Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Kolja Pikora

Korrelationsalgorithmen zur Sprecherlokalisierung mittels DSP-basiertem Echtzeitsystem und Mikrofonarray

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Kolja Pikora

Korrelationsalgorithmen zur Sprecherlokalisierung mittels DSP-basiertem Echtzeitsystem und Mikrofonarray

Masterthesis eingereicht im Rahmen der Masterprüfung
im Masterstudiengang Informations- und Kommunikationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ulrich Sauvagerd
Zweitgutachter : Prof. Dr.-Ing. Hans Peter Kölzer

Abgegeben am 30. August 2011

Kolja Pikora

Thema der Masterthesis

Korrelationsalgorithmen zur Sprecherlokalisierung mittels DSP-basiertem Echtzeitsystem und Mikrofonarray

Stichworte

MCCC, SLP, Mikrofonarray, DSP, Echtzeitsystem, variable Verzögerungskette, VDL,

Kurzzusammenfassung

In dieser Arbeit werden die Algorithmen Multichannel Crosscorrelation und Spatial Linear Prediction für eine Sprecherloaklisierung auf einem Signalprozessor untersucht. Die Lokalisierung erfolgt mit einem achtkanaligen Mikrofonarray, wahlweise in linearem oder zirkularem Aufbau. Desweiteren ist eine variable Verzögerungskette implementiert, mit der die Signale unverzögert zueinander aus dem Signalprozessor ausgegeben werden. Beide Algorithmen wurden in MATLAB simuliert und in Echtzeitumgebung getestet. Die Implementierung auf dem D.Module.C6713 erfolgt in der Programmiersprache C.

Kolja Pikora

Title of the paper

Correlationalgorithms for localizing a speech signal using a DSP based Realtimesystem and Microphone Array

Keywords

MCCC,SLP, Microphonearray, DSP, Realtime System, variable delayline, VDL

Abstract

In this masterthesis the two algorithms Multichannel Crosscorrelation and Spatial Linear Prediction are analyzed and applied in Speechlocalization on a Signal Processor. For the localization an eight channel microphone array is optionally used in linear or circular design. Moreover a variable delayline for an undelayed output of the microphonesignals is implemented. The algorithms were simulated in MATLAB and tested under real time conditions. The project is implemented on the d.module C6713 from TI in the programming language C and Assembler.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADU	Analog Digital Umsetzer
AIF	Anti-Imaging-Filter
AKF	Autokorrelationsfunktion
CCS	Code Composer Studio
CPU	Central Processing Unit
D.SignT	Digital Signalprocessing Technology
DAU	Digital Analog Umsetzer
DFT	Diskrete Fouriertransformation
DOA	Direction of Arrival
DSP	Digitaler Signalprozessor
EDMA	Enhanced Direct Memory Access
FFT	Fast Fouriertransformation
FIR	Finite Impulse Response
GJBF	Griffith-Jim Beamformer
IIR	Infinite Impulse Response
KKF	Kreuzkorrelationsfunktion
LSI	Linear Shiftinvariant
LTI	Linear Timeinvariant
McBSP	Multi-channel Buffered Serial Port
MCCC	Multichannel Crosscorrelation Coefficient
SDOA	Sample Difference of Arrival
SDRAM	Synchronous Dynamic Random Access Memory

SLP	Spatial Linear Prediction
SNR	Signalstörabstand
TDOA	Time Difference Of Arrival
UCA	Uniform Circular Array
UCYA	Uniform Cylindrical Array
ULA	Uniform Linear Array
VDL	Variable Delay Line

Symbolverzeichnis

Griechische Symbole

$\underline{\Phi}_{rs}(\cdot)$	Kreuzleistungsdichtespektrum
λ_{min}	Kleinste Wellenlänge, die im Signal auftritt
ϕ	Phasenverschiebung zwischen zwei Signalen
$\varphi_{rs}(\cdot)$	Kreuzkorrelationsfunktion
Ψ_n	Winkel eines Mikrofons n zum Referenzmikrofon
$\rho_{a,x,y}$	Korrelationskoeffizient zwischen Signal x und y
$\mathbf{\Sigma}$	Diagonalmatrix mit Standardabweichungen der Kanäle
σ_x	Standardabweichung des x-ten Signals
σ_x^2	Varianz des x-ten Signals
ς_θ	Streuung (Mittlerer quadratischer Fehler) eines Winkels
τ	Zeitverzögerung
θ	Schalleinfallswinkel am Mikrofonarray
α_n	Dämpfungsfaktor
$\upsilon_n(\cdot)$	Additives Rauschen am Eingang eines Mikrofons

Lateinische Buchstaben

$\mathbf{a}_{1:L}(\cdot)$	Vektor mit den Spatial Linear Prediction Koeffizienten
b_n	Filterkoeffizient eines FIR-Filters
c	Schallgeschwindigkeit bei 20°C, $c = 343\frac{m}{s}$
d	Abstand zwischen zwei Mikrofonen
$E[\cdot]$	Erwartungswertoperator
$e_0(\cdot)$	Prädiktionsfehler
$E(\cdot)$	Energie im Signal
f_a	Abtastfrequenz in Hz
f_{Ao}	Abtastfrequenz nach der Interpolation
f_{Takt}	Taktfrequenz des Prozessors

$F_n(\cdot)$	Funktion für die Laufzeitunterschiede
$H(z)$	Übertragungsfunktion eines Filters
$Im(\cdot)$	Imaginärteil einer komplexen Größe
$J(\cdot)$	Fehlerfunktion, mittlerer quadratischer Prädiktionsfehler
K	Größe für einen SDOA
L	Anzahl der Mikrofone in C-Zählweise
L_{EDMA}	Anzahl der vom EDMA aufgenommenen Daten
L_{FFT}	Länge der FFT
L_{sig}	Anzahl der im Algorithmus verwendeten Daten
M_n	Platzhalter für Mikrofon n
p	Angenommener TDOA
r	Radius
$\mathbf{R}_a(p)$	Kovarianzmatrix für einen Winkel p
$\tilde{\mathbf{R}}_a(p)$	Normierte Kovarianzmatrix für einen Winkel p
r_{a,y_i,y_j}	Kovarianz zwischen Signal i und j
$\mathbf{r}_{a,1:L}$	Vektor aus Kovarianzen
$Re(\cdot)$	Realteil einer komplexen Größe
$s(\cdot)$	Quellensignal
t	Zeitpunkt in Sekunden
T_{S,M_0}	Ausbreitungszeit von der Quelle zum Referenzmikrofon
$\underline{W}_x^k$	k$^{\text{ter}}$ Twiddlefaktor einer FFT der Länge x
$x_n(\cdot)$	Eingangssignal eines Mikrofons
$y_n(\cdot)$	Ausgangssignal eines Mikrofons

Spezielle Symbole

$*$	Faltungssymbol
$\det[\cdot]$	Determinante einer Matrix
$\mathcal{F}$	Fouriertransformierte eines Signals
x^{-1}	Inverser Wert eines Skalar oder einer Matrix
$\underline{x}^*$	konjugiert komplexe Größe
$max(\cdot)$	Maximum einer Funktion
$min(\cdot)$	Minimum einer Funktion
$\mathbf{x}^T$	Transponierte eines Vektors
$\underline{x}$	Komplexe Größe
$\hat{x}$	Schätzwert einer Größe

1. Einführung

In dieser Arbeit wird ein Digitaler Signalprozessor (DSP)-basiertes Echtzeitsystem entwickelt, mit dem die Ermittlung des relativen Winkels einer Sprachquelle zu einem Mikrofonarray, sowie eine variable Verzögerung der Kanäle möglich ist.
Mikrofonarrays werden in zahlreichen Anwendungen eingesetzt. So ist es möglich, mit einem Mikrofonarray Störquellen in einer Umgebung auszublenden oder eine Hall- und Echoreduzierung in einem Signal durchzuführen. Dies findet z. B. Anwendung in Freisprecheinrichungen der Mobiltelefone oder bei hochqualitativen Audioaufnahmen. Des Weiteren werden Mikrofonarrays dann eingesetzt, wenn zum Beispiel Quellen lokalisiert und deren Schallemission gemessen werden sollen. Für diesen Zweck wurden in den letzten Jahren zahlreiche Algorithmen entwickelt, die unabhängig vom genauen Ansatz stets das Ziel haben, die Richtung einer Quelle und gegebenenfalls deren Intensität zu ermitteln. Dabei besteht oftmals auch der Anspruch der Echtzeitfähigkeit. Insbesondere dann, wenn bewegliche Quellen detektiert werden müssen, soll eine hohe zeitliche Auflösung der Winkel gegeben sein. Noch anspruchsvoller wird es, wenn die Detektion ein Teil eines Gesamtsystems ist und die Daten nach der Verarbeitung an weitere Systemelemente weitergegeben werden müssen.
Das Ziel dieser Arbeit soll sein, ein Echtzeitsystem zu entwickeln, das neben der Quellendetektion auch eine variable Verzögerung und Weitergabe der Eingangsdaten ermöglicht.

1.1. Motivation und Hintergrund

Die Arbeit ist Teil eines Projektes an der Hochschule für Angewandte Wissenschaften Hamburg. In diesem Projekt wird ein Roboter mit Sprachsignalen gesteuert, die über ein Mikrofonarray - das auf dem Roboter befestigt ist - aufgenommen werden. Das Gesamtsystem besteht aus insgesamt vier kaskadierten DSP-Boards, die unterschiedliche Aufgaben übernehmen. Der in dieser Arbeit entwickelte Algorithmus hat die Aufgabe, die Daten für einen Rauschreduktionsalgorithmus aufzubereiten und außerdem den Winkel der Quelle an die Spracherkennung weiterzugeben.
Ein theoretischer Ansatz für die Berechnung des Winkels eines Quellsignals liegt in der Betrachtung der Laufzeitunterschiede der Mikrofonsignale bei einem Mikrofonarray. Die Laufzeitunterschiede sind für eine bekannte geometrische Anordnung der Mikrofone abhängig vom Winkel der Quelle. Die in dieser Arbeit verwendeten Methoden sind die Spatial

Linear Prediction (SLP) sowie die Berechnung des Multichannel Crosscorrelation Coefficient (MCCC). Beides sind Kreuzkorrelationsmethoden, welche die Laufzeitunterschiede über die Redundanz zwischen den Mikrofonsignalen ermitteln.

1.2. Aufbau der Arbeit

Das erste Kapitel dieser Arbeit enthält Erläuterungen über den Hintergrund sowie die Motivation für den Aufbau eines Systems zur Ortung eines Sprechers.
Anschließend wird im zweiten Kapitel eine Zielsetzung für die Arbeit vorgenommen und gezeigt, welche Bedingungen das System erfüllen muss und welcher Algorithmus für diesen Zweck geeignet scheint.
Im dritten Kapitel werden die theoretischen Grundlagen, die für das Verständnis der Arbeit vonnöten sind, erläutert.
Im vierten Kapitel wird der Aufbau des Echtzeitsystems und die dazugehörigen Komponenten erklärt.
Im fünften Kapitel werden allgemeine Umgebungsparameter und deren gegenseitige Beeinflussung diskutiert.
Das sechste Kapitel stellt anschließend die Simulation der Algorithmen und die Tests der verwendeten Methoden dar.
Das siebte Kapitel beinhaltet die Implementierung auf dem DSP.
Im achten Kapitel werden die Ergebnisse der Echtzeittests darstellt.
Im neunten Kapitel wird eine Zusammenfassung und ein Ausblick auf weitere Arbeiten gegeben.

2. Problemformulierung und Wahl der Algorithmen

Im Rahmen dieser Masterarbeit wird ein Echtzeitsystem entwickelt, dass neben der Detektion einer Sprachquelle eine variable Verzögerung (im Folgenden als Variable Delay Line (VDL) bezeichnet) der Kanäle ermöglichen soll. Aus dieser Aufgabenstellung und der im letzten Kapitel dargestellten Einordnung in das Gesamtprojekt können Ziele formuliert werden, welche der Algorithmus erfüllen muss.
Der Algorithmus muss

1. breitbandige Signale wie Sprache verarbeiten können,
2. die Richtung der Quelle mit einer hohen Genauigkeit liefern,
3. auch in verhallter und verrauschter Umgebung die Quelle detektieren und
4. echtzeitfähig sein.

Das erste Ziel kann durch die Auswahl geeigneter Algorithmen erfüllt werden. Ein theoretischer Ansatz für die Berechnung der Schalleinfallsrichtung - in der Literatur oft als Direction of Arrival (DOA) bezeichnet - eines Quellsignals liegt in der Betrachtung der Laufzeitunterschiede (Time Difference Of Arrival (TDOA)) der Mikrofonsignale in einem Mikrofonarray. Die TDOA können bei bekannter geometrischer Anordnung des Mikrofonarrays in Abhängigkeit des Winkels der Quelle formuliert werden.
Für die Berechnung eines DOA, also für die Lokalisierung einer Signalquelle gibt es zwei verschiedene Ansätze, die man im Allgemeinen in subraumbasierte Verfahren (subspace-Verfahren) und die räumliche Suche mittels Beamformer (beam forming techniques) unterteilen kann. Die subraumbasierten Verfahren haben den Vorteil, dass der DOA mit einer höheren Genauigkeit geliefert wird. Dafür sind sie aber weitaus komplexer, und benötigen einen höheren Rechenaufwand bei der Implementierung. Um in dieser Arbeit Echtzeitfähigkeit garantieren zu können, werden zwei Algorithmen aus dem Bereich der räumlichen Suche mittels Beamformer verwendet[1]. Diese sind zum einen die SLP zum anderen die Methode des MCCC. Die beiden Verfahren basieren auf der grundlegenden Idee der Kreuzkorrelation,

[1] Der Begriff Beamforming beschreibt das Ausrichten einer Sensorgruppe (hier eines Mikrofonarrays) auf die gewünschte Quelle

also der Erkennung von Redundanz zwischen den Mikrofonsignalen. Die beiden Algorithmen benötigen eine Anzahl von $N \geq 2$ Mikrofone. Der Aufbau des Mikrofonarrays ist beliebig und bestimmt lediglich den Rechenaufwand. Im Hinblick darauf, dass der Roboter später möglichst von allen Seiten aus gesteuert werden soll, wird in dieser Arbeit neben einem linearen Array auch ein Mikrofonarray in zylindrischem Aufbau verwendet.
Das zweite und dritte Ziel hängt mit der sog. blinden Quellentrennung zusammen. In natürlicher Umgebung treten Sprachsignale selten in reiner Form auf. Meist werden sie von Hintergrundgeräuschen oder Echos überlagert. Der Mensch überhört Geräusche mit kleinen Lärmpegeln[2] und kann sich in einem Stimmwirrwarr sogar gezielt auf einen Gesprächspartner konzentrieren. Dieser sog. Cocktailparty-Effekt wirkt sich jedoch negativ auf die Sprachsignalverarbeitung aus, da die bekannten Algorithmen störungsfrei sehr gute Ergebnisse liefern, in hallenden und rauschenden Umgebungen dagegen hohe Fehlerraten haben. Durch geeignete Einstellungen von Umgebungsparametern der verwendeten Algorithmen können die Fehlerraten eingedämmt werden.
Der vierte Ziel, Echtzeitfähigkeit zu gewährleisten, stellt in sich weitere Bedingungen auf, die das System erfüllen muss. Diese Bedingungen ergeben sich aus der Definition für „Echtzeit“

> „Ein Programm rechnet in Echzeit, wenn es Eingabedaten so schnell verarbeitet, wie der Vorgang, der sie erzeugt.“ [Köl10]

sowie aus der Definition des Begriffs „Echtzeitsystem“

> „[...] Betrieb eines Rechnersystems [...], bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind.“ [WB05]

Auf diese Anwendung bezogen, können somit folgende Forderungen definiert werden:

- Die Verarbeitung der Daten darf lediglich so lange dauern, wie der Eingang benötigt, um neue Daten aufzunehmen $\Rightarrow$ *Rechtzeitigkeit*,
- die Aufnahme der Daten, sowie die Verarbeitung muss parallel vonstatten gehen $\Rightarrow$ *Gleichzeitigkeit*,
- es muss einen kontinuierlichen Datenfluss zwischen Ein- und Ausgangsschnittstelle geben $\Rightarrow$ *Kontinuität*,
- im Hinblick auf die spätere Sprachsignalerkennung dürfen keine Daten ungenutzt überschrieben werden $\Rightarrow$ *Zuverlässigkeit*.

Die Einhaltung der vier Grundziele soll bei der Entwicklung dieser Arbeit im Mittelpunkt stehen und eine ordnungsgemäße Funktion im Gesamtsystem gewährleisten.

[2] Dies wird auch als selektive Wahrnehmung bezeichnet.

3. Mathematische Grundlagen

Die Abschätzung der Laufzeitunterschiede - TDOA - erfolgt über die Messung der Zeitdifferenz eines Signals zwischen verschiedenen Mikrofonen eines Mikrofonarrays. In dieser Arbeit werden zwei Mikrofonarrays verschiedener Anordnung verwendet, die dargestellt werden. Aus den Darstellungen werden die für den Algorithmus relevanten mathematischen Eigenschaften der Mikrofonarrays erläutert. Anschließend wird eine Modellierung der Mikrofoneingangssignale vorgenommen, mit deren Grundlage die zu untersuchenden Algorithmen hergeleitet werden. Diese Herleitungen sind hauptsächlich an [BCH08] angelehnt. Abschließend werden eine effektive Methode zur Berechnung von Kreuzkorrelationen vorgestellt und - für diese Arbeit relevante - statistische Eigenschaften von Sprachsignalen erläutert.
Für die Darstellung numerischer Größen werden an dieser Stelle folgende Vereinbarungen getroffen:

- Matritzen werden durch Großbuchstaben und fett gekennzeichnet,
- Vektoren werden durch Kleinbuchstaben und fett gekennzeichnet,
- komplexe Größen werden durch Unterstriche markiert,
- alle anderen Größen sind skalar.

Des Weiteren soll an dieser Stelle die Vereinbarung vorgenommen werden, dass die Nummerierung der Elemente in Vektoren und Matritzen an die Implementierung in C angelehnt sind. Das heißt, die Zählweise der Mikrofone beginnt bei 0 und endet bei $L = N - 1$.

3.1. Aufbau der Mikrofonarrays

Mikrofonarrays können je nach Anwendungsgebiet in verschiedenen geometrischen Anordnungen aufgebaut werden. Da in dieser Arbeit lediglich der Azimutwinkel relativ zum Array errechnet werden soll, werden nur zweidimensionale Anordnungen verwendet. Es wird angenommen, dass eine reflexionsarme Umgebung vorliegt, in der nur die direkten Pfade an den Mikrofonen ankommen. Auf den Einfluss eventueller Reflexionen wird in den Herleitungen nicht eingegangen.

Als Audruck für die Laufzeitunterschiede in den Mikrofonarrays wird die Funktion $F_n(\tau)$ eingeführt, deren mathematische Formulierung aus jeder beliebigen Struktur eines Mikrofonarrays hergeleitet werden kann.

3.1.1. Lineares Mikrofonarray

Abbildung 3.1 zeigt eine Skizze eines linearen äquidistanten Mikrofonarrays (Uniform Linear Array (ULA)) mit einer Anzahl von N Mikrofonen und einer einzelnen Signalquelle S.

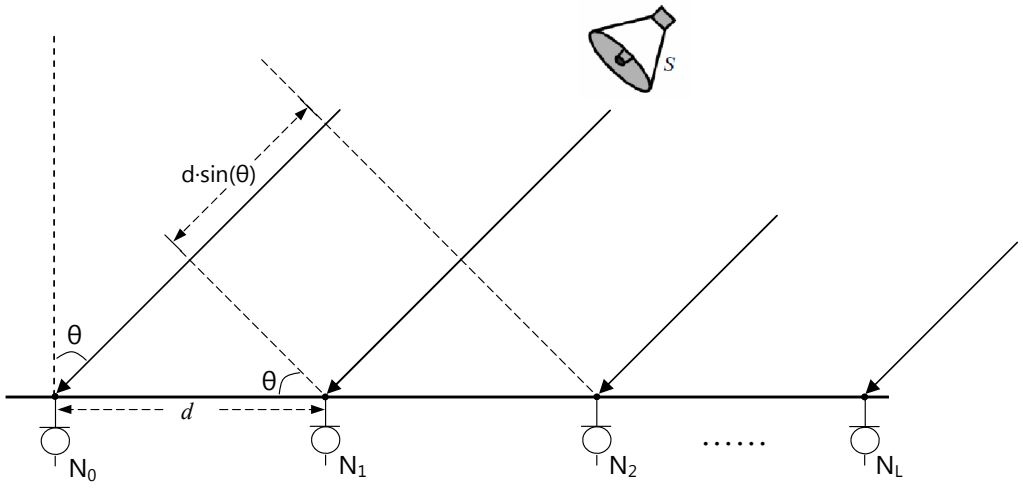


Abbildung 3.1.: Lineares, Äquidistantes Mikrofonarray (ULA)

Unter der Voraussetzung, dass sich die Quelle im Fernfeld befindet, kann als Vereinfachung angenommen werden, dass die einfallenden Schallwellenfronten parallel und quasistationär sind[3]. Somit müssen die Schallwellen ausgehend von der Signalquelle unterschiedliche Abstände zu den einzelnen Mikrofonen zurücklegen. Der Zeitunterschied identischer Eingangssignale zwischen zwei benachbarten Mikrofonen wird in einem ULA als Laufzeitdifferenz τ_{ULA} bezeichnet. Sie ergibt sich über die Formel

$$\tau_{ULA} = \frac{d \cdot \sin(\theta)}{c}. \tag{3.1}$$

Dabei ist $c \approx 343\,m/s$ die Schallgeschwindigkeit[4], d der Abstand zwischen einem Mikrofonpaar und θ der Einfallswinkel. Für die Laufzeitdifferenz zwischen dem Referenzmikrofon M_0

[3] Diese Annahme ist in der Literatur als Fernfeldbedingung bekannt.
[4] Diese Approximation ergibt sich für eine Raumtemperatur von 20°C.

und Mikrofon M_n ergibt sich $F_{n,ULA}(\tau)$ als lineare Funktion:

$$F_{n,ULA}(\tau) = n \cdot \tau_{ULA},\ n = 0, 1, \ldots, L. \tag{3.2}$$

3.1.2. Zirkulares Mikrofonarray

Ein lineares Array kann abhängig von der Aufstellungsrichtung lediglich einen Winkel von 180° detektieren. Ist es jedoch notwendig, eine Quelle im gesamten Umkreis von 360° zu lokalisieren, so muss auf einen zirkularen Aufbau (Uniform Circular Array (UCA)) zurückgegriffen werden (vgl. Abb. 3.2).

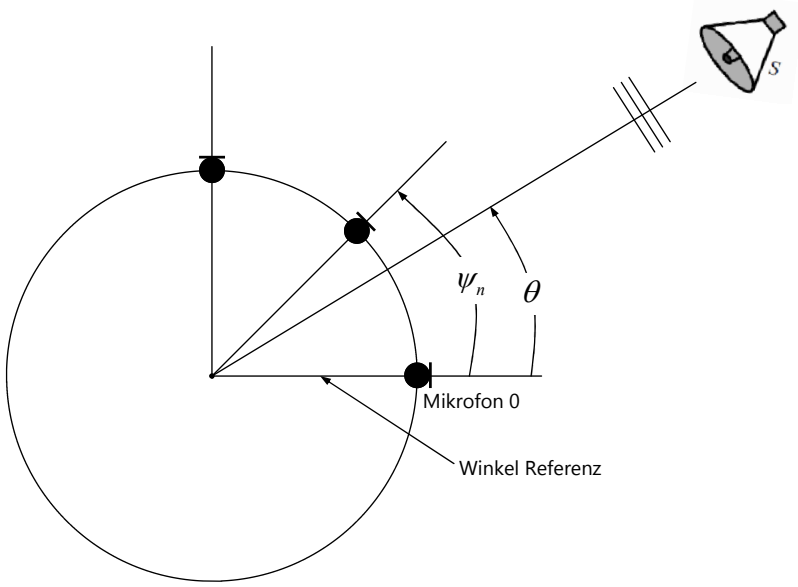


Abbildung 3.2.: Zirkulares, Äquidistantes Mikrofonarray (UCA)

Die Zeitverzögerung τ_{UCA} des ankommenden Signals ergibt sich aus dem Zusammenhang zwischen dem Winkel des Arrays zur Quelle und den relativen Verzögerungen zwischen den Mikrofonen M_0 und M_n [DB07]. Unter Einhaltung der Fernfeldbedingung ist die zeitliche Verzögerung des ankommenden Signals zwischen einem Mikrofon M_n und dem Mittelpunkt des Arrays über die Formel

$$\tau_{n,UCA}(\theta) = \frac{r}{c} \cdot \cos(\theta - \psi_n), mit\ n = 0, 1, \ldots, L \tag{3.3}$$

gegeben, in der r der Radius des Mikrofonarrays ist und ψ_n der Winkel eines Mikrofons M_n relativ zum Referenzmikrofon M_0. ψ_n ist unter Äquidistanz der zirkularen Anordnung gegeben durch

$$\psi_n = \frac{2\pi n}{N}, \, mit \; n = 1, 2, \ldots, L. \tag{3.4}$$

Die Funktion $F_{n,UCA}$ folgt abschließend aus den obigen Betrachtungen als eine relative Zeitverzögerung

$$F_{n,UCA}(\tau) = \tau_{0,UCA}(\theta) \; - \; \tau_{n,UCA}(\theta) = \frac{r}{c} \cdot \left[\cos\theta \; - \; \cos\left(\theta \; - \; \frac{2\pi n}{N} \right) \right]. \tag{3.5}$$

3.1.3. Zylindrisches Mikrofonarray

Lineare und zirkulare Mikrofonarrays haben den Nachteil, dass das Signal der Quelle an jedem Mikrofon etwa gleichmäßig gedämpft vorliegt. Dies resultiert aus den relativ geringen Ausmaßen der Arrays. Die geometrische Form eines Zylinders, an dem die Mikrofone außen in einer Reihe angebracht sind, kann für eine genauere Annäherung von τ ausgenutzt werden. Die Signale an der Rückseite des Zylinders sind gedämpfter und werden somit bei der Annäherung des TDOA keinen großen Einfluss haben.
Prinzipiell kann das zylindrische Modell (Uniform Cylindrical Array (UCYA)) exakt so hergeleitet werden wie das zirkulare Modell, da die Mikrofone hier um 90° gekippt sind. Dadurch beschreibt im Gegensatz zum zirkularen Array der Elevationswinkel den TDOA. Mathematisch stimmen der Azimutwinkel im zirkularen und der Elevationswinkel im zylindrischen Array bei gegebener Äquidistanz überein.
In dieser Arbeit werden zwei Mikrofonarrays verwendet. Das eine in linearer, das andere in zylindrischer Form.

3.1.4. Räumlicher Alias-Effekt

Ein Problem bei der Konstruktion vom Mikrofonarrays ist die Dimensionierung des Abstands d zwischen den Mikrofonen. Der erste Gedanke dabei ist, den Abstand möglichst groß zu machen, damit eine möglichst große Zeitverzögerung zwischen den Mikrofonen entsteht. Allerdings tritt ab einem bestimmten Abstand der sog. Räumliche Alias-Effekt auf, der die Annäherung des TDOA maßgeblich beeinflusst.
Wie oben erwähnt, wird der TDOA über die Messung der Zeitdifferenz zwischen zwei oder mehr Mikrofonen angenähert. Er ist abhängig vom Abstand d zwischen den Mikrofonen. Dies entspricht im Spektralbereich einer Phasenverschiebung der Signale. Die Phasenverschiebung ϕ sollte nicht größer werden als π, da dies nicht zu unterscheiden ist von einer Verschiebung von $2\pi - \phi$ und umgekehrt. Räumliches Aliasing tritt auf, wenn die Verzögerung größer als π wird. Dann könnten Signale, die zu zwei Winkeln θ_1 und $\theta_2 = 2\pi - \theta_1$

aufgenommen werden, dasselbe Ergebnis liefern. Die Bedingung zum Verhindern von räumlichen Aliasing ist damit gegeben durch

$$2\pi f_{max}\tau \leq \pi. \tag{3.6}$$

f_{max} entspricht der größten, relevanten, im Signalspektrum enthaltenden Frequenz[5]. Für ein ULA ergibt sich durch Einsetzen der Gleichung 3.1 und Umstellen nach d Gleichung 3.7

$$\begin{aligned} d &\leq \frac{1}{2}\left(\frac{c}{f_{max}\sin(\theta)}\right) \\ &\leq \frac{1}{2}\left(\frac{\lambda_{min}}{\sin(\theta)}\right), \end{aligned} \tag{3.7}$$

in der λ_{min} die kleinste Wellenlänge ist, die im Signal auftritt. Im Fall des größtmöglichen Winkels - bei einem linearen Array 90° - ergibt sich

$$d \leq \frac{\lambda_{min}}{2}. \tag{3.8}$$

Dieser Wert wird auch in der Literatur stets als Richtwert für die Dimensionierung des Abstandes angegeben (z. B. [BCH08, S.190]).
Bei gleicher Vorgehensweise für ein UCA bzw. UCYA ergibt sich nach Einsetzen von $\tau_{n,UCA}$ aus Gleichung 3.5 in Gleichung 3.6 ein Radius des Arrays von

$$r \leq \frac{\lambda_{min}}{2 \cdot \left[\cos\theta - \cos\left(\theta - \frac{2\pi n}{N}\right)\right]} \tag{3.9}$$

und daraus über den Umfang des Array $N \cdot d = 2\pi r$ der Abstand zwischen zwei Mikrofonen

$$d \leq \frac{\pi \cdot \lambda_{min}}{N \cdot \left[\cos\theta - \cos\left(\theta - \frac{2\pi}{N}\right)\right]}. \tag{3.10}$$

Auch hier kann ein maximaler Winkel zwischen zwei Mikrofonen von 90° eingesetzt werden:

[5] Bei Sprache wäre dies 3.4kHz, die Frequenz, die eine ausreichende Verständlichkeit der Sprache garantiert.

$$d \leq \frac{\lambda_{min}}{\sqrt{2}} \cdot \frac{2\pi}{N}. \tag{3.11}$$

Annäherungsweise kann der Abstand in einem UCA auch mit der Gleichung 3.8 für das ULA dimensioniert werden. Für ein UCYA muss jedoch diese Gleichung verwendet werden, da sonst die Krümmung des Arrays nicht berücksichtigt wird.

3.2. Signalmodell

Um die beiden Algorithmen SLP und MCCC in den nachfolgenden Abschnitten zu erschließen, ist es sinnvoll, an dieser Stelle ein geeignetes Signalmodell zu definieren und damit die Entstehung der Eingangssignale herzuleiten. Bei der Definition des Signals wird außerdem angenommen, dass sich genau eine Quelle in der Umgebung befindet, die lokalisiert werden soll. Auf den Fall einer Mehrfachquellenumgebung wird nicht eingegangen. Weiterhin wird eine reflexionsarme Umgebung betrachtet.
Wird das Mikrofon M_0 als Referenzmikrofon angenommen, so kann das Signal $y_n(t)$, das am n$^{\text{ten}}$ Mikrofon zum Zeitpunkt t aufgenommen wird, folgendermaßen ausgedrückt werden:

$$y_n(t) = x_n(t) + v_n(t), \tag{3.12}$$

wobei $x_n(t)$ das Eingangssignal am n$^{\text{ten}}$ Mikrofon ist und $v_n(t)$ das additive Rauschen, das als unkorreliert zum Signal angenommen wird. $x_n(t)$ kann durch das unbekannte Signal $s(t)$ der Quelle wie folgt beschrieben werden:

$$x_n(t) = \alpha_n \cdot s(t - T_{S,M_0} - F_n(\tau)). \tag{3.13}$$

Dabei ist α_n der Dämpfungsfaktor, hervorgerufen durch die Ausbreitung der Wellen und T_{S,M_0} die Ausbreitungszeit von der Quelle zum Referenzmikrofon. Gleichung 3.13 in Gleichung 3.12 eingesetzt, ergibt das Signalmodell

$$y_n(t) = \alpha_n \cdot s(t - T_{S,M_0} - F_n(\tau)) + v_n(t), \tag{3.14}$$

das den Herleitungen der Algorithmen zugrunde gelegt wird.

3.3. Methode der Spatial Linear Prediction

Bei der Spatial Linear Prediction (SLP) - zu übersetzen etwa mit 'Räumliche Lineare Schätzung' - wird die grundlegende Idee der Kreuzkorrelation - also der Erkennung von Redundanz zwischen den Mikrofonen - verfolgt. Generell funktioniert die SLP mit einer Anzahl von $N > 2$ Mikrofonen. Dabei steigt die Qualität der TDOA-Erkennung mit der Anzahl der Mikrofone [DB07], da der SLP-Algorithmus die Redundanzinformationen vieler Mikrofone zur räumlichen Schätzung ausnutzt [BCH08].
Befindet sich die Quelle im Fernfeld und werden die Rauschanteile vernachlässigt, so kann aus Formel 3.14 das Signal y_n als

$$y_n \{t + F_n(\tau)\} = \alpha_n \cdot s(t - T_{S,M_0}) \tag{3.15}$$

dargestellt werden. Folglich ist das Signal $y_0(t)$ in Phase mit $y_n \{t + F_n(\tau)\}$, wenn das richtige τ gewählt wird. Wird das Referenzmikrofon M_0 betrachtet, so wird nun versucht, die Abtastwerte dieses Mikrofons schrittweise mit den Abtastwerten der anderen L Mikrofone abzugleichen (in den Gleichungen soll dieser Abgleich im Folgenden mit dem Index a verdeutlicht werden). Ausgehend von dieser Betrachtung scheint es geeignet, ein Fehlersignal zu skizzieren, das genau diese Vorgehensweise mathematisch formuliert. Der Prädiktionsfehler ist gegeben durch

$$e_0(t, p) = y_0(t) - \mathbf{y}_{a,1:L}^T(t, p)\, \mathbf{a}_{1:L}(p). \tag{3.16}$$

Darin ist $p \in [\tau_{min}, \tau_{max}]$ eine Variable für den angenommenenen TDOA τ, mit welcher der Raum quasi nach der Quelle abgetastet wird,

$$\mathbf{y}_{a,1:L}(t, p) = [y_1 \{t + F_1(p)\} \cdots y_L \{t + F_L(p)\}]^T \tag{3.17}$$

der mit dem jeweiligen p verschobene Signalvektor und

$$\mathbf{a}_{1:L}(p) = [a_1(p)\, a_2(p) \cdots a_L(p)]^T \tag{3.18}$$

ein Vektor, der die sogenannten 'Forward Spatial Linear Prediction'-Koeffizienten beinhaltet[6] und eine Gewichtung des verschobenen Signals darstellt. Im Idealfall wird Gleichung 3.16 genau dann gleich null, wenn das richtige p gefunden ist. Liegt jedoch Rauschen vor, so

[6] Wird das Mikrofon M_0 als Referenzmikrofon ausgewählt, spricht man von 'Forward'-SLP. Ist M_L das Referenzmikrofon, so wird der Begriff 'Backward'-SLP verwendet.

kann Gleichung 3.16 lediglich minimiert werden. $J_0(p)$ sei der mittlere quadratische Prädiktionsfehler

$$J_0(p) = E\left[e_0^2(t,p)\right], \tag{3.19}$$

dessen Minimierung zu dem linearen System

$$\mathbf{R}_{a,1:L}(p)\,\mathbf{a}_{1:L}(p) = \mathbf{r}_{a,1:L}(p) \tag{3.20}$$

führt ($E[\cdot]$ ist der Erwartungswertoperator). Diese Gleichung stellt die räumliche Korrelationsmatrix zwischen den Kanälen dar, die durch

$$\mathbf{R}_{a,1:L}(p) = E\left[\mathbf{y}_{a,1:L}(t,p)\,\mathbf{y}_{a,1:L}^T(t,p)\right] \tag{3.21}$$

$$= \begin{bmatrix} \sigma_{y_1}^2 & r_{a,y_1y_2}(p) & \cdots & r_{a,y_1y_L}(p) \\ r_{a,y_2y_1}(p) & \sigma_{y_2}^2 & \cdots & r_{a,y_2y_L}(p) \\ \vdots & \vdots & \ddots & \vdots \\ r_{a,y_Ly_1}(p) & r_{a,y_Ly_2}(p) & \cdots & \sigma_{y_L}^2 \end{bmatrix} \tag{3.22}$$

gegeben ist und auf der Hauptdiagonalen die Varianzen der Kanäle

$$\sigma_{y_n}^2 = E\left[y_n^2(t)\right]$$

und in den weiteren Elementen die Kovarianzen zwischen den Mikrofonsignalen y_1 bis y_L

$$r_{a,y_iy_j}(p) = E\left[y_i\left\{t + F_i(p)\right\} y_j\left\{t + F_j(p)\right\}\right]$$

beinhaltet. Des Weiteren enthält das System aus 3.20 einen Vektor mit den Kovarianzen zwischen dem Referenzmikrofon M_0 und den anderen Mikrofonen

$$\mathbf{r}_{a,1:L} = \left[r_{a,y_0y_1}(p)\, r_{a,y_0y_2}(p) \cdots r_{a,y_0y_L}(p)\right]^T.$$

Die unbekannten Koeffizienten $\mathbf{a}_{1:L}(p)$ in Gleichung 3.16 können nun durch Umstellen der Gleichung 3.20 in

$$\mathbf{a}_{1:L}(p) = \mathbf{R}_{a,1:L}^{-1}(p)\,\mathbf{r}_{a,1:L}(p) \tag{3.23}$$

ersetzt werden. An dieser Stelle ist es wichtig noch einmal zu erwähnen, dass bisher eine rauschfreie Umgebung vorausgesezt wurde. Wenn kein Rauschen vorliegt, besitzt die Matrix eine Toeplitz-Form, solange $p \neq \tau$ ist. Für $p = \tau$ ist die Matrix aber nicht mehr invertierbar, da alle Spalten gleich sind. Die Matrix wird somit den Rang 1 haben. Daher kann ein Abtastwert an M_0 aus jedem anderen Abtastwert eines Mikrofones vorhergesagt werden. In der Praxis ist das Rauschen jedoch niemals null. Die Varianzen der verschiedenen Rauschsignale an den Mikrofonen werden zu den Diagonalen der Matrix hinzuaddiert, wodurch die Matrix regulär und damit invertierbar wird.
Es entsteht ein neuer Ausdruck für den Prädiktionsfehler:

$$e_0(t,p) = y_0(t) - \mathbf{y}_{a,1:L}^T(t,p)\,\mathbf{R}_{a,1:L}^{-1}(p)\,\mathbf{r}_{a,1:L}(p)\,. \tag{3.24}$$

Aufgreifen der oben genannten Idee der Minimierung des Fehlers führt zu

$$\begin{aligned} J_{0,min}(p) &= E\left[e_{0,min}^2(t,p)\right] \\ &= \sigma_{y_0}^2 - \mathbf{r}_{a,1:L}^T(p)\,\mathbf{R}_{a,1:L}^{-1}(p)\,\mathbf{r}_{a,1:L}(p)\,. \end{aligned} \tag{3.25}$$

Aus Gleichung 3.25 folgt, dass lediglich der Funktionsparameter p den Verlauf der Funktion $J_{1,min}$ bestimmt. Somit kann argumentiert werden, dass das p, welches ein Minimum der Funktion $J_{1,min}$ hervorruft, den angenäherten TDOA zwischen den Mikrofonen und damit die Richtung der Quelle repräsentiert:

$$\widehat{\tau}^{SLP} = \arg\min_p J_{0,min}(p)\,. \tag{3.26}$$

Abbildung 3.3 zeigt den schematischen Ablauf der SLP.

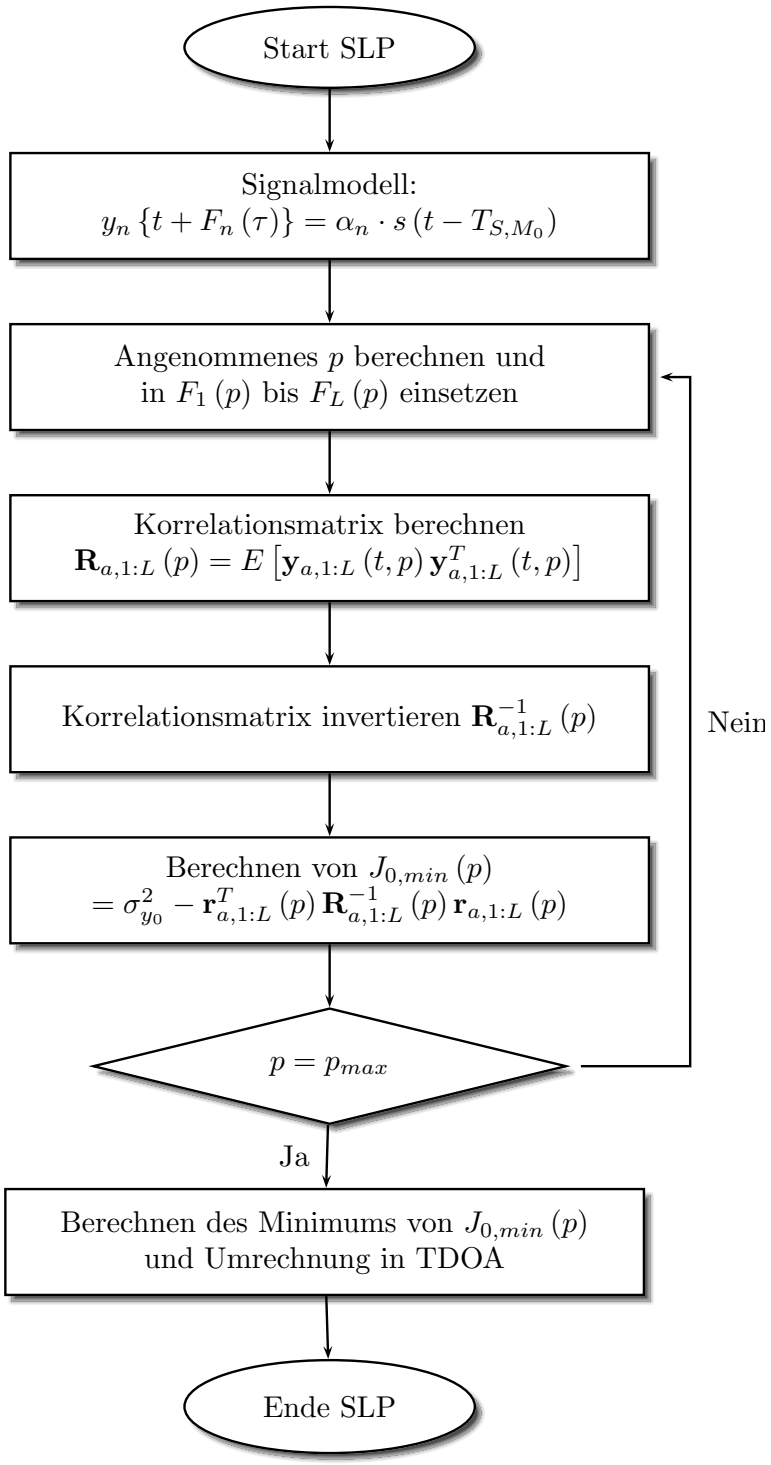


Abbildung 3.3.: Ablauf der Spatial Linear Prediction

3.4. Methode des Multichannel Cross Correlation Coefficients

Wie im letzten Abschnitt gesehen, ist die Bildung der räumlichen Korrelationsmatrix das zentrale Element der räumlichen Suche mittels Beamformer. Ein weiterer Ansatz die Korrelationsmatrix zu nutzen, um den TDOA einer Quelle zu ermitteln, liefert die Berechnung des sog. Multichannel Cross Correlation Coefficients (MCCC), etwa mit 'Mehrkanal Kreuzkorrelationskoeffizient' zu übersetzen.
Für die Herleitung des MCCC wird in Anlehnung an Gleichung 3.17 ein neuer Signalvektor definiert:

$$\mathbf{y}_a(t,p) = \left[y_0(t)\ y_1\{t+F_1(p)\}\ \cdots\ y_L\{t+F_L(p)\}\right]^T. \tag{3.27}$$

Ausgehend von diesem Signalvektor wird die dazugehörige Räumliche Korrelationsmatrix definiert:

$$\mathbf{R}_a(p) = E\left[\mathbf{y}_a(t,p)\,\mathbf{y}_a^T(t,p)\right] \tag{3.28}$$

$$= \begin{bmatrix} \sigma_{y_0}^2 & r_{a,y_0y_1}(p) & \cdots & r_{a,y_0y_L}(p) \\ r_{a,y_1y_0}(p) & \sigma_{y_1}^2 & \cdots & r_{a,y_1y_L}(p) \\ \vdots & \vdots & \ddots & \vdots \\ r_{a,y_Ly_0}(p) & r_{a,y_Ly_1}(p) & \cdots & \sigma_{y_L}^2 \end{bmatrix}. \tag{3.29}$$

Auf der Hauptdiagonalen der Matrix befinden sich die Varianzen der einzelnen Kanäle, die Nebendiagonalen beinhalten wieder die Kovarianzen $r_{a,y_iy_j}(p)$ zwischen Mikrofonsignal i und j. Zu beachten ist, dass dieser Ansatz im Gegensatz zur SLP alle N Mikrofone einbezieht. Die Varianzen können aus der Matrix herausgezogen werden, sodass folgende Faktorisierung der Matrix entsteht:

$$\mathbf{R}_a(p) = \mathbf{\Sigma}\tilde{\mathbf{R}}_a(p)\,\mathbf{\Sigma}. \tag{3.30}$$

Dabei ist

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{y_0} & 0 & \cdots & 0 \\ 0 & \sigma_{y_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ 0 & \cdots & 0 & \sigma_{y_L} \end{bmatrix}$$

eine Diagonalmatrix, welche die Standardabweichungen der Kanäle beinhaltet und

$$\tilde{\mathbf{R}}_a(p) = \begin{bmatrix} 1 & \rho_{a,y_0y_1}(p) & \cdots & \rho_{a,y_0y_L}(p) \\ \rho_{a,y_1y_0}(p) & 1 & \cdots & \rho_{a,y_1y_L}(p) \\ \vdots & \vdots & \ddots & \cdots \\ \rho_{a,y_Ly_0}(p) & \rho_{a,y_Ly_1}(p) & \cdots & 1 \end{bmatrix}$$

die normalisierte räumliche Korrelationsmatrix. Diese ist symmetrisch und beinhaltet auf den Nebendiagonalen die Korrelationskoeffizienten zwischen den Signalen am i$^{\text{ten}}$ und j$^{\text{ten}}$ Mikrofon, definiert durch:

$$\rho_{a,y_iy_j}(p) = \frac{r_{a,y_iy_j}(p)}{\sigma_{y_i}\sigma_{y_j}},\ i,j = 0,1,\ldots,L.$$

Der Korrelationskoeffizient zwischen zwei Signalen y_i und y_j hat die nachstehenden Eigenschaften, die für die weiteren Betrachtungen wichtig sind:

1. Sind y_i und y_j unkorreliert, so ist $\rho_{a,y_iy_j}(p) = 0$, es besteht keine Abhängigkeit zwischen den Signalen.
2. Sind y_i und y_j identisch, so ist $\rho_{a,y_iy_j}(p) = 1$, es besteht ein vollständig positiver Zusammenhang zwischen den Signalen.
3. Gilt $y_i = -y_j$, so sind beide Signale vollständig negativ korreliert und es gilt $\rho_{a,y_iy_j}(p) = -1$.

Der dritte Fall kann unter den getroffenen Annahmen nicht eintreten. Die Korrelationseinträge der Matrix $\tilde{\mathbf{R}}_a(p)$ nehmen damit stets Werte zwischen 0 und 1 ein, womit sichergestellt ist, dass die Matrix positiv semi-definit ist. Damit geht einher, dass die Eigenwerte der Matrix und damit auch die Determinante ≥ 0 sind. Genauer:

$$0 \leq \det\left[\tilde{\mathbf{R}}_a(p)\right] \leq 1. \tag{3.31}$$

Für den Fall, dass genau 2 Mikfrofone vorliegen: $N = 2$, so kann die Determinante von $\tilde{R}_a(p)$ sehr einfach berechnet werden:

$$\begin{aligned} \det\left[\tilde{\mathbf{R}}_a(p)\right] &= \begin{vmatrix} 1 & \rho_{a,y_0y_1}(p) \\ \rho_{a,y_1y_0}(p) & 1 \end{vmatrix} \\ &= 1 - \rho^2_{a,y_1y_0}(p). \end{aligned}$$

Hieraus ergibt sich durch Umstellen der Funktion der quadratische Korrelationskoeffizient

$$\rho^2_{a,y_0y_1}(p) = 1 - \det\left[\tilde{\mathbf{R}}_a(p)\right]. \tag{3.32}$$

In Analogie dazu ist der quadratische MCCC für N Mikrofone definert als

$$\begin{aligned}\rho^2_{a,y_0:y_L}(p) &= 1 - \det\left[\tilde{\mathbf{R}}_a(p)\right] \\ &= 1 - \frac{\det\left[\mathbf{R}_a(p)\right]}{\prod_{n=0}^{L}\sigma^2_{y_n}}.\end{aligned} \tag{3.33}$$

Der quadratische MCCC hat einige Eigenschaften, die zur Formulierung des TDOA verwendet werden können:

1. Sind zwei oder mehr Mikrofone perfekt korreliert, so ist $\rho^2_{a,y_0:y_L}(p) = 1$
2. Wenn alle Mikrofone zueinander komplett unkorreliert sind, so ist $\rho^2_{a,y_0:y_L}(p) = 0$
3. Sollte ein Signal komplett unkorreliert zu den anderen $L-1$ Signalen sein, so misst der MCCC die Korrelation zwischen den übrigen $L-1$ Signalen

Mit diesen Eigenschaften kann der TDOA zwischen den ersten beiden Mikrofonen im ersten Schritt ausgedrückt werden durch

$$\hat{\tau}^{MCCC} = \arg\max_p \rho^2_{a,y_0:y_L}(p). \tag{3.34}$$

Einsetzen der Gleichungen 3.32 und 3.33 und Umstellen der Gleichung 3.34 führt zu dem äquivalenten Ausdruck:

$$\begin{aligned}\hat{\tau}^{MCCC} &= \arg\max_p \left\{1 - \det\left[\tilde{\mathbf{R}}_a(p)\right]\right\} \\ &= \arg\max_p \left\{1 - \frac{\det\left[\mathbf{R}_a(p)\right]}{\prod_{n=0}^{L}\sigma^2_{y_n}}\right\} \\ &= \arg\min_p \det\left[\tilde{\mathbf{R}}_a(p)\right] \\ &= \arg\min_p \det\left[\mathbf{R}_a(p)\right].\end{aligned} \tag{3.35}$$

Abb. 3.4 zeigt schematisch den Ablauf der Methode der MCCC.

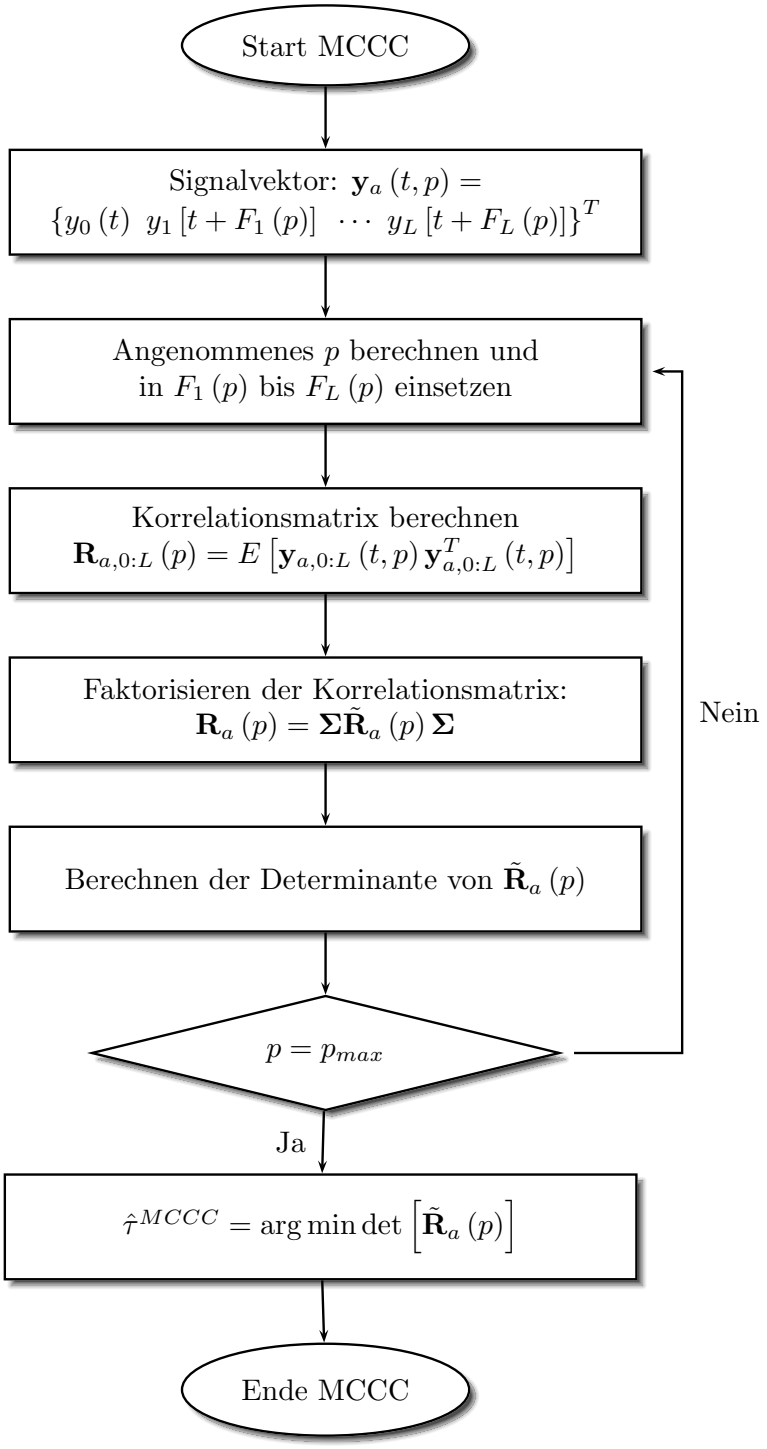


Abbildung 3.4.: Ablauf der Berechnung des Multichannel Cross Correlation Coefficients

3.5. Vergleich der beiden Methoden

Die beiden Methoden MCCC und SLP sind mathematisch mit der räumlichen Korrelationsmatrix verbunden. Der Zusammenhang wird deutlicher, wenn ein Ausdruck für $\hat{\tau}^{SLP}$ gefunden wird, der wie der $\hat{\tau}^{MCCC}$ direkt von der Korrelationsmatrix abhängt. Die Herleitung eines solchen Ausdrucks wird in [BCH08, S. 198ff.] geliefert. An dieser Stelle wird nur das Ergebnis dieser Herleitung gezeigt:

$$\hat{\tau}^{SLP} = \arg\min_{p} \frac{det\left[\tilde{\mathbf{R}}_a(p)\right]}{det\left[\tilde{\mathbf{R}}_{a,1:L}(p)\right]}. \tag{3.36}$$

Die Gleichungen 3.36 und 3.35 weisen auf den ersten Blick eine hohe Ähnlichkeit auf, jedoch sind sie im Bezug auf Praxisanwendungen sehr unterschiedlich nutzbar. Die Berechnung von $\hat{\tau}^{SLP}$ kann numerisch Instabil sein, da eine Division durch den Ausdruck $det\left[\tilde{\mathbf{R}}_{a,1:L}(p)\right]$ durchgeführt werden muss, der im Sonderfall null sein kann[7]. Diese Instabilität zeigt sich in der Herleitung der SLP in Gleichung 3.25, in der mit der inversen Korrelationsmatrix gerechnet wird. Wie schon erwähnt, ist diese bei der passenden Verschiebung $F_n(p)$ der Signale - welche die Richtung der Quelle repräsentiert - nur dann invertierbar, wenn Rauschen vorliegt. Dagegen ist die Berechnung von $\hat{\tau}^{MCCC}$ in Gleichung 3.35 immer stabil. Des Weiteren wird sich in der Simulation der Algorithmen zeigen, dass die MCCC eine weitaus eindeutigere und bessere Annäherungen des TDOA hervorbringt.
Eine näherer Betrachtung der beiden Algorithmen zeigt außerdem einen weiteren deutlichen Nachteil der SLP. Der Vergleich der Komplexität zwischen einer Matrixinversion, die meist nur numerisch implementiert werden kann und außerdem nur rekursive Verfahren kennt, und der Berechnung der Determinante, die mit einer Vorwärtsstruktur implementiert wird, lässt die Vermutung zu, dass die MCCC besser für eine Echtzeitanwendung geeignet ist. Diese Vermutung wird hinsichtlich des Ziels *Zuverlässigkeit* zu überprüfen sein.

3.6. Kreuzkorrelation als Faltung zweier Signale

Die beiden vorgestellten Algorithmen basieren auf der Berechnung der Kreuzkorrelationsfunktion (KKF) zwischen den einzelnen Kanälen. Das heißt, es wird die Ähnlichkeit zweier Signale geprüft. In diesem Abschnitt wird eine Möglichkeit dargestellt, eine - bei der Implementierung aufwendige - Kreuzkorrelation über die Faltung zweier Signale zu berechnen.
Mathematisch ist die KKF $\varphi_{rs}(\tau)$ durch folgendes Integral definiert:

[7] Dieser Sonderfall bezieht sich wieder auf das Nichtvorhandensein von Rauschen, da in diesem Fall die Determinante null ist.

$$\varphi_{rs}(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} r(t)\, s(t+\tau)\, dt. \tag{3.37}$$

Dabei seien $r(t)$ und $s(t)$ zwei verschiedene reelle Signale[8], die verglichen werden sollen und τ die angenommene Verschiebung, für die ein Integral berechnet wird. Ein weiterer Parameter ist T, die Fensterlänge, in der die beiden Signale auf Redundanz geprüft werden. $\varphi_{rs}(0)$ entspricht der Kovarianz zwischen den Signalen, die dann bildlich gesehen genau übereinander liegen. Sind beide Signale zueinander verschoben, so gibt es im besten Fall - das heißt ohne vorhandenem Rauschen - ein herausragendes Maximum, das an dem τ liegt, bei dem die beiden Funktionen genau übereinander liegen (vgl. Abb. 3.5 oben). Bei periodischen Signalen entsteht eine KKF wie in Abb. 3.5 unten. Die Periodizität ist auch in der KKF zu erkennen.

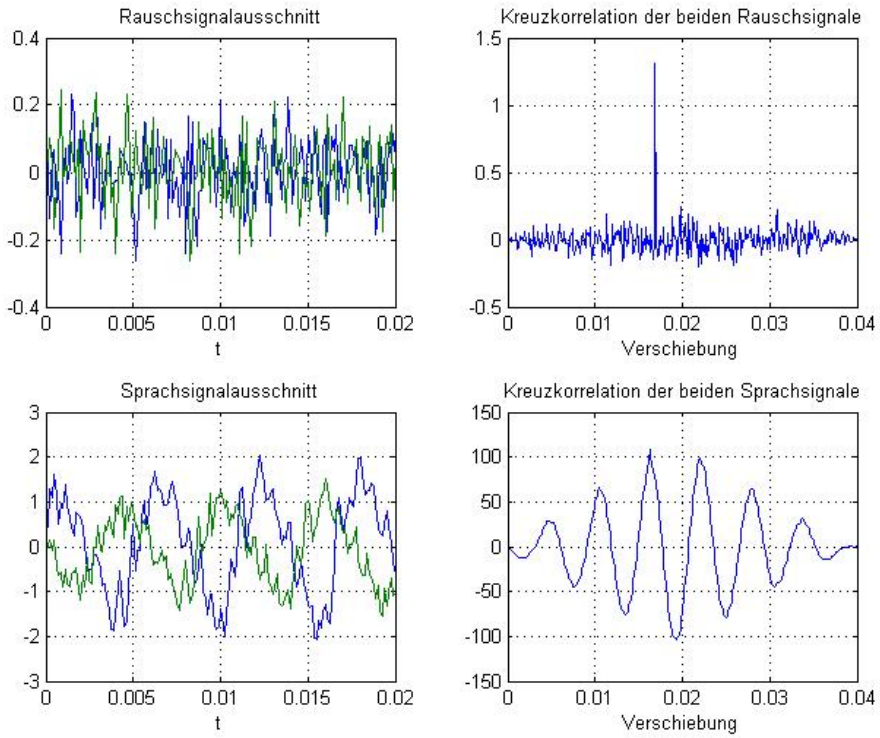


Abbildung 3.5.: Beispiele für Kreuzkorrelationen zweier zueiander verschobener Signale

In der Realität (Implementierung auf einem Rechner) ist eine Grenzwertbildung natürlich nicht möglich. Somit kann lediglich eine Schätzung der KKF berechnet werden. Zudem liegen die Daten diskret vor, wodurch Gleichung 3.37 als Kreuzkorrelationsfolge

[8] Für den Fall, dass gilt $r(t) = s(t)$, handelt es sich um die Autokorrelationsfunktion (AKF).

$$\varphi_{rs}[k] = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} r[n]\, s[n+k] \tag{3.38}$$

(mit der Fensterlänge N) dargestellt werden muss. Diese Vorschrift ist in einer Implementierung nur bedingt einsetzbar, da sie eine ganze Reihe Multiplikationen und Additionen beinhaltet. Über eine Umformung kann aber gezeigt werden, dass eine Rechenzeitoptimierung erwirkt werden kann, wenn die KKF als Faltungsintegral dargestellt wird. Zugrundegelegt wird dafür die Faltung eines Eingangssignals $s(t)$ mit der Impulsantwort $h(t)$ eines linearen zeitinvarianten Systems[9]:

$$\begin{aligned} g(t) &= s(t) * h(t) \\ &= \int_{-\infty}^{\infty} s(\tau) \cdot h(t-\tau)\, d\tau. \end{aligned} \tag{3.39}$$

Diskretisierung von Gleichung 3.39 - also Faltung einer Eingangsfolge $s[n]$ mit der Impulsantwortfolge $s[n]$ eines linearen verschiebungsinvarianten Systems[10] - führt zu

$$\begin{aligned} g[n] &= s[n] * h[n] \\ &= \sum_{k=-\infty}^{\infty} s[k] \cdot h[n-k]. \end{aligned} \tag{3.40}$$

Der Unterschied zur KKF besteht darin, dass die Integration im Zeitbereich über die Variable τ und nicht wie bei der KKF über t geschieht. Dies impliziert in den diskreten Formeln eine Vertauschung der Summationsvariablen. Um die KKF als Faltung darzustellen, wird eine Anpassung der Indizes von s aus Gleichung 3.38 vorgenommen

$$\varphi_{rs}[k] = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} r[n]\, s[k-(-n)] \tag{3.41}$$

und anschließend $r[n]$ in $r[-(-n)]$ umgewandelt

[9] Linear Timeinvariant (LTI)-System
[10] Linear Shiftinvariant (LSI)-System

$$\varphi_{rs}[k] = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} r[-(-n)]\, s[k-(-n)]\,. \tag{3.42}$$

Durch einen Vergleich der Gleichungen 3.42 und 3.40 kann die KKF als Faltung zweier Signale ausgedrückt werden:

$$\varphi_{rs}[k] = r[-k] * s[k]\,. \tag{3.43}$$

Die aufwendige Faltung wird durch die Transformation der gesamten Gleichung 3.43 in den Frequenzbereich durch eine Multiplikation ersetzt:

$$\begin{aligned} \varphi_{rs}[k] \circ\!\!-\!\!\bullet\, \underline{\Phi}_{rs}[l] &= \underline{R}[-l] \cdot \underline{S}[l] \\ &= \underline{R}^*[l] \cdot \underline{S}[l]\,, \end{aligned} \tag{3.44}$$

wobei $\underline{\Phi}_{rs}[l]$ das Kreuzleistungsdichtespektrum der beiden Signale ist. Somit wird ein Ausdruck für die KKF gefunden, der ohne die vielen Multiplikationen der Faltung auskommt. Die Vorgehensweise für die Kreuzkorrelation zweier Signale ist also wie folgt zusammenzufassen:

1. Aufnehmen der Signale $r[-k]$ und $s[k]$
2. Fouriertransformation in $\underline{R}$ und $\underline{S}$ und Berechnung von $\underline{\Phi}_{rs} = \underline{R}^* \cdot \underline{S}$
3. Inverse Fouriertransformation von $\underline{\Phi}_{rs}$ ergibt $\varphi_{rs}[k]$

Unter Verwendung einer Fast Fouriertransformation (FFT) ist dieser „Umweg“ über den Frequenzbereich schneller als der Ansatz, die Kreuzkorrelationsfolge aus 3.38 im Zeitbereich zu bilden.
Das Ergebnis einer Faltung hat stets die doppelte Länge der Eingangssignale (vorausgesetzt, die beiden Signale sind gleich lang). Damit ist auch das Ergebnis nach der Transformation in den Frequenzbereich doppelt so lang.

3.7. Eigenschaften und Analyse eines Sprachsignals

Dieser Abschnitt ist teilweise aus [Pik10] übernommen, da diese Arbeit als Vorgänger der vorliegenden angesehen werden kann.
Die Spracherkennung an sich beinhaltet eine Reihe von Aufgaben. Dazu gehören zum Beispiel die Spracherkennung im engeren Sinne (Was wurde gesagt?) oder die Sprechererkennung (Wer hat gesprochen?). Zu Beginn aller Aufgaben steht jedoch die Detektion des Sprachsignals, das heißt die Erkennung, wann eine Spracheingabe startet und wann sie endet. Um diese Entscheidung zu treffen, wird bei der Sprachsignalanalyse eine Merkmalsextraktion durchgeführt, in der die für die weitere Verarbeitung wichtigen Merkmale des Sprachsignals ermittelt werden. Diese sind z. B.:

- Energie (Lautstärke)
- Nulldurchgangsrate
- Grundfrequenz

von denen in dieser Arbeit die Energie benutzt wird. Diese Merkmale werden von verschiedenen sprecherabhängigen Parametern bestimmt, zu denen z. B. Betonungen, die Anzahl und Länge der Pausen beim Reden, das Geschlecht der sprechenden Person und die Stimmung des Sprechers gehören. Bei der Merkmalsextraktion werden die oben genannten Merkmale aus dem Signal detektiert und in einem Merkmalsvektor zusammengefasst, der spezifisch für einen Abschnitt eines Sprachsignals ist. Die Merkmalsvektoren bilden gemeinsam eine Merkmalsmatrix des Sprachsignals, die anschließend analysiert werden kann.

3.7.1. Stationarität von Sprache

Merkmalsvektoren haben nicht für einzelne Abtastwerte, sondern nur für große Signalabschnitte tatsächlich eine Bedeutung, weshalb es sinnvoll ist, ein Sprachsignal in Abschnitte zu unterteilen, bevor die Merkmale berechnet werden. Dieser Vorgang wird als Framing oder auch Rahmenbildung bezeichnet. Mit dieser Rahmenbildung nutzt man die statistischen Eigenschaften von Sprache. Aufgrund des stark veränderlichen Verhaltens kann für Sprachsignale im Allgemeinen keine Stationarität angenommen werden. Haben die Rahmen jedoch eine bestimmte Länge, so kann das Sprachsignal als quasistationär betrachtet werden, wodurch Algorithmen angewendet werden können, die für lineare und zeitinvariante Signale bestimmt sind. Diese Länge der Rahmen kann zwischen 20 und 400ms variieren [Kap08, S.9]. Laut [Eul06] ist die Stationarität eines Sprachsignals sogar nur für eine maximale Länge von etwa 15 bis 20ms gewährleistet, in [VHH98] ist eine Dauer von 20 bis 50ms angegeben. Für diese Arbeit soll die Stationarität von Sprache mit einer maximalen Länge von 50ms angenommen werden.

3.7.2. Energie eines Sprachsignals

Die Analyse eines Sprachsignals beginnt bei Detektion von Sprache. Hierbei gilt es aus einem kontinuierlichen Signal zu erkennen, wann genau eine Spracheingabe beginnt und wann sie endet. Übliche Verfahren stützen sich auf die Ermittlung der Lautstärke, also die Energie eines Signalabschnitts. Über die Berechnung der Energie eines Sprachsignals werden Aussagen darüber getroffen, ob das zum gegebenen Zeitpunkt analysierte Signalstück stimmlos oder stimmhaft ist. Die Energie berechnet sich aus

$$E(k) = \sum_{n=t_{min}}^{t_{max}} x(n)^2 \tag{3.45}$$

und kann für eine bessere Vergleichbarkeit logarithmiert werden. Dabei steht $E(k)$ für die Energie des k^{ten} Rahmens. Abbildung 3.6 zeigt für einen Sprachsignalausschnitt den Verlauf der so berechneten Werte $E(k)$ mit einer Rahmengröße von 128.

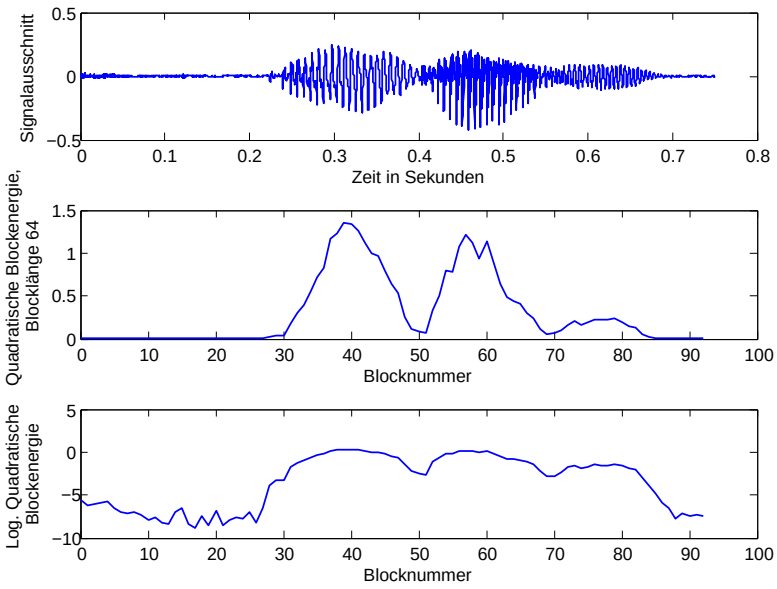


Abbildung 3.6.: Zeitverlauf eines Wortes (oben) und der blockweise berechneten Energie normal und logarithmiert

Wichtig bei der Energieberechnung ist die Dimensionierung der Schwelle, ab der ein Abschnitt als Pause eingestuft werden soll. Diese ist entweder nur sehr großzügig zu wählen, damit keine Sprachdaten verloren gehen oder empirisch zu ermitteln. Wird die Schwelle aber sehr großzügig eingestellt, muss darauf geachtet werden, dass auch Sprachpausen oder weitesgehend stimmlose Sprachblöcke analysiert werden könnten.

4. Aufbau des DSP-basierten Echtzeitsystems

In diesem Kapitel wird die Hardware beschrieben, die für den Ablauf des Algorithmus benötigt wird. Zunächst wird der Aufbau des Gesamtprojektes erläutert, in dem diese Arbeit entsteht. Anschließend werden die Mikrofonarrays und deren Aufbau und das verwendete DSP-Board beschrieben und die Plattform erklärt, auf welcher der Algorithmus in C und Assembler implementiert wurde. Die Arbeit ist als Nachfolger von [Pik10] einzuordnen, deswegen ist dieses Kapitel mit einigen Anpassungen als Vorbild genommen worden.

4.1. Aufbau des Gesamtprojekts

Abbildung 4.1 [Sau11] zeigt das Gesamtsystem, in dessen Rahmen diese Arbeit entsteht. Das Gesamtsystem besteht aus vier kaskadierten DSPs, zwischen denen ein kontinuierlicher Datenfluss besteht.

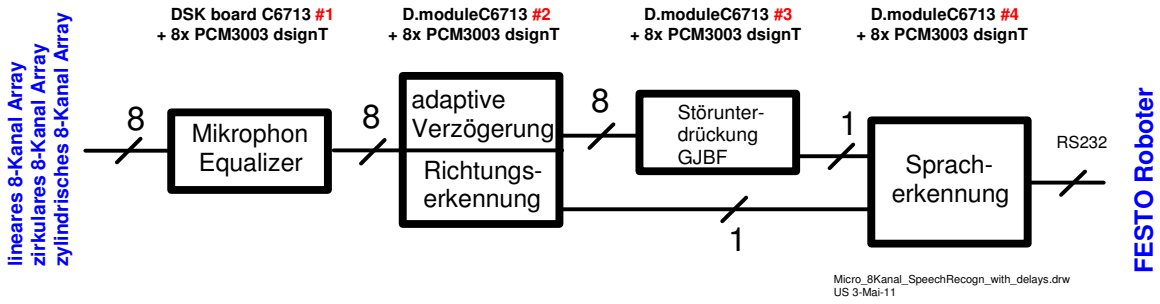


Abbildung 4.1.: Gesamtsystem zur Spracherkennung isolierter Sprachkommandos

Das erste Hardwaremodul führt dabei eine 8-Kanal Amplitudengangsentzerrung durch, da die Amplitudengänge der Mikrofonkapseln sehr unterschiedlich sind. Die nächsten beiden Platinen übernehmen weitere Aufgaben, welche die Güte der Spracherkennung auf der letzten Platine erhöhen sollen. Die Spracherkennung ist stark abhängig von der Qualität - von der Stärke des Rauschens auf dem Nutzsignal - der Eingangsdaten. Deswegen wird vor der

eigentlichen Spracherkennung eine Störungsextraktion mittels des Griffith-Jim Beamformer (GJBF)-Algorithmus durchgeführt. Dieser Algorithmus hat jedoch nur unter der Vorraussetzung eine hohe Effektivität, dass die Eingangssignale nahezu ohne Verzögerung vorliegen (vgl. hierzu [BW01, S. 98ff]). Aus diesem Grund sollen die Daten für die Störextraktion vorher in der Form aufbereitet werden, dass die Verzögerung berechnet und anschließend mit Hilfe einer VDL ausgeglichen werden. Diese Arbeit bereitet die Daten für den GJBF auf.

4.2. Die Mikrofonarrays

In diesem Abschnitt werden die Eigenschaften der drei verwendeten Mikrofonarrays, sowie deren Konstruktion beschrieben. Es werden in dieser Anwendung omnidirektionale Elektret-Kondensatormikrofonkapseln verwendet. Diese arbeiten im Niederspannungsbereich bei 1.5V in einem Frequenzbereich von 20 - 16000 Hz [Pan] und sind damit für alle Sprachverarbeitungsanwendungen geeignet. Die Ausgangsspannung der Mikrofonkapsel liegt bei weniger als 100mV und kann von Kapsel zu Kapsel variieren. Da die benötigte Eingangsspannung des Analog Digital Wandlers des PCM3003 Codec bei 1.8Vpp liegt, ist es notwendig, eine Vorverstärkerschaltung hinter die Mikrofonkapsel zu schalten.

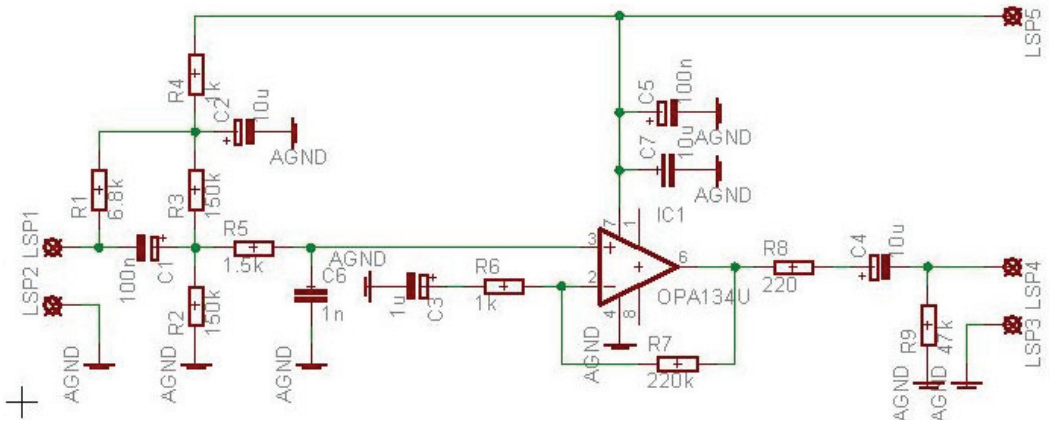


Abbildung 4.2.: Schaltung des Mikrofonverstärkers

Die Mikrofone sind nicht ideal, wodurch individuelle Phasenverzögerungen und nichtlineare Verzerrungen entstehen. Somit kann auch die anschließende Signalverarbeitung nicht mehr als ideal angesehen werden. Um die Verzerrungen trotzdem gering zu halten, wurde ein Mikrofon-Equalizer auf einem separaten Board implementiert (s. Abb. 4.1).
Der Vorverstärker hat zudem die Aufgabe einen Grundpegel (Offset) auf das Mikrofonsignal zu legen, da das Eingangssignal sowohl positive als auch negative Werte aufweist und der Bereich der Eingangsspannung des PCM3003 von 0-3.3V liegt. Abbildung 4.2 zeigt die

Schaltung des Vorverstärkers.
Die Dimensionierung des Abstands d in den Mikrofonarrays geschah nach Gleichung 3.8 für das ULA und 3.11 für das UCYA. Mit einer maximalen Frequenz von $3.4kHz$ ergibt sich für das lineare Array ein Abstand $d_{linmax} = 0.05m$. Mit der gleichen Frequenz für die zirkularen Arrays und $N = 8$ Mikrofonen ergibt sich $d_{zirmax} = 0.056m$. Hier wurde ein Abstand von 5cm gewählt. Daraus resultiert der Radius des Arrays für acht Mikrofone ($N = 8$) $r = \frac{N*d}{2pi} = 0.071m$. Abbildung 4.3 zeigt das ULA mit acht Mikrofonen. Die Front ist mit Teppich abgeklebt, um die Reflektionen gering zu halten. Abbildung 4.4 zeigt das zylindrische Mikrofonarray, dessen Oberfläche aus Schaumstoff besteht.

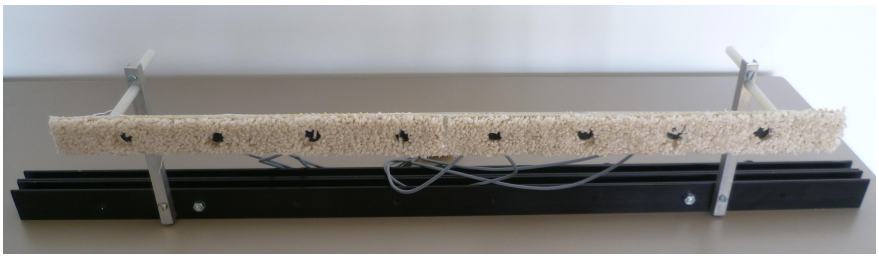


Abbildung 4.3.: Lineares Mikrofonarray mit acht Mikrofonen

Abbildung 4.4.: Zylindrisches Mikrofonarray mit acht Mikrofonen

4.3. DSP-Teilsystem

Das DSP-Teilsystem lässt sich aufteilen in die I/O-Karte D.Module.PCM3003 und das eigentliche DSP Board D.Module.C6713 der Firma Digital Signalprocessing Technology (D.SignT). Abbildung 4.5 zeigt das gesamte DSP-System, in dem die I/O-Karte auf das DSP-Board über eine Adapterplatine aufgesetzt ist.

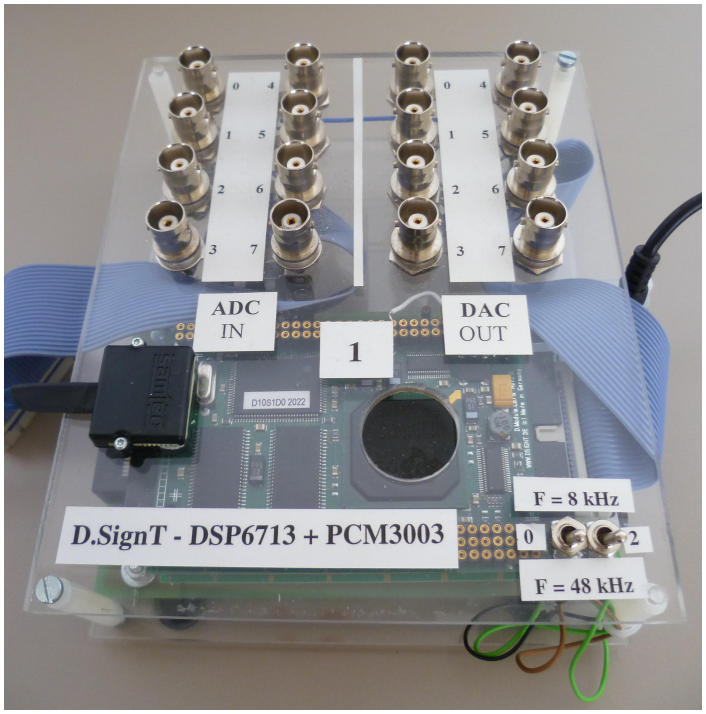


Abbildung 4.5.: DSP-Teilsystem bestehend aus D.Module.C6713 und Tochterkarte

4.3.1. Signalprozessorplatine D.Module.C6713

Der DSP auf der Platine D.Module.C6713 ist der TMS320C6713b aus der TMS320C6000-Reihe von Texas Instruments, die für rechenintensive Algorithmen entwickelt wurde. Dies wird durch die Nutzung der Very Long Instruction Word Architektur bewirkt. Der interne Speicher der C6700-Reihe ist so konzipiert, dass bis zu acht Rechenoperationen gleichzeitig pro Taktschritt durchgeführt werden können. Der DSP kann sowohl in Fixed-Point, als auch in Floating-Point Arithmetik programmiert werden. In dieser Masterarbeit wird er als floating-Point DSP verwendet. Mit einer Taktfrequenz von 300 MHz kann der DSP 1.8 GFLOPS (Giga-floating-point operations per second) ausführen. Als Schnittstellen sind zwei

Multi-channel Buffered Serial Port (McBSP), ein I2C-Anschluss, sowie ein Enhanced Direct Memory Access (EDMA) eingerichtet. Die Platine hat u. a. folgende Ausstattung:

- 64 MB SDRAM[11], der mit 100MHz arbeitet
- 2 MB Flash Speicher
- 256 KB Interner Speicher
- USB Schnittstelle zum PC inkl. JTAG Emulator
- LEDs und 4 DIP-Schalter für die Interaktion mit dem Benutzer

Abb. 4.6 zeigt ein Blockschaltbild des D.Module.C6713 [NK10].

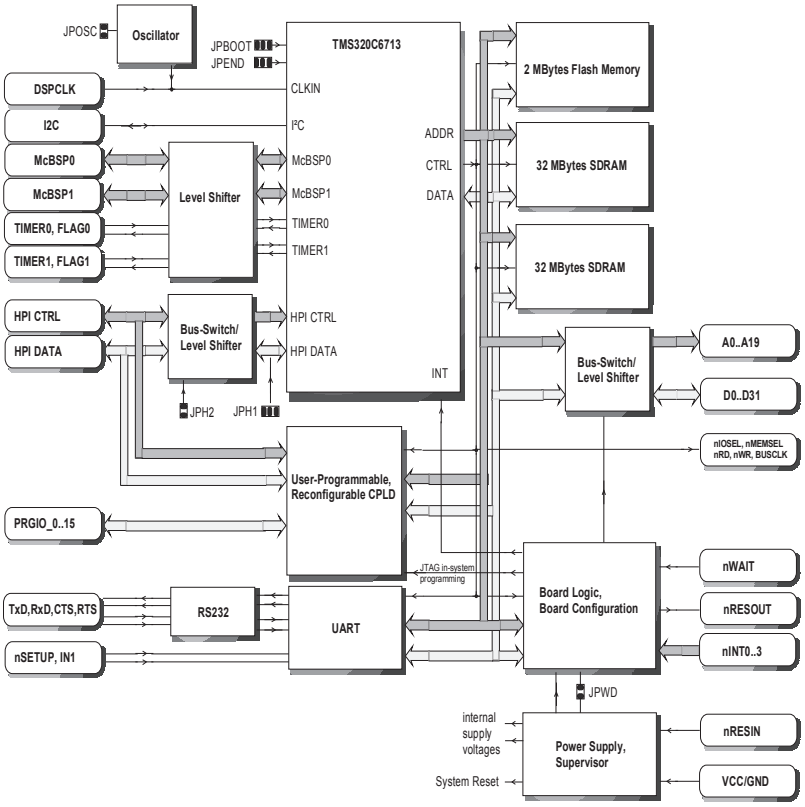


Abbildung 4.6.: Blockdiagramm des D.Module.C6713

[11] Synchronous Dynamic Random Access Memory (SDRAM)

4.3.2. D.Module.PCM3003 Tochterkarte

Das D.Module.PCM3003 ist ein Audiocodec Modul, das vor allem für Mehrkanal-Audio, Sprachsignalverarbeitung und Mikrofonarrayverarbeitung entwickelt wurde [NK05]. Es kann als vier- und achtkanal Audiocodec verwendet werden. In dieser Arbeit werden alle acht Kanäle benutzt. Das Hardwaremodul ist mit acht 16bit auflösenden A/D-D/A-Wandlern ausgestattet, deren Delta-Sigma-Modulatoren von 64. Ordnung sind.
Die A/D- und D/A-Wandler der einzelnen Kanäle werden synchron von einem einstellbaren Oszillator getaktet. Dieser Oszillator kann auf alle gängigen Abtastraten im Audiobereich eingestellt werden oder auch über einen externen Taktgeber gesteuert werden. Die Abtastrate für dieses System wird in den kommenden Kapiteln festgelegt. Generell werden alle Synchronisationssignale von dem Zusatzmodul generiert, sodass dieses als Mastergerät fungiert. Das D.Module.PCM3003 wird bei der Achtkanalvariante über zwei serielle Schnittstellen mit dem D.Module.C6713 verbunden. Abbildung 4.7 zeigt das Blockdiagramm des D.Module.PCM3003 [NK05].

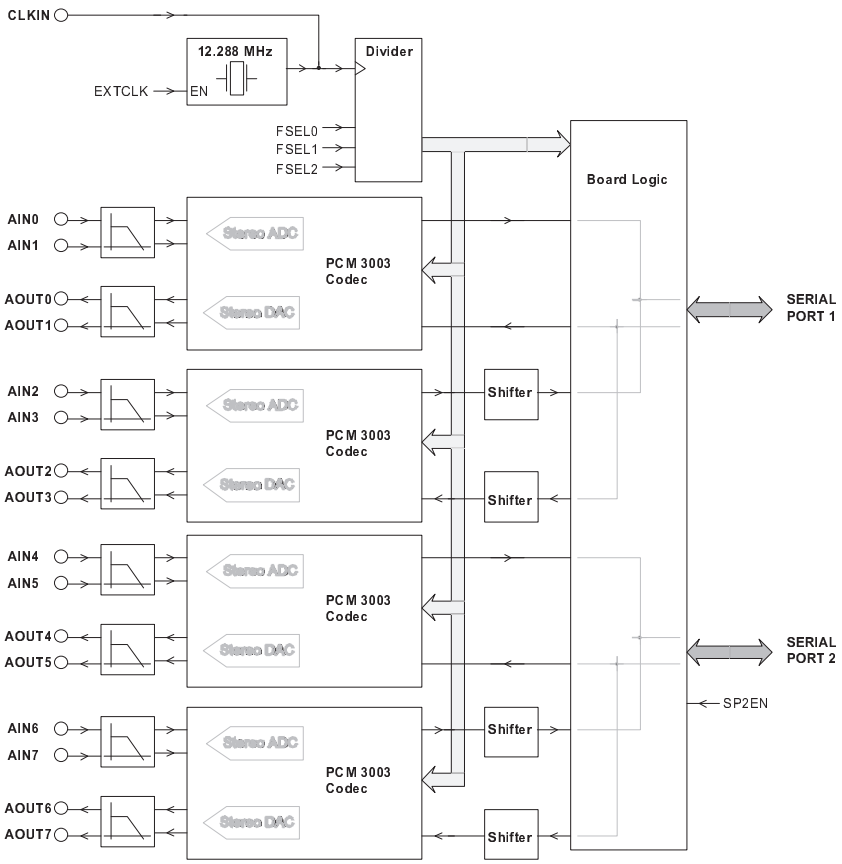


Abbildung 4.7.: Blockdiagramm des D.Module.PCM3003

4.3.3. Code Composer Studio

Das Code Composer Studio (CCS) ist die Entwicklungsumgebung für die Programmierung u.a. von DSPs der Firma Texas Instruments. CCS bietet eine sogenannte integrierte Entwicklungsumgebung für Echtzeit-DSP Anwendungen, die auf der Programmiersprache C basieren. CCS beinhaltet einen C-Compiler, einen Assembler und einen Linker. Für die bessere Darstellung von Variablen und Daten besitzt das Programm graphische Darstellungsmöglichkeiten. In dieser Arbeit wird das CCS Version 3.3 benutzt. Abbildung 4.8 zeigt ein Ablaufdiagramm für die Softwareentwicklung für die TMS320C6700-Reihe.

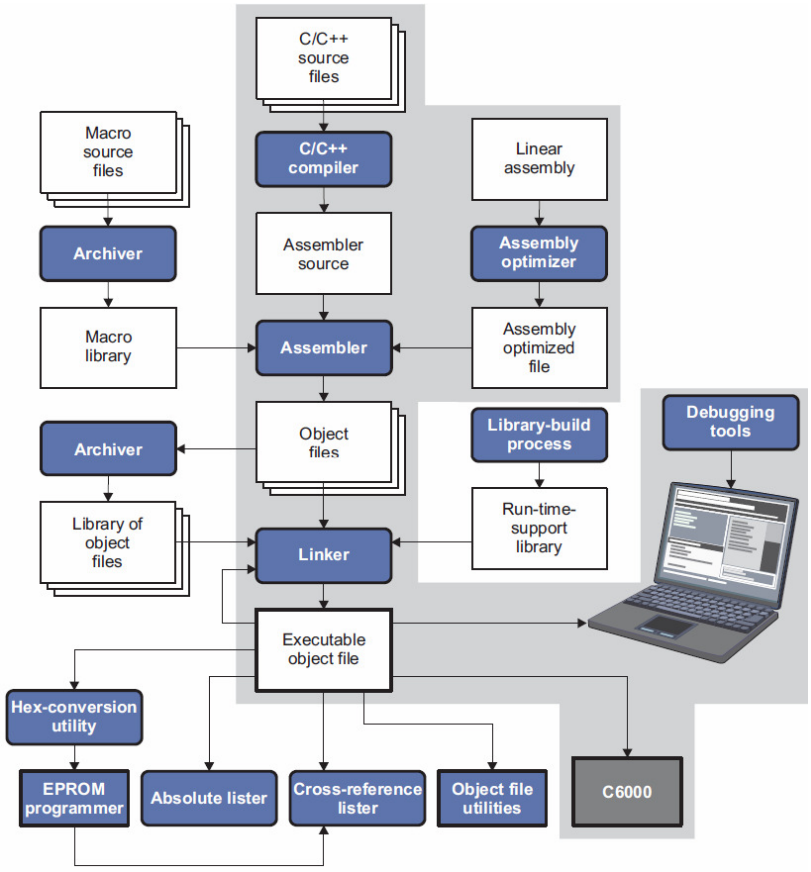


Abbildung 4.8.: Ablauf für die Softwareentwicklung

4.4. Schnittstelle zwischen DSP und PCM3003

Wie in Unterabschnitt 4.3.2 erwähnt, ist der PCM3003 Codec dazu konzipiert, Signale aufzunehmen und zu verarbeiten, die anschließend an das DSP-Board weitergegeben und dort entsprechend weiterverwendet werden können. Für die Schnittstelle zwischen DSP und dem PCM3003 Codec besitzt die DSP-Platine zwei McBSP, die jeweils die Daten zweier Audiokanäle empfangen. Der Datenaustausch zwischen dem McBSP und dem Codec findet über zwei Datenleitungen statt, eine für den Empfang, die andere für das Senden. Die Steuerinformationen wie Synchronisation und Taktung werden über eigene Leitungen übertragen. Abbildung 4.9 zeigt das Funktionsblockdiagramm [AC04] eines McBSP mit den einzelnen Pins, über die die Kommunikation mit dem Codec abläuft.

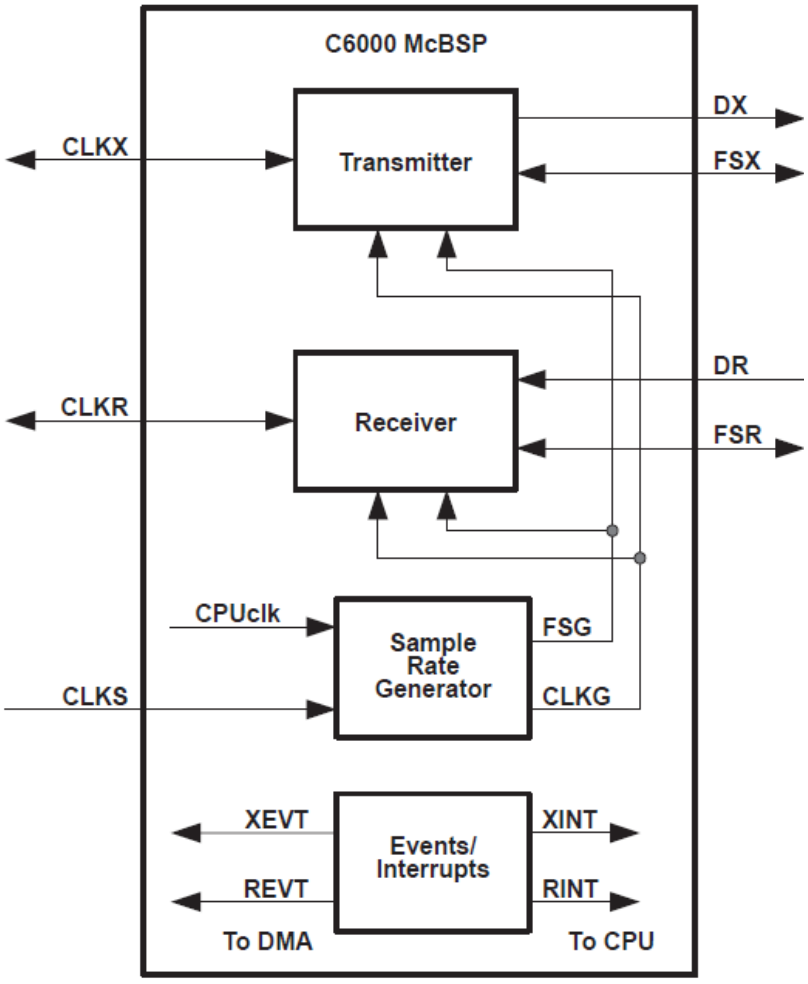


Abbildung 4.9.: Funktionsblockdiagramm eines McBSPs

Die serielle Übertragung zwischen PCM3003 und DSP geschieht nicht Kanal für Kanal in aufsteigender Reihenfolge, sondern nach folgendem Muster:

1. Codec linker Audiokanal → Channel 0
2. Codec linker Audiokanal → Channel 2
1. Codec rechter Audiokanal → Channel 1
2. Codec rechter Audiokanal → Channel 3
3. Codec linker Audiokanal → Channel 4
4. Codec linker Audiokanal → Channel 6
3. Codec rechter Audiokanal → Channel 5
4. Codec rechter Audiokanal → Channel 7

Dieses Muster entsteht dadurch, dass der PCM3003 die Ausgangsdaten zweier Analog Digital Umsetzer (ADU) auf einer Datenleitung an die DSP-Platine verschickt. Andersherum führt das PCM3003 eine Trennung der Daten von der DSP-Platine durch und verteilt sie auf zwei Digital Analog Umsetzer (DAU). Da es nur einen „FrameSync“-Takt bei dem Start der Übertragung des linken und rechten Kanals gibt, ist es schwierig zwischen den beiden zu unterscheiden. Der Datenempfang an den McBSP ist in Abbildung 4.10 dargestellt [Kle10].

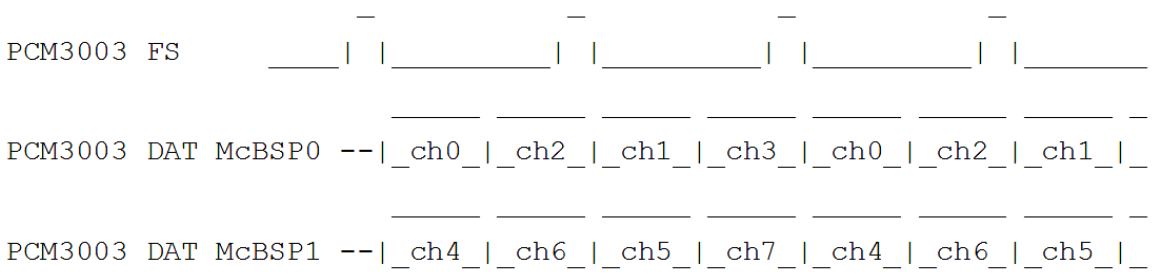


Abbildung 4.10.: Datenempfang an den McBSPs

Das wichtigste Ziel bei der Softwareentwicklung ist, die Blöcke der ADU Kanäle des Codecs so schnell wie möglich in den DSP zu lesen, um die Vorraussetzungen für eine Echtzeit-Signalverarbeitung zu gewährleisten. Der effizienteste Weg hierfür ist es, einen EDMA zu nutzen.
Die Main-Funktion in dieser Arbeit wird auf dem Demoprogramm `pcm3003.c` aufgesetzt, das von D.SignT angeboten wird. In diesem Demoprogramm sind für die Übermittlung der Daten vom PCM Codec zum DSP alle Funktionen bereits implementiert. Es übernimmt die Initialisierung und Konfiguration des PCM Codec mit den McBSPs des DSPs. Weitere Informationen können aus [AC04] bezogen werden.

Die Übertragung der Daten vom Codec zum DSP ist frei von jeglichen CPU-Einflüssen[12]. Er erhält lediglich einen Interrupt vom EDMA, wenn der Datentransfer abgeschlossen ist und die Daten verarbeitet werden können. Auf diese Art wird die Geschwindigkeit und Effizienz der Anwendung gesteigert. Wie häufig der CPU einen Interrupt erhält, ist abhängig von der Blockgröße. Je kleiner die Blockgröße, desto häufiger erhält der CPU einen Interrupt und umgekehrt. Die Übertragung der Daten lässt sich wie folgt zusammenfassen: Das Programm setzt die entsprechenden Werte in den EDMA-Registern und startet diesen. Anschließend wartet es auf die Interrupts vom EDMA, die das Ende einer Übertragung bedeuten. Sobald ein Interrupt empfangen wurde, wird die Verarbeitung der neuen Daten gestartet. Nach Beendigung der Verarbeitung wartet das Programm auf den nächsten Interrupt vom EDMA. Das Blockdiagramm in Abbildung 4.11 soll diesen Vorgang verdeutlichen:

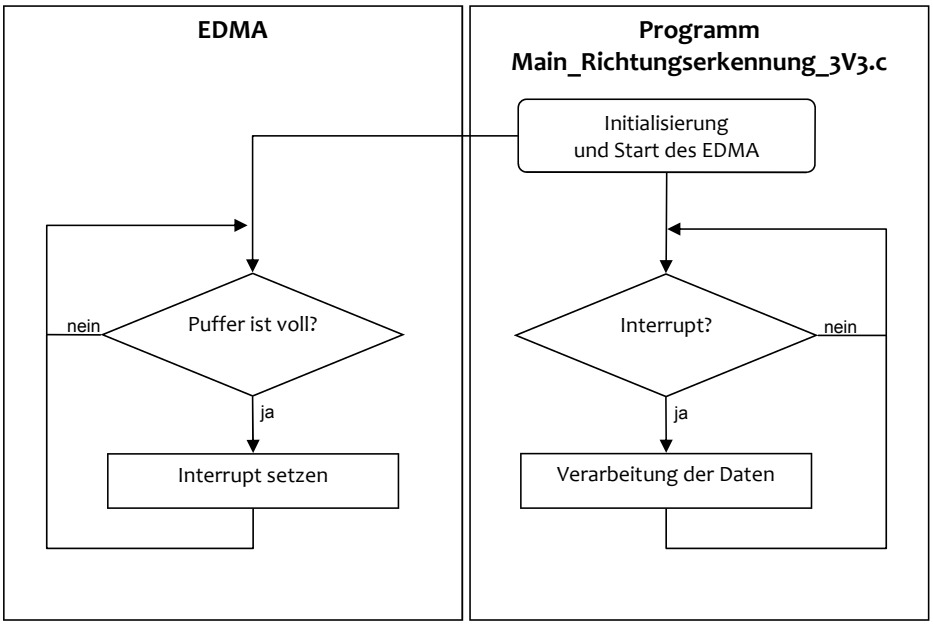


Abbildung 4.11.: Interruptbehandlung zwischen EDMA und pcm3003.c

Auch wenn diese Art der Datenübertragung die Einbindung des EDMA erlaubt, hat sie doch gewisse Einschränkungen für die Programmierung der Datenverarbeitung zur Folge. Die Verarbeitung der Daten muss in der Zeit geschehen, in welcher der EDMA die Audiodaten von dem Codec erhält. Für den Fall, dass der implementierte Code sehr rechenaufwendig ist, kann es passieren, dass Daten im Puffer vom EDMA überschrieben werden, bevor sie ausgelesen werden konnten. Solange die Datenpuffer vom EDMA kontinuierlich beschrieben und wieder geleert werden, muss die CPU genau an die EDMA-Geschwindigkeit angepasst werden. Um dieses Problem zu beheben, wird eine sog. Ping-Pong-Technik eingesetzt. Die

[12]Central Processing Unit (CPU)

Ping-Pong-Technik besagt, dass zwei Datenpuffer genutzt werden, die abwechselnd mit neuen Signaldaten gefüllt werden. Dieser Vorgang kann wie folgt beschrieben werden: Der EDMA erhält Daten und speichert sie im Ping-Puffer ab. Sobald der Puffer voll ist, speichert der EDMA die nächsten Daten im Pong-Puffer ab, sendet den Interrupt zur CPU und gibt die Daten im Ping-Puffer damit zur Verarbeitung frei. Wenn der Pong-Puffer voll ist, werden die Daten im Pong-Puffer freigegeben und die Prozedur wiederholt. Da angenommen werden kann, dass die Daten aus dem Ping-Puffer mittlerweile verwendet wurden, dürfen diese überschrieben werden. Somit verhindert diese Technik ein Überschreiben von ungenutzten Daten und gibt dem DSP außerdem mehr Zeit die Daten zu verarbeiten. Abbildung 4.12 zeigt das Prinzip der Ping-Pong-Technik.

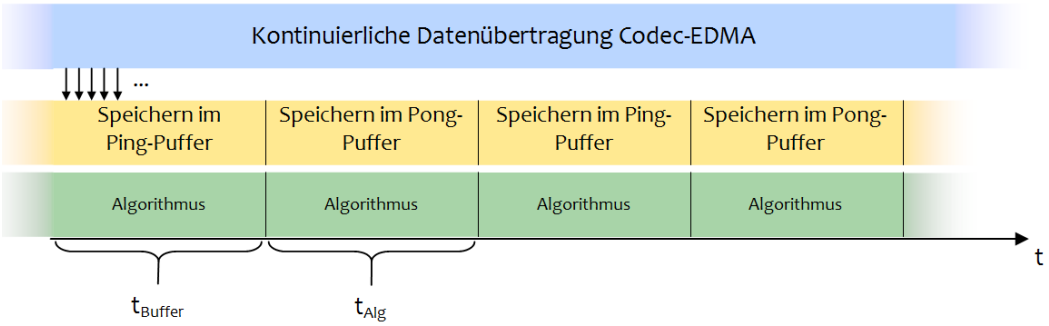


Abbildung 4.12.: Ping-Pong-Technik der Routine pcm3003.c

5. Diskussion von Umgebungsparametern

In diesem Abschnitt werden grundlegende Betrachtungen von Umgebungsparametern angestellt. Obwohl die Simulation unter MATLAB nicht echtzeitkritisch ist, soll die Erfüllung der Echtzeitfähigkeit schon hier ein zentraler Punkt sein.
Die Umgebungsparameter, die in diesem Kapitel diskutiert werden, sind:

- Winkelauflösung
- Maximal mögliche Verzögerung zwischen den Mikrofonen
- Datenlängen des EDMA und der FFT
- Zur Verfügung stehende Taktschritte

Wenn nicht anders definiert, werden alle Berechnungen mit einem Abstand zwischen den Mikrofonen von $d = 0.05\,m$ sowie $c = 343\,\frac{m}{s}$ durchgeführt.

5.1. Winkelauflösung

Angestrebt wird eine möglichst hohe Winkelauflösung, mit der eine hohe Genauigkeit bei der Winkeldetektion einhergehen würde. Da die Signaldaten auf dem DSP in diskreter Form vorliegen, können die Algorithmen nur dann ordnungsgemäß funktionieren, wenn die angenommenen TDOAs - bezogen auf die Abtastrate f_A - ganzzahlig sind. Aufgrund dieser Diskretisierung wird für eine diskrete Zeitdifferenz im Folgenden der Begriff Sample Difference of Arrival (SDOA) verwendet. Somit besteht ein direkter Zusammenhang zwischen einer möglichen Winkelauflösung und der verwendeten Abtastrate im System. Die Winkelauflösung θ_{auf,f_A} kann aus den Gleichungen für den TDOA zwischen zwei benachbarten Mikrofonen der Mikrofonarrays hergeleitet werden. Für ein ULA ergibt sich dieser aus Gleichung 3.1 für $n = 1$. Durch Einfügen der Abtastfrequenz f_A in der Gleichung ergibt sich K_{ULA} in Abtastwerten durch

$$\begin{aligned} K_{ULA} &= F_{1,lin}(\tau) \cdot f_A \\ &= \frac{d \cdot \sin(\theta)}{c} \cdot f_A. \end{aligned} \tag{5.1}$$

Es wird nun der Winkel θ gesucht, für den K_{ULA} gleich eins wird, was dem Winkel entspricht, für den der SDOA gleich eins ist:

$$\begin{aligned} 1 &= \frac{d \cdot \sin(\theta_{auf,f_A})}{c} \cdot f_A \\ \theta_{auf,f_A} &= \arcsin\left(\frac{c}{d \cdot f_A}\right) \cdot \frac{180}{\pi}. \end{aligned} \tag{5.2}$$

Da der Abstand d, sowie die Schallgeschwindigkeit c konstant sind, ergibt sich der Winkel θ_{auf,f_A} als nicht-lineare Funktion von f_A. Tabelle 5.1 gibt eine Übersicht über die möglichen Winkelauflösungen bei gängigen Abtastraten im Audiobereich.

Abtastrate (kHz)	Winkelauflösung (°)
8	59.0370
16	25.3883
24	16.6087
48	8.2167
96	4.0978

Tabelle 5.1.: Abhängigkeit der Winkelauflösung von der Abtastrate

Abbildung 5.1 stellt die Verzögerung K_{ULA} abgerundet in Abhängigkeit vom eingestellten Winkel und der Abtastfrequenz dar (s. Gleichung 5.1). Außerhalb eines Bereiches von $\pm 60°$ zeigt K_{ULA} keine Änderung, da die Vorschrift für K_{ULA} einem Sinus folgt. Dieser nimmt gegen 90° eine immer flacherer Steigung ein und zeigt keine großen Amplitudenveränderungen. Diese Tatsache zeigt schon jetzt, dass im Bereich von $\pm 60°$ gute Ergebnisse erreicht werden können.
Durch die Treppenform von K_{ULA} werden einige Winkel stets mit einem Fehler detektiert, was in Abbildung 5.2(a) deutlicher wird. Abbildung 5.2(b) zeigt den Betrag dieses Fehlers. Die Vielfachen der Winkelauflösung von 8.2197° werden mit einem Winkel von 0° erkannt, bis die Steigung der Sinusfunktion aus Gleichung 5.1 kleiner wird.

Außerhalb der 60° könnte dieser systematische Fehler durch eine Ausgleichsgerade verringert werden. Die Ausgleichsgerade müsste z. B. einen Winkel von -50° auf -60° ausgleichen,

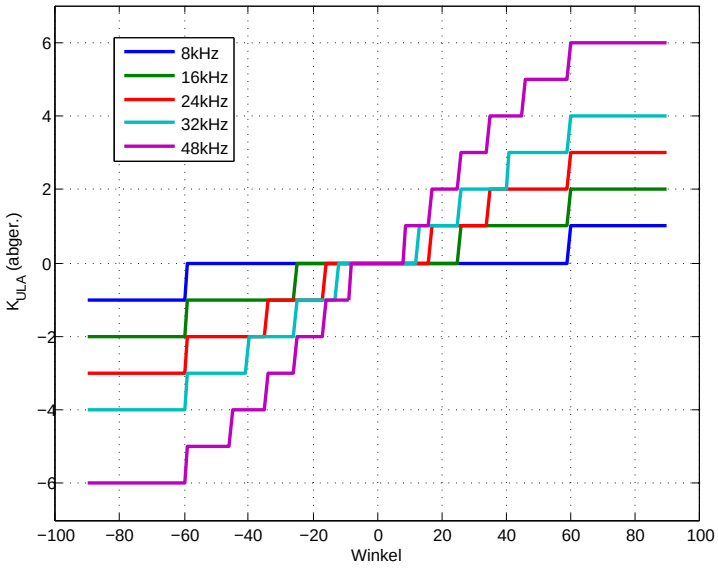
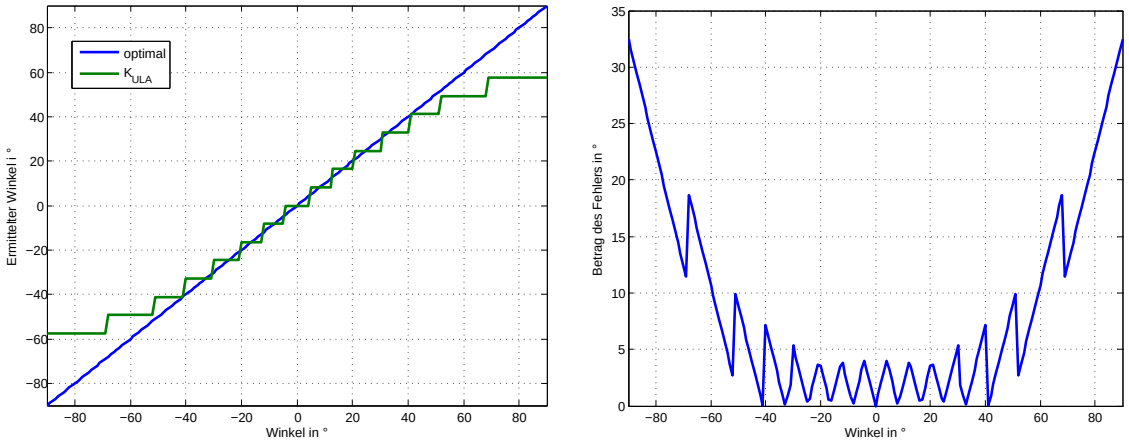


Abbildung 5.1.: K_{ULA} in Abhängigkeit von Abtastrate und Winkel



(a) Vergleich Optimum zu K_{ULA}

(b) Betrag des Fehler

Abbildung 5.2.: Analyse von K_{ULA} für $f_A = 48kHz$

da dies der Mittelwert der Winkel ist, die auf -50° detektiert werden. Da in der Anwendung jedoch das UCYA genutzt werden soll, wird in der Implementierung und der Simulation auf diese Ausgleichsgerade verzichtet und der systematische Fehler für ein ULA hingenommen. Für ein UCYA können die selben Werte für θ_{auf,f_A} verwendet werden. Dies erschließt sich aus der Betrachtung der Geometrie eines UCYAs in Abbildung 5.3:

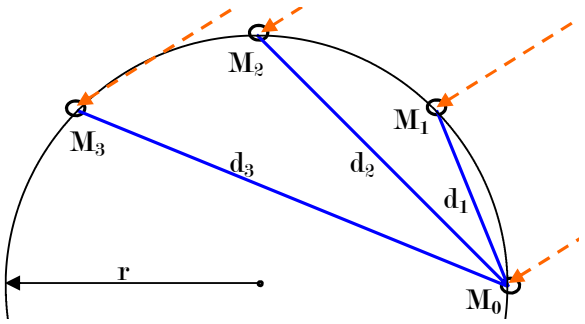


Abbildung 5.3.: Betrachtung der Winkelauflösung im UCYA

Wie in Abb. 5.3 dargestellt, kann jedes Mikrofonpaar M_0M_1, M_0M_2, ..., M_0, M_i als ein lineares Array aus 2 Mikrofonen mit unterschiedlichen Abständen angesehen werden. Folglich gilt für jedes Mikrofonpaar die Gleichung 5.1 mit dem jeweiligen Abstand d_i. Da K_{ULA} proportional zu d_i ist, interessiert nur der θ_{auf,f_A} für den kleinsten vorhandenen Abstand, der immer zwischen zwei direkt benachbarten Mikrofonen vorliegt. Dieser ist in den vorhandenen Aufbauten der ULA und UCYA gleich, damit gelten die gleichen Werte für θ_{auf,f_A}.

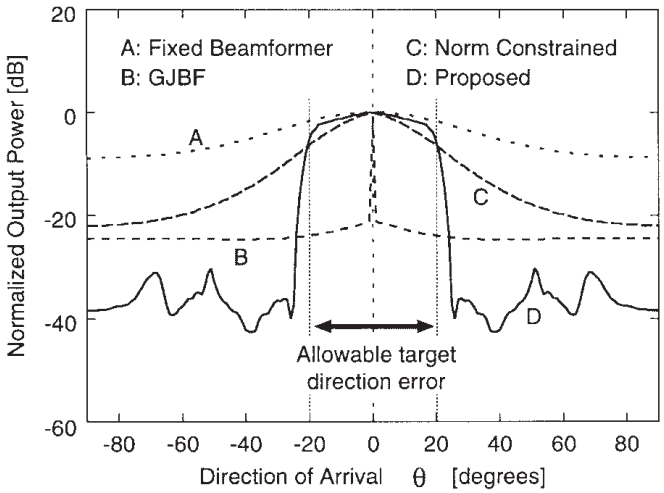


Abbildung 5.4.: Maximal erlaubter Fehler bei verschiedenen GJBF-Varianten

Der DSP kann mit jeder der Abtastraten aus Tabelle 5.1 genutzt werden. Somit ist die Entscheidung lediglich abhängig von dem Ziel, ein hohe Genauigkeit zu erhalten, was mit ei-

ner hohen Auflösung einhergeht. Abbildung 5.4 [BW01, S.99] zeigt den erlaubten Fehler für verschiedene Varianten des GJBF. Der implementierte GJBF, in dem die Daten später weiterverwendet werden, wird Typ D des GJBF sein, der einen Fehler von $\pm 20^\circ$ zulässt. Damit kann die Abtastrate schon jetzt auf mindestens $24kHz$ festgelegt werden, da der Fehler oberhalb dieser Abtastfrequenz nicht genauer als 20° detektiert werden kann.

5.2. Maximale Verzögerung zwischen den Mikrofonen

Um im nächsten Abschnitt die Länge des Fensters der KKF und damit auch die Anzahl der Abtastwerte, die der EDMA in seinem Puffer speichern soll, festzulegen, muss der maximal mögliche SDOA im Array bestimmt werden. In einem ULA tritt dieser zwischen den Mikrofonen M_0 und M_L dann auf, wenn die Quelle an den Maxima des Winkelbereichs steht (siehe Abb. 5.5).

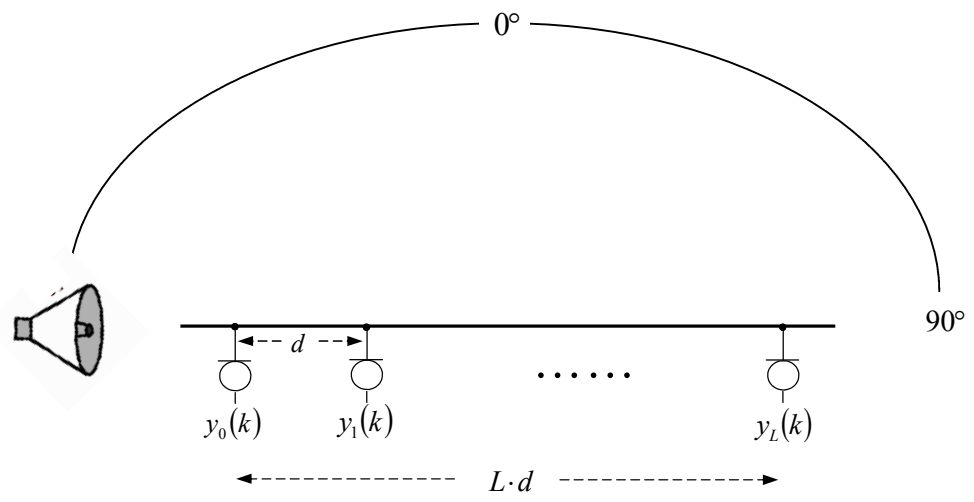


Abbildung 5.5.: Entstehung der maximalen Verzögerung im ULA

Ein ULA kann Winkel im Bereich von $\pm 90^\circ$ detektieren. $K_{max,ULA}$ ergibt sich aus Gleichung 5.1 durch Einsetzen von $\pm 90^\circ$ für den Winkel θ. Außerdem muss die Länge des Arrays $L \cdot d$ berücksichtigt werden:

$$\begin{aligned} K_{max,ULA} &= \frac{L \cdot d}{c} \cdot \sin(\pm\theta_{max}) \cdot f_A \\ &= \frac{L \cdot d}{c} \cdot \sin(\pm 90^\circ) \cdot f_A \\ &= \pm\frac{L \cdot d \cdot f_A}{c}. \end{aligned} \tag{5.3}$$

Somit ist auch dieser Parameter lediglich von der im System verwendeten Abstastfrequenz abhängig. Alle anderen Größen sind durch den Aufbau der Mikrofonarrays festgelegt. Tabelle 5.2 gibt eine Übersicht über mögliche $K_{max,ULA}$ bei gängigen Abtastfrequenzen im Audiobereich für $N = 8$:

Abtastrate (kHz)	SDOA
8	8
16	16
24	24
48	48
96	97

Tabelle 5.2.: Maximaler SDOA zwischen M_0 und M_7, ULA

Bei einem UCYA darf nicht mit der Verzögerung zwischen den Mikrofonen M_0 und M_L gerechnet werden. Die größtmögliche Verzögerung, die in einem UCYA auftreten kann, besteht zwischen zwei gegenüberliegenden Mikrofonen M_i und $M_{\frac{N}{2}-i}$, wenn die Quelle in einer Linie mit den beiden Mikrofonen liegt (s. Abb. 5.6).

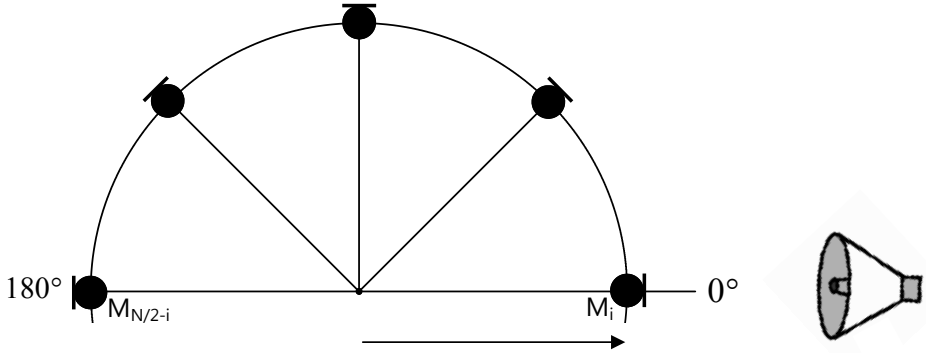


Abbildung 5.6.: Entstehung der maximalen Verzögerung im linearen Mikrofonarray

Für eine gerade Anzahl von Mikrofonen gilt dann immer $\psi_{\frac{N}{2}} = 180^\circ$. Dann kann die Quelle

bezogen auf M_i in einem Winkel von $\theta = 0^\circ$ angenommen werden. Aus Gleichung 3.5 folgt für $K_{max,UCYA}$

$$\begin{aligned} K_{max,UCYA} &= \frac{r}{c} \cdot [\cos(0^\circ) - \cos(0^\circ - 180^\circ)] \cdot f_A \\ &= \frac{r}{c} \cdot [1 - (-1)] \cdot f_A \\ &= \frac{2r}{c} \cdot f_A. \end{aligned}$$

$K_{max,UCYA}$ ist wie $K_{max,ULA}$ lediglich von der Abtastfrequenz abhängig. Tabelle 5.3 zeigt maximale SDOAs für ein UCYA mit denselben Abtastraten wie in 5.2:

Abtastrate (kHz)	Maximaler SDOA
8	3
16	6
24	9
48	19
96	39

Tabelle 5.3.: Maximaler SDOA zwischen den Mikrofonen, UCYA

Die Verwendung eines UCYA impliziert demnach kleinere zu detektierende Verzögerungen. Wie sich noch herausstellen wird, hat dies auch einen Einfluss auf die benötigte Rechenzeit des Algorithmus. Außerdem wird mit diesem Parameter die Länge der VDL eingestellt.
Es ist zu beachten, dass ein ULA mit den vorgestellten Algorithmen zur Winkeldetektion keinen Bereich von $\pm 90^\circ$ detektieren kann. Wie sich zeigen wird, existiert dieser Wunschbereich nur in der Theorie. Die Werte für $K_{max,ULA}$ werden dann noch kleiner.

5.3. Datenlänge im Algorithmus

Mit der Länge der Signale L_{sig}, die aus dem EDMA übernommen werden, um daraus die Richtung des Signals zu bestimmen, wird die Länge der FFT in den KKF (Fensterlänge) und die Rechenzeit des Gesamtalgorithmus beeinflusst. Außerdem muss die maximale Verzögerung $K_{max,ULA}$ bzw. $K_{max,UCYA}$ berücksichtigt werden. Die Verzögerung muss in die

Fensterlänge[13] hineinpassen, damit sie als Spitze in der KKF auftreten kann. Die Mindestlänge der KKF ergibt sich anschließend aus den Betrachtungen in Abschnitt 3.6. Dort wurde gezeigt, dass die Länge der FFT (L_{FFT}) der doppelten Länge der Eingangsdaten entspricht. Somit darf hier L_{sig} maximal halb so lang sein wie L_{FFT}. Aus Symmetriegründen muss die FFT außerdem eine Länge 2^m haben, somit ergibt sich $L_{sig,max} = 2^{m-1}$.

5.4. EDMA-Blocklänge und verfügbare Taktschritte

Die Anzahl der Daten, die vom EDMA aufgenommen werden (L_{EDMA}), bestimmt die Zeit, die für den gesamten Algorithmus zur Verfügung steht. L_{EDMA} muss mindestens so groß sein wie $L_{sig,max}$. Allerdings muss stets die Stationaritätseigenschaften von Sprache berücksichtigt werden (Unterabschnitt 3.7.1). Der verwendete DSP arbeitet mit einer Taktfrequenz von $f_{Takt} = 300MHz$. Die maximal zur Verfügung stehende Anzahl von Taktschritten (Zyklen) für den Algorithmus ergibt sich aus der Zeit, die der EDMA benötigt, um einen seiner Ping-Pong-Puffer zu füllen. Die Zeit t_{soll} in Sekunden ergibt sich dabei aus

$$t_{soll} = \frac{L_{EDMA}}{f_A}$$

und dargestellt in Zyklen:

$$K_{soll} = \frac{L_{EDMA}}{f_A} \cdot f_{Takt}. \tag{5.4}$$

Wird dieser Wert überschritten, ist eine Echtzeitfähigkeit unter der Definition, die hier für diese Arbeit getroffen wurde, nicht mehr gewährleistet, da in dem Fall Datenverluste auftreten würden.

[13] Die Fensterlänge ist der Zeitabschnitt, über den Korreliert wird, bzw. in der Korrelationsfolge die Anzahl der Samples N, die verglichen werden.

5.5. Anzahl der Kreuzkorrelationen

Beide verwendeteten Algorithmen basieren auf der Berechnung der räumlichen Korrelationsmatrix $\mathbf{R}_a$ für einen angenommenen TDOA bzw. SDOA. In der Praxis müssen dazu zunächst sämtliche KKFs zwischen den Mikrofonen und alle N AKFs der Mikrofonsignale berechnet werden. Anschließend werden unter Variation des SDOA bestimmte Werte aus den KKFs bzw. AKFs entnommen und in der Korrelationsmatrix übernommen. Die Anzahl der KKFs, die berechnet werden müssen, ist abhängig von der Anzahl der Mikrofone N, die verwendet werden und kann in der ersten Betrachtung auf $N^2 - N$ festgelegt werden. Unter Berücksichtigung der Symmetrieeigenschaften der KKF muss allerdings nur die Hälfte der Kreuzkorrelationen berechnet werden. Die Gesamtanzahl der Korrelationen M_{Korr} ist die Summe aus AKFs und KKFs:

$$\begin{aligned} M_{Korr} &= \frac{N^2 - N}{2} + N \\ &= \frac{N^2 + N}{2}. \end{aligned} \tag{5.5}$$

5.6. Gesamtüberblick

Wie in den letzten Abschnitten gezeigt, beeinflussen sich viele Parameter gegenseitig. Abb. 5.7 soll einen Überblick über die vorhandenen Beeinflussungen der Umgebungsparameter geben.

Die zentralen Parameter sind zum einen die EDMA-Blocklänge, zum anderen die Abtastfrequenz f_A. Mit einer hohen Abtastfrequenz wird eine hohe Winkelauflösung gewährleistet. Über Gleichung 5.4 bestimmt f_A aber auch direkt reziprokproportional die zur Verfügung stehenden Taktschritte. Damit bestimmt f_A die beiden Ziele *Winkelgenauigkeit* und *Rechtzeitigkeit* gegenläufig und wird sehr genau zu analysieren sein, damit das Optimum in der Implementierung gefunden werden kann.

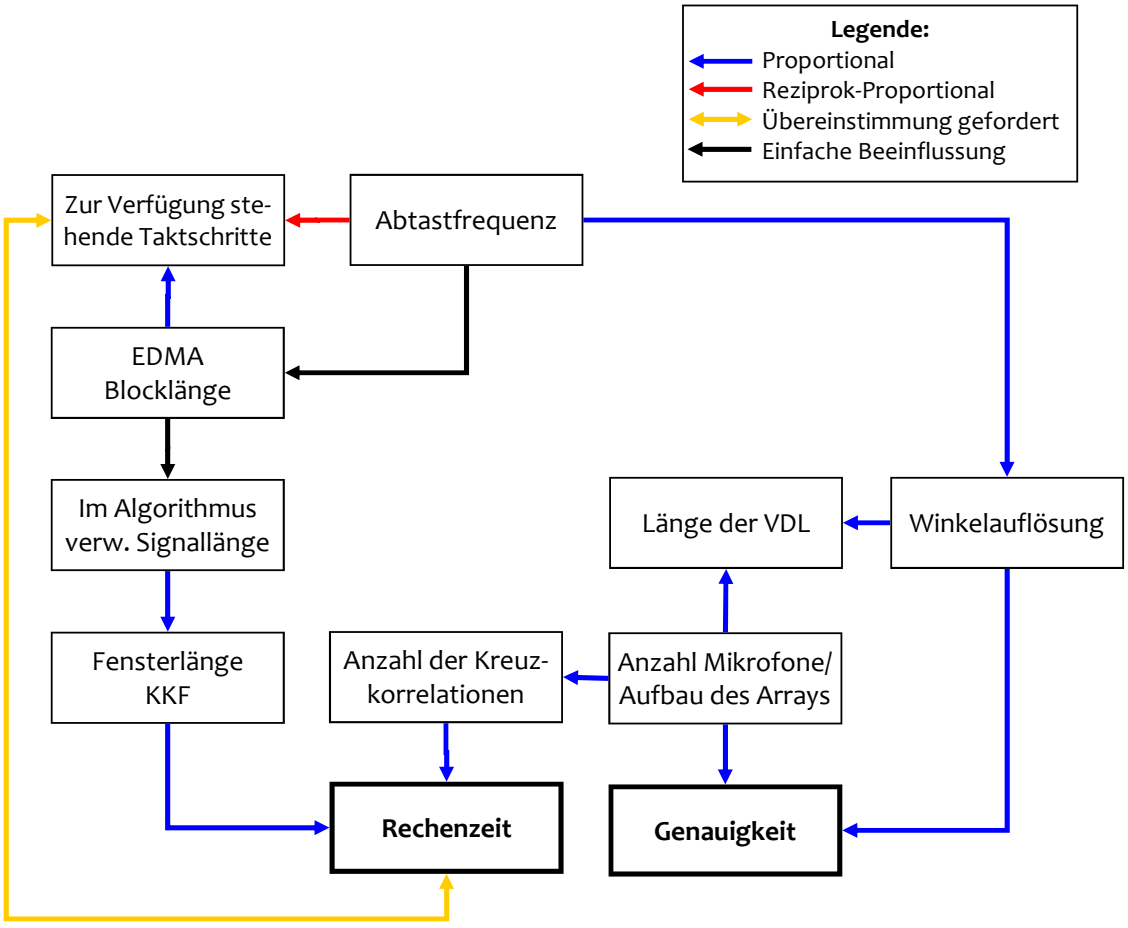


Abbildung 5.7.: Gegenseitige Beeinflussung der Systemparameter

6. Simulation des Algorithmus

Bevor die Theorie auf einem DSP implementiert wird, sollen die Algorithmen in einer Simulation unter MATLAB getestet werden. Das Ziel der Simulation ist es, die Algorithmen hinsichtlich Geschwindigkeit und Genauigkeit zu untersuchen und eine optimale Einstellung für die Umgebungsparameter zu finden. Des Weiteren werden Methoden getestet, die später auf dem DSP implementiert werden sollen. Den Abschluss dieses Kapitels bilden Simulationsergebnisse.
Werden MATLAB-Funktionen genannt, so sind diese in der Schriftart Roman dargestellt.

6.1. Interpolation der Eingangssignale

Der verwendete DSP kann mit jeder im Audiobereich gebräuchlichen Abtastrate betrieben werden. Aus Tabelle 5.1 folgt, dass die Winkelauflösung im Algorithmus proportional mit der Abtastrate steigt. Anzustreben ist also eine möglichst hohe Abtastrate, womit aber die Anzahl der zur Verfügung stehenden Taktschritte sinkt. Damit das Programm auch auf Plattformen läuft, die nicht mit jeder Abtastrate betrieben werden können, und außerdem die Möglichkeit geboten werden kann, auch niedrige Abtastraten zwecks Taktschrittgewinn zu gebrauchen, soll eine Interpolation (auch Upsampling oder Expansion) des Eingangssignals möglich sein. Eine Interpolation um den Wert B hat zur Folge, dass die Abtastrate f_A, mit der ein Eingangssignal abgetastet wird, nach der Interpolation künstlich auf $f_{Ao} = B \cdot f_A$ erhöht wurde. Eine Aufwärtsabtastung einer Folge $x(m)$ entspricht im Zeitbereich dem Einfügen von $B - 1$ Nullen zwischen den Abtastwerten von $x(m)$ (Abb. 6.1 [Har06, S.10]). Die Ausgangsfolge $y(n)$ kann dann durch den Bezug auf die erhöhte Abtastrate als Polyphasenkomponente mit beliebigen Phasenversatz p interpretiert werden. $y(n)$ ist gegeben durch

$$y(n) = x_p(n) = \begin{cases} x_p\left(\frac{n-p}{B}\right) & \forall n = mB + p \\ 0 & \forall n \neq mB + p \end{cases}, mit\ n \in \mathbb{Z}, L \in \mathbb{N}. \tag{6.1}$$

Mit dem Einfügen der Nullen wird das Spektrum des Signals nicht geändert, es wird lediglich die Frequenzachse neu skaliert.

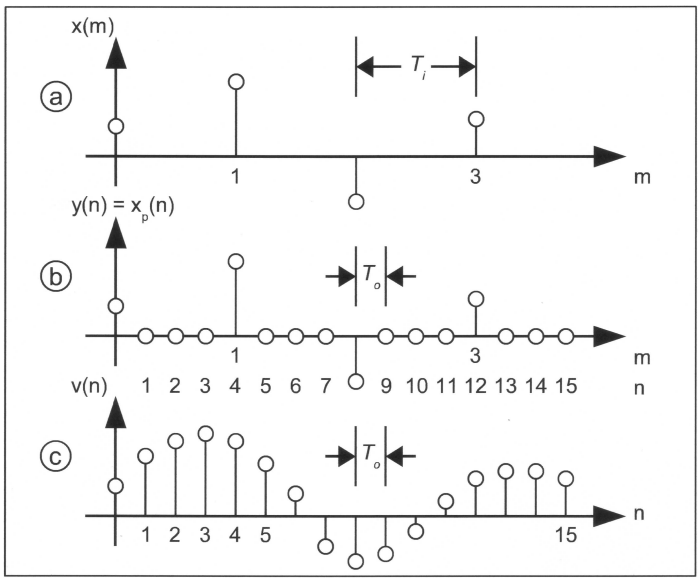


Abbildung 6.1.: Aufwärtsabtastung eines Signals mit $L = 4$: (a) Eingangssignal, (b) aufwärtsgetastetes Signal, (c) durch AIF korrekt interpoliertes Ausgangssignal

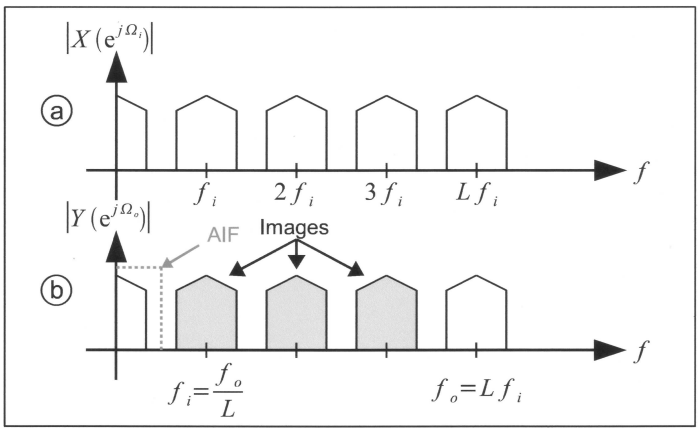


Abbildung 6.2.: Spektrale Darstellung einer Aufwärtsabtastung mit $L = 4$: (a) Betragsspektrum des Eingangssignals, (b) Betragsspektrum des Ausgangssignals

Wie in Abbildung 6.2 [Har06, S.10] verdeutlicht, entstehen dabei $L - 1$ Bilder (Images), die durch einen Anti-Imaging-Filter (AIF) entfernt werden müssen. Die Entfernung der Bilder entspricht im Zeitbereich einer Interpolation der eingefügten Nullen.
Die vollständige Interpolation setzt sich aus der Kaskade aus Aufwärtsabtastung (Einfügen von Nullen) und anschließendem AIF zusammen, wie in Abbildung 6.3 gezeigt.

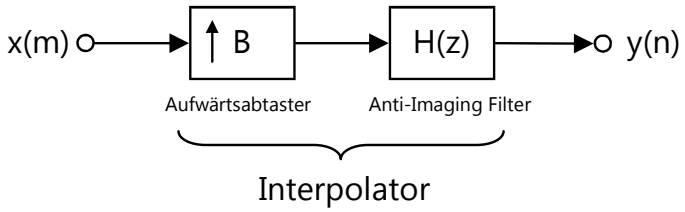


Abbildung 6.3.: Vollständiger Interpolator

$H(z)$ soll als Finite Impulse Response (FIR)-Filter realisiert werden[14]. Damit hat $H(z)$ die Übertragungsfunktion (hier mit einem Filtergrad von $G = 8$)

$$H(z) = b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + b_3 \cdot z^{-3} + \ldots + b_8 \cdot z^{-8}. \tag{6.2}$$

Durch die sogenannte Polyphasenzerlegung kann $H(z)$ nun in eine parallele Anordnung einzelner FIR-Filter dargestellt werden. Durch Ausklammern von z^{-1} in 6.2 wird $H(z)$ durch $B = 3$ Polyphasenkomponenten dargestellt:

$$\begin{aligned} H(z) = {} & b_0 + b_3 z^{-3} \\ & + (b_1 + b_4 z^{-4}) \cdot z^{-1} \\ & + (b_2 + b_5 z^{-3}) \cdot z^{-2}. \end{aligned} \tag{6.3}$$

Die Polyphasenfilterkomponenten des zerlegten Filters hängen nicht von z, sondern von z^3, oder allgemein von z^B ab. Wie in Gleichung 6.3 zu erkennen ist, wurde der ursprüngliche FIR-Filter achter Ordnung in drei einzelne Filter dritter Ordnung überführt. Eine solche Struktur wird als Polyphasenfilter bezeichnet und hat den Vorteil gegenüber normalen FIR-Filter, dass sie effizienter zu implementieren sind. Aus Gleichung 6.3 wird das Blockschaltbild für einen Polyphasenfilter abgeleitet, mit dem die Interpolation der Eingangsdaten vollzogen wird ([KS10]).

[14] Generell ist aber auch eine Realisierung als Infinite Impulse Response (IIR)-Filter möglich.

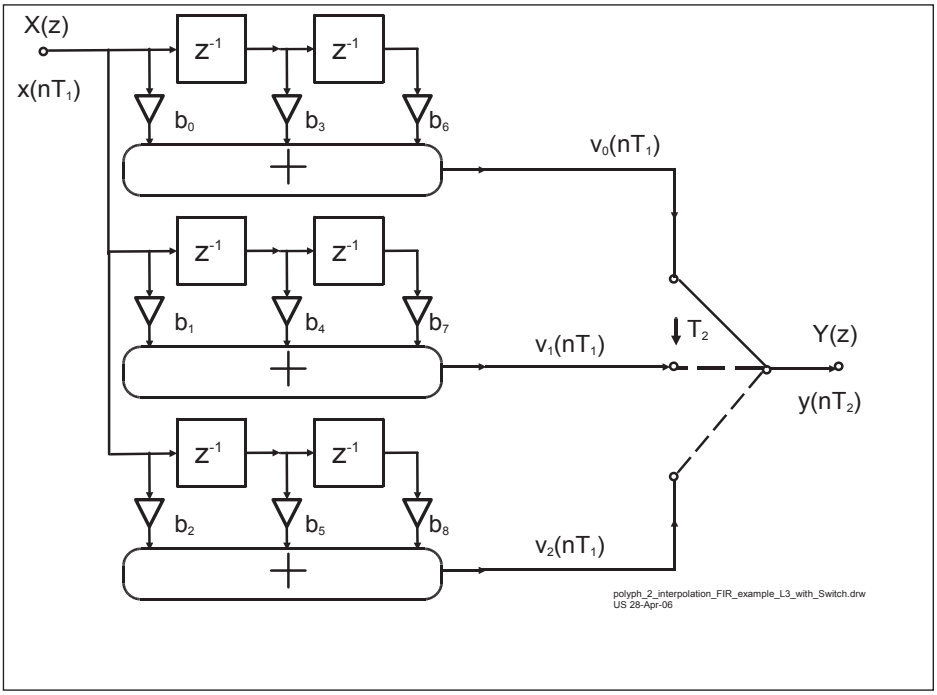


Abbildung 6.4.: Effiziente Struktur eines Polyphasenfilters mittels FIR-Filter zur Interpolation

Die Berechnung der Filterkoeffizienten, sowie deren Sortierung auf die einzelnen Teilfilter wird von der MATLAB-Funktion PolyphasenfilterDesign übernommen, die im Anhang beigelegt ist. Der Funktion müssen unter anderem die ursprüngliche Abtastrate und der Interpolationsfaktor gegeben werden, aus denen zunächst ein Tiefpassfilter und anschließend die FIR-Teilfilter berechnet werden.
Das Tiefpassfilter wird über

Code 6.1: Berechnung des Tiefpassfilters in MATLAB

```
1    [N_FIR,fo,mo,w] = firpmord( [3400 4000], [1 0], [0.01 0.001], fA*L );
2  h = firpm(N_FIR,fo,mo,w);
```

berechnet. Die MATLAB-Funktion firpmord approximiert ein FIR-Filter mit gewünschten Eigenschaften. Mit `[3400 4000]` werden die Eckfrequenzen des Durchlass- und Sperrbereich festgelegt. `[1 0]` sind die Amplituden im Durchlass- und Sperrbereich, `[0.01 0.001]` ist die Größe der Überschwinger in den jeweiligen Bereichen. `fA*L` ist die Abtastfrequenz, auf welcher das Filter arbeitet. Mit einer Abtastfrequenz von 48kHz, auf der das Filter arbeitet, ergibt sich eine Ordnung von 204. Abbildung 6.5 zeigt den Frequenzgang mit den erwähnten Eigenschaften.

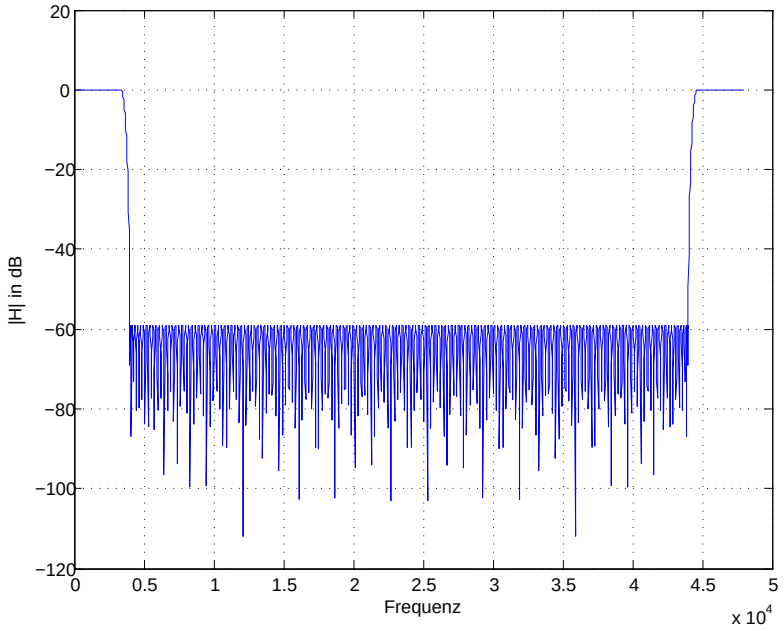


Abbildung 6.5.: Frequenzgang eines FIR-Filters mit „PolyphasenfilterDesign“ erstellt

6.2. Kreuzkorrelationen

MATLAB liefert die Funktion xcorr, mit der die KKF zweier Signale auf sehr einfachem Weg berechnet werden können. Um schon in der Simulation die spätere Implementierung abzubilden, werden die KKFs nach der in Abschnitt 3.6 vorgestellten Methode berechnet. Wird diese Methode verwendet, muss eine Besonderheit beachtet werden:
Es sei die Pseudofolge $[1, 2, 3, 4]$ mit der Länge $l = 4$ gegeben, deren AKF einmal über xcorr und einmal über die Faltung (mit Zeropadding auf $2 \cdot l = 8$) berechnet wird:

$$\varphi_{xcorr} = \begin{matrix} 4.0000 & 11.0000 & 20.0000 & 30.0000 & 20.0000 & 11.0000 & 4.0000 \end{matrix}$$
$$\varphi_{Faltung} = \begin{matrix} 30.0000 & 20.0000 & 11.0000 & 4.0000 & -0.0000 & 4.0000 & 11.0000 & 20.0000 \end{matrix} .$$

Es ist zu erkennen, dass die Ergebnisse bis auf eine zyklische Verschiebung identisch sind. Durch den Befehl fftshift wird $\varphi_{Faltung}$ in

$$\varphi_{Faltung} = \begin{matrix} -0.0000 & 4.0000 & 11.0000 & 20.0000 & 30.0000 & 20.0000 & 11.0000 & 4.0000 \end{matrix}$$

umsortiert. Somit ist der einzige Unterschied zwischen den beiden Verfahren, dass die Fal-

tung eine Null im ersten Element besitzt. Wird von der unverschobenen KKF ausgegangen, bedeutet dies, dass Arrayzugriffe, mit einem Offset von +1 gemacht werden müssen, wenn quasi auf ein Element auf der „rechten“ Seite der Korrelation zugegriffen werden muss[15].

6.3. Medianfilter

Um große Winkelsprünge in der Berechung zu vermeiden, wird ein Medianfilter eingebaut, der die Aufgabe hat, den Verlauf des Winkels zu glätten. Die Funktionsweise des Medianfilters ist in Abbildung 6.6 veranschaulicht (Medianfilter mit Ordnung $M_{Median} = 3$). Dort werden von einer Eingangsfolge jeweils der aktuelle Wert und die zwei letzten Werte entnommen und sortiert. Der mittlere Sortierwert wird anschließend als der zuletzt gemessenen angenommen. Die Verwendung eines Medianfilters impliziert also, dass die Winkel quasi in der Vergangenheit verändert werden.

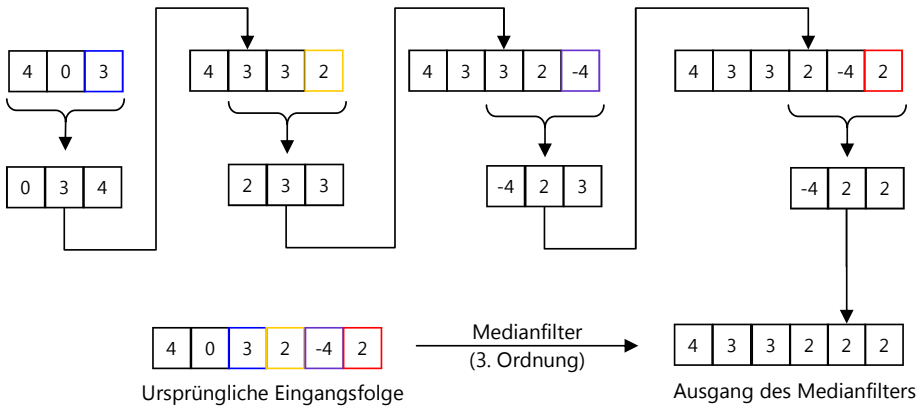


Abbildung 6.6.: Filterung einer Eingangsfolge mittels Medianfilter

Die Filterung kann erst dann starten, wenn soviele Werte in der Eingangsfolge liegen, wie die Ordnung des Filters beträgt. Anschließend werden immer drei Werte sortiert und der mittlere Wert der Sortierung als der letzte in der Folge angenommen. Wie zu sehen ist, werden die Ausreißer 0 und -4 in der ursprünglichen Eingangsfolge herausgefiltert und ein glatter Verlauf erreicht.
Die Länge des Medianfilters muss immer ungerade sein. Dies folgt aus dem Prinzip des Filters, nach dem das mittlere Element einer Sortierung ausgewählt werden muss. Mit der Länge des Filters wird einerseits eingestellt, welches Element in der Vergangenheit der Folge

[15] Eine Alternative, den Offset zu umgehen, wäre es, die FFT-Länge auf $2 \cdot l - 1$ einzustellen. Dann würde die Null in der Korrelation nicht auftreten.

angepasst wird, anderseits wie viele aneinanderhängende Ausreißer zu erkennen sein sollen. Beide Eigenschaften folgen der Gleichung $\frac{M_{Median}-1}{2}$, wobei M_{Median} die Ordnung des Filters ist.

6.4. Simulationsergebnisse und Einfluss der Parameter

Vor der Implementierung des Systems muss getestet werden, ob der Algorithmus den gestellten Anforderungen genügt. Außerdem kann in einer Simulation der Einfluss von Parametern wie z.B. der Abtastrate überprüft und das Verhalten der Algorithmen bei vorhandenem Rauschen getestet werden. Die Simulation ist im Anhang "\03_Matlab\“ unter dem Namen Simulation_SLP_MCCC_4V1.m beigelegt.
In einer Eingabemaske wird der Benutzer nach den Einstellungen für den Testlauf abgefragt. Die Einstellungen, die vorgenommen werden können, sind:

- N: Anzahl der Mikrofone im Array,
- f_A: Die ursprüngliche Abtastrate, mit der die Daten aufgenommen werden (EDMA-Abtastrate),
- B: Interpolationsfaktor, um den die Abtastrate erhöht werden soll,
- die Struktur des Arrays,
- L_{FFT}: Länge der FFT,
- Art des Eingangssignals,
- Durchführung der Energieberechung und
- Benutzen des Medianfilters.

Aus den ersten vier Angaben wird die EDMA-Datenlänge, sowie die Mindest-FFT-Länge ermittelt. Als Eingangssignal kann gewählt werden zwischen einem Sprachsignal in Form einer wav-Datei oder einem synthetisch erstellten Signal. Im Falle eines synthetischen Signals wird die Richtung, aus der das Signal kommen soll, sowie der Signalstörabstand (SNR) abgefragt.
Für die Beurteilung der Ergebnisse dient die Berechnung des mittleren quadratischen Fehlers (Streuung) des Winkels über die Formel

$$\varsigma_\theta = \sqrt{\frac{1}{H}\sum_{h=1}^{H}\left|\tilde{\theta}_h - \theta\right|^2}, \tag{6.4}$$

wobei H die Anzahl der Messungen, $\tilde{\theta}_h$ der gemessene Winkel in der i^{ten} Iteration und θ der eingestellte Winkel ist. H wird, wenn nicht anders erwähnt, auf 100 eingestellt.
Im Folgenden sollen die Algorithmen zunächst mit synthetischen Signalen getestet werden. Wenn nicht anders erwähnt, werden die Energieberechnung und der Medianfilter nicht verwendet.

6.4.1. Verifikation des Algorithmus

Zunächst wird die generelle Funktion der Simulation getestet. Die Einstellungen für diesen Testlauf sind wie folgt: $N = 8$, $f_A = 48kHz$, $B = 1$, $L_{FFT} = 256$. Abbildung 6.7 zeigt das Ergebnis der SLP und MCCC für eine Quelle bei 32° mit einem SNR der Signale von 5dB und einem ULA.

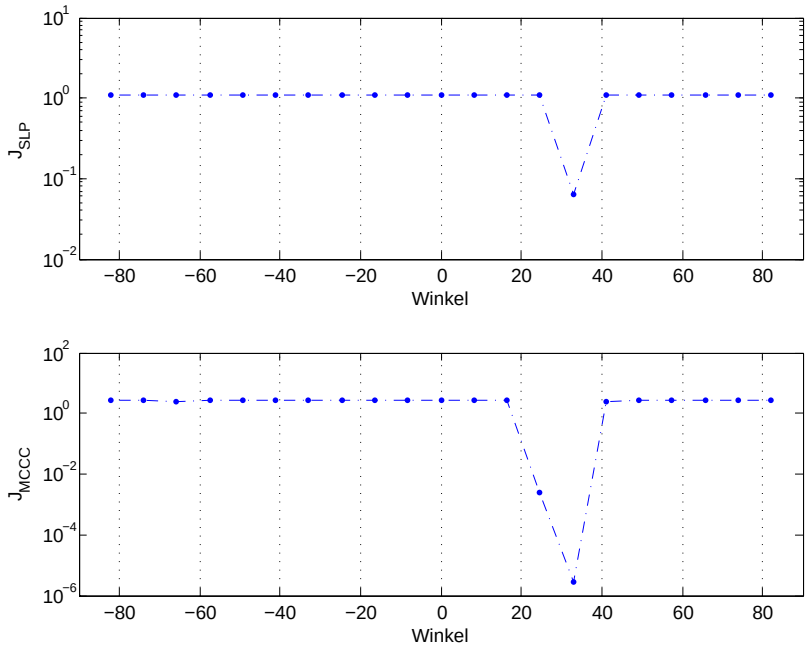


Abbildung 6.7.: Verifikation des Algorithmus: Quelle bei 32°, 5dB SNR, ULA

Die Quelle wird bei knapp 32° erkannt. Auffällig ist schon hier, dass der Wert von J in der MCCC deutlich niedriger ist als in der SLP. Das lässt auf eine niedrigere Störanfälligkeit der MCCC schließen.
Anschließend wird der Bereich verifiziert, den das ULA detektieren kann. Theoretisch müsste der Algorithmus einen Bereich von $\pm 90°$ abdecken. Wie in Abbildung 6.8 zu sehen, steigen

die Abweichungen am äußeren Rand des Bereichs sehr stark an. Somit können weder mit der SLP (oben), noch mit der MCCC (unten) die kompletten $\pm 90°$ abgedeckt werden. Der GJBF blendet die Störgeräusche außerhalb von $\pm 20°$ aus. Wie in der Abbildung zu sehen, wäre im Gesamtsystem somit ein maximal zu detektierender Winkel von $\pm 75°$ erlaubt, wenn keine Ausgleichsgerade des Fehlers eingebaut würde (s. Abschnitt 8.2.2).

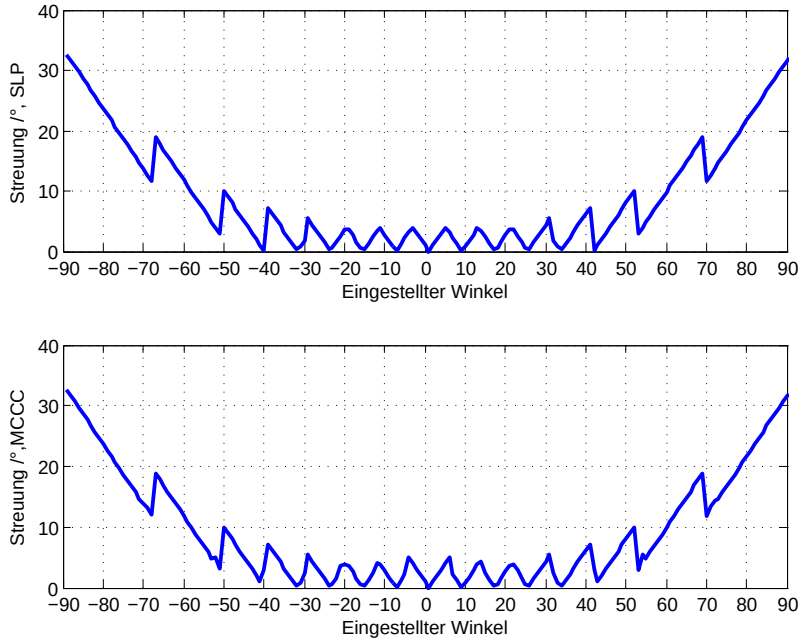


Abbildung 6.8.: Winkelbereich, SLP (oben) und MCCC (unten), ULA

Abbildung 6.9 zeigt das Ergebnis mit $L_{FFT} = 64$ und einem UCYA (die übrigen Einstellungen bleiben gleich).
Auch hier kann auf eine niedrigere Störanfälligkeit der MCCC geschlossen werden. Auffällig ist außerdem die harmonische Anordnung lokaler Minima. Die Minima liegen bei der MCCC gut erkennbar bei einem Abstand von etwa 45°. In der Fehlerfunktion der SLP weisen zwei Minima einen ähnlich kleine Wert auf wie das Minimum beim Winkel der Quelle. In weiteren Messungen mit einem SNR von 5dB hat sich gezeigt, dass diese Minima selten sogar kleinere Werte annehmen können als das beim eigentlichen Winkel.
Um die Störanfälligkeit zu überprüfen, soll im nächsten Unterabschnitt das Rauschen variiert und der Einfluss auf die Algorithmen getestet werden.

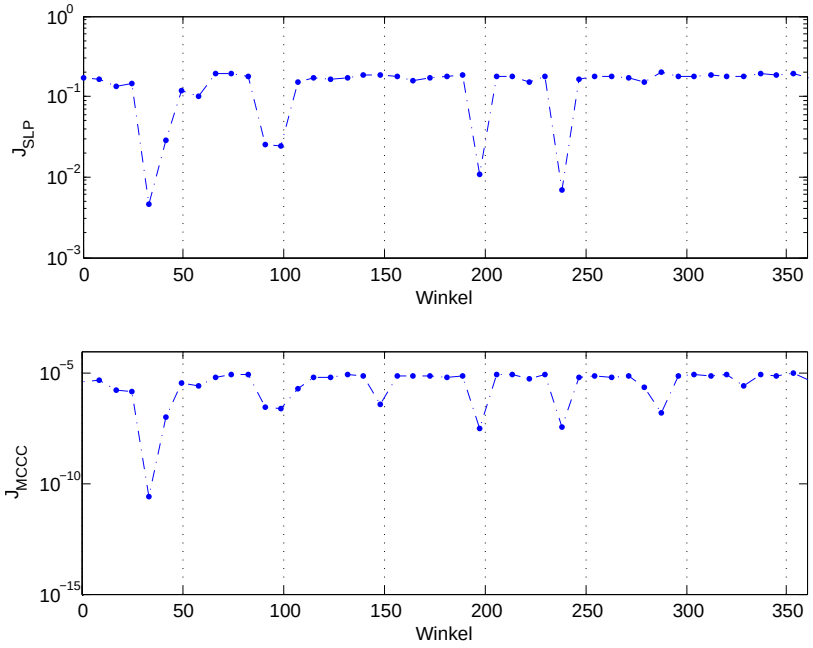


Abbildung 6.9.: Verifikation des Algorithmus: Quelle bei 32°, 5dB SNR, UCYA

6.4.2. Einfluss von Rauschen in der Umgebung

Es wird wieder mit $N = 8$, $f_A = 48kHz$, $B = 1$ und $L_{FFT} = 256$ für das ULA und $L_{FFT} = 64$ beim UCYA simuliert. Diese FFT-Längen entsprechen der Mindestlänge, wie sie in Abschnitt 5.3 festgelegt wurden. Die Quelle liegt bei 32°, wobei nun der SNR variiert wird. Abbildung 6.10 zeigt das Ergebnis der Algorithmen für verschiedene SNR im ULA, Abbildung 6.11 im UCYA. Es wurde hier nur der Bereich zwischen 1dB und 5dB SNR dargestellt, da oberhalb von 5dB keine Verbesserung mehr auftritt. Die Annahme, die Methode der MCCC wäre weniger Störanfällig kann nicht bestätigt werden. Beide Algorithmen liefern schwache Ergebnisse unterhalb von 3dB SNR. Unter Verwendung der zirkularen Anordnung liefert die MCCC bessere Ergebnisse, da die harmonischen Minima in der SLP nun sehr häufig größer sind, als das Minimum beim Quellenwinkel. Mit der SLP könnte das zirkulare Array auch mit hohen Störabständen unsichere Ergebnisse liefern. Dies wird in Abbildung 6.12 verdeutlicht. Hier laufen die Algorithmen über einen längeren Zeitraum. Zu erkennen ist deutlich, dass die Methode der SLP viele Ausreißer hat, wobei diese genau an der Stelle liegen, an der das größte harmonische Minimum in Abbildung 6.10 zu sehen ist. Die Ausreißer können durch die Verwendung eines Medianfilters oder längeren FFT-Längen reduziert werden.

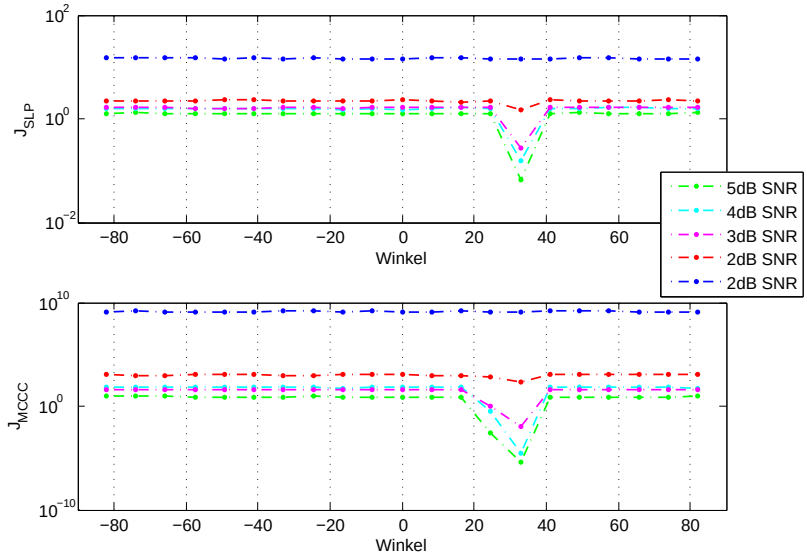


Abbildung 6.10.: Einfluss des SNR auf die Winkelerkennung, oben SLP, unten MCCC, ULA

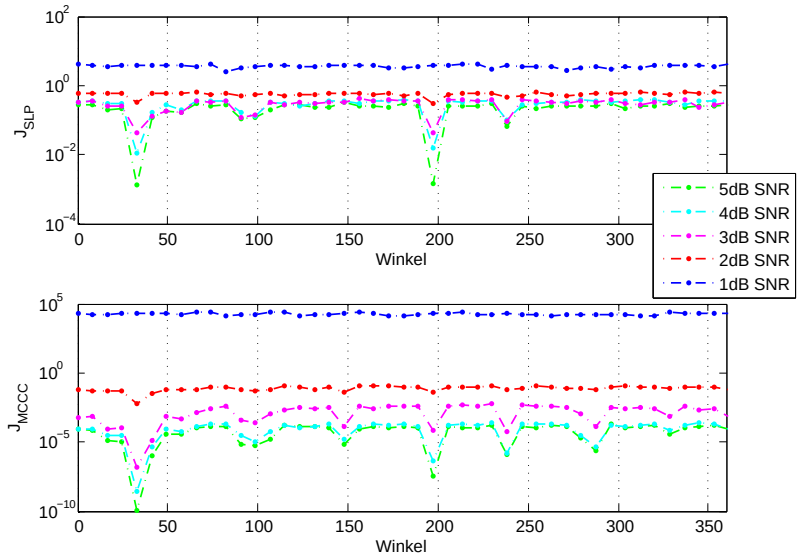


Abbildung 6.11.: Einfluss des SNR auf die Winkelerkennung, oben SLP, unten MCCC, UCYA

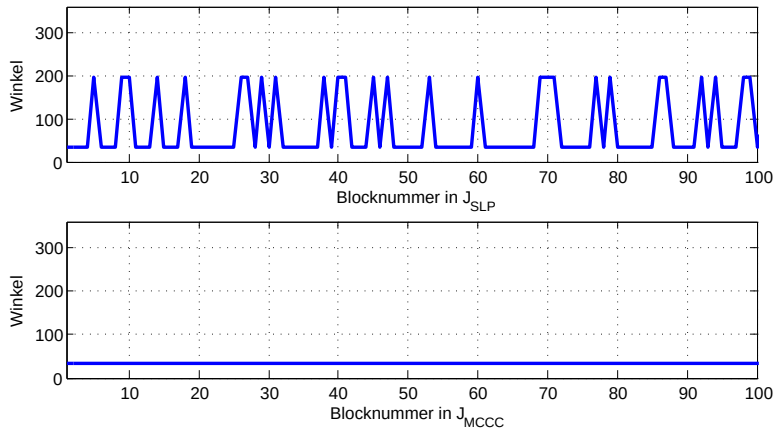


Abbildung 6.12.: Angenäherter Winkel über 100 EDMA-Blöcke, UCYA

6.4.3. Einfluss der Aufwärtsabtastung

Da die Rechenzeit reziprokproportional zu der verwendeten Abtastrate ist, diese jedoch proportional mit der Winkelgenauigkeit steigt, muss es die Möglichkeit geben, eine Aufwärtsabtastung der Eingangswerte durchführen zu können. Zu diesem Zweck wurde ein Polyphasenfilter in den Algorithmus eingebaut, der je nach Umgebung verwendet werden kann. Es soll nun der Einfluss dieses Polyphasenfilters auf die Fehlerfunktionen ermittelt werden.
Hierzu wird eine Umgebung mit variabler Abtastrate simuliert, wobei diese mit einem Polyphasenfilter der Ordnung 204 auf $48kHz$ aufwärtsabgetastet wird. Die Quelle liegt bei -38° und wird mit einem variablen SNR simuliert. Abbildung 6.14 zeigt ς_θ der SLP, Abbildung 6.13 ς_θ der MCCC für ein ULA.

Klar zu erkennen ist, dass die MCCC bis $f_A = 12kHz$ etwa auf einem Niveau verbleibt, während die Fehler in der SLP leicht ansteigen. Auffällig ist der Anstieg des Fehler bei der MCCC mit $8kHz$ und $B = 6$. Sobald der SNR etwa 5dB übersteigt, wächst der Fehler auf 40° an. Die MCCC findet in der Fehlerfunktion dann bei 0° das Minimum (siehe Abbildung 6.15.

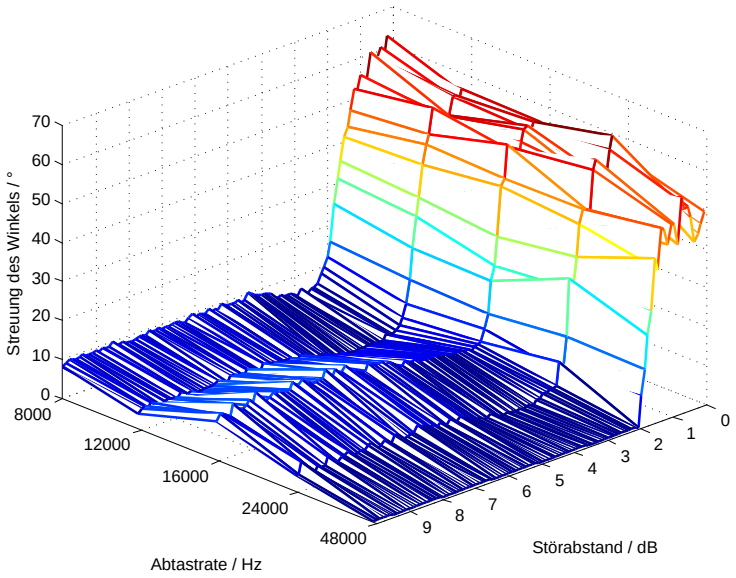


Abbildung 6.13.: ς_θ der SLP in Abhängigkeit der Abtastrate und des SNR, ULA, Abtastrate jeweils auf 48kHz interpoliert

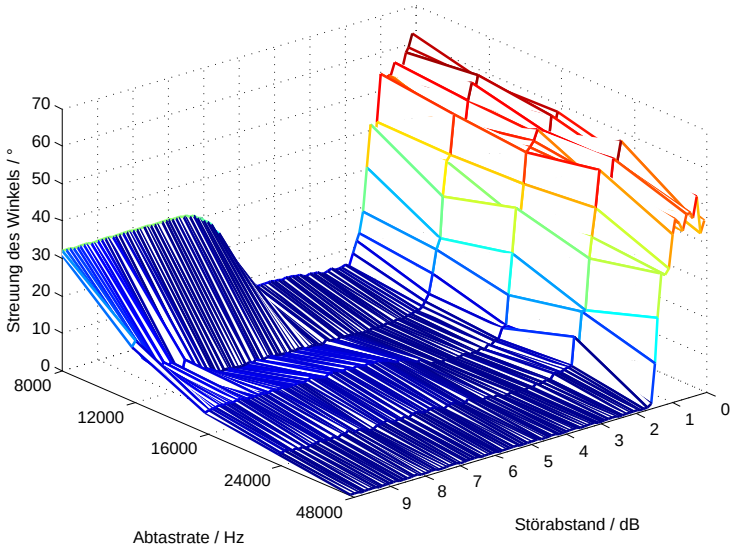


Abbildung 6.14.: ς_θ der MCCC in Abhängigkeit der Abtastrate und des SNR, ULA, Abtastrate jeweils auf 48kHz interpoliert

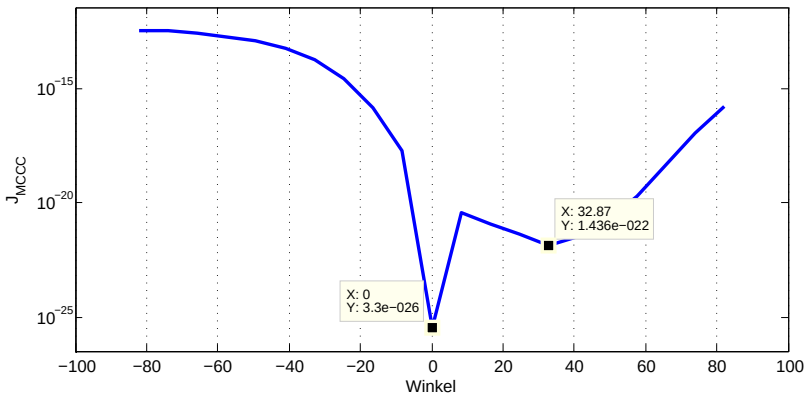


Abbildung 6.15.: Fehlerhafte Berechnung des Winkels in der MCCC, $f_A = 8kHz$, $B = 6$, $SNR = 9dB$

6.4.4. Test mit aufgenommenen Sprachsignalen

In der späteren Anwendung werden keine synthetischen Signale verwendet, sondern Sprachsignale in Echtzeit aufgenommen. Das Verhalten des Algorithmus wird hier anhand eines Sprachsignals gezeigt, das in schalldichter Umgebung mit einem ULA bei -50° mit 8kHz aufgenommen wurde. Abbildung 6.16 zeigt das Ergebnis der Simulation mit sechsfacher Interpolation auf 48kHz, einer FFT-Länge von 1024 und einem Medianfilter der Ordnung drei. Es wurde über 200 Blöcke simuliert. Wie zu erkennen ist, hat die MCCC in mehr Blöcken den richtigen Winkel extrahiert als die SLP. Allerdings haben beide Algorithmen auch in Blöcken die stimmhaft zu sein scheinen, nicht immer den richtigen Winkel herausfinden können. Abbildung 6.17 zeigt die gleiche Simulation ohne Medianfilter. Zwischen dem 15. und 30. Block ist zu erkennen, dass der Medianfilter die Ausreißer gut detektiert und somit einen glatten Verlauf des Winkels erstellt. Die FFT-Länge von 1024 wurde vom Programm auf Grundlage der Vorgaben hinsichtlich Winkelbereich und Auflösung berechnet. Wird die Länge auf 8192 Punkte erhöht, so ergeben sich die Winkelverläufe in Abbildung 6.18 und 6.19. Durch eine Verlängerung der FFT - und damit auch der Signallänge, die analysiert wird - könnte bei einem Sprachsignal also eine deutlich bessere Winkelerkennung erreicht werden.

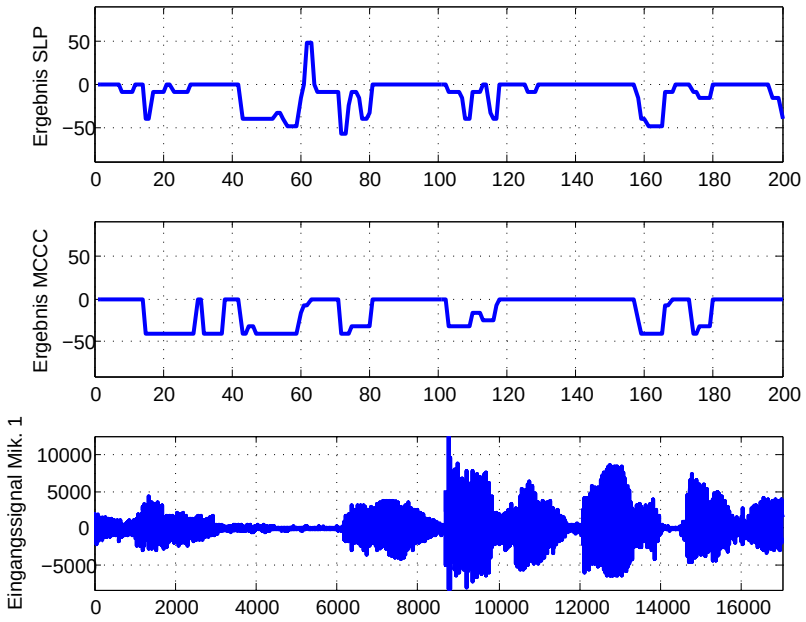


Abbildung 6.16.: Berechnung des Winkels eines Sprachsignals bei -50°, mit Medianfilter, FFT-Länge 1024, ULA

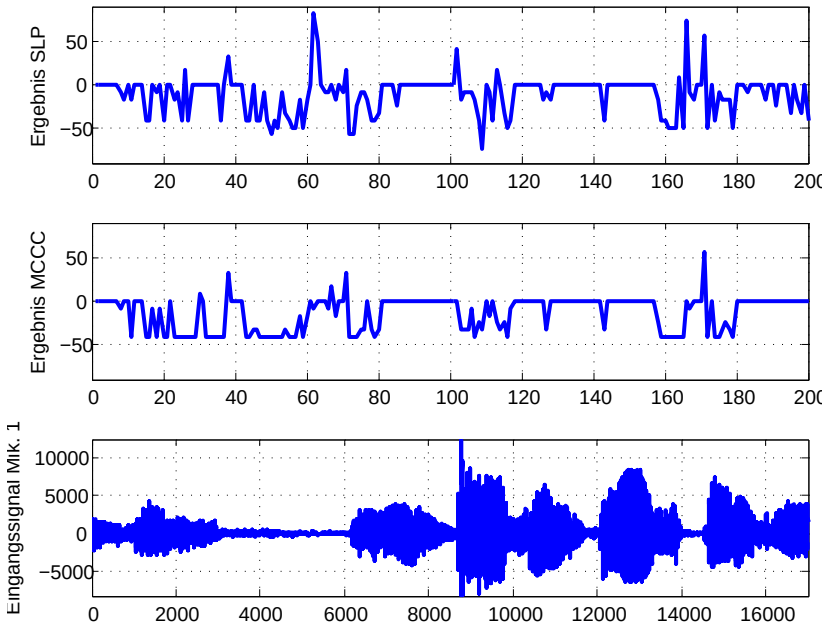


Abbildung 6.17.: Berechnung des Winkels eines Sprachsignals bei -50°, ohne Medianfilter, FFT-Länge 1024, ULA

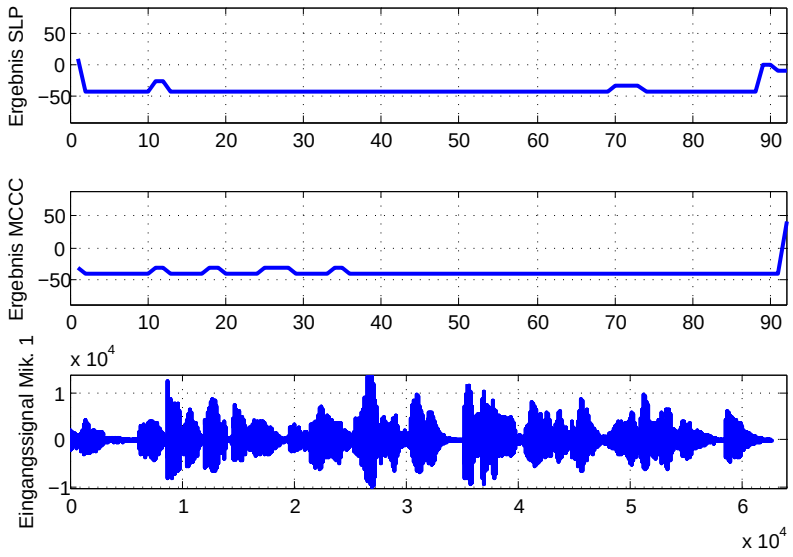


Abbildung 6.18.: Berechnung des Winkels eines Sprachsignals bei -50°, mit Medianfilter, FFT-Länge 8192, ULA

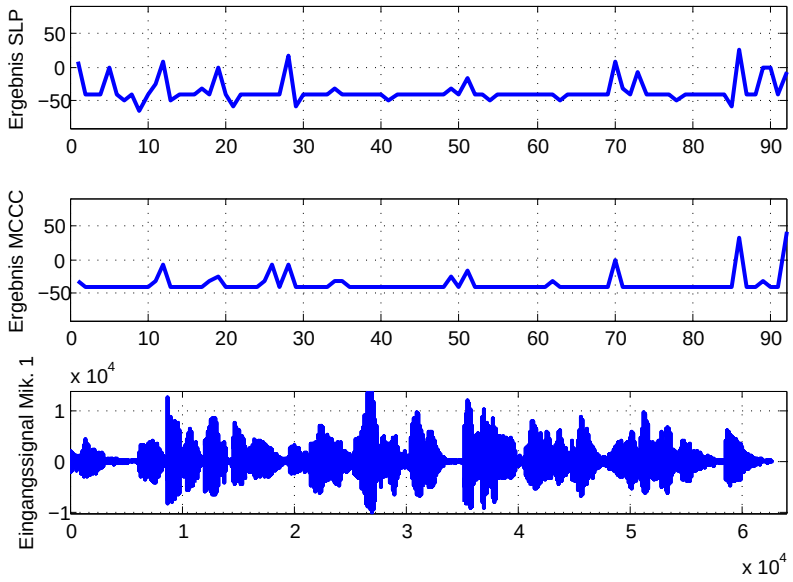


Abbildung 6.19.: Berechnung des Winkels eines Sprachsignals bei -50°, ohne Medianfilter, FFT-Länge 8192

7. Implementierung auf dem DSP

Nachdem bewiesen wurde, dass die Simulation gute Ergebnisse liefert, soll nun gezeigt werden, wie die Implementierung in der Programmiersprache C vollzogen wurde. Die Simulation unter MATLAB untersteht idealen Bedingungen hinsichtlich Geschwindigkeit des Algorithmus, Genauigkeit der Zahlen und Speicherplatz. Der DSP hat in diesen Bereichen klare Einschränkungen, die beachtet werden müssen. Funktionen müssen eventuell optimiert werden, damit die zur Verfügung stehenden Taktschritte eingehalten werden können.
Der DSP kann sowohl in Fließkomma- als auch in Festkommaarithmetik betrieben werden. Für eine einfachere Implementierung der verwendeten Algorithmen wird das Programm in Fließkommaarithmetik geschrieben[16].
Im ersten Abschnitt wird der Aufbau der Hauptfunktion `Main_Richtungserkennung.c` erläutert, woraufhin die Makros, die für den Algorithmus eingestellt weden können, gezeigt werden. Anschließend werden die Umsetzung der Methoden und mathematischen Grundlagen im Einzelnen gezeigt und die Ergebnisse vorgestellt.
Für dieses Kapitel werden folgende Vereinbarungen zur Darstellung von codespezifischen Ausdrücken getroffen:

- Codeauszüge werden in abgesetzten Codeumgebungen dargestellt,
- C-Ausdrücke im Text werden in der Schriftart `Courier New` angegeben,
- weiterhin werden MATLAB-Funktionen in der Schriftfamilie Roman dargestellt.

Dieses Kapitel enthält neben dem Implementierungsansatz auch eine genauere Dokumentation der Programmierung, damit nachfolgende Programmierer eine bessere Einleitung für Aufbauarbeiten erhalten.

[16] Aus dieser Vereinfachung resultiert natürlich ein größerer Speicherplatzbedarf des Programms.

7.1. Aufbau der Hauptfunktion

Die Datei `Main_Richtungserkennung_3V4.c` ist die Hauptdatei, in der das Hauptprogramm `main`, das den Ablauf vorgibt und steuert, gespeichert ist. Grundlage der Hauptdatei ist das Demoprogramm `pcm3003` von D.SignT, das alle Grundfunktionen für die Nutzung des EDMA in Ping-Pong-Technik und eines 8-kanaligen Mikrofonarrays enthält. Die wesentlichen Bestandteile der `main` sind:

- Initialisierungen für Methoden des Algorithmus
- Initialisierung des DSP-Moduls
- Zurücksetzen der Tochterplatine PCM3003
- Initialisierung der DAU-Blöcke
- Einrichtung der McBSPs und des EDMA
- Algorithmus zur Sprecherlokalisierung.

Der Algorithmus läuft in einer Endlosschleife, deren Beendigung nicht vorgesehen ist. Das Flussdiagramm in Abbildung 7.1 zeigt den Ablauf des Hauptprogramms.

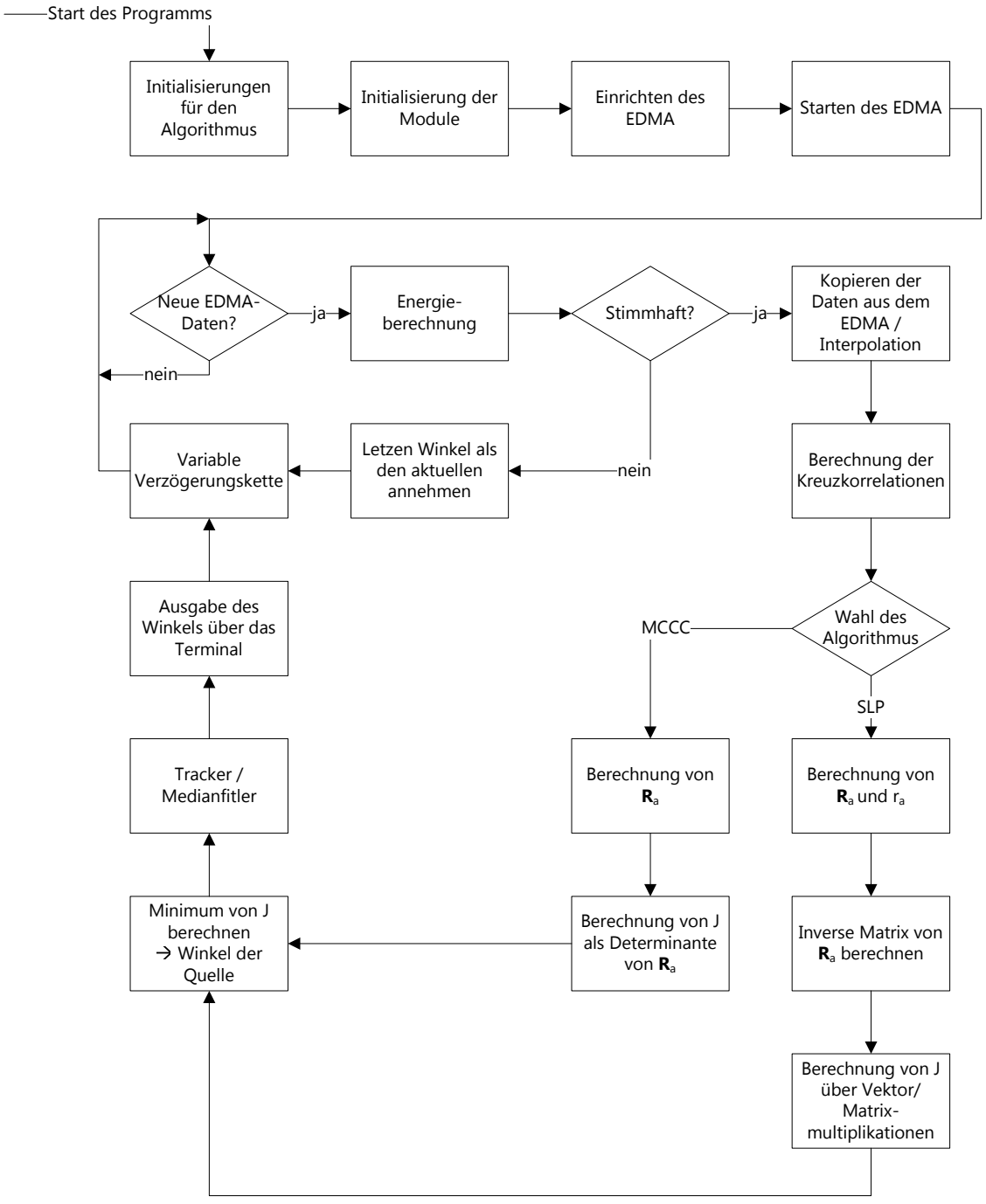


Abbildung 7.1.: Ablauf des Hauptprogramms auf dem DSP

7.2. Headerdateien des Projekts

Für eine bessere Übersicht sind sämtliche Makros und Funktionsprototypen in eigenen Headerdateien abgelegt.

7.2.1. Datei `defines.h`

In der Defintionsdatei `defines.h` können Umgebungsparameter und Grundeinstellungen für den Ablauf durch Ein- und Auskommentieren von Makros verändert werden. Des Weiteren bestimmen diese Makros teilweise die Einbindung von Unterfunktionen. Folgende Einstellungen können vollzogen werden:

Auswahl des Algorithmus Die Makros `USE_SLP` und `USE_MCCC` bestimmen den verwendeten Algorithmus.

Auswahl der Umgebung Durch Einkommentieren des Makros `SIMULATION` werden synthetisch erzeugte Signale verwendet. In dem Fall muss außerdem angegeben werden, aus welcher Richtung die synthetische Quelle liegen soll (`DOA`). `REALTIME` bestimmt, dass Daten vom ADU verwendet werden sollen.

Einbindung der Energieberechnung Durch Einkommentieren von `USEDECIDEVOICE` wird die Energieberechnung vom Sprachsignal verwendet.

Einbindung des Medianfilters Durch `USEMEDIAN` wird die Benutzung des Medianfilters zur Glättung des Winkelverlaufs bestimmt.

Einbindung der VDL `USEDELAYLINE` bindet die variable Verzögerungskette in das Programm ein.

Ausgabe der Winkel Die Winkel können über eine Serielle Schnittstelle ausgegeben werden. Hierzu muss `TERMINALOUTPUT` einkommentiert sein. Diese Möglichkeit sollte nur zu Debug-Zwecken genutzt werden.

Des Weiteren sind folgende Umgebungsvariablen hinterlegt, über deren Änderung der Benutzer die Eigenschaften und Leistungsfähigkeit des Algorithmus beeinflussen kann:
`SIGLENGTH`: Anzahl der Werte aus dem EDMA, die für die Berechnung des Winkels genutzt werden sollen.
`EDMABUFLEN`: Anzahl der Abtastwerte, die der EDMA in einem seiner Blöcke speichern soll, bevor der Block freigegeben wird.
`SAMFREQ`: Abtastrate, mit welcher der EDMA arbeiten soll.
`FREQUP`: Interpolationsfaktor der Eingangsdaten durch den Polyphasenfilter. Wenn dieser auf eins gestellt ist, wird keine Interpolation durchgeführt.

ARRAYTYPE: Aufbau des Mikrofonarrays. Gewählt werden kann zwischen dem Wert 1 (ULA) und 2 (UCYA bzw. UCA).
MICS: Anzahl der Mikrofone im Mikrofonarray. Da zum Zeitpunkt dieser Arbeit keines der Arrays mit mehr als acht Mikrofonen genutzt werden kann, darf dieser Wert nicht größer als acht eingestellt werden.
FFTLEN: Länge der FFT
Weitere Makros sind vorhanden, die im Normalfall aber unverändert bleiben sollten oder von den obengenannten beeinflusst bzw. berechnet werden. Weitere Informationen stehen als Kommentare in der Datei defines.h im Anhang.

7.2.2. Weitere Headerdateien

Die Datei variables.h enthält sämtliche Variablendeklarationen für die main und für Variablen, die in der Datei globals.h als global definiert sind.
In der Datei functions.h sind sämtliche Funktionsprototypen der Funktionen deklariert, die in dem Projekt verwendet werden.
Wenn per Makro SYNTHETICDELAY in der defines.h-Datei eingestellt, werden synthetische - durch MATLAB erzeugte - Signale als Eingangssignale verwendet. Diese stehen in der Headerdatei SyntheticSignals.h, die mit einem Matlabprogramm erstellt wurde und Synthetische Signale im Bereich von $\pm 60^\circ$ beinhaltet. Um die Headerdatei neu zu erstellen, kann die Matlabfunktion BuildSyntheticSignals.m verwendet werden, die im Anhang zu finden ist.
Wenn das Makro FREQUP größer ist als eins, so wird die UpsampleLowpass_L_All.h eingebunden. Die Filter in dieser Headerdatei wurden per MATLAB-Funktion PolyphasenfilterDesign.m erstellt. Die Headerdatei enthält die Filterkoeffizienten für die Polyphasenfilter zum Interpolieren der Eingangssignale auf verschiedene Abtastraten.
Für weitere Informationen und genauere Erklärungen dienen die jeweiligen Dateien im Anhang.

7.3. Belegung des Speichers auf dem DSP

Das DSP-Modul verfügt über 256 kByte internen Speicher und 2 MByte Flash Speicher, die für das Programm genutzt werden können. Des Weiteren können 64 MByte SDRAM verwendet werden, was aber im Hinblick auf die Geschwindigkeit des Algorithmus kritisch zu betrachten ist. Da der SDRAM lediglich mit 100MHz - also einem Drittel des Flash und internen Speichers - arbeitet, hat sich herausgestellt, dass es nicht sinnvoll ist, den SDRAM für den Echtzeitbetrieb zu nutzen.

In einer Command-Datei des Projekts muss die Größe des Stack und Heap, die für das Programm bereitstehen sollen, festgelegt werden. Während der Entwicklung haben sich rund 37 KByte für den Stack und 16 KByte für den Heap als geeignet herausgestellt. Diese Werte wurden allerdings empirisch und nicht unter Berücksichtigung des genauen Speicherbedarfs eingestellt.
In der Datei `variables.h` wurden einige Variablen in bestimmte Bereiche des Speichers abgelegt. Mit dem Schlüsselwort `near` werden Speicherkonflikte gelöst, die auftreten, wenn in der FFT-Routine von TI Arrayadressierungen durchgeführt werden. Durch die Verwendung von `near` werden die Variablen im Speicher in der Nähe der Funktion abgelegt, in der die Variablen genutzt werden. Durch den Pragmabefehl `DATA_ALIGN` sind Variablen, die von dem selbst erstellten Datentyp `COMPLEX` sind, auf gerade Adressen im Speicher abgelegt. Diese Maßnahme erhöht unter anderem die Geschwindigkeit beim Zugriff auf die Variablen in der FFT-Routine. Normalerweise bestimmt der Compiler, in welchen Teil des Speichers eine Variable abgelegt wird. Durch den Pragmabefehl `DATA_SECTION` kann klar definiert werden, in welchem Speicherbereich eine Variable liegen soll.
Sämtliche selbst geschriebenen Funktionen werden zwecks Rechenzeitoptimierung auf der Optimierungsstufe 3 verwendet. Auf dieser Optimierungsstufe werden bespielsweise Codeabschnitte, die in for-Schleifen stehen, so oft hintereinander geschrieben, wie die for-Schleife durchlaufen werden sollte. Durch diese Maßnahme wird Rechenzeit durch erhöhten Speicherplatzbedarf erkauft. Die Routinen von TI sind bereits handoptimiert, bzw. sollten nicht optimiert werden.

7.4. Umsetzung der Methoden

Jede Unterfunktion ist in einer eigenen Datei hinterlegt. Dies hat zum einen den Grund in der besseren Übersicht innerhalb der Funktionen, zum anderen können die Funktionen unabhängig voneinander auf eine Optimierung, die der Compiler durchführen soll, eingestellt werden. Projektseitig sind sämtliche Dateien eingebunden, da es nicht möglich ist, während der Laufzeit einzelne Dateien aus dem Projekt zu entfernen oder hinzuzufügen. Damit jedoch kein Speicherplatz auf dem DSP mit unbenötigten Funktionen belegt wird, sind die Funktionsdeklarationen in den Dateien mit Präprozessor-If Abfragen umklammert, um so den Code nicht zu kompilieren, wenn das jeweilige Makro nicht definiert ist (siehe Code 7.1 als Beispiel).

Code 7.1: Ausbindung von ungenutzten Code durch den Präprozessor

```
1 #ifdef USE_MCCC
2     float CalcDeterminant(float* a, int order){
3         .
4     }
5 #endif
```

Im Folgenden wird die Implementierung der verwendeten Methoden und Algorithmen erläutert.

7.4.1. Energieberechnung

Die Energieberechnung dient der Entscheidung, ob es sich bei den EDMA-Daten um ein stimmhaftes Sprachsignal handelt oder nicht. Sie wird zu Beginn des Algorithmus in der Funktion `CalcEnergy` durchgeführt. In dieser Funktion wird von den `EDMABUFLEN` langen Eingangssignalen blockweise eine Energieberechnung über `SIGLENGTH` Werte durchgeführt. Sobald die Energie für einen Block größer ist als die Schwelle `ENERGYLIMIT`, wird die Funktion abgebrochen und der Startindex dieses Blocks zurückgegeben. Wird bis zum letzten Block kein stimmhafter Anteil gefunden, so gibt die Funktion -1 zurück. Diese Vorgehensweise hat den Vorteil, dass aus Eingangsdaten, die z. B. in der Mitte oder erst am Ende stimmhaft sind, genau der Abschnitt gesucht wird, der stimmhaft ist. Es wird also eine Entscheidung darüber getroffen, aus welchem Teil des EDMA-Blockes die Richtung berechnet werden soll. Werden die Grenzen für die Energieberechnung statisch festgelegt - zum Beispiel auf den Beginn der Eingangsdaten - so würden in solch einem Fall die gesamten Eingangsdaten als stimmlos eingestuft werden. Unter Berücksichtigung der Stationariätseigenschaft von Sprache ist die Energieberechung auf diese Weise erlaubt (das heißt die Länge der EDMA-Blöcke darf 50ms nicht überschreiten).
In der `main` Funktion wird der zurückgegebene Startindex in einer temporären Variablen `tmpDummyShort` gespeichert, die bei dem anschließenden Kopieren der Daten aus dem EDMA als Offset fungiert.

7.4.2. Kopieren der EDMA-Daten und Polyphasenfilter

Grundsätzlich ist das Kopieren ganzer Blöcke nicht sinnvoll. Ein solcher Vorgang benötigt auf einem DSP sehr viele Taktschritte und sollte deswegen möglichst vermieden werden[17]. Da der Algorithmus in Fließkommaarithmetik geschrieben ist, die Daten aus dem EDMA jedoch im Datentyp `short` vorliegen, ist eine Kopie der Daten unumgänglich. Die EDMA-Daten werden nach `float` umgewandelt und in das Array `SignalArray[MICS][FFTLENGTH]` geschrieben, das vom Datentyp `COMPLEX` ist.

[17] Eine Alternative zum Kopieren wäre die Umlenkung von Zeigern auf den jeweiligen Speicher, der den Beginn eines Feldes darstellt.

Dies geschieht direkt

Code 7.2: Ausschnitt aus dem Hauptprogramm: Direktes kopieren der EDMA-Daten

```
1 for(i = 0 ; i < MICS ; i++){
2     for(j = 0 ; j < SIGLENGTH ; j++){       //if no Interpolation, just take the ADC
          Data
3         SignalArray[i][j].re = (float)(adcbuffer[ch_corr[i]][block][tmpDummyShort+j
              ])/32767.0;
4         SignalArray[i][j].im = 0.0;
5     }
6 }
```

oder, für den Fall, dass eine Interpolation der Eingangsdaten durchgeführt werden soll, über einen Filter:

Code 7.3: Ausschnitt aus `main`: Interpolation der EDMA-Daten

```
1 for(j = 0 ; j < SIGLENGTH ; j++){
2     for(k = 0 ; k < FREQUP ; k++){      //Take data form ADC and push them through
          the delays
3         tmpDummy = (float)(adcbuffer[ch_corr[i]][block][tmpDummyShort+j])/32767.0;
4         SignalArray[i][(j*FREQUP)+k].re = FIR_filter_float(lpDelays,PolyphaseCoeffs[
              k],N_delays_H_filt,tmpDummy);
5         SignalArray[i][(j*FREQUP)+k].im = 0.0;
6     }
7 }
```

In den oben aufgeführten Codeausschnitten wird in Zeile 3 zunächst die Umwandlung von `short` nach `float` durchgeführt und anschließend durch $2^{15} - 1 = 32767$ geteilt. Diese Teilung vollzieht die Anpassung auf den `float`-Bereich und begrenzt die Eingangswerte auf -1.0 bis +1.0.
Unter Abschnitt 6.1 wurde das Polyphasenfilter zur Interpolation in eine parallele Anordnung mehrerer FIR-Filter umgewandelt. Aus diesem Grund kann hier die optimierte Assemblerroutine `FIR_Filter_float` eines FIR-Filters verwendet werden. Diese Funktion berechnet den Ausgabewert des Filters für einen Schritt und speichert die Inhalte der Verzögerungsglieder in `H_filt_delays`. Somit kann die Antwort des Filters auf das Eingangssignal durch einen mehrfachen Aufruf der Funktion - wie in Code 7.3 durch eine Schleife realisiert - berechnet werden.
Während der EDMA mit der Abtastrate `SAMFREQ` arbeitet, muss nach der Interpolation mit der Pseudo-Abtastrate `FREQUP`$\cdot$ `SAMFREQ` gerechnet werden. Die gefilterten bzw. direkt kopierten Eingangsdaten sind natürlich reell, weswegen die Imaginärteile auf Null gesetzt werden.

7.4.3. Kreuzkorrelationen

Die Berechnung der Kreuzkorrelation geschieht wie in Abschnitt 3.6 beschrieben über die Multiplikation beider zu korrelierenden Signale im Frequenzbereich. Grundsätzlich muss die

Kreuzkorrelation eines Mikrofonsignals einmal mit jedem anderen Mikrofonsignal berechnet werden. Wie in Abschnitt 5.5 dargestellt, kann die Symmetrie der KKF ausgenutzt werden, um diese Anzahl um knapp die Hälfte zu reduzieren. Im Folgenden werden die Schritte dargestellt, die implementiert wurden, um das zweidimensionale Array `CorrSignals` zu berechnen. Abbildung 7.2 zeigt ein Blockschaltbild der Implementierung der Kreuzkorrelationen.

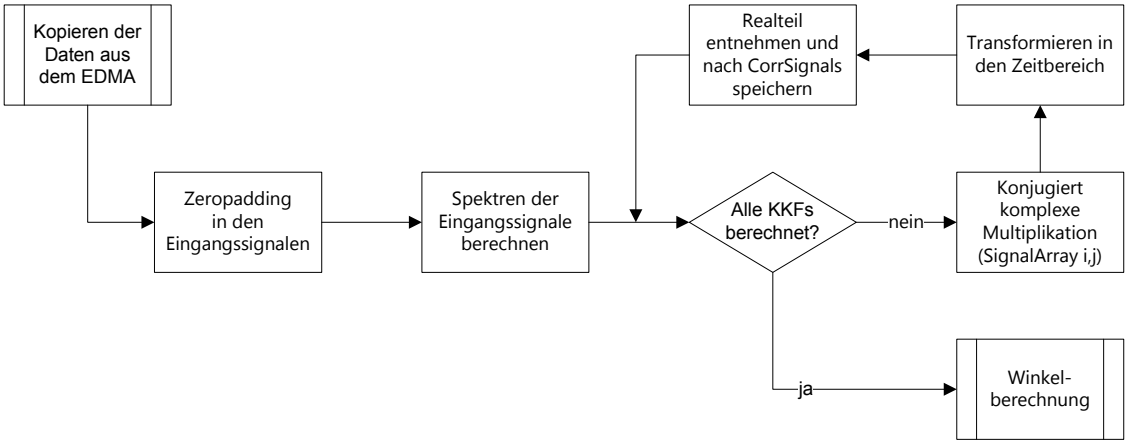


Abbildung 7.2.: Ablauf der Kreuzkorrelationen auf dem DSP

Berechnung der FFTs und Zeropadding

Im Algorithmus wird eine Vielzahl von FFTs berechnet. Es ist also sinnvoll, eine Routine zu verwenden, die möglichst wenig Taktschritte benötigt. In dieser Arbeit wird eine handoptimierte Radix-2 decimation-in-time FFT der Firma Texas Instruments eingesetzt, deren Länge einer Potenz von zwei entsprechen muss (also: $4, 8, 16, 32, \ldots$). Die Routine basiert auf dem Ansatz, eine L_{FFT}-Punkte Diskrete Fouriertransformation (DFT) in zwei kleine DFTs - eine für die ungeraden diskreten Eingangswerte, die andere für die geraden diskreten Eingangswerte - aufzuteilen. Dieser Vorgang wird für die entstehenden, kleineren DFTs fortgesetzt, bis ausschließlich DFTs der Länge 2 entstehen (siehe Abbildung 7.3). Wie in der Abbildung zu sehen ist, wird der untere Zweig jeder DFT der Länge 2 - auch als Butterfly bezeichnet - mit einem sog. Twiddlefaktor multipliziert. Diese Twiddlefaktoren werden für eine FFT-Länge von acht nach der Formel

$$\underline{W}_8^k = \cos\left(2\pi \cdot k/8\right) + j \cdot \sin\left(2\pi \cdot k/8\right)$$

und allgemein durch

$$\underline{W}^{k}_{L_{FFT}} = \cos(k \cdot \delta) + j \cdot \sin(k \cdot \delta), \; mit \; \delta = \frac{2\pi}{L_{FFT}} \tag{7.1}$$

berechnet[18]. Die Einsparung von Taktschritten in der FFT resultiert unter anderem daher, dass die Ausgänge eines Butterflies hinterher mehrfach verwendet werden können.

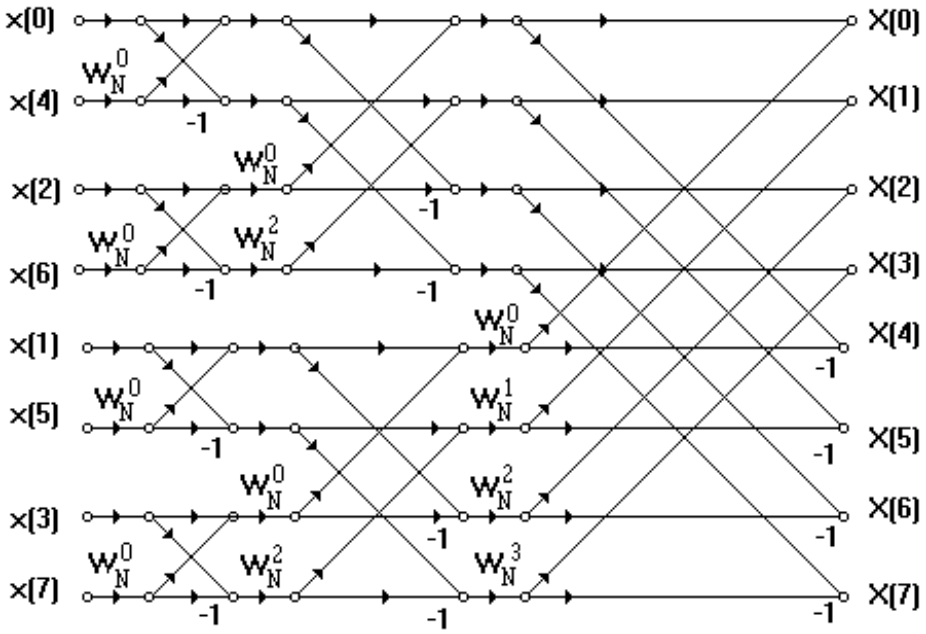


Abbildung 7.3.: Radix-2 decimation-in-time FFT Algorithmus für $L_{FFT} = 8$

Es wird die handoptimierte FFT-Routine `cfftr2_dit` verwendet. Die Eingangsdaten müssen in komplexer Form vorliegen, wobei der Real- und Imaginärteil hintereinander im Array stehen: $Re(x_0)$, $Im(x_0)$, $Re(x_1)$, $Im(x_1)$, ... Die Länge der Eingangsdaten im Speicher entspricht also $2 \cdot L_{FFT}$. Unter Verwendung des Datentyps `COMPLEX` wird eine solche Anordnung automatisch erreicht, da die unter dem Datentyp zusammengefassten Werte automatisch hintereinander im Speicher angeordnet werden. Des Weiteren muss zuvor ein Zeropadding durchgeführt worden sein, bei dem hinter die eigentlichen Nutzdaten mindestens so viele Nullen angehängt werden, wie Nutzdaten vorhanden sind. Es hat sich außerdem gezeigt, dass die Eingangsdaten durch den Pragma-Befehl `DATA_ALIGN` auf eine gerade Adresse im Speicher gelegt werden müssen, um Taktschritte zu sparen.
Wie in Abb. 7.3 zu erkennen, sind die Eingangsdaten nicht in der richtigen Reihenfolge, sondern bitweise vertauscht (sog. bit reversal). Dieses bitweise Vertauschen kann wahlweise vor- oder nach der FFT durchgeführt werden, da die FFT in der vorgestellten Form ein

[18] Diese Formel kann aus der allgemeinen Formel der DFT mit Aufteilung in gerade und ungerade Indizes hergeleitet werden.

symmetrisches Verfahren ist. Die Twiddlefaktoren müssen ebenfalls bitweise vertauscht angeordnet vorliegen. Da sich diese für eine vorgegebene Länge der FFT niemals ändern, reicht es, sie einmal zu Beginn der `main`-Funktion zu berechnen und umzusortieren. Die Initialisierung der FFT ist im folgenden Codeausschnitt dargestellt.

Code 7.4: Initialisierung der FFT

```
1 for( i = 0 ; i < FFTLEN/RADIX ; i++ )
2 {
3      w[i].re = cos(DELTA*i);                //real component of W
4      w[i].im = sin(DELTA*i);                //neg imag component
5 }                                           //see cfftr2_dit
6 digitrev_index(idxTableW, FFTLEN/RADIX, RADIX);
7 bitrev(w,idxTableW,FFTLEN/RADIX);
8 digitrev_index(idxTable, IDXTABLEN, RADIX);}
```

In der for-Schleife werden die Twiddle-Faktoren nach Gleichung 7.1 berechnet. Eine Besonderheit der Funktion `bitrev`, die das bitweise Umsortieren einer Eingangsfolge übernimmt, ist, dass die Art und Weise, wie die Eingangsfolge umgestellt wird, in einer Index-Tabelle steht. Diese wird für die jeweilige Länge der FFT in der Funktion `digitrev_index` zunächst für die Twiddlefaktoren (Codeausschnitt 7.4 Zeile 6) und anschließend für die späteren FFT-Ausgangsdaten (Codeausschnitt 7.4 Zeile 8) erstellt.
Die FFTs der einzelnen Mikrofonsignale werden in folgender for-Schleife berechnet:

Code 7.5: Berechnung der Eingangssignalspektren

```
1 for(i = 0 ; i < MICS ; i++){
2     cfftr2_dit(SignalArray[i],  w, FFTLEN);
3     bitrev(SignalArray[i], idxTable,FFTLEN);
4 }
```

Im Gegensatz zur Abbildung 7.3, in der die Eingangsdaten bitweise umsortiert und die Ausgangsdaten in der richtigen Reihenfolge sind, werden die Eingangsdaten in der richtigen Reihenfolge in die FFT-Routine gegeben und anschließend die Ausgangsdaten umsortiert.

Umsetzung der KKF

Nachdem die Signalspektren berechnet sind, werden die KKFs nach Gleichung 3.44 berechnet.

Code 7.6: Berechnung der Kreuzkorrelationen

```
1  k=0;
2  for(i = 0 ; i < MICS ; i++){
3      for(j = i ; j < MICS ; j++){
4          ConjComplexMult(i,j);
5
6          cfftr2_dit(tmpSave,w, FFTLEN);       //'ifft'
7          bitrev(tmpSave, idxTable, FFTLEN);
8
9          GetRealToCorrSignals(k);
10         k++;
11     }
12 }
```

Es folgt also zunächst eine komplexe Multiplikation (Codeausschnitt 7.6, Zeile 4), wobei einer der Kanäle konjugiert komplex sein muss. Diese Multiplikation wird in der Funktion `ConjComplexMult` durchgeführt. Diese Funktion erhält als Übergabeparameter lediglich die Indizes der beiden Spektren, die korreliert werden sollen. Das Ergebnis der komplexen Multiplikation wird in der globalen Variable `tmpSave` gespeichert. Anschließend wird die inverse-FFT berechnet. Da gilt

$$\mathcal{F}^{-1}\{c\} = \mathcal{F}\{c^*\},$$

kann für die Berechnung der inversen-FFT die selbe Routine verwendet werden, die auch zuvor für die FFTs benutzt wurde. Da das Signalspektrum für dieses Vorgehen konjugiert komplex vorliegen muss, wird der Imaginärteil in der vorherigen Funktion `CalcConjComplex` negiert.
Zuletzt muss der Realteil der Rücktransformation entnommen und eine Skalierung der Werte durchgeführt werden. Dies geschieht in der Funktion `GetRealToCorrSignals` (Codeausschnitt 7.6 Zeile 9). Die Skalierung auf L_{FFT} wird in der Funktion durch eine Multiplikation dargestellt, um Taktschritte zu sparen. Eine Divison auf dem DSP benötigt 30 Assembler-Taktschritte [Pol99], eine Multiplikation dagegen nur 3.
Es ist wichtig zu beachten, dass das Array `CorrSignals` die Größe `CORRSIGNALNUMROWS` x `FFTLEN` hat. In dem Makro `CORRSIGNALNUMROWS` ist die mathematische Vorschrift für die Anzahl der KKFs hinterlegt (Gleichung 5.5). Da die Vorschrift sowohl Multiplikationen als auch Divisionen beinhaltet, sollte das Makro außer in Variablendeklarationen nicht verwendet werden. `CorrSignals` liegt als Spaltenvektor vor:

$$\begin{bmatrix} \varphi_{0,0} & \varphi_{0,7} & \varphi_{1,1} & \varphi_{1,7} & \varphi_{2,2} & \cdots \end{bmatrix}^T$$

Die symmetrischen KKFs z. B. $\varphi_{1,0}$ werden *nicht* berechnet.

7.4.4. Multichannel Cross Correlation

Die Entscheidung, ob die MCCC verwendet werden soll oder nicht, geschieht über Ein- oder Auskommentierung des Makros `USEMCCC`. Die Fehlerfunktion der MCCC wird folgendermaßen berechnet:

Code 7.7: MCCC

```
i = 0;
for(j = minDegreeSample ; j <= maxDegreeSample ; j++){
    ExtractCorrelationMatrixToRa(j);        //Take Ra for degree j from CorrSignals
//  NormToStandardDeviations(); //no improvement of results just a scaling!
    J[i] = CalcDeterminant(Ra,MICS);        //Calculate the Determinant!
    i++;
}
```

In der `main` wurden zu Beginn Initialisierungen für den Algorithmus durchgeführt. Hierzu zählte auch, die minimal und maximal möglichen Winkel, die mit dem eingestellten Mikrofonarray zu detektieren sind, festzulegen und in `minDegreeSample` und `maxDegreeSample` zu speichern. Die Schleife in Codeabschnitt 7.7 läuft zwischen diesen Werten und berechnet den Fehler `J[i]`. Die MCCC wird in zwei bis drei Schritten vollzogen:

1. Korrelationsmatrix $\mathbf{R}_a$ aus KKFs für den aktuellen Winkel bestimmen,
2. eventuell Berechnung von $\tilde{\mathbf{R}}_a$ also Normierung auf Standardabweichungen,
3. Determinante der (eventuell normierten) Korrelationsmatrix berechnen.

Korrelationsmatrix bestimmen

Prinzipiell besteht die Korrelationsmatrix aus den Kovarianzen r_{a,y_i,y_j} für einen angenommenen Winkel p. Dieser ist in der Implementierung durch den Laufindex `j` dargestellt. Bezogen auf die KKFs der Signale bedeutet dies: r_{a,y_i,y_j} ist der Wert, der in der Kreuzkorrelierten zwischen den Mikrofonen i und j beim Index p auftritt. Es müssen also pro angenommenen Winkel lediglich die N^2 Werte aus den verschiedenen Kreuzkorrelierten entnommen werden, welche die Korrelationsmatrix $\mathbf{R}_a(p)$ zusammensetzen. Diese Zusammensetzung ist in der Funktion `ExtractCorrelationMatrixToRa` implementiert. Die Funktion belegt zunächst die obere Dreiecksmatrix von $\mathbf{R}_a(p)$ in Abhängigkeit der verwendeten Methode (MCCC oder SLP) und des Mikrofonarrayaufbaus und kopiert anschließend die Werte in die untere Dreiecksmatrix. Die Unterscheidung zwischen einem ULA und UCYA ist darum notwendig, weil die Funktionen $F(p)$ unterschiedlich sind und für den Winkel p berechnet werden müssen. Außerdem wird die Verzögerung $F_i(p)$ zum Mikrofon i beim linearen Aufbau immer relativ zum Referenzmikrofon M_0 berechnet. Codeausschnitt 7.8 zeigt die Erstellung der Korrelationsmatrix für ein ULA.

Code 7.8: Extrahierung von $\mathbf{R}_a$ im ULA

```
1  if(actDegree < 0){
2      //Source is at FFTLEN in the corrsignals, all FunkTaus < 0!
3      for(i = 0 ; i < MICS ; i++){
4          for(j = i ; j < MICS ; j++){
5              if(i == j)
6                  Ra[k] = CorrSignals[l][0];
7              else
8                  Ra[k] = CorrSignals[l][FFTLEN + actDegree*(j-i)];
9              l++;
10             k++;
11         } //end for
12         k += (i+1);
13     }   //end for
14 } //end if
15 else{
16     //Source is at 0 in the corrsignals, all FunkTaus > 0!
17     for(i = 0 ; i < MICS ; i++){
18         for(j = i ; j < MICS ; j++){
19             if(i==j)
20                 Ra[k] = CorrSignals[l][0];
21             else
22                 Ra[k] = CorrSignals[l][actDegree*(j-i)];
23             l++;
24             k++;
25         } //end for
26         k += (i+1);
27     }   //end for
28 }
```

Es wird zunächst unterschieden, ob der angenommene Winkel kleiner oder größer ist als null, da die Maxima in den KKFs dann entweder in Richtung `FFTLEN` oder in Richtung des nullten Elements liegen (Zeile 1 und 15). Eine weitere Unterscheidung muss innerhalb der Schleifen gemacht werden, wenn `i` und `j` gleich sind. Denn dann wird die *Varianz* eines Kanals gesucht, die in der KKF *immer* beim nullten Element liegt. In den Zeilen 8 und 22 wird anschließend `CorrSignals` entsprechend indiziert und die Kovarianzen für p ermittelt. Dabei ist der Ausdruck `actDegree*(j-i)` die Umsetzung der Funktion F_{lin}. In Zeile 8 müsste der Offset implementiert sein, der durch die Berechnung der KKF über das Faltungsintegral entsteht (siehe Abschnitt 6.2). Da die Zählweise der Arrayelemente in C bei 0 beginnt und bei Arraylänge-1 aufhört, ist der Offset dadurch implementiert, dass von `FFTLEN` keine 1 abgezogen wird. Da `actDegree*(j-i)` niemals null wird (wird durch die Abfrage in Zeile 1 verhindert), kommt es zu keinen Speicherübergriffen.
Beim UCYA dagegen wird die Verzögerung des Signals am Mikrofon M_i immer relativ zum letzten Mikrofon $i-1$ berechnet (für $i > 0$). Da die Vorschrift nichtlinear ist, muss vor der Extrahierung ein Array `fvontau` berechnet werden, in dem die Verzögerungen zum Mikrofon i relativ zum Referenzmikrofon M_0 abgelegt werden.

Code 7.9: Berechnung der relativen Verzögerungen im UCA bzw. UCYA

```
1 for(i = 0 ; i < MICS ; i++){
2     tmpValue = RADIUS/C;
3     tmpValue2 = tmpDegRads-PSIINRADS*(float)(i);
4     tmpValue = tmpValue * (cos(tmpDegRads)- cos(tmpValue2));
5     fvonTau[i] = (short)(tmpValue*SAMFREQ*FREQUP);
6 }
```

Die Berechnung erfolgt nach Gleichung 3.5 und ist in dem Codeabschnitt in mehrere Schritte aufgeteilt, weil der DSP bei längeren Ausdrücken Probleme mit der internen Registerbelegung bekommt und absturzgefährdet ist. Außerdem ist eine Aufteilung von Gleichungen in mehrere Codezeilen deshalb sinnvoll, weil Taktschritte dadurch gespart werden, dass der DSP in den internen Registern keine Kopien der Zwischenwerte anfertigen muss. Die Belegung von `Ra` unterscheidet sich vom linearen Array davon, dass keine Unterscheidung zwischen negativen und positiven Winkeln gemacht werden muss, da der Winkelbereich im zirkularen Array auf $0° - 359°$ definiert wurde.

Code 7.10: Extrahierung von $\mathbf{R}_a$ im UCA bzw. UCYA

```
1  for(i = 0 ; i < MICS ; i++){
2      for(j = i ; j < MICS ; j++){
3          if(i == j)
4              Ra[k] = CorrSignals[l][0];
5          else{
6              lookAtSmp = fvonTau[j] - fvonTau[i];
7              if(lookAtSmp < 0)
8                  Ra[k] = CorrSignals[l][FFTLEN + lookAtSmp];
9              else
10                 Ra[k] = CorrSignals[l][lookAtSmp];
11         }
12         l++;
13         k++;
14     } //end for
15     k += (i+1);
16 }   //end for
```

Es wird jeweils eine Variable `lookAtSmp` berechnet, die aus den relativen Verzögerungen der Mikrofone M_i und M_{i-1} zum Referenzmikrofon erzeugt wird. Die Indizierung der `CorrSignals`-Matrix ist ähnlich der für einen linearen Aufbau.
Die Berechnung von $\mathbf{R}_a$ in der SLP stimmt mit der in der MCCC überein. Es ist lediglich berücksichtigt, dass die Schleife in Zeile 1 von Codeabschnitt 7.10 und in den Zeilen 3 und 17 in Codeabschnitt 7.8 bei eins beginnt, da die Kovarianzen zum Mikrofon M_0 in $\mathbf{R}_a$ nicht vorkommen.
Die Matrix `Ra` ist in der Implementierung nicht direkt als Matrix hinterlegt. Vielmehr ist `Ra` ein eindimensionales Feld, in dem die Zeilen der Matrix hintereinander liegen (ähnlich wie bei der Variablen `CorrSignals`).

Normierung auf Standardabweichgungen

Unter Umständen kann es sinnvoll sein, $\mathbf{R}_a$ auf die Standardabweichungen der einzelnen Kanäle zu normieren. Dies macht dann Sinn, falls sich herausstellt, dass die Determinantenwerte an den darstellbaren Wertebereich des Datentyps `float` angrenzen[19]. Bei der Einkommentierung der Funktion ist aber darauf zu achten, dass N mal eine Wurzel berechnet und N^2 Divisionen durchgeführt werden. Hinzu kommen Multiplikationen und Overhead durch Schleifen. Laut [Pol99] benötigt eine Quadratwurzel 40 Assemblertaktschritte, eine Division 30, das wiederum zu einer Erhöhung des Bedarfs an Taktschritten von mindestens 2240 Taktschritten, bzw. $7.5\mu s$ *pro Winkel* führen würde (Beispiel mit $N = 8$).
Es sollte somit möglichst vermieden werden auf die Standardabweichungen zu normieren, um Taktschritte einzusparen.

Berechnung der Determinante

In Zeile 5 des Codeausschnitts 7.7 wird die Determinante von $\mathbf{R}_a$ berechnet, die den Fehler für den Winkel p darstellt. Die Berechnung der Determinante wird in der Funktion `CalcDeterminant()` durchgeführt.
Bis zu einer Matrixgröße von drei sind Determinanten recht einfach zu implementieren, da die Berechnung einer klaren mathematischen Form folgt. Bei Matritzen höherer Ordnungen können das gaußsche Eliminationsverfahren oder aufwendigere Verfahren, wie die Leibniz-Formel oder der Laplace'sche Entwicklungssatz verwendet werden. Die zuletzt genannten Verfahren sind jedoch zu aufwendig zu implementieren, weswegen hier das gaußsche Eliminationsverfahren angewandt wird.
Mit Hilfe des gaußschen Eliminationsverfahrens wird eine beliebige Matrix der Größe N

$$\det(\mathbf{A}_n) = \begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{vmatrix}$$

unter Verwendung elementarer Zeilenumformungen in die obere Dreiecksform gebracht:

[19] `float` hat 32 Bit, womit ein Wertebereich von $1.5e^{-45} \ldots 3.4e^{38}$ dargestellt werden kann.

$$\det(\mathbf{A}_n) = \begin{vmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n-1} & b_{1,n} \\ 0 & b_{2,2} & \cdots & b_{2,n-1} & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & \cdots & b_{n,n-1} & b_{n,n} \end{vmatrix}.$$

Die Determinante kann nun als Produkt der Elemente auf der Hauptdiagonalen berechnet werden

$$\begin{aligned} \det(\mathbf{A}_n) &= (-1)^r \cdot b_{1,1} \cdot b_{2,2} \cdot \ldots \cdot b_{n,n} \\ &= (-1)^r \cdot \prod_{i=1}^{n} b_{i,i}, \end{aligned} \tag{7.2}$$

wobei mit r die Anzahl der durchgeführten Zeilenvertauschungen bezeichnet ist. Es kann in der Funktion bestimmt werden, ob Zeilenvertauschungen durchgeführt werden sollen oder nicht. Eine Zeilenvertauschung macht dann Sinn, wenn das Element einer Zeile gleich null ist, das in der Spalte steht, die zu null gesetzt werden soll. Dies impliziert in dem Code einen Vergleich, sowie eventuelles Tauschen von Reihen, was viel Zeit kosten würde. Da davon ausgegangen werden kann, dass bei vorhandenem Rauschen in der Korrelationsmatrix keine Nullen vorkommen, ist ein Zeilentausch jedoch nicht notwendig. Codeausschnitt 7.11 zeigt das Eliminationsverfahren:

Code 7.11: Gaußsches Eliminationsverfahren

```
1  for(i = 0; i < order - 1; i++)  %order: Order of the Matrix
2  {
3      for(k = i + 1; k < order; k++)
4      {
5          factor = -1.0 * a[i+(k*order)] /  a[(i*order)+i];
6          for(j = i; j < order; j++)
7          {
8              a[(k*order)+j] += (factor * a[(i*order)+j]);
9          }
10     }
11 }
```

`Ra` wird in der Funktion unter dem Namen `a` übernommen und ist somit ebenfalls eine eindimensionale Matrix mit hintereinander angeordneten Zeilen. Die Arrayindizierungen `i*order` bzw. `k*order` simulieren quasi die zweidimensionale Matrix. Der Ausdruck `(k*order)+j` greift auf das j^{te} Element in der k^{ten} Zeile zu, also auf $\mathbf{R}_a(k, j)$.

7.4.5. Spatial Linear Prediction

Die SLP unterscheidet sich erst nach der Extrahierung der Korrelationsmatrix von der MC-CC. Dabei muss beachtet werden, dass die Kovarianzen des ersten Mikrofonsignals nicht in $\mathbf{R}_a$ vorkommen. Die Kovarianzen zwischen dem ersten und den anderen L Mikrofonsignalen werden in das Feld `ra` geschrieben, woraufhin eine Matrixinversion, sowie die Realisierung der Gleichung 3.23 - also eine mehrfache Matrixmultiplikation - implementiert ist. Die gesamte SLP ist in Codeabschnitt 7.12 abgebildet.

Code 7.12: SLP

```
i = 0;
for(j = minDegreeSample ; j <= maxDegreeSample ; j++){

    ExtractCorrelationMatrixToRa(j);        //Take Ra for degree j from CorrSignals
    Extract_ra_FromCorrelationMatrix(j);    //Take ra for degree j from CorrSignals
    MatrInv(Ra,Ra);

    //First Multiplication  b = ra' * Ra^1
    MatrMult_C_o3(ra,Ra,tmpArray1,1,MICS-1,MICS-1);
    //Second Multiplication c = b * ra
    MatrMult_C_o3(tmpArray1,ra,tmpVal,1,MICS-1,1);

    //calculate J!
    J[i] = CorrSignals[0][0] - tmpVal[0];   //var0 - c;

    i++;
}
```

Matrixinversion

Für die Inversion der Matrix wird eine Routine aus [Pen70, S. 349 und S. 371] verwendet, die ursprünglich in der Programmiersprache Fortran implementiert und später vom Autor in C umgeschrieben wurde. Die Routine arbeitet mit eindimensionalen Eingangs- und Ausgangsmatritzen und berechnet die Inverse der Eingangsmatrix in Fließkommaarithmetik. Die Funktion benutzt den Gauß-Jordan Algorithmus auf numerische Weise, wobei die Einheitsmatrix auf der linken Seite über eine Abbruchbedingung angenähert wird. Diese Bedingung dient dazu, die Iterationsdauer des Algorithmus zu steuern, indem eine Zahl als null angenommen wird, obwohl sie nur sehr klein ist.
Um die Genauigkeit dieser Methode zu verifizieren, wurde eine Matrixinversion in MATLAB mit der numerischen Methode verglichen. Dazu wurden 30 Matritzen mit einer Dimension von zwei bis acht mit MATLAB invertiert und der quadratische mittlere Fehler der Matrixelemente des Ergebnisses der Routine berechnet. Wie in Abbildung 7.4 zu erkennen ist, sinkt der Fehler bei zunehmender Dimension der Matrix. Da der Fehler oberhalb von der Dimension drei einen Fehler von 10^{-4} nicht übersteigt, kann die Genauigkeit als ausreichend bezeichnet werden. Bei einer Dimension von zwei ereicht der Fehler teilweise den Promillebereich.

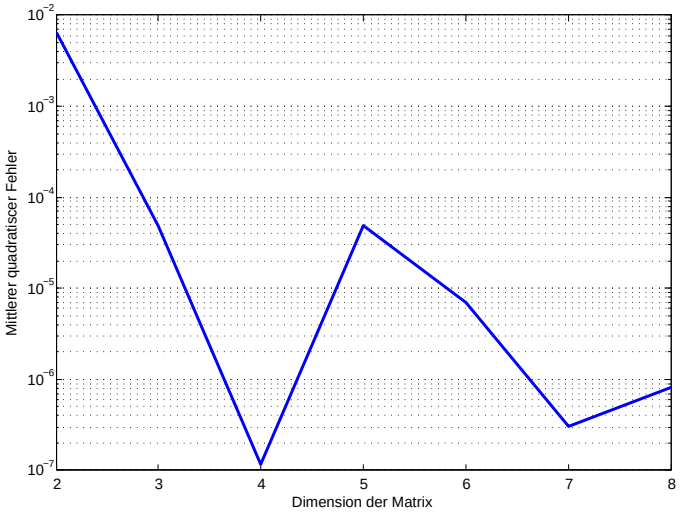


Abbildung 7.4.: Mittlerer quadratischer Fehler der Matrixinversion in C

Matrixmultiplikationen

Nach der Inversion folgt die Umsetzung der Gleichung 3.23 in der Form zweier Matrixmultiplikationen (Zeilen 8-11 in Codeausschnitt 7.12. Die Multiplikation wird in der Funktion `MatrMult_C_o3` durchgeführt. Dieser werden - neben den beiden Multiplikanden und dem Ergebnisarray - drei Variablen, welche die Größe der Matritzen beschreiben, übergeben. Somit kann diese Funktion variabel für jede Art von Matrixmultiplikation verwendet werden, solange die Regeln der Matrixmultiplikation beachtet werden.

7.4.6. Finden und Speichern des Winkels

Beide Winkelerkennungsalgorithmen liefern eine Fehlerfunktion $J(p)$, deren Minimum bei dem p liegt, an dem die Quelle gesehen wurde. Es wird also das Minimum von J in der Funktion `FindMinimumOfJ` berechnet. Diese Funktion gibt den Index des Minimums zurück, aus welchem der Winkel abhängig vom Aufbau des Arrays umgerechnet wird. Anschließend wird der Winkel in das Array `EstDoa` abgelegt, der die einstellbare Länge `NUM_OF_DOA_SAVE` hat und als Moduloarray mit der Indizierungsvariablen `AlgCounter` benutzt wird. `AlgCounter` zeigt immer auf den aktuellen Winkel in `EstDoa` und wird erst ganz am Ende eines Algorithmusdurchlaufs inkrementiert.

7.4.7. Variable Verzögerung mittels VDL

Am Ende des Algorithmus werden die ADU-Eingangsdaten der Mikrofone mittels einer VDL verzögert, sodass sie über den DAU quasi mit einem Winkel von 0° ausgegeben werden. Die Länge der VDL wird - wie in Abschnitt 5.2 erwähnt - durch die Abtastrate, sowie den Aufbau des Mikrofonarrays festgelegt. Das Prinzip der VDL wird im Folgenden erläutert.
Jeder Mikrofonkanal verfügt über eine eigene Verzögerungskette der Länge `DELAYLENGTH`. Zusätzlich zeigen pro Kanal jeweils ein Lesezeiger (`rptr`), sowie ein Schreibezeiger (`wptr`) auf eines der Elemente in der Verzögerungskette. Diese Zeiger bewegen sich durch die Verzögerungskette und schreiben neue Daten hinein bzw. lesen Daten aus (Abbildung 7.5).

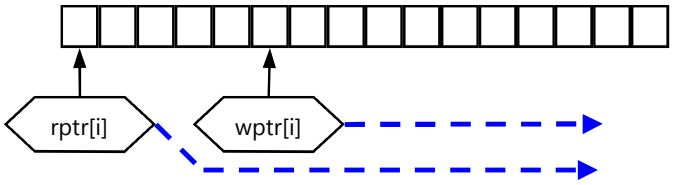


Abbildung 7.5.: Aufbau der VDL für ein Mikrofon *i* (1)

Der Abstand zwischen den beiden Zeigern beschreibt den derzeitigen Winkel, also die Verzögerung, die zwischen dem Referenzmikrofon und Mikrofon *i* vorliegt. Dabei gilt:

- Liegen die beiden Zeiger aufeinander, so ist die Verzögerung gleich null,
- liegt der Lesezeiger im Speicher hinter dem Schreibezeiger, so herrscht eine positive Verzögerung.
- Umgekehrt liegt eine negative Verzögerung vor, wenn der Lesezeiger vor dem Schreibezeiger steht.

Sobald einer der Zeiger am Ende der Verzögerungskette angelangt ist, wird er auf das erste Element umgelenkt, sodass eine Art Moduloregisterzugriff entsteht (In Abbildung 7.6 am Beispiel des Schreibezeigers dargestellt).

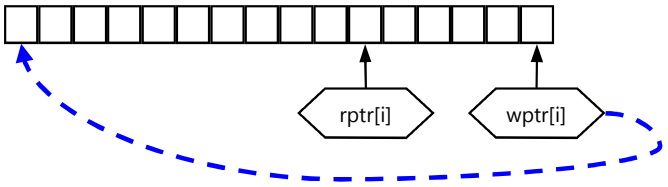


Abbildung 7.6.: Aufbau der VDL für ein Mikrofon *i* (2)

Sobald der Algorithmus eine neue Verzögerung berechnet, die von der vorherigen abweicht, wird lediglich der Pointer zum Lesen umgelenkt. Und zwar genau um so viele Elemente in der Verzögerungskette, dass zwischen den Pointern die gewünschte Verzögerung besteht. In Abbildung 7.7 wird der Lesezeiger um drei in Richtung des Schreibezeigers gelenkt, die Quelle hat sich also in Richtung des Winkels Null bewegt.

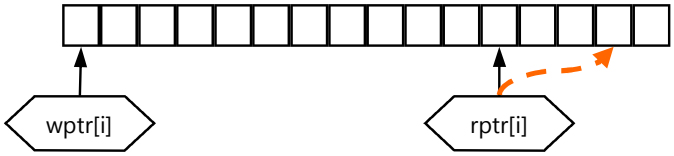


Abbildung 7.7.: Aufbau der VDL für ein Mikrofon i (3)

Bei einer solchen Umlenkung des Lesepointers kann es dazu kommen, dass Daten die gerade in die Kette geschrieben wurden, übersprungen werden (siehe Abbildung). Andersherum kann es passieren, dass Daten doppelt ausgelesen werden, wenn sich der Winkel in einer Richtung vergrößert. In beiden Fällen würden zeitweise falsche Daten in den DAU geschrieben werden Schlimmstenfalls könnten sich ganze Wortabschnitte wiederholen. Um diesen Fehler gering zu halten, gibt es die Möglichkeit über Interpolation die Daten anzunähern. Hinsichtlich des Ziels der *Zuverlässigkeit* muss geprüft werden, ob eine Interpolation nötig ist, anders gesagt, wie viele Daten maximal verloren gehen oder wiederholt werden könnten. Der in diesem Sinne ungünstigste Fall tritt auf, wenn eine Quelle beim Maximum des Winkelbereichs detektiert und dann direkt beim Minimum des Winkelbereichs gesehen wird. Bei einem linearen Aufbau mit acht Mikrofonen und einer Abtastrate von $48kHz$ würde der Sprung $120°$ bedeuten (berücksichtigt wird hier das Simulationsergebnis aus Unterabschnitt 6.4.1, nach dem der maximal zu detektierende Bereich auf $\theta_{max} = \pm 60°$ festgelegt wurde). Aus Gleichung 5.3 ergibt sich damit eine maximale Verzögerung von 42 Abtastwerten. Dementsprechend würde der Fehler 84 Abtastwerte betragen. Der Spracherkennungsalgorithmus von [Kla10] läuft mit 8kHz, das heißt von den 84 Werten werden nur 14 fehlerhafte Werte - bzw. 1.75ms - analysiert. Bei einer verwendeten Blocklänge von 240 Abtastwerten macht diese Anzahl nur einen Bruchteil aus. Ein Test mit der Simulationsumgebung des Spracherkennungsalgorithmus hat gezeigt, dass sich die Qualität der Spracherkennung nicht verschlechtert, wenn 1.75ms in den Eingangsdaten verfälscht werden. Eine Interpolation ist somit nicht notwendig.
Die VDL ist in der Funktion `DelayLine` implementiert. Diese erhält als Eingang den aktuellen Winkel und, da sie auf die DAU- und ADU-Daten des EDMA zugreift, die Nummer des akutellen EDMA-Blocks `block` und die interne Kanalreihenfolge `ch_corr` des EDMA. In der Funktion wird zunächst der Lesezeiger an den aktuellen Winkel angepasst (in Codeabschnitt 7.13 am Beispiel eines positiven Winkels gezeigt).

Code 7.13: Einstellunge des Lesezeigers in der VDL

```
1 if(tau >=0){                              //If greater or equal to zero
2     for(i = 0 ; i < MICS ; i++){
3         k = tau*i;                            //sampledelay for one microphone i
4         rptr[i] = wptr[i] - k;              //new readpointer!!
5         while (rptr[i] < delays[i]) {    //if assault to the next field!
6             rptr[i] += DELAYLENGTH;             //then add Length of Field.
7         }                                       //--> Its some kind of modulo
8     }   //end for
9 }   //end if
```

Für jedes Mikrofon wird die Verzögerung in Samples berechnet (Zeile 3) und anschließend der Lesezeiger neu eingestellt (Zeile 4). Sollte die Variable `rptr` dadurch aus dem Feld delays hinauslaufen, muss `DELAYLENGTH` zum Zeiger hinzuaddiert werden (Zeilen 5-6). Diese Operation entspricht einer Modulo-Operation. Codeausschnitt 7.14 zeigt das anschließende Beschreiben der VDL mit den Eingangsdaten vom ADU (Zeile 4) und Ausgabe der Daten über den DAU (Zeile 5).

Code 7.14: Belegung der VDL, bzw. Zugriff auf Daten in der VDL

```
1  //Now the output is defined (later the dacChannels!!!
2  for(i = 0 ; i < MICS ; i++) {
3      for(j = 0 ; j < EDMABUFLEN ; j++) {
4          *wptr[i]++ = adcbuffer[ch_corr[i]][block][j];
5          dacbuffer[ch_corr[i]][block][j] = *rptr[i]++;       //get the actual value
               from line
6
7          if ((wptr[i]-delays[i]) >= DELAYLENGTH) {               //some kind of
               modulo:
8              wptr[i] -= DELAYLENGTH;
9          }
10         if ((rptr[i]-delays[i]) >= DELAYLENGTH) {               //some kind of
               modulo:
11             rptr[i] -= DELAYLENGTH;
12         }
13     }
14 }
```

Ab Zeile 7 wird wieder eine Art Modulo-Operation durchgeführt, da die Lese- und Schreibezeiger in Zeilen 4 und 5 durch das Inkrementieren aus dem Feld gelaufen sein könnten.

7.4.8. Medianfilter

Wie in Unterabschnitt 6.3 erläutert, wird ein Medianfilter eingesetzt, um Ausreißer bei der Winkelberechnung zu erkennen und herauszufiltern. Der Medianfilter ist in der Funktion `Tracker` implementiert[20]. Diese erhält das Array `EstDoa`, also die letzten detektierten

[20] Der Begriff Tracking kann etwa mit Nachführung übersetzt werden. Diese Methode dient im Allgemeinen dazu, aus einem Strom von Beobachtungsdaten Informationen über den Verlauf der Bewegung eines Objektes zu extrahieren und Abweichungen zu minimieren.

Winkel, sowie `AlgCounter`, der den aktuellen Winkel in dem Array markiert. Dort wird eine Anzahl von `MEDIANORDER` Daten aus dem Array `EstDoa` geholt, zunächst sortiert und anschließend aufsteigend in die richtige Reihenfolge gebracht. Eine Besonderheit beim Zugriff auf `EstDoa` ist im folgenden Codeausschnitt dargestellt.

Code 7.15: Zugriff auf die Winkel im Array EstDoa

```
1 //First put the actual and the last two values in the median_Array
2         for(i = 0 ; i < MEDIANORDER ; i++){
3             tmp_Integer = AlgCounter-i;
4             if(tmp_Integer < 0)     //Some Kind of Modulodecision, because EstDoa is
                  "Moduloarray"
5                   tmp_Integer = NUM_OF_DOA_SAVE + tmp_Integer;
6             median_Array[i] = EstDoa[tmp_Integer];
7         }
```

Es müssen die letzten beiden Winkel, sowie der aktuelle Winkel aus `EstDoa` geholt werden. Wenn aber die Laufvariable `AlgCounter` null oder eins ist, dann würde `tmp_integer` in Zeile drei einen negativen Wert erhalten. Da in Zeile sechs mit Hilfe dieser Variable auf `EstDoa` zugegriffen wird, darf sie nicht negativ werden, sondern muss in dem Fall auf `NUM_OF_DOA_SAVE + tmp_Integer` (Zeile 5) gebracht werden. Die Sortierung erfolgt anschließend über ein einfaches Bubblesort-Verfahren.
Da der Medianfilter den Winkel verändert, der eine bestimmte Zeit in der Vergangenheit detektiert wurde, muss die Länge des Filters unter Berücksichtigung der Stationaritätseigenschaften von Sprachsignalen eingestellt werden. Eine Filterlänge von $M_{Median} = 11$ bedeutet beispielsweise, dass der Winkel, der vor $\frac{M_{Median}-1}{2} = 6$ EDMA-Blöcken berechnet wurde, als aktueller Winkel angenommen wird. Dementsprechend wird vorrausgesezt, dass sich die Quelle innerhalb der letzten sechs Blöcke nicht bewegt hat und sich außerdem die statistischen Eigenschaften nicht verändert haben. Wird eine Länge von $t_{quasi} = 50ms$ angenommen, in der das Sprachsignal Quasistationariät aufweist, so muss die Filterlänge folgender Beziehung genügen:

$$t_{quasi} \geq \frac{M_{Median} - 1}{2} \cdot \frac{L_{EDMA}}{f_A}. \tag{7.3}$$

Mit einer Abtastrate von 48kHz - deren Verwendung wie in der Simulation gezeigt, gute Ergebnisse liefert - und einer EDMA-Blocklänge von 800 würde sich aus Gleichung 7.3 beispielsweise eine maximale Länge des Medianfilters von sieben ergeben.

7.4.9. Ansteuerung der Seriellen Schnittstelle

Zu Debugzwecken können die Winkel über eine serielle Schnittstelle ausgegeben und in einem Terminal am Rechner angezeigt werden. Durch Auskommentierung des Makros

TERMINALOUTPUT kann die Ausgabe deaktiviert werden. Der folgende Codeausschnitt zeigt die Vorbereitung der Daten und das anschließende Senden über die Schnittstelle.

Code 7.16: Ansteuerung der Seriellen Schnittstelle zu Debugzwecken

```
1     tmpDummyShort = (short)EstDoa[AlgCounter-MEDIANMIDDLE];
2     if(tmpDummyShort == 0){
3         p2s = stoa( 0 );
4         p2s[0] = '0';
5     }
6     else if(tmpDummyShort>0){    //greater then zero
7         p2s = stoa( (short)(DegreeResolution*(tmpDummyShort+1)));
8         p2s[0] = '+';
9     }
10    else{
11        p2s = stoa( (short)(-DegreeResolution*(tmpDummyShort-1)) );
12        p2s[0] = '-';
13    }
14        send_string ( p2s ); send_string(" \r\n");
```

Es werden zwei im Demoprogramm enthaltene Funktionen verwendet. stoa wandelt ein 16-Bit Integer in eine Zeichenkette um, send_string sendet die Daten über ein FIFO-Register. In Zeile 1 wird der aktuelle Winkel in einer Shortvariablen gespeichert. Dabei muss natürlich der durch den Medianfilter angepasste Winkel AlgCounter-MEDIANMIDDLE als aktueller Winkel verwendet werden. Anschließend folgt eine Unterscheidung, ob der Winkel gleich null, negativ oder positiv ist, damit ein '+' oder '-'Zeichen mitgesendet wird. Die Funktion stoa wandelt einen Wert von 1 in den Wert 00000 um, 2 in 00001 usw. Dementsprechend muss zum tmpDummyShort 1 addiert werden, wenn es sich um einen positiven Winkel handelt und 1 abgezogen werden, wenn der Winkel negativ ist. Die Multiplikation mit DegreeResolution - die mögliche Winkelauflösung, die zu Beginn der Hauptfunktion berechnet wird - bewirkt eine Anzeige des Winkels in Grad.

8. Testergebnisse

In diesem Abschnitt werden die Ergebnisse der Implementierung dargestellt. Zunächst wird ein sog. Profiling durchgeführt, um zu prüfen, wie viele Taktschritte die einzelnen Funktionen benötigen und die Umgebungsparameter endgültig einzustellen. Anschließend wird die grundlegende Funtktionsweise verifiziert, woraufhin Ergebnisse in schalldichter und realer Umgebung dargestellt werden.

8.1. Profiling und Festlegung der Umgebungsparameter

Das Profiling dient dazu, die Geschwindigkeit der einzelnen Funktionen in Abhängigkeit der Umgebungsparamter zu bestimmen. Damit kann eine Aussage darüber getroffen werden, wie die Umgebungsparameter letztendlich eingestellt werden müssen, damit die vorgegebene Anzahl von Taktschritten eingehalten werden kann.
Unter CCS darf das Profiling nur im Simulator durchgeführt werden, da es auf dem DSP falsche Ergebnisse liefert. Allerdings dürfen die Ergebnisse auch im Simultor nur als Abschätzung behandelt werden, da nicht klar ist, wieviele Taktschritte später auf dem DSP z. B. durch die EDMA-Behandlung und der anderen Peripherie verbraucht werden.

8.1.1. Vergleich zwischen SLP und MCCC

Tabelle 8.1 zeigt einen Vergleich zwischen SLP und MCCC für die Berechnung der Fehlerfunktion eines Winkels in einem ULA mit `MICS`=8. Zu erkennen ist, dass die SLP mehr als doppelt so viele Taktschritte benötigt als die MCCC. Dies ist auf die aufwendige Matrixinversion zurückzuführen, die fünf mal so viel Zeit benötigt wie die Berechnung der Determinante in der MCCC. Wird sogar die Normierung auf die Standardabweichungen aus der MCCC genommen, reduziert sich $C_{MCCC,ULA}$ auf ein fünftel von $C_{SLP,ULA}$. Dasselbe Bild ergibt sich in Tabelle 8.2, die einen Vergleich zwischen SLP und MCCC für die Berechnung der Fehlerfunktion eines Winkels in einem UCYA mit `MICS`=8 zeigt. Wird ein UCYA verwendet, werden knapp 2000 Taktschritte mehr benötigt, da die Berechnung der Funktion $F(\tau)$ durchgeführt werden muss. Gegenüber der MCCC verliert die SLP sogar noch mehr an Geschwindigkeit, da $F(\tau)$ zweimal berechnet wird (zur Extrahierung von $\mathbf{R}_a$ und r_a).

MCCC		SLP	
Funktion	Taktschritte	Funktion	Taktschritte
ExtractCorrelationMatrixToRa	738	ExtractCorrelationMatrixToRa	446
(NormToStandardDeviations	10863)	Extract_ra_FromCorrelationM.	40
CalcDeterminant	11613	MatrInv	55746
		Matr_Mult_C_o3 (1x8 · 8x8)	539
		Matr_Mult_C_o3 (1x8 · 1)	119
$C_{MCCC,ULA}$	**23214**	$C_{SLP,ULA}$	**56890**

Tabelle 8.1.: Vergleich zwischen MCCC und SLP für eine Fehlerberechnung, ULA

MCCC		SLP	
Funktion	Taktschritte	Funktion	Taktschritte
ExtractCorrelationMatrixToRa	3486	ExtractCorrelationMatrixToRa	3300
(NormToStandardDeviations	10863)	Extract_ra_FromCorrelationM.	2600
CalcDeterminant	11506	MatrInv	55746
		Matr_Mult_C_o3 (8x8 · 1x8)	539
		Matr_Mult_C_o3 (1x8 · 1)	119
$C_{MCCC,UCYA}$	**25855**	$C_{SLP,UCYA}$	**62304**

Tabelle 8.2.: Vergleich zwischen MCCC und SLP für eine Fehlerberechnung, UCYA

Die oben dargestellten Werte gelten für die Berechnung *eines* Winkels, also eines Wertes in der Fehlerfunktion J. Wird der gesamte Winkelbereich nach der Quelle abgesucht, erhöht sich die Anzahl abhängig von der Winkelauflösung. Für ein ULA wurde ein Bereich von $\pm 80^\circ = 160^\circ$ festgelegt. Mit einer Abtastrate von 48kHz und der daraus resultierenden Auflösung von 8.2167° müssen also Fehler für 19 verschiedene Winkel berechnet werden. Tabelle 8.3 zeigt die resultierenden Taktschritte für ein ULA. Aufgrund dieser Werte wird nochmals deutlich, dass die Normierung auf die Standardabweichungen nur dann durchgeführt werden sollte, wenn es unbedingt notwendig scheint.
Tabelle 8.4 zeigt die resultierenden Taktschritte für ein UCYA, wenn der gesamte Winkelbereich abgesucht wird. Aus dem Winkelbereich von 360° und einer Auflösung von 8.2167° ergeben sich 43 Winkel, deren Fehler berechnet werden müssen. Die SLP würde nun fast 10ms für den eigentlichen Algorithmus benötigen. Damit ist zu bezweifeln, dass die SLP mit der gegebenen Zielsetzung überhaupt verwendet werden kann. Die MCCC benötigt ohne Normierung immer noch nur knapp ein Viertel der Zeit der SLP. Da die MCCC wegen des mathematischen Ansatzes ohnehin eine höhere Genauigkeit hat, werden die späteren Messungen ausschließlich mit der MCCC durchgeführt.

MCCC		SLP	
Funktion	Taktschritte	Funktion	Taktschritte
ExtractCorrelationMatrixToRa	13030	ExtractCorrelationMatrixToRa	8484
(NormToStandardDeviations	205370)	Extract_ra_FromCorrelationM.	760
CalcDeterminant	219797	MatrInv	1059174
		Matr_Mult_C_o3 (1x8 · 8x8)	10241
		Matr_Mult_C_o3 (1x8 · 1)	2261
$C_{MCCC,ULA}$	**438197**	$C_{SLP,ULA}$	**1080920**

Tabelle 8.3.: Vergleich zwischen MCCC und SLP für den gesamten Winkelbereich, ULA

MCCC		SLP	
Funktion	Taktschritte	Funktion	Taktschritte
ExtractCorrelationMatrixToRa	171104	ExtractCorrelationMatrixToRa	162920
(NormToStandardDeviations	471313)	Extract_ra_FromCorrelationM.	122062
CalcDeterminant	506616	MatrInv	2452824
		Matr_Mult_C_o3 (8x8 · 1x8)	23716
		Matr_Mult_C_o3 (1x8 · 1)	5236
$C_{MCCC,UCYA}$	**1149033**	$C_{SLP,UCYA}$	**2766758**

Tabelle 8.4.: Vergleich zwischen MCCC und SLP für den gesamten Winkelbereich, UCYA

8.1.2. Taktschritte bei der Interpolation

Es soll nun geprüft werden, wie sich das Polyphasenfilter auf die Rechenzeit des Algorithmus auswirkt. Ein Problem bei der Verwendung des Polyphasenfilters ergibt sich daher, dass L_{FFT} und L_{Sig} an den Interpolationsfaktor angepasst werden müssen. Soll eine gewisse Länge der FFT eingehalten werden, muss bei einer hohen Interpolationsrate die Signallänge L_{Sig} unter Berücksichtigung der maximalen Verzögerung verringert werden. Dies hätte jedoch zur Folge, dass die Energieberechnung über weniger Abtastwerte liefe, was unter Umständen zu Energieberechnungen führen würde, die wenig aussagekräftig wären. Wird jedoch L_{FFT} erhöht, gibt es Speicherprobleme auf dem DSP. Es hat sich gezeigt, dass der Speicher mit einer FFT-Länge von 256 ausreicht. Dementsprechend wird diese FFT-Länge als Obergrenze angenommen.
Es wurden Filter erstellt, die das Eingangssignal von verschiedenen Abtastraten auf eine Länge von 128 interpolieren. Diese Eingangssignale haben gleichzeitig immer eine Länge von mindestens 32 Abtastwerten, damit die Energieberechnung aussagekräftig bleibt und die maximal mögliche Verzögerung berücksichtigt wird.

Tabelle 8.5 zeigt, dass eine Interpolation im Vergleich mit dem einfachen Kopieren mindestens doppelt so viele Taktschritte benötigt, sodass nach Möglichkeit eine hohe Abtastrate

Abtastrate	B	L_{Sig}	Filterordnung	Taktschritte
8kHz	2	64	68	743738
8kHz	3	42	102	727411
8kHz	4	32	136	728374
16kHz	2	64	136	1048940
16kHz	3	42	204	1076215
16kHz	4	32	271	1082737
24kHz	2	64	204	1394040
24kHz	3	42	305	1380982
24kHz	4	32	407	1337769
48kHz	2	64	407	2285632
Ohne Interpolation (nur kopieren)				
wahlweise	1	128	–	358757
wahlweise	1	64	–	179369

Tabelle 8.5.: Taktschritte bei der Interpolation bzw. Kopie der Eingangsdaten

verwendet werden soll. Werden 8kHz verwendet, können diese nicht auf 48kHz interpoliert werden, da die FFT-Länge sonst 256 Punkte überschreiten und dann der Speicher auf dem DSP nicht mehr ausreichen würde. Maximal ist eine Interpolation auf 32kHz möglich, was einer Winkelauflösung von etwa 12° entspräche. Für diesen Vorgang müssten knapp 2.3ms aufgewendet werden. Es soll nun festgelegt werden, dass der EDMA auf 48kHz arbeiten soll, damit eine ausreichende Winkelauflösung vorliegt und keine Interpolation notwendig ist.

8.1.3. Profiling des gesamten Programms

Im Folgenden wird die Geschwindigkeit des Algorithmus mit verschiedenen Einstellungen der Umgebungsparameter getestet und mit dem rechnerisch erlaubten T_{soll} verglichen. Dabei wird aber darauf verzichtet, das Profiling für sämtliche mögliche Einstellungen zu zeigen. Vielmehr wird ein Beispiel für die Einstellung der Makros gegeben. Aufgrund der Erkenntnisse im Vergleich zwischen MCCC und SLP wird für die Einstellung geeigneter Umgebungsparameter nur die MCCC verwendet.

Die Entscheidung, ob die Taktschritte ausreichend sind, kann erst unter Berücksichtigung der EDMA-Länge getroffen werden. Werden die Ergebnisse für C_{Gesamt} in Gleichung 5.4 als K_{soll} eingesetzt, ergiben sich EDMA-Längen von 168 für das ULA und 178 für das UCYA.
Die Ergebnisse des Profilings werden folgendermaßen zusammengefasst:

- Die SLP benötigt aufgrund der Matrixinversion fünfmal so viele Taktschritte wie eine MCCC ohne Normierung.

Parameter	**Einstellung**	
Arrayaufbau	ULA	UCYA
Mikrofone	8	
Signallänge	128	64
FFT-Länge	256	128
Ordnung Medianfilter	3	
Funktion	**Taktschritte**	
Energieberechnung	42653	20265
Kopie der Daten	358757	179369
Zeropadding	3127	550
8x FFT	43421	19000
36x KKF	260489	106380
MCCC	232939	677858
Tracker	101	
Vezögerungskette	112304	
C_{Gesamt}	1053791	1115827

Tabelle 8.6.: Taktschritte für den gesamten Algorithmus

- Im Hinblick auf den gesamten Algorithmus macht es keinen Unterschied, ob ein UCYA oder ULA verwendet wird, solange die Anzahl der analysierten Daten im UCYA halbiert wird.
- Eine Interpolation wird vermieden, indem der EDMA mit 48kHz betrieben wird.
- Die EDMA-Länge muss länger sein als die eigentliche Anzahl von Daten, die im Algorithmus verwendet wird.

8.1.4. Endgültige Einstellung der Parameter

Um das Profiling abzuschließen, sollen nun die Umgebungsparameter endgültig eingestellt werden. Die Basis dafür bilden die Ergebnisse in den vorangegangenen Unterabschnitten, sowie die der Simulationen.
Die Mikrofonarrays werden mit allen acht Mikrofonen benutzt. Die Abtastrate des EDMA wird auf 48kHz eingestellt. Damit ist der Interpolationsfaktor auf 1 einzustellen, es wird also keine Interpolation durchgeführt. Unter Verwendung eines ULA wird die Länge der Daten, die aus dem EDMA geholt werden, auf 128 eingestellt, womit die Länge der FFT auf 256 festgelegt wird. Auch wenn sich gezeigt hat, dass längere FFTs - also mehr Werte, über die korreliert wird - eine bessere Winkelerkennung für Sprachsignale gewährleisten, kann die Anzahl nicht so hoch gesetzt werden. Speicher und Rechenzeit sind begrenzt, weswegen

ein Kompromiss gefunden werden muss. Die EDMA-Länge wird auf 900 festgelegt. Wie im vorangegangenen Unterabschnitt dargestellt, würde theoretisch eine EDMA-Länge von 168 reichen. In der Praxis reicht dieser Wert aber nicht aus.
Mit diesen Parametern ergibt sich eine maximal erlaubte Medianfilterlänge von sieben, aufgrund von Rechenzeitersparnis wird die Länge auf drei festgelegt. Als Energielimit hat sich der Pseudowert 1 als geeignet erwiesen. In diesem Fall werden die Pausen eines Sprechers mit normaler Lautstärke, der etwa zwei Meter vom Array entfernt steht, komplett detektiert.
Für das UCYA werden lediglich die Signal- und FFT-Länge auf jeweils die Hälfte gesetzt. Alle anderen Einstellungen bleiben gleich.

8.2. Tests in Echtzeit

Zur Verifikation der Funktionsweise des programmierten Algorithmus soll nun das Verhalten des Systems untersucht werden. Hierzu wurde die Funktion zunächst im Schallmessraum unter perfekten Bedingungen geprüft. Anschließend wurde das System in einer verhallten Umgebung aufgebaut und getestet. In den folgenden Unterabschnitten werden zunächst der Aufbau des Systems und danach die Tests in den jeweiligen Umgebungen erläutert.

8.2.1. Aufbau des Systems

Das Mikrofonarray wird vor den beiden Lautsprecher aufgebaut. Die Mikrofonverstärker sind auf einer extra Adapterplatine aufgebaut, deren Ausgangssignale in den ADU des ersten DSP geführt werden, auf dem die Richtungserkennung abläuft. In Kaskade mit diesem DSP wird ein zweiter geschaltet, auf dem der GJBF läuft und der die DAU-Signale des ersten in seine ADU-Eingänge erhält. Abbildung 8.1 zeigt den Aufbau des gesamten Aufbaus skizzenhaft.

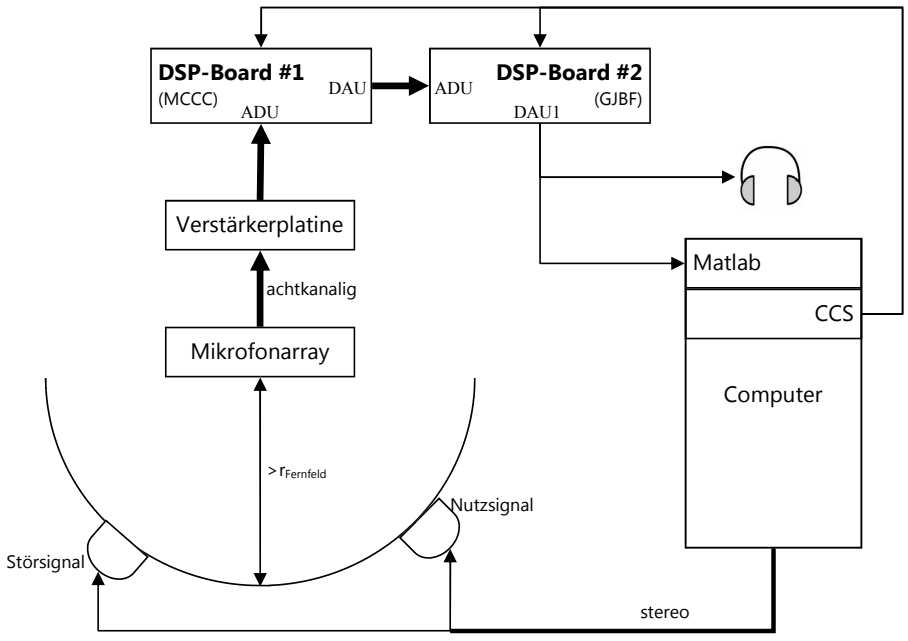


Abbildung 8.1.: Skizze des Aufbaus

Zur Vereinfachung wurde in der Skizze nur ein zentraler Computer dargestellt. Da die Line-In und Line-Out Anschlüsse ein Übersprechen verursachen, müssen bei den Tests, die zeitgleiches Aufnehmen und Abspielen erfordern, zwei Rechner benutzt werden. Des Weite-

ren werden die beiden DSPs von verschiedenen Rechnern aus gesteuert. Die Lautsprecher müssen im Abstand von $r_{Fernfeld}$ zum Mikrofonarray aufgebaut werden, damit die Fernfeldbedingung eingehalten wird und die Audiowellen vom Lautsprecher quasiebene Wellen am Mikrofonarray darstellen. Der Lautsprecher sei an dieser Stelle als Punktquelle (Monopol) angenommen, der Kugelwellen ausstrahlt. Mit dieser Vereinfachung kann der Radius, nach dem das Fernfeld beginnt, über folgende Bedingung bestimmt werden ([Gör08, S. 36f.], [Bus05, S.15]):

$$\frac{2\pi}{\lambda_{max}} \cdot r_{Fernfeld} >> 1. \tag{8.1}$$

λ wird durch die maximal relevante Frequenz bestimmt, die für Sprachanwendungen bei 3.4kHz liegt. Für den Versuchsaufbau wird ein Radius von 1.60m gewählt, mit dem die Fernfeldbedingung erfüllt wird.
Die Versuche im Schallmessraum unterliegen nahezu perfekten Bedingungen, da keinerlei Reflektionen der Signale von Wänden oder Boden auftreten. Alle Schallreflektierenden Flächen sind mit 80cm tiefen Schaumstoffkeilen bestückt, die den Schall im hörbaren Bereich unterdrücken. Von Außen dringt zudem kein Geräusch in den Raum, da der SNR auf 40dB eingestellt ist. Der Test in verhallter Umgebung geschieht in einem Labor des Departments, um zu ermitteln, wie sich der Algorithmus in einer schlechtest möglichen Umgebung verhält.

8.2.2. Test des entwickelten Algorithmus

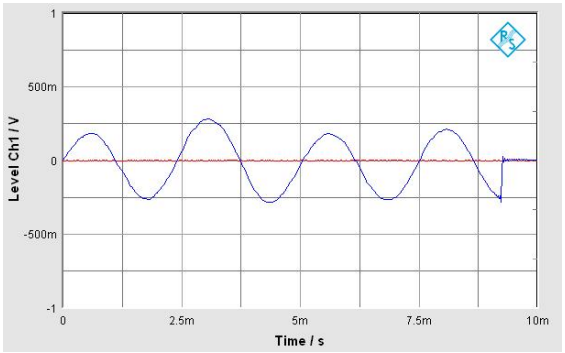
Folgende fünf Tests wurden durchgeführt:

1. Verifikation der EDMA-Länge
2. Test der grundlegenden Funktion
3. Ermittlung des Winkelbereichs beim Linearen Mikrofonarray
4. Verhalten mit Störungen in der Umgebung
5. Kaskadierung mit dem GJBF

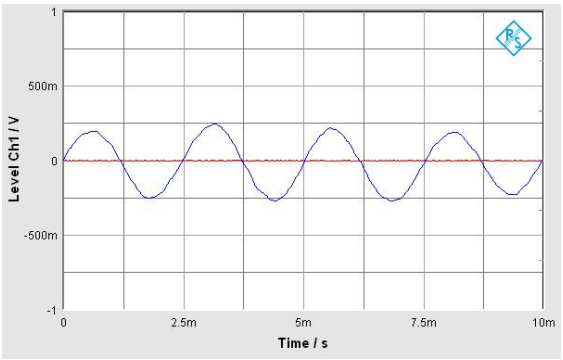
Tests 2 und 3 wurden in beiden Umgebungen, die Tests 1,4 und 5 nur in der verhallten Umgebung durchgeführt. Im Folgenden werden die Ergebnisse dargestellt.

Verifikation der EDMA-Länge

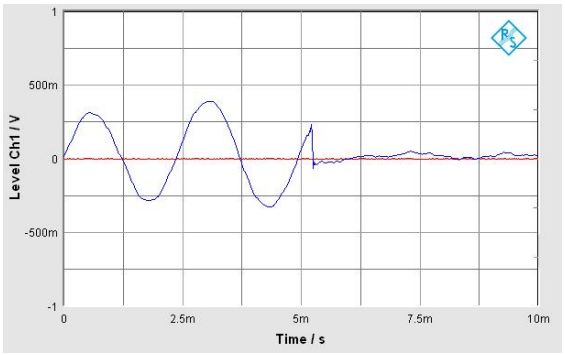
Eines der anfangs gestellten Bedingungen, ist die *Zuverlässigkeit*. Da eine Spracherkennung durchgeführt werden soll, darf das Sprachsignal nicht zu stark verfälscht werden. Ein Parameter, der groß genug dimensioniert werden muss, damit die Verfälschung nicht eintritt, ist die EDMA-Länge. Wird diese zu kurz gewählt, werden Abtastwerte im EDMA überschrieben, bevor sie im Algorithmus verwendet wurden.
Zunächst wurde ein Sinussignal mit 400Hz abgespielt und nach dem Algorithmus auf dem Board über DAU-Ausgang 1 ausgegeben. Das Ausgangssignal wurde anschließend mit einem Audio Analyzer aufgenommen und angezeigt. Unter Verwendung der MCCC ergaben sich die Aufnahmen in Abbildung 8.2.



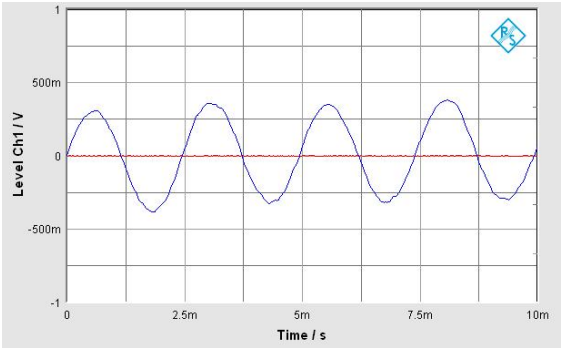
(a) ULA, EDMA-Länge 400, MCCC



(b) ULA, EDMA-Länge 800, MCCC



(c) UCYA, EDMA-Länge 400, MCCC



(d) UCYA, EDMA-Länge 800, MCCC

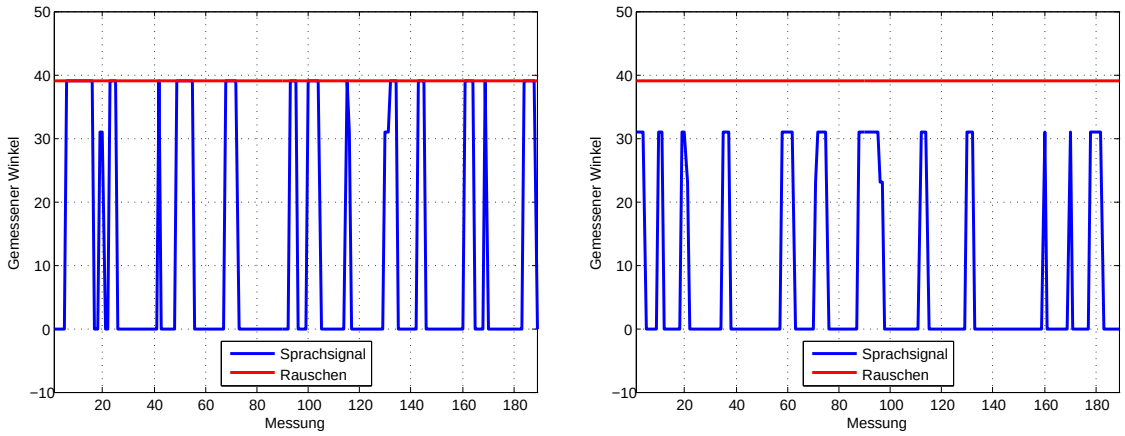
Abbildung 8.2.: Verifikation der EDMA-Länge anhand eines Sinussignals

Beide Algorithmen wiesen bei einer EDMA-Länge von 800 Abtastwerten keine Unterbrechungen der Sinusschwingungen auf. Zwischen einer Länge von 400 und 800 traten Unterbrechungen sehr sporadisch auf, bei 400 fast in jeder Darstellung auf dem Audio Analyzer.
Im Anhang befinden sich im Pfad „04_Audiodateien\VeriEDMALaenge“ Beispiele

für Signale, die bei verschiedenen EDMA-Längen aufgenommen wurden. Die Datei „Test_1_MCCC_EDMALEN500_UCYA_KNACKEN.wav“ ist dort als Beispiel für ein verfälschtes Signal abgelegt. Deutlich ist dort ein Knacken zu hören, was auf das Überschreiben der Daten zurückzuführen ist. „Test_1_MCCC_EDMALEN900_UCYA.wav“ ist ein Beispiel für ein störfreies Signal.

Grundlegende Funktion

Für den Test der Grundfunktion des Algorithmus werden in beiden Umgebungen weißes Rauschen und anschließend ein Sprachsignal abgespielt. Abbildung 8.3 zeigt den Vergleich der Winkelverläufe in beiden Umgebungen für das ULA.



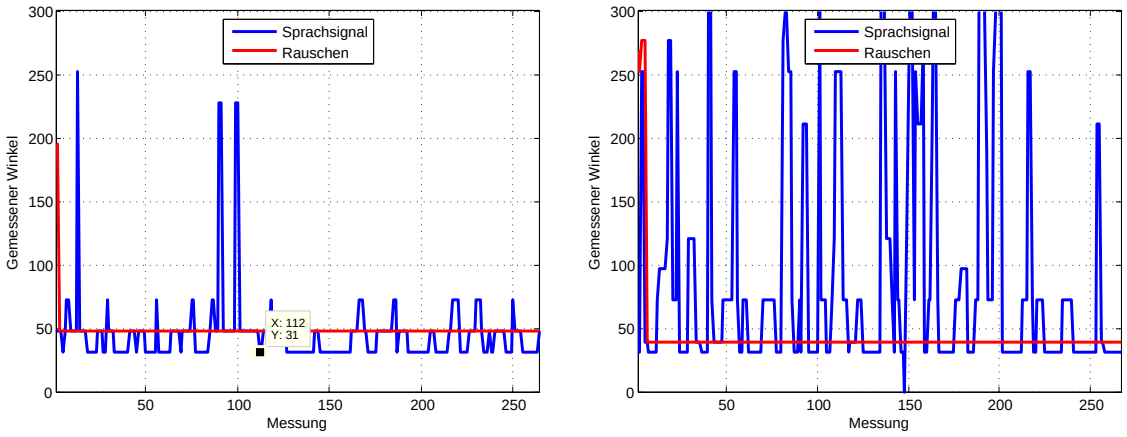
(a) Winkelverlauf, Schallmessraum, ULA

(b) Winkelverlauf, verhallte Umgebung, ULA

Abbildung 8.3.: Grundlegende Verifikation des Algorithmus, ULA

Wie zu erkennen ist, springt der ermittelte Winkel für das Sprachsignal zwischen 0° und knapp 40° hin und her. Dieser Effekt tritt in beiden Umgebungen auf, wobei in der verhallten Umgebung tendenziell mehr DOAs von 0° gemessen worden sind, was auf die Reflektionen von den Wänden zurückzuführen ist. Da die schalldichte Umgebung nahezu perfekten Bedingungen unterliegt, weisen die Sprünge auch auf ein generelles Problem des Algorithmus bei der Detektion von Sprachsignalen hin. Außerdem könnte die Fehlerfunktion J keine ausgeprägten Minima besitzen. Vergleicht man diese Ergebnisse mit denen für ein UCYA (Abbildung 8.4), fällt zudem eine Schwäche des ULA auf. Wenn die Daten dort fehlerhaft analysiert werden, dann wird, so scheint es, immer ein Winkel von 0° angenommen.

In Abbildung 8.4(b) treten fehlerhafte Messungen bei etwa 250°, 300°, 210° und 75° auf. Dies zeigt, wie stark Reflektionen an Wänden das Ergebnis verfälschen können.



(a) Winkelverlauf, Schallmessraum, UCYA (b) Winkelverlauf, verh. Umgebung, UCYA

Abbildung 8.4.: Grundlegende Verifikation des Algorithmus, UCYA

Winkelbereich des ULA

In der späteren Anwendung wird es sinnvoll sein, das UCYA zu verwenden, weil es den gesamten Umkreis erfassen kann. Trotzdem wird die Qualität der MCCC hier anhand des Winkelbereichs des ULA überprüft. Es wurde zunächst ein fünf Sekunden langes weißes Rauschen in 10° Schritten abgespielt. Die Winkel wurden über die RS-232 Schnittstelle aufgenommen und anschließend die Streuung berechnet. Abbildung 8.5 zeigt den Vergleich des Winkelverlaufs für ein ULA zwischen schalldichter und verhallter Umgebung.

Die Ergebnisse in beiden Umgebungen unterscheiden sich nur geringfügig voneinander. In beiden Umgebungen bleibt der Fehler des DOA unter 20°, was wegen des GJBF als Grenze festgelegt wurde. Zwischen -50° und-10° ist aber im Winkelverlauf der verhallten Umgebung eine große Abweichung erkennbar. Dies ist auf die Ungenauigkeit im Versuchsaufbau zurückzuführen, da der Winkel des Lautsprechers zum Array nicht ausreichend genau genug eingestellt werden konnte. Kleine Abweichungen - es reichen wenige Zentimeter - führen wegen der Winkelauflösung von 8° zu großen DOA-Abweichungen.
Abbildung 8.6 zeigt den Winkelverlauf, der mit einem Sprachsignal aufgenommen wurde. Aufgrund der Erkenntnisse in der grundlegenden Verifikation werden die Messdaten einer Glättung unterzogen. Die DOAs von 0° werden entfernt. Ein Sprachsignal kann dementsprechend zwischen -60° und +60° mit einer ausreichenden Genauigkeit detektiert werden.

Auffällig ist die hohe Abweichung in der verhallten Umgebung zwischen 10° und 60°. Während die Winkel im negativen Bereich sehr eng an denen der schalldichtern Umgebung liegen - maximal 5° Unterschied -, weichen die Winkel im positiven Bereich mit 15° sehr stark ab. Der Grund dafür könnten wieder Ungenauigkeiten bei der Richtungseinstellung der Quel-

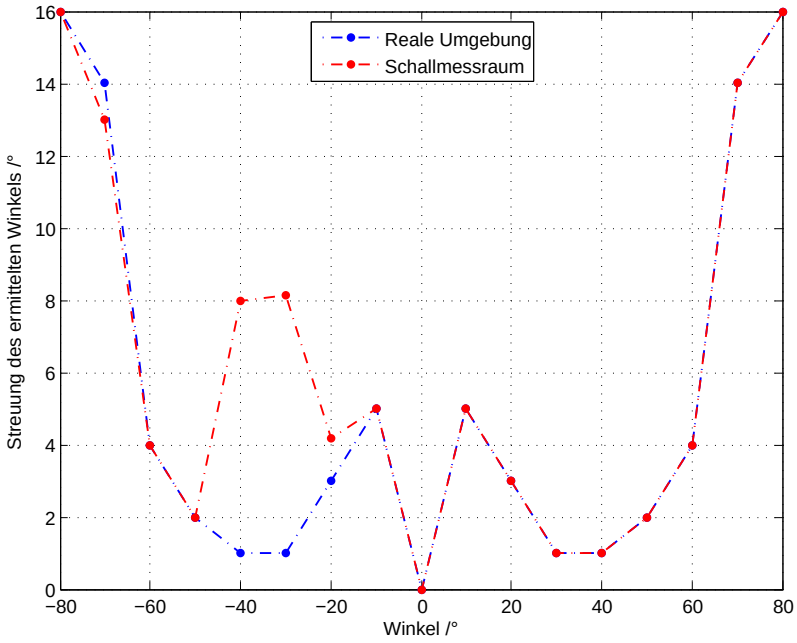


Abbildung 8.5.: Vergleich Schallmessraum mit verhallter Umgebung, Rauschen, ULA

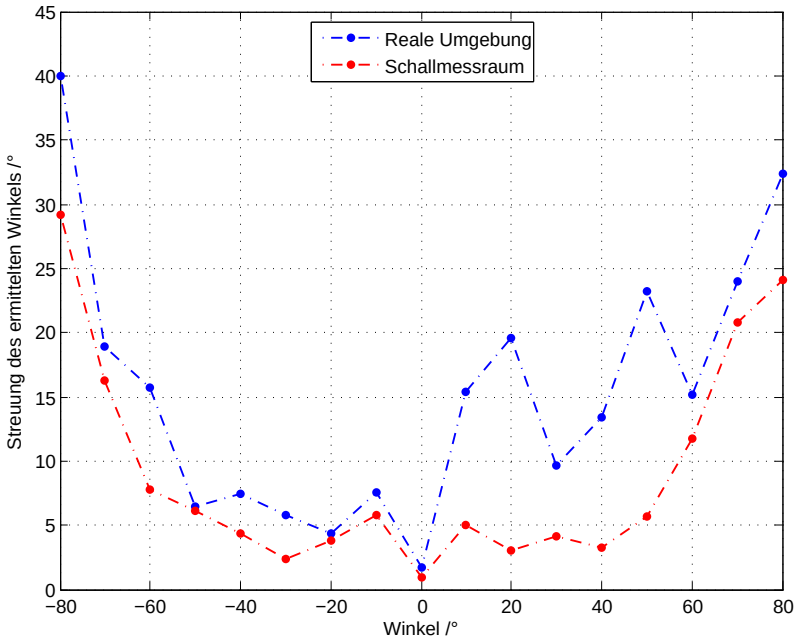


Abbildung 8.6.: Vergleich Schallmessraum mit verhallter Umgebung, Sprachsignal, ULA

le sein. Als die Quelle im positiven Bereich aufgestellt war, wurden aber bis zu 9% mehr Winkel von 0° gemessen, als wenn sie im negativen Bereich stand. Dies wiederum spricht für einen größeren Einfluss von Reflektionen. Abbildung 8.7 zeigt eine Skizze der verhallten Umgebung, in der gemessen wurde. Dort war genau am Rand des positiven Winkelbereichs eine ebene Reflektionsfläche, deren Abstrahlung quasi mit der gleichen Dämpfung am ULA ankam wie die direkte Schallwelle. Dadurch ergaben viele Messungen einen Winkel von 0°, was nochmals die Schwäche des ULA aufzeigt.

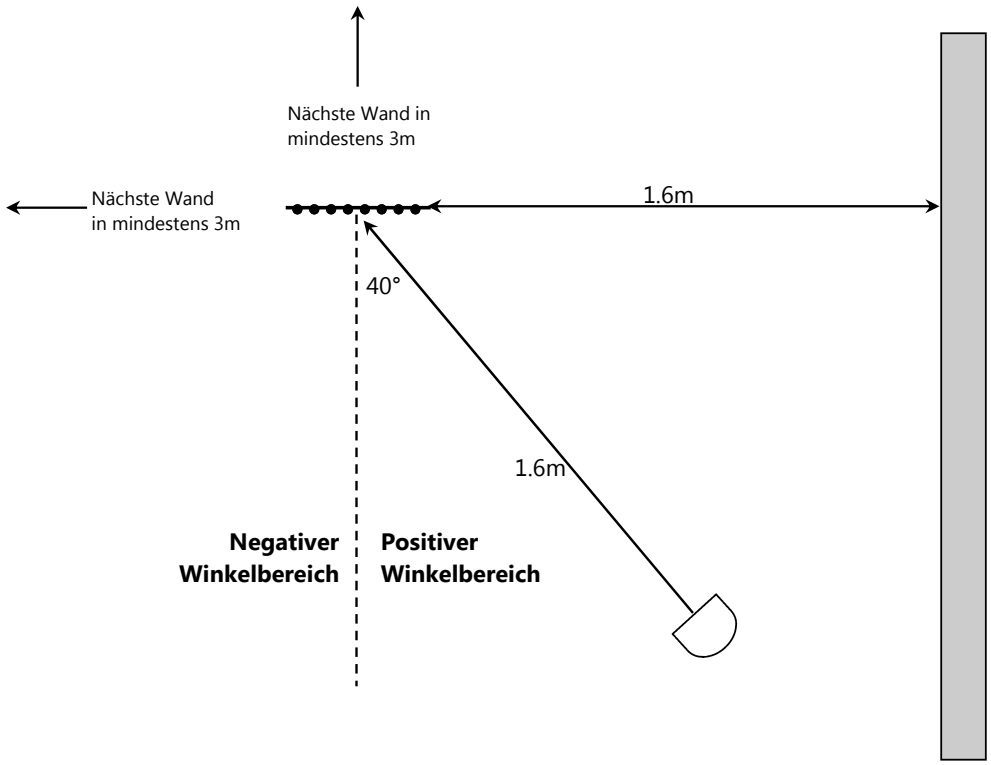


Abbildung 8.7.: Skizze der verhallten Umgebung

Der Systematische Fehler, der in Abschnitt hergeleitet wurde, tritt auch in den Abbildungen 8.5 und 8.6 auf. Mit einer Ausgleichsgerade könnte dieser Fehler ausgeglichen werden.

Untersuchungen hinsichtlich des Cocktailparty-Effekts

Im Gesamtsystem besteht die Aufgabe des Algorithmus darin, die Daten für die Rauschreduktion aufzubereiten. Liegt der Cocktailparty-Effekt vor, das heißt, existieren Rauschen und Störsignale in der Umgebung, so muss sich der Beamformer trotzdem nach dem Nutzsignal ausrichten. In diesem Unterabschnitt soll geprüft werden, inwieweit der Cocktailparty-Effekt

die Fehlerrate beeinflusst. Bevor in diesem Unterabschnitt die Ergebnisse der Kaskadierung mit dem GJBF gezeigt werden, soll das generelle Verhalten des Algorithmus auf definierte Störungen in der Umgebung geprüft werden.
Hierzu werden zwei Lautsprecher auf 30° bzw 330° vor dem UCYA angeordnet und anschließend weißes Rauschen aus beiden Lautsprechern abgestrahlt. Das Rauschen wird dabei ein- und ausgeblendet, wobei die beiden Signale gegenläufig sind. Abbildung 8.8 zeigt die Signalverläufe, sowie die blockweise berechnete Energie und den ermittelten DOA.

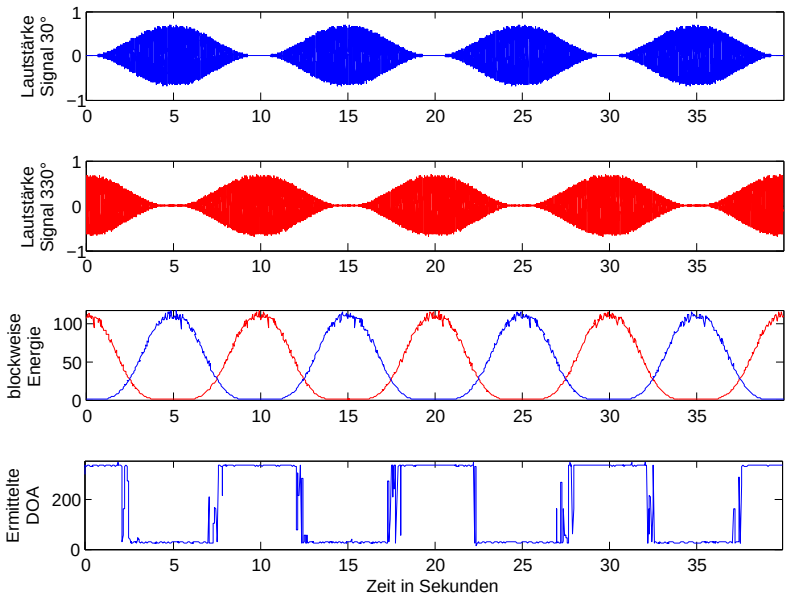


Abbildung 8.8.: Rauschen aus zwei Quellen, UCYA

Immer wenn sich die Energien „kreuzen“, also beide Signale gleichlaut sind, wechselt der Beamformer auf die jeweilige lautere Quelle. Der Wechsel ist entweder sehr klar (beim Zeitpunkt $t = 22.5$ Sekunden) oder erfolgt erst nach einigen Fehlmessungen (z. B. beim Zeitpunkt $t = 27.5$ Sekunden). Aus der Länge der Fehlmessungen bzw. aus der Energie, die in beiden Signalen vorliegt, wenn der Winkel klar erkannt wird, kann eine Aussage über die maximale Lautstärke der Störung gemacht werden. Zu diesem Zweck ist die Zeitspanne von 26 bis 29 Sekunden in der folgenden Abbildung vergrößert dargestellt.

Da die Amplitude der Energie abhängig ist von der Länge der Daten, über welche die Energie berechnet wird, ist in der Abbildung die Energiedifferenz prozentual zum Maximalwert dargestellt. Die roten Linien markieren die Grenzen, ab denen die Winkelerkennung weitesgehend konstant ist. Beträgt die Differenzenergie etwa 35%, kann das Nutzsignal konstant detektiert werden. Bei den anderen Übergängen in Abbildung 8.8 schwanken die Grenzen zwischen 6% und 30%. Es muss also ein gewisser Lautstärkeunterschied herrschen, damit sich der Beamformer nach dem Nutzsignal ausrichten kann. Eine genaue Grenze kann aber

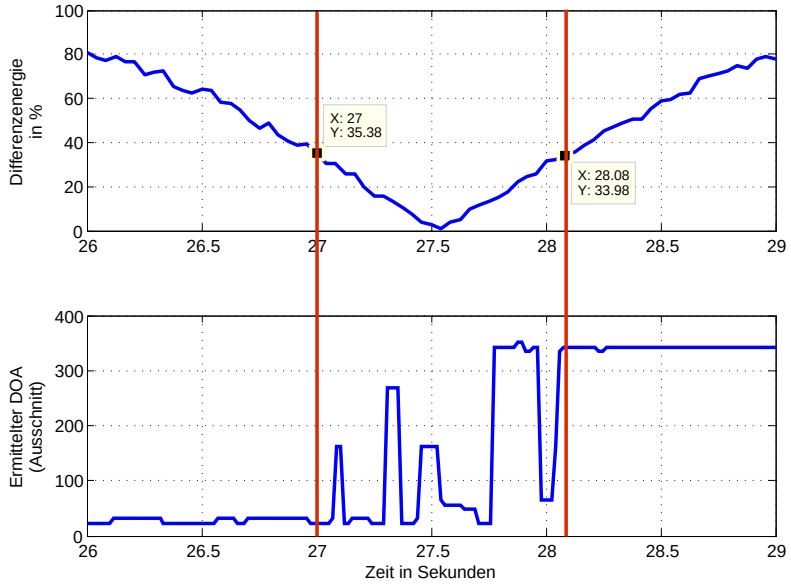


Abbildung 8.9.: Rauschen aus zwei Quellen, Ausschnitt, UCYA

nicht festgelegt werden.
Der Test der Kaskade aus Richtungserkennung und GJBF wurde in zwei Schritten durchgeführt. Zunächst wurde das Mikrofonarray simuliert, indem das Störsignal auf dem DSP mit einer künstlichen Verzögerung belegt und das Nutzsignal auf 0° belassen wurde. Anschließend wurde das Mikrofonarray mit eingebunden, ein Störsignal auf -40° gestellt und das Nutzsignal zwischen 0° und 45° variiert. Das Ausgangssignal vom GJBF wurde gleichzeitig aufgenommen und in einem wav-Signal gespeichert. Im Ordner „04_Audiodateien\Kaskade aus Richtungserkennung und GJBF“ im Anhang sind folgende wav-Dateien zu finden:

Original.wav ist das Stereosignal, das abgespielt wurde. Dabei liegt das Nutzsignal auf dem rechten, das Störsignal auf dem linken Kanal.

VerzKünstlichNutzsignal0grad_StörungMinus60.wav ist das Ergebnis, bei dem das Störsignal künstlich auf 60° verzögert wurde. Das Nutzsignal verblieb auf 0°.

Nutzsignalgrad_StörungMinus40_UCYA.wav** sind die Ergebnisse, bei denen das Störsignal auf -40° vor dem Mikrofonarray abgespielt wurde und das Nutzsignal nacheinander zwischen 0° und 45° variiert wurde. Dabei steht „**“ für 0°,15°,30° und 45°.

In dem Beispiel mit der künstlichen Verzögerung der Störung ist die Leistung der GJBF erkennbar. Das Störsignal liegt außerhalb des Bereicsh von $\pm 20°$ und wird daher fast vollständig ausgeblendet. Den direkten Vergleich zur Echtzeitumgebung zeigt die Datei „Nutzsignal0grad_StörungMinus40_UCYA.wav“. Dort ist das Umgebungsrauschen (in diesem Fall

eine Klimaanlagenlüftung), das bei den Messungen vorlag, gut zu hören. Trotzdem verschwindet das Störsignal fast vollständig. Wird der Winkel des Nutzsignals erhöht, so verschlechtert sich das Ergebnis geringfügig.
Abbildung 8.4 zeigte sehr viele Ausreißer bei der Detektion des Sprachsignals. Die wav-Dateien weisen trotz dieser vielen Ausreißer ein deutlich vermindertes Störgeräusch auf. Dies ist darauf zurückzuführen, dass der GJBF nicht als D-Typ, sondern nur als C-Typ auf dem kaskadierten DSP implementiert wurde. Der C-Typ dämpft die Störgeräusche nicht so stark ab (vgl. Abbildung 5.4), wodurch das Nutzsignal außerhalb der Toleranzgrenze von $\pm 20^\circ$ liegen darf. Wenn der GJBF als D-Typ implementiert würde, wäre es wahrscheinlich, dass das Nutzsignal in vielen Fällen ebenso abgedämpft wird, wie die eigentliche Störung.

9. Zusammenfassung

In dieser Masterarbeit wurde ein DSP-basiertes Echtzeitsystem entwickelt, mit dem der Winkel eines Sprachsignals detektiert werden kann. Außerdem werden die Eingangssignale mittels VDL so verzögert, dass sie quasi mit einer Verzögerung von 0° vorliegen. Der Algorithmus wurde zunächst unter der Simulationsoberfläche MATLAB aufgebaut. Anschließend wurde ein Programm auf dem DSP-Board d.module C6713 implementiert, das die Richtungserkennung in Echtzeit durchführt. Die Arbeit ist Teil eines Gesamtprojekts zur Spracherkennung.

Der Algorithmus kann in sechs Schritte unterteilt werden.

1. Energieberechnung eines Eingangssignals, um zwischen stimmhaften und stimmlosen Signalen zu unterscheiden,
2. Interpolation der Eingangsdaten, wenn nötig,
3. Kreuzkorrelation aller Eingangssignalpaare unter Verwendung einer handoptimierten Radix-2-FFT der Firma Texas Instruments,
4. weiterverwenden der Korrelationssignale in den beiden Algorithmen SLP und MCCC und Ermittlung der Richtung der Signalquelle,
5. Trackeralgorithmus mit Medianfilter, der den Winkelverlauf glättet und Ausreißer entfernt und
6. Verzögerung aller Mikrofonsignale in einer variablen Verzögerungskette, sodass die Signale über den Ausgang des DSPs mit einem Winkel von möglichst 0° ausgegeben werden.

Um die Umgebungsparameter endgültig einzustellen, wurde vor den Testläufen ein Profiling durchgeführt, mit dem die Geschwindigkeit der Programmfunktionen geprüft wurde. Anschließend wurden Echtzeittests in zwei Umgebungen durchgeführt. Zunächst wurde in einer schalldichten Umgebung geprüft, ob die grundlegende Funktionalität vorhanden ist und mit den Ergebnissen verglichen, die in verhallter Umgebung gemacht wurden. Anschließend wurden Tests hinsichtlich der späteren Anwendung des Gesamtsystems durchgeführt und geprüft, wie sich die Winkelerkennung bei vorhandenem Cocktailparty-Effekt verhält.

9.1. Beurteilung der Ergebnisse

Zu Beginn der Arbeit wurden einige Ziele definiert, deren Einhaltung nun diskutiert werden soll. Zu diesen Zielen zählte die Detektierbarkeit von breitbandigen Signalen, eine hohe Genauigkeit bei der Winkelerkennung, stabile Winkeldetektion auch in verhallter Umgebung, sowie echtzeitfähigkeit.
Beide verwendeten Algorithmen können breitbandige Signale detektierten, da sie die Infomation über den Winkel durch Analyse der Redundanzen zwischen den Mikrofonsignalen erhalten und grundsätzlich unabhängig von der in den Signalen enthaltenen Frequenzen sind.
Die Winkelgenauigkeit ist lediglich abhängig von der Abtastrate und kann im Hinblick auf die Eigenschaften des GJBF als ausreichend eingestuft werden. Da dieser alle Störgeräusche außerhalb von $\pm 20^\circ$ weitesgehend ausblendet, können mit einer Winkelauflösung von etwa 8° gute Ergebnisse erzielt werden.
Es hat sich gezeigt, dass die MCCC unabhängig vom Aufbau des Mikrofonarrays sehr anfällig in der verhallten Umgebung ist. Im Vergleich mit den Ergebnissen im schalldichten Raum werden Quellen am Rande des Winkelbereichs des ULA mit einem teilweise doppelt so hohem Fehler erkannt. Wird ein Sprachsignal mit einem UCYA detektiert, so zeigt der Winkelverlauf in schalldichter Umgebung fast keinerlei Ausreißer, die innerhalb des Toleranzbereichs liegen. Dagegen hat der Algorithmus in verhallter Umgebung sehr mit den Reflektionen zu kämpfen. Die Tatsache, dass trotzdem gute Ergebnisse erlangt wurden, ist darauf zurückzuführen, dass nicht die D-Typ Variante des GJBF benutzt wurde, welche die Störung mit 40dB abdämpft.
Zuletzt wurde Echtzeitfähigkeit gefordert.

Rechtzeitigkeit Indem die Länge des EDMA so weit vergrößert wurde, dass in den ersten Tests kein Knacken in den Signalen zu hören war, konnte bewiesen werden, dass die Daten mindestens so schnell verarbeitet werden, wie der EDMA neue Daten aufnimmt.

Gleichzeitigkeit Die parallele Aufnahme und Verarbeitung der Daten wird durch die Nutzung des EDMA gewährleistet. Dieser nimmt die Signaldaten am ADU kontinuerlich auf und füllt seine Blöcke, die zur Verarbeitung freigegeben werden, sobald sie voll sind.

Kontinuität Ein kontinuierlicher Datenfluss wird durch die Verwendung des EDMA suggeriert. Da die Ausgangsdaten mit der gleichen Frequenz ausgegeben werden wie der EDMA die Eingangsdaten aufnimmt, kommt ein kontinuierlicher Datenfluss zustande.

Zuverlässigkeit Durch die Wahl einer geeigneten VDL-Länge wird sichergestellt, dass nicht zu viele Daten bei der Umlenkung der Lesezeiger in der VDL überschrieben oder wiederholt werden. Die Einhaltung der *Rechtzeitigkeit* gewährleistet zudem, dass der EDMA keine ungenutzten Daten überschreibt.

Alle Forderungen wurden eingehalten, womit eine Echtzeitfähigkeit des Systems garantiert ist.
Im Vergleich der beiden implementierten Algorithmen liefert die MCCC bessere Ergebnisse, was darauf zurückzuführen ist, dass die Korrelationsmatritzen über alle N Mikrofonsignale erstellt werden. Außerdem hat sich gezeigt, dass die SLP nicht geeignet ist für eine Echtzeitimplementierung, da die Matrixinversion zu viele Taktschritte benötigt.
In der Arbeit wurden Mikrofonarrays in linearer und zylindrischer Form verwendet. Das lineare Array zeigte eine Schwäche in der Hinsicht, dass am Rande des Winkelbereichs ein systematischer Fehler auftrat, der zu großen Winkelabweichungen führte.

9.2. Ausblick auf weitere Arbeiten

Der Algorithmus liefert in der vorliegenden Form sehr gute Ergebnisse. Es gibt jedoch Ansätze, die Leistungsfähigkeit noch zu verbessern.
Die Möglichkeiten auf dem verwendeten DSP werden durch den begrenzten Speicherplatz und die Geschwindigkeit des Prozessors stark eingeschränkt. Sollte es möglich sein, einen schnelleren DSP zu verwenden, so könnte die Abtastrate durch Interpolation auf 96kHz oder mehr erhöht werden, um die Winkel mit einer noch höheren Genauigkeit zu detektieren. Mit mehr Rechenzeit könnte auch eine Abtastrate von 8kHz und einer hohen Interpolationsrate verwendet werden, da der EDMA in diesem Fall auch mehr Zeit benötigt, um die Blöcke zu befüllen. Wäre mehr Speicherplatz vorhanden, so würde die Erhöhung der FFT-Länge dazu führen, dass längere Abschnitte eines Sprachsignals korreliert würden, was zu einer stabileren Winkelerkennung führte. Die Ausreißer bei der Detektion des Sprachsignals würden dann vermieden werden, wie in der Simulation gezeigt.
Im Hinblick auf die Geschwindigkeit der einzelnen Funktionen könnte die Optimierung des -o3-optimierten Assemblercodes des Compilers per Hand noch mehr Gewinn einbringen. Durch eine direkte Implementierung eines Polyphasenfilters in Assembler und geeignete Ausnutzung der Pipeline auf dem DSP könnte die Interpolation in weniger Taktschritten vonstatten gehen.
Sollte in einer späteren Anwendung ein ULA verwendet werden, so müsste eine Ausgleichsgerade eingebaut werden, um den systematischen Fehler - hervorgerufen durch die Winkelauflösung - zu minimieren.
Der GJBF, wie er zum Zeitpunkt dieser Arbeit bestand, war lediglich als C-Typ implementiert. Dieser dämpft die Störgeräusche nicht so stark ab, wie andere Variationen. Mit Verwendung eines anderen (besseren) Typs des GJBF, könnte die Störunterdrückung noch bessere Ergebnisse liefern. Dann müsste allerdings die Winkeldetektion auch in verhallter Umgebung noch stabiler gemacht werden, damit mehr Anteile des Nutzsignals in den Toleranzwinkelbereich geschoben werden könnten.

Literaturverzeichnis

[AC04] ANJANAIAH, Shaku ; COBB, Brad ; TEXAS INSTRUMENTS (Hrsg.): *TMS320C6000 McBSP Initialization, Application Report*. Texas Instruments, 2004

[BCH08] BENESTY, Jacob ; CHEN, Jingdong ; HUANG, Yiteng: *Microphone Array Signal Processing*. Springer-Verlag Berlin Heidelberg, 2008. – ISBN 978–3–540–78611–5

[Bus05] BUSSE, Stefan: *Lokalisierung von Schallquellen in geschlossenen Triebwerksprüfständen*. Technische Universität Berlin, 2005

[BW01] BRANDSTEIN, Michael ; WARD, Darren: *Microphone Arrays Signal Processing Techniques and Applications*. Springer-Verlag Berlin Heidelberg New-York, 2001. – ISBN 3–540–41953–5

[DB07] DMOCHOWSKI, Jacek ; BENESTY, Jacob: Direction of Arrival Estimation Using the Parameterized Spatial Correlation Matrix. (2007). `http://externe.emt.inrs.ca/users/benesty/papers/aslp_may2007.pdf`

[Eul06] EULER, Stephan: *Grundkurs Spracherkennung*. Friedr. Vieweg und Sohn Verlag, 2006. – ISBN 3–8348–0003–1

[Gör08] GÖRNE, Thomas: *Tontechnik*. Carl Hanser Verlag München, 2008. – ISBN 978–3–446–41591–1

[Har06] HARB, Oliver: *Automatische Programmcode-Generierung für effiziente Abtastratenumsetzer mit Polyphasen-Filtern auf einem Signalprozessor*. Hochschule für Angewandte Wissenschaften Hamburg - Fakultät für Elektrotechnik und Informatik, 2006

[Kap08] KAPS, Alexander M.: *Mehrkanalige Geräuschreduktion bei Sprachsignalen mittels Kalman-Filter*, Technische Universität Darmstadt, Diss., 2008

[Köl10] KÖLZER, Prof. Dr.-Ing. H. P.: *Digitale Bildverarbeitung und Mustererkennung*. März 2010. – Folien zur Vorlesung, Foliensatz: Einführung, Folie 34

[Kla10] KLAUKIN, Christoph: *Entwicklung und Realisierung eines Systems für die automatische Spracherkennung mit einem erweiterten TI DSP Modul zur Robotersteuerung*. Hochschule für Angewandte Wissenschaften Hamburg - Fakultät für Elektrotechnik und Informatik, 2010

[Kle10] KLEMENZ, Adolf: *PCM3003 Demo Program*. 2010

[KS10] KÖLZER, Prof. Dr.-Ing. H. P. ; SAUVAGERD, Prof. Dr. U.: *Digital Signal Processing*. November 2010. – Folien zur Vorlesung, Foliensatz: Efficient structures for Decimators and Interpolators, Folie 25

[NK05] NÖLKER, Norbert ; KLEMENZ, Adolf ; D.SIGNT (Hrsg.): *D.Module.PCM3003 Revision 1.0 Technical Data Sheet*. D.SignT, 2005. `http://www.dsignt.de/download/tddpcm3003.pdf`. – Zugriff: 13.06.2011

[NK10] NÖLKER, Norbert ; KLEMENZ, Adolf ; D.SIGNT (Hrsg.): *D.Module.C6713 Technical Data Sheet*. D.SignT, 2010. `http://www.dsignt.de/download/tdd6713.pdf`. – Zugriff: 22.07.2011

[Pan] PANASONIC: *Panasonic Sensors&Transducers*, `www.angliac.co.uk/product_search/datasheets/process.asp?datasheet_id=16700`. – Zugriff: 03.06.2011

[Pen70] PENNINGTON, Ralph: *Introductory Computer Methods and Numerical Analysis*. Macmillan: Collier-Macmillan, New York, 1970. – ISBN 0–0239–3830–7

[Pik10] PIKORA, Kolja: *DSP-basiertes Echtzeitsystem zur Sprecherlokalisierung mittels Mikrofonarray und Root-MUSIC*. Hochschule für Angewandte Wissenschaften Hamburg - Fakultät für Elektrotechnik und Informatik, 2010

[Pol99] POLAND, Syd: *TMS320C67xx Divide and Square Root Floating-Point Functions*. 1999

[Sau11] SAUVAGERD, Prof. Dr. U.: Forschungsbericht 2010/2011. 2011. – Forschungsbericht

[Sax09] SAXENA, Anshul K.: *Wideband Audio source localization usind Microphone Array and MUSIC Algorithm*. University of Applied Sciences Hamburg - Faculty of Engineering and Computer Scienece. `http://opus.haw-hamburg.de/volltexte/2009/752/pdf/Thesis08.pdf`. Version: 2009

[Var02] VARMA, Krishnaraj: *Time-Delay-Estimate Based Direction-of-Arrival Estimation for Speech in Reverberant Environments*. Virginia Polytechnic Institute and State University - Faculty of The Bradley Department of Electrical and Computer Engineering. `http://scholar.lib.vt.edu/theses/available/etd-10302002-220938/unrestricted/Thesis.pdf`. Version: 2002

[VHH98] VARY, Peter ; HESS, Wolfgang ; HEUTE, Ulrich: *Digitale Sprachsignalverarbeitung*. Teubner Verlag, 1998. – ISBN 3–5190–6165–1

[WB05] WÖRN, Heinz ; BRINKSCHULTE, Uwe: *Echtzeitsysteme*. Springer-Verlag Berlin Heidelberg, 2005. – ISBN 3–540–20588–8

Anhang

A. Inhalt der beigelegten CD

Die Programmcodes, Simulationsdateien und wav-Dateien sind auf einer CD hinterlegt, die bei Prof. Dr. Ulrich Sauvagerd und Prof. Dr.-Ing. Hans Peter Kölzer hinterlegt ist.

Inhaltlich ist der Datenträger wie folgt gegliedert:

\01_pdf enthält diese Arbeit im pdf-Format.

\02_Programmcode enthält die Quelltexte zur Programmierung auf dem DSP und ein CCS-Projekt.

\03_Matlab enthält eine Simulationsdatei und Hilfsdateien für die Simulation der Algorithmen. Außerdem einen Ordner mit aufgenommen Sprachsignalen für Simulationszwecke.

\04_Audiodateien enthält Audidateien, welche die Ergebnisse dokumentieren.

B. Empfohlenen Einstellungen in der `defines.h`

Die richtige Einstellung der Umgebungsparameter ist grundlegend für eine ordnungsgemäße Funktion des Programms. Aus diesem Grund wurde die Datei `defines.h` so gestaltet, dass lediglich der Algorithmus, der Aufbau des Arrays und die verwendeten Methoden ein- oder auskommentiert werden müssen. Alle weiteren Makros werden anschließend mit Präprozessoranweisungen definiert. Abbildung B.1 zeigt ein Blockschaltbild, das zur optimalen Einstellung der Makros verwendet werden kann.

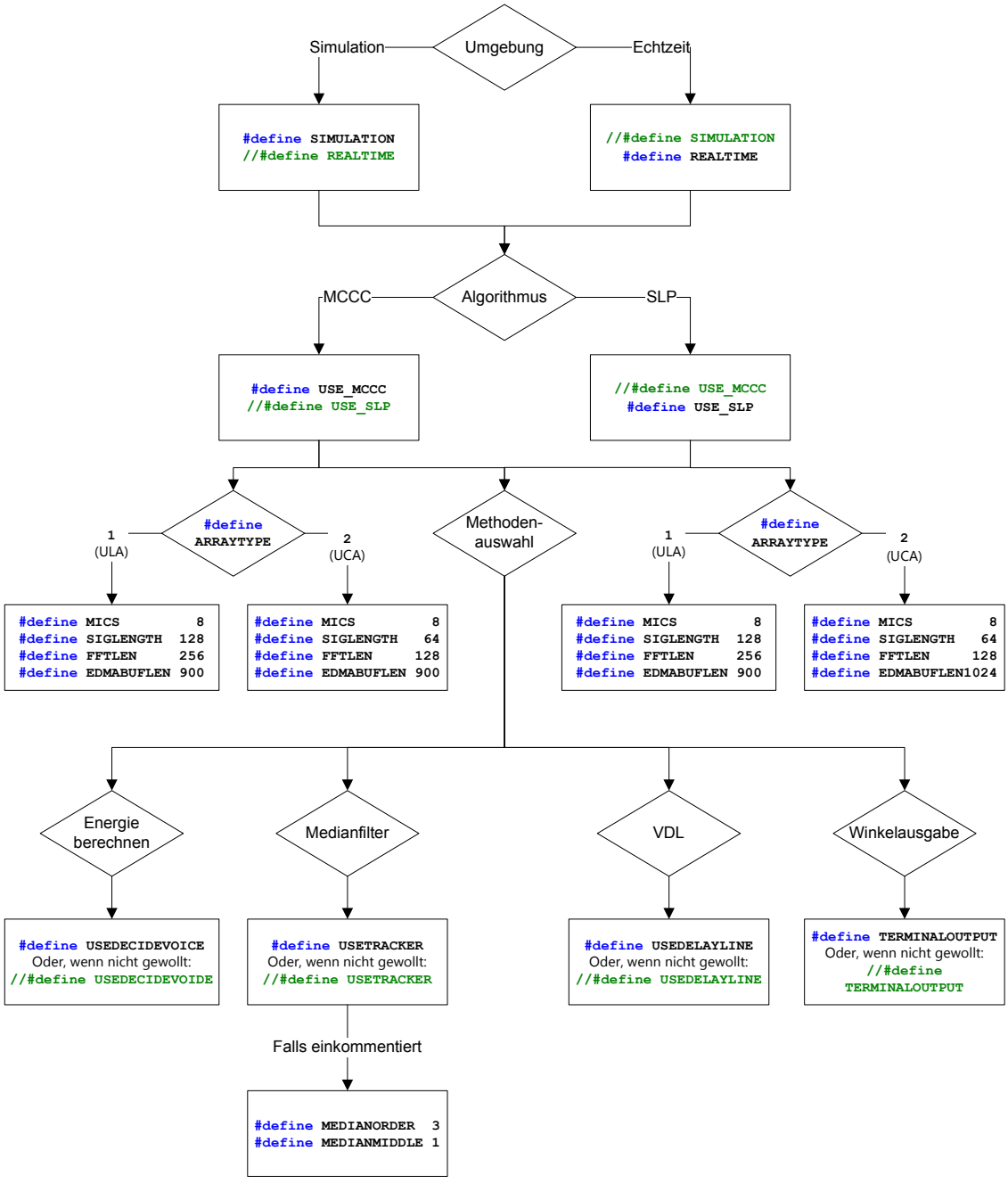


Abbildung B.1.: Blockschaltbild zur Einstellung der Makros in `defines.h`

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 30. August 2011	
Ort, Datum	Unterschrift