Software zum Formatieren von SD-Karten empfehlenswert. Das Image und die benötigte Software können frei aus dem Internet geladen werden.

Das Image mit dem Betriebssystem kann unter [13] im Verzeichnis *„respeakerv2“* unter *„debian“* geladen werden. Auf der Seite sind mehrere Ausführungen verfügbar. Der grundlegende Aufbau der Datei lautet:

respeaker-debian-9-[Version]-[Variante]-[Datum]-4gb.img.xz

Es sind jeweils zwei verschiedene Versionen und Varianten vorhanden. Die Unterscheidungen sind in Tabelle 3.1 dargestellt. Bei dem Datum handelt es sich um das Veröffentlichungsdatum.

Tabelle 3.1: Beschreibung der verschiedenen Image Ausführungen

		Beschreibung
Version	**iot**	Keine grafische Benutzeroberfläche. Stellt nur das Terminal dar.
	lxqt	Beinhaltet eine grafische Benutzeroberfläche (Desktop GUI)
Variante	**flasher**	Schreibt das Betriebssystem auf die eMMC. Nach dem Schreibvorgang kann die SD-Karte entnommen werden
	sd	Das Betriebssystem befindet sich auf der SD-Karte. Die SD-Karte muss während der gesamten Betriebszeit im Kartensteckplatz bleiben.

Die *lxqt-sd* Ausführung ist empfehlenswert, da die grafische Benutzeroberfläche die Bedienung erleichtert. Zudem bringt die Variante mit der SD-Karte die oben genannten Vorteile.

Bevor das Image auf die SD-Karte geschrieben wird, ist empfehlenswert die SD-Karte zu formatieren. Dafür eignet sich die Software *SD Card Formatter.* Der Vorteil dieser Software gegenüber dem Formatieren über Windows liegt daran, dass selbst mit einem Image beschriebene SD-Karten auf ihre ursprüngliche Partition formatiert werden können.

Nach dem Formatieren der SD-Karte kann das Betriebssystem mit der Software *balenaEtcher* auf die SD-Karte geschrieben werden. Das genaue Vorgehen zum Beschreiben der SD-Karte und das Verwenden der Sofrware kann aus der ReSpeaker Wiki [9] entnommen werden.

Nach Abschluss des Schreibvorgangs kann die SD-Karte in den SD-Kartensteckplatz des ReSpeaker Core v2.0 eingesteckt werden. Über den PWR_IN Micro USB Anschluss wird das Board mit Spannung versorgt (DC 5V). Die SD-Karte darf während der gesamten Zeit, in der das Board mit Spannung versorgt wird, nicht entnommen werden. Der ReSpeaker Core v2.0 bootet automatisch von der SD-Karte und die PWR, USER1 und USER2 LEDs leuchten auf. Die USER1 LED blinkt dabei im Takt eines Herzschlags und die USER2 LED leuchtet bei jedem Zugriff auf die SD-Karte auf. Damit ist der ReSpeaker Core v2.0 betriebsbereit.

Um mit dem Gerät interagieren zu können, sind mehrere Möglichkeiten vorhanden. Die erste Möglichkeit ist es die notwendigen Peripheriegeräte wie Monitor, Maus und Tastatur direkt an die jeweiligen Anschlüsse des ReSpeaker Core v2.0 anzuschließen. Die andere Möglichkeit bestehen darin das Gerät mit Hilfe eines Computers ferngesteuert zu bedienen. Eine Variante ist es den ReSpeaker Core v2.0 anhand eines Micro USB Kabels über den USB OTG Anschluss mit dem Computer zu verbinden. Mit einer geeigneten Software wie beispielsweise *PuTTY* kann auf das Terminal des ReSpeaker Core v2.0 zugegriffen werden. Eine andere Variante ist es den ReSpeaker Core v2.0 mit dem Internet zu verbinden und mithilfe der Software *VNC Viewer* die Benutzeroberfläche des Gerätes auf dem Computer zu projizieren. Eine Anleitung zum Einrichten und verwenden der Programme befindet sich in der ReSpeaker Wiki [9].

3.2.2 Installation der Sprachverarbeitungspakete

Für die Installation der Pakete benötigt der ReSpeaker Core 2.0 eine aktive Internetverbindung. Die Installation wird über eine Texteingabe im Terminal durchgeführt.

Als erstes ist es empfehlenswert alle auf dem Gerät befindlichen Pakete zu aktualisieren. Dies kann über die folgende Eingabe im Terminal durchgeführt werden.

```
sudo apt update
sudo apt upgrade
```

Nachdem das System aktualisiert ist, können die Sprachverarbeitungspakete installiert werden.

Für den ReSpeaker Core v2.0 ist eine Closed Source Solution und eine Open Source Solution verfügbar. Für diese Arbeit wird die Open Source Solution verwendet, um den Quellcode einsehen zu können.

Die Open Source Solution basiert auf dem *Voice-Engine* Paket. Dieses Paket enthält in der Programmiersprache *Python* geschriebene Skripte zur Sprachverbesserung und -verarbeitung. Um den vollen Funktionsumfang nutzen zu können, werden noch weitere zusätzliche Pakete benötigt. Das Voice-Engine Paket kann unter [14] eingesehen werden.

Zur Installation des Voice-Engine Pakets muss der folgende Befehl in das Terminal eingegeben werden.

```
pip install voice-engine
```

Dadurch werden die im Paket enthaltenen Module installiert und stehen global zur Verfügung. Durch Eingabe des Befehls

```
git clone https://github.com/voice-engine/voice-engine.git
```

wird der Inhalt des Voice-Engine Pakets in das aktuelle Verzeichnis kopiert. Darin enthalten sind Beispielskripte zur Nutzung der im Paket enthaltenen Module, sowie Skripte der Module selbst.

Das Paket beinhaltet folgende Module [14]:

- Direction of Arrival (DOA), Lokalisierung einer Audioquelle
- Beamforming (BF)
- Noise Suppression (NS), Rauschunterdrückung
- Keyword Spotting (KWS), Schlüsselwort-Erkennung
- Acoustic Echo Cancelation (AEC), Akustische Echokompensation
- Voice Activity Detection (VAD), Sprachpausenerkennung

Die Module DOA und BF können ohne weitere Pakete ausgeführt werden. Die weiteren Module benötigen für ihre Ausführung zusätzliche Pakete.

Zur Nutzung der Rauchunterdrückung ist das *webrtc-audio-processing* Paket notwendig. Dieses wird durch die folgende Terminaleingabe installiert.

```
pip install webrtc-audio-processing
```

In dieser Arbeit werden nur die Module DOA, BF und NS untersucht und getestet. Die Installationsschritte für die weiteren notwendigen Pakete der Voice-Engine sind im Anhang unter A.2 zu finden.

Nun müssen noch die Mikrofonverstärkungen eingestellt werden. Dies erfolgt über den *Alsamixer* mit dem folgenden Befehl:

```
sudo alsamixer
```

Mit Betätigen der F6-Taste ist die Soundkarte *seeed-8mic-voicecard* auszuwählen. Über die Pfeiltasten können die Verstärkungen eingestellt werden. Nach [15] liegen die Standardwerte für die Verstärkungen auf einem Wert von 65 (vergl. Abbildung 3.4). Bei den in dieser Arbeit durchgeführten Tests werden diese Einstellungen verwendet.

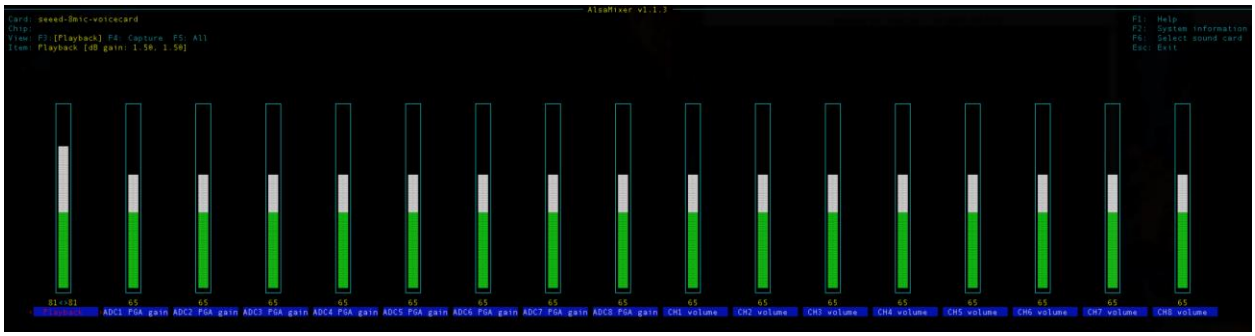


Abbildung 3.4: Alsamixer Einstellungen für den ReSpeaker Core v2.0 [15]

Durch betätigen der ESC-Taste gelangt man zurück zur Terminaleingabe. Über den Befehl

```
sudo alsactl store
```

werden die eingestellten Werte gespeichert.

3.3 Überblick und Einsatz der Sprachverbesserungsfunktionen

Bevor die drei Sprachverbesserungsalgorithmen zur Lokalisation einer Audioquelle, Beamforming und Rauschunterdrückung analysiert und getestet werden, wird hier kurz beschrieben worum es sich dabei handelt und welche Gründe ihr Einsatz hat.

Lokalisation einer Audioquelle

Wie der Name schon sagt, handelt es sich dabei um die Positionsbestimmung einer Audioquelle. In Verbindung mit Sprachverarbeitungsverfahren, wird damit die Position des Sprechers ermittelt. Anhand dieser Information können weitere sprachverarbeitende Maßnahmen, wie das Beamforming durchgeführt werden. Für die Lokalisation einer Audioquelle sind mehrere Mikrofone notwendig. Mit geeigneten Verfahren kann dadurch die Richtung des Sprechers ermittelt werden.

Beamforming

Beim Beamforming handelt es sich um ein Richtverfahren. Dafür wird die räumliche Ausdehnung des Mikrofonarrays genutzt, um eine Richtwirkung zu erzielen. Damit lassen sich Störquellen außerhalb der fokussierten Richtung ausblenden. Die Ausrichtung wird elektronisch vorgenommen. Mithilfe der Positionsdaten des Sprechers kann der Fokus des Beamformers auf diesen gesetzt werden. Störgeräusche außerhalb der Richtung des Sprechers werden durch den Beamformer gedämpft.

Rauschunterdrückung

Bei der Rauschunterdrückung handelt es sich, im Gegensatz zum Beamforming, um ein einkanaliges Verfahren. Mit der Rauschunterdrückung werden alle Störgeräusche außerhalb des Sprachsignals gedämpft. In Verbindung mit einem Mikrofonarray kann die Rauschunterdrückung nach dem Beamforming durchgeführt werden. Damit ist es möglich Störgeräusche in unmittelbarer Nähe zum Sprecher zu dämpften.

3.4 Anwendung der Sprachverarbeitungsmodulen

Die in der Voice-Engine befindlichen Module sind so konzipiert, dass diese miteinander verlinkt werden können. Es ist ein Signalfluss von einem Modul zum nächsten realisierbar, wodurch verschiedene Verarbeitungsschritte nacheinander durchgeführt werden. Zudem besteht die Möglichkeit die Daten von einem Modul in mehrere Module einzuspeisen. Um einen Signalfluss zu realisieren, wird jedem Modul eine Referenz zu den verlinkten Modulen übergeben. Bei jedem neuen Datenpaket werden dann die verarbeiteten Daten an die referenzierten Module weitergereicht.

In der Programmiersprache Python werden Klassen und ausgelagerte Funktionen als Module bezeichnet. In der Voice-Engine beinhalten alle audioverarbeitenden Module eine Klasse. Für einen funktionierenden Signalfluss wird in jeder Klasse eine Referenz der nachfolgenden Klasse hinterlegt. Anhand dieser können die verarbeitenden Daten weiterleitet werden. Da jede Klasse Referenzen hinterlegen muss, erben alle Klassen von der übergeordneten Klasse *Element*. Diese besitzt eine Methode `link(sink),` womit die Referenzen der nachfolgenden Klassen übergeben werden können. Zum Weiterreichen der verarbeiteten Daten ist die Methode `put(data)` der Klasse Element zuständig. In dieser werden nacheinander die `put(data)` Methoden der referenzierten Klassen aufgerufen und damit die verarbeiteten Daten weitergereicht. Jede Klasse in dem Voice-Engine Paket verfügt über eine eigene `put(data)` Methode, in welcher die Daten der vorherigen Klasse übergeben werden. Die aktuelle Klasse, die im Signalfluss an der Reihe ist, verarbeitet die Daten aus ihrer *put*-Methode und übergibt diese durch den Aufruf der *put*-Methode ihrer übergeordneten Klasse Element weiter. In dieser werden die verarbeiteten Daten zu den referenzierten Klassen weitergereicht. Dadurch wird das Audiosignal nacheinander von den einzelnen Klassen verarbeitet.

Um nun einen Signalfluss zu erstellen, ist zuerst eine Quelle notwendig. Diese sendet einen Audiostream aus. Dazu wird eine Instanz der Klasse *Source* benötigt. Das Voice-Engine Paket stellt mehrere Source Klassen zur Verfügung. Je nach Klasse werden die Audiosignale direkt aus den Mikrofonen des ReSpeaker Core v2.0 oder aus einer Datei entnommen. Bei den Dateien kann der Audiostream aus einer WAVE-Audiodatei oder einer Textdatei mit binär codierten Rohdaten entnommen werden. Nachdem eine Instanz der Source Klasse erstellt ist, kann über die `link(sink)` Methode eine Instanz der zu verlinkenden Klassen übergeben werden. Über diese Methode können anschließend weitere Instanzen verlinkt werden. Müssen mehrere Instanzen nacheinander verlinkt werden, so kann die Methode `pipeline(*args)` verwendet werden. Die einzelnen Instanzen werden nacheinander mit einem Komma getrennt eingetragen.

Der von Source ausgegebene Audiostream ist vom Typ *String*. In diesem sind die einzelnen Abtastwerte, je nach Konfiguration, in einer Auflösung von 16 oder 32 Bit binär hinterlegt. Beinhaltet der Audiostream mehrere Kanäle, so werden die einzelnen Abtastwerte der Kanäle hintereinander hinterlegt. Der Audiostream wird blockweise mit einer bestimmten Anzahl an Abtastwerten ausgegeben. Zur Veranschaulichung ist in Abbildung 3.5 ein Ausschnitt eines Audiostreams mit 4-Kanälen und 16 Bit Auflösung über zwei Abtastwerte dargestellt.

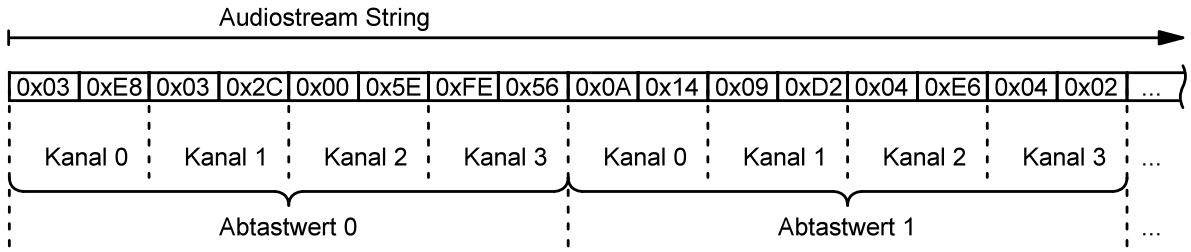
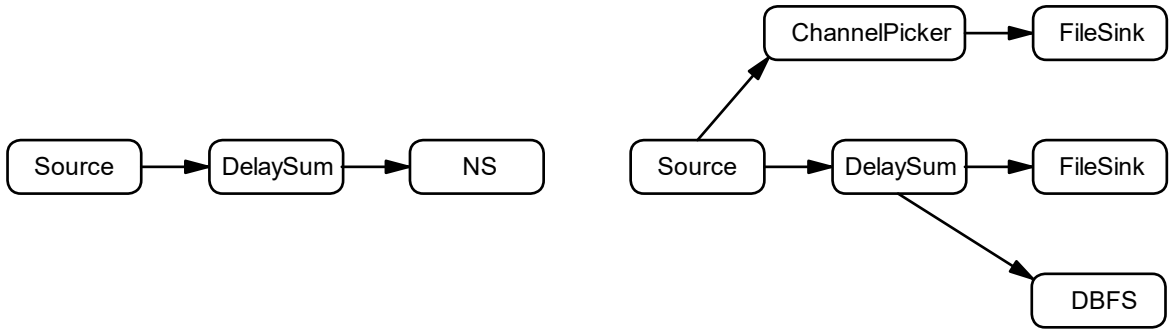


Abbildung 3.5: Beispiel eines Audiostream Strings mit 4-Kanälen und 16 Bit Auflösung

Zur Veranschaulichung der Verlinkung von Modulen, sind in Abbildung 3.6 zwei Signalflüsse grafisch dargestellt. Bei (a) handelt es sich um einen einfachen Signalfluss ohne Verzweigungen. Bei (b) ist ein Signalfluss mit mehreren Verzweigungen dargestellt.



(a) Einfach Verzweigter Signalfluss (b) Mehrfach verzweigter Signalfluss

Abbildung 3.6: Beispiele für Signalflüsse von Audiosignalen

Programmiertechnisch kann das Beispiel (a) aus Abbildung 3.6 wie folgt gelöst werden. Zuerst müssen die benötigten Klasseninstanzen erzeugt werden.

```
src = Source(rate=sampleRate, frames_size=frameSize, channels=channels)
bf = DelaySum(channels=channels, frames_size=frameSize, max_offset=max_offset)
ns = NS(rate=sampleRate, channels=1)
```

Anschließend erfolgt die Verlinkung der einzelnen Instanzen miteinander. Da im Signalflussdiagramm alle Module nacheinander ausgeführt werden, kann die Methode `pipeline(*args)` auf die Instanz von Source angewendet werden.

```
src.pipeline(bf, ns)
```

Zum Schluss muss der Signalfluss gestartet werden und am Ende des Programms wieder gestoppt. Dazwischen kann sich eine Schleife mit einer Abbruchbedingung befinden.

```
src.pipeline_start()
while True:
     try:
          time.sleep(1)
     except KeyboardInterrupt:
          print('quit')
          break
src.pipeline_stop()
```

Für das Beispiel (b) aus Abbildung 3.6 sieht der Quellcode folgendermaßen aus. Zuerst müssen auch hier die benötigten Klasseninstanzen erzeugt werden.

```
src = Source(rate=sampleRate, frames_size=frameSize, channels=channels)
ch1 = ChannelPicker(channels=channels, pick=1)
raw_audio_write = FileSink(name=filenameRaw, rate=sampleRate, channels=1)
bf = DelaySum(channels=channels, frames_size=frameSize, max_offset=max_offset)
bf_audio_write = FileSink(name=filenameBf, rate=sampleRate, channels=1)
dbfs = DBFS()
```

Anschließend erfolgt die Verlinkung der einzelnen Instanzen miteinander. Da im Signalflussdiagramm zwei Signalflüsse von Source ausgehen, kann hier die Methode `pipeline(*args)` zweimal auf die Instanz von Source angewendet werden. Die zusätzliche Verlinkung von DelaySum zu DBFS kann mit der Methode `link(sink)` auf die Instanz von DelaySum erreicht werden.

```
src.pipeline(ch1, raw_audio_write)
src.pipeline(bf, bf_audio_write)
bf.link(dbfs)
```

Zum Schluss muss noch der Signalfluss gestartet werden und am Ende des Programms wieder gestoppt. Der Vorgang hier ist derselbe wie auch bei Beispiel a).

```
src.pipeline_start()
while True:
     try:
          time.sleep(1)
     except KeyboardInterrupt:
          print('quit')
          break
src.pipeline_stop()
```

Für eine Übersicht der in der Voice-Engine enthaltenen Klassen ist im Anhang unter A.3 eine Klassenübersicht dargestellt.

4 Analyse und Test der Lokalisation einer Audioquelle

In diesem Kapitel wird der in der Voice-Engine befindliche Algorithmus zur Lokalisation einer Audioquelle analysiert. Weiter werden Simulationen auf Basis des Algorithmus durchgeführt, um seine Leistungsfähigkeit zu überprüfen. Anschließend wird der Algorithmus auf dem ReSpeaker Core v2.0 ausgeführt und getestet. Die Tests werden einmal in einem reflexionsarmen Raum und einmal in einer realen Umgebung durchgeführt. Zum Schluss werden die Ergebnisse zusammengefasst und bewertet.

4.1 Funktionale Analyse des implementierten Algorithmus

Für die Lokalisation der Audioquelle wird hier ein zeitbasiertes Verfahren genutzt, mit dem die Signallaufzeitdifferenz zwischen zwei Mikrofonpaaren bestimmt wird. Der Oberbegriff zu diesem Verfahren lautet Time Differenz of Arrival (TDOA). Anhand der Zeitdifferenzen kann die Position der Audioquelle abgeschätzt werden.

Der Algorithmus zur Lokalisierung einer Audioquelle befindet sich im Voice-Engine Paket in dem Python Skript „*doa_respeaker_v2_6mic_array.py“.* Darin wird zum einen das Modul Element importiert und ein Modul namens *gcc_phat*. Dieses Modul enthält eine Funktion zur Abschätzung der Zeitdifferenz zwischen zwei Signalen. Der Name des Moduls kommt durch die darin verwendete Generalized Cross Correlation - Phase Transform (GCC-PHAT) Methode zustande.

Die Klasse im Lokalisationsmodul heißt `DOA`. Zur Lokalisation einer Audioquelle enthält die Klasse eine Methode namens `get_direction`. Beim Aufruf dieser Methode wird ein Winkel zur Audioquelle anhand des zuvor aufgenommenen Audioblöcken ausgegeben. Zur Verwendung der Methode muss die Klasse initiiert werden. Die Initialisierungsmethode der Klasse sieht wie folgt aus:

```
def __init__(self, rate=16000, chunks=50):
```

Für eine korrekte Lokalisation muss zuerst die Abtastfrequenz (`rate`) übergeben werden. Weiter kann die Anzahl der zwischengespeicherten Audioblöcken für den Lokalisationsalgorithmus übergeben werden (`chunks`). Das Konzept dahinter ist, dass die `get_direction` Methode nach der Detektion eines Schlüsselwortes aufgerufen wird. Anhand der zwischengespeicherten Audioblöcken wird der Lokalisationsalgorithmus über die gesamte Länge des Schlüsselwortes durchgeführt. Es wird dadurch ein Winkel zur Richtung des Sprechers ausgegeben, welcher das Schlüsselwort gesagt hat.

In den folgenden Abschnitten wird zuerst die Funktionsweise der GCC-PHAT Methode erläutert. Anschließend wird der Algorithmus des gcc_phat Moduls dargestellt und zum Schluss wird die Funktionsweise des Lokalisationsalgorithmus in der `get_direction` Methode analysiert und erläutert.

4.1.1 TDOA Berechnung anhand der GCC-PHAT Methode

Die GCC-PHAT Methode basiert auf der Kreuzkorrelation zweier Signale. Grob gesprochen stellt die Korrelation ein Maß für die Ähnlichkeit zweier Signale dar [4]. Bei der Korrelation werden zwei Signale über einen zeitlichen Parameter gegeneinander verschoben. Die Kreuzkorrelation zwischen zwei diskreten Signalen ist definiert durch

$$R_{x_1x_2}(k) = \sum_{n=-\infty}^{\infty} x_1(n) \cdot x_2(n+k)\,. \tag{4.1}$$

Die Kreuzkorrelation lässt sich auch mit Hilfe des Faltungsoperators darstellen durch:

$$R_{x_1x_2}(k) = x_1(-k) * x_2(k)\,. \tag{4.2}$$

In der Praxis kann nur ein endliches Segment des Signals erfasst und verarbeitet werden. Die Kreuzkorrelationsfolge kann daher nur geschätzt werden. Die Gleichung für zwei Signale der Länge *N* ist dargestellt durch [16]

$$\hat{R}_{x_1x_2}(k) = \frac{1}{N} \sum_{n=0}^{N-1-k} x_1(n) \cdot x_2(n+k)\,. \tag{4.3}$$

Bei längeren Signalausschnitten kann mithilfe der Fourier-Transformation der Rechenaufwand verringert werden [5]. Dafür wird auf die Signale eine Fourier-Transformation angewendet, wodurch sich eine einfache Multiplikation im Frequenzbereich ergibt. Die Fourier-Transformation der Kreuzkorrelationsfunktion wird auch als spektrale Kreuzleistungsdichte bezeichnet [17]. Diese ist definiert durch

$$S_{X_1X_2}(f) = X_1(f) \cdot X_2^*(f)\,, \tag{4.4}$$

wobei $(\,.\,)^*$ die komplexe Konjugation kennzeichnet. Anschließend kann über die inverse Fourier-Transformation die Kreuzkorrelationsfunktion berechnet werden.

Wie schon angedeutet, stellt die Korrelation die Ähnlichkeit zweier Signale dar. Wird die Kreuzkorrelation zwischen zwei Signalen berechnet, wovon eins der Signale eine um *D* verschobene Version des anderen darstellt, dann ergibt sich bei der Kreuzkorrelationsfunktion ein Maximum an dem Zeitpunkt *D*. Die Verzögerung kann wie folgt ausgedrückt werden:

$$\hat{D} = \arg\max_k \hat{R}_{x_1x_2}(k)\,. \tag{4.5}$$

In realen Anwendungen können störende Faktoren wie beispielsweise Rauschen und Schallreflexionen die Position des Maximums beeinflussen [18]. Zur Steigerung der Stabilität wurde die Generalized Cross Correlation (GCC) eingeführt [19]. Dabei handelt es sich um Gewichtungsfunktionen, die mit der Kreuzleistungsdichte multipliziert werden. Der allgemeine Ausdruck lautet

$$\hat{R}_{x_1x_2}^{(g)}(\tau) = \mathcal{F}^{-1}\{X_1(f) \cdot X_2^*(f) \cdot \psi(f)\}\,, \tag{4.6}$$

wobei ψ für eine Gewichtungsfunktion steht.

Es stehen mehrere Gewichtungsfunktionen zur Verfügung, die je nach Anwendungsfall unterschiedliche Ergebnisse liefern. Eine der in [19] vorgestellten Gewichtungen ist die Phase Transform (PHAT) Gewichtung. Definiert ist sie als

$$\psi_p(f) = \frac{1}{|X_1(f) \cdot X_2^*(f)|} = \frac{1}{\left|S_{X_1X_2}(f)\right|}. \tag{4.7}$$

Eingesetzt in Gleichung (4.6) ergibt sich das GCC-PHAT zu

$$\hat{R}_{x_1x_2}^{(p)}(\tau) = \mathcal{F}^{-1}\left\{\frac{X_1(f) \cdot X_2^*(f)}{|X_1(f) \cdot X_2^*(f)|}\right\} = \mathcal{F}^{-1}\left\{\frac{S_{X_1X_2}(f)}{\left|S_{X_1X_2}(f)\right|}\right\}. \tag{4.8}$$

Die Position des Maximums von $\hat{R}_{x_1x_2}^{(p)}(\tau)$ entspricht der Verzögerung der Signale zueinander:

$$\widehat{D}_p = \arg\max_{\tau} \hat{R}_{x_1x_2}^{(p)}(\tau)\,. \tag{4.9}$$

Handelt es sich um diskrete Signale, so stellt das $\widehat{D}_p$ über die Abtastfrequenz der Signale eine diskrete Zeiteinheit dar. Anstelle einer Verzögerung resultiert eine Verschiebung.

Vorteile der PHAT Gewichtung sind eine genaue Verzögerungsschätzung bei Breitband- und quasiperiodischen Signalen sowie eine zuverlässige Schätzung in Umgebungen mit rauschähnlichen Störgeräuschen und Schallreflexionen [20]. Daher findet die PHAT Gewichtung bei den Fällen Verwendung, in denen Signale mithilfe von Mikrofonarrays erfasst werden und additive Störungen in Form von Rauschen und Schallreflexionen vorliegen. Denn in solchen Fällen können die Signalverzögerungen unter Verwendung typischer korrelationsbasierter Verfahren nicht genau gefunden werden, da die Korrelationsmaxima nicht präzise extrahiert werden können. Durch die Verwendung der PHAT Gewichtung werden die Maxima in der Korrelationsfunktion schärfer, was zu einer genaueren Messung der Verzögerung führt. Aufgrund der genannten Vorteile wird die PHAT Gewichtung oft bei Sprachsignalen in hallenden Räumen angewendet [18].

Die PHAT Gewichtung hat jedoch ein Makel, der durch die Normierung entstehen. Es kommt dabei vermehrt zu Fehlschätzungen, wenn die Signalenergie gering ist [19]. Insbesondere wenn $S_{X_1X_2}(f)$ in mehreren Frequenzbändern gleich null ist, dann ist die Phase in diesen Frequenzbändern undefiniert. Es kommt zu unregelmäßigen Schätzungen der Phase, die in einem Intervall von $[-\pi\ \pi]$ gleichmäßig verteilt sind. Dadurch ergibt sich nach der inversen Fourier-Transformation eine unbestimmte Signalfolge, die einem weißen Rauschen ähnelt. Die darin enthaltenen Maxima repräsentieren keine sinnvollen Verzögerungen, was bei einer Auswertung zu einer Fehlschätzung führt.

4.1.2 GCC-PHAT Algorithmus

Der GCC-PHAT Algorithmus befindet sich im Voice-Engine Paket in dem Python Skript *„gcc_phat.py“.* Das Modul beinhaltet die `gcc_phat` Funktion, in der die Zeitdifferenz zwischen zwei Signalen berechnet wird.

Der `gcc_phat` Funktion können fünf Parameter übergeben werden. Die Parameter `sig` und `refsig` sind Arrays mit den einzelnen Abtastwerten der jeweiligen Signale. Der Parameter `fs` steht für die Abtastrate der Signale, `max_tau` ist die größtmöglich auftretende Zeitdifferenz zwischen den Signalen und `interp` ist ein Faktor, mit dem die generalisierte Korrelationsfolge zusätzlich interpoliert wird, um eine feinere Abstufung der Zeitdifferenz zu ermöglichen.

```
def gcc_phat(sig, refsig, fs=1, max_tau=None, interp=1):
```

Eine Multiplikation im Frequenzbereich der diskreten Fourier Transformation entspricht einer periodischen Faltung im Zeitbereich, wohingegen in diesem Fall eine aperiodische Faltung notwendig ist, wie sie auch bei der Berechnung von Systemantworten verwendet wird [4]. Um die verursachten Umlauffehler [16] zu vermeiden, muss die Signallänge bei der FFT die Bedingung $N \geq N_x + N_y - 1$ erfüllen [4]. Aufgrund der Bedingung wird in der nächsten Zeile die Anzahl der Abtastwerte beider Signale ermittelt und miteinander addiert.

```
    n = sig.shape[0] + refsig.shape[0]
```

Anschließend wird das Spektrum der Signale anhand einer *n*-Punkt FFT berechnet. Da das übergebene *n* aufgrund der vorherigen Zeile länger als das Signal ist, wird das Signal bis zur Länge *n* mit Nullen aufgefüllt (zero padding). Es resultiert ein Spektrum mit *n* Frequenzbänder.

```
    SIG = np.fft.rfft(sig, n=n)
    REFSIG = np.fft.rfft(refsig, n=n)
```

Die GCC-PHAT Berechnung wird über zwei Zeilen durchgeführt. Zuerst wird die spektrale Kreuzleistungsdichte (vergl. Gleichung (4.4)) berechnet und in der Variable *R* hinterlegt. Anschließend wird in der zweiten Zeile die generalisierte Kreuzkorrelationsfolge nach Gleichung (4.8) berechnet. Bei der Funktion zur Berechnung der IFFT kann die Anzahl der Frequenzbänder angegeben werden, die für die Berechnung der IFFT verwendet werden sollen. Ist die Angabe höher als die zur Verfügung stehenden Frequenzbänder, so werden die zusätzlichen Frequenzbänder mit Nullen aufgefüllt. Dadurch kann die Auflösung der ausgegebenen Folge erhöht werden. In diesem Fall wird über die `interp` Variable die Anzahl der Frequenzbänder um ein Vielfaches erhöht. Es resultiert eine generalisierte Kreuzkorrelationsfolge mit einer feineren Auflösung und der damit verbundenen kleineren Zeitkonstante.

```
    R = SIG * np.conj(REFSIG)
    cc = np.fft.irfft(R / np.abs(R), n=(interp * n))                    # (*)
```

Als nächstes wird die maximal mögliche Verschiebung beider Signale berechnet. Wird keine maximale Zeitdifferenz in Sekunden (`max_tau`) der Funktion übergeben, so wird die Verschiebung auf den maximalen Wert gesetzt. Dieser ergibt sich durch die Signallänge multipliziert mit dem

Interpolationsfaktor. Wird eine maximale Zeitdifferenz übergeben, so wird diese in eine maximale Verschiebung umgerechnet.

```
    max_shift = int(interp * n / 2)
    if max_tau:
        max_shift = np.minimum(int(interp * fs * max_tau), max_shift)
```

Zur Darstellung von negativen und positiven Verzögerungen, wird die berechnete generalisierte Kreuzkorrelationsfolge verschoben und auf den durch `max_tau` berechneten Ausschnitt getrimmt. Durch die FFT mit einer doppelten Signallänge, erhält man nach der IFFT eine Folge mit der doppelten Länge des Signalausschnitts. Aufgrund der durch die IFFT resultierenden Eigenschaft der Periodizität der Folge, kann die linke Hälfte der Folge als eine positive und die rechte Hälfte als eine negative Verschiebung dargestellt werden. Mathematisch kann es durch $x[(-k)_N] = x[N-k]$ beschrieben werden [4]. Durch Vertauschen beider Ausschnitte ergibt sich in der Mitte der Folge eine Verschiebung von 0. Die Folge stellt dadurch eine Verschiebung von `-max_tau` bis `max_tau` dar.

```
    cc = np.concatenate((cc[-max_shift:], cc[:max_shift + 1]))
```

In der nächsten Zeile wird die Position des ersten Maximums in dem Ausschnitt der generalisierten Kreuzkorrelationsfolge ermittelt. Ist ein Maximum gefunden, so wird der Index ausgegeben und die Verschiebung (`shift`) berechnet.

```
    shift = np.argmax(np.abs(cc)) - max_shift
```

Die ermittelte Verschiebung wird über die Abtastrate des Signals und des Interpolationsfaktors in eine Zeit in Sekunden umgerechnet.

```
    tau = shift / float(interp * fs)
```

Zum Schluss werden die ermittelte Zeitdifferenz und die generalisierte Kreuzkorrelationsfolge zurückgegeben.

```
    return tau, cc
```

Eine Problematik in dem Algorithmus ergibt sich in der markierten Zeile (siehe S. 21) zur Berechnung der GCC-PHAT Methode. Dort findet eine Division durch einen Wert statt, welcher sich gegebenenfalls zu Null ergibt. Darauf reagiert das System mit einer auftretenden Fehlermeldung während der Laufzeit.

4.1.3 Lokalisierungsalgorithmus über TDOA

Der hier verwendete Lokalisationsalgorithmus liefert lediglich einen Winkel zur Audioquelle. Der Algorithmus befindet sich in der Methode `get_direction`, welche beim Aufruf den ermittelten Winkel ausgibt. Die Grundidee des Algorithmus ist es die sechs Mikrofone paarweise in drei lineare Mikrofonarrays aufzuteilen und anhand der Signallaufzeitdifferenzen jeweils einen Winkel zu

bestimmen. Dafür bilden die gegenüberliegenden Mikrofone ein Paar miteinander (MIC1 mit MIC4, MIC2 mit MIC5 und MIC3 mit MIC6).

```
self.pair = [[0, 3], [1, 4], [2, 5]]
```

Für jedes Mikrofonpaar wird eine Signallaufzeitdifferenz ermittelt und anhand dieser ein Winkel zur Audioquelle berechnet. Es ergeben sich also drei Signallaufzeitdifferenzen (`tau`) und drei Winkel (`theta`). Dafür werden zwei Arrays erstellt.

```
tau = [0, 0, 0]
theta = [0, 0, 0]
```

Anschließend können die Signallaufzeitdifferenzen und Winkel berechnet werden. Dazu wird der binäre Audiostream der Mikrofone in ein Integer Datentyp umgewandelt und in ein Array abgelegt.

```
buf = b''.join(self.queue)
buf = np.fromstring(buf, dtype='int16')
```

Über eine Schleife werden dann die Signallaufzeitdifferenzen zwischen den Mikrofonpaaren mithilfe der `gcc_phat` Funktion (siehe Kapitel 4.1.2) berechnet. Der Funktion werden durch den Ausdruck `buf[v[0]::8], buf[v[1]::8]` die Abtastwerte des jeweiligen Mikrofonpaares übergeben. Der Rückhabewert der Funktion ist die Signallaufzeitdifferenz des Mikrofonpaares. Dieser wird in dem Array `tau` abgelegt. Anschließend wird der Winkel berechnet und im Array `theta` abgelegt.

```
for i, v in enumerate(self.pair):
     tau[i], _ = gcc_phat(buf[v[0]::8], buf[v[1]::8], fs=self.sample_rate,
                          max_tau=MAX_TDOA_6, interp=1)
     theta[i] = np.arcsin(tau[i] / MAX_TDOA_6) * 180 / np.pi
```

Die Berechnung des Winkels anhand der Signallaufzeitdifferenz zwischen zwei Mikrofonsignalen findet hier durch folgende Gleichung statt:

$$\theta = \sin^{-1}\left(\frac{\tau}{\tau_{Max}}\right). \tag{4.10}$$

Das τ ist die Signallaufzeitdifferenz und τ_{Max} ist die maximale Laufzeit zwischen zwei Mikrofonen. Im Quellcode ist das τ_{Max} (`MAX_TDOA_6`) definiert als

$$\tau_{Max} = \frac{d}{c}, \tag{4.11}$$

wobei d der Abstand der beiden Mikrofone zueinander und c die Schallgeschwindigkeit ist. Im Quellcode sieht die Berechnung folgendermaßen aus:

```
SOUND_SPEED = 340.0
MIC_DISTANCE_6 = 0.09218
MAX_TDOA_6 = MIC_DISTANCE_6 / float(SOUND_SPEED)
```

Setzt man nun τ_{Max} aus Gleichung (4.11) in die Gleichung (4.10) ein, so ergibt sich daraus die allgemeine Gleichung zur Berechnung des Winkels anhand der Signallaufzeitdifferenz für eine lineare Mikrofonanordnung [1]:

$$\theta = \sin^{-1}\left(\frac{c \cdot \tau}{d}\right). \tag{4.12}$$

Das c ist die Schallgeschwindigkeit, τ ist die Signallaufzeitdifferenz und d ist der Abstand der beiden Mikrofone zueinander.

Die Gleichung kann anhand von geometrischen Gegebenheiten hergeleitet werden. Diese ist jedoch nur gültig, wenn die eingehenden Schallwellen dem Modell einer ebenen Welle entsprechen. Ist die dafür notwendige Bedingung erfüllt (vergl. (2.3)), so kann die Gleichung (4.12) mithilfe der Abbildung 4.1 hergeleitet werden. Durch das Modell der ebenen Welle und der Anwendung von trigonometrischen Funktionen, kann die Strecke berechnet werden, die die Wellenfront zum Passieren beider Mikrofone benötigt (siehe Abbildung 4.1). Anhand der Wellenausbreitungsgeschwindigkeit, in diesem Fall die Schallgeschwindigkeit, kann die dafür benötigte Zeit berechnet werden. Zusammengefasst ergibt sich die Signallaufzeitdifferenz durch die folgende Gleichung [1]:

$$\tau = \frac{d\ \sin(\theta)}{c}. \tag{4.13}$$

Durch umstellen nach θ ergibt sich die Gleichung (4.12).

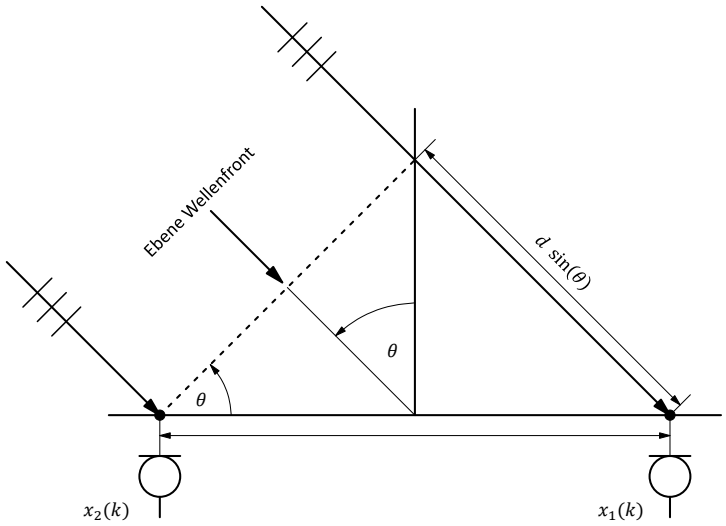


Abbildung 4.1: Darstellung zur Berechnung des Winkels zwischen zwei Mikrofonen über τ

Ein Winkel von 0° deutet auf eine Wellenfront orthogonal zur Mikrofonachse. Ein Winkel von ±90° deutet auf eine Wellenfront entlang der Mikrofonachse an. In Bezug auf die Signallaufzeitdifferenz τ ergibt sich bei $\tau = 0s$ ein Winkel $\theta = 0°$, bei $\tau = \tau_{Max}$ ein Winkel $\theta = 90°$ und bei $\tau = -\tau_{Max}$ ein Winkel $\theta = -90°$.

Aufgrund des Verlaufes des Arkussinus ist der Winkel θ nur in einem Bereich von -90° bis 90° fest bestimmt. Wenn man einen vollen Kreis mit dem Wertebereich von -180° bis 180° betrachtet, so liefert der Arkussinus zwei mögliche Ergebnisse bei einem Wert. In Verbindung mit dem linearen Mikrofonarray ist ein definierter Winkel nur gegeben, wenn $\tau = \pm\tau_{Max}$ ist. Dies entspricht einer Wellenfront entlang der Mikrofonachse. In den Fällen $\tau \neq \pm\tau_{Max}$ existieren, auf dem vollen

Kreis gesehen, zwei Winkel. Bei den resultierenden Winkeln handelt es sich um zwei an der Mikrofonachse gespiegelte Winkel. Dadurch kann man keinen Aufschluss darüber geben, ob die Wellenfront oberhalb oder unterhalb der Mikrofonachse anliegt. Anhand eines linear angeordneten Mikrofonarrays ist es somit nur in einem Bereich von -90° bis 90° möglich einen definierten Winkel auszugeben. Die Abbildung 4.2 stellt die Winkel in Bezug zu τ in einem vollen Kreis dar.

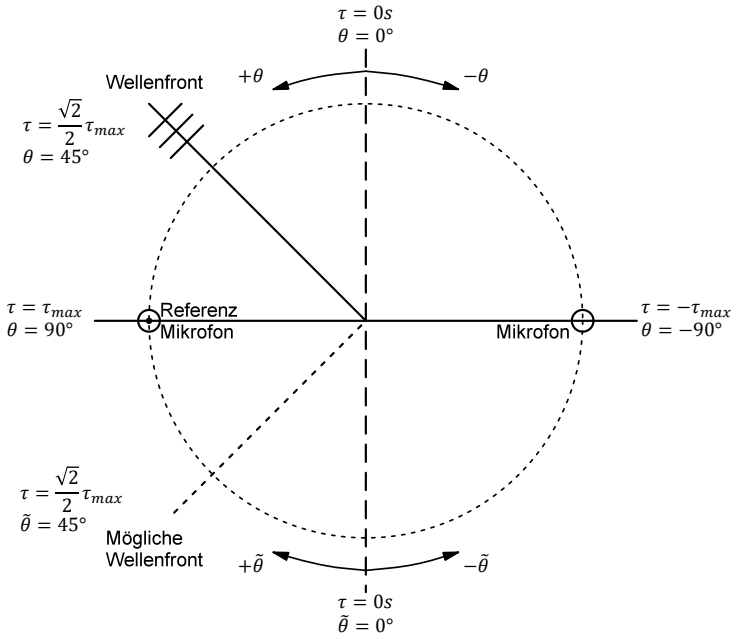


Abbildung 4.2: Darstellung des Winkels zwischen zwei Mikrofonen

Um einen Winkel zur Audioquelle ausgeben zu lassen, reicht ein Mikrofonpaar aus. Es ergeben sich jedoch zwei mögliche Winkel, in denen sich die Audioquelle befinden könnte. Um eine Aussage zu treffen, welcher der beiden Winkel die Audioquelle repräsentiert, werden die berechneten Winkel der weiteren Mikrofonpaare einbezogen. Durch die hier paarweise angeordneten Mikrofone ergeben sich drei um 60° versetzte lineare Mikrofonarrays. Aufgrund der versetzten Anordnung der Mikrofonpaare zueinander, kann im Gegensatz zu einer linieren Mikrofonanordnung, ein definierter Winkel zur Audioquelle ausgegeben werden.

Um einen definierten Winkel in Bezug zum Mikrofonarray zu generieren, wird hier eine Fallunterscheidung anhand der Informationen aus den drei Mikrofonpaaren durchgeführt.

```
min_index = np.argmin(np.abs(tau))
if (min_index != 0 and theta[min_index - 1] >= 0) or
   (min_index == 0 and theta[len(self.pair) - 1] < 0):
     best_guess = (theta[min_index] + 360) % 360
else:
     best_guess = (180 - theta[min_index])
best_guess = (best_guess + 30 + min_index * 60) % 360
```

Grundlegend wird in der Fallunterscheidung zuerst untersucht, welcher der drei Mikrofonpaare einen Winkel am nächsten zur Null hat. Basierend auf dem ausgewählten Mikrofonpaar, wird der Winkel eines versetzten Mikrofonpaares überprüft. Je nachdem, ob der Winkel des versetzten Mikrofonpaares positiv oder negativ ist, wird basierend auf dem ausgewählten Mikrofonpaar

entschieden, welcher seiner zwei Winkel die Richtung zur Audioquelle darstellt. Zum Schluss wird mit dem ausgewählten Winkel und der Position des Mikrofonpaares auf dem Board, ein absoluter Winkel berechnet. Durch die Berechnungen und der Fallunterscheidung im Quellcode ist der ausgegebene Winkel im Uhrzeigersinn positiv ausgelegt und entspricht an der Position von MIC2 einem Winkel von 0°. Der Wertebereich des Ausgabewinkels beträgt 0° bis 359°. Die Abbildung 4.3 zeigt die drei Mikrofonpaare mit ihrem Wertebereich an.

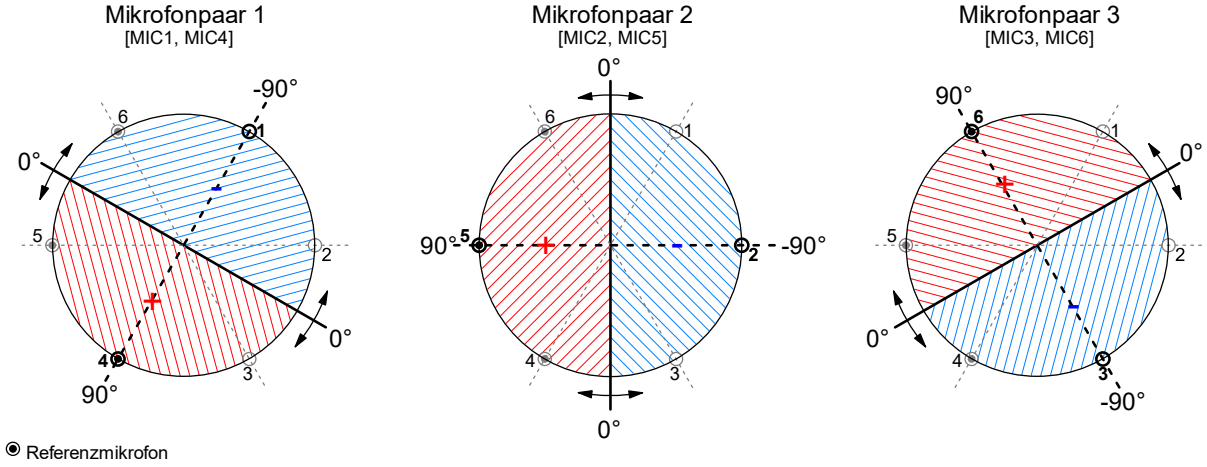


Abbildung 4.3: Darstellung der positiven und negativen Wertebereiche von den Mikrofonpaaren

Auf den vollen Kreisumfang gesehen, entstehen durch die verwendete Methode sechs Bereiche, in denen jeweils ein Mikrofonpaar als Referenz für den Ausgabewinkel φ dient. Durch die Fallunterscheidung und die paarweise Aufteilung der Mikrofone, umschließt jedes Mikrofonpaar einen Bereich von 2-Mal -30° bis +30°. In Abbildung 4.4 sind die Wertebereiche der Mikrofonpaare dargestellt und damit auch wann welches Mikrofonpaar als Referenz für die Winkelausgabe dient.

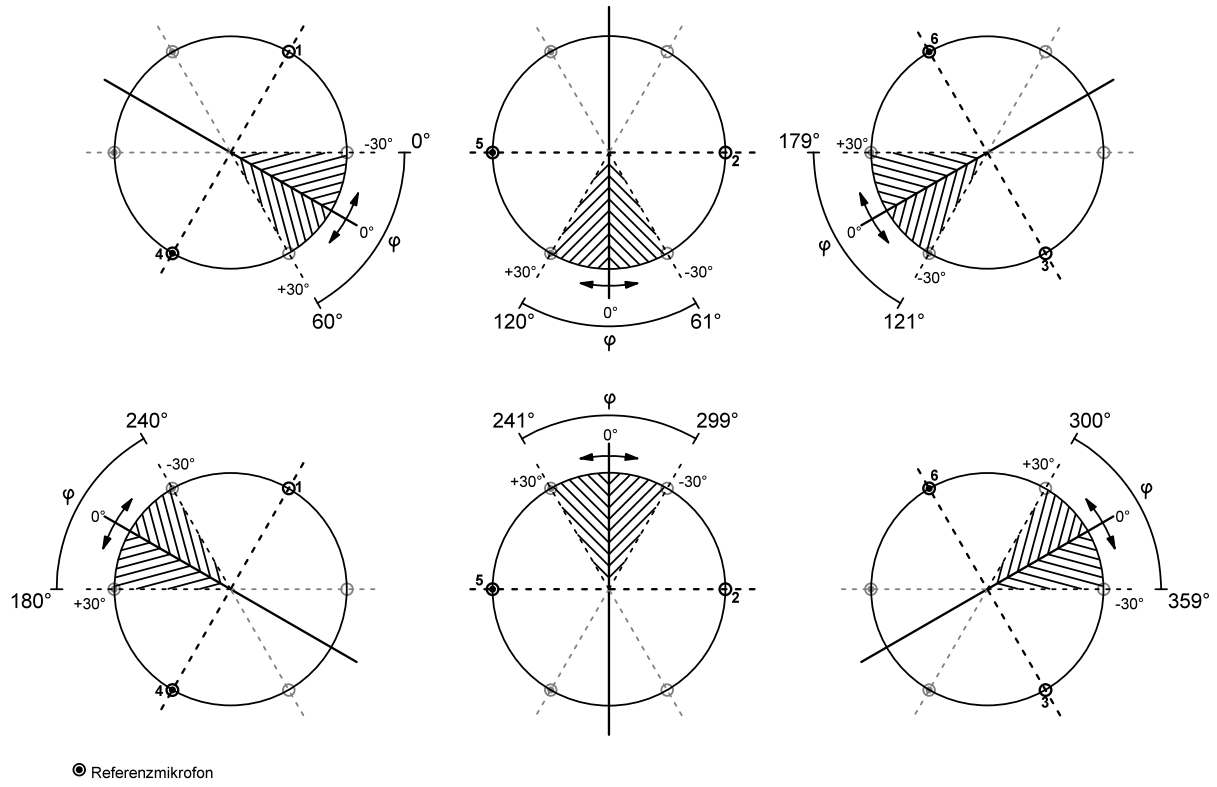


Abbildung 4.4: Darstellung des aktiven Mikrofonpaares und dessen Wertebereich in Abhängigkeit der Richtung der Audioquelle.

Um einen festen Punkt in einem dreidimensionalen Raum zu beschreiben, sind drei Parameter notwendig. In einem Kugelkoordinatensystem sind es der Azimutwinkel, der Elevationswinkel und die Distanz. Ein lineares Array liefert anhand der Gleichung (4.12) nur einen Wert. Man genügt sich oft mit dem Azimutwinkel. Der berechnete Winkel ist jedoch nicht direkt der Azimutwinkel. Bei dem berechneten Winkel handelt es sich um den sogenannten *broadside* Winkel. In einem Kugelkoordinatensystem befindet sich der Winkel zwischen der orthogonalen Ebene der Mikrofonachse und der Richtung der Schallwelle (siehe Abbildung 4.5).

Der broadside Winkel ist über die folgende Gleichung mit dem Azimut- und Elevationswinkel verknüpft [1]:

$$\theta = \sin^{-1}(\sin(az)\cos(el))\,. \tag{4.14}$$

Dabei steht az für den Azimutwinkel und el für den Elevationswinkel. Anhand der Gleichung ist zu erkennen, dass der broadside Winkel nur mit dem Azimutwinkel übereinstimmt, wenn der Elevationswinkel gleich Null ist. Bei einem Azimutwinkel ungleich Null und mit steigendem Elevationswinkel, weicht der broadside Winkel immer weiter vom Azimutwinkel ab.

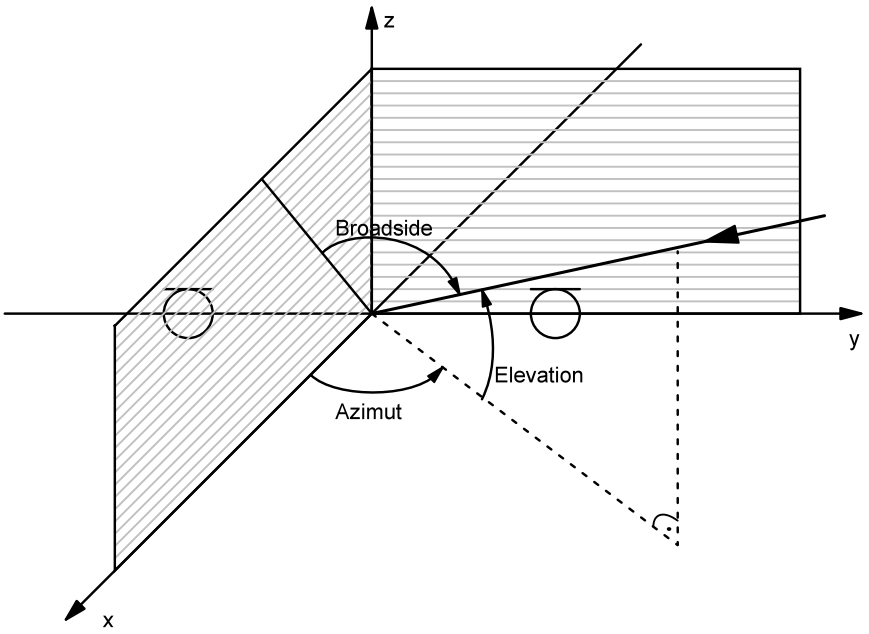


Abbildung 4.5: Darstellung des broadside Winkels in Bezug zum Azimut- und Elevationswinkel

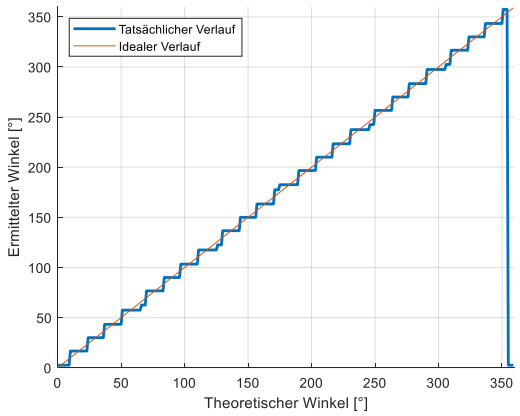
4.2 Überprüfung des Algorithmus per Simulation

In diesem Abschnitt wird der Lokalisationsalgorithmus anhand von Simulationen auf seine Leistungsfähigkeit geprüft. Die Simulationen werden in MATLAB durchgeführt. Dafür wird der Algorithmus zur Lokalisierung einer Audioquelle sinngemäß in die Programmierumgebung von MATLAB implementiert. Zum Schluss wird eine Laufzeituntersuchung des Lokalisationsalgorithmus auf dem ReSpeaker Core v2.0 durchgeführt.

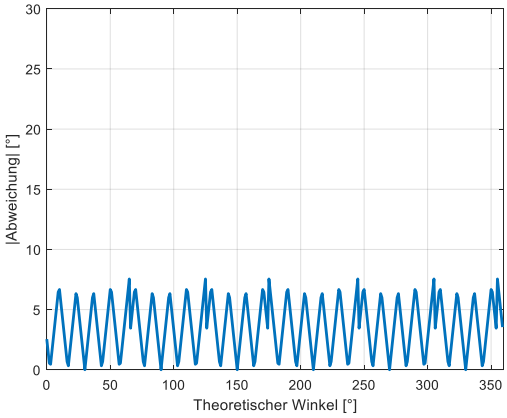
4.2.1 Überprüfung der Lokalisationsgenauigkeit

Zur Überprüfung der Lokalisationsgenauigkeit wird eine Audioquelle simuliert, die in einer vorgegebenen Distanz und in einem bestimmten Elevationswinkel um das Mikrofonarray rotiert. Die Audioquelle wird in einem Bereich von 0° bis 360° (Azimut) mit einer Abstufung von 1° rotiert. Der Ausgabewinkel des Lokalisationsalgorithmus wird in Bezug zum Azimutwinkel dargestellt.

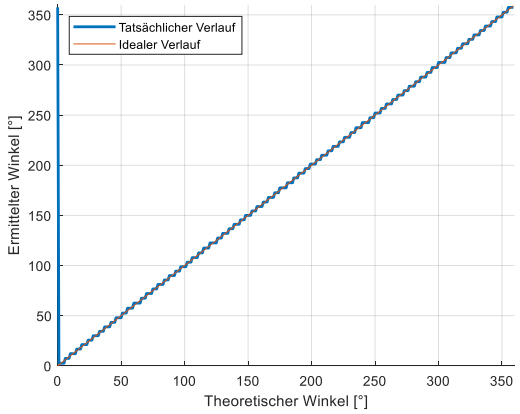
In der ersten Simulation wird der Elevationswinkel auf 0° eingestellt und die Distanz auf 2 m. Die Abtastfrequenz wird einmal auf 16000 Hz und einmal auf 48000 Hz gesetzt. Der Interpolationsfaktor wird standardmäßig auf eins belassen. Als Signal dient ein weißes Rauschen. Dies stellt, aufgrund seines breiten Frequenzspektrums, optimale Bedingungen für die GCC-PHAT Methode. Zusätzlich zur Winkelmessung wird die absolute Abweichung aufgetragen. In Abbildung 4.6 sind die Simulationsergebnisse zu sehen.



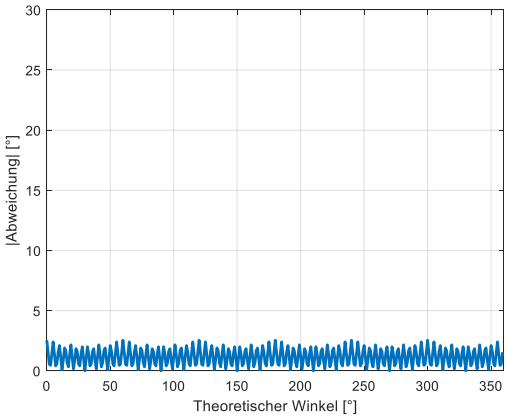
(a) Ermittelter und simulierter Winkel fs=16000 Hz



(b) Absolute Abweichung fs=16000 Hz



(c) Ermittelter und simulierter Winkel fs=48000 Hz

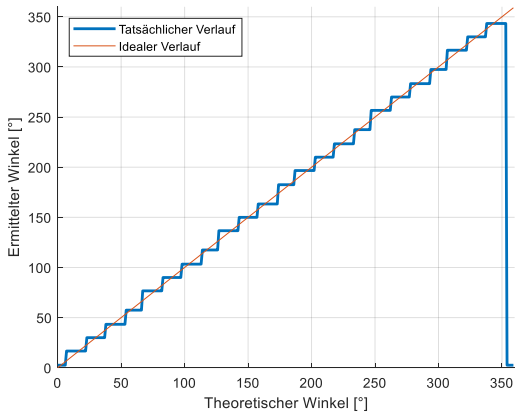


(d) Absolute Abweichung fs=48000 Hz

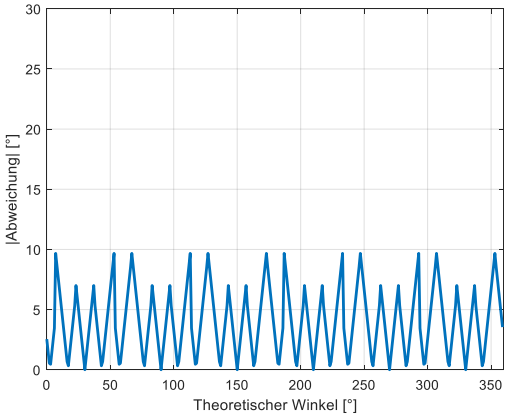
Abbildung 4.6: Ermittelter und simulierter Winkel mit absoluter Abweichung bei Abtastfrequenzen von 16000 Hz und 48000 Hz und einem Elevationswinkel von 0°

An den Simulationen ist erkennbar, dass der ermittelte Winkel bestimmte Abstufungen aufweist. Diese Abstufungen entstehen durch die diskrete Zeitkonstante aus der Kreuzkorrelation mit diskreten Mikrofonsignalen. Vergleicht man die ermittelten Winkel und die absoluten Abweichungen bei einer Abtastfrequenz von 16000 Hz mit denen bei einer Abtastfrequenz von 48000 Hz, so ist erkennbar, dass mit einer höheren Abtastrate eine höhere Auflösung und somit eine geringere absolute Abweichung möglich ist.

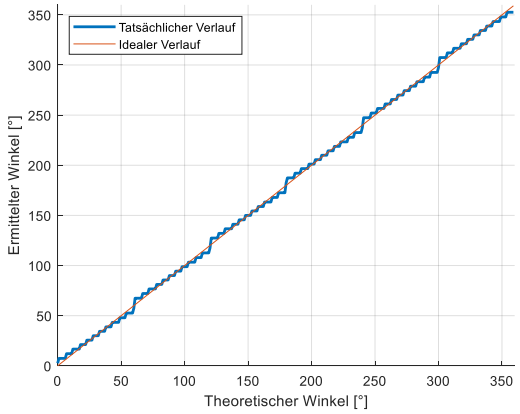
In der zweiten Simulation wird der Elevationswinkel auf 30° angehoben. Die restlichen Parameter bleiben dieselben wie auch bei der ersten Simulation. Die Simulationsergebnisse sind in Abbildung 4.7 dargestellt



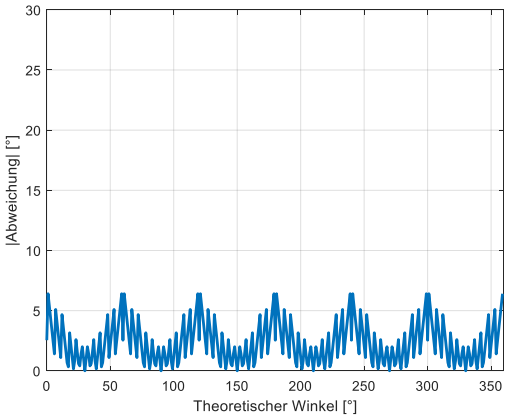
(a) Ermittelter und simulierter Winkel fs=16000 Hz



(b) Absolute Abweichung fs=16000 Hz



(c) Ermittelter und simulierter Winkel fs=48000 Hz

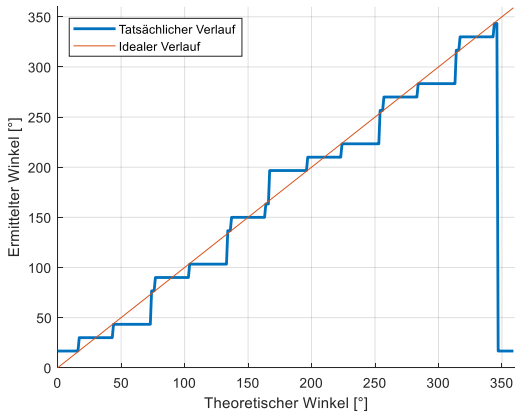


(d) Absolute Abweichung fs=48000 Hz

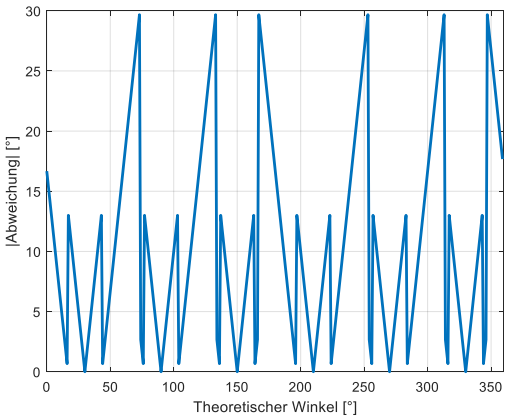
Abbildung 4.7: Ermittelter und simulierter Winkel mit absoluter Abweichung bei Abtastfrequenzen von 16000 Hz und 48000 Hz und einem Elevationswinkel von 30°

Beim genauen Betrachten der Simulationsergebnisse, ist eine leicht gröbere Abstufung der Winkel zu erkennen. Schaut man sich die absolute Abweichung an, so sind bei einer Abtastrate von 16000 Hz einige erhöhte Spitzen sichtbar und bei einer Abtastrate von 48000 Hz ist ein dreieckförmiger Verlauf erkennbar. Der Verlauf der Abweichung ist periodisch mit einem Maximum alle 60°. Schaut man sich den dazu ermittelten Winkelverlauf an, so ist zu erkennen, dass abflachende Bereiche entstehen. In den Bereichsgrenzen alle 60° ist ein deutlicher Sprung sichtbar. Die sechs Bereiche entstehen durch die drei Mikrofonpaare und der damit zugehörigen Fallunterscheidung. Jedes Mikrofonpaar deckt durch ihre um 60° versetzte Anordnung einen auf das Mikrofonpaar basierten Bereich von -30° bis +30° ab (vergl. Abbildung 4.4). Aufgrund der Tatsache, dass der ermittelte Winkel nicht dem Azimutwinkel entspricht, entsteht anhand der Umrechnungsgleichung (4.14) eine steigende Abweichung zu den Bereichsgrenzen.

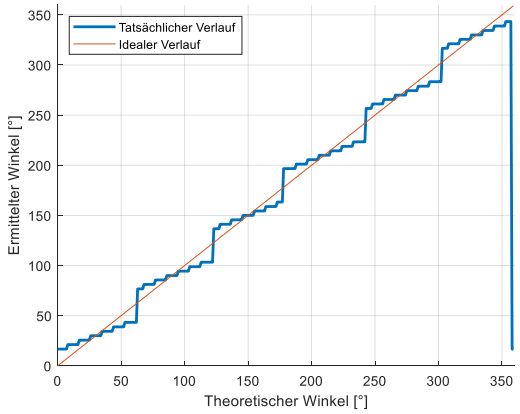
Noch deutlicher sind diese Effekte bei einem höheren Elevationswinkel zu sehen. Die Abbildung 4.8 zeigt die Simulationsergebnisse bei einem Elevationswinkel von 60°.



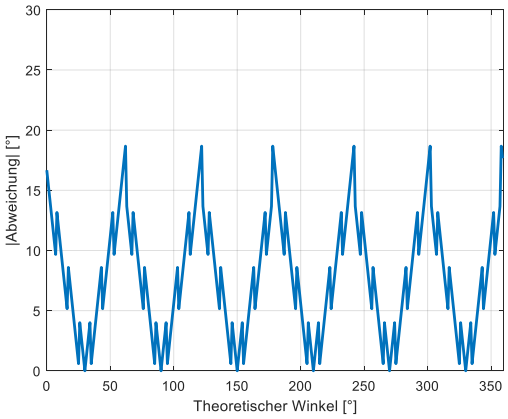
(a) Ermittelter und simulierter Winkel fs=16000 Hz



(b) Absolute Abweichung fs=16000 Hz



(c) Ermittelter und simulierter Winkel fs=48000 Hz



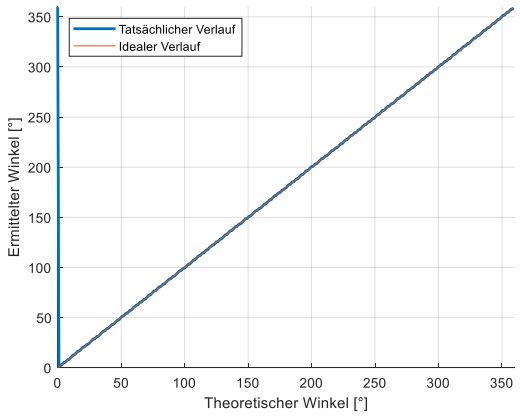
(d) Absolute Abweichung fs=48000 Hz

Abbildung 4.8: Ermittelter und simulierter Winkel mit absoluter Abweichung bei Abtastfrequenzen von 16000 Hz und 48000 Hz und einem Elevationswinkel von 60°

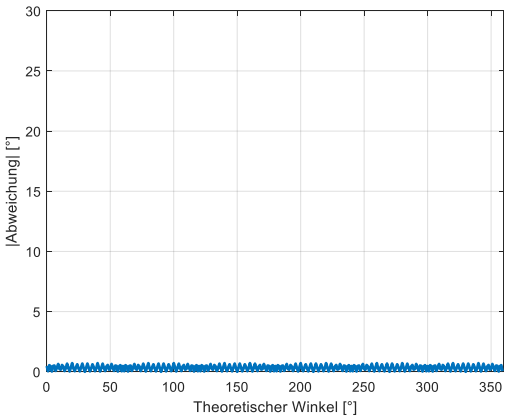
Bei einer Abtastrate von 16000 Hz sind bei den ermittelten Winkeln weniger Abstufungen sichtbar. Die Abweichungen sind im Vergleich zu der Simulation mit einem Elevationswinkel von 0° deutlich erhöht und enthalten ausgeprägte Spitzen alle 60°. Bei einer Abtastrate von 48000 Hz sind bei den ermittelten Winkeln große Sprungstellen zu sehen. Dies äußert sich auch bei den Abweichungen mit einem dreieckigen Verlauf.

Zum Schluss wird nochmal der Einfluss des Interpolationsfaktors untersucht. Dafür wird eine Abtastrate von 16000 Hz, ein Elevationswinkel von 0° und ein Interpolationsfaktor von 10 eingestellt. Zu sehen sind die Simulationsergebnisse in Abbildung 4.9.

Durch die hohe Interpolation sind bei den ermittelten Winkeln kaum Abstufungen zu erkennen (siehe Abbildung 4.9 (a)). Dies ist auch bei den Abweichungen einsehbar (siehe Abbildung 4.9 (b)). Im Vergleich zu Abbildung 4.6 (b) ist die maximale Abweichung durch die 10-fache Interpolation um einen Faktor von 10 geringer.



(a) Ermittelter und simulierter Winkel



(d) Absolute Abweichung

Abbildung 4.9: Ermittelter und simulierter Winkel mit absoluter Abweichung bei Abtastfrequenzen von 16000 Hz, einem Elevationswinkel von 0° und einem Interpolationsfaktor von 10

4.2.2 Auswirkung der Blocklängenwahl bei einem Sprachsignal

Da die GCC-PHAT Methode für breitbandige Signale ausgelegt ist, wird untersucht, ab welcher Blocklänge eine zuverlässige Lokalisation eines Sprachsignals möglich ist. Wie in Abschnitt 2.3 angedeutet, ist ein Ausschnitt eines Sprachsignal ab einer bestimmten Dauer als breitbandig anzunehmen. Es wird demzufolge überprüft, ab welcher Dauer die Bandbreite des Sprachsignals ausreicht, sodass über die GCC-PHAT Methode stabile Winkel ausgegeben werden.

Dafür wird ein aufgenommenes Sprachsignal aus einer festen Richtung ausgegeben. Das verwendete Sprachsignal ist im Anhang unter B.1 dargestellt. In jedem Durchgang wird das gesamte Signal mit einer festgelegten Blocklänge in den Lokalisationsalgorithmus eingespeist. Anschließend wird geprüft, wie viele der ermittelten Winkel im angegebenen Toleranzbereich liegen. Für die Simulation wird eine Abtastfrequenz von 16000 Hz gewählt. Der Azimut- und Elevationswinkel wird auf 0° eingestellt und die Toleranz auf ±8° gesetzt.

Die Abbildung 4.10 stellt die Fehlmessungen, die außerhalb des Toleranzbereichs liegen, prozentual in Bezug zur Blocklänge an. Bei geringen Blocklängen ist die Fehlerrate hoch. Ab einer Blocklänge über 1800 Abtastwerten sinkt die Fehlerrate auf unter 10 % ab. Auf die Abtastrate gesehen entspricht es einer Zeit von 125 ms. Eine Fehlerrate von 0 % ist erst ab Blocklänge von über 10000 Abtastwerten gegeben. Bei einer Abtastzeit von 16000 Hz entspricht es einer Zeit von 625 ms. Die Fehlerrate variiert jedoch je nach verwendetem Sprachsignal und darf nicht als absolut angesehen werden.

Überwiegend ergeben sich Fehlmessungen in den Blöcken, bei denen das Sprachsignal einen stationären und periodischen Verlauf annimmt. In diesen Blöcken ist das Signal nicht ausreichend breitbandig, wodurch das Verfahren versagt. Weiter kommt es in den Sprachpausen zu Fehlmessungen, weil die Signalenergie in diesen Bereichen gering ist.

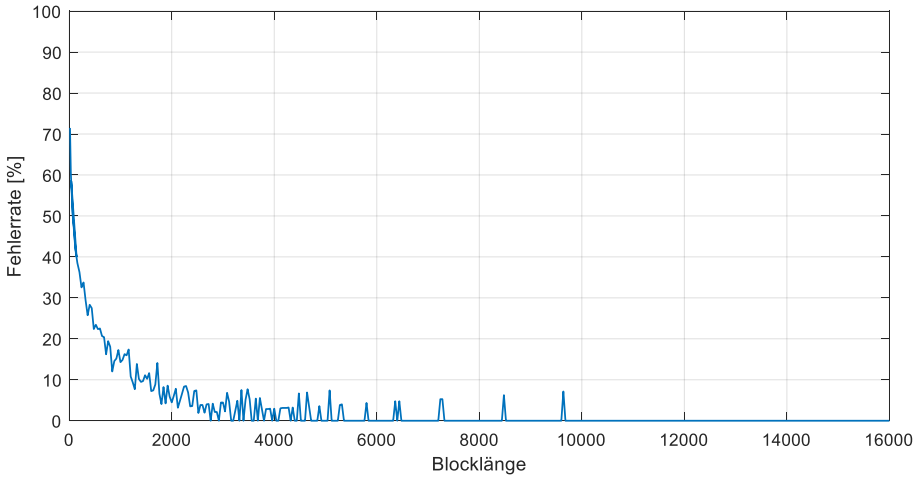


Abbildung 4.10: Lokalisationsfehlerrate pro Blocklänge bei einem Sprachsignal

4.2.3 Laufzeituntersuchung des Lokalisationsalgorithmus

Ein wichtiger Punkt bei der Untersuchung von Algorithmen ist die Messung der benötigten Berechnungszeit auf der Zielhardware. Dafür wird auf dem ReSpeaker Core v2.0 in den Lokalisationsalgorithmus ein zufallsgeneriertes Signal mit unterschiedlicher Blocklänge eingespeist. Die Berechnungszeit wird über eine Differenzzeitmessung durchgeführt. Es finden 1000 Berechnungen pro Block statt. Anschließend wird über die vorliegenden Zeitmessungen die mittlere Berechnungszeit ermittelt. Gemessen wird einmal mit einem Interpolationsfaktor von 1 und einmal mit einem Interpolationsfaktor von 10. Die Messergebnisse sind in Abbildung 4.11 dargestellt.

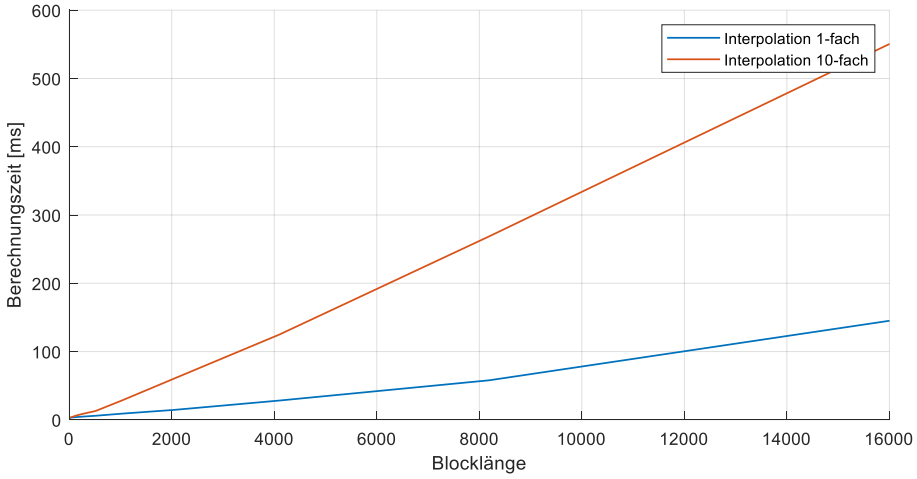


Abbildung 4.11: Berechnungszeit des Lokalisationsalgorithmus pro Blocklänge

Zu sehen ist, dass mit steigender Blocklänge die Berechnungszeit relativ linear ansteigt. Mit einem zusätzlichen Interpolationsfaktor von 10 ist die Steigung 4-mal höher.

Es stellt sich nun die Frage, ob die Hardware überhaupt leistungsfähig genug ist, die Berechnungen durchzuführen, bevor der nächste Block bereitsteht. Jeder Block stellt über die Abtastrate eine gewisse Zeitdauer dar. Diese ist über die Formel

$$Zeitdauer = \frac{Blocklänge}{Abtastrate} \tag{4.15}$$

gegeben. Anhand der Zeitdauer und der Berechnungszeit kann darauf geschlossen werden, wie lang die Blocklänge gewählt werden muss, sodass die Berechnungszeit noch unter der Zeitdauer liegt. Dafür wird die Differenz zwischen der Zeitdauer und der Berechnungszeit über die Blocklänge aufgetragen. Bei einer positiven Differenz ist die Berechnungszeit kürzer als die Dauer, die der Block darstellt. Ist die Differenz negativ, so benötigt die Berechnung eine längere Zeit als der Block darstellt. In diesem Fall kommt die Hardware mit der Berechnung nicht hinterher, wodurch sich die Datenpakete aufstauen und es zu Verzögerungen kommt. In Abbildung 4.12 ist die Zeitdifferenz über die Blocklänge für eine Abtastrate von 16000 Hz, 48000 Hz und 16000 Hz mit einem Interpolationsfaktor von 10 dargestellt.

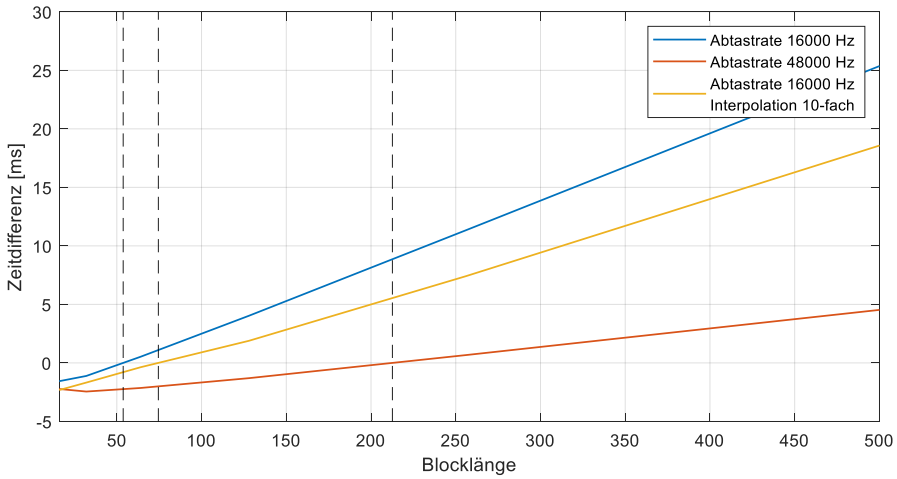


Abbildung 4.12: Zeitdifferenz zwischen Block- und Berechnungszeit in Bezug zur Blocklänge

Bei einer Abtastrate von 16000 Hz und einer Blocklänge von 55 Abtastwerten, ist die Berechnung schneller als die Zeitdauer, die die Blocklänge durch die Abtastzeit darstellt. Bei einer Abtastrate von 48000 Hz ist erst ab einer Blocklänge von 215 Abtastwerten die Berechnungszeit kürzer. Bei einer Abtastrate von 16000 Hz und einem Interpolationsfaktor von 10 ist nach 75 Abtastwerten die Berechnungszeit kürzer. Vergleicht man die absolute Abweichung bei einer Abtastrate von 48000 Hz (siehe Abbildung 4.6 (d)) mit der bei 16000 Hz und einem Interpolationsfaktor von 10 (siehe Abbildung 4.9 (b)), so ist die Variante mit der geringeren Abtastrate und Interpolation genauer und schneller.

4.3 Test des Algorithmus auf der Hardware

Die nachfolgenden Tests untersuchen die Leistungsfähigkeit des Lokalisationsalgorithmus in Verbindung mit dem ReSpeaker Core v2.0. Untersucht wird die Lokalisationsgenauigkeit eines Sprachsignals in einer idealen und einer realen Umgebung. Bei der idealen Umgebung handelt es sich um einen reflexionsarmen Raum, in dem entstehende Reflexionen des Audiosignals stark gedämpft werden. Bei der realen Umgebung handelt es sich um einen gewöhnlichen Raum.

Zum Testen der Lokalisationsgenauigkeit wird ein spezielles Messprogramm auf dem ReSpeaker Core v2.0 implementiert. Dieses wird für alle folgenden Messungen verwendet. Der Signalfluss für das Messprogramm ist in Abbildung 4.13 dargestellt.

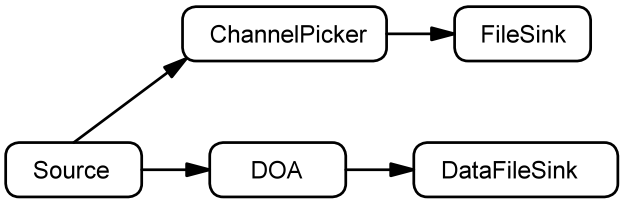


Abbildung 4.13: Signalfluss des DOA Messprogramms

Es wird einmal der von MIC2 aufgenommene Audiostream für Analysezwecke in eine Audiodatei gespeichert. Zudem werden die vom Lokalisationsalgorithmus ermittelten Winkel in eine Textdatei geschrieben. Dafür wurde das DOA Modul so weit verändert, dass es für jedes ankommende Datenpaket einen Winkel ermittelt und diesen weiterleitet. Um die ermittelten Winkel zu analysieren, werden diese in eine Datei geschrieben. Dazu wurde das Modul `DataFileSink` implementiert. Dieses schreibt alle ihm übergebenen Werte in eine Textdatei.

4.3.1 Test der Lokalisationsgenauigkeit in idealer Umgebung

Durch die Messungen soll geprüft werden, ob der Lokalisationsalgorithmus auf dem ReSpeaker Core v2.0 entsprechend der Simulation arbeitet. Um die Ergebnisse mit der Simulation vergleichen zu können, ist eine ideale Umgebung notwendig. Die Messungen werden deshalb in einem reflexionsarmen Raum durchgeführt. Darin werden Reflexionen und äußere Störeinflüsse durch den Aufbau des Raumes minimiert. Um die Messung anwendungsnah durchzuführen, wird ein Sprachsignal verwendet. Das verwendete Sprachsignal ist im Anhang unter B.1 dargestellt. Es werden zwei Messungen durchgeführt. Gemessen wird über einen Azimutwinkel von 0° bis 360° mit einem Elevationswinkel von 0° und 30°.

Versuchsaufbau und Versuchsdurchführung:

Das Sprachsignal wird anhand eines Lautsprechers ausgegeben. Dieser befindet sich in einem Abstand von 2 m und auf gleicher Höhe zum ReSpeaker Core v2.0. Der ReSpeaker Core v2.0 wird auf eine Testvorrichtung montiert, die es ermöglicht, das Gerät 360° um die eigene Achse zu drehen und um einen Winkel von 0° und 30° zu neigen. Anhand dieser Vorrichtung muss der Lautsprecher nicht verstellt werden. Die Einstellungen des Azimut- und Elevationswinkels erfolgen alle anhand der Testvorrichtung.

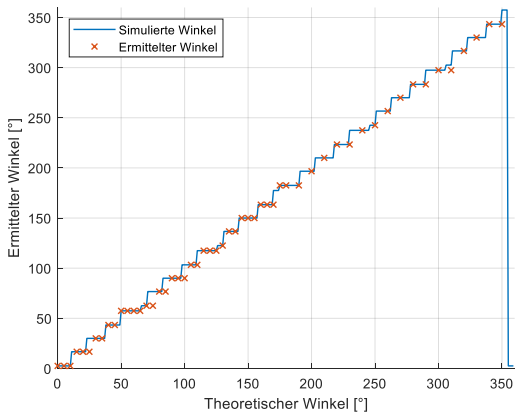
Die für die Messung vorgenommenen Einstellungen für den ReSpeaker Core v2.0 sind eine Abtastrate von 16000 Hz mit einer Blocklänge von 3200 Abtastwerten. Diese Blocklänge liefert laut Abbildung 4.10 eine geringe Fehlerrate und das System ist durch eine Winkelausgabe im 5 Hz Takt noch relativ reaktionsschnell. Der Interpolationsfaktor wird auf dem Standartwert von 1 belassen.

Mithilfe der Testvorrichtung wird der Azimutwinkel im Bereich von 0° bis 180° in 5° Schritten und im Bereich von 180° bis 360° in 10° Schritten verstellt. Nach jedem eingestellten Azimutwinkel wird das Messprogramm auf dem ReSpeaker Core v2.0 sowie die Audiodatei mit dem Sprachsignal gestartet. Die Messungen werden einmal mit einem Elevationswinkel von 0° und einmal mit 30° durchgeführt.

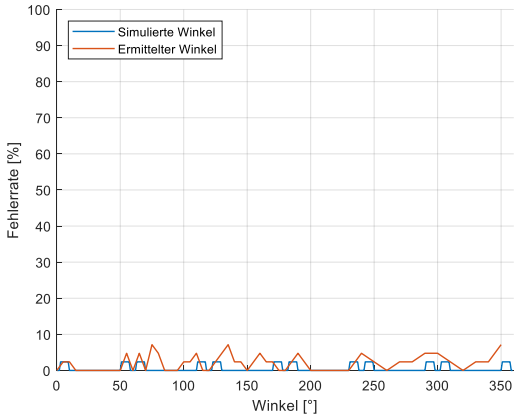
Beobachtung und Ergebnis:

Während der Messung kam es teilweise zu Fehlmessungen. Dabei lag der ermittelte Winkel weit außerhalb des Sollwinkels. Dieses Verhalten ergibt sich durch eine falsch ausgegebene Zeitdifferenz im GCC-PHAT Modul und wird durch die anschließende Fallunterscheidung verstärkt. Weiter kam es zu Sprüngen zwischen benachbarten Winkeln. Dieses Verhalten trat verstärkt in den Winkelbereichen auf, bei denen eine Abstufung eines Winkels zum anderen erfolgt.

Die Abbildung 4.14 zeigt die Messergebnisse bei einem Elevationswinkel von 0° an. In Abbildung 4.14 (a) sind die ermittelten Winkel des Tests mit den simulierten Winkeln dargestellt. In Abbildung 4.14 (b) sind die Fehmessungen, die außerhalb des Toleranzbereiches von ±15° liegen, prozentual aufgetragen.



(a) Simulierter und ermittelter Winkel

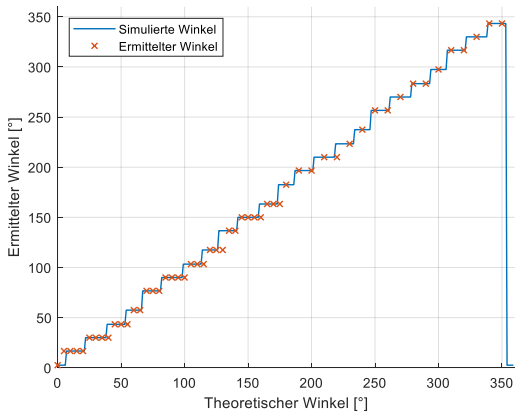


(b) Fehlerrate der simulierten und ermittelten Winkel bei einer Toleranz über ±15°

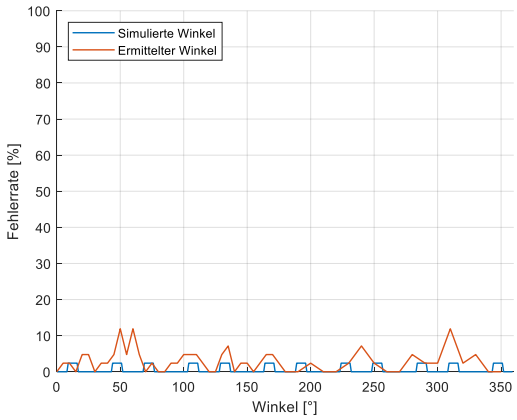
Abbildung 4.14: Messergebnisse bei idealer Umgebung mit einem Elevationswinkel von 0° und einer Abtastrate von 16000 Hz

Die im Messraum ermittelten Winkel stimmen überwiegend mit den simulierten Winkeln überein. Die Anzahl der Fehlmessungen ist im Vergleich zu den simulierten Ergebnissen nur leicht erhöht.

In der Abbildung 4.15 sind die Messergebnisse bei einem Elevationswinkel von 30° dargestellt. Die im Messraum ermittelten Winkel bei einem Elevationswinkel von 30°, stimmen auch überwiegend mit den simulierten Winkeln überein (siehe Abbildung 4.15 (a)). Es ist jedoch im Vergleich zu der Messung bei einem Elevationswinkel von 0° eine höhere Fehlerrate zu sehen.



(a) Simulierter und ermittelter Winkel



(b) Fehlerrate der simulierten und ermittelten Winkel bei einer Toleranz über ±15°

Abbildung 4.15: Messergebnisse bei idealer Umgebung mit einem Elevationswinkel von 30° und einer Abtastrate von 16000 Hz

4.3.2 Test der Lokalisationsgenauigkeit in realer Umgebung

Bei diesem Test soll der Lokalisationsalgorithmus auf seine Stabilität und Genauigkeit unter realen Bedingungen geprüft werden. Der Test findet daher in einem gewöhnlichen Raum statt, in dem Störungen in Form von Reflexionen und äußere Störgeräusche stattfinden. Als Audioquelle wird dasselbe Sprachsignal verwendet. Dieses wird ebenso über einen Lautsprecher ausgegeben.

Versuchsaufbau und Versuchsdurchführung

Der ReSpeaker Core v2.0 wird für den Test auf einem Tisch aufgestellt. Der Lautsprecher wird in einem Abstand von 600 mm zum Mikrofonarray und einer Höhe von 350 mm von der Tischkante aufgestellt. Daraus resultiert ein Elevationswinkel von 30°. Der Messaufbau ist in Abbildung 4.16 dargestellt.

Es werden zwei Messungen durchgeführt. Für beide Messungen werden eine Abtastrate von 16000 Hz und eine Blocklänge von 3200 Abtastwerten gewählt. In der ersten Messung wird der Interpolationsfaktor auf 1 belassen und in der zweiten Messung wird der Interpolationsfaktor auf 10 erhöht.

Bei der Messung mit einem 1-fachen Interpolationsfaktor wird über einen Azimutwinkel von 0° bis 360° gemessen. Im Bereich von 0° bis 180° wird alle 5° und im Bereich von 180° bis 360° alle 10° eine Messung durchgeführt. Bei der Messung mit einem 10-fachen Interpolationsfaktor wird nur im Bereich von 0° bis 180° mit einer Schrittweite von 5° gemessen.

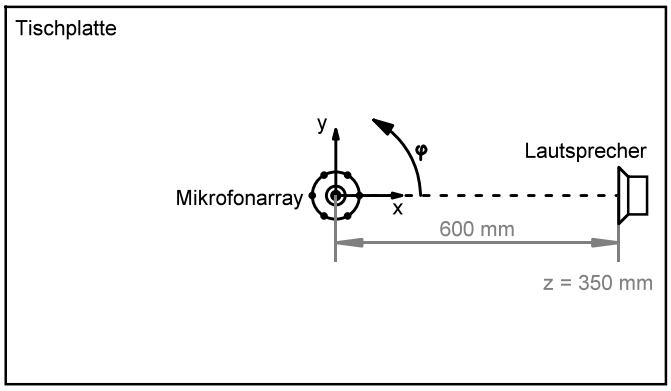
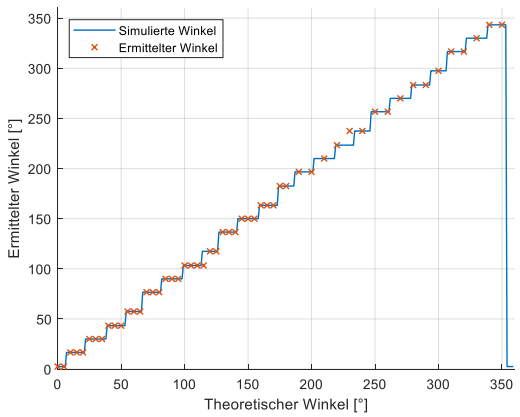


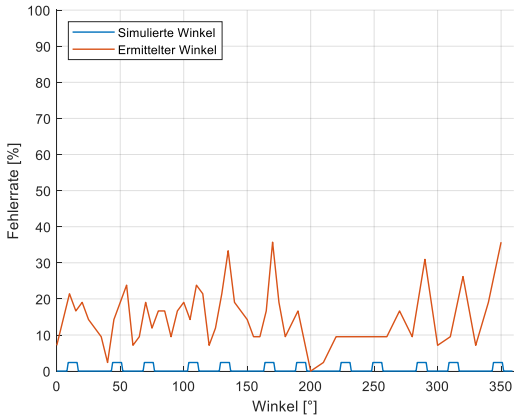
Abbildung 4.16: Messaufbau DOA Messung reale Umgebung

Beobachtung und Ergebnis:

Bei der Messung mit einem 1-fachen Interpolationsfaktor stimmen die ermittelten Winkel überwiegend mit denen aus der Simulation überein (siehe Abbildung 4.17 (a)). Die Anzahl der Fehlmessungen nahm im Vergleich zu den Fehlmessungen im Messraum deutlich zu (vergl. Abbildung 4.15 (b) mit Abbildung 4.17 (b)). Bei einem eingestellten Azimutwinkel von 30°, 90°, 150°, 210°, 270° und 330° nahm die Anzahl der Fehlmessungen ab. Bei diesen Winkeln ist immer ein Mikrofonpaar orthogonal zur Audioquelle ausgerichtet.



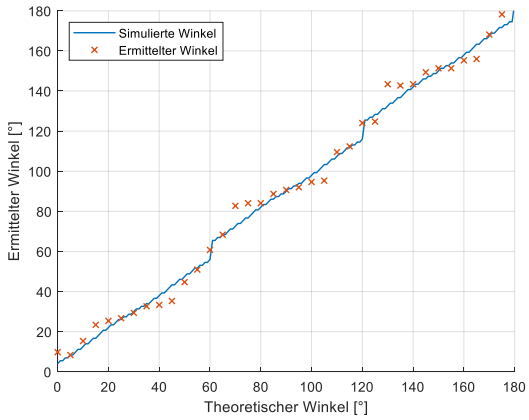
(a) Simulierter und ermittelter Winkel



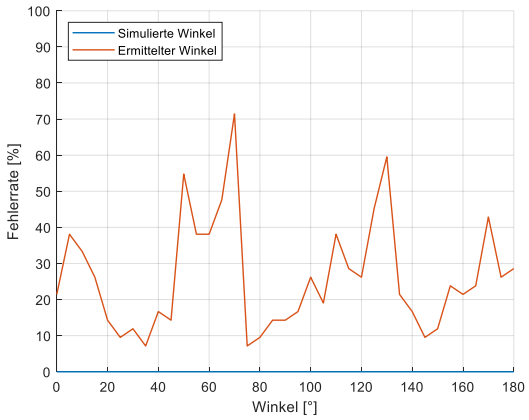
(b) Fehlerrate der simulierten und ermittelten Winkel bei einer Toleranz über ±15°

Abbildung 4.17: Messergebnisse bei realer Umgebung bei einer Abtastrate von 16000 Hz und einem Interpolationsfaktor von 1

Die Messergebnisse des Tests mit einem Interpolationsfaktor von 10 sind in Abbildung 4.18 dargestellt. Bei den ermittelten Winkeln ist eine Abweichung zu den simulierten Winkeln zu sehen. Die Anzahl der Fehlmessungen ist im Vergleich zu den vorigen Messungen deutlich erhöht. Bei den simulierten Werten sind hingegen keine Fehlmessungen außerhalb des Toleranzbereichs vorhanden. Bei realen Bedingungen führt die Interpolation zu deutlichen Schwankungen der Winkelausgabe.



(a) Simulierter und ermittelter Winkel



(b) Fehlerrate der simulierten und ermittelten Winkel bei einer Toleranz über ±15°

Abbildung 4.18: Messergebnisse bei realer Umgebung bei einer Abtastrate von 16000 Hz und einem Interpolationsfaktor von 10

4.4 Zusammenfassung und Bewertung

Die Differenzzeitmessung zwischen zwei Signalen über die GCC-PHAT Methode liefert insbesondere bei breitbandigen Signalen gute Ergebnisse. Bei Sprachsignalen kommt es zwischenzeitig zu Fehlmessungen. Diese kommen aufgrund der stellenweisen geringen Bandbreite des Sprachsignals im anliegenden Analysefenster zustande.

Der verwendete Algorithmus für die Ausgabe eines Winkels zur Audioquelle weist Schwächen auf. Durch die Einteilung der sechs Mikrofone in drei lineare Arrays kann nur der broadside Winkel ausgegeben werden. Dieser liefert mit steigendem Elevationswinkel eine hohe Abweichung zum Azimutwinkel. Der Vorteil einer zirkularen Mikrofonanordnung gegenüber einer linearen Anordnung ist die Möglichkeit den Azimut- und Elevationswinkel auszugeben. Dieser wird im Lokalisationsalgorithmus nicht wahrgenommen. Ein weiterer Schwachpunkt ist, dass es bei einer Fehlschätzung der Laufzeitdifferenz groben Abweichungen zum Sollwinkel kommt. Diese groben Abweichungen kommen aufgrund der Fallunterscheidung zustande.

Anhand der Möglichkeit zur Interpolation der generalisierten Kreuzkorrelationsfolge kann die Auflösung der Laufzeitdifferenz erhöht werden. Im Vergleich zur Erhöhung der Abtastrate liefert die Interpolation eine kürzere Berechnungszeit und eine höhere Winkelauflösung.

Die Messungen zur Lokalisation des Sprachsignals zeigten einige Schwächen auf. Es traten sowohl bei der Simulation als auch beim Testen in idealer und realer Umgebung Fehschätzungen außerhalb eines Toleranzbereiches von ±15° auf. Wo die Fehlerrate bei der Simulation unter 3 % lag, waren es bei der idealen Umgebung, je nach Elevationswinkel, stellenweise bis zu 13 %. Bei der realen Umgebung hingegen betrug die Fehlerrate maximal 35 %. Mit einem Interpolationsfaktor von 10 stiegen die Fehlschätzungen in realer Umgebung stellenweise auf 70 %, wohingegen bei der Simulation keine Fehlschätzungen entstanden.

Laut Abbildung 4.10 kann die Anzahl der Fehlschätzungen durch eine Erhöhung der Blocklänge verringert werden. Mit steigender Blocklänge wird das System jedoch träge und reagiert verzögert. Eine weitere mögliche Verbesserung ist es den Lokalisationsalgorithmus nur während der aktiven Sprache zu berechnen.

5 Analyse und Test des Beamformings

In diesem Kapitel wird der in der Voice-Engine befindliche Algorithmus zum Beamforming analysiert. Anschließend werden Simulationen auf Basis des Algorithmus durchgeführt, um seine Leistungsfähigkeit zu überprüfen. Weiter wird der Algorithmus auf dem ReSpeaker Core v2.0 ausgeführt und getestet. Zum Schluss werden die Ergebnisse zusammengefasst und bewertet.

5.1 Funktionale Analyse des implementierten Algorithmus

Bei dem hier verwendeten Beamformer handelt es sich um einen Delay and Sum (DS) Beamformer. Der Algorithmus dazu befindet sich im Voice-Engine Paket in dem Python Skript *„delay_sum.py“*. Die darin enthaltene Klasse heißt `DelaySum` und die Initialisierungsmethode sieht wie folgt aus:

```
def __init__(self, channels=8, frames_size=160, max_offset=None):
```

Die benötigten Parameter der Klasse sind `channels, frames_size und max_offset`. Über `channels` kann die Anzahl der Mikrofonkanäle übergeben werden. Der Parameter `frames_size` beschreibt die Blocklänge des Audiosignals und `max_offset` ist die maximale Signalverschiebung zwischen zwei Signalen.

Bevor weiter auf den Algorithmus eingegangen wird, wird zunächst im folgenden Abschnitt die Funktionsweise eines Delay and Sum Beamformers erläutert. Anschließend wird auf die Implementierung eingegangen.

5.1.1 Delay and Sum Beamformer

Der Delay and Sum Beamformer ist einer der ältesten und einfachsten [21] Möglichkeiten einen gerichteten Empfang durch ein Mikrofonarray zu erhalten. Der Beamformer arbeitet aufgrund der Tatsache, dass das Audiosignal mit einer Laufzeitverzögerung an den Mikrofonen ankommt. Der Delay and Sum Beamformer verzögert die anliegenden Mikrofonsignale um eine vorgegebene Zeit und summiert diese auf. Die Verzögerungen haben einen direkten Bezug zu der Zeitdauer, die das Audiosignal zum Passieren der Mikrofone benötigt. Dadurch werden Signale aus einer bestimmten Position phasengleich aufsummiert, wodurch eine Verstärkung erreicht wird. Signale aus anderen Positionen liegen nach der Verzögerung nicht phasengleich an, wodurch sie sich bestenfalls gegenseitig auslöschen. Um zu vermeiden, dass das Ausgabesignal nach der Summation N-mal größer als die Amplitude jedes erfassten Mikrofonsignals ist, wird das Ausgabesignal auf die Anzahl der Mikrofone skaliert. Durch gezielte Änderung der Verzögerungszeiten können verschiedene Richtungen angepeilt und fokussiert werden. Dieses wird als *steering* bezeichnet.

Es sind zwei Hauptgruppen von Beamformingverfahren vorhanden. Diese sind datenunabhängig (oder fest) und datenabhängig (oder adaptiv). Die datenunabhängigen Techniken legen ihre Parameter fest und behalten sie während der Verarbeitung des Eingangssignals bei. Datenabhängigen Techniken aktualisieren ihre Parameter auf Grundlage des Eingangssignals. Dadurch können Quellen mit Störgeräuschen gezielt gedämpft werden.

Bei dem Delay and Sum Beamformer handelt es sich um einen datenunabhängigen Typ [22]. Die Verzögerungen für verschiedenen Fokussierungen können während der Laufzeit berechnet oder vorher fest eingestellt werden. Die Verzögerungszeiten hängen von der Position der Mikrofone, der Schallgeschwindigkeit und der angepeilten Richtung ab.

Die Abbildung 5.1 zeigt das Funktionsprinzip des Delay and Sum Beamformers. Dargestellt sind zwei Erfassungsszenarien. Links kommt das Audiosignal aus der angepeilten Richtung und rechts außerhalb dieser. Im ersten Szenario sind die Verzögerungen des Beamformers auf die Richtung der Audioquelle ausgelegt. Die entstehenden Laufzeitverzögerungen der einzelnen Mikrofonsignale werden durch die eingestellten Verzögerungen ausgeglichen. Die einzelnen Mikrofonsignale liegen dadurch zeitgleich auf. Nach der Summation und der Skalierung liegt ein einzelner Impuls mit derselben Amplitude vor. Im zweiten Szenario werden die einzelnen Mikrofonsignale ebenso verzögert. Aufgrund der veränderten Position der Audioquelle, kommt es zu anderen Laufzeitverzögerungen. Die Mikrofonsignale stimmen nach der eingestellten Verzögerung zeitlich nicht mehr überein. In diesem Fall hat das Ausgangssignal eine geringere Amplitude.

Das Ausgabesignal $y(t)$ des DS Beamformers ist im Zeitbereich definiert durch

$$y(t) = \frac{1}{N}\sum_{m=1}^{N} \boldsymbol{x}_m(t - \boldsymbol{\tau}_m) \tag{5.1}$$

wobei N für die Anzahl der Mikrofone steht, x_m das Signal des Mikrofones mit dem Index m darstellt und τ_m die Verzögerung für das Mikrofon mit dem Index m. In diesem Fall werden alle Mikrofonsignale gleich gewichtet. Es handelt sich dabei um eine spezielle Form des DS Beamformers. In der allgemeinen Form kann jedem Mikrofonsignal eine Gewichtung zugeordnet werden. Der allgemeine DS Beamformer ist im Zeitbereich definiert durch

$$y(t) = \sum_{m=1}^{N} \boldsymbol{w}_m \boldsymbol{x}_m(t - \boldsymbol{\tau}_m) \tag{5.2}$$

wobei $\boldsymbol{w}_m$ die jeweilige Gewichtung des Mikrofonsignals darstellt [21].

Bei diskreten Signalen werden die Verzögerungen als Verschiebungen dargestellt. Der allgemeine DS Beamformer für diskrete Signale ergibt sich zu:

$$y(n) = \sum_{m=1}^{N} \boldsymbol{w}_m \boldsymbol{x}_m(n - \boldsymbol{k}_m) \tag{5.3}$$

Das $\boldsymbol{k}_m$ steht für die Verschiebungen für das Mikrofon mit dem Index m

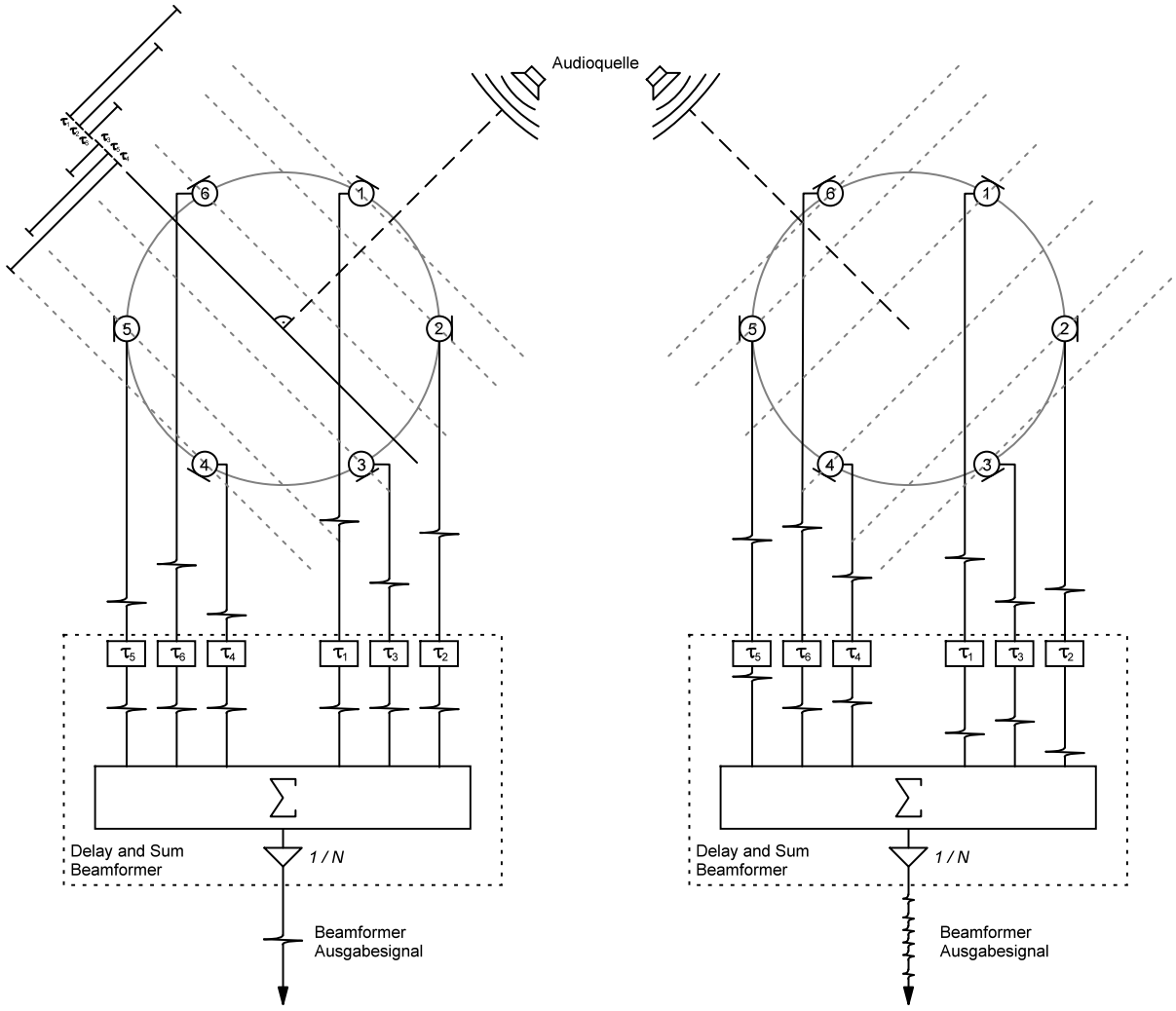


Abbildung 5.1: Darstellung der Signalverzögerungen eines Delay and Sum Beamformers. Links Audiosignal aus bevorzugter Richtung, rechts Audiosignal aus nicht bevorzugter Richtung

Das Anpeilen und Fokussieren einer Richtung mit einem Mikrofonarray erforderten die Fähigkeit die Signale der Mikrofone zu verzögern. Bei diskreten Signalen können die Verzögerungen nur ein Vielfaches der Abtastperiode darstellen. Die im Zeitbereich berechneten Verzögerungen müssen somit auf das Vielfache der Abtastperiode gerundet werden. Diese Quantisierung der Verzögerungen beeinflusst die Leistungsfähigkeit des Beamformers.

Um diesen Effekt zu verringern kann zum einen die Abtastfrequenz erhöht werden. Eine andere Möglichkeit ist es das Problem mathematisch zu lösen. Eine Möglichkeit ist es das Signal auf eine höhere Abtastfrequenz zu interpolieren, dann verschieben und anschließend wieder auf die ursprüngliche Abtastfrequenz bringen. Eine weitere Möglichkeit ist es den gesamten Vorgang im Frequenzbereich durchzuführen. Dafür werden die Signale über eine FFT in den Frequenzbereich gebracht, in diesem verschoben, summiert und anschließend mit der IFFT wieder in den diskreten Zeitbereich transformiert. Nähere Informationen zu den genannten Verfahren können unter [23] eingesehen werden.

Wie schon erwähnt hängen die Verzögerungszeiten von der Position der Mikrofone, der Schallgeschwindigkeit und der angepeilten Richtung ab. Die Verzögerungen sind alle auf einen frei wählbaren Referenzpunkt bezogen. Oft bietet sich dafür der Mittelpunkt des Mikrofonarrays an.

Die Verzögerungen können mit dem hier folgenden Ansatz für jede beliebige Arraygeometrie berechnet werden. Als Voraussetzung muss das Modell der ebenen Welle zutreffen (siehe Abschnitt 2.2). Bezogen auf das in Abschnitt 2.1 vorgestellte Kugelkoordinatensystem kann die Ausbreitungsrichtung der ebenen Welle vektoriell dargestellt werden als:

$$\boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} -\cos\theta\cos\varphi \\ -\cos\theta\sin\varphi \\ -\sin\theta \end{bmatrix}. \tag{5.4}$$

Mithilfe der Positionen der Mikrofone und dem Wellenvektor ergeben sich die Verzögerungen durch Anwendung des Skalarprodukts zu

$$\boldsymbol{\tau}_m = \frac{\boldsymbol{a}^T \boldsymbol{m}_m}{c} = -\frac{1}{c}\left(m_{m,x} \cdot \cos\theta\cos\varphi + m_{m,y} \cdot \cos\theta\sin\varphi + m_{m,z} \cdot \sin\theta\right), \tag{5.5}$$

wobei c die Schallgeschwindigkeit und $\boldsymbol{m}_m = \begin{bmatrix} m_{m,x} & m_{m,y} & m_{m,z} \end{bmatrix}$ die Position der Mikrofone in kartesischen Koordinaten darstellt [24].

Für ein zirkular gleichverteiltes Mikrofonarray kann die folgende Gleichung für die Verzögerungen verwendet werden [25]:

$$\boldsymbol{\tau}_m = -\frac{r}{c}\cos\theta\cos\left(\frac{2\pi m}{N} - \varphi\right). \tag{5.6}$$

Das r ist der Radius des Arrays, c ist die Schallgeschwindigkeit, m ist der Index des Mikrofons und N ist die Anzahl der Mikrofone im Array.

5.1.2 Analyse des Delay and Sum Algorithmus

Der DS Beamformer Algorithmus läuft in einem separaten Thread ab. Darin wird der Algorithmus so lange durchgeführt, bis er manuell gestoppt wird. Grundlegend ist der Ablauf so, dass zuerst die Verzögerungen berechnet werden und anschließend die Summation mit der Skalierung stattfindet. Zur Berechnung der Verzögerungen wird die GCC-PHAT Methode (siehe Abschnitt 4.1.1) verwendet. Dadurch richtet sich der Beamformer automatisch auf die Audioquelle aus. Eine Manuelle Ausrichtung des Beamformers ist im Algorithmus nicht vorgesehen.

Um die Berechnungen durchzuführen, werden zuerst die Mikrofondaten aus dem binären Audiostream extrahiert und in das Array `data` als Integer Datentyp abgelegt.

```
data = self.queue.get()
data = np.fromstring(data, dtype='int16')
```

Die Berechnung der Verzögerungen wird über eine Schleife durchgeführt. Für jedes Mikrofon ist eine Verzögerung notwendig. Diese benötigen einen gemeinsamen Referenzpunkt, damit der DS Beamformer ordnungsmäßig funktioniert. Als Referenzpunkt wird hier MIC1 gewählt. Die Verzögerungen werden über die GCC-PHAT Methode berechnet. Dafür wird ausgehend vom ersten Mikrofon die Signalverzögerung zu jedem weiteren Mikrofon berechnet. Da es sich um diskrete Signale handelt, wird anstelle einer Verzögerung eine ganzzahlige Verschiebung berechnet. Die berechneten Verschiebungen werden in das Array `offsets` hinterlegt.

```
offsets = [0] * self.channels
for i in range(1, self.channels):
     offsets[i], _ = gcc_phat(data[i::self.channels], data[0::self.channels],
                              max_tau=self.max_offset, interp=1)
```

Nachdem die einzelnen Verschiebungen berechnet sind, werden die Mikrofonsignale verschoben, aufsummiert und skaliert. Da es sich um begrenzte Signalausschnitte handelt, werden bei einer Verschiebung einige Abtastwerte außerhalb der Signalgrenzen geschoben, was zu einem Datenverlust und einem nicht kontinuierlichen Ausgabesignal führt. Um dies zu vermeiden, wird der Signalausschnitt in zwei Segmente geteilt. Das Ausgabesignal wird aus dem zweiten Segment des vorherigen und dem ersten Segment des aktuellen Signalausschnitts zusammengesetzt. Die Verschiebung wird durch eine Aufteilung des Signalausschnitts in zwei unterschiedlich lange Segmente bewerkstelligt. Der Vorgang wird durch Abbildung 5.2 verdeutlicht. Diese stellt zwei Mikrofonsignale dar, wovon das erste Signal nicht und das zweite Signal um drei Elemente verschoben wird. Bei gleichbleibender Verschiebung entsteht ein kontinuierliches Ausgabesignal.

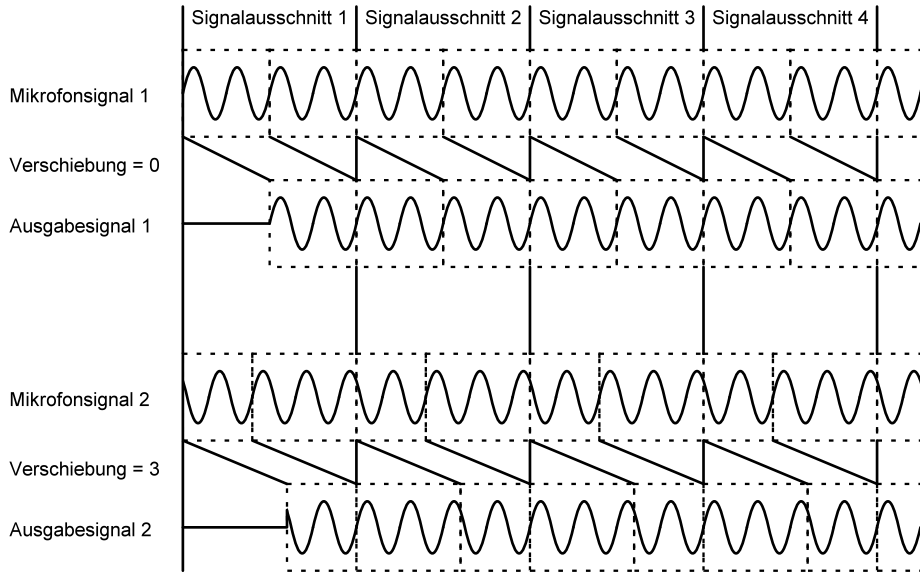


Abbildung 5.2: Darstellung der Verschiebung von Mikrofonsignalen für ein kontinuierliches Ausgabesignal

Die Mikrofonsignale werden nacheinander über eine Schleife verschoben und aufsummiert. Zum Schluss wird das Summensignal skaliert und über die `put` Methode zum nächsten Modul weitergereicht.

```
self.sum.fill(0)
half = int(self.frames_size / 2)
for i in range(self.channels):
     offset = int(offsets[i])
     self.sum[:half - offset] += (self.last[i::self.channels])[half + offset:]
     self.sum[half - offset:] += (data[i::self.channels])[:half + offset]

self.last = data
out = np.array(self.sum / self.channels, dtype='int16')
super(DelaySum, self).put(out.tostring())
```

5.2 Überprüfung des Algorithmus per Simulation

In diesem Abschnitt wird der Beamforming Algorithmus anhand von Simulationen auf seine Leistungsfähigkeit überprüft. Eine Möglichkeit die Leistungsfähigkeit eines Beamformers darzustellen, ist es über das sogenannte Beampattern. Dies ist im folgenden Abschnitt dargestellt. Zusätzlich wird noch eine Laufzeituntersuchung des Beamforming Algorithmus auf dem ReSpeaker Core v2.0 durchgeführt.

5.2.1 Erstellung des Beampattern

Eine wichtige Aussage über die Leistungsfähigkeit des Beamformers liefert das Beampattern. Dieses stellt das Eingangs-/ Ausgangsverhalten bei einer oder mehreren Frequenzen über jeden Winkel dar. Dadurch wird die Richtcharakteristik des Beamformers dargestellt.

Zur Aufnahme des Beampattern wird der Delay and Sum Beamformer auf einen Azimut- und Elevationswinkel von 0° ausgerichtet. Simuliert wird über einen Azimutwinkel von -180° bis 180° mit Frequenzen bis 7000 Hz. Die Abbildung 5.3 stellt das Beampattern für den Delay and Sum Algorithmus bei einem Elevationswinkel von 0° und einer Abtastfrequenz von 16000 Hz dar.

An dem Beampattern ist zu erkennen, dass unterhalb einer Frequenz von 1000 Hz keine Dämpfung stattfindet. Dies ist der großen Wellenlänge des Signals im Verhältnis zum Arraydurchmesser geschuldet. Die Verschiebung der Wellensignale zwischen den Mikrofronen ist aufgrund des geringen Arraydurchmessers nicht ausreichend, als dass sie sich additiv auslöschen.

Auf dem Beampattern ist nur ein schmaler Streifen sichtbar, in dem die Dämpfung auf über 40 dB steigt. Darüber hinaus beträgt die Dämpfung meist um die 6 dB. Aufgrund der breiten Hauptkeule ist die Richtwirkung gering.

In einem Frequenzbereich zwischen 4000 Hz bis 6000 Hz ist bei einem Winkel um die 125° so gut wie keine Dämpfung sichtbar. Dies kommt durch den räumlichen Alias-Effekt zustande. Die Wellenlänge des Signals entspricht dabei dem Mikrofonabstand. Das resultierende Mikrofonsignal stellt gleichphasige Wellen dar. Die Signale überlagern sich konstruktiv, wodurch ein ungedämpftes Ausgabesignal resultiert.

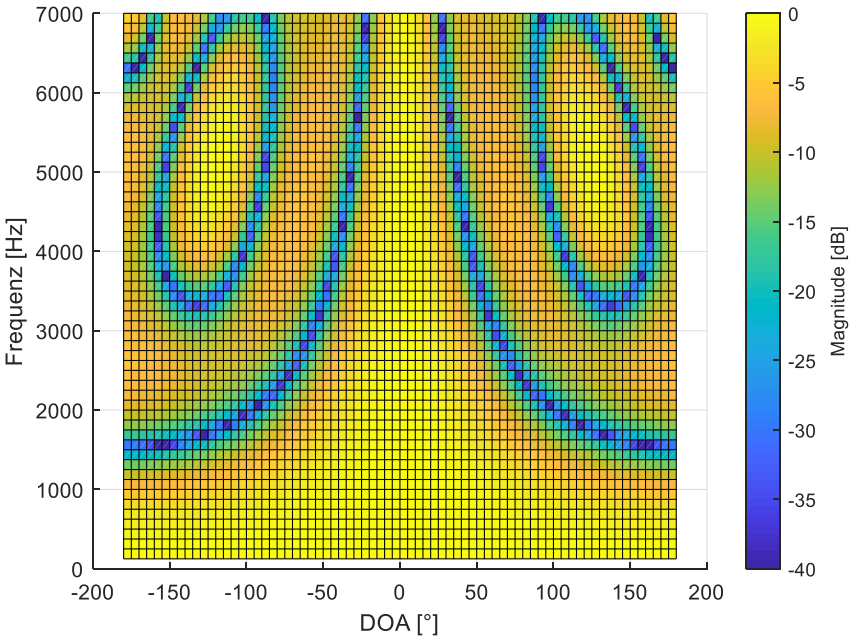


Abbildung 5.3: Beampattern des DS Beamformers für eine Ausrichtung von 0° Azimut- und Elevationswinkel bei einer Abtastfrequenz von 16000 Hz

Zusätzlich wird untersucht, ob eine Erhöhung der Abtastfrequenz eine Verbesserung des Beampattern mit sich bringt. Das Beampattern mit denselben Ausrichtung aber einer Abtastfrequenz von 48000 Hz ist in Abbildung 5.4 dargestellt.

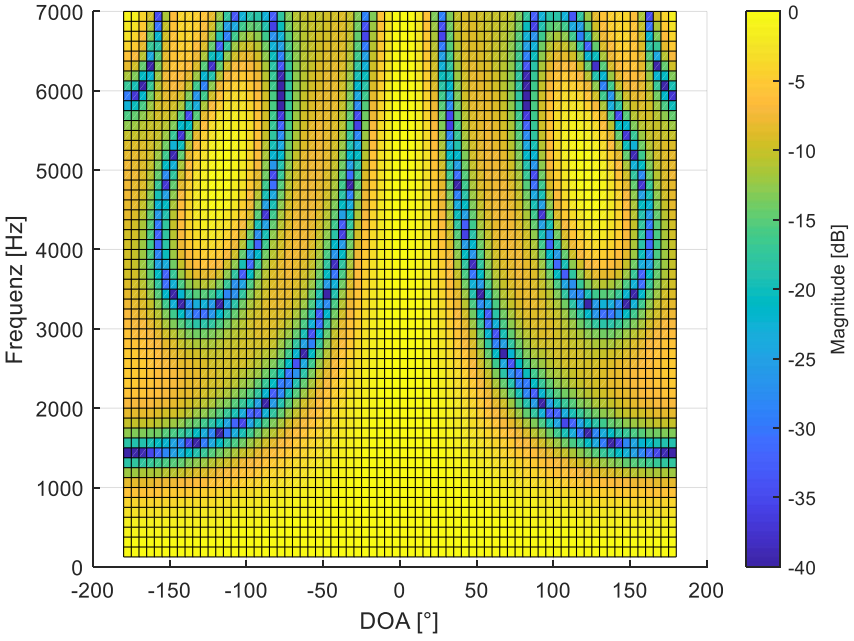


Abbildung 5.4: Beampattern des DS Beamformers für eine Ausrichtung von 0° Azimut- und Elevationswinkel bei einer Abtastfrequenz von 48000 Hz

Im Vergleich zu dem Beampattern mit einer Abtastrate von 16000 Hz sind keine großen Veränderungen und Verbesserungen sichtbar. Eine Abtastfrequenz von 16000 Hz ist somit ausreichend.

5.2.2 Laufzeituntersuchung des Beamforming Algorithmus

Die Untersuchung der Berechnungszeit für den Delay and Sum Beamforming Algorithmus wird über eine Differenzzeitmessung durchgeführt. Dafür wird ein zufallsgeneriertes Signal in den DS Algorithmus auf dem ReSpeaker Core v2.0 eingespeist und die Zeit zwischen Ein- und Ausgabe gemessen. Es werden Signale mit verschiedenen Blocklängen eingespeist. Für jeden Block werden 1000 Messungen durchgeführt und anschließend die mittlere Berechnungszeit ermittelt.

Die Berechnungszeit wird zum einen für eine automatische Ausrichtung und zum anderen für eine manuelle Ausrichtung gemessen. Abbildung 5.5 zeigt die Berechnungszeiten beider Messungen über die Blocklänge auf.

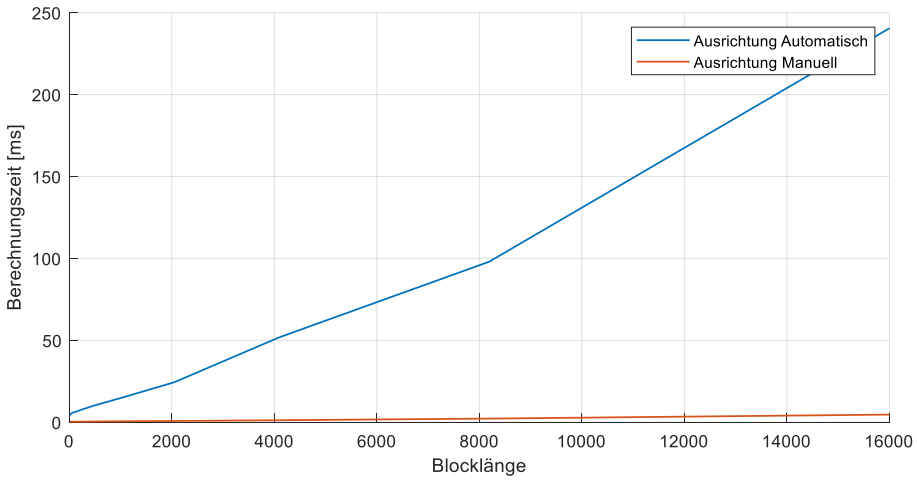


Abbildung 5.5: Berechnungszeit des BF-Algorithmus pro Blocklänge

Aufgrund des 6-Mal durchlaufenden GCC-PHAT Algorithmus zur Berechnung der Verzögerungen bei der automatischen Ausrichtung, ist die Berechnungszeit deutlich höher als bei der manuell eingestellten Ausrichtung. Auch im Vergleich zu dem Lokalisationsalgorithmus aus Kapitel 4, ist die Berechnungszeit doppelt so lang.

Wie auch bei dem Lokalisationsalgorithmus wird einmal geprüft, ab welcher Blocklänge die Berechnungszeit kürzer als die Zeitdauer ist, die die Blocklänge über die Abtastzeit darstellt. Dafür wird die Differenz zwischen der Zeitdauer und Berechnungszeit gebildet und über die Blocklänge dargestellt. Bei einer positiven Differenz ist die Berechnungszeit kürzer als die Dauer, die die Blocklänge in Verbindung mit der Abtastrate darstellt. Ist die Differenz negativ, so benötigt die Berechnung eine längere Zeit als der Block darstellt. In Abbildung 5.6 ist die Zeitdifferenz über die Blocklänge für eine Abtastzeit von 16000 Hz und 48000 Hz für den Algorithmus mit der automatischen Ausrichtung dargestellt.

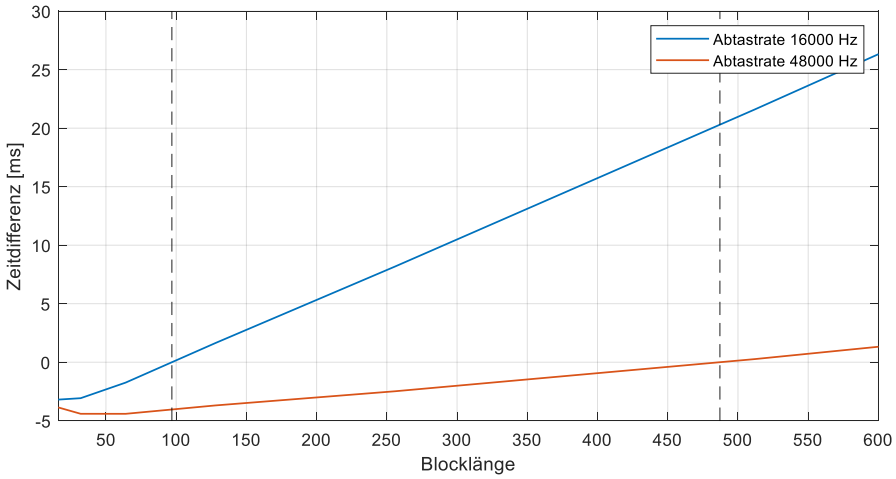


Abbildung 5.6: Zeitdifferenz zwischen Block- und Berechnungszeit in Bezug zur Blocklänge

Laut Abbildung 5.6 ist zu erkennen, dass bei einer Abtastzeit von 16000 Hz und einer Blocklänge von 95 Abtastwerten die Berechnungszeit kürzer ist. Bei einer Abtastrate von 48000 Hz ist die Berechnungszeit erst nach einer Blocklänge von 485 Abtastwerten kürzer.

5.3 Test des Algorithmus auf der Hardware

In der folgenden Messung wird die Leistungsfähigkeit des Beamforming Algorithmus in Verbindung mit dem ReSpeaker Core v2.0 getestet. Dafür wird ein Beampattern aufgenommen. Um die Ergebnisse anschließend mit den Simulierten vergleichen zu können, erfolgt die Messung in einem reflexionsarmen Raum.

Für die Aufnahme des Beampattern wird ein Messprogramm auf den ReSpeaker Core v2.0 implementiert. Der Signalfluss des Messprogramms ist in Abbildung 5.7 dargestellt.

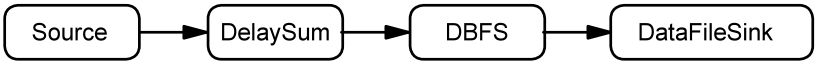


Abbildung 5.7: Signalfluss des Beamforming Messprogramms

Um die Dämpfungen für das Beampattern zu berechnen, wird das quadratische Mittel (RMS) vom Ausgabesignal des Beamformers berechnet und in eine Datei geschrieben. Um nun die Dämpfungen für jeden Winkel zu bestimmen, werden die gemessenen RMS-Werte auf den RMS-Wert normiert, welcher im Fokus des Beamformers bei der eingestellten Frequenz gemessen wurde.

Die Berechnung des quadratischen Mittels findet im DBFS Modul statt. Das dBFS steht für Dezibel Full Scale und stellt die Vollaussteuerung vom ADC in Dezibel dar. Darin wird das quadratische Mittel in Dezibel berechnet und auf den Maximalwert vom ADC normiert. Ein dBFS Wert von 0 entspricht einer Vollaussteuerung.

Um den Beamformer auf eine bestimmte Richtung zu fokussieren, wird die Klasse des Delay and Sum Beamformers um die Methode `set_beam(az, el)` erweitert. Die Methode berechnet nach Gleichung (5.6) die Verzögerungen für den Beamformer.

Versuchsaufbau und Versuchsdurchführung:

Der ReSpeaker Core v2.0 wird auf eine Testvorrichtung montiert, die es ermöglicht das Gerät 360° um die eigene Achse zu drehen und um einen Winkel von 0° und 30° zu neigen. Ein Lautsprecher wird auf gleicher Höhe in einem Abstand von 2 m aufgestellt. Die Neigung wird auf einen Winkel von 30° eingestellt, da das Beampattern für einen Elevationswinkel von 30° aufgenommen wird. Der Beamformer wird auf einen Azimutwinkel von 0° und einem Elevationswinkel von 30° eingestellt. Die Abtastfrequenz beträgt 16000 Hz und die Blocklänge 1600 Abtastwerte.

Da die Messungen viel Zeit in Anspruch nehmen, wird der Azimutwinkel in einer Schrittweite von 10° und die Frequenzen in einem Bereich von 500 Hz bis 4000 Hz in 500 Hz Schritten und bis 7000 Hz in 1000 Hz Schritten verstellt.

Ergebnis:

Das aufgenommene Beampattern mit den normierten Messwerten ist in Abbildung 5.8 dargestellt.

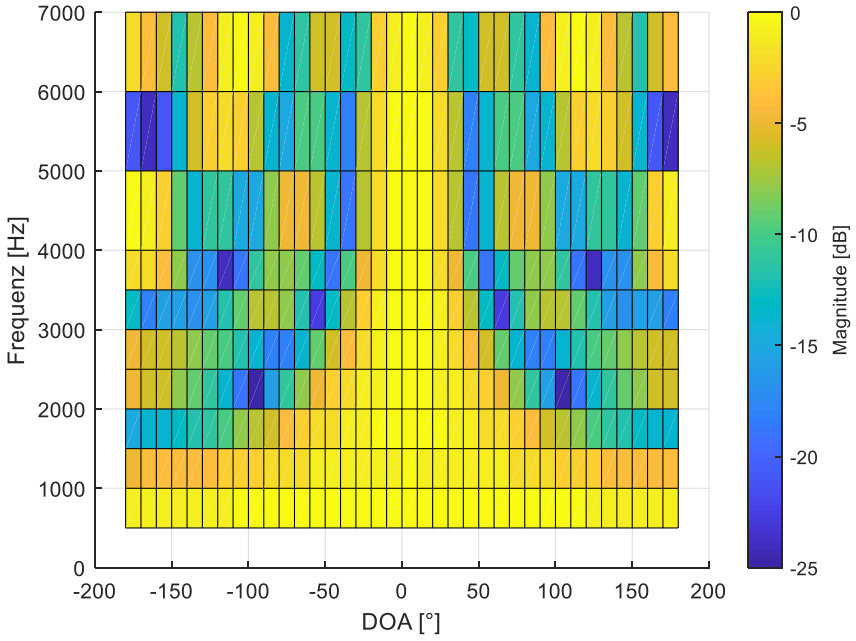
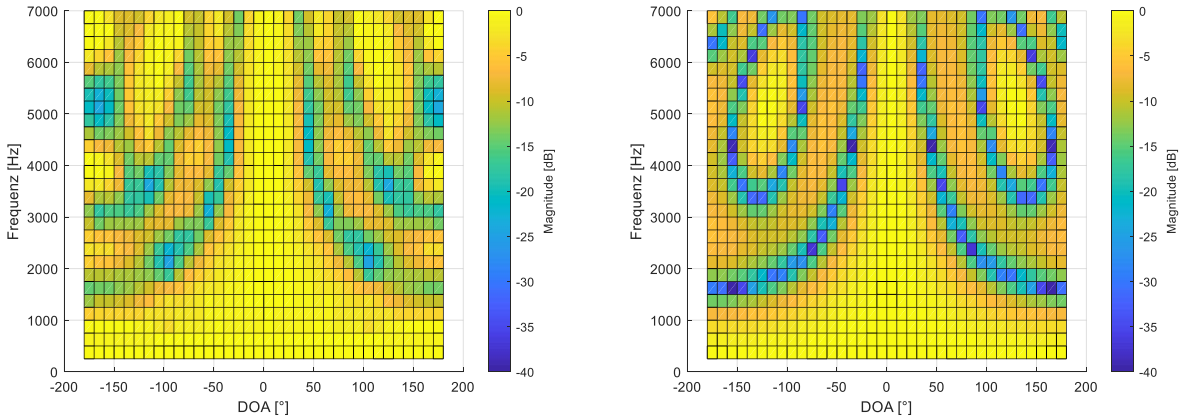


Abbildung 5.8: Gemessenes Beampattern des DS Beamformers für eine Ausrichtung von 0° Azimut- und 30° Elevationswinkel

Es ist eine gewisse Ähnlichkeit zu dem simulierten Beampattern erkennbar. In Abbildung 5.9 ist das gemessene Beampattern interpoliert und im direkten Vergleich zu dem simulierten Beampattern dargestellt. Beim Vergleich beider Beampattern miteinander, sind bei dem gemessenen (Abbildung 5.9 (a)) weniger stark ausgeprägte Dämpfungsspitzen erkennbar. Die Werte außerhalb der Spitzen stimmen jedoch größtenteils überein.



(a) Gemessenes Beampattern

(b) Simuliertes Beampattern

Abbildung 5.9: Gemessenes und Simuliertes Beampattern des DS Beamformers für eine Ausrichtung von 0° Azimut- und 30° Elevationswinkel

5.4 Zusammenfassung und Bewertung

Der Delay and Sum Beamformer stellt eine simple Variante des Beamformings dar. Die Implementierung ist schlicht gehalten und erfolgt nur im Zeitbereich. Durch die recht hohe Abtastrate von 16000 Hz ergibt sich für das Beampattern keine Verschlechterung im Vergleich zu höheren Abtastraten.

Das simulierte und in idealer Umgebung aufgenommene Beampattern ähneln sich in ihrer Form. Einen Unterschied stellt die maximale Dämpfung dar. Diese liegt bei dem simulierten Beampattern bei 40 dB wohingegen beim aufgenommenen Beampattern die maximale Dämpfung bei 25 dB liegt.

Aufgrund der geringen Ausdehnung des Arrays können Frequenzen unterhalb von 1000 Hz nicht gedämpft werden. Durch die geringe Ausdehnung ist auch nur eine geringe Richtwirkung gegeben.

6 Analyse und Test der Rauschunterdrückung

In diesem Kapitel wird der implementierte Algorithmus der Rauschunterdrückung untersucht und getestet. Im ersten Abschnitt wird auf den Algorithmus eingegangen und dessen Funktionsweise erläutert. In den beiden darauffolgenden Abschnitten wird die Leistungsfähigkeit des Algorithmus anhand von Simulationen und in Verbindung mit dem ReSpeaker Core v2.0 getestet. Zum Schluss werden die Ergebnisse zusammengefasst und bewertet.

6.1 Funktionale Analyse des implementierten Algorithmus

Die Rauschunterdrückung ist im Gegensatz zu den vorherigen Modulen in C++ implementiert. Um die Funktionen der Rauschunterdrückung aus der Python Umgebung nutzen zu können, wird ein Wrapper (hier SWIG) verwendet.

Das Modul zur Rauschunterdrückung befindet sich im Voice-Engine Paket unter den Namen *„NS.py“*. In diesem Skript wird ein zusätzliches Modul Namens *„webrtc_audio_processing.py“* importiert, welches als Schnittstelle für die in C++ programmierten Funktionen dient. Die im NS-Modul enthaltene Klasse heißt `NS` und die Initialisierungsmethode sieht wie folgt aus:

```
def __init__(self, rate=16000, channels=1):
```

Übergeben werden der Klasse einmal die Abtastrate (`rate`) und die Anzahl der Kanäle, auf die die Rauschunterdrückung angewendet werden soll (`channels`).

In den nachfolgenden Abschnitten wird zuerst der allgemeine Aufbau eines Rauschunterdrückungsverfahren dargestellt und anschließend der im Modul verwendete Aufbau.

6.1.1 Allgemeiner Aufbau von Rauschunterdrückungsverfahren

Bei der Rauschunterdrückung handelt es sich um ein einkanaliges Verfahren. Zur Beschreibung des Rauschunterdrückungsverfahren wird für das Rauschen ein vereinfachtes additives Modell angewendet. Die verschiedenen Rauschquellen werden zusammengefasst und es wird angenommen, dass sich das Rauschen additiv zum Originalsignal überlagert. Mathematisch kann das wie folgt dargestellt werden:

$$x(n) = s(n) + r(n)\,. \tag{6.1}$$

Dabei ist n der Zeitindex und $x(n)$ beschreibt das verrauschte Mikrofonsignal. Dieses Signal stellt sich aus dem diskreten Originalsignal $s(n)$ und dem diskreten Rauschsignal $r(n)$ zusammen.

Es sind verschiedene Verfahren zur Rauschunterdrückung vorhanden. Einige Verfahren arbeiten im Zeitbereich, während andere Verfahren Transformationen verwenden und die Rauschunterdrückung im jeweiligen transformierten Bereich durchführen. Die meisten Verfahren zur

Rauschunterdrückung beruhen auf der Kurzzeit-Fouriertransformation (engl. Short-Time Fourier Transform) (STFT) des Sprachsignals. Dazu wird die Kurzzeit-Stationarität des Sprachsignals ausgenutzt [26].

Rauschunterdrückungsverfahren, die auf der Kurzzeit-Fouriertransformation basieren, sind grundsätzlich identisch aufgebaut. In Abbildung 6.1 ist der Aufbau dargestellt.

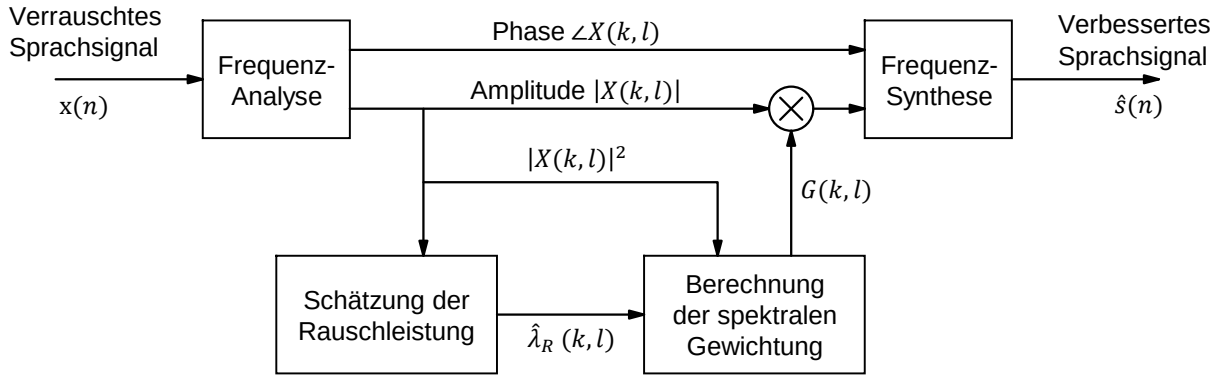


Abbildung 6.1: Allgemeiner Rauschunterdrückungsalgorithmus [26]

Bei dem STFT Verfahren wird das Zeitsignal $x(n)$ mit Hilfe einer Fensterfunktion $h(n)$ und der schnellen Fouriertransformation (FFT) in sich überlappende Signalblöke aufgeteilt und in den Frequenzbereich transformiert. Nach Gleichung (6.1) ergibt sich für das Fourier-Transformierte verrauschte Signal [26]:

$$\begin{aligned} X(k,l) &= \sum_{i=0}^{N-1} h(i)x\left(\frac{lN}{2}+i\right)e^{-\frac{2\pi ik}{N}} \\ X(k,l) &= \sum_{i=0}^{N-1} h(i)\left(s\left(\frac{lN}{2}+i\right)+r\left(\frac{lN}{2}+i\right)\right)e^{-\frac{2\pi ik}{N}} \\ X(k,l) &= S(k,l)+R(k,l) \end{aligned} \tag{6.2}$$

Der Indizes k steht für den Frequenzindex und l steht für den Blockindex. N ist die Anzahl der Abtastwerte pro Block. Um den Effekt der blockweisen Verarbeitung zu verringern, ist die Überlappung der benachbarten Signalblöcken in Gleichung (6.2) auf 50 % gesetzt [26].

Zur weiteren Verarbeitung wird die spektrale Kurzzeitgröße $X(k,l)$ in ihren Betrag $|X(k,l)|$ und Phase $\angle X(k,l)$ aufgeteilt. Die Phase wird während des gesamten Vorgangs nicht modifiziert und an das Synthesemodul weitergeleitet. In dem Modul zur Rauschleistungsschätzung wird aus dem Betragsquadrat der verrauschten spektralen Kurzzeitgröße $|X(k,l)|^2$ eine Schätzung der Rauschleistung $\hat{\lambda}_R(k,l) \approx E\{|R(k,l)|^2\}$ vorgenommen. Aus der Rauschleistung $\hat{\lambda}_R(k,l)$, der Schätzung der Leistung des Sprachanteils $\hat{\lambda}_S(k,l) \approx E\{|S(k,l)|^2\}$ sowie dem Betragsquadrat des Kurzzeitspektrum $|X(k,l)|^2$ wird die Gewichtungsfunktion $G(k,l,\hat{\lambda}_S(k,l),\hat{\lambda}_R(k,l),|X(k,l)|^2)$ bestimmt [26]. Der Übersichtlichkeit haltbar werden die Abhängigkeiten der Leistungen weggelassen $G(k,l,\hat{\lambda}_S(k,l),\hat{\lambda}_R(k,l),|X(k,l)|^2) = G(k,l)$. Anhand der Gewichtungsfunktion wird eine Schätzung

des Sprachanteils vorgenommen. Das geschätzte Betragsspektrum des Sprachanteils ergibt sich durch [26]:

$$\left|\hat{S}(k,l)\right| = G(k,l)|X(k,l)|\,. \tag{6.3}$$

Weiter wird das Betragsspektrum des Sprachanteils mit der Phase zusammengesetzt:

$$\hat{S}(k,l) = G(k,l)|X(k,l)|e^{\angle X(k,l)}\,. \tag{6.4}$$

Zum Schluss wird im Synthese Modul eine inverse Fouriertransformation durchgeführt und die Überlappung durch ein *Overlap-and-Add* Verfahren rückgängig gemacht, um das Zeitsignal wieder zu rekonstruieren [26].

Zur Berechnung der Gewichtungsfunktion sind die Größen $\hat{\lambda}_S(k,l)$, $\hat{\lambda}_R(k,l)$ und $|X(k,l)|^2$ relevant. Es genügen jedoch auch auf die Rauschleistung normierte Größen zur Berechnung der Gewichtungsfunktion. Diese sind nach [27] das a posteriori Signal-Rausch-Verhältnis (engl. Signal-to-Noise Ratio) (SNR) $\gamma(k,l)$ und das a priori SNR $\xi(k,l)$. Definiert sind diese als

$$\gamma(k,l) = \frac{|X(k,l)|^2}{\hat{\lambda}_R(k,l)} \tag{6.5}$$

und

$$\xi(k,l) = \frac{\hat{\lambda}_S(k,l)}{\hat{\lambda}_R(k,l)}\,. \tag{6.6}$$

6.1.2 Aufbau der Rauschunterdrückung im NS-Modul

Die in dem NS-Modul implementierte Rauschunterdrückung basiert auf den in Abbildung 6.1 dargestellten Ablauf. Die Abbildung 6.2 zeigt den prinzipiellen Ablauf der implementierten Rauschunterdrückung.

Wie auch im vorherigen Abschnitt beschrieben, wird hier die Kurzzeit-Fouriertransformation verwendet. Zur Verarbeitung wird ein Zeitsignal mit einer Dauer von 10 ms verwendet. Bei einer Abtastfrequenz von 16000 Hz ergeben sich 160 Abtastwerte. Die Blocklänge dafür wird auf 256 gesetzt. Die restlichen 96 Werte des Blocks werden mit den vorherigen Abtastwerten aufgefüllt, was zu einer Überlappung von 37,5 % führt. Anschließend findet eine Fensterung und FFT statt.

Die Rauschunterdrückung ist in zwei Teilen aufgebaut. Im ersten Teil wird die Rauschleistung geschätzt und im zweiten Teil wird die Gewichtungsfunktion bestimmt und daraufhin das Sprachsignal ermittelt. Für die Schätzung der Rauschleistung als auch für die Berechnung der Gewichtungsfunktion sind verschiedene Verfahren mit unterschiedlichen Eigenschaften vorhanden.

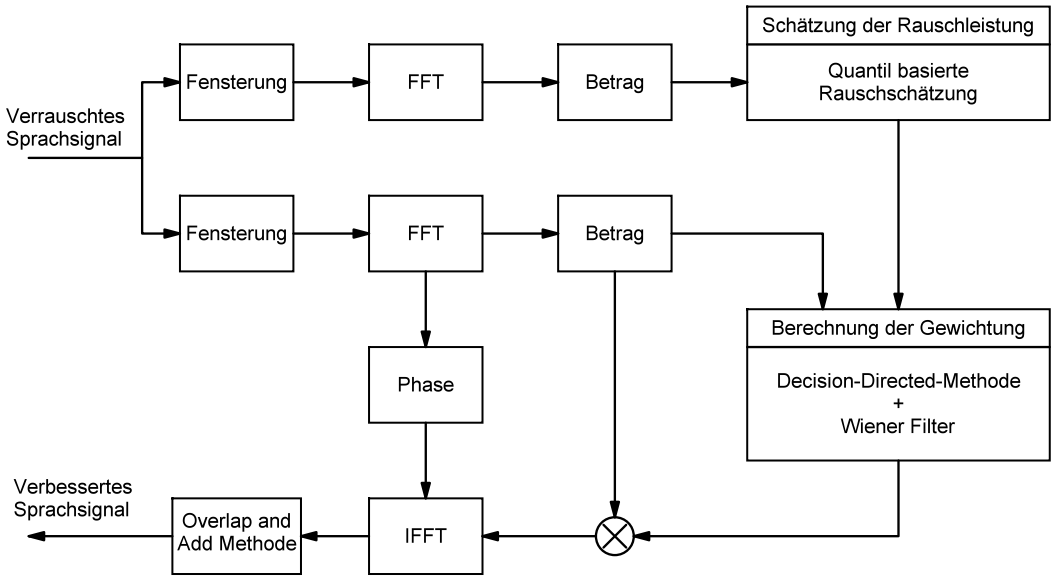


Abbildung 6.2: Ablauf des NS-Algorithmus

Traditionelle Verfahren zur Schätzung des Hintergrundrauschens beruhen auf einer Sprachpausenerkennung. Dabei wird die Energie des Signalausschnitts ermittelt und bei einer Unterschreitung eines Schwellwertes wird der Signalausschnitt als eine Sprachpause gewertet. Dieser Signalausschnitt wird dann als Rauschen interpretiert und fließt in die Schätzung des Rauschsignals ein [28]. Ein Nachteil des Verfahrens ist, dass die Schätzung des Rauschsignals nur in den Sprachpausen möglich ist und bei geringen Signal-Rausch-Verhältnissen die Resultate schlecht ausfallen. Eine weitere Variante, die auch hier verwendet wird, ist die Quantil basierte Rauschschätzung. Dieses Verfahren beruht auf der Tatsache, dass das Sprachsignal nicht in allen Frequenzbändern permanent vertreten ist. Das Rauschsignal hingegen liegt über eine signifikante Dauer in jedem Frequenzband mit einem bestimmten Energiepegel an [28]. Diese Tatsache kann genutzt werden, um die Rauschleistung aus dem verrauschten Sprachsignal abzuschätzen. Dafür wird zu jedem Frequenzband des verrauschten Sprachsignals das q-Quantil über mehrere Blöcke bestimmt. Dieses fließt dann in die Schätzung des Rauschleistungsspektrum ein. Der Vorteil daran ist, dass keine Sprachpausenerkennung notwendig ist und die Schätzung der Rauschleistung auch während der Sprachabschnitten stattfindet.

Zur Berechnung der Gewichtungsfunktion wird hier ein Wiener Filter verwendet. Das ist mit der Annahme eines additiven Rauschens (vergl. Gleichung (6.1)) sowie unkorrelierten und stationären Sprach- und Rauschsignals, gegeben durch [29]

$$G(k,l) = \frac{\hat{\lambda}_S(k,l)}{\hat{\lambda}_S(k,l) + \hat{\lambda}_R(k,l)}. \tag{6.7}$$

Verallgemeinert kann der Wiener Filter mit den einstellbaren Parametern α und β durch

$$G(k,l) = \left(\frac{\hat{\lambda}_S(k,l)}{\hat{\lambda}_S(k,l) + \alpha \cdot \hat{\lambda}_R(k,l)}\right)^{\beta} \tag{6.8}$$

beschrieben werden [29]. Mit Variation der beiden Parameter α und β können unterschiedliche Filtereigenschaften erzielt werden. Wobei α den Rauschunterdrückungsfaktor darstellt [30].

Anstelle der Leistungsspektren kann der Wiener Filter auch mit dem a priori SNR beschrieben werden als

$$G(k,l) = \left(\frac{\xi(k,l)}{\xi(k,l)+\alpha}\right)^{\beta}. \tag{6.9}$$

In der hier implementierten Rauschunterdrückung wird der Wiener Filter in Form der Gleichung (6.9) verwendet. Darin ist der Parameter β auf 1 belassen und α variiert je nach eingestellter Rauschunterdrückungsstufe.

Zur Berechnung der Gewichtung nach dem Wiener Filter wird das a priori SNR $\xi(k,l)$ benötigt. Die Schätzung des a priori Signal-Rausch-Verhältnisses wird über die *Decision-Directed-Methode* bewerkstelligt. Die Gleichung zur Schätzung des a priori SNR ist gegeben durch [31]

$$\hat{\xi}(k,l) = \alpha G(k,l-1)^2\gamma(k,l-1) + (1-\alpha)P[\gamma(k,l)-1] \tag{6.10}$$

mit

$$P[x] = \begin{cases} x & wenn\ x \geq 0 \\ 0 & sonst. \end{cases} \tag{6.11}$$

Das a priori SNR wird mit Hilfe der Gewichtungsfunktion und dem a posteriori des vorherigen Zeitblocks geschätzt $G(k,l-1)^2\gamma(k,l-1)$. Das $P[\,.\,]$ ist eine limitierende Funktion und soll verhindern, dass die Schätzung $\gamma(k,l)-1$ negativ wird. Für den Parameter α können Werte zwischen 0 und 1 gewählt werden. In Verbindung mit dem Wiener Filter führen Werte für ein α nahe 1 zu einer verzerrten Sprachausgabe, wohingegen geringere Werte die Verzerrung reduzieren, es aber zu sogenannten *musical noise* kommt. Es hat sich gezeigt, dass die Verwendung von $\alpha = 0{,}98$ gute Eigenschaften liefert [31]. Dieser Wert wird auch im Algorithmus verwendet.

Nach der Berechnung der Gewichtung wird anhand der Gleichung (6.4) das Spektrum des Sprachanteils geschätzt und mit Hilfe der IFFT in den Zeitbereich transformiert. Zum Schluss wird mit dem Overlap-and-Add Verfahren die Überlappung rückgängig gemacht und das Zeitsignal rekonstruiert.

6.2 Überprüfung des Algorithmus per Simulation

In diesem Abschnitt wird die Leistungsfähigkeit der Rauschunterdrückung geprüft. Diese wird anhand des Dämpfungsgrades des Störgeräusches bestimmt. Anschließend wird noch eine Laufzeituntersuchung auf dem ReSpeaker Core v2.0 durchgeführt.

6.2.1 Überprüfung des Dämpfungsgrades des Störgeräusches

Für die Simulationen wird der unter [32] befindliche Quellcode verwendet. Es handelt sich dabei um den extrahierten Quellcode der Rauschunterdrückung aus dem WebRTC Modul.

Der NS-Algorithmus bietet vier verschiedene Intensitätsstufen. Die Höhe der Dämpfung wird anhand des Verhältnisses der Ein- und Ausgabeleistung des eingespeisten weißen Rauschens ermittelt. Die sich ergebenen Dämpfungen sind in Tabelle 6.1 dargestellt.

Tabelle 6.1: Dämpfungsgrad der Rauschunterdrückung

Intensitätsstufe	Dämpfung in dB
0	-6
1	-12
2	-18
3	-21

Der zeitliche Signalverlauf des weißen Rauschens und der dazugehörigen rauschunterdrückten Signale sind in Abbildung 6.3 dargestellt. Besonders bei einer höheren Intensitätsstufen ist zu erkennen, dass das System eine gewisse Zeit zum Einschwingen braucht.

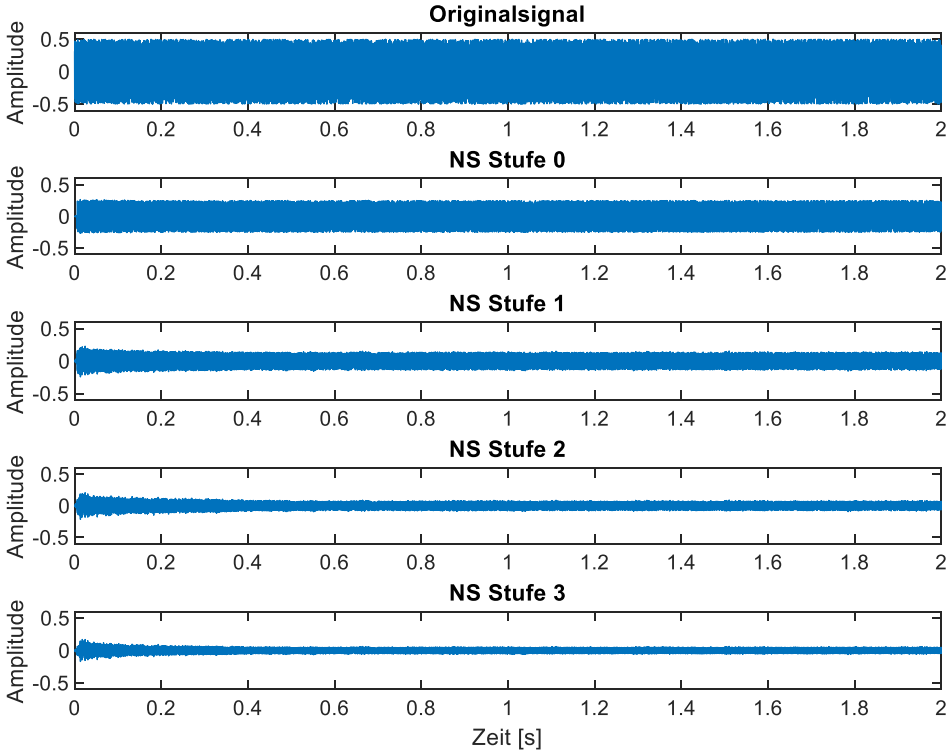


Abbildung 6.3: Signalverlauf eines rauschunterdrückten weißen Rauschsignals bei verschiedenen Intensitätsstufen

Als nächstes wird die Rauschunterdrückung an einem verrauschten Sprachsignal getestet. Dazu wird ein Sprachsignal mit einem weißen Rauschen überlagert. Das verrauschte Signal besitzt ein

Signal-Rausch-Verhältnis von 14 dB. In Abbildung 6.4 ist das reine Sprachsignal (oben), das verrauschte Sprachsignal (mittig) und das rauschunterdrückte Signal (unten) mit einer Intensitätsstufe von 3 dargestellt. Eine Simulation mit allen Intensitätsstufen ist im Anhang unter B.2 zu finden.

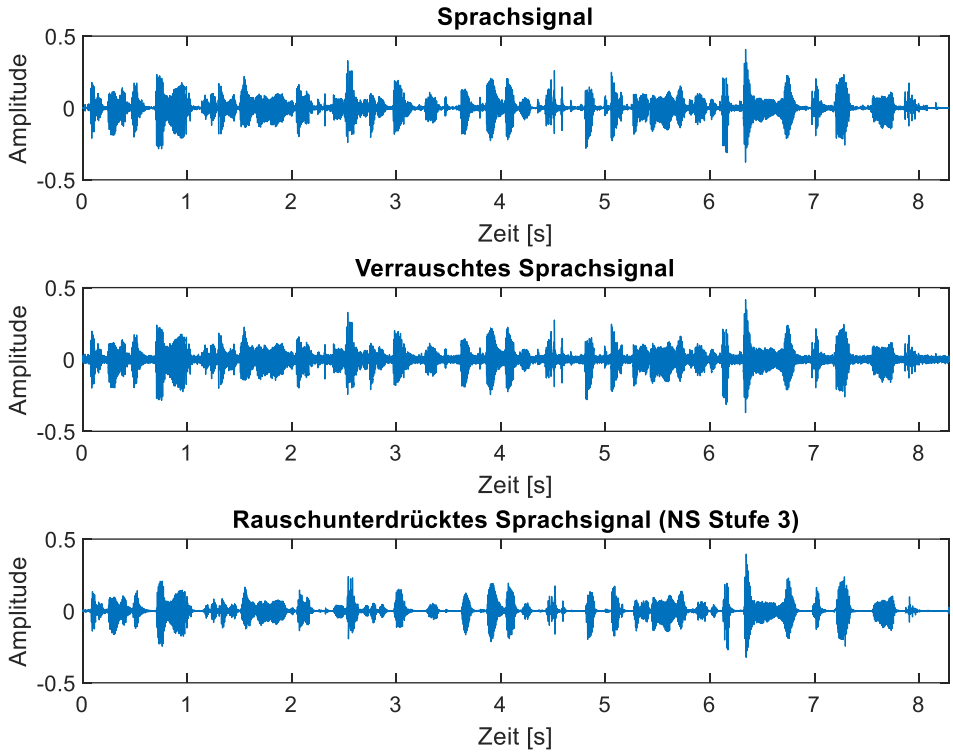


Abbildung 6.4: Signalverlauf eines Sprachsignals (oben), verrauschten Sprachsignals (mittig) und des rauschunterdrückten Signals (unten) bei einer Intensitätsstufe von 3

In der Abbildung 6.4 ist zu erkennen, dass das Rauschen stark gedämpft ist. Im Vergleich zum reinen Sprachsignal wirkt das rauschunterdrückte Signal zudem leicht gedämpft. Schaut man sich das Spektrogramm der Signale an (siehe Abbildung 6.5), so ist eine deutliche Dämpfung des Rauschens zu erkennen. Es werden aber auch einige Sprachanteile unterdrückt. Insbesondere die Anteile mit einer geringen energiedichte. Dieser Effekt verstärkt sich je geringer das Signal-Rausch-Verhältnis ist.

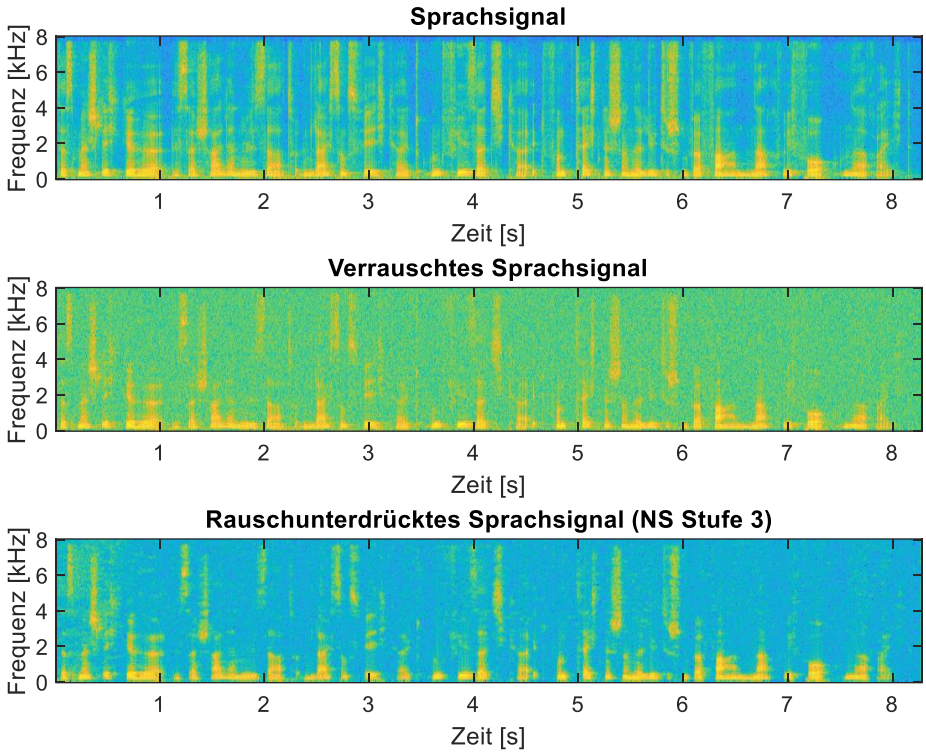


Abbildung 6.5: Spektrogramm eines Sprachsignals (oben), verrauschten Sprachsignals (mittig) und des rauschunterdrückten Signals (unten) bei einer Intensitätsstufe von 3

Anstelle eines breitbandigen Störsignals wird das Verhalten des Rauschunterdrückungsalgorithmus auf ein schmalbandiges Störsignal geprüft. Dafür wird das Sprachsignal mit einem Sinussignal überlagert. Die Frequenz des Sinussignals beträgt 750 Hz und das SNR 14 dB. Der zeitliche Signalverlauf ist in Abbildung 6.6 dargestellt. Auch hier ist zu erkennen, dass das Störsignal gedämpft wird.

Im Spektrogramm erkennt man, dass das System im Vergleich zum Rauschen eine längere Zeit benötigt, um die Sinusstörung auf denselben Betrag zu dämpfen (siehe Abbildung 6.7). Es sind um die drei Sekunden dafür notwendig.

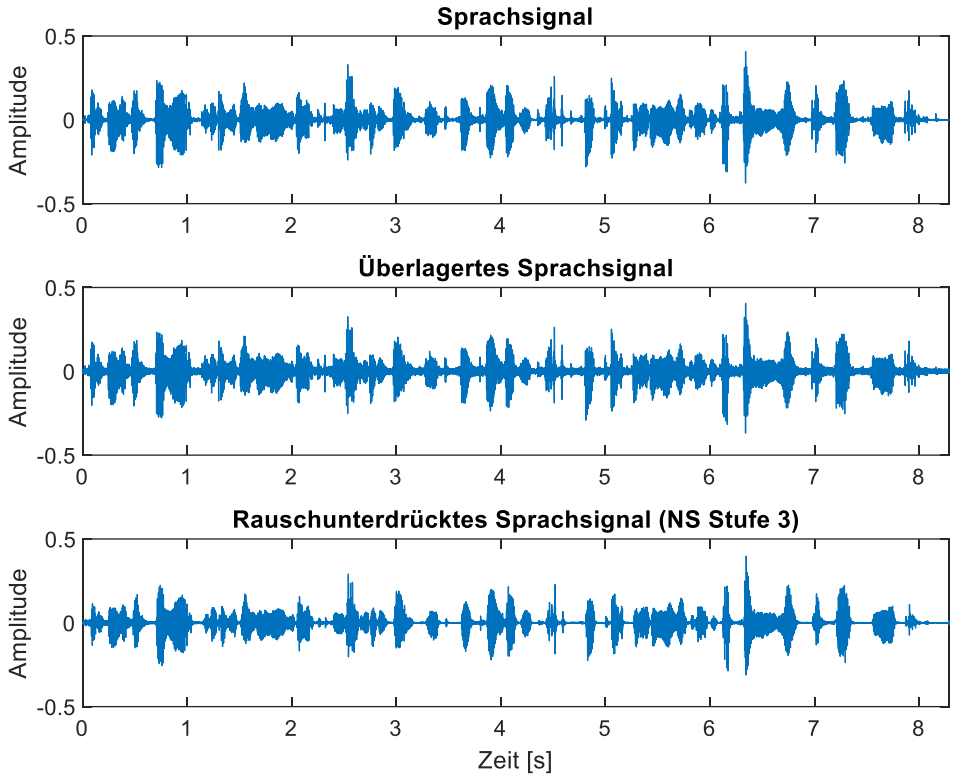


Abbildung 6.6: Signalverlauf eines Sprachsignals (oben), überlagerten Sprachsignals mit einem Sinussignal von f=750 Hz (mittig) und des rauschunterdrückten Signals (unten) bei einer Intensitätsstufe von 3

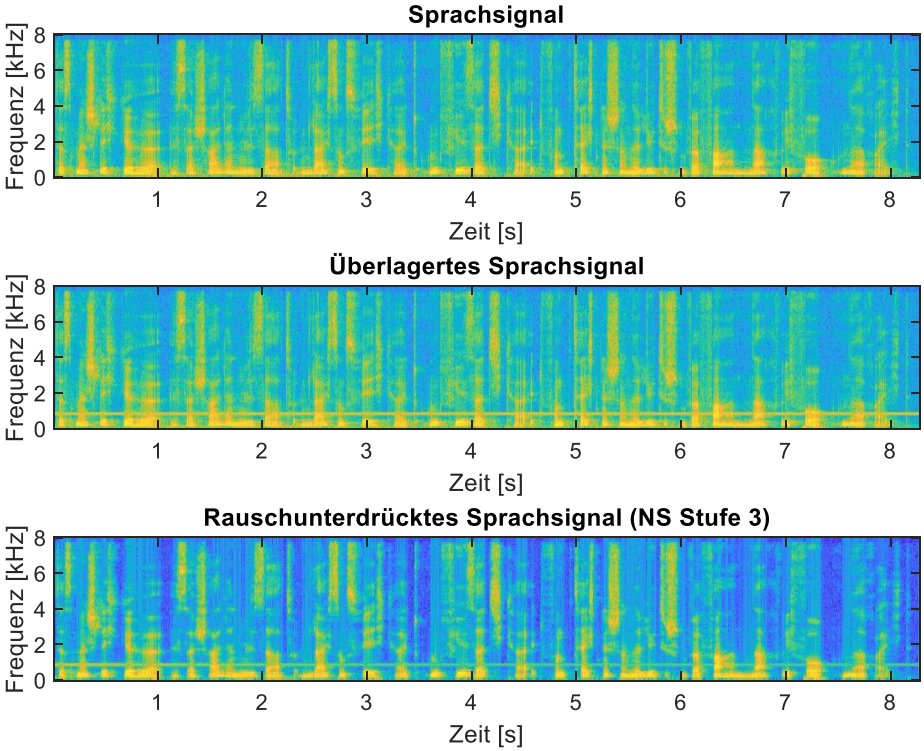


Abbildung 6.7: Spektrogramm eines Sprachsignals (oben), überlagerten Sprachsignals mit einem Sinussignal von f=750 Hz (mittig) und des rauschunterdrückten Signals (unten) bei einer Intensitätsstufe von 3

6.2.2 Laufzeituntersuchung des NS-Algorithmus

Die Laufzeitmessung findet auf dem ReSpeaker Core v2.0 anhand einer Differenzzeitmessung statt. Es wird ein weißes Rauschen mit 16000 Werten erstellt und in den NS-Algorithmus eingespeist. Die Abtastfrequenz wird auf 16000 Hz gestellt, was eine Signallänge von einer Sekunde darstellt. Es werden 1000 Messungen durchgeführt und anhand der Zeitmessungen die mittlere Berechnungszeit ermittelt. Die Messung ergab eine mittlere Berechnungszeit von 38 ms für eine Signallänge von einer Sekunde. Für einen Analyseblock von 10 ms ergeben sich dadurch nur 380 µs.

6.3 Test des Algorithmus auf der Hardware

In diesem Abschnitt wird der Rauschunterdrückungsalgorithmus in Verbindung mit dem ReSpeaker Core v2.0 getestet. Die Messungen finden in Kombination mit dem Beamforming statt. Es werden insgesamt drei Messungen durchgeführt.

In der ersten Messung wird das Sprach- und Störsignal aus derselben Richtung ausgegeben. Der Beamformer wird dabei fest auf diese Richtung ausgerichtet. In der zweiten Messung bleibt der Beamformer auf die Richtung des Sprachsignals ausgerichtet, das Störsignal wird aber um 70° versetzt ausgegeben. In der dritten Messung bleiben die Positionen der Audioquellen wie bei Messung zwei, der Beamformer wird jedoch auf eine automatische Ausrichtung gestellt.

Für die Messungen wird ein spezielles Messprogramm auf den ReSpeaker Core v2.0 implementiert. Der Signalfluss ist in Abbildung 6.8 dargestellt.

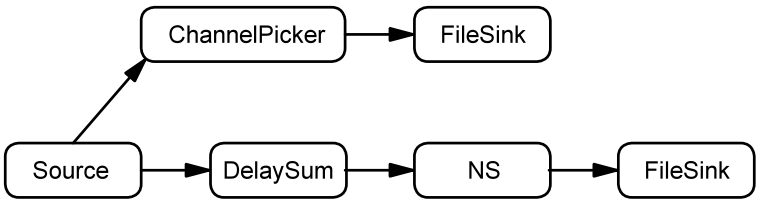


Abbildung 6.8: Signalfluss des NS Messprogramms

Bei der Messung wird das Signal von MIC2, welches das verrauschte Sprachsignal darstellt, sowie das fokussierte und rauschunterdrückte Signal aufgenommen. Die Abtastrate beträgt 16000 Hz und die Blocklänge 1600 Abtastwerte. Die Intensität der Rauschunterdrückung wird auf die maximale Stufe (Stufe 3) gestellt.

Der Messaufbau ist bei allen drei Messungen derselbe. Gemessen wird in einem gewöhnlichen Raum. Für die Ausgabe des Sprach- und Störsignals wird ein Lautsprecher verwendet. Das Signal-Rausch-Verhältnis beträgt bei allen Messungen um die 14 dB. Der Messaufbau ist in Abbildung 6.9 dargestellt.

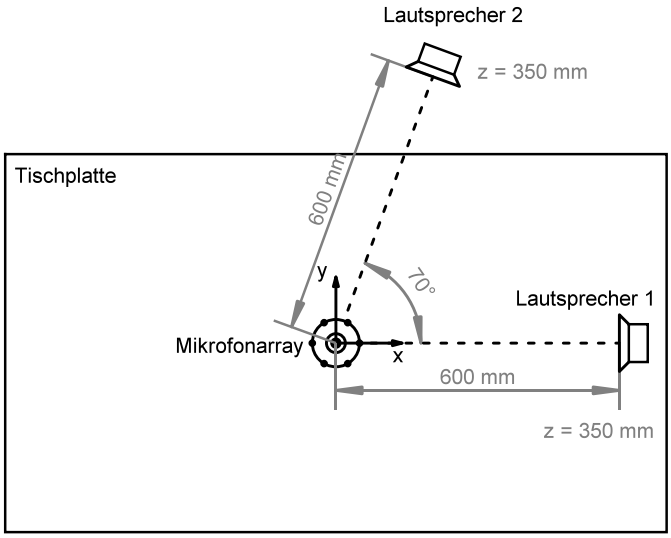


Abbildung 6.9: Messaufbau BF-NS-Messung

Als Störsignal wird anstelle eines weißen Rauschens ein Lüftergeräusch verwendet, da es eine realitätsnahe Störquelle darstellt. Bei dem Sprachsignal handelt es sich um dasselbe wie auch schon in den Messungen zuvor. Der genaue Wortlaut ist im Anhang unter B.1 zu finden. Das Spektrogramm des Störsignals und des Sprachsignals ist in Abbildung 6.10 dargestellt. Das Störsignal hat besonders im unteren Frequenzbereich eine hohe Leistungsdichte und nimmt mit steigender Frequenz ab.

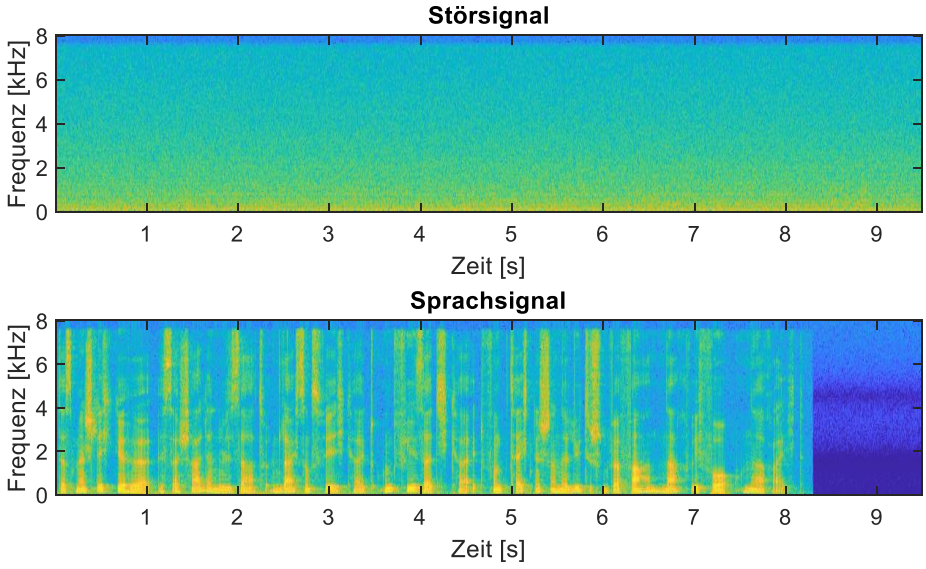


Abbildung 6.10: Spektrogramm des Störsignals (oben) und Sprachsignals (unten)

Messung 1:

Versuchsvorbereitung und Durchführung

Das Stör- und Sprachsignal werden aus derselben Richtung von Lautsprecher 1 ausgegeben. Der Beamformer wird fest auf die Richtung des Sprachsignals (Lautsprecher 1) ausgerichtet. Da beide Signale aus derselben Richtung ausgegeben werden, hat der Beamformer keinen Einfluss auf das Störsignal.

Das Messprogramm wird auf dem ReSpeaker Core v2.0 gestartet und die beiden Audiosignale abgespielt. Nach Beendigung des Sprachsignals wird das Störsignal noch einige Sekunden weiter aufgenommen. In diesem Abschnitt kann der Dämpfungsgrad des Störsignals ermittelt werden.

Ergebnis:

Der Signalverlauf des verrauschten und rauschunterdrückten Sprachsignals ist in Abbildung 6.11 dargestellt.

Das rauschunterdrückte Sprachsignal weist so gut wie keine Störung auf. Wie auch in der Simulation, ist eine geringe Dämpfung des Sprachsignals erkennbar. Berechnet man die Dämpfung über den Bereich, in dem nur das Störsignal vorliegt, so ergibt sich eine Dämpfung von um die 21,4 dB. Dieser Wert entspricht dem in der Simulation gemessenen Wert.

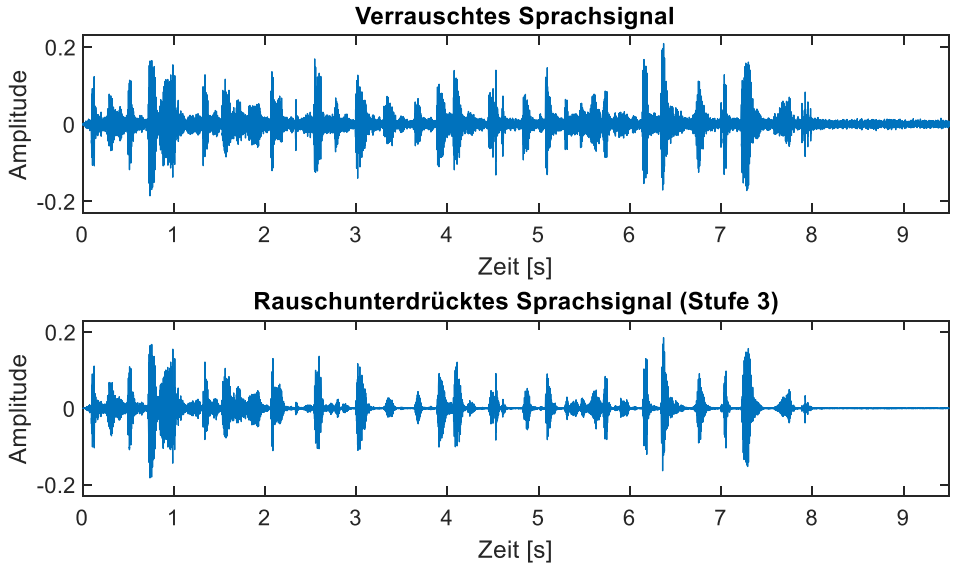


Abbildung 6.11: Messung 1, Signalverlauf des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

Betrachtet man das Spektrogramm beider Signale (siehe Abbildung 6.12), so ist zu erkennen, dass die Frequenzanteile des Störsignals nahezu vollständig entfernt wurden. Wird das Spektrogramm des rauschunterdrückten Sprachsignals (Abbildung 6.12 unten) mit dem des reinen Sprachsignals (Abbildung 6.10 unten) verglichen, so sind einige Anteile des Sprachsignals gedämpft dargestellt.

In den ersten Sekunden ist sichtbar, dass das System eine gewisse Zeit brauch, um die Störung zu kompensieren. Im letzten Abschnitt ist noch ein kleiner Teil des Störsignals bei einer Frequenz um die 300 Hz erkennbar.

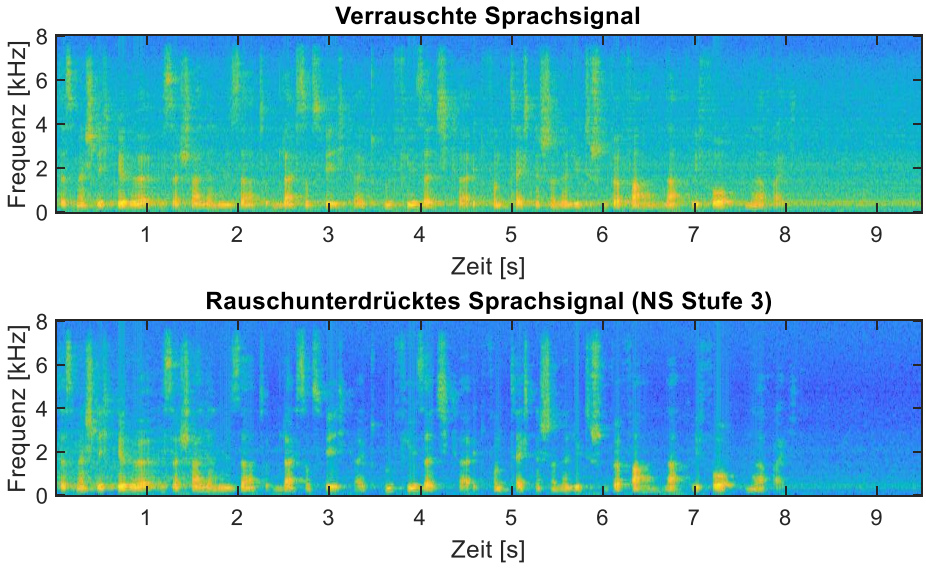


Abbildung 6.12: Messung 1, Spektrogramm des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

Messung 2:

Versuchsvorbereitung und Durchführung

Das Störsignal wird um 70° versetzt zu dem Sprachsignal ausgegeben. Nach Abbildung 6.9 wird das Störsignal aus Lautsprecher 2 und das Sprachsignal aus Lautsprecher 1 ausgegeben. Der Beamformer wird fest auf die Richtung des Sprachsignals (Lautsprecher 1) ausgerichtet. Da das Störsignal aus einer anderen Richtung kommt, sollte der Beamformer dieses Dämpfen.

Wie auch bei der Messung 1 wird das Messprogramm auf dem ReSpeaker Core v2.0 gestartet und die beiden Audiosignale abgespielt. Um die reine Dämpfung des Störsignals zu berechnen, wird das Störsignal noch einige Sekunden weiter aufgenommen.

Ergebnis:

In Abbildung 6.13 sind der Signalverlauf das verrauschte Sprachsignal (oben) und das rauschunterdrückte Sprachsignal (unten) dargestellt.

Auch hier sind dieselben Effekte wie bei der Messung 1 zu erkennen. Betrachtet man die Dämpfung des reinen Störsignals im letzten Abschnitt, so beträgt diese 22,1 dB. Die Dämpfung ist im Gegensatz zu der Messung 1, wo der Beamformer keinen Einfluss hatte, um 0,7 dB stärker.

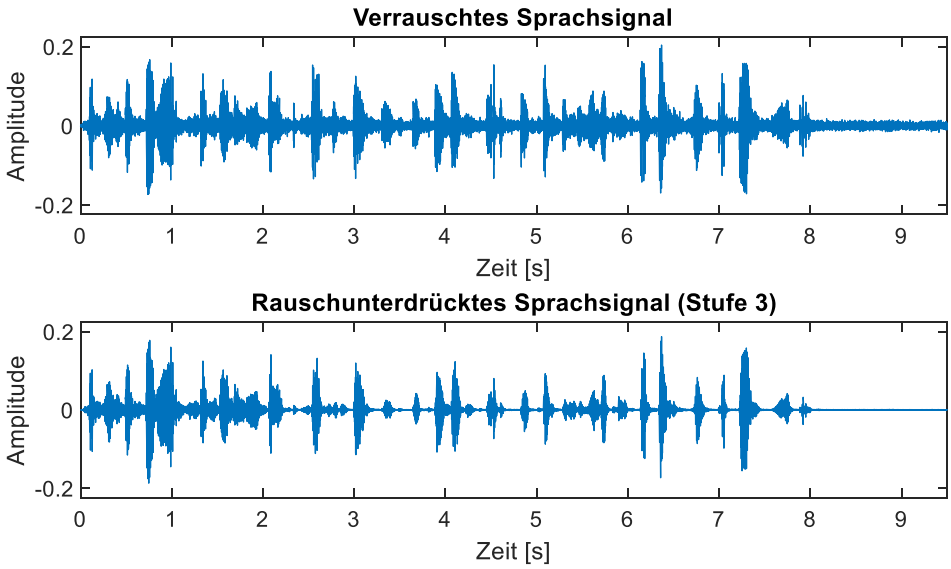


Abbildung 6.13: Messung 2, Signalverlauf des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

Schaut man sich das Spektrogramm beider Signale an (siehe Abbildung 6.14), so ist besonders am Ende zu erkennen, dass in dem Bereich von 2 kHz bis 6 kHz eine etwas stärkere Dämpfung des Störsignals stattfindet. Zudem ist noch sichtbar, dass das Störsignal am Anfang schwächer ausgeprägt ist.

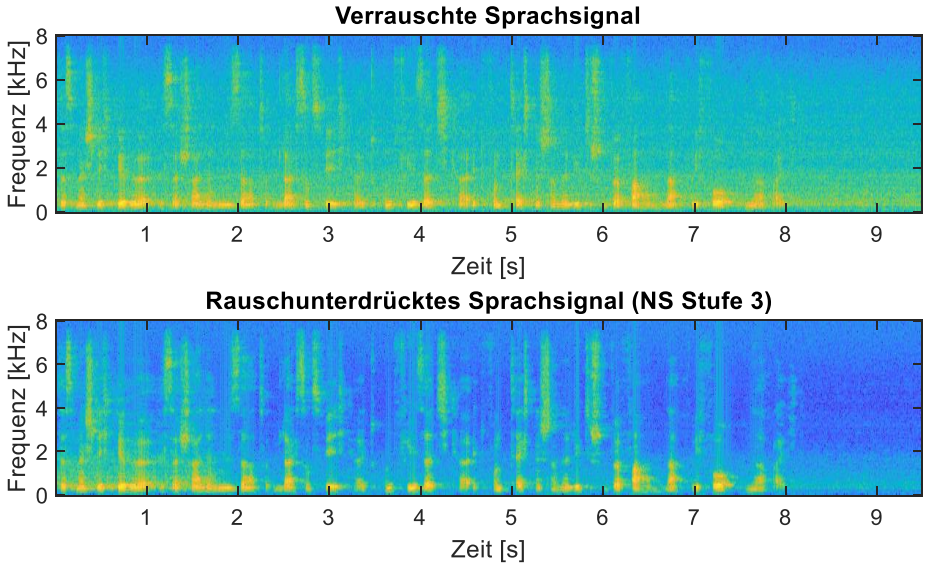


Abbildung 6.14: Messung 2, Spektrogramm des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

Die zusätzliche Dämpfung von 0,7 dB im Vergleich zu Messung 1 ist in Bezug zum Beampattern (siehe Abbildung 5.8) zu gering. Das liegt daran, dass das Störsignal keine Sinusförmige Form aufweist und es bei einer zeitlichen Verschiebung der Signale nicht offensichtlich zu gegenphasigen Signalen kommt. Zudem befindet sich ein Großteil der Energie des Störsignals im unteren Frequenzbereich. Diese Frequenzen können aufgrund der geringen Ausdehnung der Mikrofone nicht ausreichend gedämpft werden.

Simuliert man das Beampattern mit dem verwendeten Störsignal aus der Messung, so ergibt sich der in Abbildung 6.15 dargestellte Verlauf.

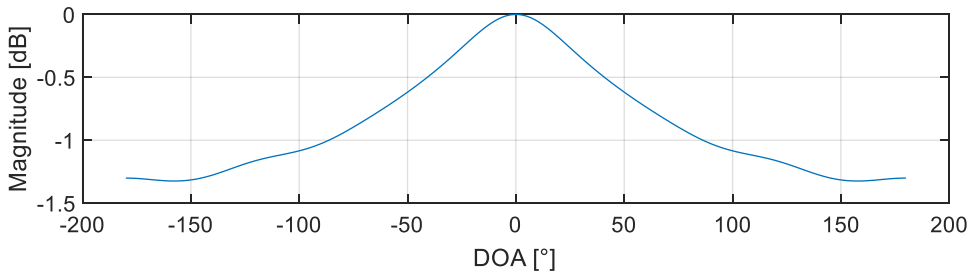


Abbildung 6.15: Beampattern für das Lüftergeräusch aus der Messung

Schaut man sich die Dämpfung bei einem Winkel von 70° an, so beträgt diese um die 0,8 dB. Dieser Wert entspricht näherungsweise dem aus der Messung.

Messung 3:

Versuchsvorbereitung und Durchführung

Bei dieser Messung wird das Störsignal auch um 70° versetzt zum Sprachsignal ausgegeben. Nach Abbildung 6.9 wird das Störsignal aus Lautsprecher 2 ausgegeben und das Sprachsignal aus Lautsprecher 1. Der Beamformer wird auf die automatische Fokussierung gestellt.

Wie auch bei den Messungen zuvor, wird das Messprogramm auf dem ReSpeaker Core v2.0 gestartet und die beiden Audiosignale abgespielt. Nach dem Ende des Sprachsignals wird über eine kurze Zeit das reine Störsignal aufgenommen, um die Dämpfung des Störsignals zu berechnen.

Ergebnis:

Die Abbildung 6.16 zeigt den Signalverlauf des verrauschten und rauschunterdrückten Sprachsignals. Im Vergleich zu den vorherigen Messungen, sind hier keine großen Veränderungen sichtbar. Die Dämpfung des Störgeräusches im letzten Abschnitt beträgt 21,1 dB. Dieser Wert ähnelt dem aus Messung 1. Daraus lässt sich vermuten, dass sich der Beamformer am Ende auf das Störsignal ausgerichtet hat.

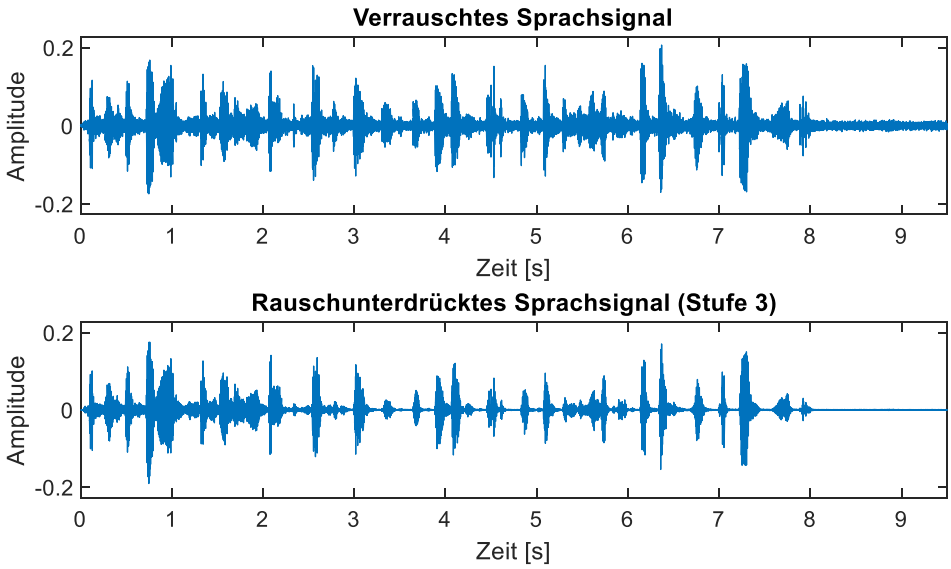


Abbildung 6.16: Messung 3, Signalverlauf des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

In dem Spektrogramm beider Signale (siehe Abbildung 6.17) erkennt man im Vergleich zu den vorherigen Messungen, dass in einigen Bereichen Anteile von dem Sprachsignal stärker gedämpft wurden. Daraus kann geschlossen werden, dass sich der Beamformer in diesen Bereichen auf die Störquelle ausgerichtet hat und somit das Sprachsignal gedämpft wurde. In anderen Bereichen ist wiederrum das Störgeräusch stärker gedämpft. Die Ausrichtung des Beamformers wechselt somit zwischen den beiden Audioquellen.

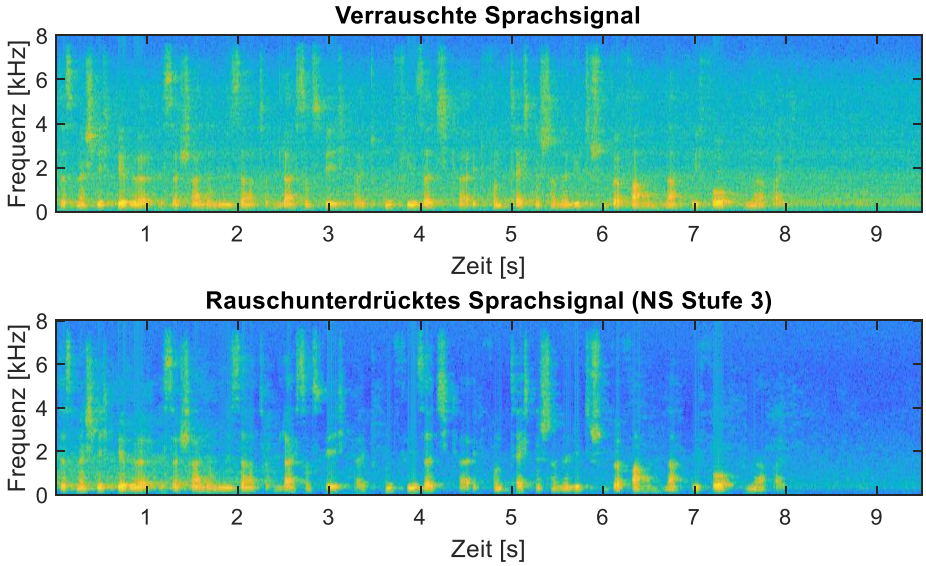


Abbildung 6.17: Messung 3, Spektrogramm des verrauschten Sprachsignals (oben) und rauschunterdrückten Sprachsignals (unten).

6.4 Zusammenfassung und Bewertung

Die Rauschunterdrückung lieferte bei den Simulationen durchweg positive Ergebnisse. Das Störgeräusch wird selbst bei aktiver Sprache zuverlässig abgeschätzt und gedämpft. Dadurch kann das System schnell auf Störgeräusche reagieren. Zudem werden nicht nur breitbandige, sondern auch schmalbandige Störgeräusche gedämpft. Im Vergleich zu den breitbandigen Störgeräuschen benötigt das System jedoch eine etwas längere Zeitspanne bis zur maximalen Dämpfung.

Die Intensität der Dämpfung ist mit maximal 21 dB hoch, dadurch können auch starke Störgeräusche auf ein Minimum reduziert werden. Bei geringen Signal-Rausch-Verhältnissen werden jedoch auch größere Sprachanteile gedämpft. Für bessere Signal-Rausch-Verhältnisse ist es empfehlenswert eine geringere Intensitätsstufe zu verwenden, da die Rauschunterdrückung bei der höchsten Stufe zu aggressiv vorgeht.

Vergleicht man die simulierten mit den auf dem Gerät gemessenen Ergebnisse, so sind keine Unterschiede in Bezug zu der Intensität der Dämpfung erkennbar. Die Messungen mit dem Beamformer zeigen, dass sich die Verwendung des Beamformers nur gering auf die Dämpfung von breitbandigen Störgeräuschen auswirkt. Wo der fest eingestellte Beamformer noch eine geringe Dämpfung erzielt, bringt der automatisch ausrichtender Beamformer hingegen kaum einen Vorteil. Durch die ständige Berechnung der Verzögerungen kommt es vor, dass sich der Beamformer nicht nur auf den Sprecher, sondern zwischenzeitig auf die Störquelle ausrichtet. Um dies zu vermieden können die Verzögerungen nur während der aktiven Sprache berechnet werden. Der Beamformer bleibt dabei so lange auf den Punkt ausgerichtet, bis wieder ein Sprachsignal vorliegt.

7 Fazit

In der Arbeit wurde das 6-Mikrofon-Array von ReSpeaker vorgestellt und eingerichtet. Die frei verfügbaren Sprachverbesserungsalgorithmen zur Lokalisation einer Audioquelle, Beamforming und Rauschunterdrückung wurden analysiert und anhand von Simulationen und Tests auf der Hardware untersucht. Das Ziel war die Leistungsfähigkeit der genannten Sprachverbesserungsalgorithmen in Simulationen und in Verbindung mit der Hardware darzustellen.

Der Algorithmus zur Lokalisation einer Audioquelle zeigte sowohl bei den Simulationen als auch bei den Tests auf der Hardware einige Kritikpunkte und Schwächen auf. Ein Kritikpunkt ist die grundlegende Ausgabe des ermittelten Winkels. Dieser weicht bei einem steigenden Elevationswinkel bereichsweise immer stärker von dem Azimutwinkel ab. Eine Ausgabe des Azimut- und Elevationswinkels wurde nicht wahrgenommen, obwohl diese aufgrund der Mikrofonanordnung möglich ist.

Die Lokalisationsgenauigkeit und -zuverlässigkeit ist in idealer und besonders bei realer Umgebung nur schlecht gegeben. Wo in der Simulation die Anzahl der Fehlmessungen außerhalb eines Toleranzbereiches von ±15° bei maximal 3 % lagen, waren es in der idealen Umgebung 13 % und in der realen Umgebung 35 %. Bei einem Interpolationsfaktor von 10 waren die simulierten Ergebnisse alle im Toleranzbereich, wohingegen bei der realen Umgebung es stellenweise zu einer Fehlerrate von 70 % kam.

Der implementierte Beamformer beruht auf einem simplen und einfachen Verfahren. Das simulierte und aufgenommene Beampattern ähneln sich in ihrer Form und es ist nur bei der maximalen Dämpfung ein Unterschied zu erkennen. Wo das simulierte Beampattern eine maximale Dämpfung von 40 dB erreicht, liegt die maximale Dämpfung beim aufgenommenen Beampattern bei 25 dB.

Die Richtwirkung des Beamformers ist gering. Frequenzen unterhalb von 1000 Hz werden nicht gedämpft. Zudem werden rauschähnliche Signale nur schwach um maximal 1 dB bis 2 dB gedämpft. Aufgrund dieser Tatsache bringt die Verwendung des Beamformers nur einen geringen Mehrwert beim praktischen Einsatz.

Die Implementierung mit der automatischen Ausrichtung des Beamformers zeigt Schwächen auf. Der Beamformer bleibt nicht konsequent auf dem Sprecher ausgerichtet, sondern richtet sich zwischenzeitig auch auf die Störquelle aus. Dies resultiert mit einem gedämpften Sprachsignal.

Die Rauschunterdrückung liefert sowohl bei den Simulationen als auch bei den Tests auf der Hardware durchweg positive Ergebnisse. Mit einer maximalen Dämpfung von 21 dB, können selbst starke Störgeräusche auf ein Minimum gedämpft werden. Bei geringen Signal-Rausch-Verhältnissen werden jedoch auch Teile des Sprachsignals gedämpft.

Durch das verwendete Rauschunterdrückungsverfahren findet selbst bei aktiver Sprache eine Rauschreduktion statt. Dadurch kann das Störsignal schneller unterdrückt werden. Auch schmalbandige Störsignale werden vom System detektiert und gedämpft.

Die untersuchten Sprachverbesserungsalgorithmen überzeugen bis auf die Rauschunterdrückung nur wenig. Der Lokalisationsalgorithmus bedarf einer Optimierung im Bereich der Winkelausgabe und der Stabilität. Der Delay and Sum Beamformer stellt in Verbindung mit dem Mikrofonarray nur eine geringe Richtwirkung dar. Zudem ist die automatische Ausrichtung nicht optimal.

Abschließend lässt sich über den ReSpeaker Core v2.0 sagen, dass dieser eine gute Plattform für Sprachapplikationen bietet. Aufgrund der kompakten Bauweise mit dem Mikrofonarray und der Recheneinheit auf einer Platine kann das Gerät in viele Applikationen eingebunden werden.

Literaturverzeichnis

[1] THE MATHWORKS, INC.: *Spherical Coordinates*. URL https://de.mathworks.com/help/phased/ug/spherical-coordinates.html – Abruf 2019-05-15

[2] GIANCOLI, D. C.: *Physik: Lehr- und Übungsbuch* : Pearson Studium, 2010 (Pearson Studium - Physik)

[3] TEUTSCH, H.: *Modal Array Signal Processing: Principles and Applications of Acoustic Wavefield Decomposition* : Springer Berlin Heidelberg, 2007 (Lecture Notes in Control and Information Sciences)

[4] FREY, Thomas ; BOSSERT, Martin: *Signal- und Systemtheorie* : *Mit 26 Tabellen, 64 Aufgaben mit Lösungen und 84 Beispielen*. 2., korrigierte Auflage 2008. Wiesbaden : Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2009 (Studium)

[5] PFISTER, Beat ; KAUFMANN, Tobias: *Sprachverarbeitung*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2017

[6] VARY, P. ; MARTIN, R.: *Digital Speech Transmission: Enhancement, Coding and Error Concealment* : Wiley, 2006

[7] EULER, Stephen: *Grundkurs Spracherkennung* : *Vom Sprachsignal zum Dialog - Grundlagen und Anwendung verstehen - Mit praktischen Übungen*. 1. Aufl. Wiesbaden : Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH Wiesbaden, 2006 (Computational Intelligence)

[8] SEEED TECHNOLOGY CO.,LTD.: *ReSpeaker Core v2.0 - Seeed Studio*. URL https://www.seeedstudio.com/ReSpeaker-Core-v2-0.html – Abruf 2019-04-29

[9] SEEED TECHNOLOGY CO.,LTD.: *ReSpeaker Core v2.0 - Seeed Wiki*. URL http://wiki.seeedstudio.com/ReSpeaker_Core_v2.0/ – Abruf 2019-04-29

[10] SEEED TECHNOLOGY CO.,LTD.: *Grove System*. URL http://wiki.seeedstudio.com/Grove_System/ – Abruf 2019-05-03

[11] X-POWERS LIMITED: *AC108 Datasheet*. URL https://www.distrelec.de/Web/Downloads/_t/ds/SeeedStudio_AC108_eng_tds.pdf. – Aktualisierungsdatum: 2017-11-01 – Abruf 2019-05-04

[12] GANTEN, Peter H. ; ALEX, Wulf: *Debian GNU/Linux* : *Grundlagen, Einrichtung und Betrieb*. 3., überarbeitete Auflage. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2007 (Univention)

[13] FANGCLOUD: *ReSpeaker System Images*. URL https://v2.fangcloud.com/share/7395fd138a1cab496fd4792fe5?lang=en – Abruf 2019-05-04

[14] YIHUI XIONG: *voice-engine*. URL https://github.com/voice-engine/voice-engine – Abruf 2019-05-08

[15] GITHUB, INC.: *respeakerd*. URL https://github.com/respeaker/respeakerd – Abruf 2019-05-05

[16] ORFANIDIS, Sophocles J.: *Optimum signal processing* : *An introduction*. 2. ed. New York, NY : McGraw Hill, 1988

[17] KUTTNER, Thomas: *Praxiswissen Schwingungsmesstechnik*. Wiesbaden : Springer Fachmedien Wiesbaden, 2015

[18] BRANDSTEIN, M. S. ; SILVERMAN, H. F.: A robust method for speech signal time-delay estimation in reverberant rooms. In: *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing* : IEEE Comput. Soc. Press, 1997, S. 375–378

[19] KNAPP, C. ; CARTER, G.: *The generalized correlation method for estimation of time delay*. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24 (1976), Nr. 4, S. 320–327

[20] POURMOHAMMAD, A. ; AHADI, S. M.: *Real Time High Accuracy 3-D PHAT-Based Sound Source Localization Using a Simple 4-Microphone Arrangement*. In: *IEEE Systems Journal* 6 (2012), Nr. 3, S. 455–468

[21] JOHNSON, Don H. ; DUDGEON, Dan E.: *Array signal processing* : *Concepts and techniques*. Upper Saddle River, NJ : PTR Prentice Hall, 1993 (Prentice-Hall signal processing series)

[22] LI, Jian ; STOICA, Petre: *Robust adaptive beamforming*. Hoboken, NJ : John Wiley, 2006 (Wiley Series in Telecommunications and Signal Processing v.88)

[23] FUNG, James: *Effects of Steering Delay Quantization in Beamforming* (2003). URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.477&rep=rep1&type=pdf

[24] WÖLFEL, Matthias ; MCDONOUGH, John: *Distant speech recognition*. Chichester U.K. : Wiley, 2009

[25] NAIDU, Prabhakar S.: *Sensor array signal processing*. Boca Raton, Fla : CRC Press, 2001

[26] MARKUS SCHWAB: *Modellbasiertes einkanaliges Rauschreduktionsverfahren angewendet auf gestörte Sprachsignale kombiniert mit einem mehrkanaligen Beamformingverfahren*. Berlin, Technischen Universität Berlin. Dissertation. 2010. URL https://pdfs.semanticscholar.org/7a04/f1a1c160a9af77936a9303c7ce61663ad998.pdf

[27] MCAULAY, R. ; MALPASS, M.: *Speech enhancement using a soft-decision noise suppression filter*. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28 (1980), Nr. 2, S. 137–145

[28] STAHL, V. ; FISCHER, A. ; BIPPUS, R.: Quantile based noise estimation for spectral subtraction and Wiener filtering. In: *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)* : IEEE, 2000, S. 1875–1878

[29] LIM, J. S. ; OPPENHEIM, A. V.: *Enhancement and bandwidth compression of noisy speech*. In: *Proceedings of the IEEE* 67 (1979), Nr. 12, S. 1586–1604

[30] ABD EL-FATTAH, Marwa A. ; DESSOUKY, Moawad I. ; ABBAS, Alaa M. ; DIAB, Salaheldin M. ; EL-RABAIE, El-Sayed M. ; AL-NUAIMY, Waleed ; ALSHEBEILI, Saleh A. ; ABD EL-SAMIE, Fathi E.: *Speech enhancement with an adaptive Wiener filter*. In: *International Journal of Speech Technology* 17 (2014), Nr. 1, S. 53–64

[31] EPHRAIM, Y. ; MALAH, D.: *Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator*. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32 (1984), Nr. 6, S. 1109–1121

[32] GITHUB, INC.: *WebRTC_NS* : *Noise Suppression Module Port From WebRTC*. URL https://github.com/cpuimage/WebRTC_NS – Abruf 2019-06-06

[33] YIHUI XIONG: *speexdsp for python*. URL https://github.com/xiongyihui/speexdsp-python – Abruf 2019-05-08

[34] JOHN WISEMAN: *py-webrtcvad*. URL https://github.com/wiseman/py-webrtcvad – Abruf 2019-05-08

[35] GENUIT, Klaus: *Sound-Engineering im Automobilbereich* : *Methoden zur Messung und Auswertung von Geräuschen und Schwingungen*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AEC	Acoustic Echo Cancelation
BF	Beamforming
CPU	Central Processing Unit
DDR	Double Data Rate
DOA	Direction of Arrival
DS	Delay and Sum
eMMC	Embedded Multi Media Card
GCC-PHAT	Generalized Cross Correlation - Phase Transform
GPIO	General Purpose Input Output
GPU	Graphical Processing Unit
GUI	Graphical User Interface
KWS	Keyword Spotting
NS	Noise Suppression
PMU	Power Management Unit
RAM	Random-Access Memory
TDOA	Time Difference of Arrival
UART	Universal Asynchronous Receiver Transmitter
VAD	Voice Activity Detection

Anhang

A ReSpeaker Core v2.0

A.1 ReSpeaker Core v2.0 Blockschaltbild

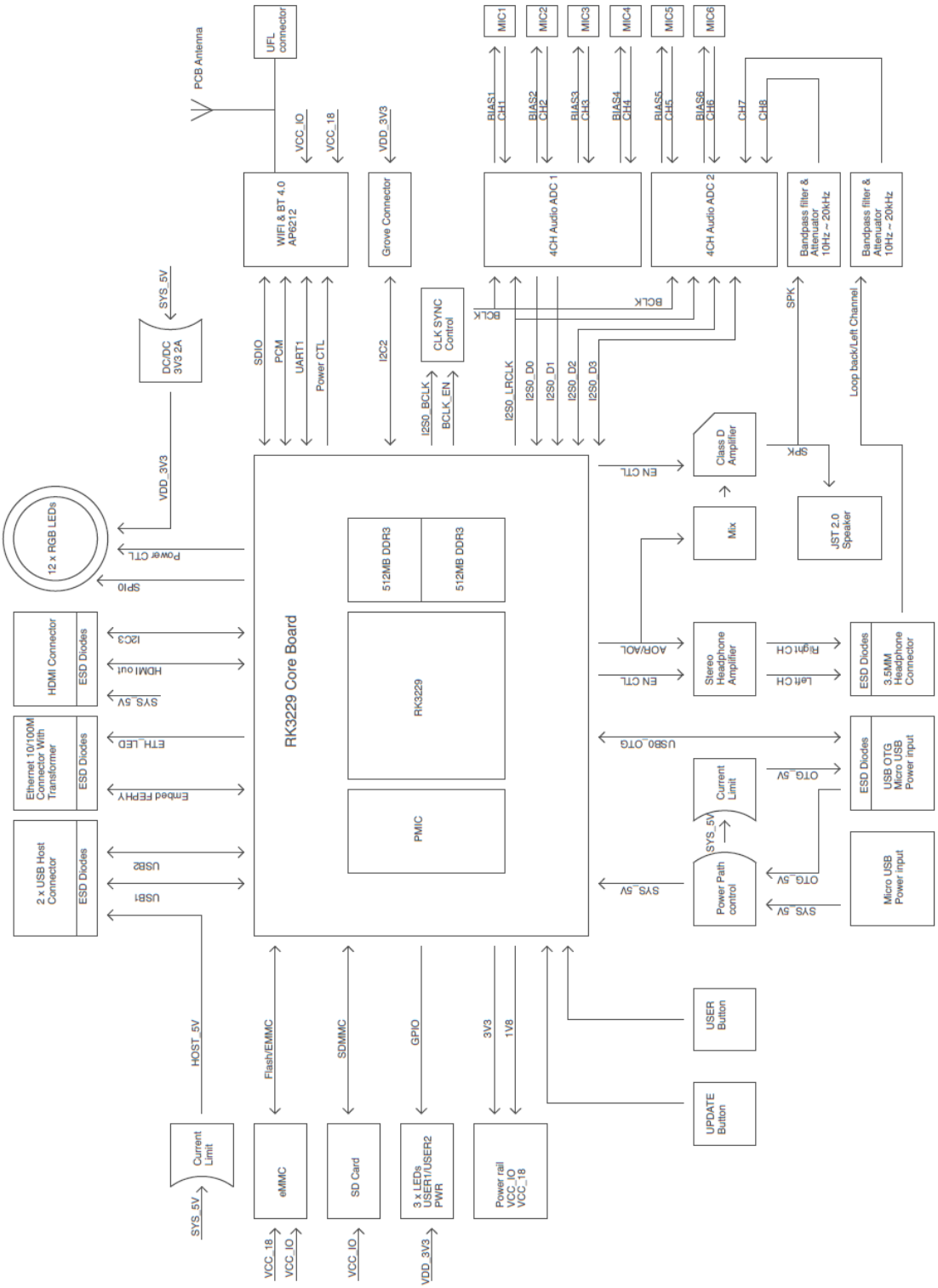


Abbildung A.1: Blockschaltbild des ReSpeaker Core v2.0

A.2 Installation weiterer Pakete

Um die Module KWS, AEC und VAD aus der Voice-Engine nutzen zu können sind weitere Pakete notwendig. Die Terminalbefehle zur Installation dieser Pakete sind hier einmal aufgelistet.

Für die Schlüsselwort-Erkennung muss ein Verzeichnis geladen werden und das notwendige Python Paket manuell gebildet werden. Zur Installation des Pakets sind die folgenden Anweisungen notwendig [14].

```
git clone --depth 1 https://github.com/Kitt-AI/snowboy.git snowboy_github
cd snowboy_github
sudo apt install libatlas-base-dev swig
python setup.py build
sudo pip install .
```

Für die akustische Echokompensation ist das *speexdsp* Paket notwendig. Zur Installation des Pakets sind die folgenden Anweisungen notwendig [33]:

```
sudo apt-get install libspeexdsp-dev
pip install speexdsp
```

Für die Sprachpausenerkennung ist das *webrtcvad* Paket notwendig. Das paket kann mit dem folgenden Befehl installiert werden [34]:

```
pip install webrtcvad
```

Zusätzlich ist noch ein Paket zum Ansteuern des LED Rings vom ReSpeaker Core v2.0 erhältlich. Dies kann mit

```
pip install pixel-ring
```

installiert werden [9].

A.3 Voice-Engine Klassenübersicht

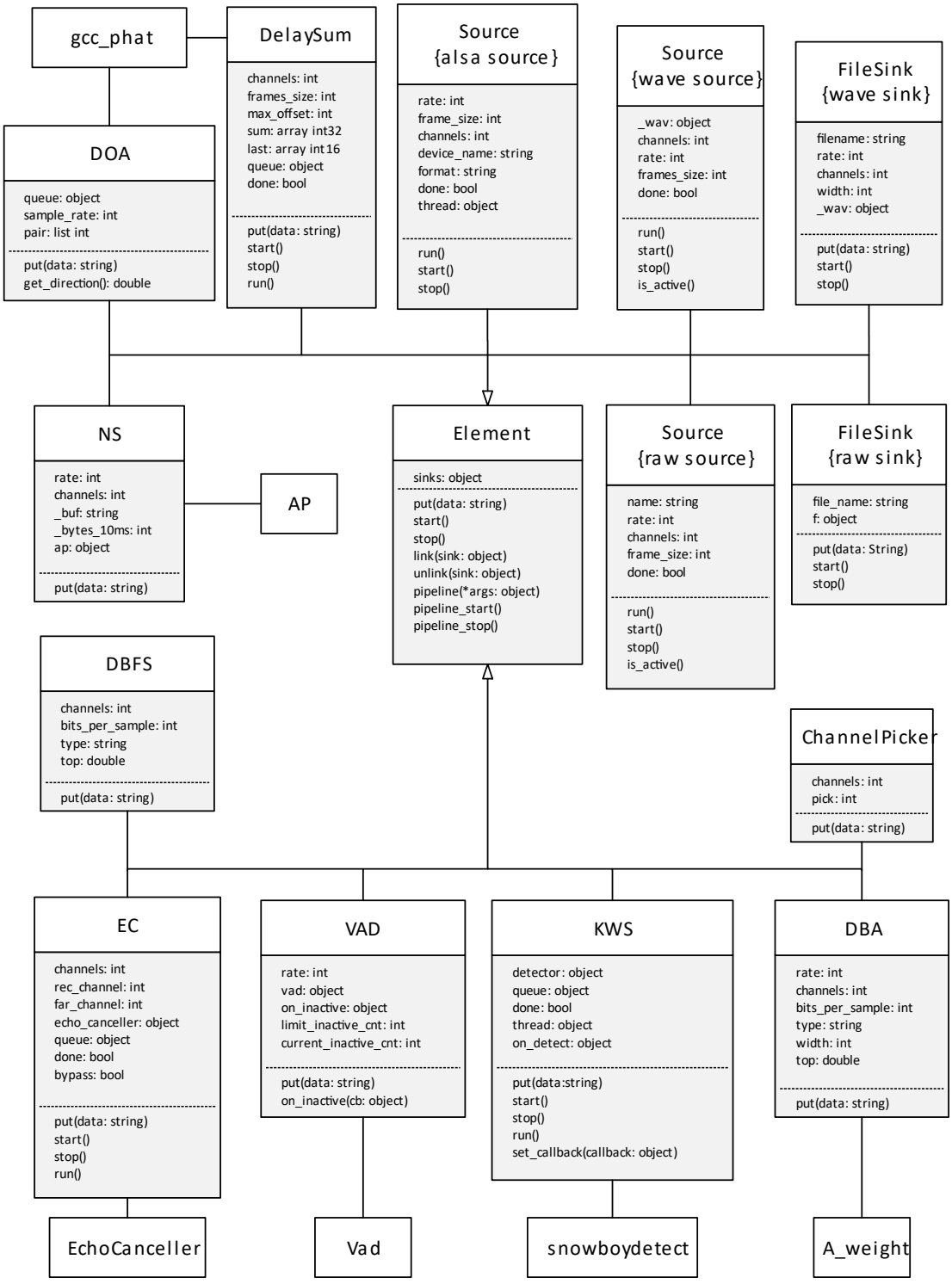


Abbildung A.2: Klassenübersicht des Voice-Engine Pakets

B Simulation

B.1 Verwendetes Sprachsignal

Der Wortlaut des Sprachsignals lautet:

„Das menschliche Gehör […] ist als Schallanalysator in Leistungsfähigkeit und Vielseitigkeit von technisch-analytischen Verfahren nach wie vor unerreicht.“ [35]

In Abbildung B.1 ist der Signalverlauf und das Spektrogramm des Sprachsignal dargestellt.

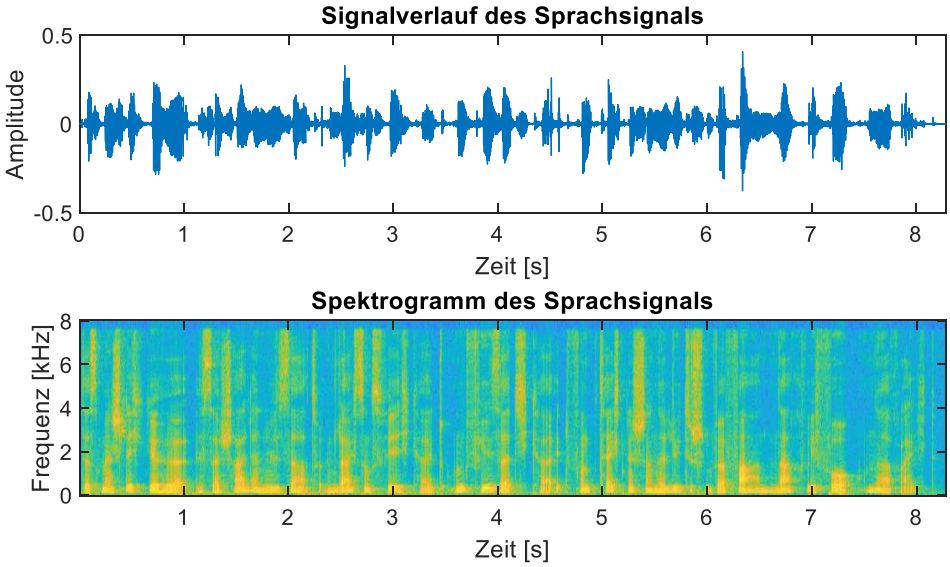


Abbildung B.1: Signalverlauf (oben) und Spektrogramm (unten) des Sprachsignals

B.2 Darstellung der Rauschunterdrückungsstufen

In Abbildung B.2 sind die Signalverläufe eines rauschunterdrückten Sprachsignals mit einem Signal-Rausch-Verhältnis von 14 dB bei verschiedenen Intensitätsstufen dargestellt.

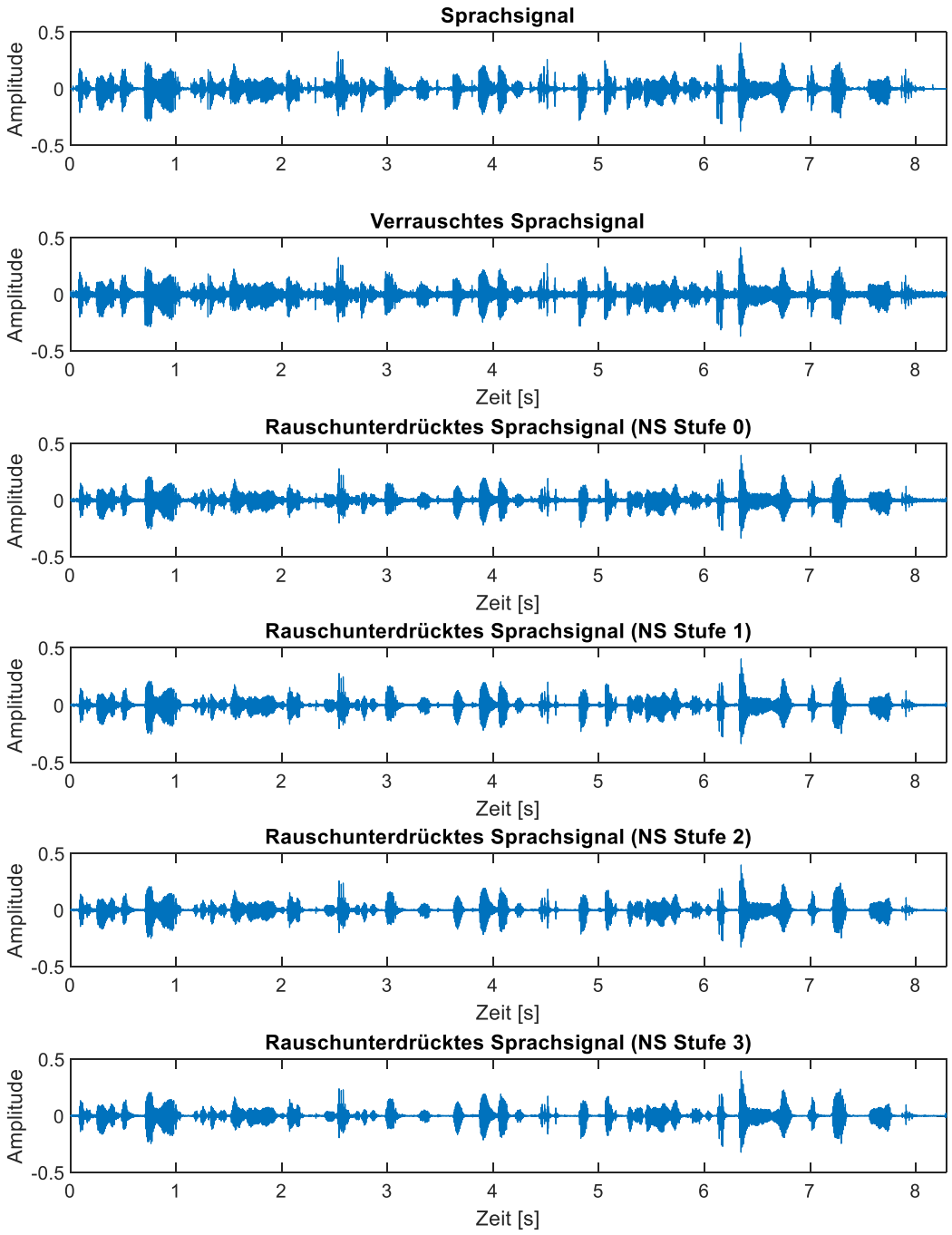


Abbildung B.2: Signalverlauf eines rauschunterdrückten Sprachsignals bei verschiedenen Intensitätsstufen

In Abbildung B.3 sind die Spektrogramme eines rauschunterdrückten Sprachsignals mit einem Signal-Rausch-Verhältnis von 14 dB bei verschiedenen Intensitätsstufen dargestellt.

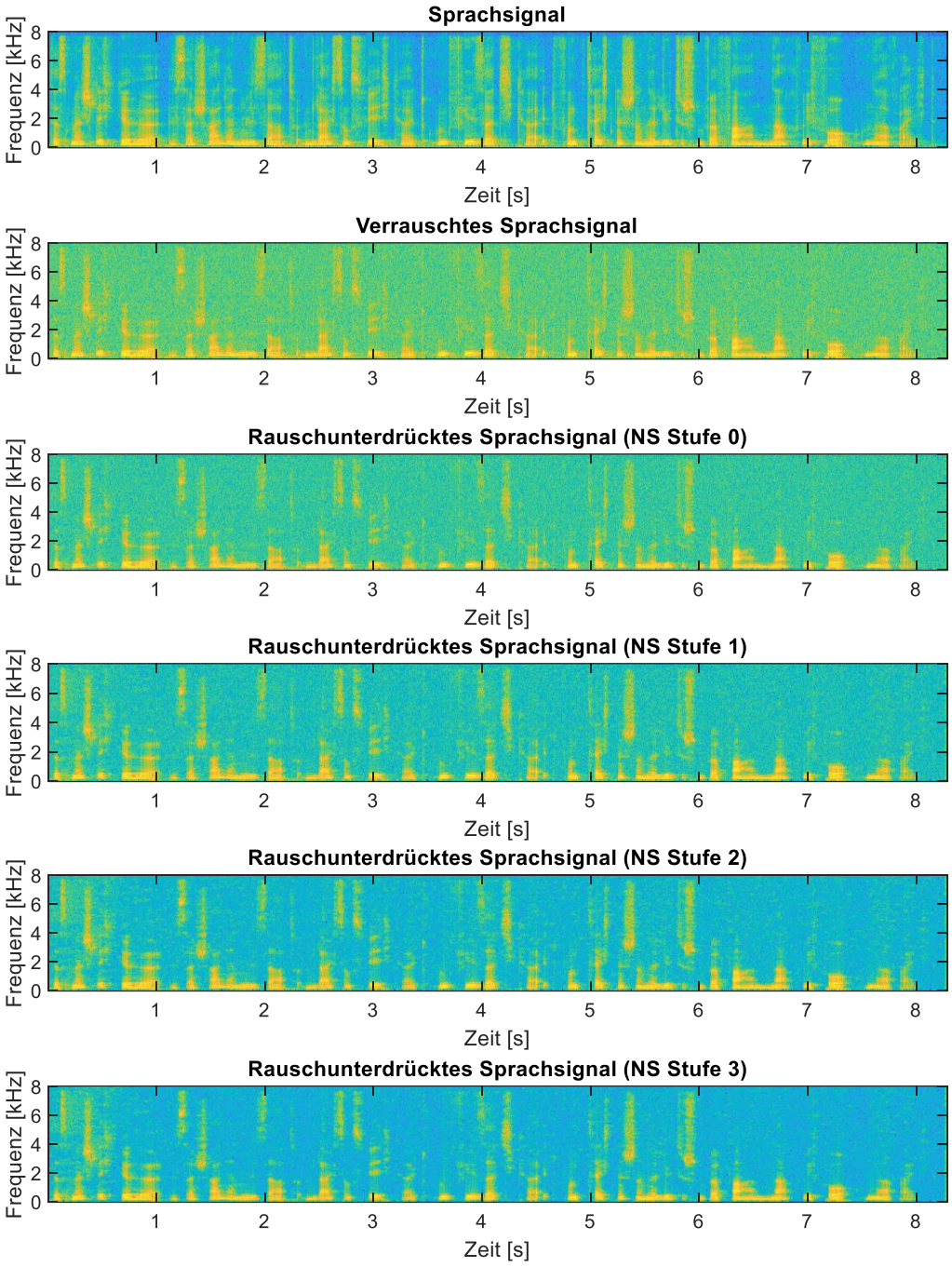


Abbildung B.3: Spektrogramm eines rauschunterdrückten Sprachsignals bei verschiedenen Intensitätsstufen

C Inhalt des Datenträges

Der Beiliegende Datenträger ist wie folgt strukturiert:

- **\Bachelorarbeit_ArturHeck.pdf**
  Dies ist die vorliegende Bachelorarbeit als PDF.
- **\MATLAB**
  Dieser Ordner enthält die MATLAB-Skripte, die in dieser Arbeit entwickelt und verwendet wurden. Der Ordner ist in zwei weitere Hauptgruppen unterteilt:
  - **Messergebnisse\:** Skripte zur Auswertung der Messergebnisse
  - **Simulationen\:** Skripte mit Simulationen und die eingebundenen Algorithmen
- **\Messprogramme**
  Dieser Ordner beinhaltet die in Python erstellten Messprogramme für den ReSpeaker Core v2.0.
- **\voice_engine**
  Dieser Ordner stellt das Voice-Engine Paket dar. Darin befinden sich die Python Skripte mit den Sprachverarbeitungsalgorithmen.

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Heck

Vorname: Artur

dass ich die vorliegende Bachelorarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Analyse und Test eines zirkularen 6-Mikrofon-Arrays von ReSpeaker

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Hamburg	18.06.2019	
Ort	Datum	Unterschrift im Original