

# **Acoustic localisation of a speaker source in a reverberant environment**

**Mathias Buder**

mathias.buder@gmail.com

## **Zusammenfassung**

In der vorliegenden Arbeit erfolgt die Entwicklung eines Echtzeitsystems zur räumlichen Detektion einer Sprachquelle unter Verwendung eins kugelförmigen Mikrofonarrays. Die gesamte im Vorfeld durchgeführte Simulation wurde dabei mit der Entwicklungsumgebung MATLAB R realisiert. Die im zweiten Teil beschriebene Echtzeitimplementierung erfolgte auf Basis des D.Module.C6713 in der Programmiersprache C.

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>8</b>
1.1 Problemformulierung . . . . .	8
1.2 Zielsetzung . . . . .	10
<b>2 Konzeptionierung</b>	<b>12</b>
2.1 Grundlegende Annahmen und deren mathematische Zusam- menhänge . . . . .	12
2.1.1 System- und Signalmodell . . . . .	12
2.1.2 Sensormodell . . . . .	14
2.2 Schätzverfahren . . . . .	19
2.2.1 Kreuzkorrelation . . . . .	19
2.2.2 Mehrkanal-Kreuzkorrelationskoeffizient (MCCC) . . . . .	24
2.3 Eigenschaften von Sprachsignalen . . . . .	27
2.3.1 Stationarität . . . . .	27
2.3.2 Energie . . . . .	27
2.4 Mikrofonarray . . . . .	28
2.4.1 Untersuchung des kugelförmigen Mikrofonarrays . . . . .	31
2.4.2 Mikrofonarray Neudesign/Konstruktion . . . . .	32
2.5 Simulation . . . . .	40
2.5.1 Systemparameter . . . . .	42
2.5.2 Erstellung synthetischer Signale . . . . .	49
2.5.3 Kreuzkorrelation unter Verwendung der FFT . . . . .	51
2.5.4 Algorithmus zur Berechnung des MCCC . . . . .	51
2.5.5 Optimierungsverfahren . . . . .	58
<b>3 Realisierung</b>	<b>66</b>
3.1 Verwendete Hardware . . . . .	66
3.1.1 Schallwandler Front-End . . . . .	66
3.1.2 DSP Back-End . . . . .	68
3.1.3 Kommunikationsschnittstelle zwischen DSP und PCM3003 . . . . .	70

<i>INHALTSVERZEICHNIS</i>	2
3.1.4 EDMA-Verfahren . . . . .	<b>72</b>
3.1.5 Ping-Pong-Speicherverfahren . . . . .	<b>74</b>
3.2 Algorithmische Umsetzung . . . . .	<b>75</b>
3.2.1 Aufbau des C-Projekts . . . . .	<b>76</b>
3.2.2 Headerdateien des Projekts . . . . .	<b>77</b>
3.2.3 DSP Speicherbelegung . . . . .	<b>78</b>
3.2.4 Ablauf der Hauptroutine . . . . .	<b>79</b>
3.2.5 Methodenumsetzung . . . . .	<b>79</b>
3.2.6 Displaykomponente des Systems . . . . .	<b>87</b>
<b>4 Echtzeitversuch</b>	<b>95</b>
4.1 Untersuchung des Zeitverhaltens . . . . .	<b>95</b>
4.2 Versuchsaufbau . . . . .	<b>96</b>
4.3 Ergebnisse . . . . .	<b>98</b>
4.3.1 Funktionstest . . . . .	<b>98</b>
4.3.2 Verifikation der Winkelauflösung . . . . .	<b>99</b>
<b>5 Zusammenfassung</b>	<b>104</b>
5.1 Bewertung der Ergebnisse . . . . .	<b>105</b>
5.2 Ausblick . . . . .	<b>106</b>

# Tabellenverzeichnis

2.1	Laufzeitmessung an kugelförmigem Mikrofonarray bei den Winkeln $\phi = 0, \theta = 0$ . . . . .	32
2.2	Laufzeitmessung aus zwei unterschiedlichen Schalleinfallsrichtungen . . . . .	36
2.3	Toleranzbereich bei unterschiedlichen Abtastfrequenzen . . . . .	36
2.4	Winkelauflösung bei unterschiedlichen Abtastfrequenzen . . . . .	43
2.5	Abzusuchende Richtungen $S_{\phi,\theta}$ bei unterschiedlichen Abtastfrequenzen . . . . .	47
2.6	Maximale SDOA bei zwei gegenüberliegenden Mikrofonen . . . . .	47
2.7	Gewählte Systemparameter im Überblick . . . . .	49
3.1	McBSP Kanalzuordnung . . . . .	74
4.1	Profiling des Tracking-Algorithmus . . . . .	96

# Abbildungsverzeichnis

1.1	Richtungsbestimmung einer bewegten Sprachquelle im Raum.	9
1.2	Verwendete Hardware . . . . .	9
1.3	Blockschaltbild des Detektionssystems . . . . .	10
2.1	Illustration des Mikrofonarray Systems . . . . .	13
2.2	Sensormodell eines linearen Arrays . . . . .	15
2.3	Beschreibung der Signalquelle im Raum unter Verwendung von Azimuth- und Elevationswinkel. . . . .	16
2.4	Laufwegdifferenz als Projektionsvektor zwischen zwei Sensoren im Bezug auf den Schalleinfallsvektor $\mathbf{s}$ . . . . .	18
2.5	Mikrofonarray im akustischen Fernfeld . . . . .	19
2.6	Kreuzkorrelation zwischen $x(t)$ und $y(t)$ . Die Funktion $y(t)$ ist dabei eine skalierte und verzögerte Version von $x(t)$ . . . . .	21
2.7	Zeitverlauf eines Sprachsignals mit einer Länge von $T = 50ms$ sowie der Blockenergie bei einer Blocklänge von 256 Abtastwerten. . . . .	28
2.8	Zeitverlauf eines kurzen Sprachsignals sowie der Blockenergie bei einer Blocklänge von 256 Abtastwerten. . . . .	29
2.9	kugelförmiges Mikrofonarray . . . . .	31
2.10	Sensoranordnung auf der Kugeloberfläche . . . . .	31
2.11	Schallverhalten an schallharter Kugel ( $r = 50mm$ ) mit einer Schallwelle der Wellenlänge $\lambda = 114mm$ und einer Frequenz von $f = 3kHz$ . . . . .	33
2.12	Prinzip der Mikrofonkonstruktion . . . . .	34
2.13	Prototyp des würfelförmigen Mikrofonarrays . . . . .	35
2.14	Prinzip des Steckmechanismus . . . . .	37
2.15	Grundkomponenten des Mikrofonarrays . . . . .	38
2.16	Haltemechanismus der Mikrofonkapseln . . . . .	39
2.17	Drehung der Arraygeometrie zur exakten Positionierung der Mikrofonkapseln . . . . .	39
2.18	Vergleich zwischen Kugel- und Würfelgeometrie . . . . .	40

2.19 Variabel verstellbare Mikrofonabstände . . . . .	41
2.20 Fertigung der Einzelteile unter Verwendung einer CNC-Fräsmaschine . . . . .	41
2.21 Vergleich zwischen 3D Konstruktion und Fertigungsergebnis . . . . .	42
2.22 Algorithmus zur Validierung des abzubildenden Winkelraums . . . . .	45
2.23 Ergebnis der Winkelvalidierung für $e_{\phi rms}$ , $e_{theta rms}$ und $\bar{e}_{\phi,\theta rms}$ bei einer Abtastfrequenz von $f_a = 48kHz$ . . . . .	46
2.24 Schalleinfallsrichtung mit der maximal messbaren Laufzeitdifferenz . . . . .	48
2.25 Algorithmus zur zeitlichen Verschiebung von N Signalen . . . . .	50
2.26 Algorithmus zur Berechnung der nötigen Kreuzkorrelationsfunktionen $r_{y_i,y_j}$ sowie Varianzen $\sigma_{y_n}^2$ . . . . .	52
2.27 Algorithmus zur Berechnung des Mehrkanal-Kreuzkorrelationskoeffizienten . . . . .	53
2.28 Ergebnis des MCCC unter Verwendung eines synthetisch verzögerten Sprachsignals bei zwei unterschiedlichen Schalleinfallsrichtungen sowie SNR und einer Abtastfrequenz von $f_a = 44,1kHz$ . . . . .	54
2.29 Ergebnis der Kreuzkorrelations unter Verwendung eines synthetisch verzögerten Sinussignals bei einer Abtastfrequenz von $f_a = 48kHz$ . . . . .	55
2.30 Ergebnis des MCCC unter Verwendung eines synthetisch verzögerten periodischen Signals der Form $y(n) = \sin\left(2\pi\frac{f}{f_a}n\right)$ bei einer Abtastfrequenz von $f_a = 48kHz$ . . . . .	56
2.31 Wegstrecke einer Rauschquelle die sich diagonal durch den Raum bewegt . . . . .	58
2.32 Simulationsergebnis einer Rauschquelle bei einer Blockenergieschwelle von -10dB. . . . .	59
2.33 Wegstrecke einer männlichen Sprechquelle die sich Zig-Zag-förmig von Position 1 bis 9 um das Mikrofonarray bewegt. . . . .	60
2.34 Simulationsergebnis eines bewegten männlichen Sprechers (deutschsprachig) bei einer Blockenergieschwelle von -30dB. . . . .	61
2.35 Simulationsergebnis eines bewegten weiblichen Sprechers (englischsprachig) bei einer Blockenergieschwelle von -20dB. . . . .	62
2.36 Methode zur Optimierung der Suchgeschwindigkeit mit Hilfe einer variablen Schrittweiten. . . . .	63
2.37 Histogramm eines bewegten weiblichen Sprechers . . . . .	63
2.38 Vergleich zwischen Winkelverlauf mit und ohne Histogrammschätzung beim weiblichen Sprecher mit einer Ringspeicherlänge von 50 und einer Schwelle von 15% . . . . .	64

2.39 Vergleich zwischen Winkelverlauf mit und ohne Histogrammschätzung beim männlichen Sprecher mit einer Ringspeicherlänge von 50 und einer Schwelle von 15% . . . . .	65
3.1 Blockdiagramm der verwendeten Hardware . . . . .	67
3.2 Stromlaufplan des Mikrofonvorverstärkers für einen Kanal . . . . .	68
3.3 DSP-Flachbaugruppe D.Module.C6713 und A/D-Wandler D.Module.PCM3003 . . . . .	69
3.4 Blockdiagramm des D.SignT Moduls D.Module.PCM3003 (vgl. [3, S. 1]) . . . . .	69
3.5 Blockdiagramm des D.SignT DSP D.Module.C6713 (vgl. [4, S. 1]) . . . . .	71
3.6 Blockdiagramm der TMS320C6713b CPU (vgl. [17, S. 13]) . . . . .	72
3.7 McBSP Fubktionsblockdiagramm (vgl. [16, S. 2]) . . . . .	73
3.8 Reihenfolge des Kanalmultilexing durch den McBSP . . . . .	73
3.9 Interrupthandling zwischen EDMA-Controller und der Hauptroutine . . . . .	75
3.10 Ping-Pong-Verfahren der Hauptroutine (vgl. [12, S. 35]) . . . . .	76
3.11 Ablaufdiagramm der Hauptroutine main() . . . . .	80
3.12 Ablaufdiagramm der Kreuzkorrelationsroutine . . . . .	82
3.13 Zusammensetzung des Kommunikationsprotokolls zur Übertragung der Winkel unter Verwendung der UART-Schnittstelle. . . . .	88
3.14 Graphische Benutzeroberfläche zur anschaulichen Darstellung der Winkelverläufe . . . . .	89
3.15 Model-View-Controller Konzept der graphischen Benutzeroberfläche. Die durchgezogenen Linien symbolisieren eine direkte Verbindung. Linien in gestrichelter Darstellung bedeuten die Kommunikation über einen indirekten Weg mit Hilfe des Entwurfsmusters „Beobachter“. . . . .	90
3.16 Darstellung des Zustandsautomaten der Klasse SerialValueHandler. . . . .	92
3.17 Schema zur Dekodierung eines Zeichenketten-Frame . . . . .	93
4.1 Untersuchung des Zeitverhaltens mit einer EDMA-Dauer von $T_{EDMA} = 10,7\text{ms}$ . . . . .	97
4.2 Versuchsaufbau der Echtzeitmessung . . . . .	98
4.3 Einstellung der theoretischen Winkel am Mikrofonarray . . . . .	99
4.4 Messung der Schalleinfallsrichtung aus unterschiedlichen Winkeln in einem hallenden Raum. . . . .	100
4.5 Verifikation der Winkelauflösung im Bereich $0 \leq \theta \leq 49$ . . . . .	102
4.6 Verifikation der Winkelauflösung im Bereich $56 \leq \theta \leq 92$ . . . . .	103

# Quelltextverzeichnis

3.1	Aufbau des C-Pojekts . . . . .	77
3.2	Einlesen eines Datenblocks vom ADC inkl. Zero-Padding . . . . .	81
3.3	Erstellung der Twiddlefaktoren innhalb einer Schleife in der <code>init()</code> -Funktion. . . . .	83
3.4	Berechnung der Kreuzkorrelationsfunktionen . . . . .	84
3.5	Histogrammerstellung für den Raumwinkel $\phi$ . . . . .	86
3.6	Konvertierung vom Datentyp <code>float</code> in Typ <code>short</code> mit 3 Nachkommastellen. . . . .	87
3.7	Java-Funktion zum Export von Ergebniswerten in im MAT- Format . . . . .	93

# Kapitel 1

## Einführung

### 1.1 Problemformulierung

Die vorliegende Arbeit beschäftigt sich mit dem Design, der Simulation sowie der Implementierung eines digitalen Echtzeitsystems zur Lokalisierung einer bewegten Sprachquelle innerhalb eines geschlossenen Raumes. Ziel ist die Erstellung eines Algorithmus, der auf Basis geeigneter Schätzverfahren die Richtung (Horizontal- und Vertikalwinkel) dieser Signalquelle, bezogen auf ein Referenzkoordinatensystem, ermittelt (Abb. 1.1). Als zugrunde liegendem theoretischen Ansatz zur Bestimmung der Schalleinfallsrichtung (DOA<sup>1</sup>) eines Quellsignals werden hier die Laufzeitunterschiede (TDOA<sup>2</sup>) zwischen den einzelnen Mikrofonen sowie die bekannte Geometrie der Mikrofonanordnung<sup>3</sup> ausgenutzt. Eine graphische Darstellung soll letztlich mittels serieller Übertragung der geschätzten Ergebnisse an ein Anzeigegerät realisiert werden.

Zur Lösung dieser Aufgabenstellung stehen ein kugelförmiges Mikrofonarray mit acht Kondensatormikrofonen (Abb. 1.2a) inkl. Mikrofonvorverstärker, ein leistungsstarker digitaler Signalprozessor (Abb. 1.2b) der Firma Texas Instruments® sowie ein Desktopcomputer zur Verfügung.

Abb. 1.3 zeigt ein Prinzip-Blockschaltbild des Systems. Dieses lässt sich wie dargestellt in einen analogen -und digitalen Systemabschnitt klassifizieren. Im Rahmen dieser Arbeit soll die Signalverarbeitung im digitalen Systemabschnitt im Vordergrund stehen, da bereits alle Komponenten des Analogabschnitts funktionsfähig zur Verfügung gestellt wurden.

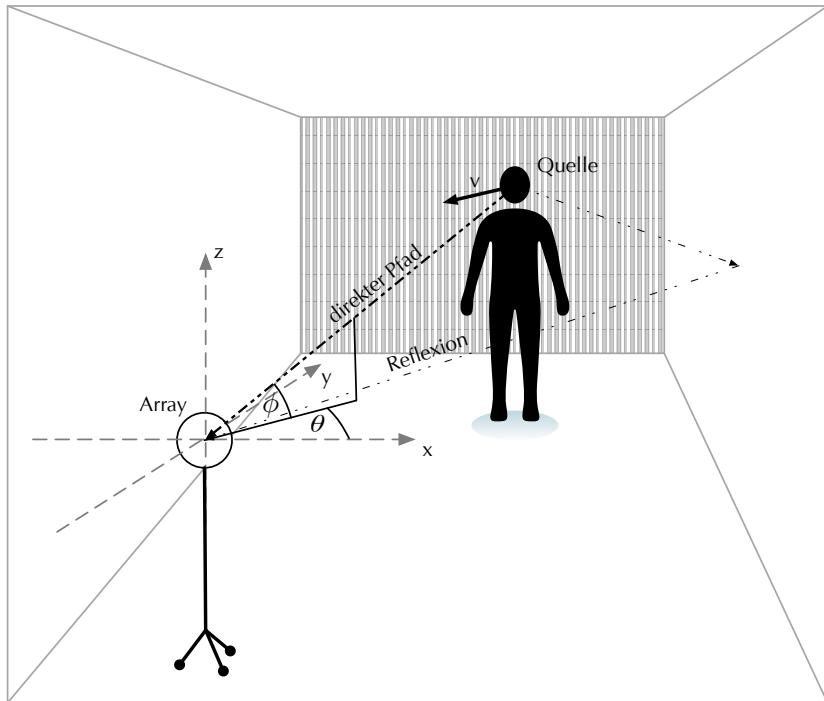
Die Ergebnisse dieser Arbeit wurden auf Basis von Pikora [12], Müller [11]

---

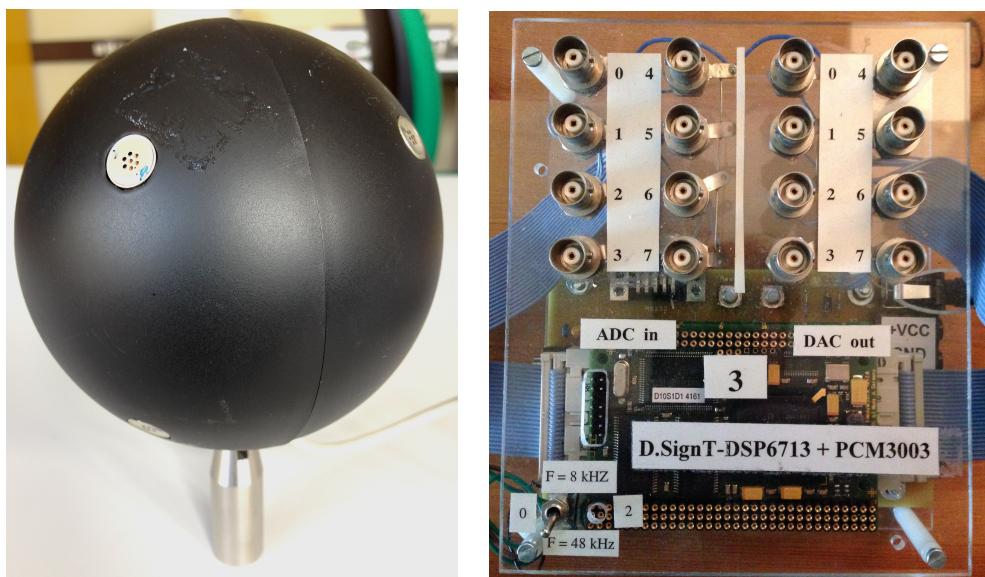
<sup>1</sup>Direction-Of-Arrival

<sup>2</sup>Time-Difference-Of-Arrival

<sup>3</sup>Mikrofonarray



**Abb. 1.1:** Richtungsbestimmung einer bewegten Sprachquelle im Raum.

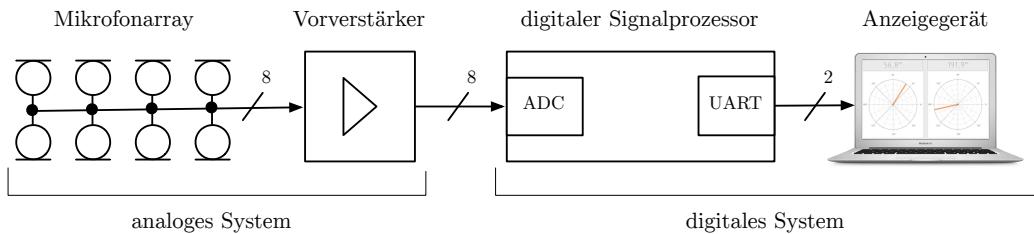


(a) kugelförmiges Mikrofonarray

(b) Digitaler Signalprozessor

**Abb. 1.2:** Verwendete Hardware

sowie des in Benesty u. a. [1, S. 196ff] und Dmochowski u. a. [2] beschriebenen Verfahrens des Mehrkanal-Kreuzkorrelationskoeffizienten (MCCC<sup>4</sup>) erzielt.



**Abb. 1.3:** Blockschaltbild des Detektionssystems

## 1.2 Zielsetzung

Auf Basis der unter Abschnitt 1.1 genannten Problemstellung lassen sich folgende vom System zu erfüllende Ziele definieren:

1. Fähigkeit zur Verarbeitung breitbandiger Signale wie Sprache
2. Fähigkeit zur räumlichen Richtungsdetektion der Quelle mit ausreichend hoher Genauigkeit (Horizontal- sowie Vertikalwinkel)
3. Fähigkeit zum Einsatz in Umgebungen mit hohem Rauschanteil und/oder Reflexionsfaktor
4. Fähigkeit zur Datenverarbeitung in Echtzeit

Der Begriff „Echtzeit“ lässt sich dabei wie folgt definieren:

**Def.:** Ein Programm rechnet in Echtzeit, wenn es Eingabedaten so schnell verarbeitet, wie der Vorgang, der sie erzeugt. [10, S. 34]

Im Hinblick auf eine spätere Implementierung mit Verwendung eines digitalen Signalprozessors lassen sich aus den oben genannten Zielen folgende Forderungen spezifizieren, um eine ordnungsgemäße Funktion des Gesamtsystems zu gewährleisten:

1. Verarbeitung von Sprachsignalen bis zu einer oberen Grenzfrequenz von  $f_{max} = 3kHz$

---

<sup>4</sup>Multichannel Cross-Correlation Coefficient

2. Robustes Verfolgen einer Sprachquelle innerhalb eines definierten Winkelraums:
  - Azimuth:  $0 \leq \theta < 360$
  - Elevation:  $-90 < \phi < 90$
3. Die Dauer der Datenverarbeitung muss gleich oder kleiner der Datenaufzeichnungsdauer sein. Demzufolge muss die Bedingung  $T_{Processing} \leq T_{Frame}$  dauerhaft erfüllt sein.
4. Der Prozess von Datensammlung und Datenverarbeitung muss zwingend nebenläufig stattfinden.
5. Zur Sicherstellung der Ergebniskontinuität muss ein kontinuierlicher Datenstrom zwischen Ein- und Ausgabeschnittstelle gewährleistet sein.

# Kapitel 2

## Konzeptionierung

Die grundlegenden Prinzipien und die daraus resultierenden Verfahren der Signalverarbeitung unter Verwendung von Mikrofonarrys sind ein essentieller Bestandteil dieser Arbeit. Es ist somit zwingend notwendig, die damit verbundenen mathematischen Zusammenhänge sowie die daraus abgeleiteten Methoden zu verstehen und gezielt einsetzen zu können.

Im folgenden Kapitel werden zunächst die elementaren Größen und Modelle der Array Signalverarbeitung eingeführt. Bezogen auf diese Grundlagen erfolgt anschließend der Entwurf eines mathematischen Modells zur Lösung der unter Abschnitt 1.1 beschriebenen Problemstellung.

### 2.1 Grundlegende Annahmen und deren mathematische Zusammenhänge

Im folgenden werden die System- sowie Signaleigenschaften des hier vorliegenden Systems untersucht. Die zu verarbeitenden Signale werden im weiteren Verlauf als Quellensignal und das von den Mikrofonen gelieferte als Sensorsignal bezeichnet.

#### 2.1.1 System- und Signalmodell

Abb. 2.1a illustriert den grundlegenden Aufbau eines Mikrofonarray Systems. Dieses besteht aus einem Quellensignal  $s(k)$  sowie aus  $N + 1$  Sensorsignalen  $y_n(k)$  die jeweils über eine der  $N + 1$  Kanalimpulsantworten  $g_n$  mit dem Quellensignal verknüpft sind. Solch eine Signalkonstellation wird als SIMO<sup>1</sup>-

---

<sup>1</sup>single-input multiple-output

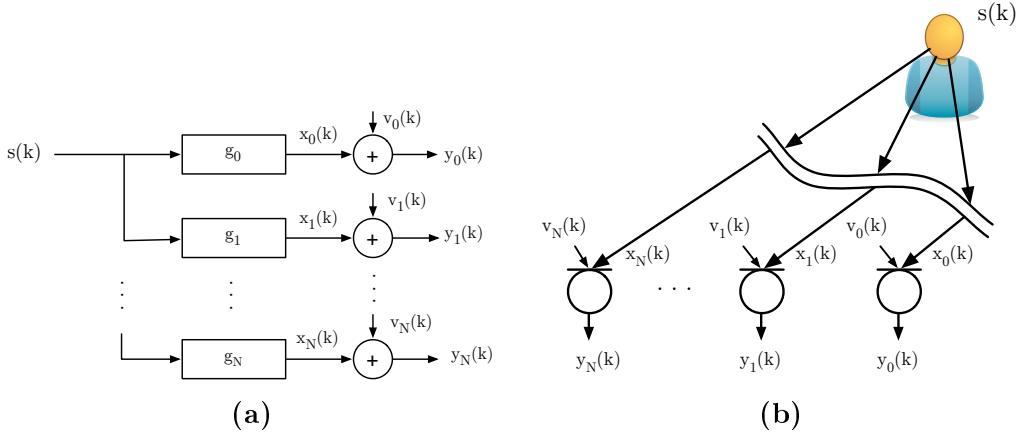


Abb. 2.1: Illustration des Mikrofonarray Systems

Modell [1, S. 141] bezeichnet und kann durch folgende Gleichung beschrieben werden

$$y_n(k) = g_n * s(k) + v_n(k) \quad (2.1)$$

wobei  $v_n(k)$  das dem Signal additiv überlagerte Umgebungsrauschen darstellt.

Bei genauerer Untersuchung von Abb. 2.1b können über die vom System verursachte Transformation des Quellsignals  $s(k)$  in das Sensorsignal  $y_n(k)$ , unter Berücksichtigung von akustischen Freifeldbedingungen<sup>2</sup>, folgende Aussagen getroffen werden:

- $s(k)$  wird durch die Entfernung von der Signalquelle zu den Sensoren um den Faktor  $\alpha_n$  gedämpft/skaliert. Es gilt dabei:  $0 \leq \alpha_n \leq 1$
- $s(k)$  wird durch die Entfernung zwischen der Signalquelle und den Mikrofonarray um eine Schallausbreitungszeit  $T_S$  verzögert
- $s(k)$  wird durch die Entfernung  $d$  zwischen den einzelnen Sensoren um eine relative Zeit  $\tau_{0n}$  verzögert.  $\tau_{0n}$  bezieht sich dabei auf das Referenzmikrofon  $M_0$ .

Unter diesen Annahmen lässt sich  $y_n(k)$  wie folgt ausdrücken

$$y_n(k) = \alpha_n s(k - T_S - \tau_{n1}). \quad (2.2)$$

---

<sup>2</sup>ohne jegliche Reflexionen, nur der direkte Ausbeitungspfad wird betrachtet

Aus dieser Erkenntnis, dem in Gleichung 2.1 gezeigten Zusammenhang und der Vernachlässigung des additiven Rauschens  $v(t)$  können die Kanalimpulsantworten  $g_n$  wie folgt gefunden werden:

$$\alpha_n s(k - T_S - \tau_{n0}) = \underbrace{\alpha_n \delta(k - T_S - \tau_{n0})}_{g_n} * s(k) \quad (2.3)$$

Zur Ermittlung des Systemverhaltes ist es zwingend notwendig genaue Kenntnis über den Frequenz- und Phasengang zu erlangen. Diesen erhält man durch Fouriertransformation der  $N + 1$  Impulsantworten  $g_n$ .

$$g_n = \alpha_n \delta(k - T_S - \tau_{n1}) \quad \circ \bullet \quad \underline{G}_n(e^{j\omega}) = \alpha_n e^{-j\omega(T_s + \tau_{n0})} \quad \text{für } n = 1 \dots N \quad (2.4)$$

Der gewünschte Amplituden- und Phasengang lässt sich nun direkt aus dem komplexen Frequenzgang  $\underline{G}_n(e^{j\omega})$  ablesen.

$$\begin{aligned} \text{Amplitudengang: } |\underline{G}_n(e^{j\omega})| &= \alpha_n \\ \text{Phasengang: } \varphi_n(f) &= -2\pi f(T_s + \tau_{n0}) \end{aligned} \quad (2.5)$$

Aus dem Frequenzgang leitet sich ein konstanter Amplitudengang  $\alpha_n$  (Dämpfungsverzerrung) sowie ein frequenzproportionaler Phasengang (Phasenverzerrung)  $-2\pi f(T_s + \tau_{n0}) \sim f$  ab. Diese Systemeigenschaft lässt sich direkt einem idealen verzerrungsfreien LTI<sup>3</sup>-System zuordnen [7, S. 63ff]. Das hier zu implementierende reale System muss das in Gleichung 2.5 dargestellte Verhalten innerhalb der Signalbandbreite möglichst gut approximieren.

## 2.1.2 Sensormodell

### Lineares zweidimensionales Sensorarray

Abb. 2.2 zeigt den Aufbau eines linearen äquidistanten Mikrofonarrays (ULA<sup>4</sup>) mit einer Anzahl von  $N + 1$  Sensoren und dem Sensorabstand  $d$ . Bei einer solchen Anordnung legen die Schallwellen, abhängig vom Schalleinfallswinkel  $\theta$  unterschiedlich lange Wege von der Signalquelle zu den einzelnen Sensoren zurück. Unter Berücksichtigung einer ebenen Wellenfront sowie einer bekannten Arraygeometrie lässt sich mit Hilfe der Laufzeitdifferenz  $\tau_{ULA}$  und der Schallgeschwindigkeit<sup>5</sup>  $c \approx 343 \frac{m}{s}$  folgender Zusammenhang definieren

---

<sup>3</sup>Linear Time Invariant

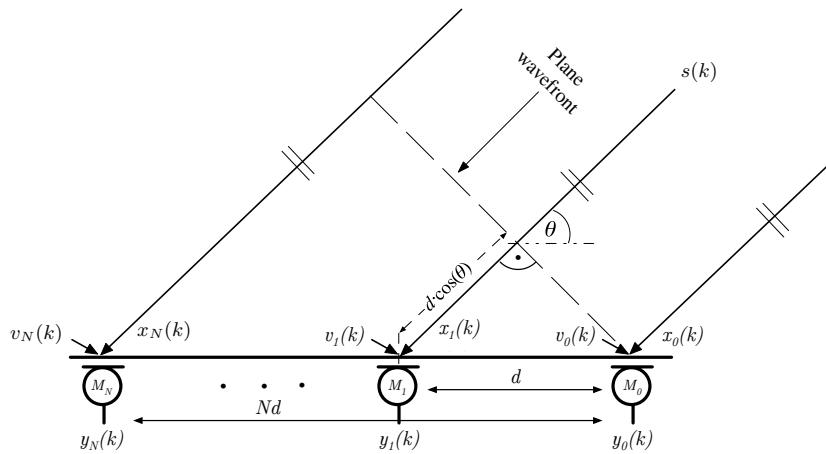
<sup>4</sup>Uniform Linear Array

<sup>5</sup>Approximierte Geschwindigkeit bei einer Raumtemperatur von 20°C

$$\tau_{ULA} = \frac{d}{c} \cdot \cos(\theta) \quad (2.6)$$

Für den hier beschriebenen Fall kann die Laufzeitdifferenz zwischen Referenzsensor  $M_0$  und Sensor  $M_N$  durch die lineare Funktion  $f_{0n,ULA}(\tau)$  ausgedrückt werden

$$f_{0n,ULA}(\tau) = n \cdot \tau_{ULA} \quad \text{mit} \quad n = 1, 2, \dots, N \quad (2.7)$$



**Abb. 2.2:** Sensormodell eines linearen Arrays

### Dreidimensionales Sensorarray

Zur räumlichen Detektion einer Schallquelle ist es notwendig eine Sensorgeometrie zu finden, in der die Richtungsinformation durch den Azimuthwinkel  $\theta$  sowie den Elevationswinkel  $\phi$  ausgedrückt werden kann. Eine solches Array lässt sich mit Hilfe einer Matrix  $\mathbf{M}$  repräsentieren, wobei jede Spalte die Position eines der  $N + 1$  Sensoren in einem dreidimensionalen euklidschen Raum beschreibt.

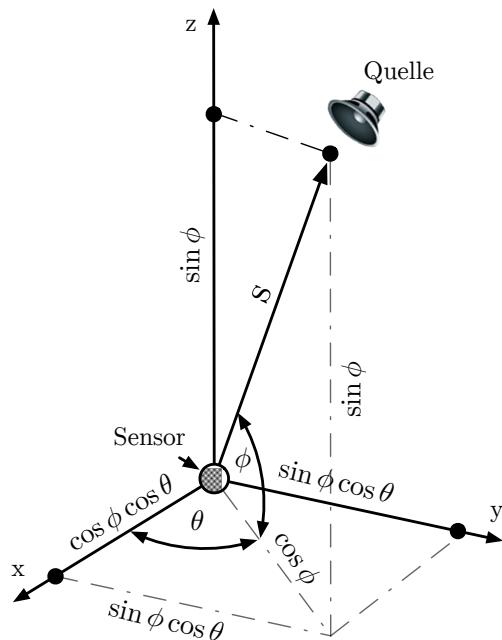
$$\mathbf{M}_{3 \times N+1} = \begin{bmatrix} m_{x_0} & m_{x_1} & \dots & m_{x_N} \\ m_{y_0} & m_{y_1} & \dots & m_{y_N} \\ m_{z_0} & m_{z_1} & \dots & m_{z_N} \end{bmatrix} \quad (2.8)$$

Abb. 2.3 zeigt den Referenzsensor im Zentrum dieses Raumes mit einem aus einer beliebigen Richtung eintreffenden Signal. Dieses Signal kann über den Einheitsvektor  $\mathbf{s}$ , welcher senkrecht zur von der Quelle Emittierten ebenen

Wellenfront steht, beschrieben werden. Im Hinblick auf eine spätere Extraktion der oben genannten Raumwinkel wird  $\mathbf{s}$  in Kugelkoordinaten ausgedrückt.

$$\mathbf{s}_{3 \times 1} = \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix} = \begin{bmatrix} \cos(\phi) \cos(\theta) \\ \cos(\phi) \sin(\theta) \\ \sin(\phi) \end{bmatrix} \quad (2.9)$$

Der Azimuthwinkel kann dabei die Werte  $0 \leq \theta < 360$  und der Elevationswinkel  $-90 \leq \phi \leq 90$  annehmen.



**Abb. 2.3:** Beschreibung der Signalquelle im Raum unter Verwendung von Azimuth- und Elevationswinkel.

Als nächster Designschritt folgt der Zusammenhang zwischen dem Schalleinfallswinkel und der Laufzeitdifferenz der einzelnen Sensorpaare. Abb. 2.4 illustriert ein Sensorpaar, das durch die Spaltenvektoren  $\mathbf{m}_0$  und  $\mathbf{m}_1$  repräsentiert wird. Der Verbindungsvektor  $\mathbf{m}_{01}$  zwischen den Sensoren berechnet sich mit  $\mathbf{m}_1 - \mathbf{m}_0$ . Zu beachten ist, dass  $\mathbf{s}$  nun in Richtung der Sensoren zeigt und somit gilt  $\mathbf{s} = -[\cos(\phi) \cos(\theta) \cos(\phi) \sin(\theta) \sin(\phi)]^T$ . Die zusätzliche Distanz, die der Schall zurücklegen muss, um von Sensor  $M_0$  zu  $M_1$  zu gelangen wird als Laufwegunterschied  $s_{01}$  bezeichnet. Wie in Abb. 2.4 dargestellt ergibt sich  $s_{01}$  durch Projektion von  $\mathbf{m}_{01}$  mit Bezug auf den Schalleinfallsvektor  $\mathbf{s}$ .  $s_{01}$  kann somit durch das Skalarprodukt

$$s_{01} = (\mathbf{m}_1 - \mathbf{m}_0)^T \mathbf{s} = \mathbf{m}_{01}^T \mathbf{s} \quad (2.10)$$

ermittelt werden. Folglich bildet sich die Laufzeitdifferenz  $\tau_{01}$  durch

$$\tau_{01} = [\mathbf{m}_{01}^T \mathbf{s}] \cdot c^{-1} \quad (2.11)$$

Zur Bestimmung der Laufzeitdifferenzen zwischen allen Sensoren muss zunächst eine Matrix  $\mathbf{A}$  definiert werden, in der die Verbindungsvektoren  $\mathbf{m}_{0n}$  aller  $N$  Sensorpaarkombinationen definiert sind.

$$\mathbf{A}_{3 \times N} = [\mathbf{m}_{01} \ \mathbf{m}_{02} \ \dots \ \mathbf{m}_{0N}] \quad (2.12)$$

Dabei gilt für  $\mathbf{m}_{3 \times 1}$  der Ausdruck

$$\mathbf{m}_{1N} = \begin{bmatrix} x_{0N} \\ y_{0N} \\ z_{0N} \end{bmatrix} = \begin{bmatrix} x_N - x_0 \\ y_N - y_0 \\ z_N - z_0 \end{bmatrix} \quad (2.13)$$

Die gesuchten Laufzeitdifferenzen können nun in einem Vektor  $\boldsymbol{\tau}_{N \times 1}$  beschrieben werden. Für ein statisches Sensorarray sind diese ausschließlich von der Schalleinfallsrichtung und damit von den Winkeln  $\phi$  und  $\theta$  abhängig.

$$\begin{aligned} \boldsymbol{\tau}(\phi, \theta)_{N \times 1} &= \left[ (\mathbf{A}^T)_{N \times 3} \cdot \mathbf{s}_{3 \times 1} \right]_{N \times 1} \cdot c^{-1} \\ &= f_n(\phi, \theta) \quad \text{mit } n = 1, 2, \dots, N \end{aligned} \quad (2.14)$$

Ein Sensorarray mit beliebiger Geometrie kann so analog zu Gleichung 2.7 durch die nicht-lineare Funktion  $f_n(\phi, \theta)$  beschrieben werden. Auf Grund einer nicht-quadratischen Matrix  $\mathbf{A}$  muss dessen Inverse als Pseudoinverse  $\mathbf{A}^+$  gebildet werden [13, S. 185]. Gleichung 2.14 kann so mit Bezug auf  $\mathbf{s}$  in

$$\hat{\mathbf{s}}(\phi, \theta)_{3 \times 1} = \left\{ \underbrace{\left[ \left( \mathbf{A} \cdot \mathbf{A}^T \right)^{-1} \cdot \mathbf{A} \right]_{3 \times N}}_{\mathbf{A}^+} \cdot \hat{\boldsymbol{\tau}}(\phi, \theta)_{N \times 1} \right\}_{3 \times 1} \cdot c \quad (2.15)$$

umgeformt werden.  $\hat{\mathbf{s}}(\phi, \theta)$  beschreibt dabei den geschätzten Normalvektor<sup>6</sup> zur Schalleinfallsrichtung und  $\hat{\boldsymbol{\tau}}$  die Schätzung der paarweisen Laufzeitdifferenz [20, S. 71].

---

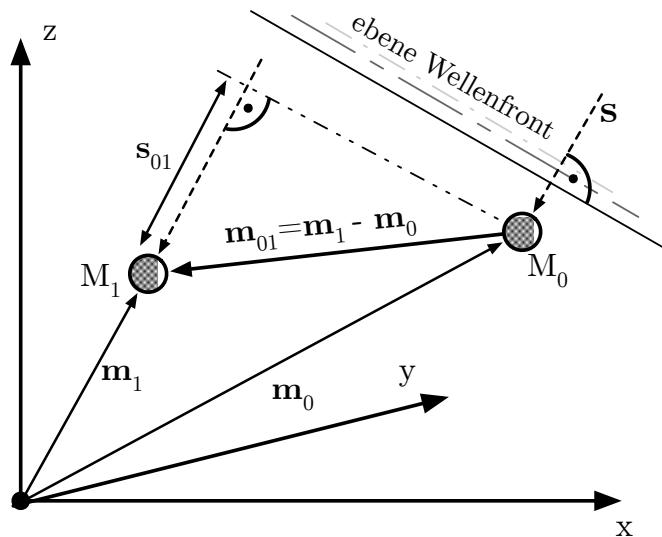
<sup>6</sup>  $\mathbf{s}$  steht senkrecht zur einlaufenden Wellenfront

### Räumliches Aliasing

Basierend auf den Ausführungen zum räumlichen Alias-Effekt in [12, 8 f] ist bei der Dimensionierung des Sensorarrays folgendes Kriterium zu berücksichtigen:

$$d_{min} \leq \frac{\lambda}{2} = \frac{c}{2f_{max}} \quad (2.16)$$

$d_{min}$  ist dabei die geringste Distanz zwischen zwei Sensoren und  $f_{max}$  die größte im Signal enthaltene Frequenz.

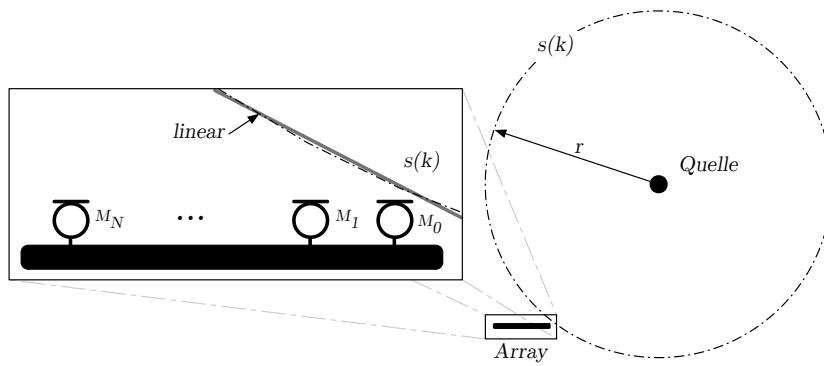


**Abb. 2.4:** Laufwegdifferenz als Projektionsvektor zwischen zwei Sensoren im Bezug auf den Schalleinfallsvektor  $s$

### Akustisches Fernfeld

Zur Sicherstellung einer ebenen Wellenfront muss sich die Signalquelle wie in Abb. 2.5 dargestellt, im akustischen Fernfeld befinden. Unter diesen Umständen darf die von der Quelle emittierte Kugelwelle  $s(k)$  auf einem kleinen Bereich linear approximiert werden. Für die Entfernung  $r$ , von Array bis zu dem Punkt, an dem die Fernfeldbedingung angenommen werden kann muss folgendes gelten[5, S. 36f]:

$$k \cdot r \gg 1 \quad \text{mit} \quad k = \frac{2\pi}{\lambda} = \frac{2\pi f}{c} \quad (2.17)$$



**Abb. 2.5:** Mikrofonarray im akustischen Fernfeld

## 2.2 Schätzverfahren

In Abschnitt 2.1.2 wurde gezeigt, dass die Laufzeitdifferenz  $\tau$  zwischen den einzelnen Sensorpaaren genutzt werden kann, um mit Hilfe geometrischer Zusammenhänge die Schalleinfallsrichtung zu bestimmen. Da  $\tau$  ein unbekannter Parameter ist, kann dieser mit Hilfe geeigneter Verfahren aus mindestens zwei Sensorsignalen geschätzt werden. Aufgrund der unter Abschnitt 2.1.1 abgeleiteten Systemeigenschaften wie der Verzögerung und Skalierung des Quellsignals durch die Sensorgeometrie kann hier die Kreuzkorrelation als Schätzverfahren angewandt werden.

### 2.2.1 Kreuzkorrelation

Die Kreuzkorrelation zweier kontinuierlicher Funktionen  $x(t)$  und  $y(t)$  über einem Intervall  $T$  ist definiert mit

$$r_{xy}(\tau) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T x(t) \cdot y(t + \tau) dt \quad (2.18)$$

Die Eigenschaften dieser Schätzmethode lassen sich im Bezug auf die hier vorliegenden Signale anhand eines anschaulichen Beispiels erläutern. Wie in Abb. 2.6 dargestellt sei  $x(t)$  ein Rechteckimpuls der Pulsbreite  $T_p$  und  $y(t) = \alpha \cdot x(t - \Delta t)$  eine um  $\alpha$  skalierte sowie um  $\Delta t$  verzögerte Version von  $x(t)$ . Als Ergebnis der Kreuzkorrelation ergibt sich ein Dreieckimpuls der doppelten Pulsbreite. Dieser ist um die Verzögerungszeit  $\Delta t$  zentriert und hat den Spitzenwert  $\alpha \cdot T_p$ . In Anbetracht einer realen Implementierung ist die in Gleichung 2.18 dargestellte Grenzwertbildung nicht möglich. Die

Korrelation kann so nur über einem endlichen Intervall  $T < \infty$  stattfinden was zu einer Schätzung  $\hat{\Delta t}$  der Laufzeitverzögerung  $\Delta t$  durch

$$\hat{\Delta t} = \arg \max_{\tau} [\hat{r}_{xy}(\tau)] \quad (2.19)$$

führt. Zu erkennen ist, dass diese Methode eine Funktion liefert, die eine Aussage über die Ähnlichkeit der korrelierten Signale macht. An der Stelle, an der sich beide Funktionen am meisten ähneln, besitzt die Kreuzkorrelationsfunktion  $\hat{r}_{xy}(\tau)$  ihr Maximum. Zu beachten ist, dass beide Signale frei von jeglichem Gleichanteil sind. Wird diese Bedingung verletzt, besitzt  $\hat{r}_{xy}(\tau)$  ihren Maximalwert stets bei  $\tau = 0$ .

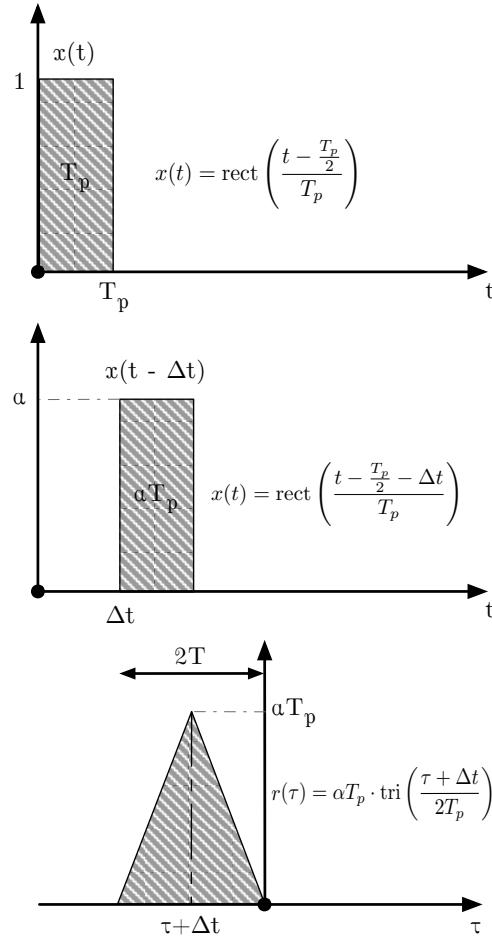
Da die hier zu verarbeitenden Signale in diskreter Form vorliegen, muss das Korrelationsintegral in eine Korrelationssumme der Fensterlänge  $K$  umgeformt werden. Unter der Bedingung  $N < \infty$  folgt die geschätzte Kreuzkorrelationsfolge  $\hat{r}_{xy}[k]$  der Länge  $2K - 1$  durch

$$\hat{r}_{xy}[k] = \frac{1}{K} \sum_{n=0}^{K-1} x[n] \cdot y[n+k] \quad \text{mit} \quad -(K-1) \leq k \leq K-1 \quad (2.20)$$

Zu berücksichtigen ist dabei allerdings, dass nur Abtastwerte aus dem Beobachtungsintervall  $0 \leq n \leq K-1$  zur Verfügung stehen. Wird beispielsweise nach Gleichung 2.20  $\hat{r}_{xy}[k]$  für den Verschiebezeitpunkt  $k = 5$  und einer Fensterlänge  $K = 10$  berechnet, wird auf den Wert  $x[14]$  der Musterfunktion zugegriffen. Dieser Abtastwert existiert jedoch nicht. Die Summationsgrenzen sind daher zu modifizieren [8, S. 321 ff]. Es folgt:

$$\hat{r}_{xy}[k] = \begin{cases} \frac{1}{K} \sum_{n=0}^{K-1-k} x[n] \cdot y[n+k] & \text{für } 0 \leq k \leq K-1 \\ \frac{1}{K} \sum_{n=-k}^{K-1} x[n] \cdot y[n+k] & \text{für } -(K-1) \leq k < 0 \\ 0 & \text{sonst} \end{cases} \quad (2.21)$$

Bei Betrachtung von Gleichung 2.20 im Hinblick auf den aufzuwendenden Rechenaufwand fällt auf, dass diverse Additions- sowie Multiplikationsoperationen durchzuführen sind. Zur Korrelation von zwei Datenfolgen der Länge  $K$  müssen  $(2K-1) \cdot (K-1)$  Additionen sowie  $(2K-1) \cdot K$  Multiplikationen ausgeführt werden. Der Rechenaufwand wächst somit quadratisch mit wachsender Fensterlänge  $K$ , wodurch sich diese Rechenmethode für eine zeitkritische Implementierung als ungeeignet erweist.



**Abb. 2.6:** Kreuzkorrelation zwischen  $x(t)$  und  $y(t)$ . Die Funktion  $y(t)$  ist dabei eine skalierte und verzögerte Version von  $x(t)$ .

### Schnelle Kreuzkorrelation

Im Hinblick auf eine spätere Echtzeitimplementierung ist eine effiziente Methode zur Berechnung der Kreuzkorrelation von großem Interesse. Es kann gezeigt werden, dass dies über eine Umformung der Korrelationssumme in eine Faltungssumme erreicht werden kann. Aufgrund des hier vorliegenden LTI-Systems kann dessen Ausgangsfolge  $g[n]$  durch die Faltung zwischen einer Eingangsfolge  $s[n]$  und der Systemimpulsantwortfolge  $h[n]$  definiert werden:

$$\begin{aligned} g[n] &= s[n] * h[n] \\ &= \sum_{k=-\infty}^{\infty} s[k] \cdot h[n-k] \end{aligned} \quad (2.22)$$

Ein Vergleich von Gleichung 2.20 und Gleichung 2.22 ergibt, dass lediglich die Summationsvariablen vertauscht sind. Als erster Umformungsschritt erfolgt so zunächst eine Anpassung der Indizes von  $y$

$$\hat{r}_{xy}[k] = \frac{1}{K} \sum_{n=0}^{K-1-k} x[n] \cdot y[k - (-n)] \quad \text{für } 0 \leq k \leq K-1 \quad (2.23)$$

Des weiteren wird  $x[n]$  in  $x[-(-n)]$  umgeformt, womit folgt

$$\hat{r}_{xy}[k] = \frac{1}{K} \sum_{n=0}^{K-1-k} x[-(-n)] \cdot y[k - (-n)] \quad \text{für } 0 \leq k \leq K-1 \quad (2.24)$$

Unter Nutzung von Gleichung 2.22 kann die Kreuzkorrelierte  $\hat{r}_{xy}[k]$  als Faltungsoperation ausgedrückt werden

$$\hat{r}_{xy}[k] = x[-n] * y[n] \quad (2.25)$$

Zur Rechenzeitoptimierung kann nun unter Berücksichtigung folgender Korrespondenz

$$\hat{r}_{xy}[k] \circ \bullet \underline{R}_{xy}[l] = \underline{X}[-l] \cdot \underline{Y}[l] \quad (2.26)$$

$$= \underline{X}[l]^* \cdot \underline{Y}[l] \quad (2.27)$$

auf die FFT<sup>7</sup>, einer schnellen Variante der diskreten Fourier Transformation (DFT<sup>8</sup>) und deren Inversen (IDFT<sup>9</sup>) zurückgegriffen werden.  $\underline{R}_{xy}[l]$  beschreibt dabei das Kreuzleistungsdichtespektrum. Eine effiziente Variante der geschätzten Kreuzkorrelationsfunktion  $\hat{r}_{xy}[k]$  kann so wie folgt als schnelle Kreuzkorrelation errechnet werden:

$$\hat{r}_{xy}[k] = \text{IDFT} \{ (\text{DFT}\{x[n]\})^* \cdot \text{DFT}\{y[n]\} \} \quad (2.28)$$

Zur Anwendung dieses Verfahrens müssen zwei Bedingungen in Bezug auf die Eigenschaften von DFT bzw. FFT erfüllt sein:

---

<sup>7</sup>Fast Fourier-Transformation

<sup>8</sup>Diskrete Fourier-Transformation

<sup>9</sup>Inverse Diskrete Fourier-Transformation

1. Die DFT-Länge  $K$  muss auf Grund der hier verwendeten Radix-2-FFT einer Zweierpotenz mit  $K = 2^p$  und  $p \in \mathbb{N}^+$  entsprechen. Diese Form der FFT zerlegt die herkömmliche DFT sukzessiv in zwei DFT der halben Länge, bis schließlich eine Transformationslänge zwei erreicht wird [21, S. 222 ff].
2. Um einen Überfaltungsfehler zu vermeiden, müssen den Datenfolgen  $x[n]$  und  $y[n]$  der Länge  $K$  vor der Transformation in den Frequenzbereich mindestens  $K - 1$  Nullen angehängt werden (Zeropadding) [21, S. 215 ff]. Auf Grund der Bedingung aus Punkt 1 wird die Datenfolge um eine weitere Null auf die Länge  $2K$  verlängert. Der zusätzliche Ergebniswert befindet sich an der ersten Stelle  $\hat{r}_{xy}[0] = 0$  und kann ignoriert werden.
3. Nach der Rücktransformation in den Zeitbereich muss die gesamte Funktion  $\hat{r}_{xy}[k]$  um dessen halbe Länge auf der Zeitachse nach rechts verschoben werden. Die Zeitkomponente  $k = 0$  befindet sich dann im Zentrum des Koordinatensystems, wodurch negative sowie positive Verzögerungen abgelesen werden können. Dies kann z.B. in MATLAB® mit der eingebetteten Funktion `fftshift()` erreicht werden.

### Erwartungstreue Schätzung

Wie bereits im vorangegangenen Abschnitt erwähnt, wird eine Korrelation über einen endlich langen Signalabschnitt berechnet. Da die Eingabedaten in einem kontinuierlichen Datenstrom vorliegen, muss diesem eine gewisse Menge an Werten entnommen und anschließend verarbeitet werden. Betrachtet man diese Datenentnahme im Hinblick auf eine Fensterfunktion, entspricht dies einer Multiplikation des Datenstroms mit einem Rechteckfenster wie bereits in Abb. 2.6 illustriert. Diese Fenstermethode weist folgende spektrale Eigenschaften auf (vgl. [8, S. 301]):

$$\hat{r}_{xy}(t) = s_1(t) \cdot \frac{1}{T} \operatorname{rect}\left(\frac{t}{T}\right) * s_2(t) \cdot \frac{1}{T} \operatorname{rect}\left(\frac{t}{T}\right) \quad (2.29)$$

$$= \mathcal{F} \left\{ s_1(t) \cdot \frac{1}{T} \operatorname{rect}\left(\frac{t}{T}\right) * s_2(t) \cdot \frac{1}{T} \operatorname{rect}\left(\frac{t}{T}\right) \right\} \quad (2.30)$$

$$= \underline{S_1}(f) * \operatorname{si}(\pi f T) \cdot \underline{S_2}(f) * \operatorname{si}(\pi f T) \quad (2.31)$$

$$= \underline{S_1}(f) \cdot \underline{S_2}(f) * \operatorname{si}^2(\pi f T) \quad (2.32)$$

$$= s_1(t) * s_2(t) \cdot \frac{1}{T} \operatorname{tri}\left(\frac{t}{T}\right) \quad (2.33)$$

Die Entnahme von Daten mit einem rechteckförmigen Fenster bewirkt im Faltungs-/Korrelationsergebnis eine Gewichtung durch ein dreieckförmiges Fenster (Barlett-Fenster). Man beschreibt ein solches Vorgehen als nicht-erwartungstreue. Die Aussagekraft der Korrelationswerte sinkt linear mit steigender Entfernung zur Korrelogrammmitte. Um aus diesem Ergebnis eine erwartungstreue Schätzung zu erhalten, ist die gesamte Folge mit der Umkehrfunktion des Dreieckfensters zu gewichten. Wie sich später noch zeigen wird, ist diese Ausgleichsoperation bei der hier implementierten Anwendung nicht notwendig.

### 2.2.2 Mehrkanal-Kreuzkorrelationskoeffizient (MCCC)

Das unter Abschnitt 2.2.1 eingeführte Schätzverfahren ist lediglich in der Lage, Aussagen über die Korrelation zweier Signale zu geben. Zur Verarbeitung von  $N + 1$  Signalen bedarf es einer Erweiterung dieser Methode [1, S. 196ff], [2].

Die grundsätzliche Idee der Verwendung von Arrays mit mehr als zwei Sensoren ist das hohe Maß an redundanter Information. Diese kann extrahiert und zur Erhöhung der Systemrobustheit in realer akustischer Umgebung (z.B. stark hallender Raum, additiv überlagerte Störungen durch Umgebungsgeräusche) genutzt werden. Nutzt man die in Abschnitt 2.1.2 beschriebenen Zusammenhänge, kann die Richtcharakteristik des Arrays in eine gewünschte Schalleinfallsrichtung geschwenkt werden. Im Umkehrschluss kann dann den Laufzeitverzögerungen zwischen den einzelnen Sensorpaaren insgesamt genau eine Schalleinfallsrichtung zugeordnet werden. Um die letztere Eigenschaft auszunutzen, wird zunächst ein Signalvektor der Form

$$\mathbf{y}_{\phi,\theta}(k) = [y_0(k) \quad y_1[k + f_1(\phi, \theta)] \quad \dots \quad y_N[k + f_N(\phi, \theta)]]^T \quad (2.34)$$

definiert.  $f_n(\phi, \theta)$  beschreibt dabei die in Gleichung 2.14 beschriebenen Verzögerungen, mit denen die Sensorsignale zeitlich zueinander ausgerichtet werden. Unter Angabe eines angenommenen Azimuth- und Elevationswinkels kann so der Array-Beobachtungsvektor  $\mathbf{y}_{\phi,\theta}[k]$  beliebig im Raum ausgerichtet werden. Zur Nutzung der unter Gleichung 2.34 genannten Laufzeitbeziehungen erfolgt nun die Berechnung der räumlichen Korrelationsmatrix  $\mathbf{R}_{\phi,\theta}$ . Diese beschreibt die Signalleistung zwischen den einzelnen Sensorsignalen und wird auch als Kovarianzmatrix bezeichnet.

$$\mathbf{R}_{\phi,\theta} = \mathbb{E} \left\{ \mathbf{y}_{\phi,\theta}(k) \mathbf{y}_{\phi,\theta}^T(k) \right\} \quad (2.35)$$

$$= \begin{bmatrix} \sigma_{y_0}^2 & r_{\phi,\theta|y_0y_1} & \dots & r_{\phi,\theta|y_0y_N} \\ r_{\phi,\theta|y_1y_0} & \sigma_{y_1}^2 & \dots & r_{\phi,\theta|y_1y_N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{\phi,\theta|y_Ny_0} & r_{\phi,\theta|y_Ny_1} & \dots & \sigma_{y_N}^2 \end{bmatrix} \quad (2.36)$$

$r_{\phi,\theta|y_iy_j}$  beschreibt dabei die Kreuzkorrelationsfunktion zwischen dem i-ten und j-ten Signal. Die Hauptdiagonale von  $\mathbf{R}_{\phi,\theta}$  beinhaltet die Varianz  $\sigma_{y_n}^2$  der einzelnen Sensorkanäle. Unter Annahme mittelwertfreier Signale und einer Blocklänge  $K$  können diese mit

$$\sigma_{y_n}^2 = \mathbb{E} \left\{ y_n(k)^2 \right\} = \frac{1}{K} \sum_{k=0}^{K-1} y_n(k)^2 \quad (2.37)$$

errechnet werden. Durch Faktorisierung von  $\mathbf{R}_{\phi,\theta}$  mit

$$\mathbf{R}_{\phi,\theta} = \boldsymbol{\Sigma} \tilde{\mathbf{R}}_{\phi,\theta} \boldsymbol{\Sigma} \quad (2.38)$$

und der Diagonalmatrix  $\boldsymbol{\Sigma}$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{y_0}^2 & 0 & \dots & 0 \\ 0 & \sigma_{y_1}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \sigma_{y_N}^2 \end{bmatrix}$$

resultiert eine symmetrische Matrix  $\tilde{\mathbf{R}}_{\phi,\theta}$ .

$$\tilde{\mathbf{R}}_{\phi,\theta} = \begin{bmatrix} 1 & \rho_{\phi,\theta|y_0y_1} & \dots & \rho_{\phi,\theta|y_0y_N} \\ \rho_{\phi,\theta|y_1y_0} & 1 & \dots & \rho_{\phi,\theta|y_1y_N} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{\phi,\theta|y_Ny_0} & \rho_{\phi,\theta|y_Ny_1} & \dots & 1 \end{bmatrix} \quad (2.39)$$

Diese beinhaltet die Korrelationskoeffizienten zwischen dem  $i^{ten}$  und  $j^{ten}$  Signal in der Form

$$\rho_{\phi,\theta|y_iy_j} = \frac{r_{\phi,\theta|y_iy_j}}{\sigma_{y_i}\sigma_{y_j}} \quad (2.40)$$

$$i, j = 0, 1, \dots, N \quad \text{und} \quad \rho_{\phi,\theta|y_iy_j} \in [-1, 1]$$

Aufgrund der Symmetrie, der positiven Semi-Definitheit und der Tatsache, dass alle Diagonalelemente der Matrix  $\tilde{\mathbf{R}}_{\phi,\theta}$  gleich eins sind kann gezeigt werden, dass für den Wertebereich der Determinanten stets gilt:

$$0 \leq \det [\tilde{\mathbf{R}}_{\phi,\theta}] \leq 1 \quad (2.41)$$

Für ein Array mit zwei Sensoren kann gezeigt werden:

$$\tilde{\mathbf{R}}_{\phi,\theta} = \begin{bmatrix} 1 & \rho_{\phi,\theta|y_0y_1} \\ \rho_{\phi,\theta|y_1y_0} & 1 \end{bmatrix} \quad (2.42)$$

$$\begin{aligned} \det [\tilde{\mathbf{R}}_{\phi,\theta}] &= 1 - \rho_{\phi,\theta|y_0y_1}^2 \\ \rho_{\phi,\theta|y_0y_1}^2 &= 1 - \det [\tilde{\mathbf{R}}_{\phi,\theta}] \end{aligned} \quad (2.43)$$

Für den allgemeinen Fall mit einer Anzahl von  $N + 1$  Sensoren folgt (ohne Beweis):

$$\rho_{\phi,\theta|y_0:y_N}^2 = 1 - \det [\tilde{\mathbf{R}}_{\phi,\theta}] \quad (2.44)$$

Dieses Korrelationsverfahren weist somit folgende Eigenschaften auf:

1.  $0 \leq \rho_{\phi,\theta|y_0:y_N}^2 \leq 1$
2. Sind die Signale vollständig korreliert gilt  $\rho_{\phi,\theta|y_0:y_N}^2 = 1$
3. Sind die Signale vollständig unkorreliert gilt  $\rho_{\phi,\theta|y_0:y_N}^2 = 0$
4. Ist eines der  $N + 1$  Signale unkorreliert, so wird die Korrelation über die verbleibenden  $N$  Signale gemessen.

$$\tau^{MCCC} = \arg \max_{\phi,\theta} \left\{ 1 - \det [\tilde{\mathbf{R}}_{\phi,\theta}] \right\} \quad (2.45)$$

$$= \arg \max_{\phi,\theta} \left\{ 1 - \frac{\det [\mathbf{R}_{\phi,\theta}]}{\prod_{n=0}^N \sigma_{y_n}^2} \right\} \quad (2.46)$$

$$= \arg \min_{\phi,\theta} \det [\tilde{\mathbf{R}}_{\phi,\theta}] \quad (2.47)$$

$$= \arg \min_{\phi,\theta} \det [\mathbf{R}_{\phi,\theta}] \quad (2.48)$$

## 2.3 Eigenschaften von Sprachsignalen

Dieser Abschnitt befasst sich mit den Eigenschaften von Sprachsignalen sowie dessen Bedeutung für die spätere Wahl der Systemparameter in Abschnitt 2.5.1. Da das hier zu realisierende System keine Spracherkennung durchführt, sonder die Richtung einer Sprachquelle im Raum detektieren werden im folgenden die Stationarität sowie der Energieverlauf von Sprache diskutiert [12, 23 f].

### 2.3.1 Stationarität

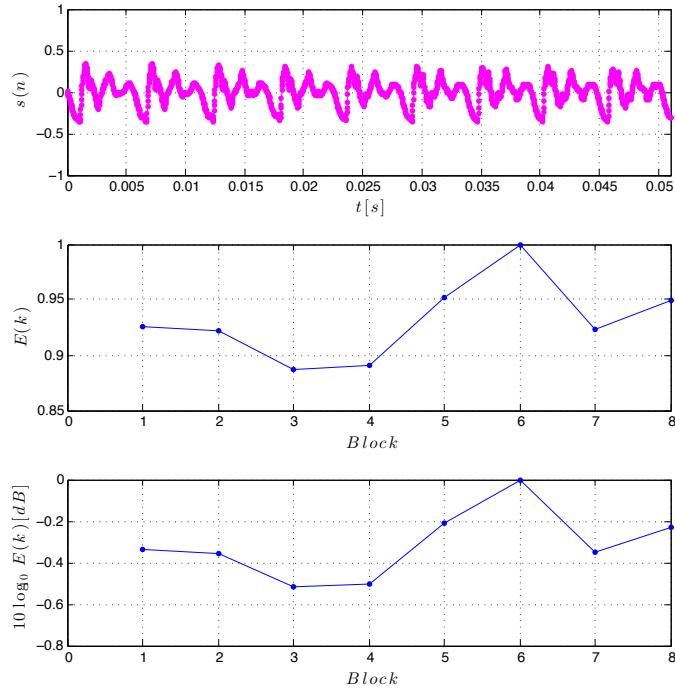
Zur statistischen Bewertung von Sprachsignalen ist es sinnvoll, eine gewisse Anzahl von Stichproben in die Betrachtung einzubeziehen. Das Signale wird deswegen in Abschnitte eine festen Länge unterteilt, was im allgemeinen als Rahmenbildung bezeichnet wird [12, S 23]. Aufgrund der bei Sprache üblichen starken Pegelschwankungen, kann diese im allgemeinen nicht als stationär betrachtete werden. Unter der Bedingung einer ausreichend kurzen Rahmenlänge kann aber von einer Quasistationarität ausgegangen werden. In Pikora [12] wird eine Rahmenzeit von 50ms als quasistationär angenommen. Unter sucht man ein Sprachsignale mit dieser Länge auf dessen Energieverlauf resultiert, dass sich die Energie nur um ein Minimum ändern wie in Abb. 2.7 dargestellt. Im folgenden wird daher diese Dauer als maximale Rahmenlänge spezifiziert.

### 2.3.2 Energie

Zur Richtungsdetektion einer Sprachquelle ist es sinnvoll zu unterscheiden, zu welchem Zeitpunkt tatsächlich gesprochen wird und zu welchem ausschließlich Umgebungsgeräusche vorliegen. Hierzu wird die Energie des Signals blockweise nach Gleichung 2.49 ermittelt.

$$E(k) = \sum_{n=0}^{N-1} s(n)^2 \quad (2.49)$$

$E(k)$  steht hier für die Energie im  $k^{ten}$  Signalblock. Abb. 2.8 illustriert den Energieverlauf eines Sprachsignals mit einer Dauer von 0,4s und einer Signalblocklänge von 256 Abtastwerten. Ausschlaggebend bei der Energieberechnung ist die Dimensionierung der Schwelle, ab der ein Abschnitt als Pause interpretiert wird. Hierbei bietet es sich an, dass Signal im Vorfeld zu sichten und die Schwelle empirisch zu ermitteln. Bei Betrieb des Systems in realer Umgebung ist diese abhängig von den Raumgegebenheiten einzustellen.



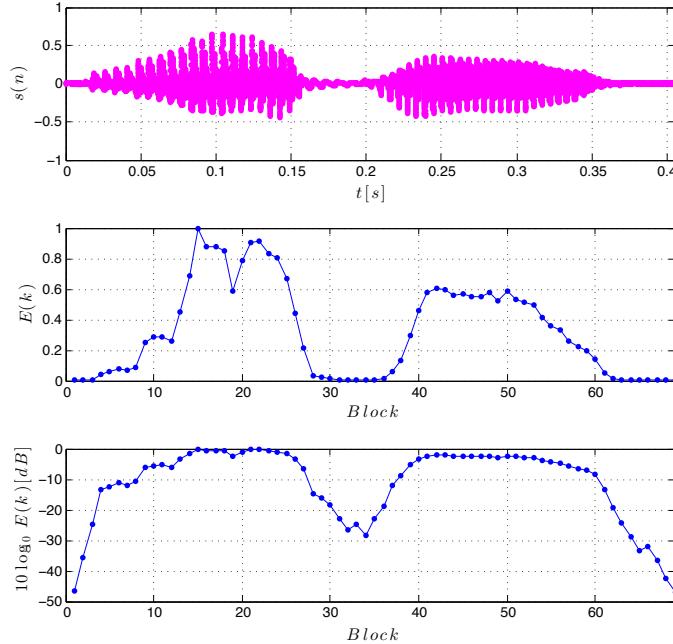
**Abb. 2.7:** Zeitverlauf eines Sprachsignals mit einer Länge von  $T = 50ms$  sowie der Blockenergie bei einer Blocklänge von 256 Abtastwerten.

## 2.4 Mikrofonarray

Abb. 2.9 zeigt zwei Ansichten des kugelförmigen Mikrofonarrays. Dieses besteht aus einem Kunststoffkörper in den acht Mikrofonkapseln, die äquidistant in einer Würfelgeometrie angeordnet sind, wobei die Würfecken die Berührungs punkte zur Kugeloberfläche bilden. Zur Beschreibung der Mikrofonpositionen in solch einer geometrischen Anordnung wird der unter Abschnitt 2.1.2 eingeführte Vektor  $\mathbf{m}_{3 \times 1}$  in Kugelkoordinaten verwendet. Jedes Mikrofon kann so durch den Kugelradius  $r$  sowie seinen Azimuth- und Elevationswinkel beschrieben werden.

$$\mathbf{m}_n = \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = r \cdot \begin{bmatrix} \cos(\phi_n) \cos(\theta_n) \\ \cos(\phi_n) \sin(\theta_n) \\ \sin(\phi_n) \end{bmatrix} \quad \text{mit } n = 0, 1, \dots, 7 \quad (2.50)$$

Zur Berechnung der Mikrofonwinkel werden zunächst die bekannten geometrischen Zusammenhänge am Würfel genutzt. Die Raumdiagonale  $d$  wird mit der Kantenlänge  $a$  durch folgende Gleichung bestimmt



**Abb. 2.8:** Zeitverlauf eines kurzen Sprachsignals sowie der Blockenergie bei einer Blocklänge von 256 Abtastwerten.

$$d = 2r = a \cdot \sqrt{3} \quad (2.51)$$

Die Kantenlänge entspricht hier dem Abstand zum jeweils benachbarten Mikrofon. Dieser beträgt mit dem oben genannten Radius

$$a = \frac{2r}{\sqrt{3}} = \frac{2 \cdot 50mm}{\sqrt{3}} \approx 57,7mm \quad (2.52)$$

Zur Ermittlung der Elevationswinkel  $\phi_n$  kann die halbe Kantenlänge genutzt werden.

$$\frac{a}{2} = r \cdot \cos(\phi) \cos(\theta) \quad (2.53)$$

Durch Einsetzen von Gleichung 2.52 in Gleichung 2.54 folgt

$$\frac{2r}{2\sqrt{3}} = r \cdot \cos(\phi) \cos(\theta) \quad (2.54)$$

$$\frac{1}{\sqrt{3}} = \cos(\phi) \cos(\theta) \quad (2.55)$$

Da sich die Mikrofonvektoren auf das Zentrum des Würfels beziehen resultiert ein Azimuthwinkel von  $\theta = \frac{\pi}{4} = 45^\circ$ . Nach Einsetzen in Gleichung 2.54 folgt

$$\frac{1}{\sqrt{3}} = \cos(\phi) \underbrace{\cos\left(\frac{\pi}{4}\right)}_{\frac{1}{\sqrt{2}}} \quad (2.56)$$

Damit resultiert unabhängig vom Kugelradius  $r$  der Mikrofonwinkel  $\phi$  mit

$$\phi = \cos^{-1}\left(\sqrt{\frac{2}{3}}\right) \approx 35,2^\circ \quad (2.57)$$

Die Elevationswinkel  $\phi_n$  sind damit betragsmäßig alle identisch. Lediglich das Vorzeichen alterniert zwischen den Mikrofonpositionen in der oberen und der unteren Halbkugel. Für die Winkelpaare in der oberen Halbkugel folgt somit:

$$\begin{aligned} \phi_n &= \phi \\ \theta_n &= \frac{\pi}{4} \cdot (2n + 1) \quad \forall n = 0, \dots, 3 \end{aligned} \quad (2.58)$$

Die Winkelpaare in der unteren Halbkugel sind bestimmt durch:

$$\begin{aligned} \phi_n &= -\phi \\ \theta_n &= \frac{\pi}{4} \cdot (2n - 7) \quad \forall n = 4, \dots, 7 \end{aligned} \quad (2.59)$$

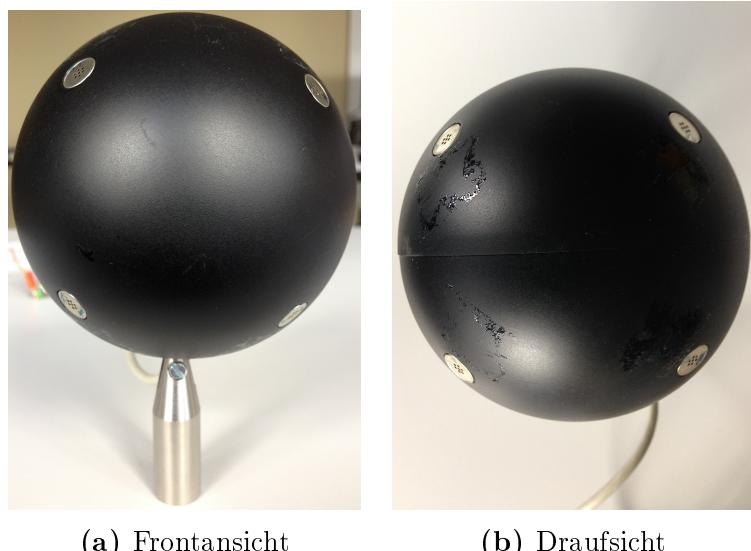
Zur Dimensionierung des Kugelradius wird auf das Kriterium des räumlichen Aliasing in Gleichung 2.16 zurückgegriffen. Mit der minimalen Distanz  $d_{min} = a$  zwischen zwei Mikrofonen folgt

$$a = \frac{2r}{\sqrt{3}} \leq \frac{c}{2f_{max}} \quad (2.60)$$

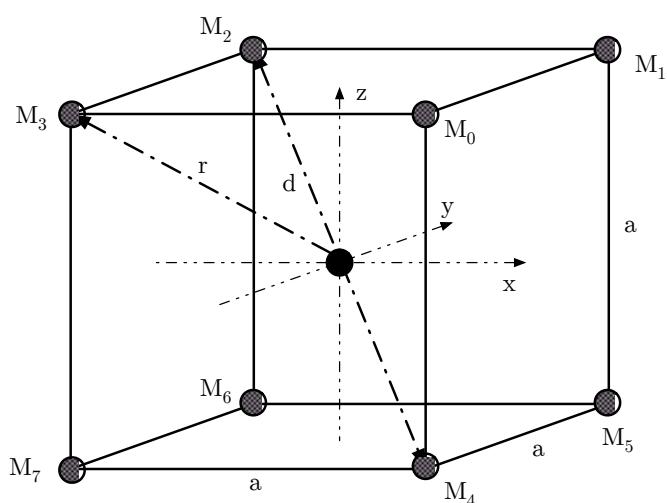
$$r \leq \frac{c}{4f_{max}} \cdot \sqrt{3} \quad (2.61)$$

Für eine maximale Signalfrequenz von  $f_{max} = 3kHz$  resultiert ein Radius von

$$r \leq \frac{343 \frac{m}{s}}{4 \cdot 3kHz} \cdot \sqrt{3} \leq 49,5mm \quad (2.62)$$



**Abb. 2.9:** kugelförmiges Mikrofonarray



**Abb. 2.10:** Sensoranordnung auf der Kugeloberfläche

#### 2.4.1 Untersuchung des kugelförmigen Mikrofonarrays

Zur Verifikation des angenommenen Schallverhaltens an der schallharten Kugeloberfläche wird im folgenden eine Messung der Sensorlaufzeiten aus einer definierten Richtung durchgeführt und anschließend mit den theoretisch berechneten verglichen. Tab. 2.1 zeigt einen tabellarischen Überblick der Mess-

ergebnisse. Der Fehler wird hier in Abtastwerten angegeben und bezieht sich auf eine Frequenz von  $f_a = 48kHz$ .

Sensorpaar	$\tau_{theo}[\mu s]$	$\tau_{gemessen}[\mu s]$	Fehler [Sample]
$m_{13}$	272	168	5
$m_{14}$	184	168	1
$m_{17}$	384	168	11
$m_{24}$	288	168	6
$m_{28}$	384	168	11

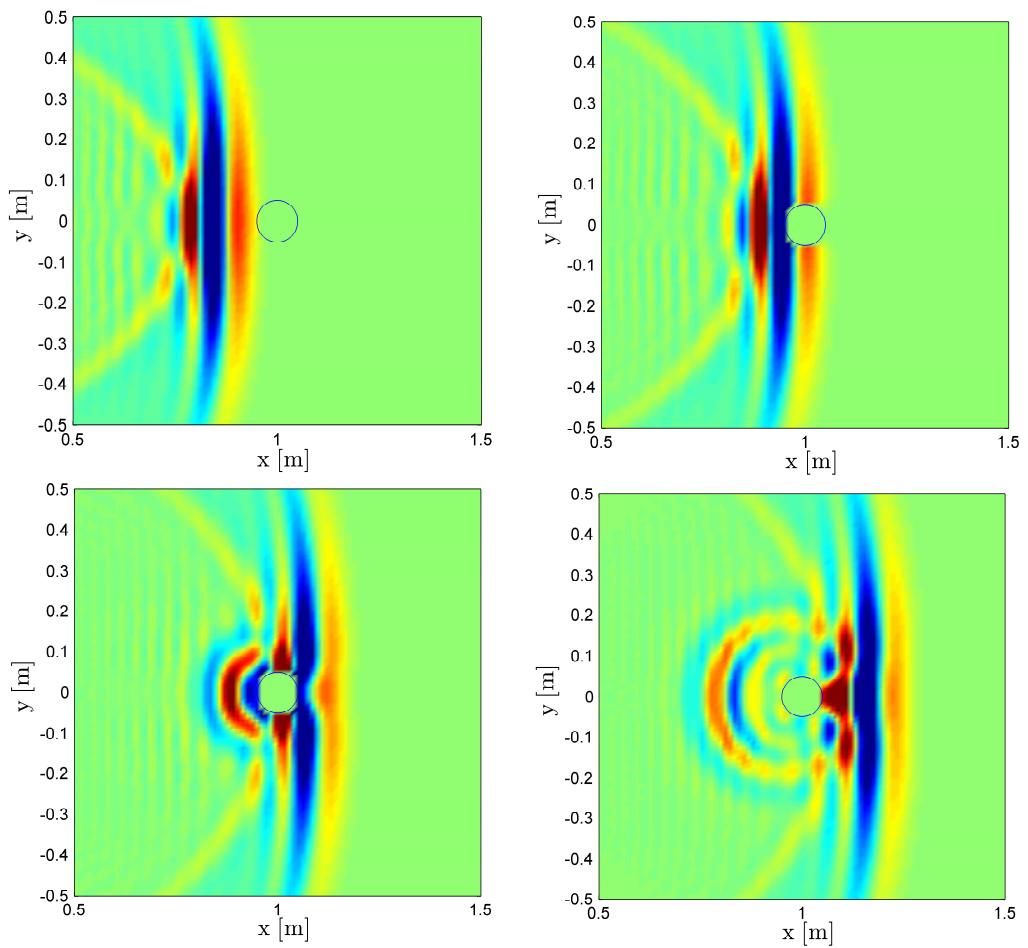
**Tab. 2.1:** Laufzeitmessung an kugelförmigem Mikrofonarray bei den Winkeln  $\phi = 0, \theta = 0$ .

Die bei dieser Messung auftretenden Fehler deuten darauf hin, dass sich die Schallwelle nicht wie erwartet geradlinig am Array vorbei bewegen, sondern mit großer Wahrscheinlichkeit an der Kugeloberfläche gebeugt wird. Abb. 2.11 zeigt eine Simulation (erstellt mit dem MATLAB®-Skript aus [15]) die das Verhalten eines sinusförmigen Schall-Burst mit einer Frequenz von  $f = 3kHz$  an einer schallharten Kugel mit dem Radius  $r = 50mm$  untersucht. Es ist deutlich zu erkennen, dass die Schallwelle erheblich durch das Objekt beeinflusst wird. Eine theoretische Berechnung der Sensorlaufzeiten mit dem in Abschnitt 2.1.2 beschriebenen Modell ist somit nicht möglich. Da das hier zu implementierende System nur mit diskreten Zeiten und somit mit diskreten Winkeln arbeitet können die ermittelten Fehler nicht toleriert werden. Ein Fehler von einem Abtastwert würde einen Fehler von einem Winkelschritt verursachen, wodurch eine zuverlässige Richtungsdetektion unmöglich wäre.

#### 2.4.2 Mikrofonarray Neudesign/Konstruktion

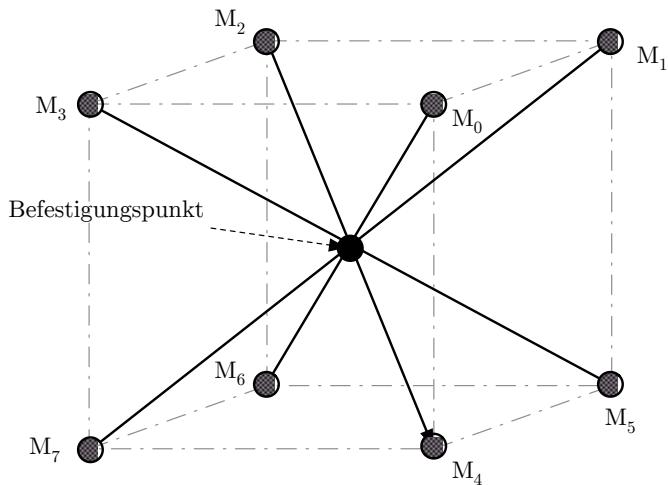
Auf Basis der unter Abschnitt 2.4.1 gewonnenen Erkenntnisse sowie der im Abschnitt 2.1.2 erläuterten Zusammenhänge erfolgt in diesem Abschnitt eine Neukonstruktion des kugelförmigen Mikrofonarrays.

Um eine zuverlässige Aussage über die Schallausbreitung treffen zu können, muss der Körper des Mikrofonarray auf seine minimal nötigen Elemente reduziert werden. Abb. 2.12 zeigt die prinzipielle Konstruktionsidee. Alle schallharten Bestandteile die einen unvorhersehbaren Einfluss auf die Schallausbreitung haben könnten, wurden entfernt, so dass der Körper nun lediglich aus dünnen Drähten besteht. Da sich die Mikrofonkapseln im Vergleich zur Kugel exakt an den selben Stellen befinden, kann dieses Array als Acht-Punkte-Approximation der Kugelgeometrie betrachtet werden.



**Abb. 2.11:** Schallverhalten an schallharter Kugel ( $r = 50\text{mm}$ ) mit einer Schallwelle der Wellenlänge  $\lambda = 114\text{mm}$  und einer Frequenz von  $f = 3\text{kHz}$ .

Zur Prüfung des Schallverhaltens wurde der in Abb. 2.13 illustrierte Prototyp hergestellt. Die Drahtkonstruktion besteht dabei aus 1mm dicken Kupferdrähten, die an den Enden fest mit Lötzinn verbunden sind. Die Mikrofonkapseln wurden mit Hilfe von transparentem Kunststoff (Plexiglas) sowie Heißklebstoff in der oben beschriebenen Abstandsmetrik angeordnet. Die Kapseln selber wurden dabei im Hinblick auf eine spätere Mikrofonkonstruktion aus Pertinax<sup>10</sup> (inkl. einer dünnen Oberflächenschicht aus Kupfer) lediglich mit Klebeband fixiert.



**Abb. 2.12:** Prinzip der Mikrofonkonstruktion

### Laufzeitmessung am Prototyp

Tab. 2.2 zeigt die Ergebnisse von zwei Laufzeitmessungen aus unterschiedlichen Schalleinfallsrichtungen. Gemessen wurden immer die Laufzeitdifferenzen zum Referenzmikrofon  $M_1$ . Zur Verifikation der Geometrie wurde das Array mit einem durch einen Lautsprecher abgestrahlten Rechteckimpuls angeregt. Die dazu benötigten theoretischen Laufzeitdifferenzen wurden mit der in Gleichung 2.14 eingeführten Mikrofonfunktion  $f_n(\phi, \theta)$  errechnet. Die Differenz zwischen theoretischem und gemessenem Wert wird als Anzahl von Abtastwerten mit einer Frequenz von  $f_a = 48kHz$  angegeben. Die Messergebnisse weichen nur leicht von den theoretisch berechneten ab, so dass sich die Arraykonstruktion aus dünnen Drähten als geeignet erweist.

Da die Konstruktion, bedingt durch die dünnen Kupferdrähte, mechanisch sehr instabil ist und die Mikrofonkapseln im Vergleich zur Kugel nicht exakt

<sup>10</sup>Phenolharz mit Papierfasern (sog. Hartpapier)



**Abb. 2.13:** Prototyp des würzelförmigen Mikrofonarrays

gleich angeordnet sind, wird im Folgenden eine Neukonstruktion des Array vorgestellt.

### Äquidistante Arraykonstruktion

Auf Basis der Array-Prototyp Idee wird in diesem Abschnitt ein Array entwickelt, das über eine ausreichend hohe Genauigkeit sowie mechanische Festigkeit verfügt.

Zur Bestimmung der minimal nötigen Herstellungsgenauigkeit wird die Eigenschaft des hier vorliegenden diskreten Systems genutzt. Auf Grund der Zeitdiskretisierung können nur Laufzeitdifferenzen gemessen werden, die ein Vielfaches der Abtastperiode  $T_a$  betragen. Die Strecke  $d_{T_a}$ , die vom Schall innerhalb einer Abtastperiode zurückgelegt wird, ergibt sich durch

$$d_{T_a} = c \cdot T_a = \frac{c}{f_a} \quad (2.63)$$

Tab. 2.3 zeigt unterschiedliche Toleranzbereiche bei den im Audiobereich gän-

Sensorpaar	$\tau_{theo}[\mu s]$	$\tau_{gemessen}[\mu s]$	Fehler [Sample]
$\phi = 0, \theta = 0$			
$m_{12}$	0	0	0
$m_{13}$	168	165	0,14
$m_{14}$	168	162	0,28
$m_{15}$	0	12	0,57
$m_{16}$	0	0	0
$m_{17}$	168	155	0,62
$m_{18}$	168	150	0,84
$\phi = 45, \theta = 45$			
$m_{12}$	-84	-90	0,28
$m_{13}$	0	0	0
$m_{14}$	84	84	0
$m_{15}$	119	112	0,33
$m_{16}$	34	50	-0,76
$m_{17}$	119	130	-0,52
$m_{18}$	203	190	0,62

**Tab. 2.2:** Laufzeitmessung aus zwei unterschiedlichen Schalleinfallsrichtungen

gigen Abtastraten. Um einen Messfehler der Laufzeitdifferenz zu vermeiden, muss der entsprechende Toleranzbereich eingehalten werden.

Abtastfrequenz [kHz]	$d_{Ta}$ [mm]	Toleranz [mm]
8	42,8	$\pm 21,4$
16	21,4	$\pm 10,7$
24	14,3	$\pm 7,1$
48	7,1	$\pm 3,5$
96	3,5	$\pm 1,8$

**Tab. 2.3:** Toleranzbereich bei unterschiedlichen Abtastfrequenzen

Als grundsätzliche Konstruktionsidee wird hier ein Steckmechanismus eingesetzt. Die Einzelteile können so aus einem flachen Werkstoff (z.B. einer Blechtafel oder Pertinaxplatten) ausgeschnitten werden. Fertigungsprozesse solcher Art sind aufwandsarm da die Einzelteile mit einem CNC<sup>11</sup>-Laser oder einer CNC-Wasserschneidemaschine aus großen Werkstofftafeln ausgearbeitet werden können. Auf Grund der hohen Schneidegeschwindigkeit moderner

<sup>11</sup>Computerized Numerical Control

Werkzeugmaschinen und der geringen Einricht -und Spannzeit kann solch eine Konstruktion bereits in geringer Stückzahl kostengünstig hergestellt werden.

Zur Befestigung der einzelnen Teile miteinander werden, wie in Abb. 2.14 dargestellt, entsprechende Ausschnitte eingearbeitet an denen diese dann zusammengesteckt werden. Um eine ausreichend hohe Festigkeit der Steckverbindung zu gewährleisten, werden diese Ausschnitte entsprechend einer Übergangspassung angefertigt. Abb. 2.15 zeigt die beiden X-förmigen Grundelemente der Konstruktion. Diese bilden nach dem Zusammenstecken die vier durch das Zentrum laufenden Streben der in Abb. 2.12 gezeigten Prinzipskizze.

Zur Befestigung der empfindlichen Mikrofonkapseln wird ebenfalls ein Steckmechanismus entworfen. Wie in Abb. 2.16 illustriert, besteht die Halterung aus einem kleinen Stück Pertinax mit einem halboffenen kreisrunden Ausschnitt. Der Durchmesser dieses Halbkreises entspricht exakt dem der Mikrofonkapsel von 9,7mm wobei an den beiden Ecken der Öffnung zwei nach innen ragenden Rundungen angebracht sind. Diese werden beim Einsticken nach außen gebogen und sorgen für eine ausreichend hohe Klemmkraft zum Fixieren der Kapsel. Durch diesen Mechanismus lassen sich die Mikrofone nach dem Einbau feinjustieren und können darüber hinaus leicht aus dem Array entfernt werden.



**Abb. 2.14:** Prinzip des Steckmechanismus

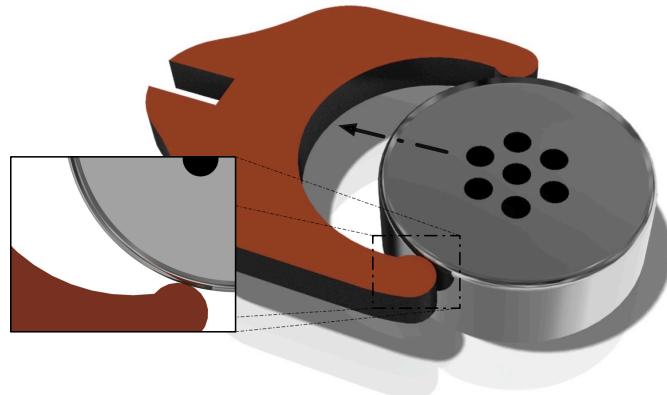
Um die Mikrofonkapseln im Vergleich zur Kugelgeometrie exakt gleich auszurichten, wird die gesamte Arraygeometrie, wie in Abb. 2.17 dargestellt,



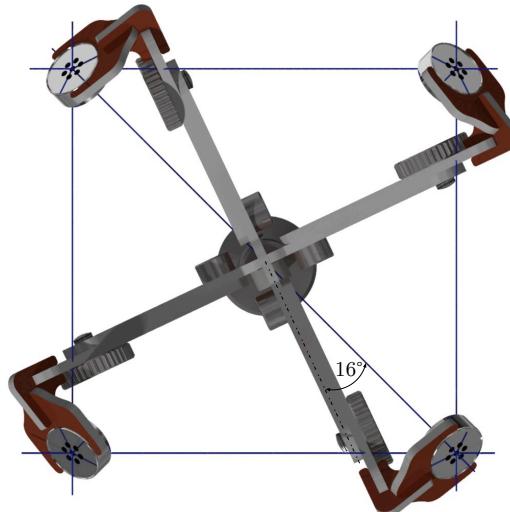
**Abb. 2.15:** Grundkomponenten des Mikrofonarrays

um einen Winkel von 16 gedreht. Die Mikrofone sind so für den Schall von allen Seiten ohne direktes Hindernis erreichbar. Der Abstand zwischen zwei gegenüberliegenden Kapseln beträgt wie oben genannt 100mm, so dass sich die Kapseloberseite, wie in Abb. 2.18 gezeigt, genau auf der Kugeloberfläche befindet. Die in blau eingezeichneten Linien stellen dabei die gedachten Würfelkanten dar und sind somit in der realen Konstruktion nicht vorhanden. Die doppel-x-förmige Konstruktion bildet somit die Kugeloberfläche genau an den acht Mikrofonpositionen ab. Des weiteren ist der Grundkörper, wie in Abb. 2.19 dargestellt, mittels verschiebbaren Verlängerungsstreben versehen. Mit diesen ist es möglich, den Mikrofonabstand stufenlos bis auf 147mm zu vergrößern.

Zur Herstellung des Prototypen wird die in Abb. 2.20 dargestellte CNC-



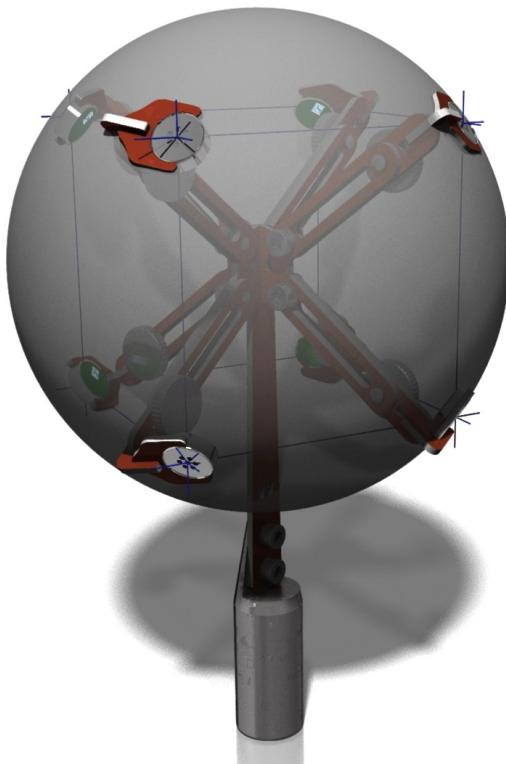
**Abb. 2.16:** Haltemechanismus der Mikrofonkapseln



**Abb. 2.17:** Drehung der Arraygeometrie zur exakten Positionierung der Mikrofonkapseln

Fräsmaschine eingesetzt. Diese verfügt über eine Toleranz von  $\pm 0,1\text{mm}$  und liegt damit weit unter der minimal nötigen Toleranz in Tab. 2.3. Die Einzelteilkonstruktionen müssen hierfür in das DXF<sup>12</sup>-Dateiformat konvertiert werden, wodurch sie ohne weitere Maschinenprogrammierung direkt zur Fertigung verwendet werden könnten. Zu beachten ist, dass die Software der Maschine keine Korrektur der Fräserabmaße vornimmt. Jedes Einzelteil muss diesbezüglich erneut bearbeitet werden, so dass alle Außenkonturen um den

<sup>12</sup>Drawing Interchange File Format



**Abb. 2.18:** Vergleich zwischen Kugel- und Würfelgeometrie

Fräserradius vergrößert und alle Innenkonturen um den Fräserradius verkleinert werden. Abb. 2.21 zeigt das zusammengesetzte sowie verdrahtete Mikrofonarray.

## 2.5 Simulation

Zur Verifikation des in Abschnitt 2.1 dargestellten mathematischen Modells wird eine umfangreiche Simulation in MATLAB® erstellt. Bezüglich der späteren Echtzeitimplementierung ist es notwendig, alle nötigen Programmfunctionen so zu erstellen, dass diese ebenfalls mit der zur Verfügung stehenden DSP- und C-Bibliothek umgesetzt werden können. Obwohl die Simulation nicht echtzeitkritisch ist, soll die Erfüllung des Echtzeitkriteriums bereits hier einen zentralen Punkt darstellen. Die Simulation beinhaltet folgende Komponenten:

1. Ermittlung systembedingter Parameter

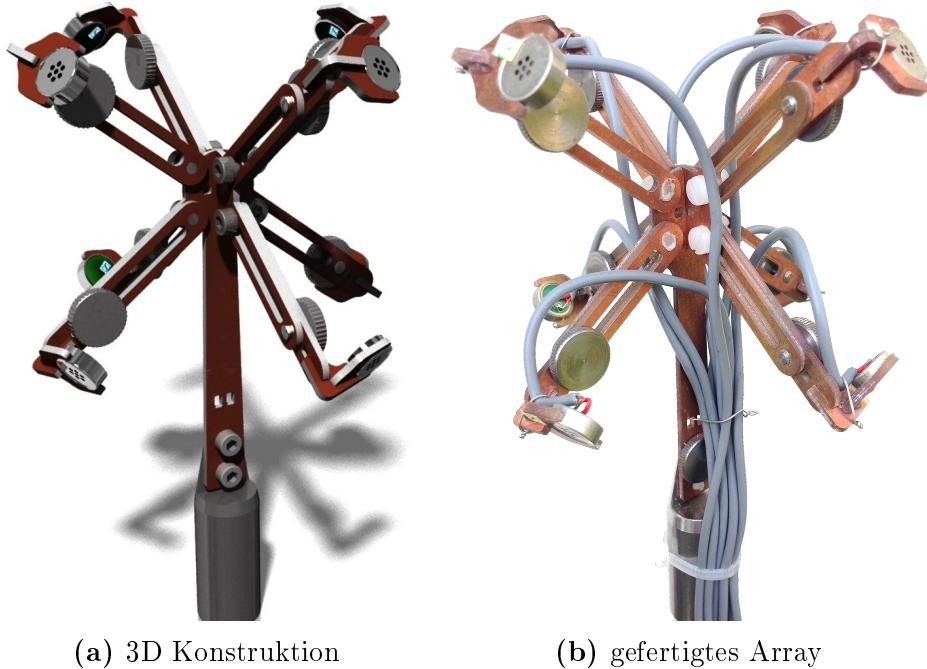


**Abb. 2.19:** Variabel verstellbare Mikrofonabstände



**Abb. 2.20:** Fertigung der Einzelteile unter Verwendung einer CNC-Fräsmaschine

2. Erstellung synthetischer Signale
3. Kreuzkorrelation unter Verwendung der FFT
4. Algorithmus zur Berechnung des MCCC



**Abb. 2.21:** Vergleich zwischen 3D Konstruktion und Fertigungsergebnis

5. Optimierungsverfahren zur Ergebnis- und Geschwindigkeitsverbesserung

### 2.5.1 Systemparameter

Im Folgenden werden alle systembedingten Parameter ermittelt. Diese sind

- Winkelauflösung inkl. Validierung des Winkelraums
- Anzahl abzusuchender Richtungen
- Signalblocklänge
- Anzahl der Kreuzkorrelationen

#### Winkelauflösung

Als grundsätzliche Anforderung an das System ist die mit der vorliegenden Hardware maximal zureichende Winkelauflösung abzubilden. Das heißt, je kleiner die Winkelauflösung (entspricht der Winkelschrittweite), desto größer ist die erreichbare Genauigkeit bei der Winkeldetektion. Da wie bereits erwähnt ein diskretes System realisiert wird, können nur Laufzeiten gemessen

werden, die einem ganzzahligen Vielfachen der Abtastperiode  $T_a$  entsprechen. Auf Grund dessen wird im Folgenden die Laufzeitdifferenz als SDOA<sup>13</sup> bezeichnet. Durch diese Zeitrasterung entsteht ebenfalls eine Winkelrasterung im Abstand der Winkelschrittweite, wodurch ein direkter Zusammenhang zwischen Winkelaufösung und verwendeter Systemabtastrate entsteht. Auf Grund der äquidistanten Anordnung benachbarter Sensoren ist die Winkelaufösung  $\alpha_{\phi,\theta}$  für Azimuth- und Elevationswinkel identisch.  $\alpha_{\phi,\theta}$  kann unter Verwendung von Gleichung 2.6 und  $d = d_{min}$  errechnet werden.  $d_{min}$  beschreibt dabei die minimale Distanz zwischen zwei Mikrofonen. Durch Hinzufügen der Abtastperiode  $T_a$  folgt  $\alpha_{\phi,\theta}$  mit

$$\alpha_{\phi,\theta} = \frac{\pi}{2} - \arccos \left( 1 \cdot T_a \cdot \frac{c}{d_{min}} \right) \quad (2.64)$$

Der kleinste zu erreichende Winkelschritt hängt also direkt mit der kleinsten messbaren Laufzeitdifferenz von  $1 \cdot T_a$  zusammen. Tab. 2.4 zeigt mögliche Winkelaufösungen bei den im Audiobereich üblichen Abtastfrequenzen.

Abtastfrequenz [kHz]	Winkelauflösung [°]
8	47,9
16	21,8
24	14,3
48	7,1
96	3,5

**Tab. 2.4:** Winkelauflösung bei unterschiedlichen Abtastfrequenzen

### Validierung des Winkelraums

Zur Validierung des abzubildenden Winkelraums ist es notwendig, jede mögliche Kombination aus Azimuth- und Elevationswinkel nachzumessen. Dazu wird ein Simulationsskript erstellt, das in Abhängigkeit vom eingestellten Winkelpaar den mittleren quadratischen Fehler  $e_{rms}$  berechnet. Dieser ist geeignet, um die Abweichung eines Schätzers von dem zu schätzenden Wert zu ermitteln. Da die Schätzung jeder Richtung jeweils zwei Winkel liefert, werden die Fehler zunächst getrennt ermittelt.

Der Fehler  $e_{\phi|rms}$  für den Elevationswinkel  $\phi$  ist gegeben durch

$$e_{\phi|rms} = \sqrt{\frac{1}{M-1} \sum_{n=0}^{M-1} |\hat{\phi}_n - \phi|^2}. \quad (2.65)$$

---

<sup>13</sup>Sample Difference of Arrival

Für den Fehler  $e_{\theta|rms}$  des Azimuthwinkel  $\theta$  folgt so

$$e_{\theta|rms} = \sqrt{\frac{1}{M-1} \sum_{n=0}^{M-1} |\hat{\theta}_n - \theta|^2}. \quad (2.66)$$

Durch bilden des arithmetischen Mittelwerts resultiert der Gesamtfehler pro Richtung  $\bar{e}_{\phi,\theta|rms}$  durch

$$\bar{e}_{\phi,\theta|rms} = \frac{e_{\phi|rms} + e_{\theta|rms}}{2}. \quad (2.67)$$

Abb. 2.22 zeigt das Flussdiagramm des Validierungsalgorithmus. Hier erfolgt die Wertvalidierung in einer dreifach geschachtelten Schleife. Für jedes Winkelpaar wird der gleiche Signalblock verwendet und in  $M$  Unterblöcke geteilt. Diese werden alle entsprechend dem gewählten Winkelpaar synthetisch verzögert und nacheinander dem MCCC-Algorithmus zugeführt. Die Ergebnisse werden entsprechend den oben genannten Gleichungen verarbeitet und gespeichert. Der Algorithmus endet, wenn alle möglichen Winkelkombinationen durchlaufen wurden.

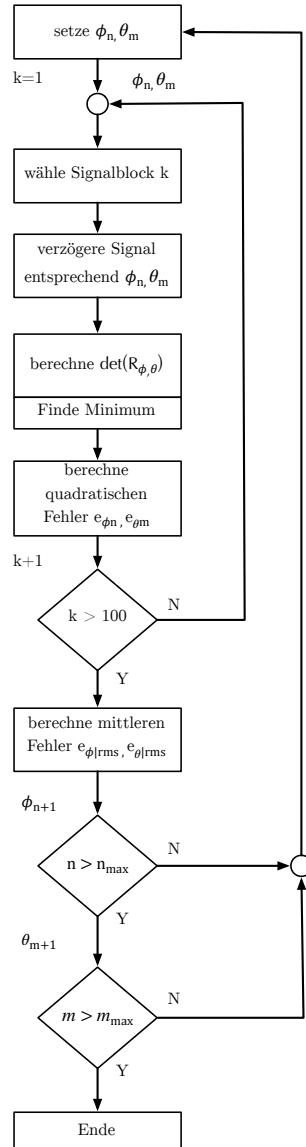
Abb. 2.23 zeigt den Winkelfehler  $e_{\phi|rms}$ ,  $e_{\theta|rms}$  sowie  $\bar{e}_{\phi,\theta|rms}$  in Abhängigkeit beider Raumwinkel. Als Quellensignal dient mittelwertfreies bandbegrenztes weißes Rauschen, mit einer Bandbreite von  $B_{Noise} = 2700Hz$  im Bereich  $300Hz \leq f_{Noise} \leq 3000Hz$ . Jeder Winkel wird mit einer Anzahl von  $M = 100$  Durchläufen simuliert. Die Auswertung der Fehlerkurven ergibt einen maximalen Fehler von:

$$\begin{aligned} e_{\phi|max} &= \max_{\phi,\theta} e_{\phi|rms} = 0,05 \\ e_{\theta|max} &= \max_{\phi,\theta} e_{\theta|rms} = 0,4 \\ \bar{e}_{\phi,\theta|max} &= \max_{\phi,\theta} \bar{e}_{\phi,\theta|rms} = 0,25 \end{aligned}$$

Auf Grund der sehr geringen Fehlerwerte kann davon ausgegangen werden, dass der Winkelraum innerhalb unter Berücksichtigung der Winkelschrittweite fehlerfrei abgebildet werden kann.

### Anzahl abzusuchender Richtungen

Unter Verwendung der in Tab. 2.4 angegebenen Winkelauflösungen sowie dem in Abschnitt 1.2 genannten Winkelbereich für  $\phi$  und  $\theta$  ergeben sich die in Tab. 2.5 dargestellten Werte für die abzusuchenden Richtungen  $S_{\phi,\theta}$ . Der Suchaufwand wächst bei jeder Verdopplung der Abtastfrequenz überproportional an. Wie sich später zeigen wird, muss, um eine ausreichend hohe Win-

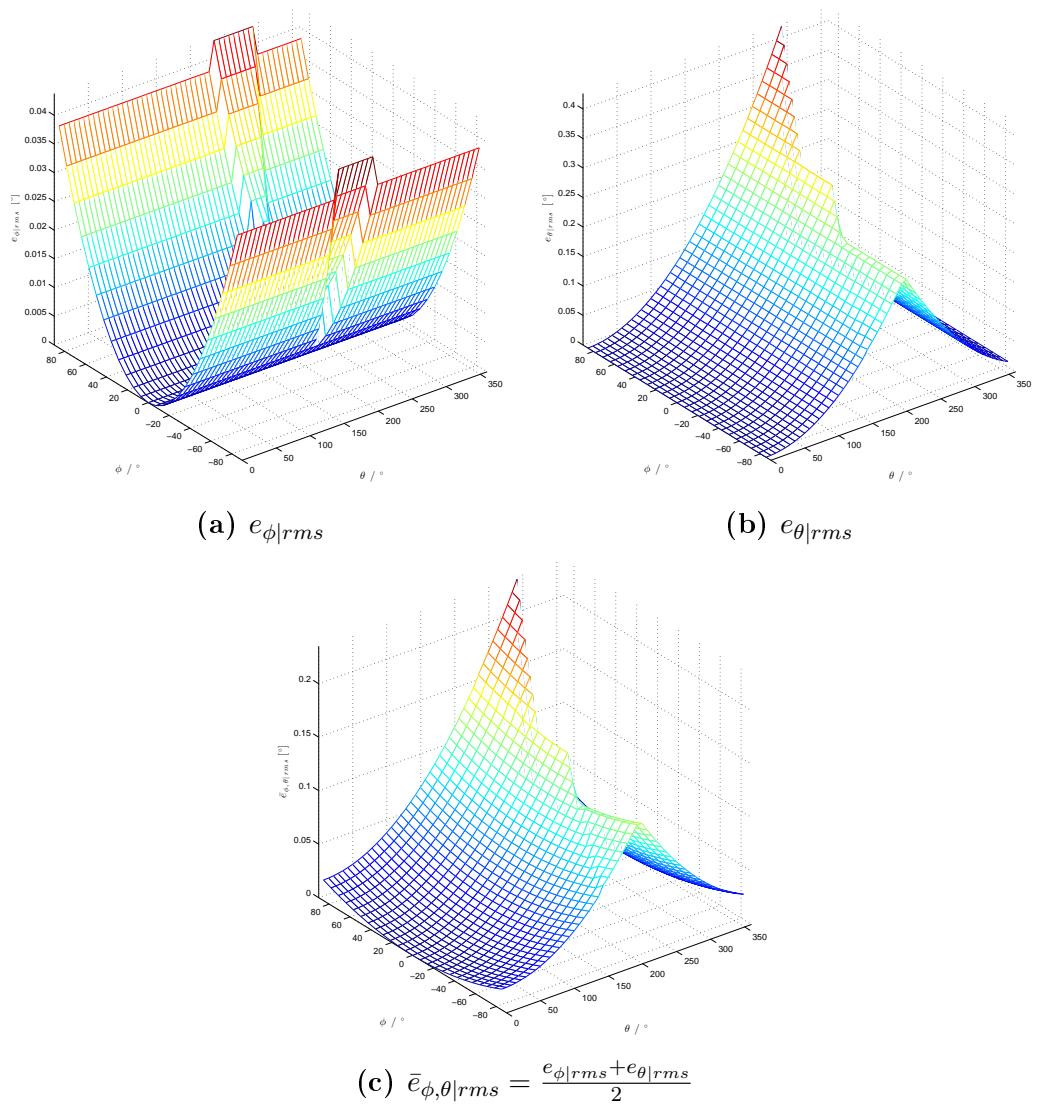


**Abb. 2.22:** Algorithmus zur Validierung des abzubildenden Winkelraums

keine Auflösung zu erreichen, ein optimiertes Suchverfahren entwickelt werden, wodurch das Echtzeitkriterium erfüllt wird.

### Signalblocklänge

Zum Festlegen der Signalblocklänge  $L$  und somit auch der Fensterlänge  $K$  der Kreuzkorrelation muss die maximale am Array messbare SDOA bestimmt



**Abb. 2.23:** Ergebnis der Winkelvalidierung für  $e_{\phi|rms}$ ,  $e_{\theta|rms}$  und  $\bar{e}_{\phi,\theta|rms}$  bei einer Abtastfrequenz von  $f_a = 48kHz$

Abtastfrequenz [kHz]	$N_\phi$	$N_\theta$	$S_{\phi,\theta} = N_\phi \cdot N_\theta$
8	4	8	32
16	9	17	153
24	13	26	338
48	26	51	1326
96	51	102	5202

**Tab. 2.5:** Abzusuchende Richtungen  $S_{\phi,\theta}$  bei unterschiedlichen Abtastfrequenzen

werden. Diese entsteht an der hier verwendeten würfelförmigen Mikrofonanordnung genau dann, wenn der Elevationswinkel  $\phi = \pm 45$  beträgt und der Azimuthwinkel  $\theta$  sich im 45 Raster bewegt. In diesem Fall muss der Schall die Distanz  $2 \cdot r$  zwischen zwei gegenüberliegenden Mikrofonen zurücklegen. Die maximale Verzögerung  $K_{max}$  bezogen auf die Abtastfrequenz ergibt sich durch

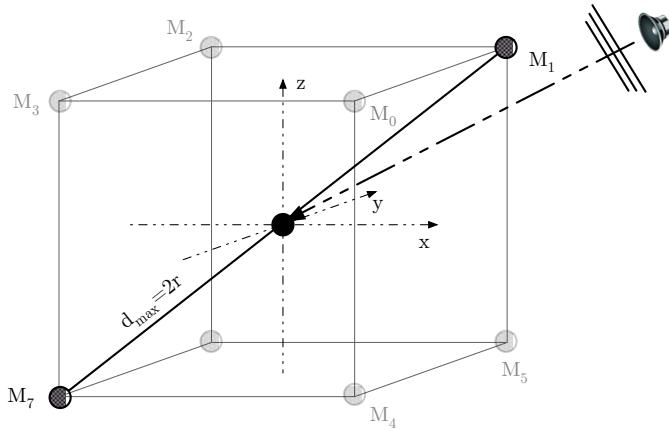
$$K_{max} = \frac{2r}{c} \cdot f_a \quad (2.68)$$

Tab. 2.6 zeigt die maximalen SDOA in Abhängigkeit verschiedener Abtastfrequenzen.

Abtastfrequenz [kHz]	Maximale SDOA [Sample]
8	$\pm 2$
16	$\pm 5$
24	$\pm 7$
48	$\pm 14$
96	$\pm 28$

**Tab. 2.6:** Maximale SDOA bei zwei gegenüberliegenden Mikrofonen

Bei der Wahl der Signalblocklänge  $L$  muss die maximale Verzögerung berücksichtigt werden.  $L$  muss mindestens so groß gewählt werden, dass die maximal positive sowie maximal negative SDOA als Maximum im Korrelogramm sichtbar werden. Weiterhin sind die in Abschnitt 2.2.1 genannten Kriterien im Bezug zur FFT einzuhalten. Die Signalblocklänge  $L$  muss somit die halbe Länge der FFT-Länge  $K$  besitzen. Weiterhin gilt für  $K = 2^p$  und  $p \in \mathbb{N}^+$ . Somit folgt für die Signalblocklänge  $L = 2^{p-1}$ .



**Abb. 2.24:** Schalleinfallsrichtung mit der maximal messbaren Laufzeitdifferenz

### Anzahl der Kreuzkorrelationen

Wie bereits in Abschnitt 2.2.2 erwähnt ist es notwendig, für die Ermittlung der Schalleinfallsrichtung die räumliche Korrelationsmatrix  $\mathbf{R}_{\phi,\theta}$  zu erstellen. Dazu müssen zunächst alle nötigen Kreuzkorrelationsfunktionen berechnet werden. Anschließend werden diesen Funktionen Funktionswerte abhängig von den gewünschten Winkeln, entnommen um  $\mathbf{R}_{\phi,\theta}$  zu erstellen. Da die Kreuzkorrelatinsfunktionen zwischen allen Mikrofonpaaren erstellt werden, ist deren Anzahl abhängig von der Mikrofonanzahl  $N$ . Auf Grund der Symmetrieeigenschaft von  $\mathbf{R}_{\phi,\theta}$  müssen  $\frac{N^2-N}{2}$  Kreuzkorrelationsfunktionen sowie  $N$  Varianzen berechnet werden.

### Wahl der Systemparameter

Nach der Untersuchung aller Systemparameter ist es nun notwendig, einen Satz von Parametern zu wählen, mit dem ein reales System auf einem DSP mit endlicher Rechenkapazität realisiert werden kann. Da auf Grund von Vorgaben die Anzahl der Mikrofone mit  $N = 8$  feststeht ergeben sich zur Berechnung  $\frac{N^2-N}{2} = 28$  Kreuzkorrelationen sowie 8 Varianzen. Ziel ist es, eine möglichst kleine Winkelauflösung zu erreichen. So wird die Abtastfrequenz des Systems mit  $f_a = 48kHz$  festgelegt, wodurch sich eine Winkelauflösung von  $\alpha_{\phi,\theta} = 7,1$  ergibt. Nach Tab. 2.5 resultiert so eine Anzahl von  $S_{\phi,\theta} = 1326$  abzusuchenden Richtungen, für jeweils die räumliche Korrelationsmatrix gebildet und dessen Determinante berechnet werden müssen. Auf Grund der Stationaritätseigenschaft von Sprachsignalen ist die Größe des Signalsblocks

so klein wie möglich zu wählen. Gegenläufig dazu bewegt sich die benötigte Rechenzeit, welche den Wert

$$T_{calc_{max}} \leq \frac{L}{f_a} \leq L \cdot T_a \quad (2.69)$$

nicht überschreiten darf. Die minimale Signalblocklänge beträgt bei dieser Abtastfrequenz  $L = 28$ . In Bezug auf die DSP Implementierung wird der Signalblock auf eine Länge von  $L = 256$  gesetzt. Daraus ergibt sich ein Rechenfenster von  $T_{calc_{max}} = 5,33ms$ .

Parameter	Variable	Wert
Anzahl Mikrofone	$N$	8
Abtastfrequenz	$f_a$	48 kHz
Anzahl KKF	-	28
Anzahl Varianz	-	8
Winkelauflösung	$\alpha_{\phi,\theta}$	7, 1
Anzahl Suchrichtungen	$S_{\phi,\theta}$	1326
maximale Rechenzeit	$T_{calc_{max}}$	5,33 ms

**Tab. 2.7:** Gewählte Systemparameter im Überblick

### 2.5.2 Erstellung synthetischer Signale

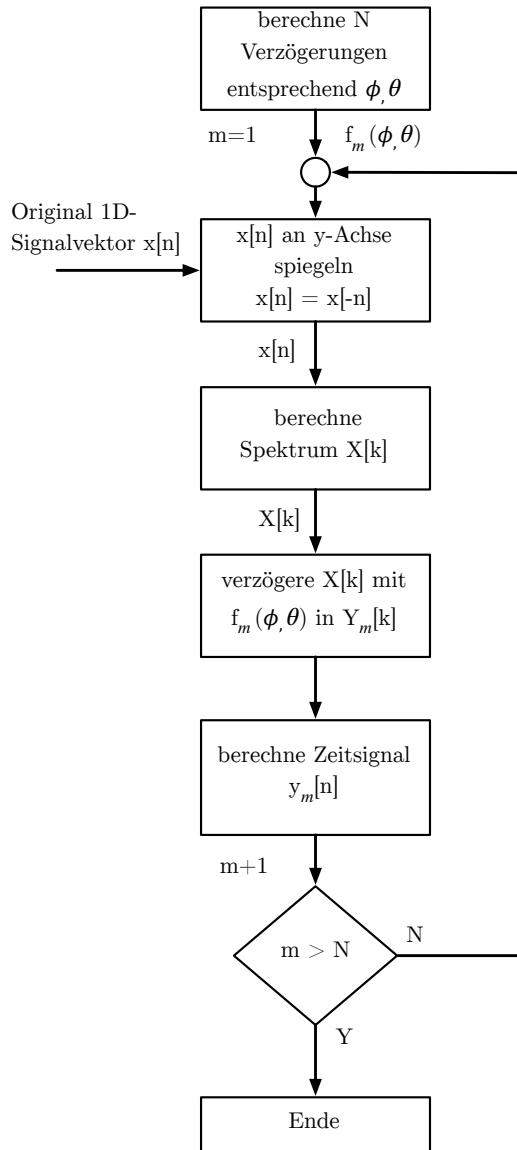
Zur Verifikation korrekter Algorithmusfunktionalität ist es notwendig, über Testsignale zu verfügen, bei denen genaue Kenntnis über die zu erwartenden Ergebnisse vorliegen. Am besten eignen sich dazu synthetisch erzeugte Signale. So lassen sich beliebige Testsignale für jede gewünschte Schalleinfallsrichtung für eine beliebige Anzahl von Mikrofonen erzeugen.

Wie in Abschnitt 2.1.2 beschrieben, lassen sich zu jeder Schalleinfallsrichtung  $(\phi, \theta)$ , bezogen auf das Referenzmikrofon, und jedem sich im Array befindenden Mikrofone jeweils eine Verzögerung zuordnen. Im Umkehrschluss kann so aus einem eindimensionalen Signalvektor ein  $N+1$ -Dimensionaler erzeugt werden in dem jedes der weiteren  $N$  Mikrofone entsprechend einer gewünschten Schalleinfallsrichtung zeitlich verschoben wird. Diese Verschiebung kann mit dem Verschiebungssatz der Fourier-Transformation in Gleichung 2.70 erreicht werden, wobei der Parameter  $N$  für die DFT-Länge steht [21, S. 213].

$$x[n - m] \xrightarrow[N]{DFT} W_N^{mk} \cdot X[k] \quad (2.70)$$

Zur Sicherstellung eines reelen Spektrums nach der Transformation kann die Symmetrieeigenschaft der Fourier-Transformation genutzt werden. Ist die

Zeitfunktion gerade, d.h.  $x[n] = x[-n]$ , ergeben sich alle komplexen Anteile zu Null. Um dieses Verhalten auch bei ungeraden Funktionen zu erzwingen, wird die Folge an der Y-Achse gespiegelt. Nach der Multiplikation mit dem komplexen Verschiebungsterm und der Rücktransformation in den Zeitbereich können die gespiegelten Folgenelemente verworfen werden. Abb. 2.25 zeigt ein Flussdiagramm des implementierten Algorithmus.



**Abb. 2.25:** Algorithmus zur zeitlichen Verschiebung von N Signalen

### 2.5.3 Kreuzkorrelation unter Verwendung der FFT

Der Ablauf zur Berechnung der  $\frac{N^2-N}{2}$  Kreuzkorrelationsfunktionen sowie der  $N$  Varianzen ist in Abb. 2.26 dargestellt. Im Vergleich zur beschriebenen Mathematik in Abschnitt 2.2.2 werden die Sensorsignale vor der Korrelation nicht entsprechend einer Schalleinfallsrichtung zeitlich zueinander ausgerichtet. Grund hierfür ist der enorme Rechenaufwand der dadurch entstehen würde. Der Berechnungsvorgang der Kreuzkorrelation müsste in diesem Fall entsprechend der angegebenen Anzahl von Richtungen  $S_{\phi,\theta}$  in Tab. 2.5 wiederholt werden, wodurch sich eine Echtzeitimplementierung als schwierig erweist. Da sich die Zeitverzögerung zwischen einem Signal und dessen verschobener Kopie in der Position der Kreuzkorrelationsfunktionen auf dessen Zeitachse widerspiegelt, kann die Verschiebung der Zeitsignale vor der Korrelation oder die Verschiebung des Korrelogramms nach der Korrelation um jeweils die selbe Zeit als äquivalent betrachtet werden. Unter Anwendung der Korrelogrammverschiebung kann Rechenzeit eingespart werden, da die oben angegebene Anzahl von Korrelationsoperationen nur einmal pro Signalblock berechnet werden muss.

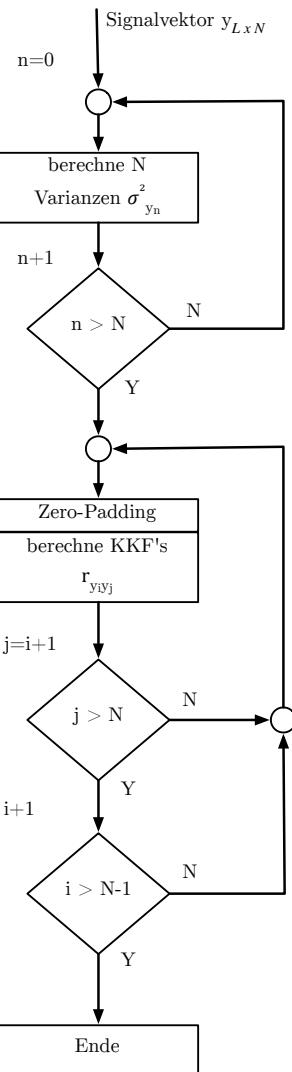
### 2.5.4 Algorithmus zur Berechnung des MCCC

Abb. 2.27 zeigt den Ablauf zur Berechnung des Mehrkanal-Kreuzkorrelationskoeffizienten. Zu Beginn wird abhängig vom Azimuth- und Elevationswinkel die räumliche Korrelationsmatrix  $\mathbf{R}_{\phi,\theta}$  gebildet. Wie bereits in Abschnitt 2.5.3 erwähnt, erfolgt dies durch Korrelogrammverschiebung unter Verwendung von  $f_n(\phi, \theta)$  mit  $\mathbf{R}_{\phi,\theta} =$

$$\begin{bmatrix} \sigma_{y_0}^2 & r_{y_0y_1}(k + f_1(\phi, \theta)) & \dots & r_{y_0y_N}(k + f_N(\phi, \theta)) \\ r_{y_0y_1}(k + f_1(\phi, \theta)) & \sigma_{y_1}^2 & \dots & r_{y_1y_N}(k + f_N(\phi, \theta) - f_1(\phi, \theta)) \\ \vdots & \vdots & \ddots & \vdots \\ r_{y_0y_N}(k + f_N(\phi, \theta)) & r_{y_1y_N}(k + f_N(\phi, \theta) - f_1(\phi, \theta)) & \dots & \sigma_{y_N}^2 \end{bmatrix} \quad (2.71)$$

Anschließend erfolgt die Berechnung der Determinante, wobei dessen Wert in einem separaten zweidimensionalen Feld  $\det_{\phi,\theta}$  abgelegt wird. Dieser Vorgang wird entsprechend den Angaben in Tab. 2.5 für jede messbare Richtung wiederholt. Abschließend wird das Minimum von  $\det_{\phi,\theta}$  ermittelt und der entsprechenden Richtung zugeordnet.

Abb. 2.28 zeigt das Ergebnis der MCCC-Simulation mit Verwendung eines synthetisch verzögerten Sprachsignals bei zwei unterschiedlichen Schallein-

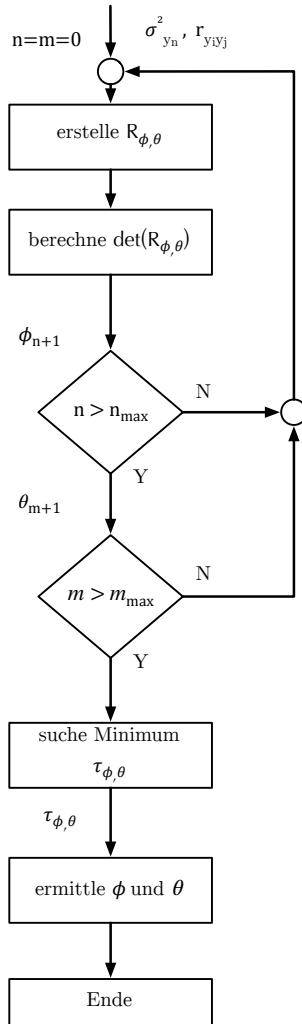


**Abb. 2.26:** Algorithmus zur Berechnung der nötigen Kreuzkorrelationsfunktionen  $r_{y_i, y_j}$  sowie Varianzen  $\sigma^2_{y_n}$

fallsrichtungen sowie SNR<sup>14</sup>. Das Sprachsignal wurde dabei mit unkorreliertem weißen Rauschen entsprechend angegebenem SNR überlagert. Zu erkennen ist, dass die Richtung auch bei geringem SNR gut detektiert werden kann.

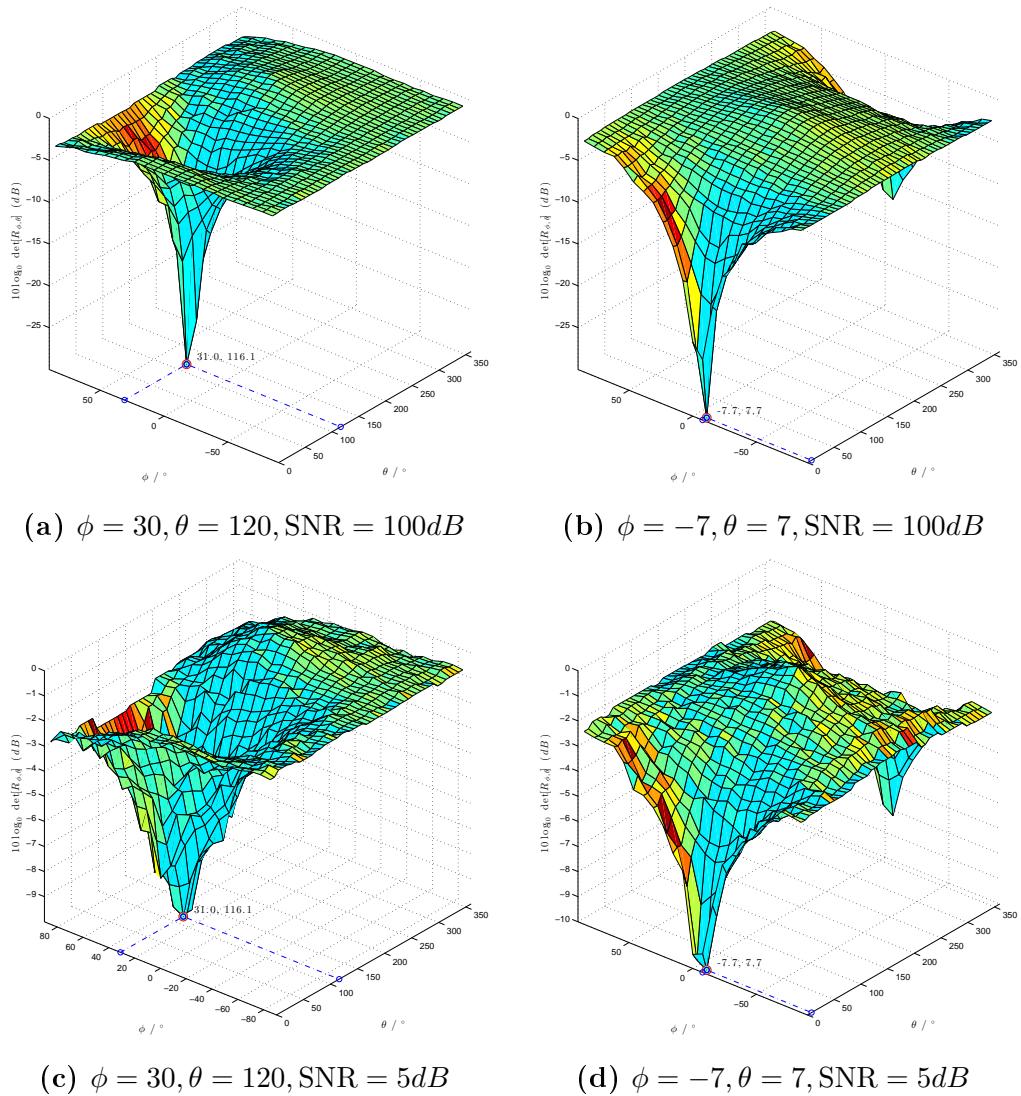
Zur Untersuchung des Korrelationsverfahrens auf die Entstehung des räumlichen Aliasing-Effekts wird die Reaktion des Systems auf ein rein periodi-

<sup>14</sup>Signal-Rausch-Verhältnis

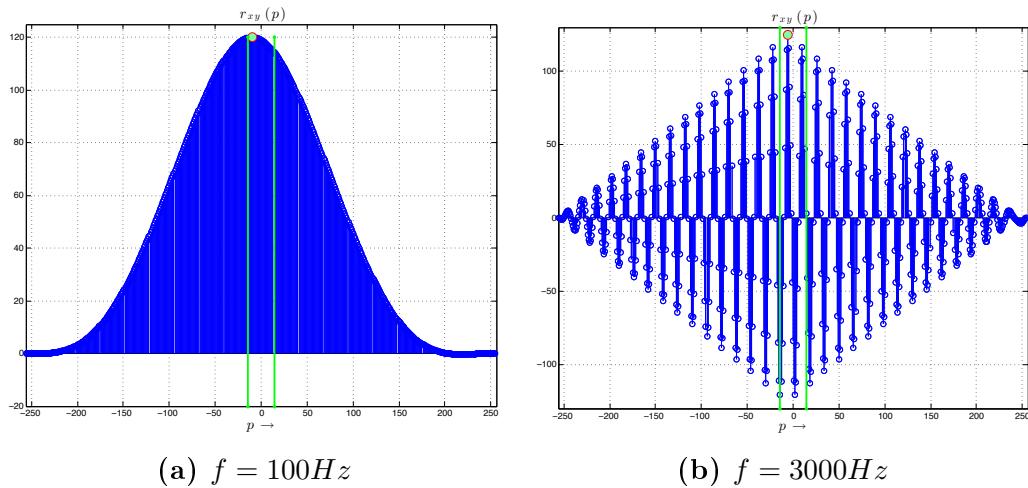


**Abb. 2.27:** Algorithmus zur Berechnung des Mehrkanal-Kreuzkorrelationskoeffizienten

sches Signal der Form  $y[n] = \sin\left(2\pi\frac{f}{f_a}n\right)$  bei unterschiedlichen Frequenzen beobachtet. Abb. 2.30 illustriert sechs Ergebnisse der MCCC im Frequenzintervall  $100Hz \leq f \leq 3kHz$ . Zu sehen ist, dass sich die Ausprägung des Minimums bei steigender Frequenz deutlich verbessert. Grund hierfür ist die immer schmäler werdende Hauptkeule der Kreuzkorrelationsfunktion wie in Abb. 2.29 dargestellt. Weiterhin treten in Abb. 2.30 zusätzliche lokale Minima auf, deren Anzahl und Abstand von der Menge sowie Höhe der Nebenkeulen im Suchbereich (zwischen den grünen Linien in Abb. 2.29) abhängt. Bei



**Abb. 2.28:** Ergebnis des MCCC unter Verwendung eines synthetisch verzögerten Sprachsignals bei zwei unterschiedlichen Schalleinfallrichtungen sowie SNR und einer Abtastfrequenz von  $f_a = 44,1\text{kHz}$ .



**Abb. 2.29:** Ergebnis der Kreuzkorrelationen unter Verwendung eines synthetisch verzögerten Sinussignals bei einer Abtastfrequenz von  $f_a = 48kHz$ .

Verletzten des Kriteriums in Gleichung 2.16 befinden sich mehrere Spalten innerhalb des Suchbereichs der KKF, was die Lokalisierung des korrekten Maximums erschwert. Auf Grund der hier verwendeten nicht erwartungstreuen Schätzung (siehe Abschnitt 2.2.1) erfährt die Kreuzkorrelation eine Gewichtung durch das Dreieckfenster, so dass die Nebenmaxima gedämpft werden und die Wahrscheinlichkeit der Entstehung von räumlichem Aliasing deutlich verringert wird.

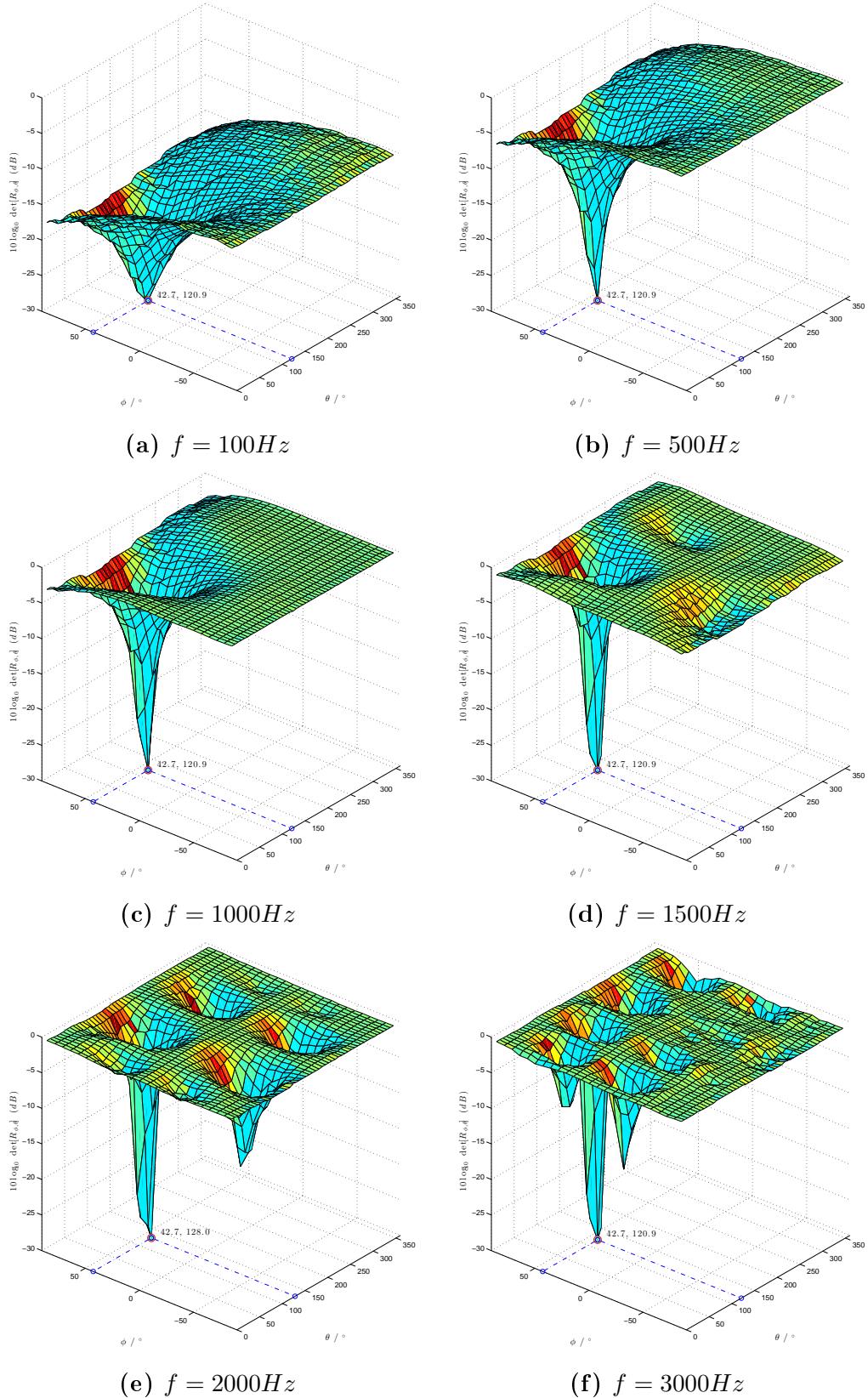
## Bewegte Sprachquelle

Um das Verhalten des Systems auf eine sich durch den Raum bewegende Sprachquelle zu untersuchen, werden synthetische Signale aus zwei folgenden Signaltypen erstellt:

- bandbegrenztes weißes Rauschen
  - Sprache (männlich und weiblich)

Folgende Simulationsparameter wurden verwendet:

- Anzahl der Mikrofone  $N = 8$
  - Abtastfrequenz  $f_a = 44,1\text{kHz}$
  - Signalblocklnge  $L = 256$  (Frame)

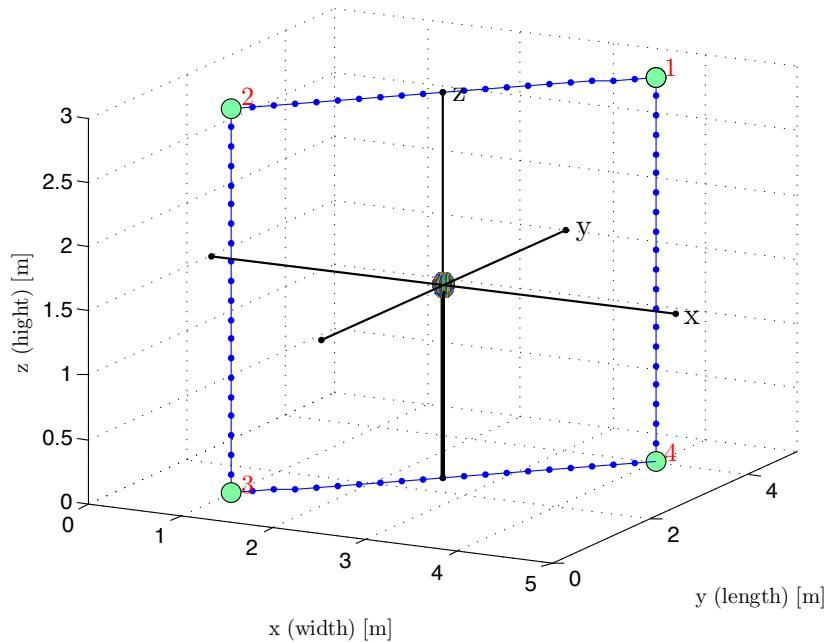


**Abb. 2.30:** Ergebnis des MCCC unter Verwendung eines synthetisch verzögerten periodischen Signals der Form  $y(n) = \sin\left(2\pi \frac{f}{f_a} n\right)$  bei einer Abtastfrequenz von  $f_a = 48kHz$ .

- FFT-Länge  $K = 2 \cdot L = 512$
- Winkelauflösung  $\alpha_{\phi,\theta} = 7,7$

Abb. 2.31 zeigt einen künstlichen reflexionsfreien Raum mit den Abmaßen  $5m \times 5m \times 3m$ , in dessen Zentrum das Mikrofonarray positioniert ist. Die Rauschquelle bewegt sich hier diagonal und nimmt dabei Winkel im Bereich von  $-90 \leq \phi \leq 90$  und  $\theta \in \{45, 225\}$  an. In diesem Versuch werden zunächst die vier Eckpunkte platziert und anschließend mit einer Anzahl von 81 Stützstellen linear interpoliert. Die Rauschquelle bewegt sich somit entlang der Stützstellen um eine kleine Winkelschrittweite zu realisieren. Abb. 2.32 zeigt das Ergebnis dieses Simulationsdurchlaufs. Die in rot abgebildete Linie stellt den Fehler zu den tatsächlich eingenommene Winkeln dar. Alle Signalblöcke mit einer Blockenergie von weniger als der eingezeichneten Schwelle werden mit einem Wert 0 detektiert. Da wie bereits erwähnt nur diskrete Werte eingenommen werden können ist ein Fehler von  $\pm \frac{\alpha_{\phi,\theta}}{2}$  zu tolerieren. Beim Betrachten des Azimuthwinkelverlaufs fällt auf, dass an den Stellen, an denen der Elevationswinkel  $\phi = 90$  einnimmt ein plötzlicher Fehleranstieg stattfindet. Der Grund hierfür ist, dass sich die Quelle genau im oberen Winkeltotpunkt befindet und eine Drehung auf den Azimuthwinkel nicht erkannt werden kann.

Im Folgenden soll nun eine bewegte Sprachquelle simuliert werden. Wie in Abb. 2.33 dargestellt, bewegt sie sich sägezahnförmig um das Mikrofonarray und nimmt dabei Winkel im Bereich von  $-45 \leq \phi \leq 45$  und  $0 \leq \theta < 360$  an. Als Grundlage wird eine Audiodatei der Länge  $T_{file} = 16s$  verwendet. Bei einem zurückgelegten Weg von  $s = 28,9m$  beträgt die Durchschnittsgeschwindigkeit der Quelle  $v = \frac{s}{T_{file}} = 6,5 \frac{km}{h}$  was der allgemeinen Angabe zur Schrittgeschwindigkeit nahe kommt. Abb. 2.34 und Abb. 2.35 illustrieren die Ergebnisse der beiden Simulationsdurchläufe. Zu beachten ist, dass die Signale synthetisch mit dem in Abschnitt 2.5.2 beschriebenen Verfahren (ohne Reflexion) blockweise verzögert werden. Die Phasenverschiebung zwischen den Kanälen ändert sich somit sprunghaft und nicht kontinuierlich wie in realer Umgebung. Die fehlerhaft erkannten Winkel (Winkelsprünge) sind zum großen Teil auf diese plötzliche Phasenänderung zurückzuführen. Die in rot abgebildete Linie stellt die Referenz zu den tatsächlich eingenommenen Winkeln dar. Alle Signalblöcke mit einer Blockenergie von weniger als der in rot dargestellten Schwelle werden mit dem Wert 0 detektiert. Beim Vergleich der beiden Ergebnisse zeigt sich, dass die Zuverlässigkeit der Winkelkennung deutlich voneinander abweicht. Dies lässt sich zum einen durch das unterschiedliche Frequenzspektrum zwischen männlicher und weiblicher Sprache begründen, zum anderen liegt eine große Differenz im Dynamikbe-



**Abb. 2.31:** Wegstrecke einer Rauschquelle die sich diagonal durch den Raum bewegt

reich<sup>15</sup> vor. Im Vergleich zur weiblichen Sprecherin ist der männliche etwa doppelt so groß. Die Blockenergieschwelle wird für die Simulation abhängig vom Dynamikbereich etwa in dessen Mittel gelegt. Die Ergebnisse erweisen sich trotz gelegentlicher Winkelsprünge als zufriedenstellend da die Bewegung der Quelle gut verfolgbar ist.

## 2.5.5 Optimierungsverfahren

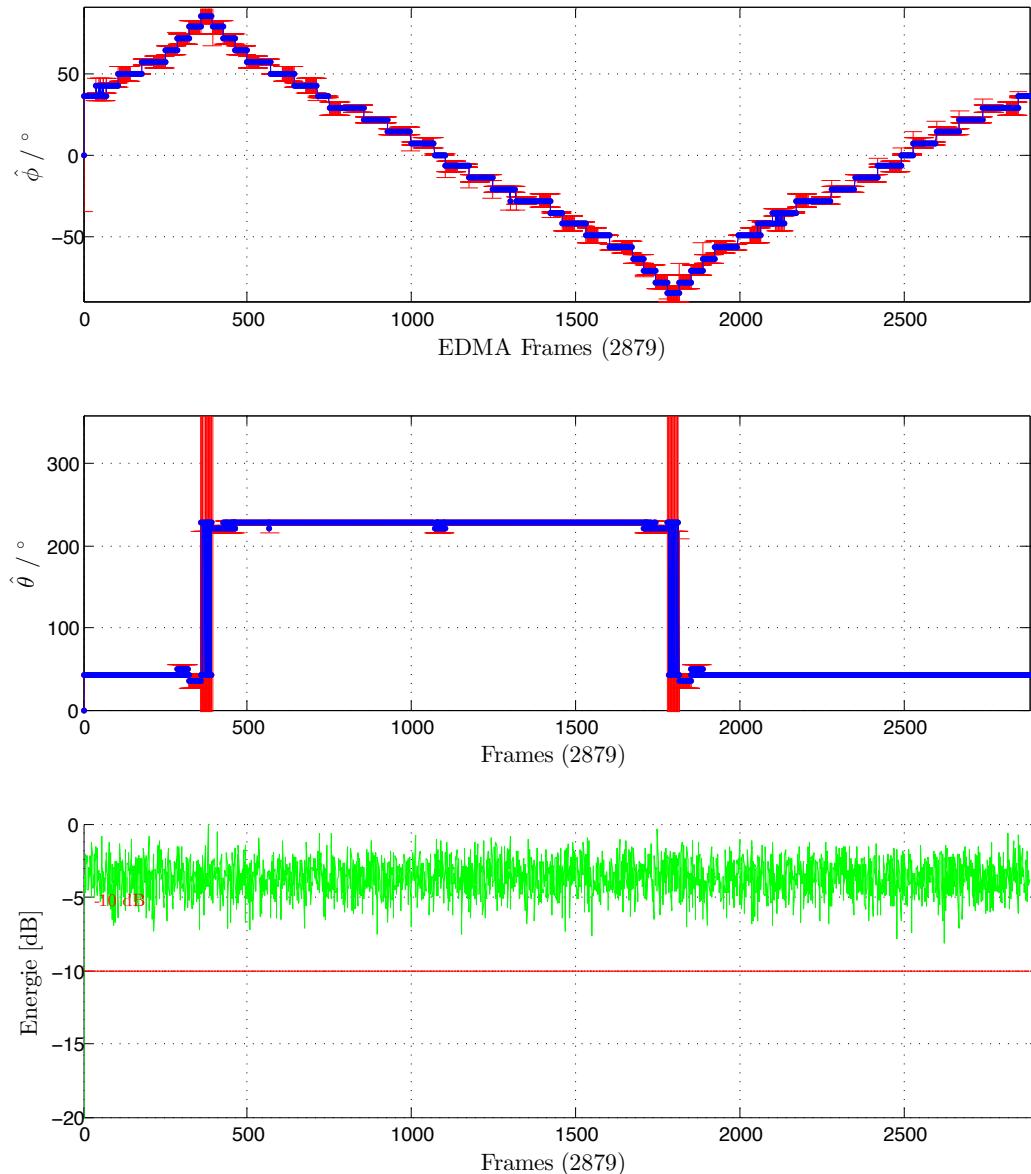
Im Hinblick auf die spätere Echtzeitimplementierung auf einem DSP gilt es, den Rechenaufwand so gering wie möglich zu halten. Auf Grund dessen wird ein Verfahren zur Optimierung der Suchgeschwindigkeit entwickelt. Als zweiter Optimierungsschritt folgt eine Histogrammschätzung der Richtungsergebnisse. Dieser vermindert den Effekt der abrupten Winkelsprünge.

### Suchgeschwindigkeit

Zur Verbesserung der Suchgeschwindigkeit wird eine Methode entwickelt, die sich mit Hilfe von variablen Schrittweiten sukzessive an den korrekten Winkel

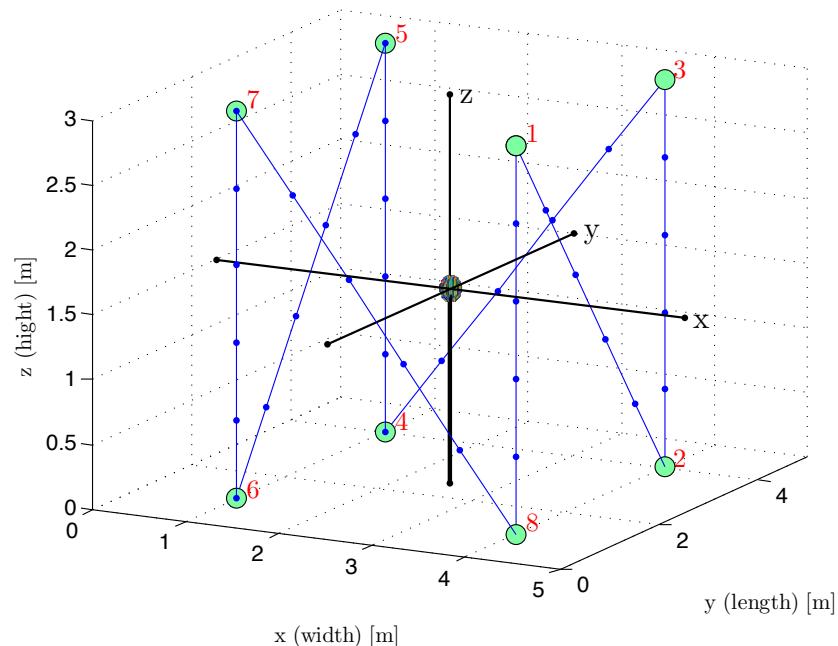
---

<sup>15</sup>Bereich zwischen größter und kleinster Signalstärke



**Abb. 2.32:** Simulationsergebnis einer Rauschquelle bei einer Blockenergieschwelle von -10dB.

annähert. Abb. 2.36 zeigt das zu Grunde liegende Prinzip. Der abzusuchende Winkelbereich wird zunächst in drei Gebiete geteilt. Nun wird an den Bereichsübergängen nach dem Minimum in der räumlichen Korrelationsmatrix  $R_{\phi,\theta}$  gesucht. Der gefundenen Winkel wird in der nächsten Iteration als neuer Startpunkt gewählt. Anschließend erfolgt eine Halbierung des Suchbe-

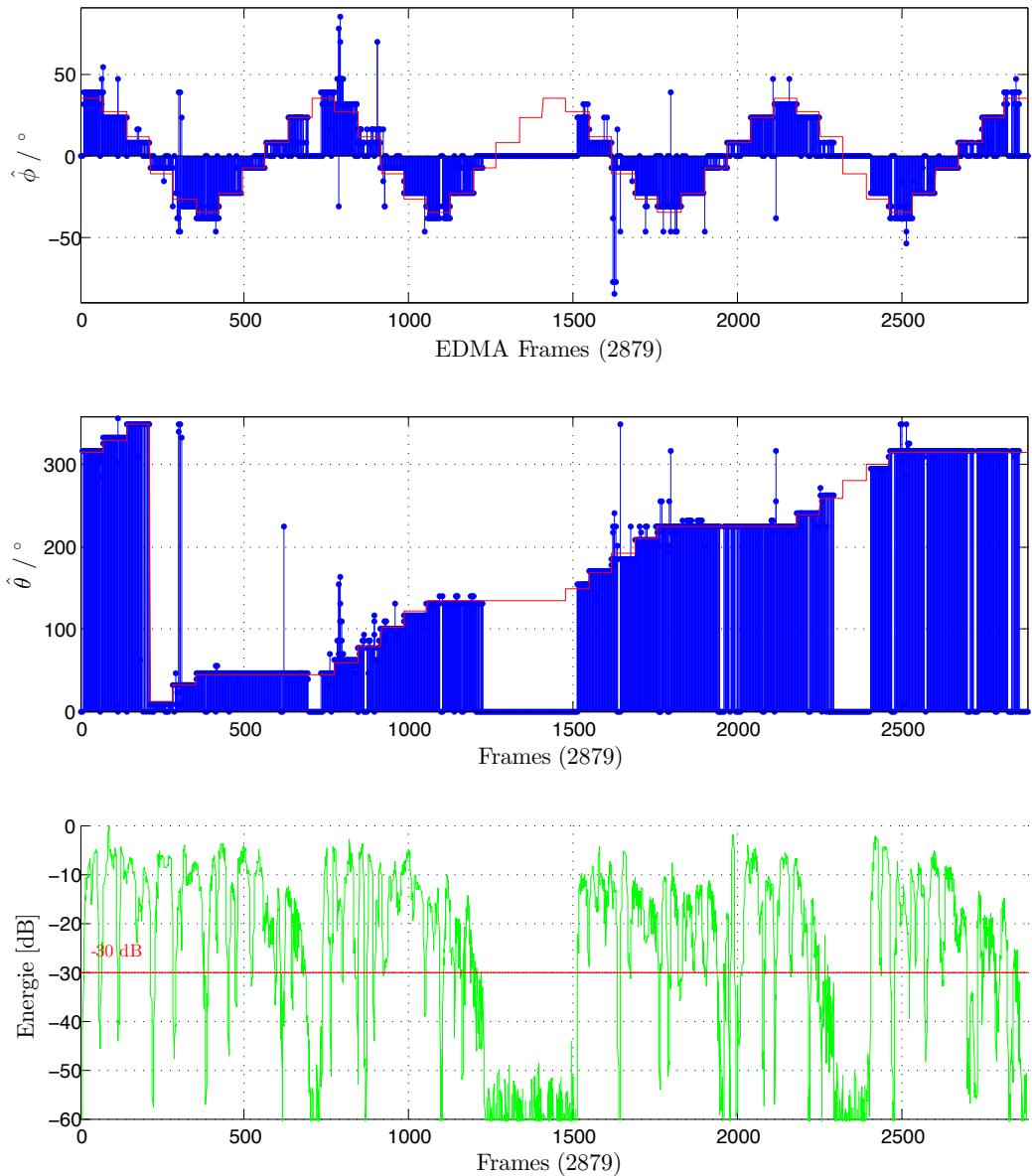


**Abb. 2.33:** Wegstrecke einer männlichen Sprechquelle die sich Zig-Zag-förmig von Position 1 bis 9 um das Mikrofonarray bewegt.

reiches und der Vorgang beginnt erneut. Dieser Algorithmus wird so lange wiederholt, bis die minimale Schrittweite erreicht wurde.

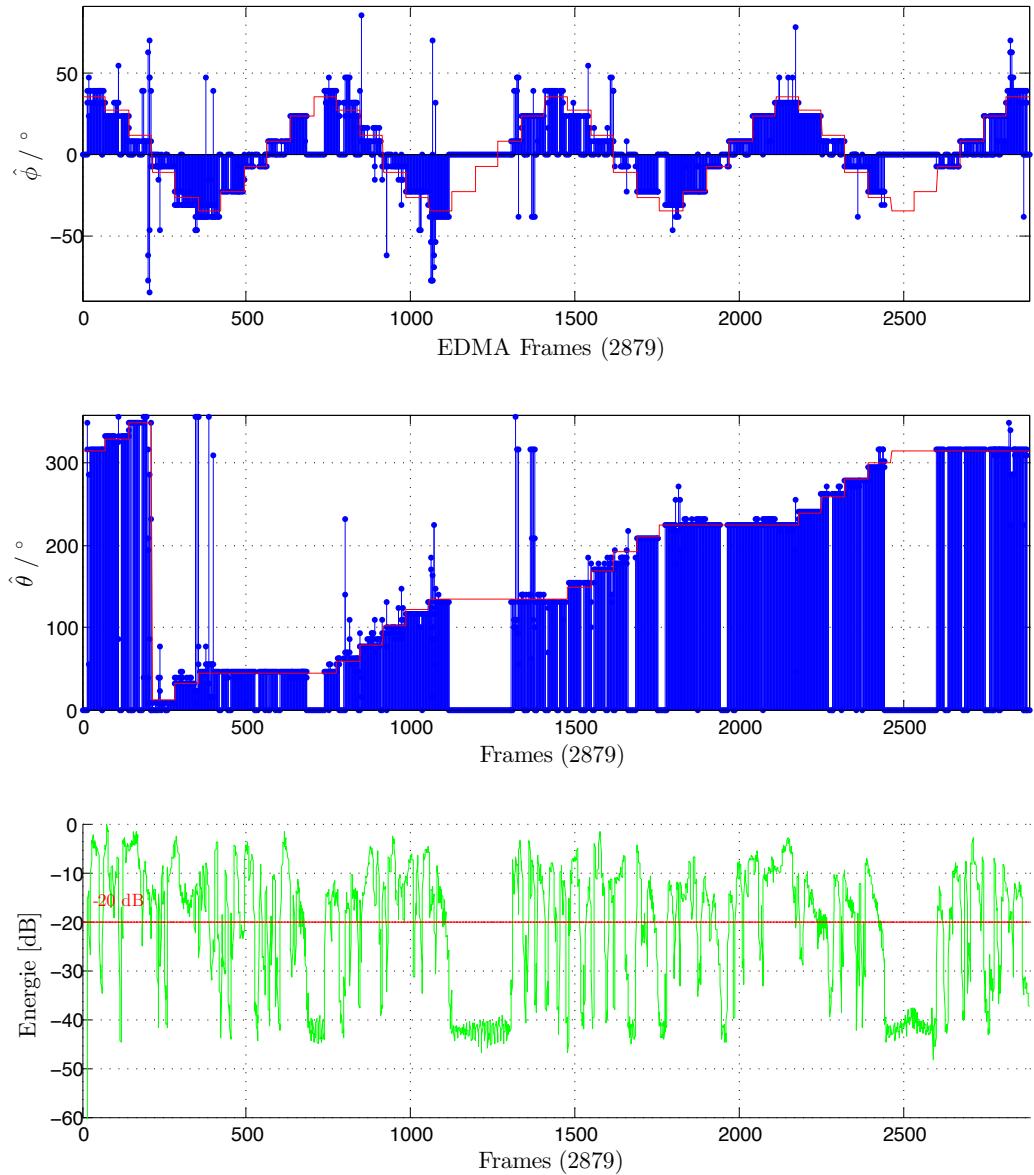
### Histogramm

Zur Reduktion der Winkelsprünge bei z.B. überlagerten Störungen wird im Folgenden eine Histogrammschätzung durchgeführt. Die detektierten Winkel werden im laufenden Betrieb kontinuierlich in einen Ringspeicher mit einer variabel einstellbaren Länge gespeichert. Nach jedem Suchvorgang erfolgt eine Häufigkeitsanalyse des Ringspeicherinhalts. Abb. 2.37 zeigt illustriert ein Beispiel für die Häufigkeitsanalyse des weiblichen Sprachsignals mit einer Ringspeicherlänge von 50 Werten sowie einer Häufigkeitsschwelle von 20%. Tritt ein gemessener Winkel so häufig auf, dass er diese Schwelle überschreitet wird er als wahrscheinlich genug eingestuft und als gültig markiert. Tritt der Fall ein, dass die Häufigkeit keines Ringspeicherwerts diese Schwelle überschreitet, markiert der Algorithmus den aktuellen Signalblock als ungültig. Abb. 2.38 und Abb. 2.39 zeigen einen Vergleich der Simulationsergebnisse des männlichen und weiblichen Sprechers mit und ohne Histogrammoptimierung. Es ist deutlich zu sehen, dass abrupte Winkelsprünge nahezu verschwi-

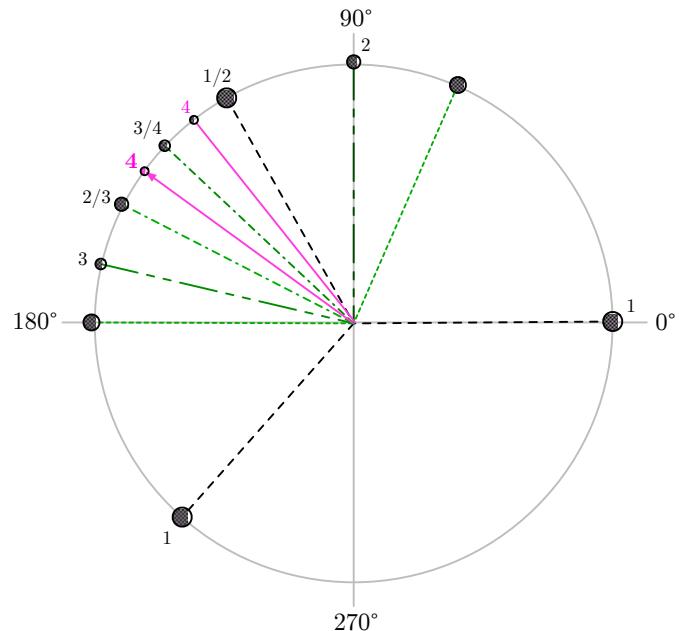


**Abb. 2.34:** Simulationsergebnis eines bewegten männlichen Sprechers (deutschsprachig) bei einer Blockenergieschwelle von -30dB.

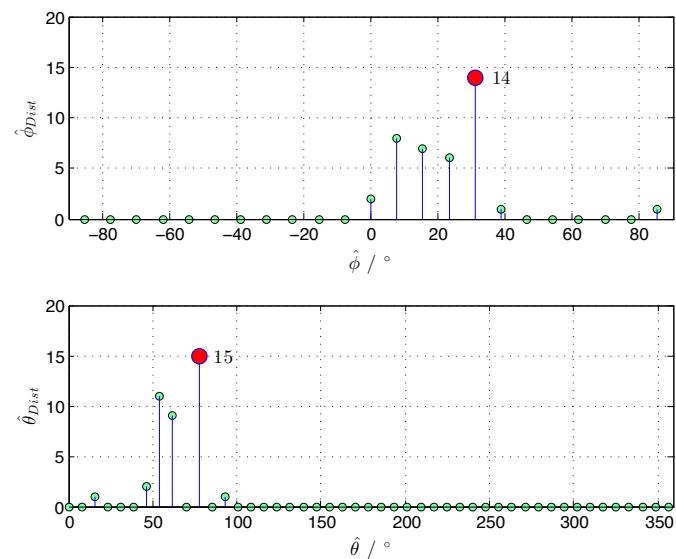
den. Bei solch einem Verfahren, das Ergebniswerte zwischenspeichert, tritt eine Verzögerung auf, die abhängig von der Ringspeicherlänge sowie der eben erwähnten Schwelle ist. Im Vergleich zum nicht optimierten Ergebnis ist das optimierte leicht zur Winkelreferenzkurve verschoben.



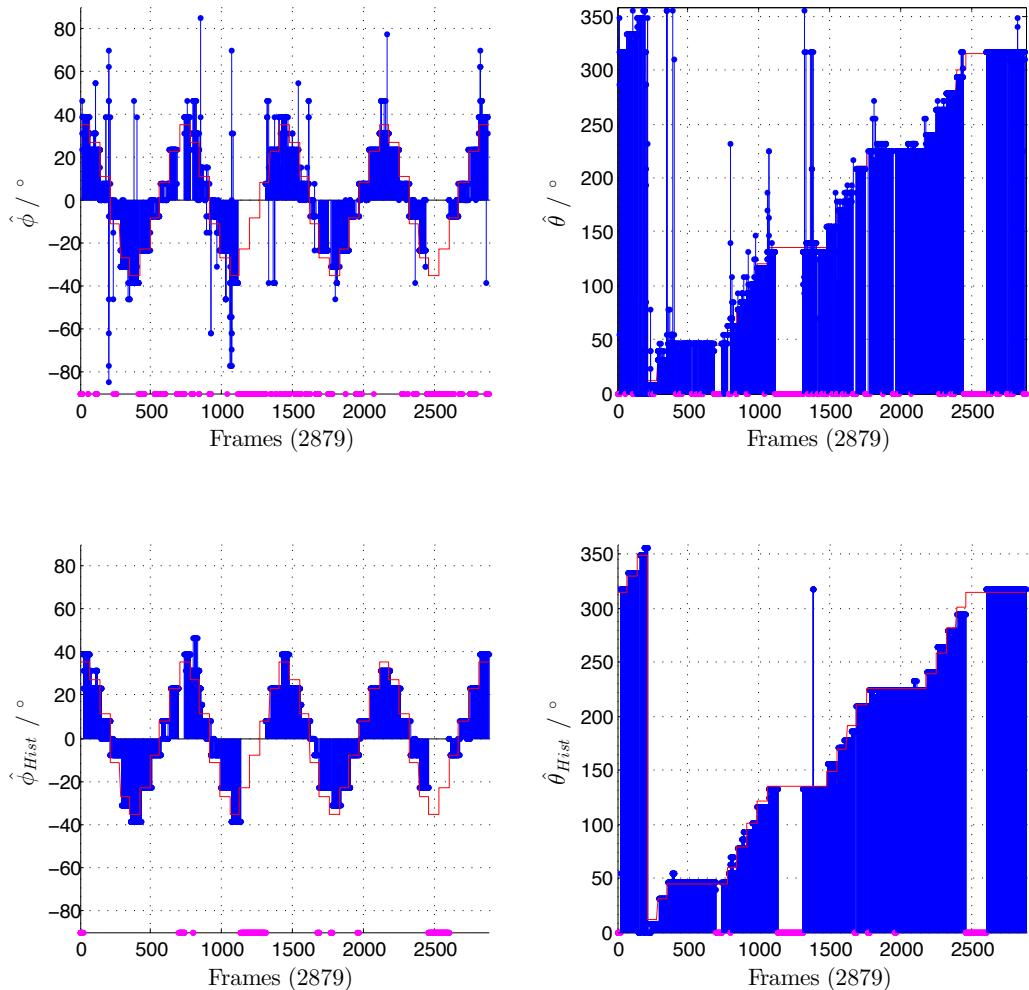
**Abb. 2.35:** Simulationsergebnis eines bewegten weiblichen Sprechers (englischsprachig) bei einer Blockenergieschwelle von -20dB.



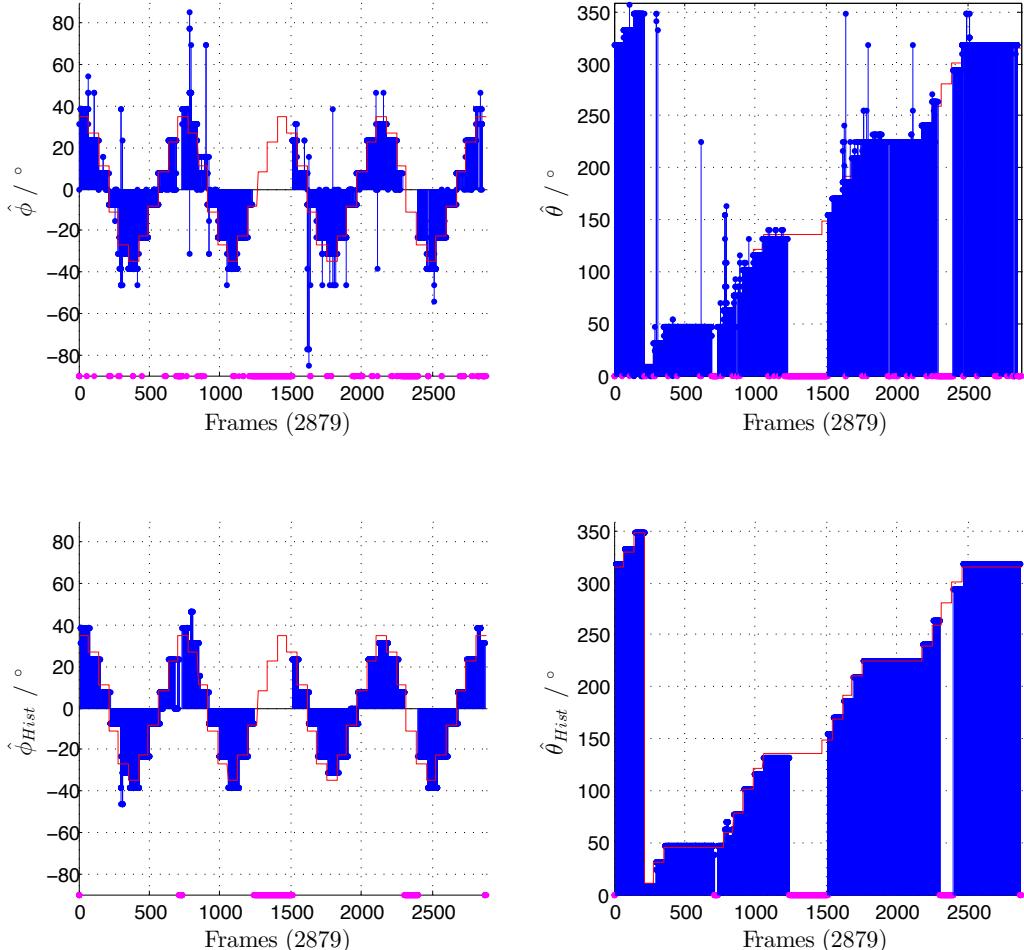
**Abb. 2.36:** Methode zur Optimierung der Suchgeschwindigkeit mit Hilfe einer variablen Schrittweiten.



**Abb. 2.37:** Histogramm eines bewegten weiblichen Sprechers



**Abb. 2.38:** Vergleich zwischen Winkelverlauf mit und ohne Histogrammschätzung beim weiblichen Sprecher mit einer Ringspeicherlänge von 50 und einer Schwelle von 15%



**Abb. 2.39:** Vergleich zwischen Winkelverlauf mit und ohne Histogrammschätzung beim männlichen Sprecher mit einer Ringspeicherlänge von 50 und einer Schwelle von 15%

# Kapitel 3

## Realisierung

In diesem Kapitel folgt die Beschreibung der Realisierung des im Kapitel 2 vorgestellten Systemkonzepts. Zunächst werden, die Hardware sowie die damit verbundenen Einschränkungen im Vergleich zur Simulation in MATLAB® erläutert. Anschließend erfolgt die Darstellung der Programmumsetzung mit der Programmiersprache C.

### 3.1 Verwendete Hardware

Zur Implementierung des 3D-Tracking-Algorithmus ist eine Hardware notwendig, die über ausreichend Peripherie zum Anschluss der nötigen Anzahl von Mikrofonen verfügt sowie über genügend Rechenleistung um alle Rechenschritte in der vorgegebenen Zeit (siehe Tab. 2.7) durchführen zu können. Abb. 3.1 zeigt eine detaillierte schematische Darstellung der verwendeten Systemkomponenten. Diese gliedern sich in drei Kategorien, die im Folgenden vorgestellt werden.

1. Schallwandler Front-End
2. DSP Back-End
3. Display (PC)

#### 3.1.1 Schallwandler Front-End

Die erste Stufe des Echtzeitsystems beinhaltet das Mikrofonarray (siehe Abb. 2.21), das die akustischen in elektrische Signale wandelt. Bei den Schallwandlern handelt es sich um omnidirektionale<sup>1</sup> Kondensatormikrofone der Firma

---

<sup>1</sup>Druckempfänger mit einer Kugelcharakteristik

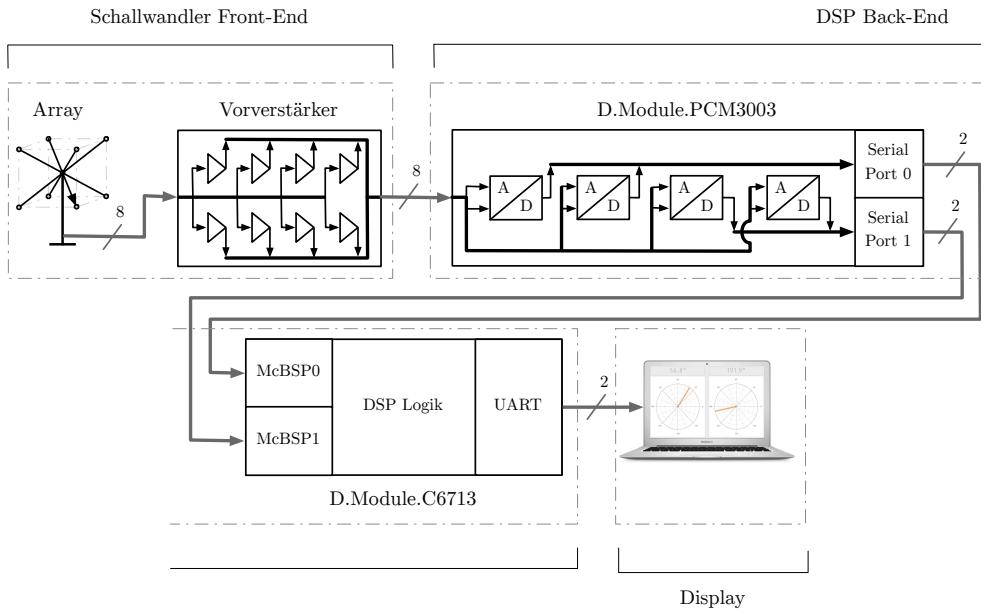
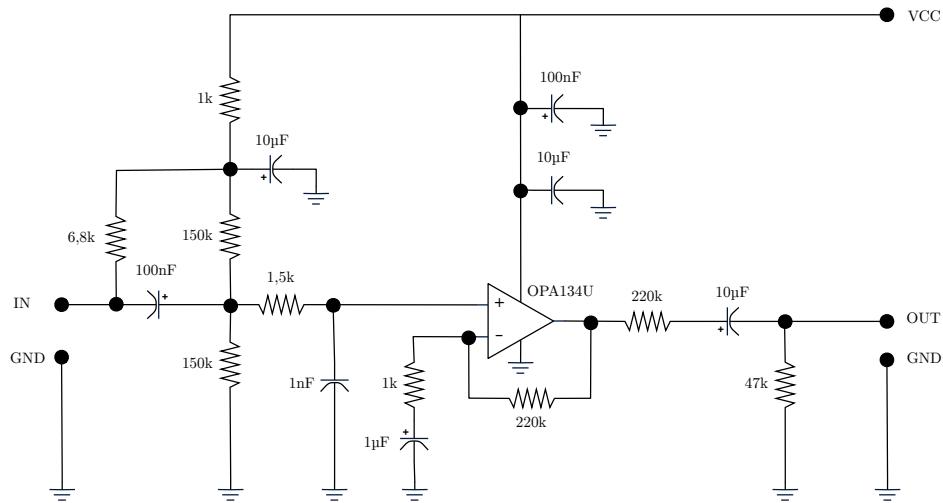


Abb. 3.1: Blockdiagramm der verwendeten Hardware

Sennheiser®. Auf Grund des niedrigen Signalpegels, den ein Mikrofon ausgibt, muss eine Verstärkungsstufe zwischen Array und ADC<sup>2</sup> geschaltet werden. Um die Bittiefe des Wandlers vollständig ausnutzen zu können gilt es, das Mikrofonsignal bis auf eine Spannung von 1,8Vpp<sup>3</sup> zu verstärken. Abb. 3.2 zeigt den Stromlaufplan der Verstärkerschaltung. Als Basiskomponente wird ein Operationsverstärker eingesetzt, der als nicht-invertierter OPV beschaltet ist. Wie in Abb. 3.2 ersichtlich wird ein Teil der Betriebsspannung auf den Signaleingang gekoppelt. Grund hierfür ist, dass das Eingangssignal während der Aufnahme positive sowie negative Werte annehmen kann. Da lediglich eine positive Betriebsspannung zur Verfügung steht, ist der OPV nur in der Lage, den Wert der Betriebsspannung als Maximalwert und den Wert Null als Minimalwert anzunehmen. Die auf den Signaleingang gekoppelte Gleichspannung wirkt hier wie ein Offset um den Signalruhepegel in die Mitte des zur Verfügung stehenden Spannungsbereiches zu verschieben. Auf Grund dessen lassen sich positive sowie negative Signalverläufe verstärken und so optimal auf den Spannungsbereich des ADC abstimmen.

<sup>2</sup>Analog/Digital-Converter

<sup>3</sup>Spitze-Spitze-Wert



**Abb. 3.2:** Stromlaufplan des Mikrofonvorverstärkers für einen Kanal

### 3.1.2 DSP Back-End

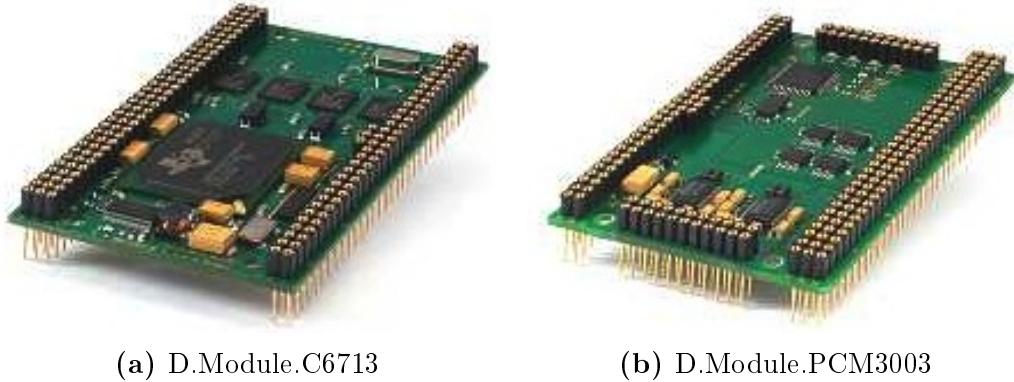
Wie in Abb. 3.1 dargestellt, besteht die zweite Signalverarbeitungsstufe aus der I/O<sup>4</sup>-Flachbaugruppe D.Module.PCM3003 sowie aus der DSP-Flachbaugruppe D.Module.C6713 der Firma D.SignT und bildet so das DSP Back-End. Abb. 3.3 zeigt beide Module im Zusammenbau mit jeweils acht Anschlüssen für ADC/DAC-Module.

Das D.Module.PCM3003 Modul ist ein Audiocodec, der speziell für Mehrkanal-Audio Signalverarbeitung sowie Mikrofonarrayverarbeitung designed wurde [3, S. 1]. Abb. 3.4 zeigt alle Funktionsblöcke in einem Blockdiagramm. Der Codec verfügt über vier Stereo-ADC/DAC Module und kann so im Vierkanal-Stereo-Mode oder im Achtkanal-Mono-Mode betrieben werden. Da in dieser Arbeit acht Mikrofonsignale simultan verarbeitet werden sollen, wird der Achtkanal-Mono-Mode gewählt. Jedes der vier ADC/DAC-Module besitzt eine Auflösung von 16 bit und einen Delta-Sigma-Modulator 64. Ordnung. Alle ADC/DAC-Sektionen werden synchrone von einem einstellbaren Oszillator getaktet und können auf die im Audiobereich gängigen Abtastraten konfiguriert werden. Die Übertragung der Abtastwerte an den DSP erfolgt über zwei serielle Schnittstellen wobei jeweils zwei ADC/DAC-Stufen zu einer Schnittstelle führen. Alle notwendigen Synchronisationssignale werden von diesem Modul erzeugt, so dass es als Mastergerät fungiert.

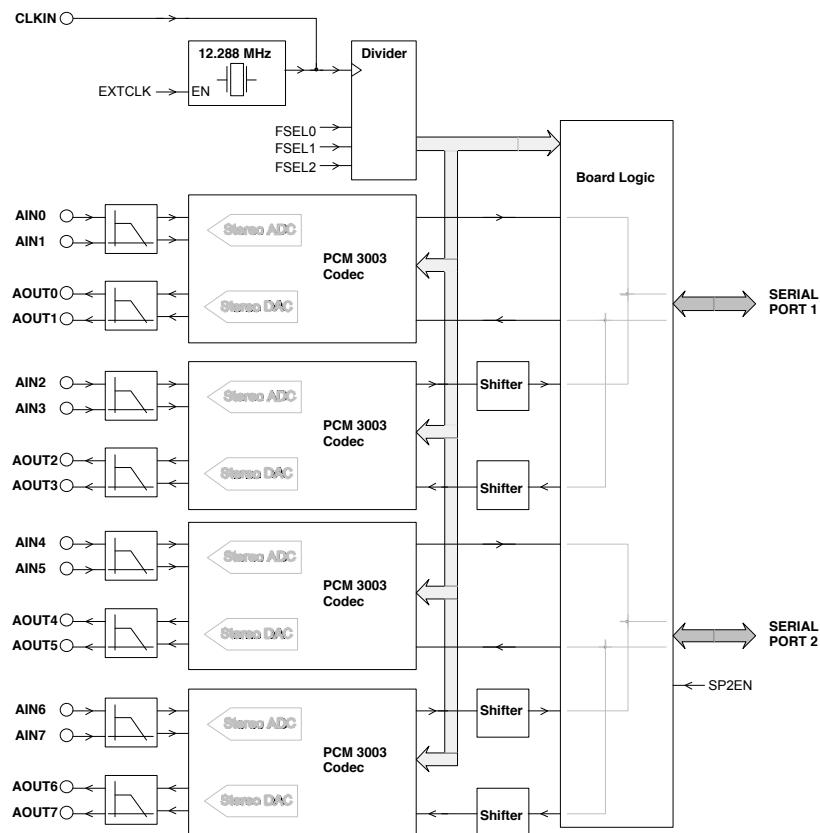
Das D.Module.C6713 Modul ist eine DSP Flachbaugruppe, dessen CPU<sup>5</sup>

<sup>4</sup>Input/Output

<sup>5</sup>Central processing unit



**Abb. 3.3:** DSP-Flachbaugruppe D.Module.C6713 und D.Module.PCM3003



**Abb. 3.4:** Blockdiagramm des D.SignT Moduls D.Module.PCM3003 (vgl. [3, S. 1])

der von Texas Instruments<sup>®</sup> entwickelte TMS320C6713b aus der Prozessorreihe TMS320C6000 ist. Dieser DSP basiert auf einer Very Long Instruction Word Architektur mit einer Wortbreite von 256 bit und ist so in der Lage, bis zu acht 32-bit Befehle für jeweils einen der acht Funktionseinheiten innerhalb eines Taktschrittes zu verarbeiten [17, S. 14]. Weiterhin kann der TMS320C6713b sowohl in Festkomma- als auch in Fließkomma-Arithmetik programmiert werden und unterstützt einen Prozessortakt von bis zu 300MHz wodurch eine Rechengeschwindigkeit von 1,8 GFLOPS<sup>6</sup> erreicht werden kann. Abb. 3.6 zeigt das Blockdiagramm mit der zur Verfügung stehenden Peripherie. Als Schnittstellen stehen unter anderem zwei Multi-channel-Buffered Serial Port ( McBSP ) und ein I2C Interface zur Verfügung, die mit Hilfe eines Enhanced Direct Memory Access ( EDMA ) Controller mit dem internen Speicher verknüpft sind. Die DSP-Flachbaugruppe bietet darüber hinaus noch folgende Funktionen:

- 64 MB SDRAM<sup>7</sup> mit 100MHz Taktfrequenz
- 2MB Flash Speicher
- 256 kB Interner Speicher (für Programmcode sowie vom Programm benötigter Speicher)
- USB Schnittstelle zum PC inkl. JTAG<sup>8</sup> Emulator
- LED's sowie Taster zur Benutzerinteraktion

Als Programmierwerkzeug wird die von Texas Instruments<sup>®</sup> entwickelte Softwarelösung Code Composer<sup>TM</sup> Studio verwendet. Dieses Eclipse-basierte Softwarepaket beinhaltet einen Sourcecode-Editor, eine Projektumgebung, eine Debug-Umgebung sowie die Möglichkeit Variable im laufenden Betrieb zu observieren und Speicherinhalte grafisch darzustellen. Code Composer<sup>TM</sup> Studio verfügt über einen C-Compiler, einen Assembler sowie einen Linker und wird in dieser Arbeit in Version 5 verwendet.

### 3.1.3 Kommunikationsschnittstelle zwischen DSP und PCM3003

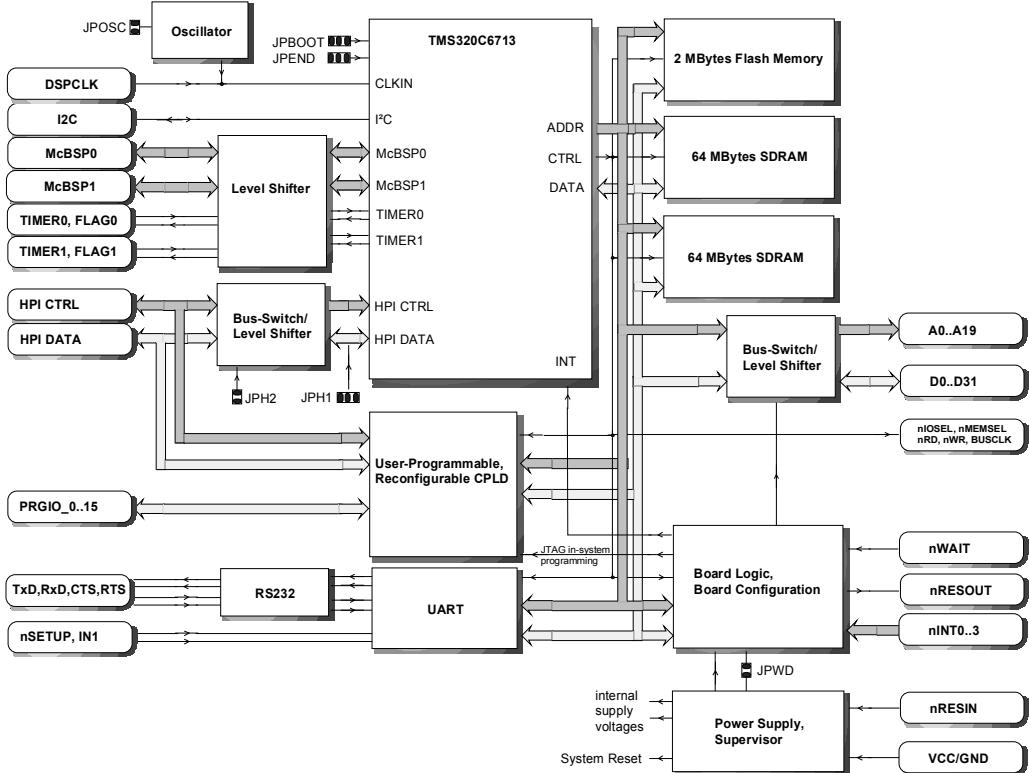
Als Kommunikationsschnittstelle zwischen DSP und PCM3003 dienen die vom DSP bereitgestellten McBSP. Der Datenaustausch findet so jeweils über

---

<sup>6</sup>Giga-floating-point operations per second

<sup>7</sup>Synchronous Dynamic Random Access Memory

<sup>8</sup>Joint Test Action Group



**Abb. 3.5:** Blockdiagramm des D.SignT DSP D.Module.C6713 (vgl. [4, S. 1])

zwei Datenleitungen statt, wobei eine die eingehenden und die andere die ausgehenden Daten enthält. Steuersignale wie Synchronisation und Taktung werden über eine separate Leitung übertragen. Abb. 3.7 illustriert das Funktionsblockdiagramm eines McBSP Moduls. Eine detaillierte Erklärung zu den Funktionen der einzelnen Pins und Register befindet sich in [16]. Die Übertragung der Daten in einem seriellen Datenstrom findet in einem speziellen Musterstatt wie in Tab. 3.1 dargestellt.

Ein Muster dieser Art entsteht, da der PCM3003 die Ausgabedaten zweier ADC auf eine Datenleitung zusammenfasst (Multiplexing) und an den DSP versendet. Im Umkehrschluss findet eine Trennung der zusammengefassten Daten statt (Demultiplexing), wenn der DSP Daten an den PCM3003 überträgt. Abb. 3.8 zeigt, in welcher Reihenfolge die Kanäle in die beiden Datenströme zusammengefasst werden [9].

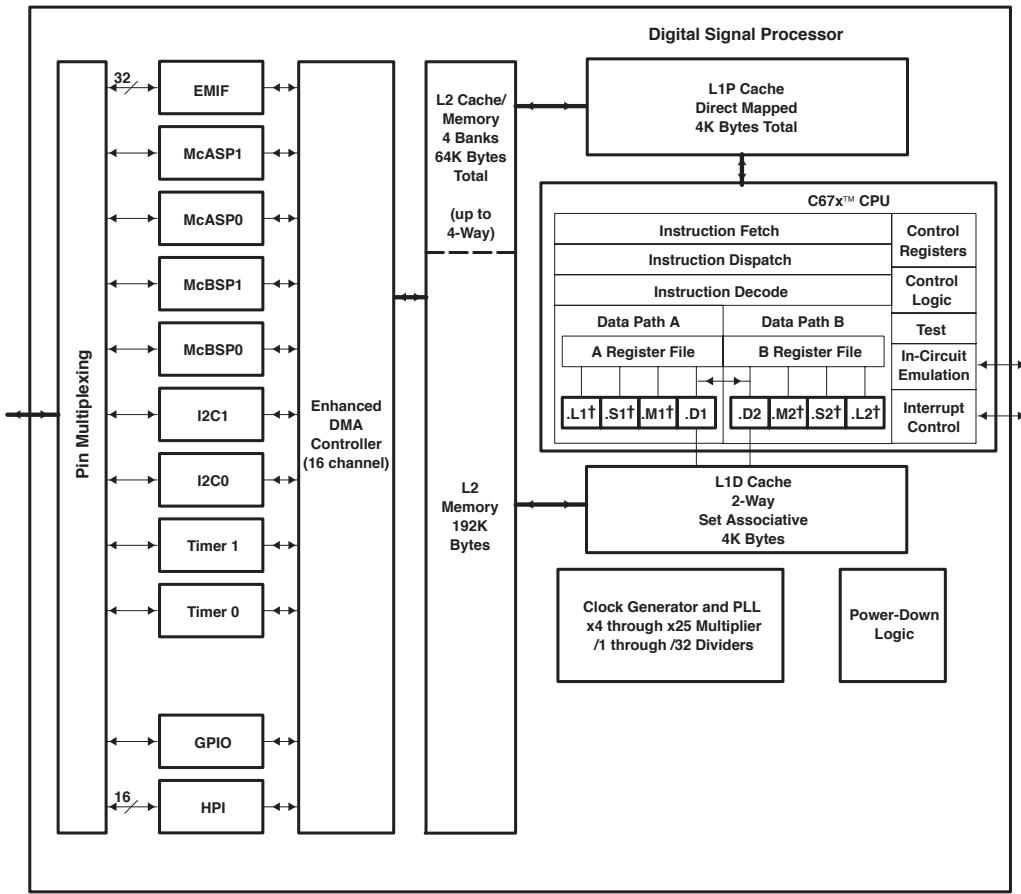


Abb. 3.6: Blockdiagramm der TMS320C6713b CPU (vgl. [17, S. 13])

### 3.1.4 EDMA-Verfahren

Der herkömmliche Weg, Abtastwerte aus einem ADC-Register zu lesen, ist durch Verwendung einer Interrupt-getriggerten Funktion (ISR<sup>9</sup>). Dies bedeutet allerdings, dass der DSP im Intervall der Abtastfrequenz seine aktuelle Operation unterbrechen muss, um die ISR auszuführen. In Anbetracht eines echtzeitfähigen Systems, in dem alle Berechnungen innerhalb eines Signalblocks (besteht hier aus einer Anzahl von 256 Abtastwerten) erfolgen müssen sind Unterbrechungen solcher Art von großem Nachteil.

Zur Vermeidung von Unterbrechungen ist es erforderlich, dass der Prozess der Datensammlung einem zusätzlichen Modul zugeteilt wird. Das System verwendet daher den EDMA-Controller zur Sammlung von Abtastwerten.

<sup>9</sup>Interrupt Service Routinen

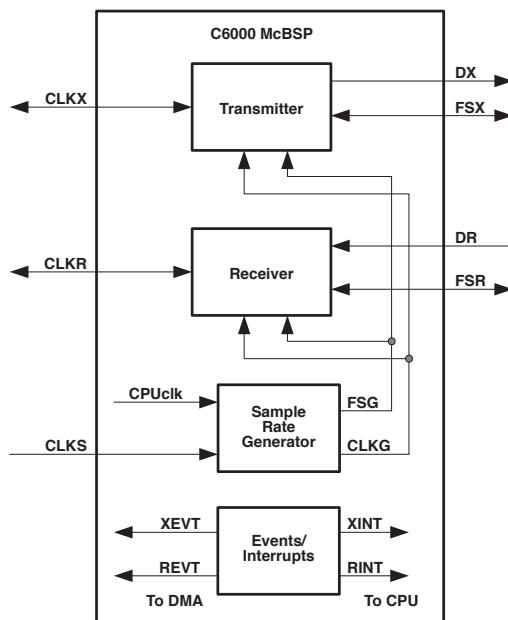


Abb. 3.7: McBSP Fubktionsblockdiagramm (vgl. [16, S. 2])

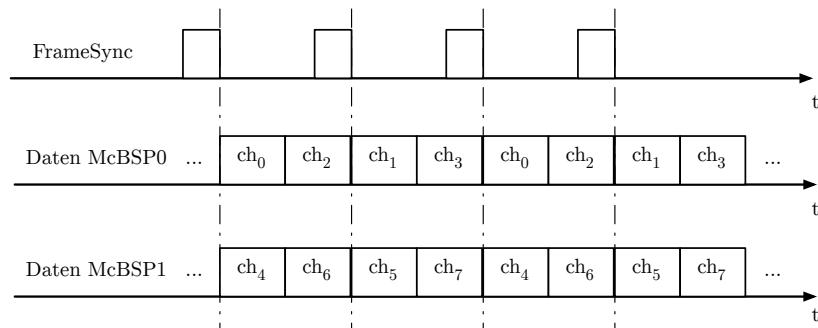


Abb. 3.8: Reihenfolge des Kanalmultilexing durch den McBSP

Mit einer vorher definierten Blockgröße informiert der EDMA-Controller den DSP mit Hilfe eines Interrupts, wann ein Block bereit zur Datenverarbeitung ist. Das Interrupt-Intervall hängt nun von der Abtastrate sowie der Blocklänge ab. Die Kommunikationsschritte zwischen DSP und EDMA-Controller lassen sich wie in Abb. 3.9 dargestellt chronologisch auflisten:

1. Die Hauptroutine initialisiert alle nötigen EDMA-Register und startet diesen.

Codec	Kanal	DSP-Kanal
1	links	0
2	links	2
1	rechts	1
2	rechts	3
3	links	4
4	links	6
3	rechts	5
4	rechts	7

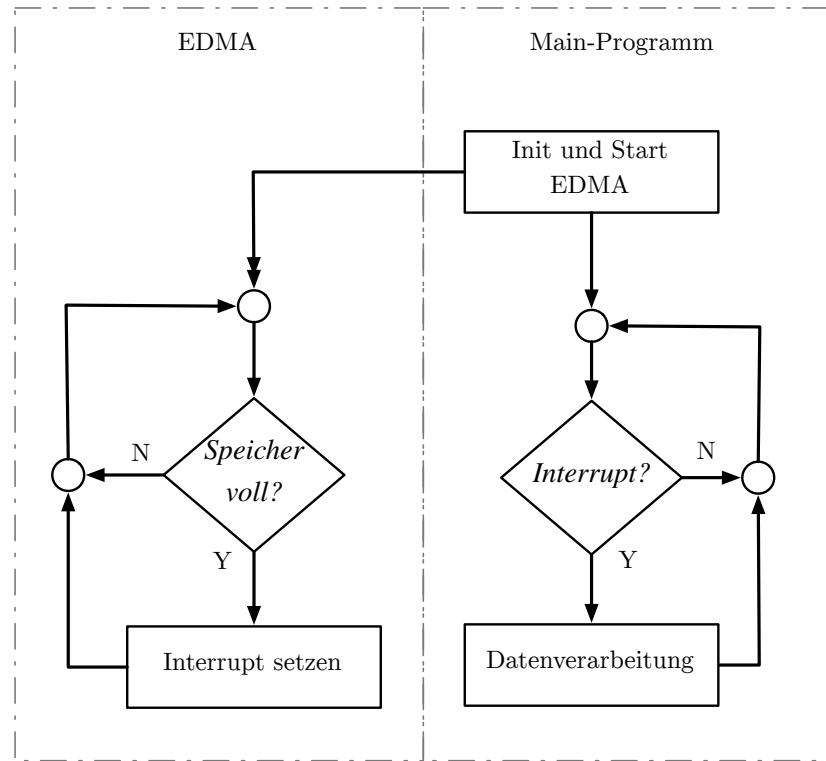
**Tab. 3.1:** McBSP Kanalzuordnung

2. Die Hauptroutine wartet auf den Interrupt vom EDMA-Controller welcher signalisiert, dass ein Signalblock bereit zur Datenverarbeitung ist-
3. Die Datenverarbeitung beginnt, sobald ein EDMA-Interrupt empfangen wurde.
4. Nach Beenden der Datenverarbeitung wartet die Hauptroutine auf den nächsten EDMA-Interrupt.

### 3.1.5 Ping-Pong-Speicherverfahren

Zur Vermeidung von Datenverlust während der Datenverarbeitung eines Signalblocks sowie die zur Schaffung einer Möglichkeit zur unkomplizierten Einstellung des Zeitverhalten der Datenverarbeitungsroutine wird im Folgenden ein Ping-Pong-Speicherverfahren (Doublebuffer) angewendet. Hierbei sind anstelle eines Speicherfeldes zwei Felder mit identischer Größe im Einsatz. Wie in Abb. 3.10 dargestellt, alterniert die Datenverarbeitung zwischen diesen beiden Speichern. Vorteil dieser Methode ist, dass während der Verarbeitung eines Datenblocks im Hintergrund (nebenläufig) der zweite Block mit neuen Daten beschrieben wird. So kommt es bei den kontinuerlich am Mikrofonarray eintreffenden Audiosignalen zu keinem Datenverlust. Weiterhin entsteht so eine definierter Zeitabschnitt, in dem der Algorithmus seine Berechnungen abgeschlossen haben muss. Die wesentlichen Schritte dieses Verfahrens lassen sich wie folgt auflisten:

1. Der EDMA-Controller erhält Daten vom der PCM3003 und speichert sie im Ping-Speicher.

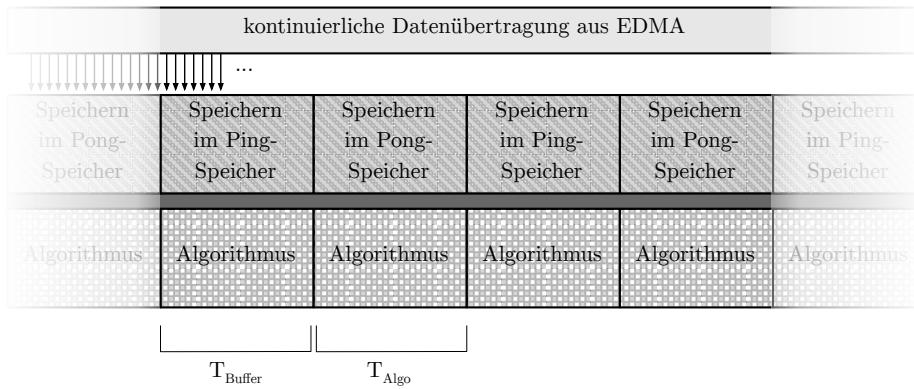


**Abb. 3.9:** Interrupthandling zwischen EDMA-Controller und der Hauptroutine

2. Ist die Blockgrenze des Ping-Speichers erreicht, beginnt der EDMA-Controller seinen Speichervorgang auf dem Pong-Speicher, sendet einen Interrupt an den DSP und gibt somit den Ping-Speicher zur Verarbeitung frei.
3. Ist die Blockgrenze des Pong-Speichers erreicht, wird dieser ebenfalls mit einem Interrupt zur Verarbeitung freigegeben und die Prozedur beginnt erneut.

## 3.2 Algorithmische Umsetzung

Im folgenden Abschnitt soll nun gezeigt werden, auf welche Weise die Implementierung der einzelnen Systemfunktionen in der Programmiersprache C umgesetzt wurde. Im Vergleich zur in Abschnitt 2.5 beschriebenen Simulation in MATLAB®, die in Bezug auf Genauigkeit sowie Geschwindigkeit ideal-



**Abb. 3.10:** Ping-Pong-Verfahren der Hauptroutine (vgl. [12, S. 35])

len Bedingungen unterliegt, besitzt der DSP diesbezüglich klare Einschränkungen, die zur Gewährleistung einer stabilen Programmfunction beachtet werden müssen.

In Bezug auf den verwendbaren Zahlenbereich arbeitet MATLAB<sup>®</sup> standardmäßig mit doppelter Genauigkeit. In C entspricht das dem Variablentyp `double` mit einer Größe von 64 bit. Da der DSP in dieser Anwendung in Fließkommaarithmetik programmiert wird und lediglich einen internen Speicher von 256kB besitzt, erfolgt die Auflösung von Zahlen maximal mit dem 32 bit Datentyp `float`. Des weiteren gilt es, Kopiervorgänge von Feldinhalten weitestgehend zu vermeiden und die von C zur Verfügung gestellte Zeigerarithmetik auszunutzen.

### 3.2.1 Aufbau des C-Projekts

Der Aufbau des C-Projekts ist in Quelltext 3.1 dargestellt. Es beinhaltet drei C-Dateien, in denen die gesamte Funktionalität abgebildet ist. Die Datei `main.c`, enthält die Hauptroutine `main()` in welcher sich der Algorithmus zur Sprecherlokalisierung befindet. In der Datei `defines_globales.c` befindet sich lediglich eine Funktion `init()`. Diese wird zu Beginn des Programms aufgerufen und initialisiert alle benötigten Konstanten. Sie Datei `functions.c` beinhaltet alle benötigten Funktionen zur Berechnung der Sprecherposition. Sie können aus `main.c` aufgerufen werden.

```

1      main.c
2          |-- bios.h
3          |-- setup.h
4          |-- functions.h
5          |-- defines_globales.h
6
7      defines_globales.c
8          |-- defines_globales.h
9          |-- functions.h
10         |-- DSPF_sp_bitrev_cplx.h
11
12     functions.c
13         |-- functions.h
14         |-- defines_globales.h
15         |-- DSPF_sp_cfft2_dit.h
16         |-- DSPF_sp_icfft2_dif.h
17         |-- DSPF_sp_bitrev_cplx.h

```

**Quelltext 3.1:** Aufbau des C-Poектs

### 3.2.2 Headerdateien des Projekts

Zur bequemeren Einstellung der zur Verfügung stehenden Programmmodi wurde die Headerdatei `setup.h` erstellt. In ihr befinden sich die folgenden Makros zur Manipulation der Programmfunction:

**Auswahl des Programmmodus** Das Makro `C_MODE_ON` bestimmt, ob das Programm im C-Mode läuft und alle hardwarespezifischen Programm-elemente ausblendet. Hierfür muss in der Datei `main.c` eine entsprechende Headerdatei gewählt werden, aus der der Algorithmus synthetisch erzeugte Abtastwerte entnehmen kann.

**Debug-Modus** Das Makro `DEBUG_MODE_ON` schaltet den Debug-Modus ein, wodurch Debug-Nachrichten unter Verwendung der Funktion `printf()` auf die Konsole geschrieben werden.

**Überprüfung des Zeitverhaltens** Das Makro `PROFILE_MODE_ON` bewirkt, dass die Messung des Zeitverhaltens mit Hilfe der GPIO<sup>10</sup> Ausgänge stattfinden kann. Zur zusätzlichen Echtzeitkontrolle kann das Makro `ADC_LOOP_THROUGH_MODE_ON` gesetzt werden, wodurch die Daten aus Kanal 1 des ADC direkt auf Kanal 1 des DAC kopiert wird.

---

<sup>10</sup>General-purpose input/output

**Serielle Übertragung der Daten** Durch das Setzen des Makros `UART_MODE_ON` versendet der DSP nach jeder Berechnung einer gültigen Sprecherposition beiden Raumwinkel  $\phi$  und  $\theta$  mit Hilfe der `UART11`-Schnittstelle.

**Einschalten der Optimierung** Das Makro `HISTOGRAM_MODE_ON` bewirkt die Filterung des Ergebnisses mit Hilfe eines Histogramms. Die Einstellungen zur Histogrammlänge sowie der Schwelle befinden sich in der Datei `defines_globales.h`.

Weiterhin existiert die Headerdatei `defines_globales.h` sowie `functions.h`. `defines_globales.h` enthält alle vordefinierten Konstanten. Zur fehlerfreien Programmfunction wird empfohlen, alle, außer die Werte in der Histogrammsektion, unverändert zu lassen. In der Datei `functions.h` befinden sich alle Funktionsprototypen, so dass durch Einbindung dieser Headerdatei in eine andere C-Datei auf alle Funktionen zugegriffen werden kann.

### 3.2.3 DSP Speicherbelegung

Das DSP-Modul verfügt wie oben genannt über 256kB internen Speicher sowie über 2MB Flash Speicher, die für das Programm genutzt werden können. Es stehen 64MB SDRAM zur Verfügung, die im Vergleich zum internen Speicher nur mit einem Drittel der Geschwindigkeit operieren. Nach einem Test des Programms unter Verwendung dieses Speichers stellte sich heraus, dass sich dieser für den Echtzeitbetrieb als ungeeignet erweist.

Die Größen für den Stack-und Heapspeicher wurden, wie durch das Demoprogramm vorgegeben, beibehalten. Der Stack hat eine Größe von 1024 Byte und der Heap eine Größe von 4096 Byte.

Aufgrund der Verwendung von acht Mikrofonen und einer Blocklänge von 256 Abtastwerten nehmen die Felder, welche die Abtastwerte sowie die Kreuzkorrelationsfunktionen enthalten, den meisten Speicherplatz ein. Mit einer Gesamtgröße von ca. 147kB nehmen sie ca. 57% des Gesamtspeichers ein.

Zur Verkleinerung der Zugriffszeiten auf Variable mit dem Datentyp `st_complex` wurden diese mit dem Pragmabefehl `DATA_ALIGN` nur auf gerade Adressen im Speicher abgelegt. Dies erhöht die Zugriffsgeschwindigkeit bei der Verwendung der FFT-Routine (vgl. [12, S. 67]).

Zur Optimierung der Geschwindigkeit wurden alle Funktionen der Datei `functions.c` wie in [19] beschrieben, in der Stufe 5 vom Compiler op-

---

<sup>11</sup>Universal Asynchronous Receiver Transmitter

timiert. Zu beachten ist, dass sich solch eine Optimierung zu Lasten des verwendeten Speichers auswirkt.

### 3.2.4 Ablauf der Hauptroutine

Abb. 3.11 zeigt den Ablauf der Hauptroutine `main()`, welche sich in der Datei `main.c` befindet. Als Grundlage dieser Datei diente das von D.SignT bereitgestellte Demoprogramm `pcm3003`, welches bereits die Nutzung des EDMA sowie die Ping-Pong-Speichertechnik für acht Kanäle beinhaltete. Die wesentlichen Bestandteile dieser Datei sind:

- Initialisierung aller notwendigen Hardware
  - DSP-Modul
  - UART-Modul
  - Zurücksetzen des PCM3003
  - Konfiguration der beiden McBSP
  - Konfiguration des EDMA-Controllers
- Initialisierung aller Variablen und Methoden des Algorithmus
- Algorithmus zur Sprecherlokalisierung

Wie in Abb. 3.11 dargestellt, läuft die Routine in einer Endlosschleife und kann nur durch Abschalten des DSP beendet werden.

### 3.2.5 Methodenumsetzung

In diesem Abschnitt folgt eine detaillierte Beschreibung, auf welche Weise die benötigten Funktionalitäten in Funktionen der Programmiersprache C umgesetzt wurden.

#### Energieberechnung

Die Berechnung der Blockenergie erfolgt als erster Schritt in der Hauptroutine. Das Ergebnis zeigt an, ob der aktuelle Signalblock als stimmhaftes Sprachsignal interpretiert wird oder nicht. Auf Grund der kleinen Abstände zwischen den Sensoren wird angenommen, dass sich die Energie von einem Mikrofon zum anderen kaum unterscheidet. Aus diesem Grund wird nur der Signalblock von Sensor 1 betrachtet. Erreicht der Energiewert die Blockenergieschwelle `ENERGIE_LIMIT` wird dieser als gültig markiert und für die Berechnung der Sprecherposition verwendet.

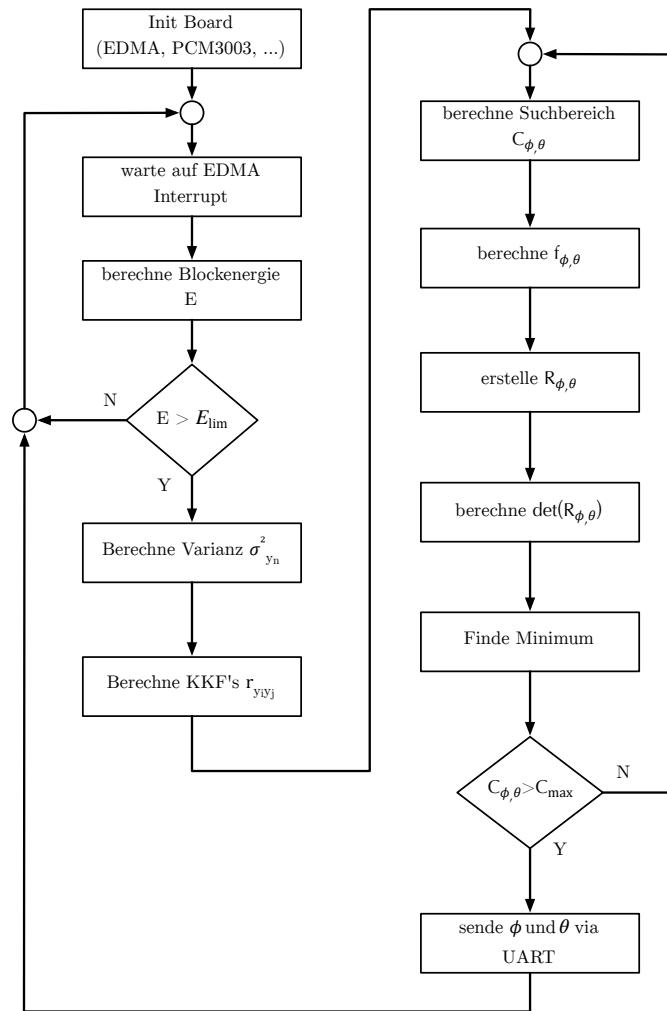


Abb. 3.11: Ablaufdiagramm der Hauptroutine main()

### Einlesen eines Datenblocks

Aufgrund der Verarbeitung der Daten in Fließkommaarithmetik ist es notwendig, die vom ADC gelieferten Festkommawerte in das Format float zu konvertieren. Da es sich hierbei um komplexwertige Signalverarbeitung handelt, findet die Speicherung in einer komplexen Struktur vom Typ `st_complex` statt. Des weiteren erfolgt wie in Quelltext 3.2 dargestellt das Zero-Padding für die zweite Hälfte des komplexen Feldes.

```

1 for ( i=0; i < int16_ScrLen; i++ ) {
2
3     // copy source buffer to complex struct
4     st_Dest[i].re = ( float ) int16_Scr[i] ) *
5         CONVERT_INT16_TO_FLOOR;
6     st_Dest[i].im = 0.0;
7
8     // do zero-padding for second half of destination buffer
9     st_Dest[i+int16_ScrLen].re = 0.0;
10    st_Dest[i+int16_ScrLen].im = 0.0;
11 }
```

**Quelltext 3.2:** Einlesen eines Datenblocks vom ADC inkl. Zero-Padding

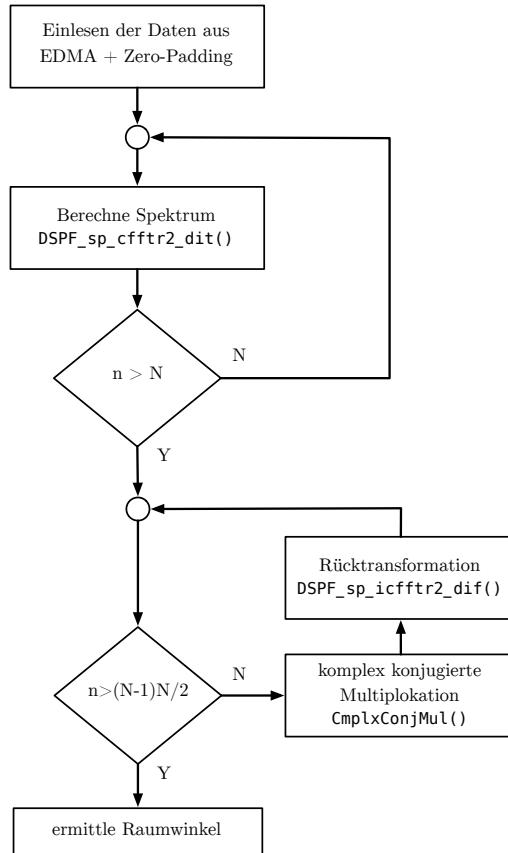
### Schnelle Kreuzkorrelation

Die Berechnung der Kreuzkorrelation wurde wie in Abschnitt 2.2.1 beschrieben über die Multiplikation der Signalspektren realisiert. Grundsätzlich müssen dafür auf Grund der Symmetrieeigenschaft 28 Kreuzkorrelationsfunktionen berechnet werden. Wie in Abb. 3.12 dargestellt erfolgt dies über sukzessive Berechnung und konjugiertkomplexe Multiplikation der Spektren mit anschließender Realteilbildung.

Zur Berechnung des Kreuzleistungsdichtespektrums über die konjugiert komplexe Multiplikation wurde folgender Zusammenhang genutzt:

$$(Re_1 + j Im_1) \cdot (Re_2 + j Im_2)^* = Re_1 Re_2 + Im_1 Im_2 + j (Im_1 Re_2 - Re_1 Im_2) \quad (3.1)$$

Des Weiteren erfolgte die Ermittlung der Signalspektren über die von Texas Instruments® zur Verfügung gestellten Softwarebibliothek TMS320C67x DSP Library[18]. Diese enthält spezielle für diesen Prozessortyp hochoptimierte Assemblerroutinen. Zur Transformation in den Frequenzbereich wurde die Funktion `DSPF_sp_cfft2_dit()` [18, S. 4-13] verwendet. Zur Rücktransformation in den Zeitbereich kam die Funktion `DSPF_sp_icfft2_dif()` [18, S. 4-34] zum Einsatz. Diese beiden Funktionen führen ihre Berechnungen unter Angabe eines Datenfeldes, einer Feldlänge sowie den Twiddlefaktoren durch. Die Daten müssen in komplexer Form vorliegen wobei Realanteil und Imaginäranteil hintereinander angeordnet sind:  $Re(x_0), Im(x_0), Re(x_1), Im(x_1), \dots, Re(x_{N-1}), Im(x_{N-1})$ . Zu beachten ist, dass das Datenfeld für Eingangs- sowie Ausgangswerte genutzt wird und die Eingangsdaten somit durch die Ausgangsdaten überschrieben werden. Da die Twiddle-Faktoren  $W_N^k$  nur von der FFT-Länge abhängig sind,



**Abb. 3.12:** Ablaufdiagramm der Kreuzkorrelationsroutine

können diese in der Initialisierungsphase erstellt werden. Sie sind gegeben durch(vgl. [14, S. 2]):

$$W_N^k = e^{-j\frac{2\pi k}{N}} = \cos(\delta \cdot k) - j \sin(\delta \cdot k) \quad \text{mit} \quad \delta = \frac{2\pi}{N} \quad (3.2)$$

Quelltext 3.3 zeigt die Erstellung dieser Faktoren innerhalb einer Schleife in der `init()`-Funktion. Des Weiteren findet hier unter Verwendung der Funktionen `digitrev_index()` und `DSPF_sp_bitrev_cplx()` die Anpassung der Twiddlefaktoren in die bitweise umsortierte Form für die FFT statt.

```

1   for( i = 0; i < N_CORRELATION/RADIX; i++ ) {
2     W[i].re = (float) cos(Delta*i);    // real pos
3     component of W
4     W[i].im = (float) -sin(Delta*i);   // neg imag
5     component of W
6   }
7   // produces index for bitrev() W
8   digitrev_index( iTwid, N_CORRELATION/RADIX, RADIX );
9   DSPF_sp_bitrev_cplx( (double*) W , iTwid,
10    N_CORRELATION/RADIX );

```

**Quelltext 3.3:** Erstellung der Twiddlefaktoren innhalb einer Schleife in der `init()`-Funktion.

Quelltext 3.4 zeigt den Aufbau der Korrelationsroutine. Hier werden zunächst alle acht Zeitsignale in den Frequenzbereich transformiert. Anschließend erfolgt die Berechnung des Kreuzleistungsdichtespektrums unter Verwendung der Funktion `CmplxConjMul()` sowie die Rücktransformation in den Zeitbereich mit Hilfe von `DSPF_sp_icfftr2_dif()`. Im Hinblick auf einen späteren Betrieb in Echtzeit ist es sinnvoll, sparsam mit dem zur Verfügung stehenden Speicher umzugehen, da die Länge eines EDMA-Rahmens unter Umständen vergrößert werden muss. Aus diesem Grund wird darauf verzichtet, jede der 28 Kreuzkorrelationsfunktionen in voller Länge abzuspeichern (512 Abtastwerte). Statt dessen existiert lediglich ein zentrales Array (`st_Buf.f_CmplxSigBufXY`) in dem genau ein KKF abgelegt werden kann. Wie in Quelltext 3.4 dargestellt, wird der KKF, unter Berücksichtigung der maximalen Korrelationsdauer (siehe Abschnitt 2.5.1) ein Datenblock der Länge  $K_{New} = 2K_{max} + 1$  entnommen und in einem separaten Feld abgelegt. Diese Vorgehen ist möglich, da wie bereits erwähnt, lediglich auf  $K_{New}$  Abtastwerte im Intervall  $-K_{max} \leq n \leq K_{max}$  zugegriffen wird. Im Vergleich zur herkömmlichen Speicherung aller Abtastwerte, erzielt dieses Vorgehen eine Speicherersparnis von 53,6 kB (entspricht ca. 30% des Gesamtspeichers).

```

1 // Calculate all FFT's in place
2 for ( i=0; i < int16_NumOfMics; i++ ) {
3     DSPF_sp_cfftr2_dit( (float*) st_Scr[i], (float*) W,
4                           int16_BufLen );
5 }
6 // Calculate Cross-Correlation-Functions in place
7 n = 0;
8 for ( i=0; i < (int16_NumOfMics-1); i++ ) {           // Row
9     for ( j=(i+1); j < int16_NumOfMics; j++ ) {       // Column
10        CmplxConjMul( st_Scr[i], st_Scr[j], st_Helper,
11                          int16_BufLen );
12        DSPF_sp_icfftr2_dif( (float*) st_Helper, (float*) W,
13                               int16_BufLen );
14        // Set Zero-Element
15        st_Dest[n][((MAX_DELAY-1)/2)] = st_Helper[0].re *
16                      INV_N_CORRELATION;
17        // Store required date block of n-th CCF (only
18                      real-component)
19        for( k=0; k < (MAX_DELAY-1)/2; k++ ) {
20            st_Dest[n][k] = st_Helper[k + int16_BufLen -
21                                (MAX_DELAY-1)/2].re * INV_N_CORRELATION;
22            st_Dest[n][k + ((MAX_DELAY+1)/2)] =
23                st_Helper[k+1].re * INV_N_CORRELATION;
24        }
25        n++;
26    }
27 }
28 }
```

**Quelltext 3.4:** Berechnung der Kreuzkorrelationsfunktionen

### Mehrkanal-Kreuzkorrelationskoeffizient

Zur Berechnung des Mehrkanal-Kreuzkorrelationskoeffizienten werden folgende Schritte durchgeführt:

1. Berechnung des Suchbereichs  $C_{\phi,\theta}$ .
2. Berechnung der Verzögerungen zu den theoretischen Raumwinkeln  $\phi$  und  $\theta$ .
3. Erstellung der räumlichen Korrelationsmatrix  $R_{\phi,\theta}$ .
4. Berechnung der Determinanten  $\det[R_{\phi,\theta}]$ .
5. Wiederholung bei Punkt 1 bis minimaler Suchbereich  $C_{\phi,\theta_{min}}$  erreicht ist.

Die gesamte Berechnung wurde in die Funktion `SearchAndFind()` ausgelagert, die u.a. einen Zeiger auf die Struktur erhält, in der Kreuzkorrelationsfunktionen enthalten sind. Hier wird zunächst wie in Abschnitt 2.5.5 beschrieben der Suchbereich  $C_{\phi,\theta}$  zur optimierten Suche berechnet. Anschließend erfolgen die Berechnung der theoretischen Verzögerungen und die Erstellung der räumlichen Korrelationsmatrix  $R_{\phi,\theta}$  sowie die Berechnung von dessen Determinante. Dieser Vorgang wird solange wiederholt, bis alle Richtungen im Suchbereich durchlaufen wurden. Anschließend erfolgt die Verkleinerung des Suchbereiches und der Vorgang beginnt erneut. Erreicht der Suchbereich sein Minimum, können die gefundenen Raumwinkel als globales Minimum angesehen werden und der Algorithmus endet.

Zur Berechnung der theoretischen Laufzeitunterschiede wurde die Funktion `UCAFunctionTauRound` verwendet. Zur Erstellung der räumlichen Korrelationsmatrix dient die Funktion `Create_R()`, die mit Hilfe der theoretischen Verzögerungen den 28 Kreuzkorrelationsfunktionen 64 Werte einnimmt, um die Matrix  $R_{\phi,\theta}$  zu bilden.

Zur Berechnung der Determinante existiert die Funktion `CalcDetNxN()`. Determinanten von Matrizen bis zu einer Größe von  $3 \times 3$  sind durch klare Rechenvorschriften zu ermitteln. Für Determinanten höherer Ordnung müssen spezielle Verfahren wie das gaußsche Eliminationsverfahren, die Leibniz-Formel oder der Laplace'sche Entwicklungssatz verwendet werden. Auf Grund eines erhöhten Rechenaufwands der letzten beiden Methoden wird im Folgenden das gaußsche Eliminationsverfahren implementiert, das in der Lage ist, eine  $N \times N$  Determinante zu berechnen. Dazu muss die Matrix **A**, gegeben durch:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,N} \\ a_{2,1} & a_{2,2} & \dots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \dots & a_{N,N} \end{bmatrix} \quad (3.3)$$

unter Verwendung elementarer Zeilenumformungen in die obere Dreiecksform gebracht werden. Dies bedeutet, alle Elemente unterhalb der Hauptdiagonalen müssen den Wert Null betragen. Die Matrix **A** mit den veränderten Koeffizienten  $b_{i,j}$  ist somit gegeben durch:

$$\mathbf{A} = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,N} \\ 0 & a_{2,2} & \dots & b_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_{N,N} \end{bmatrix} \quad (3.4)$$

Die Determinante kann nun als Produkt der Elemente auf der Hauptdiagonalen wie folgt ermittelt werden:

$$\det(\mathbf{A}) = (-1)^r \cdot b_{1,1} \cdot b_{2,2} \cdots \cdots b_{N,N} \quad (3.5)$$

$$= (-1)^r \cdot \prod_{i=1}^N b_{i,i} \quad (3.6)$$

Der Parameter  $r$  beschreibt dabei die Anzahl der durchgeführten Zeilenvertauschungen.

### Histogrammoptimierung

Quelltext 3.5 zeigt einen Ausschnitt der Histogrammerstellung für den Raumwinkel  $\phi$ . Die Erstellung des Histogramms erfolgt stets über den gesamten Ring-Speicher wobei nur gültige Werte (Werte größer als Konstante INDEX\_INVALID) entnommen werden. Um Rechenzeit einzusparen, erfolgt die Suche des Maximums bereits bei der Histogrammerstellung. Dazu wird der aktuelle Wert mit dem vorherigen verglichen. Ist der aktuelle Wert höher, wird dieser als neues Maximum markiert und dessen Index sowie Wert abgespeichert. Das identische Verfahren erfolgt im Anschluss für den zweiten Raumwinkel  $\theta$ . Erreicht der Maximalwert des Histogramms die Histogrammschwelle int16\_HistogramThreshold, wird dieser als gültiger Histogrammschätzwert markiert und kann ausgegeben werden.

```

1 // Loop over entire histogram-ring-buffer
2 for ( i=0; i < N_HISTOGRAM; i++ ) {
3
4 // Check angle for validation (voiced)
5 if ( int16_HistRingBuf[PHI][i] > INDEX_INVALID ) {
6
7 // Increment histogram value by one
8 st_Phi->int16_Hist[ int16_HistRingBuf[PHI][i] ]++;
9
10 // Check maximum / store value and index
11 if ( st_Phi->int16_Hist[ int16_HistRingBuf[PHI][i] ] >
12     st_Phi->int16_HistEstVal ) {
13     st_Phi->int16_HistEstVal = st_Phi->int16_Hist[
14         int16_HistRingBuf[PHI][i] ];
15     st_Phi->int16_HistEstIdx = int16_HistRingBuf[PHI][i];
16 }
17 // Increment histogram counter
18 PhiHistCnt++;
}
```

**Quelltext 3.5:** Histogrammerstellung für den Raumwinkel  $\phi$

### Datenversand über RS232-Schnittstelle

Das Versenden der Schätzergebnisse erfolgt unter Verwendung der UART-Schnittstelle des DSP. Als Grundlage hierfür dient die zur Verfügung gestellte Funktion `send_string()`. Diese erwartet einen Zeiger auf ein char-Array, in dem die zu sendenden Daten im ASCII<sup>12</sup>-Format enthalten sind. Da die gesamte Signalverarbeitung im Format `float` stattfindet, muss hier zunächst eine Typ-Konvertierung durchgeführt werden. Als Grundlage hierfür steht die Funktion `stoa()` zur Verfügung. Diese ist in der Lage, eine Zahl im Format `short` in das benötigte ASCII Format zu bringen und einen Zeiger auf das ASCII Zeichen enthaltende char-Array zurückzugeben. Auf Grund der Ausgabe der Raumwinkel in Radiant wird die maximale Anzahl von Nachkommastellen auf vier begrenzt. Die Umformung vom Typ `float` in Typ `short` erfolgt durch Multiplikation mit dem Faktor  $10^3$ . Diese Operation verschiebt das Komma um drei Stellen nach rechts, so dass aus der Zahl 1,2345 die Zahl 1234,6 entsteht. Wie in Quelltext 3.6 dargestellt, ergibt sich nach einer anschließenden cast-Operation der gewünscht Wert im Datentyp `short`.

```
1     int16_angle = (short) (1000 * f_angle)
```

**Quelltext 3.6:** Konvertierung vom Datentyp `float` in Typ `short` mit 3 Nachkommastellen.

Da in dieser Anwendung zwei Winkelergebnisse versendet werden müssen, wird die o.g. Methode in die Funktion `Short2ASCII()` erweitert, so dass sich das in Abb. 3.13 dargestellte Kommunikationsprotokoll ergibt. Jeder Winkel kann somit in einem 6Byte langen char-Array abgebildet werden. Vor jedem Winkelwert wird ein entsprechender Winkel- Identifizierer gesendet. Dieser zeigt an, ob es sich um den Raumwinkel  $\phi$  oder  $\theta$  handelt. Anschließend erfolgt das Senden des Wertes und dessen Vorzeichen. Hierbei ist wichtig, dass das Vorzeichen als letztes Zeichen gesendet wird, da es die Dekodierung der Zeichenkette auf Seiten des Empfängers enorm vereinfacht.

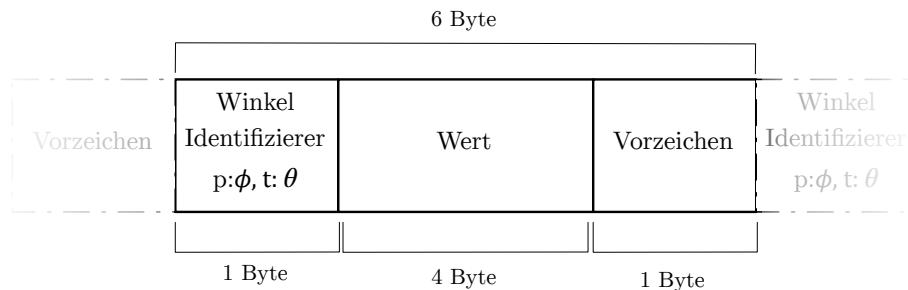
### 3.2.6 Displaykomponente des Systems

Zur visuellen Ausgabe der ermittelten Raumwinkel wird im Folgenden eine graphische Benutzeroberfläche in der Programmiersprache Java entwickelt. Wie in Abb. 3.14 dargestellt, beinhaltet diese folgende Komponenten:

- Data-LED zur Anzeige des Datenempfangs

---

<sup>12</sup>American Standard Code for Information Interchange



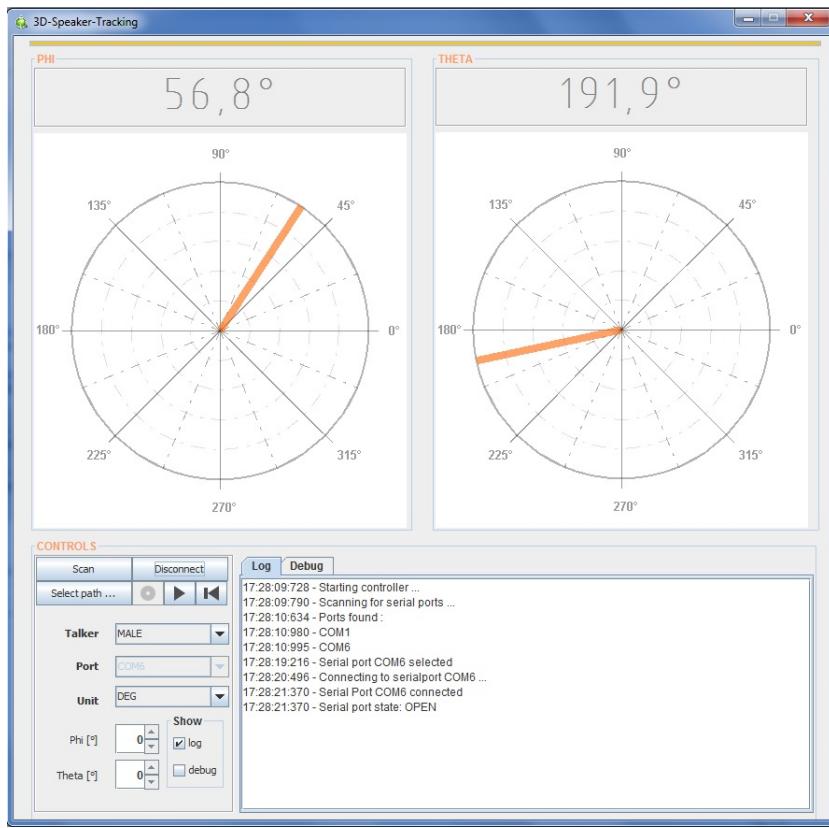
**Abb. 3.13:** Zusammensetzung des Kommunikationsprotokolls zur Übertragung der Winkel unter Verwendung der UART-Schnittstelle.

- Dezimalanzeige des aktuellen Raumwinkels  $\phi$  und  $\theta$
- Prolardiagramm zur Anzeige des aktuellen Raumwinkels  $\phi$  und  $\theta$
- Bereich für Benutzerinteraktion
  - Suchen nach sowie verbinden zu seriellen Schnittstellen
  - Abspielen von Audiodateien (männlicher u. weiblicher Sprecher)
  - Speichern von Ergebnisdaten im MAT-Format
  - Einstellung des Formats (Radian, Grad) der Dezimalzahlen für  $\phi$  und  $\theta$
  - Einstellung theoretischer Winkel zur Namensgebung der MAT-Datei
- Log und Debug Ausgabefenster

Zur übersichtlichen Organisation sowie zur robusten Kommunikation zwischen den einzelnen Klassen wird ein Model-View-Controller Konzept entworfen. Des Weiteren benötigt die Software einen Zustandsautomaten zum fehlerfreien Empfangen sowie zur robusten Dekodierung der Raumwinkel.

### Model-View-Controller

Die gesamte Software-Architektur richtet sich nach dem Model-View-Controller Konzept wie in Abb. 3.15 dargestellt. Hierbei existiert die Klasse `AppController`, die als oberste Controller-Klasse fungiert und somit die Kommunikation zwischen Model und View herstellt und überwacht. Als Model-Komponente wurden folgende Klassen implementiert:



**Abb. 3.14:** Graphische Benutzeroberfläche zur anschaulichen Darstellung der Winkelverläufe

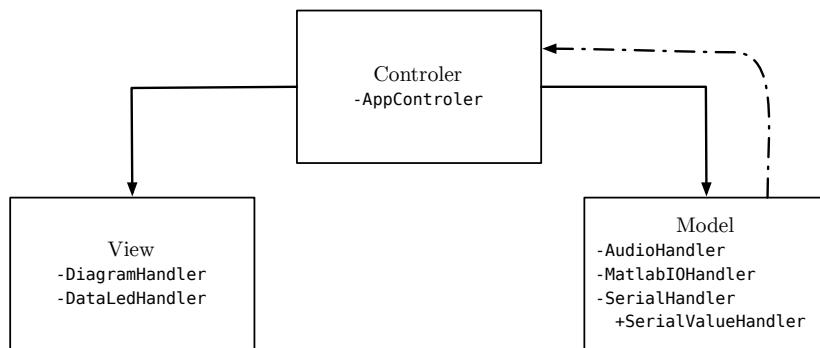
- **AudioHandler** - Dient zum Abspielen von Audiodaten über die integrierte Soundkarte
- **MatlabIOHandler** - Dient zum Speicherung der Raumwinkel in einem auf die Anwendung angepasstem Format.
- **SerialHandler** - Dient zur Kommunikation mit der seriellen Peripherie (Scan, Connect, Disconnect, ...).
  - **SerialValueHandler** - Wird von der Klasse **SerialHandler** instanziiert und implementiert den Zustandsautomaten zum Empfang und zur Dekodierung der Zeichenketten.

Die View-Komponente besteht aus folgenden Klassen:

- **ViewHandler** - Implementiert alle Methoden zur Erstellung der grafischen Benutzeroberfläche. Diese basieren auf dem von Java zur

Verfügung gestellten Swing/AWT-Framework. Des Weiteren instantiiert diese Klasse zwei Objekte der Klasse DiagramHandler zur Erstellung der beiden Polardiagramme. Mir dem Aufruf der Methode `DiagramHandler.setAnglePointer()` kann dem Winkelzeiger der jeweilige Wert des Raumwinkels zugewiesen werden.

- `DiagramHandler` - Wird aus der Klasse `ViewHandler` instantiiert und implementiert alle Methoden zur Manipulation der Polardiagramme
- `DataLedHandler` - Wird aus der Klasse `ViewHandler` instantiiert und implementiert alle Methoden zum Setzen der DataLED. Diese zeigt an, wann Daten über die UART-Schnittstelle empfangen werden.



**Abb. 3.15:** Model-View-Controller Konzept der graphischen Benutzeroberfläche. Die durchgezogenen Linien symbolisieren eine direkte Verbindung. Linien in gestrichelter Darstellung bedeuten die Kommunikation über einen indirekten Weg mit Hilfe des Entwurfsmusters „Beobachter“.

Wie in Abb. 3.15 illustriert existieren direkte unidirektionale<sup>13</sup> Kommunikationswege (durchgezogene Linien) sowie indirekte unidirektionale Kommunikationswege (gestrichelte Linie). Das bedeutet, die Kommunikation findet unter Verwendung von RMI<sup>14</sup> nur vom Controller in Richtung Model und View statt. Um den Controller darüber informieren zu können, dass neue Daten zur Anzeige in der View-Komponente zur Verfügung stehen, muss eine Kommunikationsmöglichkeit in die andere Richtung vorliegen. Um auf

<sup>13</sup>Kommunikation nur in eine Richtung

<sup>14</sup>Remote Method Invocation

rechenintensive Polling-Methoden zu verzichten, bei denen der Controller permanent in einer Schleife auf ein Ready-Flag im Model prüfen muss, wird das Beobachter-Muster (observer pattern) implementiert. Dabei existiert ein Beobachter (in diesem Fall die Controller-Klasse) und ein Veröffentlicher (in diesem Fall die Model-Klasse). Die Controller-Klasse registriert sich zunächst bei der Model-Klasse als Beobachter. Liegen Daten zur Abholung bereit, wird diese Information umgehend an alle Beobachter veröffentlicht, so dass diese entsprechend ihrer internen Logik reagieren können. In diesem Fall werden die Ergebnisse erneut unter Verwendung von RMI in der Model-Klasse abgeholt. Diese Technik ermöglicht, dass der Controller-Klasse während des Empfang und der Dekodierung der Zeichenketten Zeit für andere Algorithmusschritte zur Verfügung stehen.

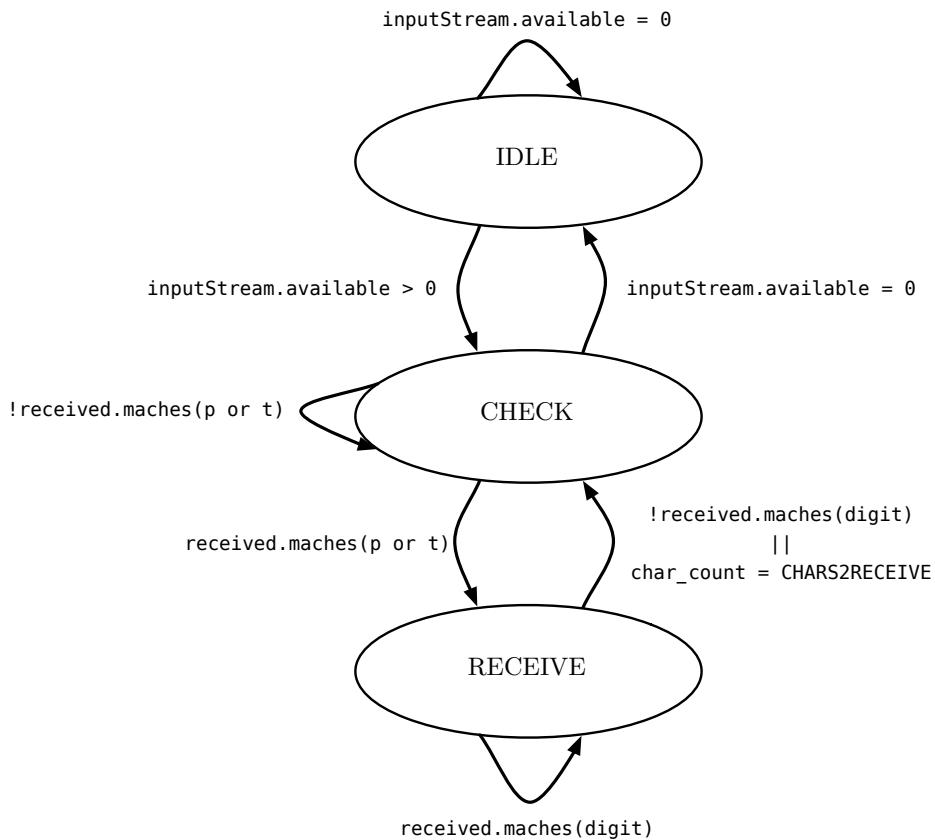
### Zustandsautomat

Abb. 3.16 zeigt den Ablauf des Zustandsautomaten zum Empfangen und zur Dekodierung der vom DSP gesendeten Zeichenketten. Werden keine Daten empfangen befindet sich der Automat im Zustand IDLE. Sobald serielle Daten zum Empfang vorliegen, wechselt er in den Zustand CHECK, empfängt die Daten Byteweise und prüft jedes Byte auf einen der beiden Winkelidentifizierer  $p$  oder  $t$ . Wird eines der beiden Prüfzeichen (Identifizierer) empfangen, begibt sich der Automat in den Zustand RECEIVE, in dem er nun fünf weitere Zeichen empfängt (vier Ziffern + Vorzeichen). Empfängt der Automat innerhalb der fünf Ziffern ein ungültiges Zeichen, wird dieser Winkel verworfen und der Zustand wechselt erneut in CHECK. Empfängt der Automat alle Ziffern korrekt, werden diese nach dem in Abb. 3.17 dargestellten Schema dekodiert und der Zustand wechselt erneut in CHECK. Dieser Vorgang wiederholt sich so lange, bis keine Daten zum Empfang zur Verfügung stehen. Der Automat wechselt dann in den Zustand IDEL und wartet auf neue Daten.

### Export der Ergebnisse im MAT-Format

Zum Export der gespeicherten Raumwinkel im MAT-Format wird das Framework jMatIO[6] verwendet. Das Framework unterstützt den Export von Daten in folgenden MATLAB® Formaten:

- Double array
- UInt8, Int8 array
- UInt64, Int64 array



**Abb. 3.16:** Darstellung des Zustandsautomaten der Klasse `SerialValueHandler`.

- Char array
- Structure
- Cell array
- Sparase array

Quelltext 3.7 zeigt eine Methode der Klasse `MatIOHandler`. Diese implementiert das Erstellen der entsprechenden MATLAB® Felder im den in Java gespeicherten Ergebniswerten. Dank der komfortablen API des `jMatIO`-Frameworks lässt sich dies durch simple Funktionsaufrufe realisieren. So lassen sich ebenfalls Zusatzinformationen wie einen Zeitstempel, der Sprecherotyp sowie die theoretisch eingestellten Raumwinkel speichern um später die Ausgabe in MATLAB® weitestgehend zu automatisieren.

t	1	2	3	4	-	Gewicht	
						$10^0$	1
1						$10^{-1}$	0,2
	2					$10^{-2}$	0,03
		3				$10^{-3}$	0,004
			4			$\Sigma$	1,234
					-	-1	<b>-1,234</b>

Abb. 3.17: Schema zur Dekodierung eines Zeichenketten-Frame

```

1  public void createFile(){
2      this.valueList.add( new MLChar( "created",
3          simpleDateFormat.format( System.currentTimeMillis()
4              ) ) );
5      this.valueList.add( new MLChar( "audioType",
6          this.audioType.toString() ) );
7      this.valueList.add( new MLDouble( "anglesSetUp",
8          this.angleSetUp, this.angleSetUp.length) );
9      this.valueList.add( new MLDouble( "phi",
10         phiValues.toArray(new Double[phiValues.size()]),
11         phiValues.size()) );
12      this.valueList.add( new MLDouble( "theta",
13         thetaValues.toArray(new Double[thetaValues.size()]),
14         thetaValues.size()) );
15
16      try {
17          new MatFileWriter( this.filepath +
18              this.fileSeparator +
19              this.fileName +
20              ".mat", valueList );
21      } catch (IOException e) {
22          e.printStackTrace();
23      }
24      this.phiValues.clear();
25      this.thetaValues.clear();
26  }

```

---

**Quelltext 3.7:** Java-Funktion zum Export von Ergebniswerten in im MAT-Format

# Kapitel 4

## Echtzeitversuch

Im diesem Kapitel erfolgt die Validierung des in Kapitel 3 beschriebenen Echtzeitsystems. Dazu ist erforderlich, das System auf das in Kapitel 1 verlangte Echtzeitkriterium zu untersuchen. Des Weiteren muss nachgewiesen werden, dass die Ergebnisse aus der Simulation in Abschnitt 2.5 unter Berücksichtigung der Raumeigenschaften reproduziert werden können.

### 4.1 Untersuchung des Zeitverhaltens

Nach dem Echtzeitkriterium aus Kapitel 1 müssen alle Rechenoperationen innerhalb der Dauer eines EDMA-Rahmens erfolgt sein. Mit einer Rahmenlänge von 256 (siehe Abschnitt 2.5.1) ergibt das eine Zeit von  $T_{EDMA} = 5,33ms$ . Die Messung des Zeitverhaltens bei dieser Rahmengröße ergibt, dass das Programm nicht in Echtzeit arbeitet und somit die Dauer des Algorithmus die des EDMA-Blocks überschreitet. Als Lösungsansatz wird zunächst die Möglichkeit untersucht, die Rahmenlänge  $L$  zu vergrößern. In Anbetracht der Zweier-Potenz-Regel (siehe Abschnitt 2.2.1) ergibt sich für die nächste Größe eine Anzahl von 512 Abtastwerten und somit eine Zeit von  $T_{EDMA} = 10,7ms$ . Unter Berücksichtigung der Stationaritätseigenschaft von Sprache (siehe Abschnitt 2.3.1) erweist sich diese Dauer als geeignet und wird für den weiteren Verlauf der Arbeit spezifiziert.

Abb. 4.1 zeigt das Zeitverhalten des Tracking-Programms bei einer EDMA-Blockgröße von 512 Abtastwerten. Die Zeitmessung des Algorithmus ergibt, dass das Echtzeitkriterium erfüllt wird, da die Summe der einzelnen Funktionszeiten  $T_{Algo}$  (siehe Tab. 4.1) kleiner ist, als die Dauer eines EDMA-BLOCKS. Werden die Daten jedoch unter Nutzung der UART-Schnittstelle versandt, vergrößert sich die benötigte Zeit und überschreitet die Dauer eines EDMA-Blocks. Da die serielle Schnittstelle für unterschiedliche Übertra-

gungsgeschwindigkeiten (Baudrate) konfiguriert werden kann, wird die Baudrate, wie dargestellt, sukzessiv erhöht. Bei einer Geschwindigkeit von 57600 Baud hat sich die benötigte Zeit zum Senden eines Datenpakets so weit verringert, dass die Dauer eines EDMA-Rahmens insgesamt nicht überschritten wird. Deutlich wird dies auch durch den ADC\_DAC\_LOOP\_THROUGH (in grün dargestellt). Hierbei wird der Speicherinhalt eines EDMA-Frames innerhalb der Hauptroutine direkt vom ADC-Speicher in den DAC-Speicher kopiert. Kann dieser Vorgang erfolgreich abgeschlossen werden entsteht kein Datenverlust (Sprünge) und der Algorithmus arbeitet in Echtzeit (Abb. 4.1c).

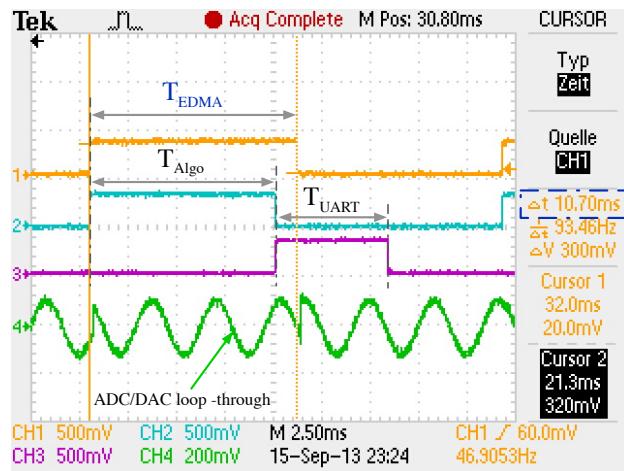
Funktio	Zeit [μs]
ADC_DAC_LOOP_THROUGH	70
Copy2CmplxStruct	430
CalcVariance	70
FastCrossCorrelation	7000
SearchAndFind	1920
CreateHistogram	10
send_string	940
Summe:	10440
EDMA Block	10600
Reserve:	160

**Tab. 4.1:** Profiling des Tracking-Algorithmus

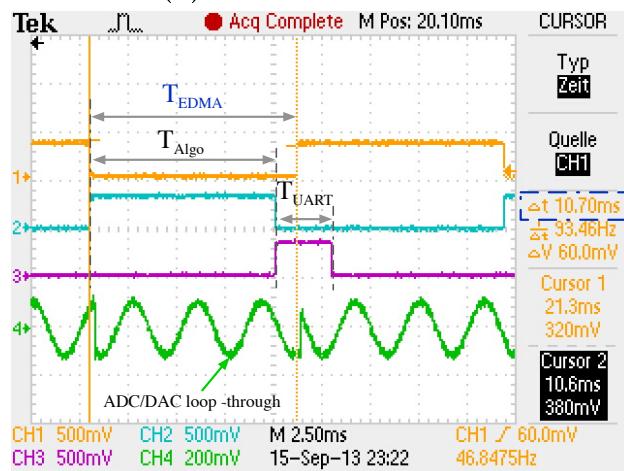
Zu beachten ist, dass die Echtzeitfähigkeit nur erreicht wird, wenn keine Daten unter Verwendung der seriellen Schnittstelle versandt werden. Grund hierfür sind die zusätzlich benötigten Funktionsaufrufe zum Erstellen des Datenpakets sowie dessen serielle Übertragung.

## 4.2 Versuchsaufbau

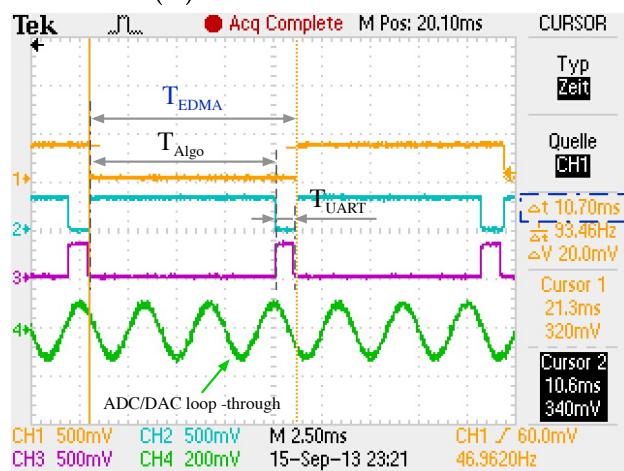
Abb. 4.2 zeigt den schematischen Versuchsaufbau der Echtzeitmessung. Um die Leistung des Systems in akustisch anspruchsvoller Umgebung testen zu können, wurde ein kleiner Raum mit den Abmaßen  $5m \times 3m \times 2,7m$  gewählt. Dieser verfügt über schallharte Wände, so dass die Systemreaktion auf ein diffuses Schallfeld sowie starke Reflexionen untersucht werden können. Zur Einhaltung des in Abschnitt 2.1.2 beschriebenen Fernfeldkriteriums bei einer oberen Sprachcheckfrequenz von 3 kHz wurde ein Abstand von 1,6m gewählt. Zur Einstellung der unterschiedlichen Winkel wird das Array wie in Abb. 4.3 dargestellt drehbar gelagert und erhält eine Winkelauflösung. Das Array



(a) Baudrate: 9600 Baud



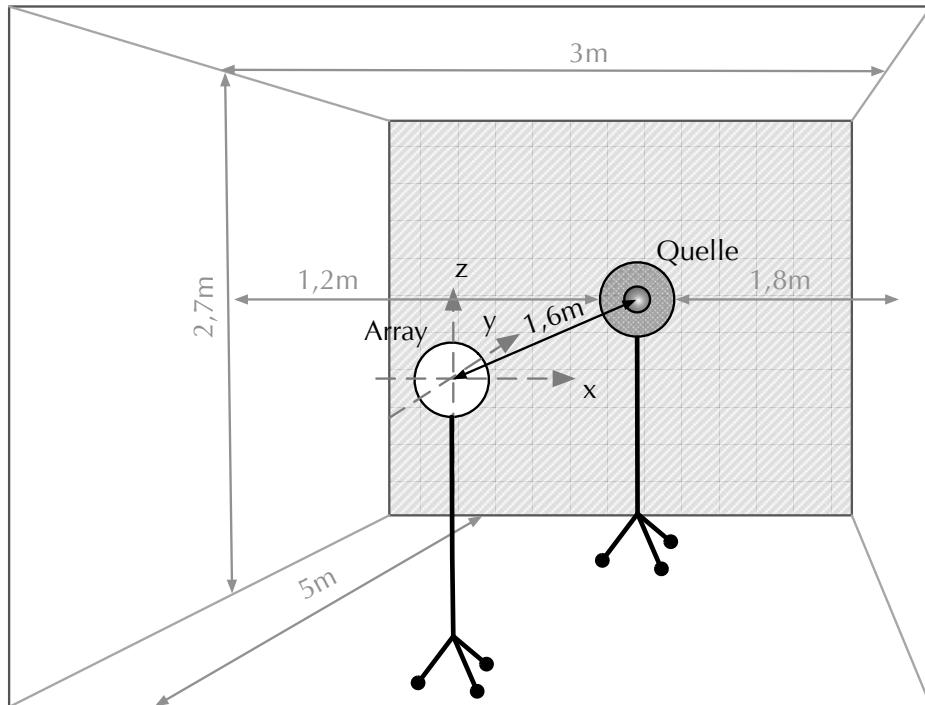
(b) Baudrate: 19200 Baud



(c) Baudrate: 57600 Baud

**Abb. 4.1:** Untersuchung des Zeitverhaltens mit einer EDMA-Dauer von  $T_{EDMA} = 10,7\text{ms}$ .

kann so mit Hilfe des Code Composer™ Studio Graph-Windows exakt ausgerichtet werden und anhand der Winkelteilung in genauen Schritten gedreht werden.



**Abb. 4.2:** Versuchsaufbau der Echtzeitmessung

## 4.3 Ergebnisse

Im Folgenden wird zunächst ein grundlegender Funktionstest des Algorithmus durchgeführt. Anschließend erfolgt eine Verifikation der Winkelauflösung. Auf Grund der hohen Anzahl von Raumwinkelkombinationen werden im Rahmen dieses Tests nur einige ausgewählt, so dass eine Tendenz sichtbar werden kann.

### 4.3.1 Funktionstest

Zum grundlegenden Test der Algorithmusfunktionalität wurden, wie in Abb. 4.4 dargestellt, vier verschiedene Schalleinfallsrichtungen eingestellt. Als Quellsignal diente eine Testdatei mit männlicher Sprache. Die Einstellungen



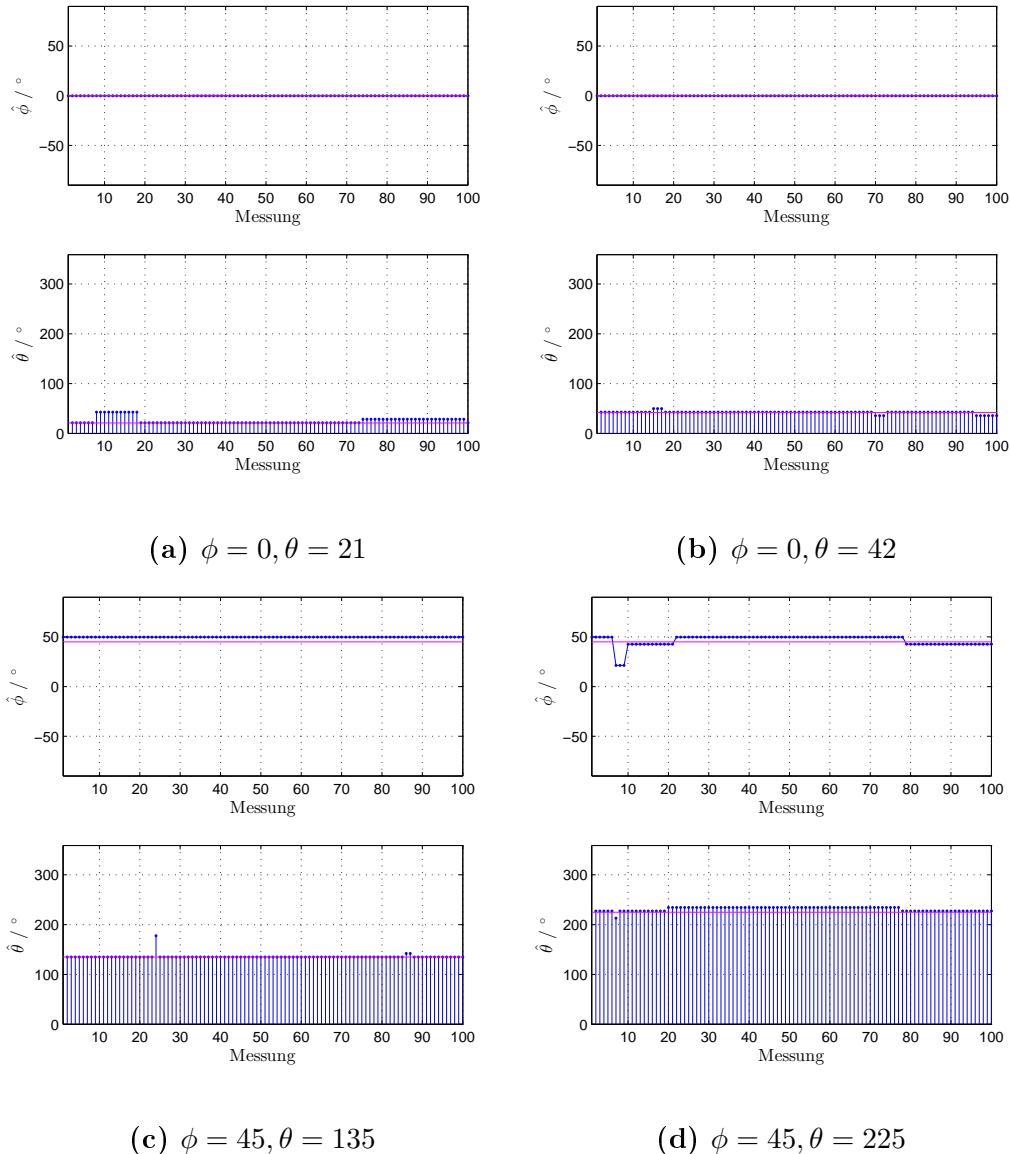
**Abb. 4.3:** Einstellung der theoretischen Winkel am Mikrofonarray

der Histogrammglättung wurden mit einer Ringspeicherlänge von 20 und einer Schwelle von 20% vorgenommen.

Es ist zunächst zu erkennen, dass der Algorithmus die Raumwinkel sicher detektiert. Auf Grund der endlich genauen Möglichkeit das Array zum Lautsprecher auszurichten, kann davon ausgegangen werden, dass Winkelfehler von einem Auflösungsschritt auf diese Einstellungstoleranz zurückzuführen sind. Die darüber hinaus auftretenden starken Winkelsprünge können den durch die Raumwände verursachten Reflexionen zugeschrieben werden.

### 4.3.2 Verifikation der Winkelauflösung

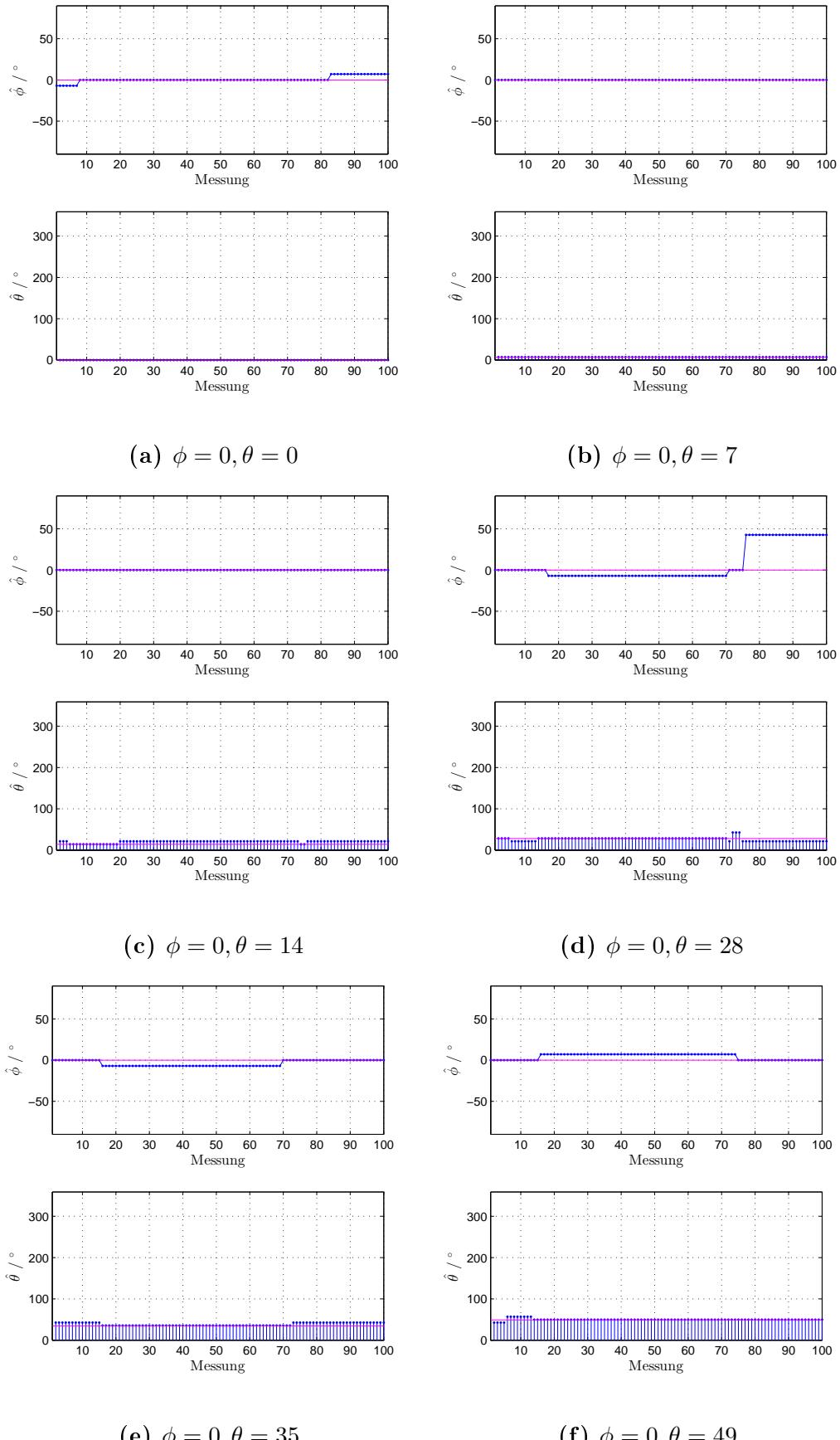
Zur Verifikation der Winkelauflösung wird das Mikrofonarray zunächst auf  $\phi = 0$  und  $\theta = 0$  ausgerichtet. Anschließend erfolgt eine Rotation um  $\theta$  in

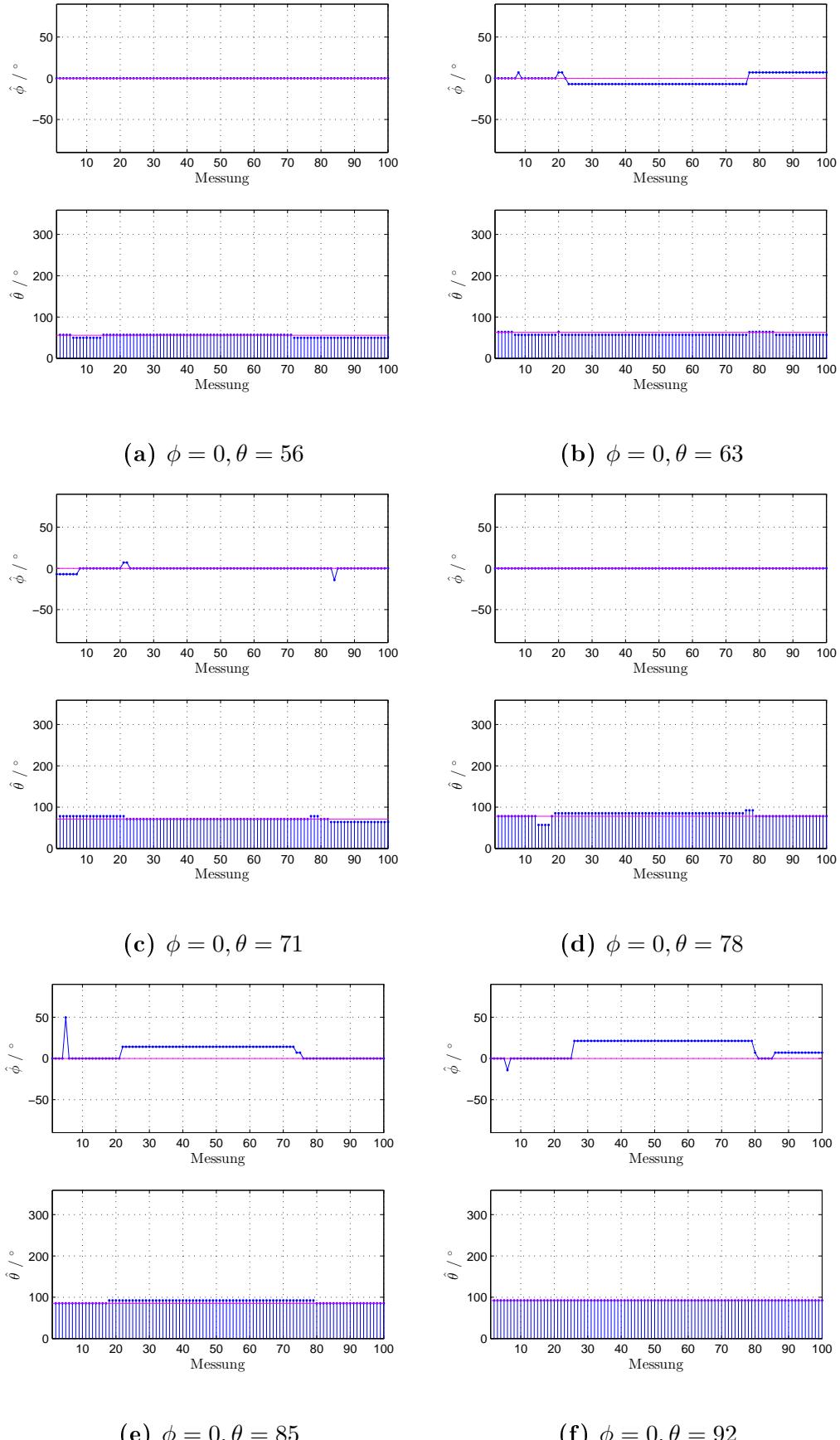


**Abb. 4.4:** Messung der Schalleinfallsrichtung aus unterschiedlichen Winkeln in einem hallenden Raum.

Abständen der Winkelauflösung. Der Winkel  $\theta$  wird dabei die Werte  $0 \leq \theta \leq 92$  annnehmen.

Abb. 4.5 und Abb. 4.6 illustrieren die Messergebnisse bei einer Anzahl von 100 Messungen pro Winkel. Hier wird ersichtlich, dass die Winkel im Abstand der in Abschnitt 2.5.1 beschriebenen Winkelauflösung eingenommen werden können. Die Abweichungen um einen Winkelschritt sind erneut auf die begrenzte Einstellgenauigkeit zurückzuführen. Auf Grund von Reflexion ergeben sich hier erneut vereinzelte Fehler in Form von abrupten Winkelsprüngen.

Abb. 4.5: Verifikation der Winkelauflösung im Bereich  $0 \leq \theta \leq 49$ .

**Abb. 4.6:** Verifikation der Winkelauflösung im Bereich  $56 \leq \theta \leq 92$ .

# Kapitel 5

## Zusammenfassung

In der vorliegenden Arbeit wurde auf Basis des Mehrkanal-Kreuzkorrelationskoeffizienten ein System entwickelt, das in der Lage ist, eine bewegte Sprachquelle innerhalb eines Raumes zu detektieren. Als grundlegende Voraussetzung galt hier die Anwendung der Model-Based-Development-Methode<sup>1</sup>, in der zunächst ein umfassendes mathematisches Modell entwickelt, simuliert und anschließend auf einer entsprechenden Hardware realisiert wird. Die gesamte Modellierung wurde dabei mit der Entwicklungsumgebung MATLAB® erstellt. Die anschließende Realisierung erfolgte auf Basis von Hardware der Firma Texas Instruments® sowie dessen Programmierumgebung Code Composer™ Studio.

Der hier implementierte Algorithmus lässt sich in folgende Teilschritte aufgliedern:

1. Energieberechnung des Eingangssignals zur Klassifizierung des Signalblocks (stimmhaft, stimmlos)
2. Kreuzkorrelation aller Eingangssignale unter Verwendung der von Texas Instruments® zur Verfügung gestellten DSP-Bibliothek
3. Verwendung der Korrelationsfunktionen zur räumlichen Suche/Detektion der Sprachquelle
4. Ergebnisglättung unter Verwendung eines Histogramms
5. Versenden der Ergebnisse mit Hilfe der serielle Schnittstelle

Zur Einstellung der benötigten Systemparameter wurden zunächst entsprechende theoretische Überlegungen getätigt. Beim anschließenden Echtzeittest

---

<sup>1</sup>Modellgetriebene Softwareentwicklung

wurde allerdings festgestellt, dass die Energieschwelle, die Länge des Histogrammspeichers und dessen Schwelle abhängig von den gegebenen Raummeigenschaften empirisch ermittelt werden mussten.

Neben der Entwicklung des Detektionsalgorithmus wurde darüber hinaus ein räumliches Mikrofonarray entworfen sowie gefertigt. Mit Hilfe des simplen Steckmechanismus konnte diese aufwandsarm sowie kostengünstig mit einer hohen Fertigungsgenaugkeit hergestellt werden. Des Weiteren ist es möglich, auf Grund des einfachen Klipp-Systems die empfindlichen Mikrofonkapseln der Forma Sennheiser® unkompliziert zu entfernen und anderweitig einzusetzen.

## 5.1 Bewertung der Ergebnisse

Zur Bewertung der erzielten Ergebnisse werden zunächst die in Kapitel 1 genannten Forderungen diskutiert.

**Verarbeitung von Sprachsignalen** - Der Algorithmus ist in der Lage, ein breitbandiges Sprachsignal zu verarbeiten da die Raumwinkel durch Analyse der Redundanz zwischen den einzelnen Mikrofonsignalen ermittelt werden. Auf Grund dessen ist das Schätzergebnis grundsätzlich unabhängig von den im Signal enthaltenen Frequenzen.

**Winkelauflösung** - Die Winkelauflösung ist lediglich abhängig von der Abtastfrequenz und kann im Hinblick auf das Auftreten des räumlichen Aliasing-Effekts nicht durch Vergrößerung der Array-Dimension erhöht werden. Auf Grund der rechenintensiven Operationen ist eine Erhöhung der Abtastfrequenz und somit der Winkelauflösung nur durch eine Erhöhung der zur Verfügung stehenden CPU-Taktschritte möglich. Diesbezüglich kann die hier erreichte Winkelauflösung durch ausreichend betrachtet werden. Während der Durchführung des Echtzeittests konnte jede Signalquelle auch bei kleinen Ortsänderungen verfolgt werden.

**Rechenzeit** - Auf Grund der durch die EDMA-Blocklänge sowie die durch die Abtastfrequenz festgelegten maximalen Rechendauer musste ein Verfahren entwickelt werden, das in der Lage ist, den Raum erst in groben und anschließend sukzessiv in immer kleiner werdenden Winkelschritten abzusuchen. Obwohl durch diese Methode das Echtzeitkriterium erzielt wird, ergeben sich fehlerhaft erkannte Winkel bis hin zu kleinen Bereiche, an denen keine Position erkannt werden kann. Eine weitere Optimierung des Systems müsste dann an dieser Stelle fortgeführt werden.

**Winkelraum** - Im Vergleich zu den Arbeiten [12] und [11], in denen auf Grund der Arraygeometrie nur ein eingeschränkter Winkelbereich abgedeckt werden konnte, erzielt das hier entwickelte System sehr gute Ergebnisse. Auf Grund der räumlichen Arraygeometrie kann jeder Raumwinkel im Raster der Winkelauflösung abgebildet werden.

Beim Betrachten des gesamten Projekts im Überblick wird ersichtlich, dass eine Methode gefunden wurde um mit Hilfe eines Acht-Kanal-Mikrofonarrays eine Sprachquelle im Raum zu detektieren. Die erzielten Messergebnisse des Echtzeitssystems in der ersten Version zeigen eine deutliche Tendenz. Zur Implementierung des Systems in einer realen Anwendung sind weiterführende Schritte notwendig, da hier die zwingende Notwendigkeit fehlerfreier Ergebnisse vorliegt.

## 5.2 Ausblick

Auf Grundlage dieser Arbeit könnte versucht werden, die Winkelauflösung durch sukzessive Signalinterpolation zu erhöhen. Dazu müsste zunächst die Schalleinfallsrichtung für eine fest eingestellte Winkelauflösung ermittelt werden. Anschließend könnte entsprechend dieser Quellenrichtung ein kürzeres Signalstück entnommen, interpoliert und erneut auf die Schalleinfallsrichtung untersucht werden.

Des Weiteren bestünde die Möglichkeit zur Untersuchung auf die Systemreaktion unter Verwendung der GCC-PHAT und deren Robustheit bei der Verwendung von Sprachsignalen.

Im Hinblick auf das realisierte Kreuzpeilungssystem in Müller [11] könnte dieses um eine Dimension erweitert werden und Signalquellen im Raum peilen.

# Literaturverzeichnis

- [1] BENESTY, Jacob ; CHEN, Jingdong ; HUANG, Yiteng: *Microphone Array Signal Processing (Springer Topics in Signal Processing)*. Softcover reprint of hardcover 1st ed. 2008. Springer, 11 2010. – URL <http://amazon.com/o/ASIN/3642097340/>. – ISBN 9783642097348
- [2] DMOCHOWSKI, J. ; BENESTY, J. ; AFFES, S.: Direction of Arrival Estimation Using the Parameterized Spatial Correlation Matrix. In: *Audio, Speech, and Language Processing, IEEE Transactions on* 15 (2007), Nr. 4, S. 1327–1339. – ISSN 1558-7916
- [3] D.SIGNT: *D.Module.PCM3003*. Revision 1.1. -: D.SignT (Veranst.), November 2011
- [4] D.SIGNT: *D.Module.C6713*. Revision 1.5. -: D.SignT (Veranst.), Oktober 2012
- [5] GÖRNE, Thomas ; 1 (Hrsg.): *Tontechnik: Schwingungen und Wellen, Hören, Schallwandler, Impulsantwort, Faltung, Sigma-Delta-Wandler, Stereo, Surround, WFS, Regiegeräte, tontechnische Praxis*. 1. Carl Hanser Verlag GmbH, 8 2006. – URL <http://amazon.de/o/ASIN/3446401989/>. – ISBN 9783446401983
- [6] GRADKOWSKI, Wojciech: *JMatIO - Matlab's MAT-file I/O in JAVA*. April 2006. – URL <http://www.mathworks.com/matlabcentral/fileexchange/10759-jmatio-matlabs-mat-file-io-in-java>. – Zugriffsdatum: 18.07.2013
- [7] KAMMEYER, Karl D.: *Nachrichtenübertragung*. Teubner B.G. GmbH, 2008. – URL <http://amazon.com/o/ASIN/383510179X/>. – ISBN 9783835101791
- [8] KAMMEYER, Karl-Dirk ; KROSCHEL, Kristian: *Digitale Signalverarbeitung: Filterung und Spektralanalyse mit MATLAB®-Übungen (German Edition)*. 1. Springer, 11 2010. – URL <http://amazon.com/o/ASIN/3642097340/>. – ISBN 9783642097348

- man Edition). 7., erw. und korrigierte Aufl. 2009. Vieweg+Teubner Verlag, 2 2009. – URL <http://amazon.com/o/ASIN/3834806102/>. – ISBN 3834806102*
- [9] KLEMENZ, Adolf: *PCM3003 Demo Program.* -: , 2010
  - [10] KÖLZER, Prof. Dr.-Ing. H. P.: *Digitale Bildverarbeitung und Mustererkennung. Image processing.* -: HAW-Hamburg (Veranst.), 2010
  - [11] MÜLLER, Markus: *Sprecherlokalisierung von zwei oder mehr Sprechern im Raum unter Verwendung von Mikrofonarrays,* HAW-Hamburg, Master Thesis, 2012
  - [12] PIKORA, Kolja: *Korrelationsalgorithmen zur Sprecherlokalisierung mittels DSP-basiertem Echtzeitsystem und Mikrofonarray,* HAW-Hamburg, Master Thesis, 2012
  - [13] PIZIAK, Robert ; ODELL, P.L.: *Matrix Theory: From Generalized Inverses to Jordan Form (Pure and Applied Mathematics: A Program of Monographs and Textbooks).* 1. Chapman and Hall/CRC, 2 2007. – URL <http://amazon.com/o/ASIN/1584886250/>. – ISBN 9781584886259
  - [14] PROF. DR.-ING. H. P. KÖLZER, Prof. Dr.-Ing. U. S.: *Fast Fourier Transform (FFT).* Folien zur Vorlesung. -: HAW-Hamburg (Veranst.), März 2012
  - [15] RUDD, Kevin: *SoundSim 2.5D Acoustic Wave and Sphere Scattering Simulator.* August 2006. – URL <http://www.mathworks.com/matlabcentral/fileexchange/12097-soundsim-2-5d-acoustic-wave-and-sphere-scattering-simulator> – Zugriffsdatum: 14.06.2013
  - [16] SHAKU ANJANAIAH, Brad C.: *TMS320C6000 McBSP Initialization.* SPRA488C. Post Office Box 655303 Dallas, Texas 75265: Texas Instruments (Veranst.), März 2004
  - [17] Texas Instruments (Veranst.): *TMS320C6713B Floating-Point Digital Signal Processor.* SPRS294B. Juni 2006
  - [18] Texas Instruments (Veranst.): *TMS320C67x DSP Library Programmer's Reference Guide.* Januar 2010
  - [19] Texas Instruments (Veranst.): *TMS320C6000 Optimization Workshop - Lab Exercises.* Revision 1.51. März 2011

- [20] VARMA, Krishnaraj: *Time-Delay-Estimate Based Direction-of-Arrival Estimation for Speech in Reverberant Environments*, The Bradley Department of Electrical and Computer Engineering Virginia Polytechnic Institute and State University, Diplomarbeit, October 2002
- [21] WERNER, Martin: *Signale und Systeme: Lehr- und Arbeitsbuch mit MATLAB®-Übungen und Lösungen (German Edition)*. 3., vollst. überarb. und erw. Aufl. 2008. Vieweg+Teubner Verlag, 7 2008. – URL <http://amazon.com/o/ASIN/3834802336/>. – ISBN 9783834802330