Bachelorthesis

Sönke Christoph Wilhelm Appel

Akustische Raumsimulation mit MATLAB für die Entwicklung von Schallquellen-Lokalisationen mit unterraumbasierten Verfahren

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Sönke Christoph Wilhelm Appel

Akustische Raumsimulation mit MATLAB für die Entwicklung von Schallquellen-Lokalisationen mit unterraumbasierten Verfahren

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.Ing. Hans Peter Kölzer
Zweitgutachter : Prof. Dr.Ing. Karin Landenfeld

Abgegeben am 26. Februar 2010

Sönke Christoph Wilhelm Appel

Thema der Bachelorthesis

Akustische Raumsimulation mit MATLAB für die Entwicklung von Schallquellen-Lokalisationen mit unterraumbasierten Verfahren

Stichworte

Mikrofonarray, Unterraumverfahren, Raumsimulation, Schallreflexionen, Spiegelquellenverfahren, MATLAB

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung von einem akustischen Raumsimulationsprogramm mit MATLAB. Das Programm soll für die Entwicklung von Schallquellen-Lokalisationen verwendet werden, speziell für Schallquellen-Lokalisationen, die mit unterraumbasierten Verfahren arbeiten.

Sönke Christoph Wilhelm Appel

Title of the paper

Development of an acoustic room-simulation for localisation of sound sources with subspace-algorithms using MATLAB

Keywords

Microphon Array, Subspace Algorithm, Room Simulation, Sound reflection, groundplane method, MATLAB

Abstract

This Thesis deals with development of an acoustic room-simulation-program using MATLAB. The program shall be used for development of localisation of sound sources, especially localisation of sound sources working with subspace allgorithms.

Danksagung

Als erstes möchte ich meiner Mutter Renate Appel und meinem Vater Bernd Appel danken, dass Sie meinen Entschluss unterstützt haben, nach meiner Lehre ein Studium über den zweiten Bildungsweg zu beginnen.

Dank an meine Wohngemeinschaft Enno Putzar, Wiebke Heinz und Hannah Athmer, die mir eine Basis während der Zeit meiner Bachelorarbeit waren und somit mir die Möglichkeit gaben, nach einem arbeitsreichen Tag zu Hause, zu Entspannung und Ablenkung zu gelangen.

Ganz besonders Danken möchte ich Herrn Prof. Dr.-Ing. Kölzer, meinem betreuenden Professor und Erstprüfer, der mich bei meiner Arbeit stark unterstützt hat, und Frau Prof. Dr.-Ing. Landenfeld für ihre Bereitschaft, meine Arbeit als Zweitprüferin zu übernehmen.

Inhaltsverzeichnis

1. Motivation

Die Motivation für diese Bachelorarbeit resultiert aus der Überlegung von Studenten, ein System zu entwickeln, mit welchem es möglich ist, Vorlesungen wiederholt zu hören ohne ein halbes Jahr warten zu müssen. Dazu wurden grobe Überlegungen vorgenommen, die im Folgenden dargelegt werden. Diese Thesis ist ein Teil des Gesamtprojekts.

1.1. Die Vision

Zum Auf- und Nacharbeiten einer Vorlesung besteht für Studenten derzeit nur die Möglichkeit, sich an die Aufzeichnungen ihrer Kommilitonen zu halten, das Thema in einem Buch oder im Skript des Professors nachzulesen. Eine deutliche Verbesserung und Vereinfachung könnte durch die Bereitstellung von Videos der Vorlesungen im Internet erzielt werden. Um die Vorlesung in ihrer Gänze wahrnehmen zu können, ist es notwendig, dass alle Einwürfe und Nachfragen der Studenten aufgenommen werden, also eine interaktive Videoaufzeichnug stattfindet. Kostentechnisch realisierbar ist diese Vision auch nur, wenn die Videoaufzeichnung vollautomatisch ohne Kameramann funktioniert.

1.2. Das Gesamtprojekt

Eine vollautomatische interaktive Videoaufzeichnung der Vorlesung soll das Ziel mehrerer Bachelor und Masterarbeiten sein. Ohne Kameramann, ohne Mikrofon in der Hand und das interaktiv. Störgeräusche wie Klimaanlage, Straßenlärm sollen aus der Tonspur herausgefiltert werden. Der Professor wird von der Kamera verfolgt. Geregelt und aufgenommen wird dies über *Mikrofonarrays*.

1.3. Gesamtprojektunterteilung für eine vollautomatische interaktive Videoaufzeichnung von Vorlesungen

In der Abbildung 1.1 ist eine Projektunterteilung dargestellt. Aus der Abbildung ist zu lesen welche Unterprojekte möglich sind und wie die Unterprojekte miteinander verknüpft sind. Rötlich hinterlegt ist der Teil des Gesamtprojekts, welcher Thema dieser Bachelorarbeit ist.

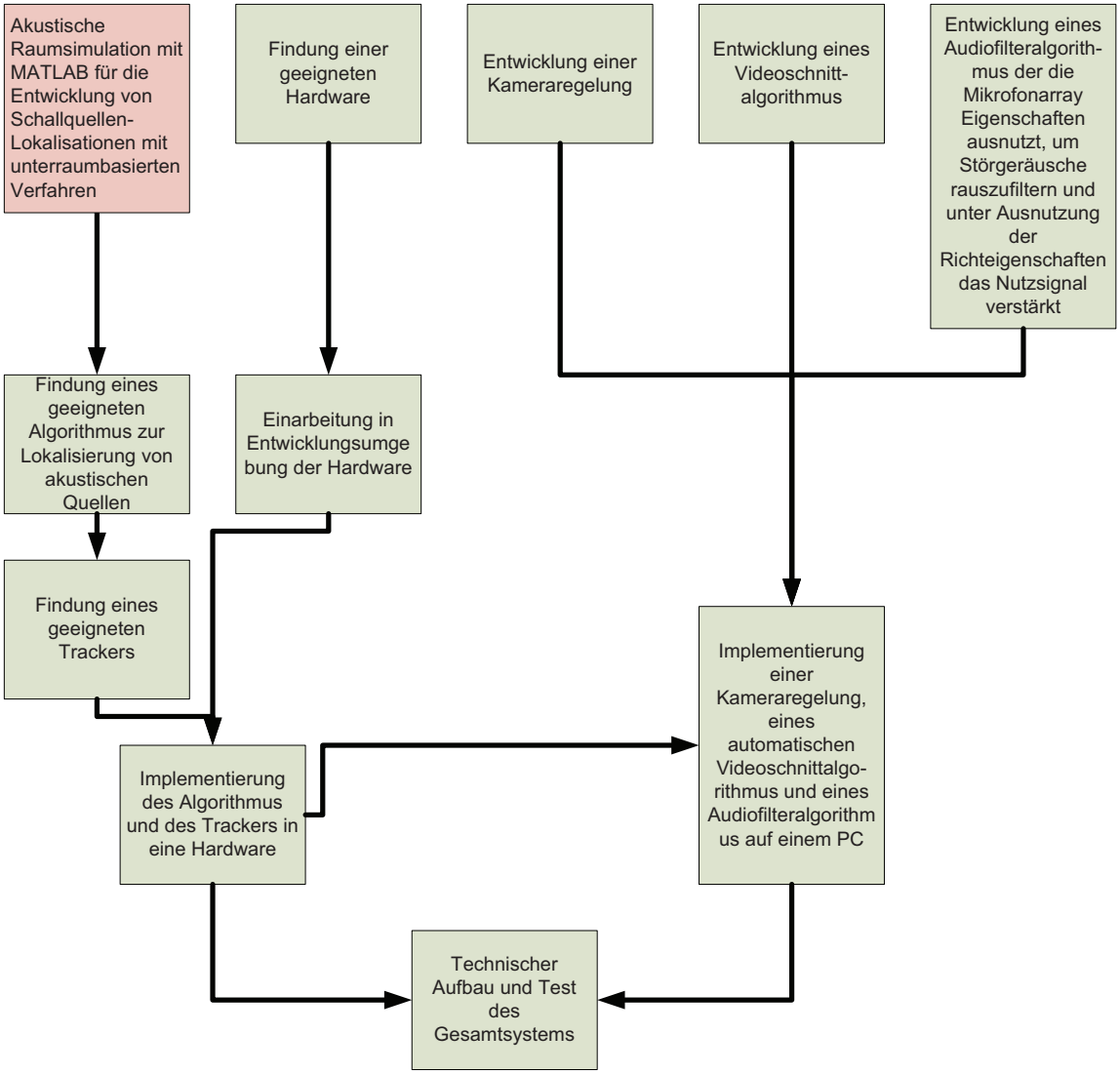


Abbildung 1.1.: Projektplan für die Entwicklung einer vollautomatischen interaktiven Video-aufzeichnung von Vorlesungen

1.4. Notwendigkeit einer Simulationssoftware

Zum Thema Mikrofonarray gibt es schon Diplom und Masterarbeiten. Leider konnten sich die Autoren wenig mit der Optimierung der Algorithmen und der Tracker befassen, da sie sehr viel Zeit in die Programmierung des neusten DSPs und damit in immer wieder andere Hardware und deren Entwicklungsumgebungen, investieren mussten. Sinnvoller wäre es, wenn die Studenten ihre Zeit unabhängig von der Hardware in die Algorithmen und Tracker investieren könnten.

Aus diesem Grund befasst sich diese Thesis mit der Entwicklung einer akustischen Raumsimulation für Mikrofonarrays unter MATLAB, mit dessen Hilfe die Studenten ihre Algorithmen und Tracker testen können.

An dieser Stelle ist festzustellen, ob bereits eine geeignete Simulationssoftware im Zuge einer Thesis oder anderweitig entwickelt wurde. Bei der Recherche zu dieser Bachelorthesis wurden eingige Simulationsprogramme gefunden. Jedoch war keine Software frei zugänglich und es wurde kein Simulationsprogramm gefunden, welches sich auch für die Anforderung von *Unterraumverfahren* eignet.

1.5. Anforderungen an eine Simulationssoftware für Unterraumverfahren

- Die Laufzeitenunterschiede zwischen den einzelnen Mikrofonen müssen genau ermittelt werden. Dies gilt besonders bei der Verwendung von Algorithmen mit Unterraumverfahren, da bei Verwendung dieser Algorithmen die Mikrofone mit einem Abstand von nur wenigen Zentimetern zueinander angeordnet werden.
- Es muß möglich sein, mehrere Audioquellen gleichzeitig einzuspielen, um auch Störquellen wie Klimaanlage, Straßenlärm und Zwischenrufe simulieren zu können. Außerdem sollten auch Algorithmen getestet werden können, welche mehrere Sprecher unterscheiden können.
- Die Symmetrie der Mikrofonanordnung muss frei gestaltbar sein.
- Bewegte Quellen sollten auch simulierbar sein, da das Simulationsprogramm auch zur Verbesserung von Trackern verwendet werden soll.
- Reflexionen an Wänden und Decken müssen berücksichtigt werden, da diese ein Hauptproblem bei der Umsetzung in einer realen Umgebung sind.

- Verschiedene Beschaffenheiten von Wänden, Decken und Böden sollten berücksichtigt werden, da diese großen Einfluss auf die Zuverlässigkeit der Algorithmen haben.

1.6. Eingrenzung des Projektumfanges

Für die Eingrenzung des Projektumfanges wurde eine grobe Programmstrukturierung erstellt, die in Abbildung 1.2 abgebildet ist.

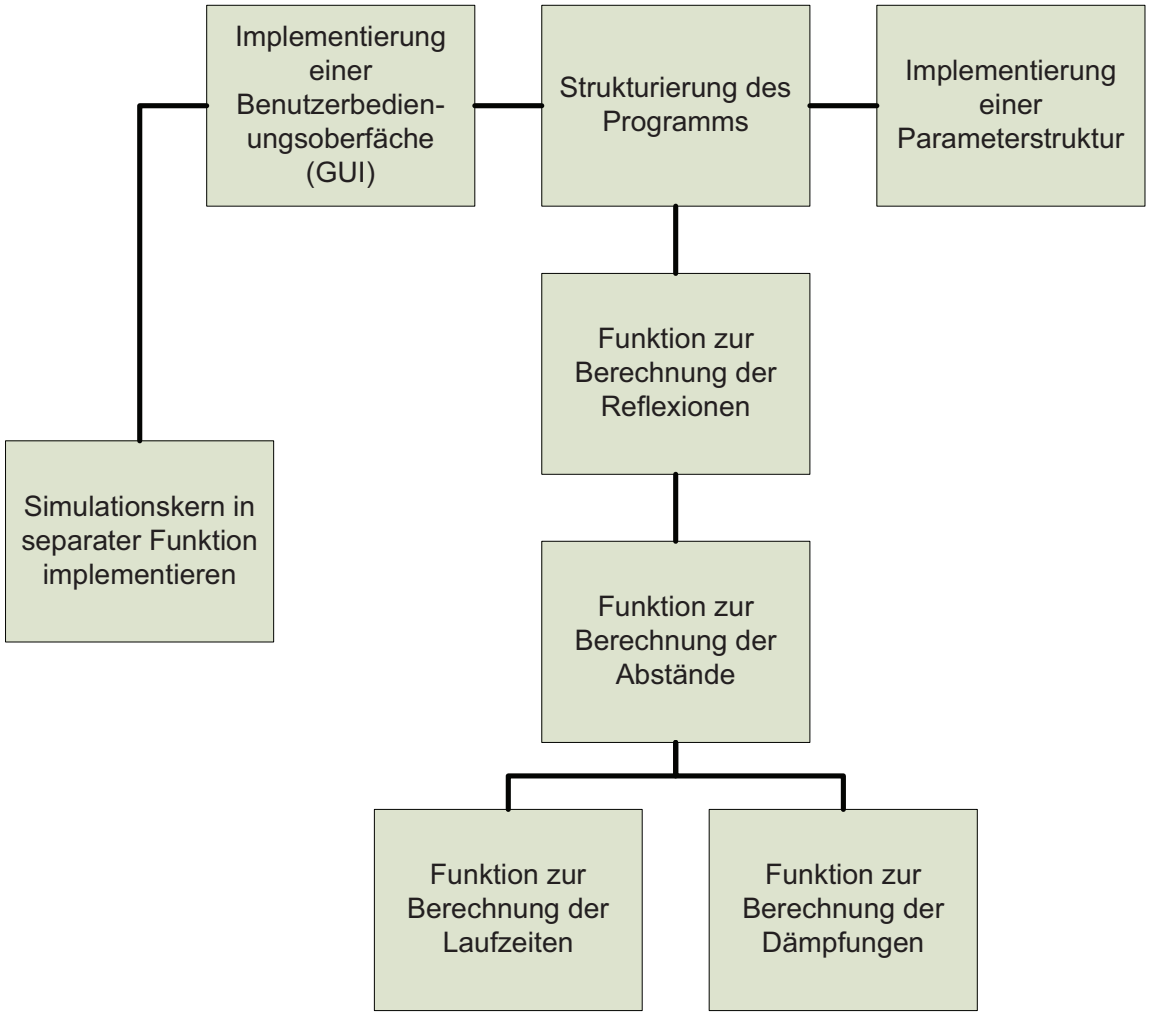


Abbildung 1.2.: Programmstrukturierung für die Entwicklung eines akustischen Raumsimulationsprogrammes

2. Allgemeiner Aufbau und Programmstruktur

Im Simulationsprogramm sind zwei Teile elementar für die Berechnung. Im ersten Teil werden die für die Simulation notwendigen Parameter aus den Koordinaten des Simualtionsaufbaus berechnet. Dies geschieht unter MATLAB in einem 3-D-Array. Die Ergebnisse der Parameterberechnung werden für die Simulation verwendet. Im zweiten Schritt werden mit Hilfe der ermittelten Parameter der Raum akustisch simuliert. Dies kann beispielsweise in einer SIMULINK- Simulation geschehen, in welcher die ermittelten Parameter in Verzögerungsglieder und Dämpfungsglieder überführt werden oder auch unter MATLAB in einer Funktion übernommen werden.

Um die Parameter frei und komfortabel einstellen zu können, wurde eine graphische Oberfläche mit dem MATLAB-Tool *GUIDE* implementiert. Über die graphische Oberfläche lassen sich die Raumabmessungen, Mikrofon- und Audioquellenpositionen, sowie die Simulationsfrequenz einstellen. Es können Datensätze geladen und gespeichert werden. Audiodateien können für die Simulation geladen werden. Die graphische Oberfläche dient dabei der Übersichtlichkeit und Bedienfreundlichkeit und ist im Anhang B auf Seite 120 dargestellt.

Von der graphischen Oberfläche aus werden sowohl die Parameterberechnungen, als auch die Raumsimulation gestartet.

In Abbildung 2.1 wird dargestellt welche externen selbst erstellten Funktionen von der graphischen Oberfläche aus aufgerufen werden. Dabei sind alle Funktionen, die direkt die Parameter und die Raumsimulation berechnen, in Schwarz gekennzeichnet. Alle Funktionen die zur Handhabung des Programms zuständig sind, sind heller dargestellt.

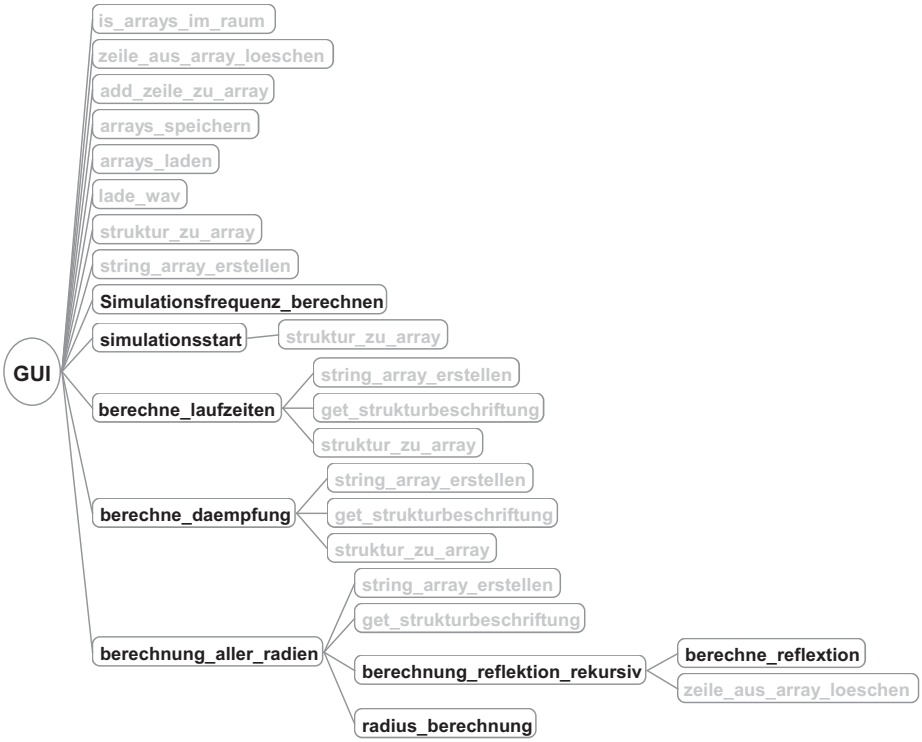


Abbildung 2.1.: Funktionsaufrufe aus der graphischen Oberfläche

Die *Callbacks* zur Bedienung der GUI und *CreateFunktion* zur Erstellung der GUI sind in Abbildung 2.1 nicht dargestellt. Im Anhang A wird das Listing A.1 der GUI aufgeführt. Erwähnenswert ist die Funktion *update_fig_uitable()*, die bei einer Veränderung eines Wertes die Tabellen und Graphiken aktualisiert. Auf diese Funktion wird im Kapitel 3 genauer eingegangen.

In Abbildung 2.2 auf Seite 13 wird eine Radienstruktur dargestellt, wie sie im Simulationsprogramm angelegt wird. Die Strukturen für die Dämpfungen und die Laufzeiten sind gleich strukturiert. Diese Strukturen wurden angelegt, damit die ermittelten Daten klar zuzuordnen sind. Alternativ hätte man die Daten in ein dreidimensionales Array schreiben können. Dies hätte aber zu großer Unübersichtlichkeit beim Lesen und Debugging des Quellcodes geführt.

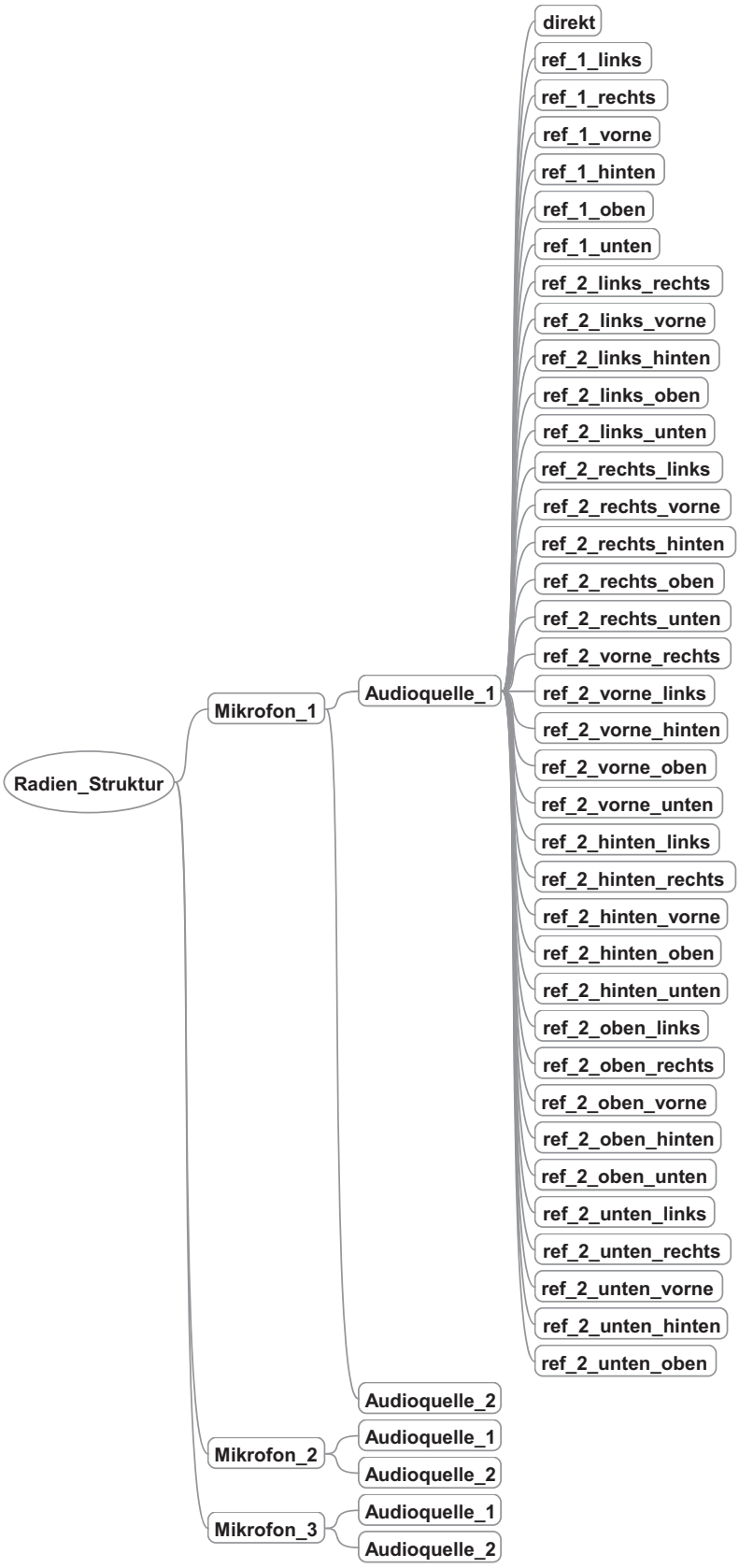


Abbildung 2.2.: Struktur der Radiendatensätze

3. Funktionen zur Handhabung des Programms

In diesem Kapitel werden alle Funktionen aufgeführt, die zur Organisation des Simulationsprogrammes verwendet werden. Dabei handelt es sich um Funktionen, die die Daten speichern und laden, Strukturen erstellen und verarbeiten oder auch Datenblöcke verarbeiten.

3.1. Lade und Speichervorgänge

Damit der Anwender verschiedene Anordnungen von Mikrofonen und Audioquellen, sowie die Raumdimensionierung nicht bei jeder Simulation neu eingeben muss, wurde eine Lade- und Speicher-Funktion implementiert. Die Funktionen *arrays_laden()* und *arrays_speichern()* starten einen sogenannten *Standarddialog*, der ein Ladefenster bzw. ein Speicherfenster öffnet. Das Listing 3.1 zum Laden der Anordnungsstrukturen ist im Folgenden aufgeführt. Nachdem der Standarddialog aufgerufen wird, in dem der Anwender eine Datei auswählen kann, wird die Datei geladen. Dabei wird überprüft, ob es sich um den richtigen Datentyp handelt und ob die Datei richtig geladen wurde. Zurückgegeben werden Raumdimensionen und die zwei 2D- Arrays für Mikrofonkoordinaten, Audioquellenkoordinaten.

Listing 3.1: Laden von Mikrofon- und Audioqellenanordnung

```
1 function [micarr audarr raum erfolgreich] = arrays_laden()
2 %Rückgabewerte:
3 %micarr = Die Koordinatenmatrix für alle Mikrofone
4 %audarr = Die Koordinatenmatrix für alle Audioquellen
5 %raum = Raumabmessungen
6 %erfolgreich = gibt an, ob der Ladevorgang erfolgreich war
7
8 %uigetfile ist ein Standarddialog zum Laden von Dateien. Durch
     diesen Befehl wird ein Fenster ausgewählt, mit dem der User
     eine Datei auswählen kann, welche er laden möchte.
     Zurückgegeben wird der Pfad und der Filename
9 [fname, pname] = uigetfile('*.mat', 'Datei␣Laden'); %Nur Daten mit
      der Endung .mat sind auswählbar
```

```
10 if fname ~= 0   %Abfrage: Wenn eine Datei ausgewählt wurde, dann
11     komplettname = [pname,fname];   %Fügt den Pfad und Filename
           zusammen
12     daten = load(komplettname); %Laden der Daten mit Load
13     %Es wird der Datenblock untersucht, ob die Datenfeldnamen
           micarr, audarr und raum im Datenblock vorhanden sind:
14     if isfield(daten, 'micarr') && isfield(daten, 'audarr') &&
           isfield(daten, 'raum')
15         micarr = daten.micarr;  %Aufspalten des Datenblocks in ein
               Mikrofonkoordinatenarray
16         audarr = daten.audarr;  %Aufspalten des Datenblocks in ein
               Audiokoordinatenarray
17         raum = daten.raum;      %Aufspalten des Datenblocks in die
               Raumabmessung
18         erfolgreich = 1;        %Rückmeldung, dass das Laden der
               Datei erfolgreich war
19     else
20         h = MsgBox('Datei vom falschen Typ', 'Fehler', 'modal');
               %Messagebox
21         uiwait(h);  %Warten bis die Messagebox bestätigt wurde
22         micarr = NaN;   %micarr mit NaN kennzeichnen
23         audarr = NaN;   %audarr mit NaN kennzeichnen
24         raum = NaN;     %raum mit NaN kennzeichnen
25         erfolgreich = 0;    %Rückmeldung, dass das Laden der Datei
               nicht erfolgreich war
26     end
27 else
28     h = errordlg('Keine Datei ausgewählt', 'Fehler', 'modal');  %
           Messagebox
29     uiwait(h);  %Warten bis die Messagebox bestätigt wurde
30     micarr = NaN;   %micarr mit NaN kennzeichnen
31     audarr = NaN;   %audarr mit NaN kennzeichnen
32     raum = NaN;     %raum mit NaN kennzeichnen
33     erfolgreich = 0;    %Rückmeldung, dass das Laden der Datei
           nicht erfolgreich war
34 end;
```

Das Listing 3.2 zum Speichern der Anordnungsstruktur verwendet ebenfalls einen Standarddialog, jedoch wird zuerst aus den 2D- Arrays eine Datenstruktur erzeugt. Nachdem der Anwender über den Standarddialog einen Pfad und einen Dateinamen ausgewählt hat, werden die Daten gespeichert. Dabei wird festgestellt, ob der Vorgang erfolgreich abgeschlossen wurde.

Listing 3.2: Speichern von Mikrofon- und Audioquellenanordnung

```
function erfolgreich = arrays_speichern(micarr, audarr, raum)
%Übergabeparameter:
%micarr = Mikrofonkoordinatenarray
%audarr = Audiokoordinatenarray
%raum = Raumabmessung
%
%Rückgabewert:
%erfolgreich = gibt eine Rückmeldung, ob der Speichervorgang
   erfolgreich war

%Die übergebenen Daten werden in eine Struktur verpackt
daten = struct('micarr',micarr,'audarr',audarr,'raum',raum);

%Die Funktion uiputfile() ist ein Standarddialog mit dem der User
   einen Pfad und einen Filename auswählt unter welchem die Daten
   gespeichert werden sollen
[fname, pname] = uiputfile('*.mat', 'Datei␣speichern␣unter');   %
   Endung der Datei wird auf .mat festgelegt
if fname ~= 0   %Abfrage: Ob ein Dateiname ausgewählt wurde
    komplettname = [pname,fname];   %Zusammenfügen des Pfades und
       Filename
    save(komplettname, '-struct', 'daten'); %Abspeichern der Daten
    erfolgreich = 1;    %Rückmeldung, dass die Speicherung
       erfolgreich war
else
    h = errordlg('Keine␣Datei␣ausgewählt', 'Fehler', 'modal');  %
       Messagebox
    uiwait(h);  %Abwarten, bis die Messagebox vom User zur
       Kenntnis genommen wurde
    erfolgreich = 0;    %Rückmeldung, dass das Speichern der Daten
        nicht erfolgreich war
end;
```

Neben dem Laden und dem Speichern von Anordnungsstrukturen wurde eine Funktion zum Laden von Wave-Dateien implementiert 3.3. Dieser Funktion wird ein Beschriftungsstring übergeben in dem steht, welche Audioquelle geladen werden soll. Auch wird hier nach dem Standarddialog der Ladevorgang gestartet und eventuell auftretende Fehler werden abgefangen.

Listing 3.3: Laden von Wave-Dateien

```
function [Datei] = lade_wav(lade_dialog_beschriftung)

```

```
3 dialog_beschriftung_string = ['Lade␣WAV-Datei␣für␣',char(
    lade_dialog_beschriftung)];
4
5 [fname,pname] = uigetfile('*.wav',dialog_beschriftung_string);
6 if fname ~= 0
7     komplettname = [pname,fname];
8     Datei = komplettname;
9 else
10    h = errordlg('Keine␣Datei␣ausgewählt', 'Fehler', 'modal');
11    uiwait(h);
12    Datei = 0;
13 end;
```

3.2. Handhabung der Strukturen

Wie im Kapitel 2 bereits erwähnt und in Abbildung 2.2 dargestellt, wird für das Handhaben der Daten eine selbst angefertigte Struktur verwendet. Zur einfachen Handhabung der Struktur wurden drei Funktionen geschrieben. In der Funktion 3.4 wird ein Cellarray erstellt, das zur Beschriftung der Anordnungsstruktur dient. Außerdem wird diese Funktion zum Beschriften von Tabellen verwendet.

Listing 3.4: Funktion zum erstellen von Cellarrays

```
1  function [str] = string_array_erstellen(laenge, name)
2  % Rückgabewert(str): Ein Cellarray mit name1_...name_n
3  %Übergabeparameter(laenge): Laenge des Cellarrays
4  %Übergabeparameter(name): Name der Beschriftung z.B.: Audioquelle
       oder Mikrofon
5  str = [name '_' num2str(1)]; %str wird vorinitialisiert
6  str = {str}; %str wird in eine Cell_Struktur umgewandelt
7  for i=2:laenge %Von 2 bis Länge des Cellarrays
8      element = [name '_' num2str(i)]; %Cellarray-Element erstellen
9      str(i) = {element}; %Cellarray-Element in ein Cell umwandeln
           und dem Cellarray anhängen
10 end
```

In der Funktion 3.5 ist ein Beschriftungs-Cellarray hinterlegt.

Listing 3.5: Funktion in der ein Beschriftungs-Cellarray hinterlegt ist

```
1 function [beschriftung] = get_strukturbeschriftung()
```

```
beschriftung = {'direkt','ref_1_links','ref_1_rechts','ref_1_vorn'
    ,'ref_1_hinten','ref_1_oben','ref_1_unten','ref_2_links_rechts'
    ,'ref_2_links_vor','ref_2_links_hinten','ref_2_links_oben','
    ref_2_links_unten','ref_2_rechts_links','ref_2_rechts_vorn','
    ref_2_rechts_hinten','ref_2_rechts_oben','ref_2_rechts_unten','
    ref_2_vorn_links','ref_2_vorn_rechts','ref_2_vorn_hinten','
    ref_2_vorn_oben','ref_2_vorn_unten','ref_2_hinten_links','
    ref_2_hinten_rechts','ref_2_hinten_vorn','ref_2_hinten_oben','
    ref_2_hinten_unten','ref_2_oben_links','ref_2_oben_rechts','
    ref_2_oben_vorn','ref_2_oben_hinten','ref_2_oben_unten','
    ref_2_unten_links','ref_2_unten_rechts','ref_2_unten_vorn','
    ref_2_unten_hinten','ref_2_unten_oben'};
```

In der Funktion 3.6 wird eine übergebene Struktur in ein 2D-Array und zwei Beschriftungs-Cellarrays zerlegt.

Listing 3.6: Funktion zum Zerlegen von einer Anordnugsstruktur

```
function [datenblock zeilennamen spaltennamen] = struktur_zu_array
    (struktur)
%Rückgabewert(datenblock) : 2D- Array mit den in der übergebenen
    Struktur enthaltenen Werten
%Rückgabewert(zeilenname) : Zeilennamen der übergebenen Struktur
%Rückgabewert(spaltenname) : Spaltennamen der übergebenen Struktur
%Übergabeparameter(struktur) : Struktur, die in ein 2D Array und
    zwei Beschriftungs-Cellarrays zerlegt werden soll

zeilennamen = fieldnames(struktur); %Zeilennamen werden aus der
    Struktur gelesen
spaltennamen = fieldnames(getfield(struktur,char(zeilennamen(1))))
    ; %Spaltennamen werden aus der ersten Zeilenstruktur gelesen
for i=1:length(zeilennamen) %Von 1 bis Zeilenlänge
    for j=1:length(spaltennamen) %Von 1 bis Spaltenlänge
        datenblock(i,j) = getfield(struktur, char(zeilennamen(i)),
            char(spaltennamen(j))); %Elemente i und j werden bis n
            und m aus der Struktur gelesen und in ein 2D-Array
           geschrieben
    end
end
```

3.3. Funktionen zur Handhabung von Arrays

Weitere drei Funktionen sind zur Handhabung der Daten-Arrays implementiert worden. Funktion 3.7 fügt eine Zeile einem Array hinzu, dabei wird überprüft, ob im Array schon mindestens eine Zeile ist.

Listing 3.7: Funktion um einem Array eine Zeile hinzuzufügen

```
function array = add_zeile_zu_array(array, neue_zeile)
%Diese Funktion fügt eine Zeile an ein Array an
%Übergabeparameter array ist ein Koordinatenvektorarray
%Übergabeparameter Zeile ist ein Koordinatenvektor
if isnan(array) %Abfrage: Ist das übergebene Array keine Zahl
    array = neue_zeile; %Ersetze Arrayinhalt durch Zeile
else
    [anzahl_zeilen anzahl_spalten] = size(array);   %Anzahl der
       Zeilen des Arrays
    array(anzahl_zeilen+1,:) = neue_zeile;  %Hängt die Zeile an
       das Ende des Arrays an
end
```

Funktion 3.8 löscht eine bestimmte Zeile aus einem Array. Ist nur noch eine Zeile vorhanden wird auf das Array *NaN* zugewiesen.

Listing 3.8: Funktion um eine bestimmte Zeile aus einem Array zu löschen

```
function [array] = zeile_aus_array_loeschen(array, zeile_loeschen)
%Rückgabewert(array): neues um zeile gekürztes Array
%Übergabeparameter(array): 2D-Array, dem eine Zeile entfernt
   werden soll
%Übergabeparameter(zeile_loeschen): Nummer der Zeile, die gelöscht
    werden soll
[anzahl_zeilen anzahl_spalten] = size(array); %ermitteln wie viele
    Zeilen das Array hat
if anzahl_zeilen>1 %wenn die Anzahl der Zeilen größer als 1 ist,
   dann
    for i=zeile_loeschen:anzahl_zeilen-1 %Von der Zeile an, die
       gelöscht werden soll bis Anzahl der Zeilen-1
        array(i,:) = array(i+1,:); %Überschreiben der Array-Zeilen
    end
    for i=1:anzahl_zeilen-1
        array_z(i,:) = array(i,:); %Kopieren des Arrays, bis auf
           die letzte Zeile
    end
    array=array_z; %Zurückschreiben des Arrays
```

```
14 else
15     array=NaN; %Array keiner Nummer zuweisen
16 end
```

Funktion 3.9 stellt fest, ob alle eingegebenen Mikrofon- und Audioquellen-Koordinaten in die Raumdimensionierung passen. Dabei wird darauf geachtet, dass kein Fehler entsteht, wenn einem Array der Wert *NaN* zugewiesen ist. Die Überprüfung wird mit dem Rückgabewert an die GUI zurückgegeben.

Listing 3.9: Funktion um festzustellen ob alle Mikrofone und Audioquellen im Raum sind

```
1  function [erfolgreich] = is_arrays_im_raum(micarr, audarr, raum)
2  %Rückgabewert(erfolgreich): Rückmeldung der Funktion, ob alle
       Koordinaten im Raum sind
3  %Übergabeparameter(micarr): Koordinaten der Mikrofone
4  %Übergabeparameter(audarr): Koordinaten der Audioquellen
5  %Übergabeparameter(raum): Abmessungen des Raumes
6  if size(micarr)>1 %hat micarr mehr als eine Zeile, dann
7      if size(audarr)>1 %hat audarr mehr als eine Zeile, dann
8          max_mic = max(micarr); %Die Maximalwerte aus micarr werden
               max_mic zugewiesen
9          max_aud = max(audarr); %Die Maximalwerte aus audarr werden
               max_aud zugewiesen
10         max_array = [max_mic; max_aud]; %Die beiden Maximalzeilen
               werden max_array zugewiesen
11     else %hat audarr nur eine Zeile, dann
12         max_mic = max(micarr); %Die Maximalwerte aus micarr werden
               max_mic zugewiesen
13         max_array = [max_mic; audarr]; %Die beiden Maximalzeilen
               werden max_array zugewiesen
14     end
15 else %hat micarr nur eine Zeile, dann
16     if size(audarr)>1 %hat audarr mehr als eine Zeile, dann
17         max_aud = max(audarr); %Die Maximalwerte aus audarr werden
               max_aud zugewiesen
18         max_array = [micarr; max_aud]; %Die beiden Maximalzeilen
               werden max_array zugewiesen
19     else %hat audarr und micarr nur eine Zeile, dann
20         if ~isnan(micarr) %Überprüfung, ob in micarr Koordinaten
               stehen
21             if ~isnan(audarr) %Überprüfung, ob in audarr
                   Koordinaten stehen
22                 max_array = [micarr; audarr]; %Die beiden
                       Maximalzeilen werden max_array zugewiesen
23             else
```

```
24                  max_array = micarr; %micarr wird max_array
                        zugewiesen
25              end
26          else
27              if ~isnan(audarr) %Überprüfung, ob in audarr
                    Koordinaten stehen
28                  max_array = audarr; %audarr wird max_array
                        zugewiesen
29              else %wenn in beiden Arrays nichts steht, dann
30                  max_array = [raum; raum]; %die Raumdimensionen
                        werden zweimal max_array zugewiesen
31              end
32          end
33      end
34  end
35  max_zeile = max(max_array); %maximalwerte ermitteln
36  differenz = raum - max_zeile; %maximalwerte von den
    Raumdimensionen abziehen. Ist ein Wert unter 0, dann ist eine
    Koordinate außerhalb des Raumes
37  ergebnis = min(differenz); %kleinsten Wert der Differenz ermitteln
38  if ergebnis<0 %Ist Ergebnis unter 0, dann ist eine Koordinate
    außerhalb des Raumes
39      erfolgreich = 0; %Rückmeldung, eine Koordinate ist außerhalb
            der Raumdimension
40  else
41      erfolgreich = 1; %Rückmeldung, alle Koordinaten sind innerhalb
            der Raumdimensionen
42  end
```

3.4. Update-Routine

An dieser Stelle sei noch eine Funktion innerhalb des GUI-m-File zu erwähnen, weil sie eine besondere Aufgabe innerhalb der GUI hat. Das Listing A.1 des GUI-m-File ist dem Anhang beigefügt. Ab Zeile 123 fängt die Funktion *update_fig_uitable* an. Diese funktion wird immer dann aufgerufen, sobald sich einer der Koordinaten der Mikrofone oder Audioquellen geändert hat, oder die Raumdimension geändert wurde. In diesem Fall wird der Graph neu gezeichnet, die Tabellen und alle Popupmenus aktualisiert und die Bedienfläche für den Simulationsstart deaktiviert. Diese zentrale Art der Aktualisierung wurde gewählt, da die Menge der Daten gering genug ist, um alle Daten immer zu aktualisieren, wenn sich ein Wert ändert und weil diese zentrale Art fehlerunanfälliger ist.

4. Berechnung der Reflexionen und deren Radien

Für jede Mikrofon-Sprachquellen-Kombination müssen alle Reflexionen I.- und II.-Ordnung berechnet werden. Dazu wird das sogenannte *Spiegelquellenverfahren* verwendet. Dardurch ergibt sich für jede Reflexion ein neuer Punkt im Raum oder außerhalb des Raums. Danach wird von jedem Punkt der Abstand zum Ziel berechnet.

4.1. Reflexionsberechnung

Da die in einem Raum sich ausbreitenden Schallwellen nicht ausschließlich direkt zum Mikrofon gelangen, sondern auch an Wänden, Decke und Fußboden reflektieren, werden im Folgendem die Reflexionen näher betrachtet.

Bei dem hier verwendeten Spiegelquellenverfahren wird die Sprachquelle an der reflektierenden Wand gespiegelt. Der Abstand zwischen der gespiegelten Quelle und dem Mikrofon entspricht der Strecke zwischen Quelle und Mikrofon bei Reflexion an der gespiegelten Wand. Der Zusammenhang ist in Abbildung 4.1 auf Seite 23 dargestellt.

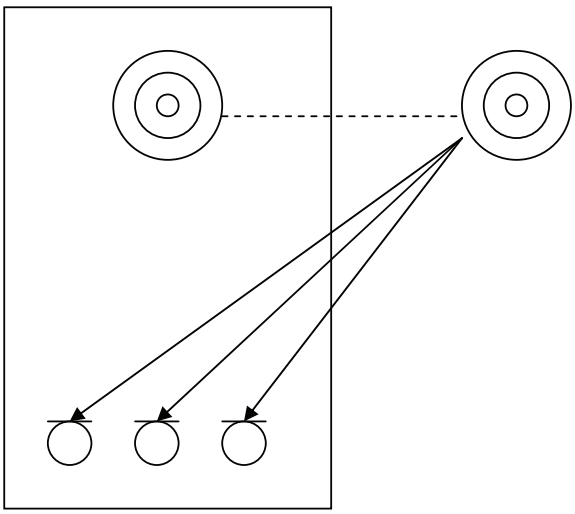


Abbildung 4.1.: Spiegelung eines Sprachquellenausgangspunktes mit einer Wand

Die Spiegelung der Sprachquelle mit einer Wand erfolgt in der Simulation mit den Formeln 4.1 und 4.2.

$$d = \frac{|\vec{n}\,(\vec{r_Q} - \vec{r_l})|}{|\vec{n}|} \tag{4.1}$$

$$\vec{r_s} = 2 \cdot d \cdot \vec{e_n} + \vec{r_Q} \tag{4.2}$$

$\vec{n}$: beliebiger senkrecht zur Ebene stehender Vektor

$\vec{r_1}$: beliebiger Vektor auf beliebigem Punkt in der Ebene

$\vec{r_Q}$: Vektor zu spiegelndem Punkt

d: Abstand des zu spiegelnden Punktes zur Ebene

$\vec{r_s}$: Vektor zum gespiegelten Punkt

$\vec{e_n}$: Einheitsvektor in Richtung des senkrecht zur Ebene stehenden Vektors

Für die Berechnung der Spiegelung in dem Simulationsprogramm vereinfacht sich die Berechnung deutlich, da die jeweilige Ebene in bzw. parallel zu den Ebenen den Achsen liegen. Es muss nur der Lotpunkt der Ebenen zum zu spiegelnden Punkt ermittelt werden. Um den Lotpunkt zu ermitteln, wird der zu spiegelnde Punkt auf die Ebene transformiert. Dazu wird die Punktkoordinate, die orthogonal zur Ebene steht in die Ebene gelegt. Das heißt bei einer Ebene, die über die x-y Achse geht und an der Stelle z steht, hat der Lotpunkt die Koordinaten x und y vom Punkt P und die z-Koordinate von der Ebene. Um den Spiegelpunkt zu

ermitteln, wird jetzt der zu spiegelnde Vektorpunkt OP vom doppelten Lotpunktvektor abgezogen.

$$\vec{P'} = 2 \cdot \vec{x_0} - \vec{P} \tag{4.3}$$

$2 \cdot \vec{x_0}$: doppelter Lotpunktvektor

$\vec{P'}$: gespiegelter Punkt

$\vec{P}$: zu spiegelnder Punkt

Die Funktion 4.1 berechnet die Reflexionen einer Koordinate an den 6 Wänden des Raumes.

Listing 4.1: Funktion zur Berechnung von Reflexionen an Raumwänden

```
1  function [positionen_reflektiert] = berechne_reflextion(punkt,
       raum)
2  %Rückgabewert(positionen_reflektiert): Koordinaten der 6
       Reflexionen an den 6 Wänden des Raumes
3  %Übergabeparameter(punkt): Punkt der im 3D-Koordinatensystem an 6
       Wänden eines Raumes gespiegelt werden soll
4  %Übergabeparameter(raum): Raumdimensionen
5  lotpunkt_links = [0 1 1].*punkt; %Ermittlung des Lotpunktes mit
       der linken Wand
6  lotpunkt_rechts = [raum(1) punkt(2) punkt(3)]; %Ermittlung des
       Lotpunktes mit der rechten Wand
7  lotpunkt_vorn = [1 0 1].*punkt; %Ermittlung des Lotpunktes mit der
        vorderen Wand
8  lotpunkt_hinten = [punkt(1) raum(2) punkt(3)]; %Ermittlung des
       Lotpunktes mit der hinteren Wand
9  lotpunkt_oben = [punkt(1) punkt(2) raum(3)]; %Ermittlung des
       Lotpunktes mit der Decke
10 lotpunkt_unten= [1 1 0].*punkt; %Ermittlung des Lotpunktes mit dem
        Boden
11
12 ref_links = 2 * lotpunkt_links - punkt; %Ermittlung der
       Koordinaten des an der linken Wand gespiegelten Punktes
13 ref_rechts = 2 * lotpunkt_rechts - punkt; %Ermittlung der
       Koordinaten des an der rechten Wand gespiegelten Punktes
14 ref_vorn = 2 * lotpunkt_vorn - punkt; %Ermittlung der Koordinaten
       des an der vorderen Wand gespiegelten Punktes
15 ref_hinten = 2 * lotpunkt_hinten - punkt; %Ermittlung der
       Koordinaten des an der hinteren Wand gespiegelten Punktes
```

```
16 ref_oben = 2 * lotpunkt_oben - punkt; %Ermittlung der Koordinaten
       des an der Decke gespiegelten Punktes
17 ref_unten = 2 * lotpunkt_unten - punkt; %Ermittlung der
       Koordinaten des am Boden gespiegelten Punktes
18
19
20 positionen_reflektiert = [ref_links; ref_rechts; ref_vorn;
       ref_hinten; ref_oben; ref_unten]; %Die Koordinaten der 6
       gespiegelten Punkte werden untereinaunder auf eine Variable
       geschrieben
```

Für die Berechnung der Reflexionen zweiter Ordnung ergeben sich $6 \cdot 5$ Reflexionen. Hier wird der Sprachsignalausgangspunkt in zwei Wänden gespiegelt. Schematisch verdeutlicht wird das in Abbildung 4.2.

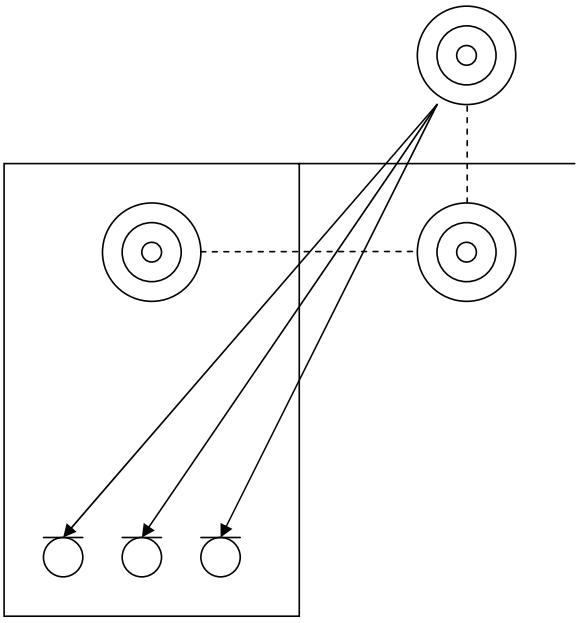


Abbildung 4.2.: Spiegelung eines Sprachquellenausgangspunktes mit zwei Wänden

Die Funktion 4.2 berechnet die Reflexionen der I.- und II.-Ordnung eines Punktes an den Wänden eines Raumes.

Listing 4.2: Funktion zur Berechnung der Reflexionen I.- und II.-Ordnung

```
1 function [posistionen] = berechnung_reflektion_rekursiv(punkt,
      raum)
2 %Rückgabewert(positionen): 37 Positionen aller Reflexionen bis zur
      II.Ordnung
```

```
%Übergabewert(punkt): Koordinaten eines Punktes, der an den Wänden des Raumes gespiegelt werden soll
%Übergabeparameter(raum): Raumdimensionen
reflektion_I_Ordnung = berechne_reflextion(punkt, raum); % Funktionsaufruf zur Ermittlung der 6 Reflexionen des Ausgangspunktes an den 6 Wänden des Raumes
for i=1:6 %Von 1 bis 6
    zwischenspeicher = berechne_reflextion(reflektion_I_Ordnung(i,:), raum); %Funktionsaufruf zur Ermittlung der 6 Reflexionen eines reflektierten Punktes der Reflexion I. Ordnung an den 6 Wänden des Raumes
    zwischenspeicher = zeile_aus_array_loeschen(zwischenspeicher, i); % Doppelspiegelungen an der selben Wand werden gelöscht
    reflektionen_II_Ordnung(((i-1)*5)+1:i*5,1:3) = zwischenspeicher; %Zuweisung des Zwischenspeichers auf reflektionen_II_Ordnung
end
posistionen = [punkt; reflektion_I_Ordnung; reflektionen_II_Ordnung]; %Zusammenfügung aller Positionen
```

Damit alle Radien zwischen jedem Mikrofon und Audioquelle mit deren gespiegelten Koordinaten bis zur II.-Ordnung ermittelt werden können, wurde die Funktion 4.3 geschrieben. Die Funktion dient somit der Organisation und fügt am Ende alle ermittelten Radien der in Abbildung 2.2 beschriebenen Struktur zu. Im Quellcode ist zu beachten, dass nicht die Reflexionen der Audioquellen berechnet werden, sondern die Reflexionen der Mikrofone. Dies wurde so festgelegt, um die Berechnung von bewegten Quellen zu vereinfachen. Es hat keinerlei Auswirkungen auf die Ergebnisse, ob die Reflexionen der Mikrofone oder die der Audioquellen berechnet wird. Der ermittelte Radius ist derselbe.

Listing 4.3: Organisation der Reflexions- und Radienberechnung

```
function [mic] = berechnung_aller_radien(micarr, audarr, raum)
%Rückgabewert(mic): Radien zwischen den Mikrofonen und den Audioquellen eingeordnet in eine Struktur, die mit Mikrofon_1 ... Mikrofon_n anfängt
%Übergabeparameter(micarr): Koordinaten aller Mikrofone
%Übergabeparameter(audarr): Koordinaten aller Audioquellen
%Übergabeparameter(raum):  Raumdimensionen
s = struct('direkt', 0); %Vorinitialisierung der Struktur s
audio = struct('Audioquelle_1', 0); %Vorinitialisierung der Struktur audio
Audiostring = string_array_erstellen(size(audarr), 'Audioquelle'); %Erzeugt eine Strukturbeschriftung der 2.Ebene
```

```
9  strukturbeschriftung = get_strukturbeschriftung(); %Lädt die
       Strukturbeschriftung der ersten Ebene
10 laenge_struktur = length(strukturbeschriftung); %Ermittlung der
       Länge der Strukturbeschriftung
11 anzahl_mikrofone = size(micarr); %Ermittlung der Anzahl der
       Mikrofone
12 anzahl_audioquellen = size(audarr); %Ermittlung der Anzahl der
       Audioquellen
13 for i=1:anzahl_mikrofone %Von 1 bis Anzahl der Mikrofone
14     posistionen = berechnung_reflektion_rekursiv(micarr(i,:), raum
           ); %Berechnung aller Reflexionen eines Mikrofons
15     radien = radius_berechnung(posistionen, audarr); %Berechnung
           aller Radien der reflektierten Mikrofone mit den
           Audioquellen
16     for j=1:size(radien) %Von 1 bis Anzahl der Audioquellen
17         for k=1:laenge_struktur %Von 1 bis Länge der Struktur
18             s = setfield(s, char(strukturbeschriftung(k)), radien(
                   j,k)); %Struktur zusammenfügen auf der ersten Ebene
19         end
20         audio = setfield(audio, char(Audiostring(j)), s); %
               Struktur zusammenfügen auf der zweiten Ebene
21     end
22     mic(i) = audio; %Struktur zusammenfügen auf der dritten Ebene
23 end
```

Reflexionen höherer Ordnungen wurden vernachlässigt, da mit der Implementierung der Reflexionen I.- und II.-Ordnung bezweckt werden sollte, dass beim Test der Algorithmen Schwierigkeiten im realen Raum mehr zur Geltung kommen. Eine Hauptschwierigkeit der Algorithmen ist dabei die Detektierung von Reflexionen, dabei ist es am wahrscheinlichsten, dass Reflexionen der ersten und zweiten Ordnung detektiert werden. Der Algorithmus muss zwischen direktem Schall und reflektierendem Schall unterscheiden. Möchte der Anwender die Störung durch den diffusen Nachhall mit testen, kann der Anwender eine Audiodatei mit weißem Rauschen mit in die Simulation einfügen.

4.2. Berechnung der Radien

Um den Abstand zwischen den Mikrofonen und den Audioquellen bzw. deren Spiegelungen zu berechnen, wird in einem ersten Schritt Formel 4.4 verwendet. Dabei wird ein Hilfsvektor zwischen den beiden Punkten berechnet. Formel 4.5 ist im zweiten Schritt erforderlich, um den Betrag des Hilfsvektors zu ermitteln. Der Betrag des Hilfsvektors entspricht dem Abstand bzw. Radius zwischen Mikrofon und Audioquelle.

$$\vec{a} = \overrightarrow{P_1 P_2} = (x_2 - x_1)\,\vec{e}_x + (y_2 - y_1)\,\vec{e}_y + (z_2 - z_1)\,\vec{e}_z = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} \tag{4.4}$$

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{4.5}$$

Im Simulationsprogramm wurde die Berechnung des Abstandes so implementiert, dass der Abstand zwischen allen Mikrofonen und allen Audioquellen, wie in der Funktion 4.4 aufgeführt ist, berechnet wird.

Listing 4.4: Funktion zum Berechnen der Abstände

```
function [radius] = radius_berechnung(micarr, audarr)
%Rückgabewert(radius): ist ein 2D- Array mit allen Radien zwischen den Mikrofonkordinaten und den Audiokoordinaten
%Übergabeparameter(micarr): Koordinaten der Mikrofone
%Übergabeparameter(audarr): Koordinaten der Audioquellen
if isnan(micarr) %hier wird überprüft ob in micarr eine Zeile steht
    radius = NaN; %in diesem Fall wird als Ergebnis NaN zurückgegeben
else
    if isnan(audarr) %hier wird überprüft ob in audarr eine Zeile steht
        radius = NaN; %in diesem Fall wird als Ergebnis NaN zurückgegeben
    else
        for j=1:size(audarr) %Von 1 bis Anzahl der Audioquellen
            for i=1:size(micarr) %Von 1 bis Anzahl der Mikrofonquellen
                micarr_z(i,:) = micarr(i,:)-audarr(j,:); % Hilfsvektor zwischen den beiden Punkten berechnen
                radius(j,i) = sqrt(micarr_z(i,1)^2+micarr_z(i,2)^2+micarr_z(i,3)^2); %Betragsberechnung des Hilfsvektors
            end
        end
    end
end
```

5. Umrechnung in Laufzeiten und Dämpfungen

Im Folgenden wird der mathematische Zusammenhang bei der Umrechnung von Radien in Laufzeitverzögerungen und Dämpfungen hergeleitet. Bei der Dämpfung durch Ausbreitung wird einmal über die Intensität und einmal über die Amplitude der Schallwelle die Berechnungsformel hergeleitet. Dabei wird deutlich, dass die Intensität quadratisch und die Amplitude der Schallwelle linear mit dem Abstand abnimmt. Hierzu wird vorweg die Ausbreitungsgeschwindigkeit von Schallwellen berechnet.

5.1. Ausbreitungsgeschwindigkeit von Schallwellen

Die Ausbreitungsgeschwindigkeit von Schallwellen ist von vier Parametern abhängig, wie in Formel 5.1 [13, Seite 467] ersichtlich ist.

$$v = \sqrt{\frac{\gamma \cdot R \cdot T}{m_{\text{Mol}}}} \tag{5.1}$$

γ: Ist eine Konstante für zweiatomige Moleküle, wie O_2 und N_2. Da Luft zu 98% aus diesen Molekühlen besteht, wird hier die Konstante für zweiatomige Moleküle eingesetzt: $1{,}4$

R: ist die universelle Gaskonstante und beträgt: $8{,}314 \frac{J}{mol \cdot K}$

m_{Mol}: molare Masse des Gases, in dem die Schallwellenausbreitung stattfindet. Hier Luft mit einem Wert von: $29 \cdot 10^{-3} \frac{kg}{mol}$

T: Ist die Umgebungstemperatur in der die Schallwellenausbreitung stattfindet. Hier wurde $20°C$ gewählt, da davon ausgegangen werden kann, dass Versuche mit einem Mikrofonarray bei Raumtemperatur stattfindet. Damit hat T einen Wert von $293K$.

Nach Einsetzen der Größen in Formel 5.1 kommt man zum Ergebniss 5.2.

$$v = \sqrt{\frac{\gamma \cdot R \cdot T_{20°C}}{m_{\text{Mol}}}} = \sqrt{\frac{1{,}4 \cdot 8{,}314 \frac{J}{mol \cdot K} \cdot 293 K}{29 \cdot 10^{-3} \frac{kg}{mol}}} = 343 \frac{m}{s} \tag{5.2}$$

5.2. Umrechnung von Radien in Laufzeitverzögerungen

Zur Umrechnung von Radien in Laufzeitverzögerungen wird Formel 5.3 verwendet.

$$T_{\text{Tot}} = \frac{r}{v_{\text{Luft}}} \tag{5.3}$$

T_{Tot}: Laufzeitverzögerung

r: Abstand zwischen Mikrofon und Audioquelle

v_{Luft}: Ausbreitungsgeschwindigkeit von Schallwellen in dem Medium Luft.

Der entsprechende MATLAB-Code wird im Listing 5.1 dargestellt. Dabei wird der Funktion eine wie in Kapitel 2 in Abbildung 2.2 beschriebene Struktur übergeben und wieder als Rückgabewert zurückgegeben. Die Struktur wird in der Funktion in Datenblöcke zerlegt, die Laufzeiten berechnet und am Ende werden die Datenblöcke wieder in Strukturen geschrieben.

Listing 5.1: Umrechnung von Radien in Laufzeitverzögerung

```
1  function [laufzeiten_array_struktur] = berechne_laufzeiten(
       radien_array_struktur)
2  %Bei den hier übergebenen Strukuren handelt es sich um Strukturen
       mit 3
3  %Ebenen. Ebene 1 mit Mikrofon_1 bis Mikrofon_n, Ebene 2 mit
       Audioquelle_1
4  %bis Audioquelle_n und Ebene 3 mit allen Wegen die die
       Schallwellen von den
5  %Audioquellen zu den Mikrofonen nimmt(direkt-->refWandxWandy),
       ingesammt 37.
6
7  laenge_des_radien_arrays = length(radien_array_struktur); %Länge
       der Struktur ermitteln
8
9  s = struct('direkt', 0); %Struktur "s" vorinitialisiert
10 audio = struct('Audioquelle_1', 0); %Struktur "audio"
       vorinitialisiert
```

```
11 Audiostring = string_array_erstellen(length(fieldnames(
       radien_array_struktur)), 'Audioquelle');
12 %erstellt Audiostring um die neue Stuktur zu beschriften
13
14 strukturbeschriftung = get_strukturbeschriftung(); %gibt die
       Strukturbeschriftung zurück, die in get_strukturbeschriftung
       definiert ist
15 laenge_struktur = length(strukturbeschriftung); %Länge der
       Strukturbeschriftung
16 for i=1:laenge_des_radien_arrays %Von 1 bis Anzahl der Mikrofone
17     [datenblock zeilennamen spaltennamen] = struktur_zu_array(
           radien_array_struktur(i)); %Wandelt die Struktur in einen 2
           D Datenblock um
18     datenblock = datenblock./343; % Berechnung der Laufzeiten für
           alle Daten im Datenblock
19     %Ab hier wird der Datenblock wieder in eine Struktur
           zurückgeführt
20     for j=1:size(datenblock) %Von 1 bis Anzahl der Audioquellen
21         for k=1:laenge_struktur%Von 1bis Anzahl der
               Schallwellenwege
22             s = setfield(s, char(strukturbeschriftung(k)),
                   datenblock(j,k)); %Zusammensetzten der Struktur auf
                    der Ebene 3
23         end
24         audio = setfield(audio, char(Audiostring(j)), s); %
               Zusammensetzen der Struktur auf Ebene 2
25     end
26     laufzeiten_array_struktur(i) = audio; %Zusammensetzen der
           Struktur auf Ebene 1
27 end
```

5.3. Dämpfung der Intensität einer Schallwelle durch Ausbreitung

Wenn sich eine punktförmige Wellenquelle in der Luft kugelförmig ausbreitet, dann wird die Energie in einem Abstand von r auf die Oberfläche der Kugelform $A = 4 \cdot \pi \cdot r^2$ gleichmäßig verteilt. Die Leistung P, die auf eine Fläche A abgestrahlt wird, ist immer gleich. Der Quotient $\frac{P}{A}$ wird als Leistungsdichte bezeichnet. Bei einer Kugelausbreitung ergibt der Quotient $\frac{P}{4 \cdot \pi \cdot r^2}$. Mit zunehmender Entfernung nimmt die Leistungsdichte quadratisch ab. Die mittlere

Leistungsdichte, die auf eine Fläche A senkrecht zur Ausbreitungsrichtung auftrift, wird als die Intensität der Welle bezeichnet. [13, Seite 478]

$$I = \frac{\langle P \rangle}{A_{\text{Kugel}}} \tag{5.4}$$

I: Intensität der Welle mit der Einheit: $\frac{\text{Watt}}{\text{Quadratmeter}} = \frac{W}{m^2}$

$\langle P \rangle$: mittlere Leistung auf der Kugeloberfläche der ausbreitenden Schallwelle, welche der mittleren erzeugten Leistung am Lautsprecher äquivalent ist.

A_{Kugel}: Kugeloberfläche der sich ausbreitenden Schallwelle

Die Leistung, die direkt an der Membran einer Audioquelle entsteht ist aus dem Produkt von Membranfläche und Intensität der Welle an der Membran errechenbar.

$$\langle P \rangle = I_{\text{Membran}} \cdot A_{\text{Membran}} \tag{5.5}$$

I_{Membran}: Schallintensität an der Lautsprechermembran

A_{Membran}: Oberfläche der Lautsprechermembran

Die Intensität der Welle auf der Kugeloberfläche ist der Quotient von mittlerer Leistung zur Kugeloberfläche.

$$I_{\text{Kugel}} = \frac{\langle P \rangle}{A_{\text{Kugel}}} = \frac{\langle P \rangle}{4 \cdot \pi \cdot r^2} \tag{5.6}$$

I_{Kugel}: Schallintensität an der Kugeloberfläche der ausbreitenden Schallwelle

Um den Zusammenhang der beiden Intensitäten ersichtlich zu machen, setzt man die Formeln 5.5 in die Formel 5.6 ein.

$$I_{\text{Kugel}} = \frac{I_{\text{Membran}} \cdot A_{\text{Membran}}}{4 \cdot \pi \cdot r^2} \tag{5.7}$$

Löst man die Gleichung nach dem Verhältniss der Intensitäten der Schallquellen auf, so erhält man die Dämpfung a_I. Aus dem konstanten Therm $\frac{A_{\text{Membran}}}{4 \cdot \pi}$ wird die Konstante α. Die Dämpfung der Intensität a_I nimmt mit dem Abstand r quadratisch zu.

$$\frac{I_{\text{Kugel}}}{I_{\text{Membran}}} = \frac{1}{r^2} \cdot \overbrace{\frac{A_{\text{Membran}}}{4 \cdot \pi}}^{\alpha} = a_I \tag{5.8}$$

Zusammengefasst ergibt sich die die Formel 5.9.

$$a_l = \alpha \cdot \frac{1}{r^2} \tag{5.9}$$

5.4. Dämpfung der Amplitude einer Schallwelle durch Ausbreitung

Wird die Berechnung der Amplitude als Ausgangs- und Eingangswert zugrunde gelegt, kommt man zu einem linearen Zusammenhang von Amplitudenverhältnis $\frac{S_{\text{max, Kugel}}}{S_{\text{max, Membran}}}$ und dem Abstand r.

Zur Übersicht wird der Rechenweg einmal dargestellt.

$$S_{\text{max, Membran}} \rightarrow I_{\text{Membran}} \rightarrow \langle P \rangle \rightarrow I_{\text{Kugel}} \rightarrow S_{\text{max, Kugel}}$$

$S_{\text{max, Membran}}$: Amplitude der Lautsprechermembran

$S_{\text{max, Kugel}}$: Amplitude an der Kugeloberfläche der ausbreitenden Schallwelle

Mit der Formel 5.10 [13, Seite 479] wird die Intensität an der Membran eines Lautsprechers berechnet.

$$I = \frac{1}{2} \cdot \rho_0 \cdot \omega^2 \cdot S_{\text{max}}^2 \cdot v \tag{5.10}$$

ρ_0: Gleichgewichtsdichte von Luft mit dem Wert: $1{,}29 \frac{kg}{m^3}$

Für die weitere Berechnung wird Formel 5.10 nach S_{max} umgestellt.

$$S_{\text{max}} = \sqrt{\frac{I \cdot 2}{\rho_0 \cdot \omega^2 \cdot v}} \tag{5.11}$$

Für die Amplitude an der Kugeloberfläche der sich ausbreitenden Schallwelle gilt folglich.

$$S_{\text{max, Kugel}} = \sqrt{\frac{I_{\text{Kugel}} \cdot 2}{\rho_0 \cdot \omega^2 \cdot v}} \tag{5.12}$$

Für die Intensität wird in Formel 5.12 Formel 5.4 eingesetzt.

$$S_{\text{max, Kugel}} = \sqrt{\frac{\langle P \rangle \cdot 2}{A_{\text{Kugel}} \cdot \rho_0 \cdot \omega^2 \cdot v}} \tag{5.13}$$

Für A_{Kugel} kann $r^2 \cdot 4 \cdot \pi$ eingesetzt werden.

$$S_{\text{max, Kugel}} = \sqrt{\frac{\langle P \rangle \cdot 2}{r^2 \cdot 4 \cdot \pi \cdot \rho_0 \cdot \omega^2 \cdot v}} \tag{5.14}$$

Da die mittlere Leistung $\langle P \rangle$ an der Membran des Lautsprechers gleich groß der mittleren Leistung auf der Kugeloberfläche ist, wird für $\langle P \rangle$ in Formel 5.14 Formel 5.5 eingesetzt.

$$S_{\text{max, Kugel}} = \sqrt{\frac{I_{\text{Membran}} \cdot A_{\text{Membran}} \cdot 2}{r^2 \cdot 4 \cdot \pi \cdot \rho_0 \cdot \omega^2 \cdot v}} \tag{5.15}$$

Für I_{Membran} wird Formel 5.10 eingesetzt.

$$S_{\text{max, Kugel}} = \sqrt{\frac{\frac{1}{2} \cdot \rho_0 \cdot \omega^2 \cdot S^2_{\text{max, Membran}} \cdot v \cdot A_{\text{Membran}} \cdot 2}{r^2 \cdot 4 \cdot \pi \cdot \rho_0 \cdot \omega^2 \cdot v}} \tag{5.16}$$

Die Ausbreitungsgeschwindigkeit v, Frequenz ω, Gleichgewichtsdichte ρ_0 und der Faktor 2 kürzen sich weg.

$$S_{\text{max, Kugel}} = \sqrt{\frac{S^2_{\text{max, Membran}} \cdot A_{\text{Membran}}}{r^2 \cdot 4 \cdot \pi}} \tag{5.17}$$

Die quadratische Amplitude an der Membran $S^2_{\text{max, Membran}}$ wird aus der Wurzel herausgezogen und auf die linke Seite gebracht, was der Dämpfung a_A entspricht.

$$\frac{S_{\text{max, Kugel}}}{S_{\text{max, Membran}}} = \sqrt{\frac{A_{\text{Membran}}}{r^2 \cdot 4 \cdot \pi}} = a_A \tag{5.18}$$

Der konstante Therm $\frac{A_{\text{Membran}}}{4 \cdot \pi}$ wird zu einer Konstanten α zusammengefasst.

$$a_A = \sqrt{\frac{\alpha}{r^2}} \tag{5.19}$$

Für eine bessere Übersicht wird der quadratische Abstand r^2 aus der Wurzel herausgezogen und es ergibt sich die endgültige Formel 5.20.

$$a_A = \sqrt{\alpha} \cdot \frac{1}{r} \tag{5.20}$$

Im Simulationsprogramm wird die Dämpfung der Amplitude a_A verwendet, da dies die dem Anwender interessante Größe ist. Sowohl dem Lautsprecher wird eine Amplitude vorgegeben, als auch vom Mikrofon wird eine Amplitude aufgenommen.

Im Folgenden MATLAB-Code 5.2 wird die Berechnung der Dämpfung durchgeführt. Auch hier wird der Funktion eine Struktur übergeben und die Funktion gibt eine Struktur zurück.

Listing 5.2: Umrechnung von Radien in Dämpfungen

```
1  function [daempfung_array_struktur] = berechne_daempfung(
       radien_array_struktur)
2  %Diese Funktion berechnet die Dämpfung aus einer Radienstruktur
3  %
4  %Übergabeparameter:
5  %radien_array_stuktur: ist eine Struktur in der alle Radienwerte
       abgespeichert sind
6  %
7  %Rückgabewert:
8  %daempfungen_array_stuktur: ist eine Struktur in der alle
       Dämpfungswerte abgespeichert werden
9
10 laenge_des_radien_arrays = length(radien_array_struktur); %Die
       Länge des Radienarrays wird ermittelt
11
12 s = struct('direkt', 0); %Die Struktur s wird vorinitialisiert
13 audio = struct('Audioquelle_1', 0); %Die Struktur audio wird
       vorinitialisiert
14 Audiostring = string_array_erstellen(length(fieldnames(
       radien_array_struktur)), 'Audioquelle'); %Ein Cellarray wird
       erstellt mit Audioquelle_1...Audioquelle_n
15
16 strukturbeschriftung = get_strukturbeschriftung(); %Die
       Strukturbeschriftung wird geladen.
17 laenge_struktur = length(strukturbeschriftung); % Die Länge der
       Strukturbeschriftung wird ermittelt
18
19
20 for i=1:laenge_des_radien_arrays %Von 1 bis Anzahl der Mikrofone
```

```
21      [datenblock zeilennamen spaltennamen] = struktur_zu_array(
            radien_array_struktur(i)); %Die Struktur i wird in einen
            Datenblock zerlegt
22      kleinste_radien = min(datenblock); %Der kleinste Radius wird
            ermittelt
23      kleinster_radius(i) = min(kleinste_radien); %Der kleinste
            Radius wird ermittelt
24  end
25  kleinster_radius = min(kleinster_radius); %Der kleinste Radius
        wird ermittelt, damit später die geringste Dämpfung auf 1
        normiert werden kann
26
27  for i=1:laenge_des_radien_arrays %Von 1 bis Anzahl der Mikrofone
28      [datenblock zeilennamen spaltennamen] = struktur_zu_array(
            radien_array_struktur(i)); %Die Struktur i wird in einen
            Datenblock zerlegt
29      datenblock = 1./datenblock; %Es wird der Kehrwert der Radien
            ermittelt
30      datenblock = datenblock.*kleinster_radius; %Die Daten werden
            auf den kleinsten Radius normiert.
31      %Ab hier wird die Struktur wieder zusammengesetzt
32      for j=1:size(datenblock) %Von 1 bis Anzahl der Audioquellen
33          for k=1:laenge_struktur %Von 1 bis Anzahl der Reflexionen
34              s = setfield(s, char(strukturbeschriftung(k)),
                    datenblock(j,k)); %Die einzelnen Werte werden der
                    Struktur übergeben
35          end
36          audio = setfield(audio, char(Audiostring(j)), s); %Erste
                Verschachtelung
37      end
38      daempfung_array_struktur(i) = audio; %Zweite Verschachtelung
39  end
```

6. Simulation des akustischen Raumes

Die Simulation wurde in einem ersten Anlauf unter SIMULINK implementiert. Dabei wurde festgestellt, dass das Modell zwar fehlerfrei funktionierte, aber sehr langsam war. Die Simulation muss zwar nicht echtzeitfähig sein, jedoch sollte sie nicht ein unakzeptables Zeitlimit überschreiten. Aus diesem Grund wurde die Simulation ein zweites Mal unter MATLAB implementiert. Das MATLAB-Model ist um ein vielfaches schneller als das SIMULINK-Model. Da das SIMULINK-Model funktionsfähig und besonders anschaulich ist wird es dennoch im Folgendem dargestellt.

6.1. SIMULINK-Model

In SIMULINK werden die fertig berechneten unter MATLAB berechneten Parameter in SIMULINK-Übertragungssysteme übergeführt. Aus einer Wave-Datei werden vom System Daten eingelesen. Das Simulationsergebnis kann dann mehreren (Anzahl der Mikrofone) Wave-Dateien zugeführt werden. Außerdem können die Ergebnisse dem Workspace zur Verfügung gestellt werden. Eine schematische Darstellung ist in der Abbildung 6.1 auf Seite 38 dargestellt.

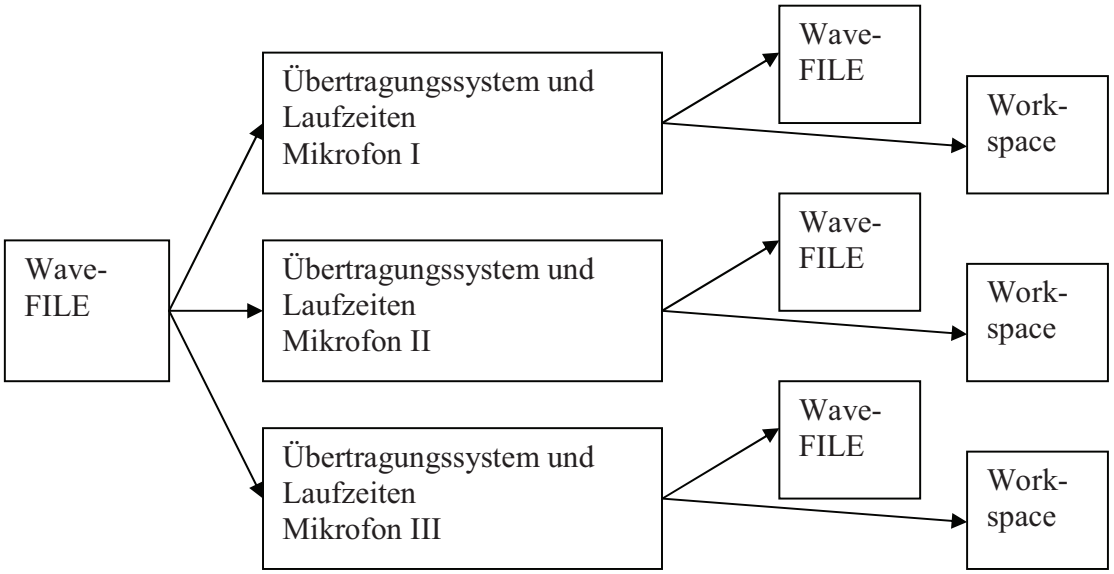


Abbildung 6.1.: Schematische Darstellung einer SIMULINK-Simulation

Bricht man die Darstellung weiter herunter, kann jeder Filter, Dämpfung und Laufzeit als Systemblock dargestellt werden. Dies ist in Abbildung 6.2 dargestellt.

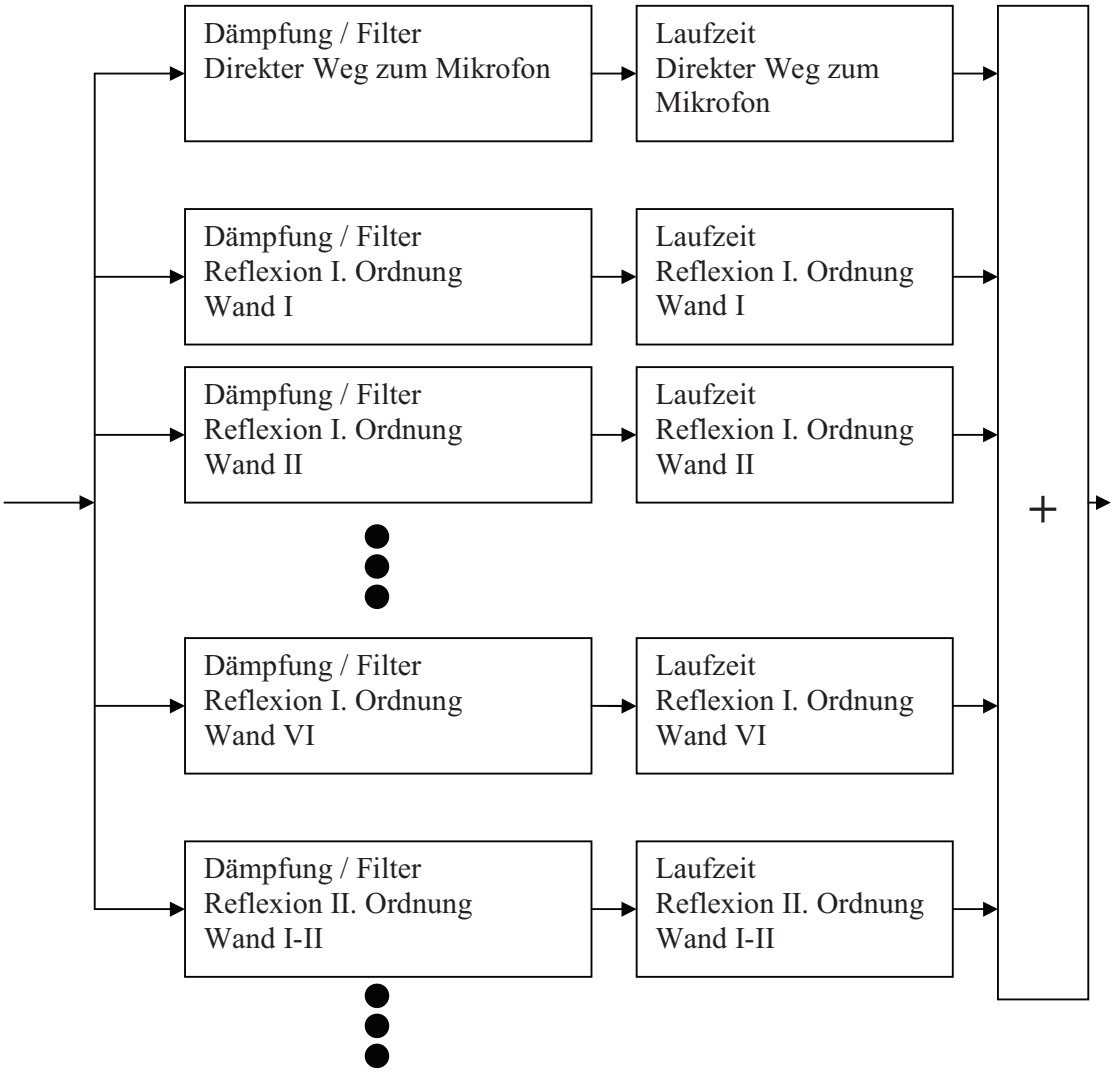


Abbildung 6.2.: Darstellung der einzelnen SIMULINK-Systemblöcke

In Abbildung 6.3 ist das SIMULINK-Simulationsmodel abgebildet.

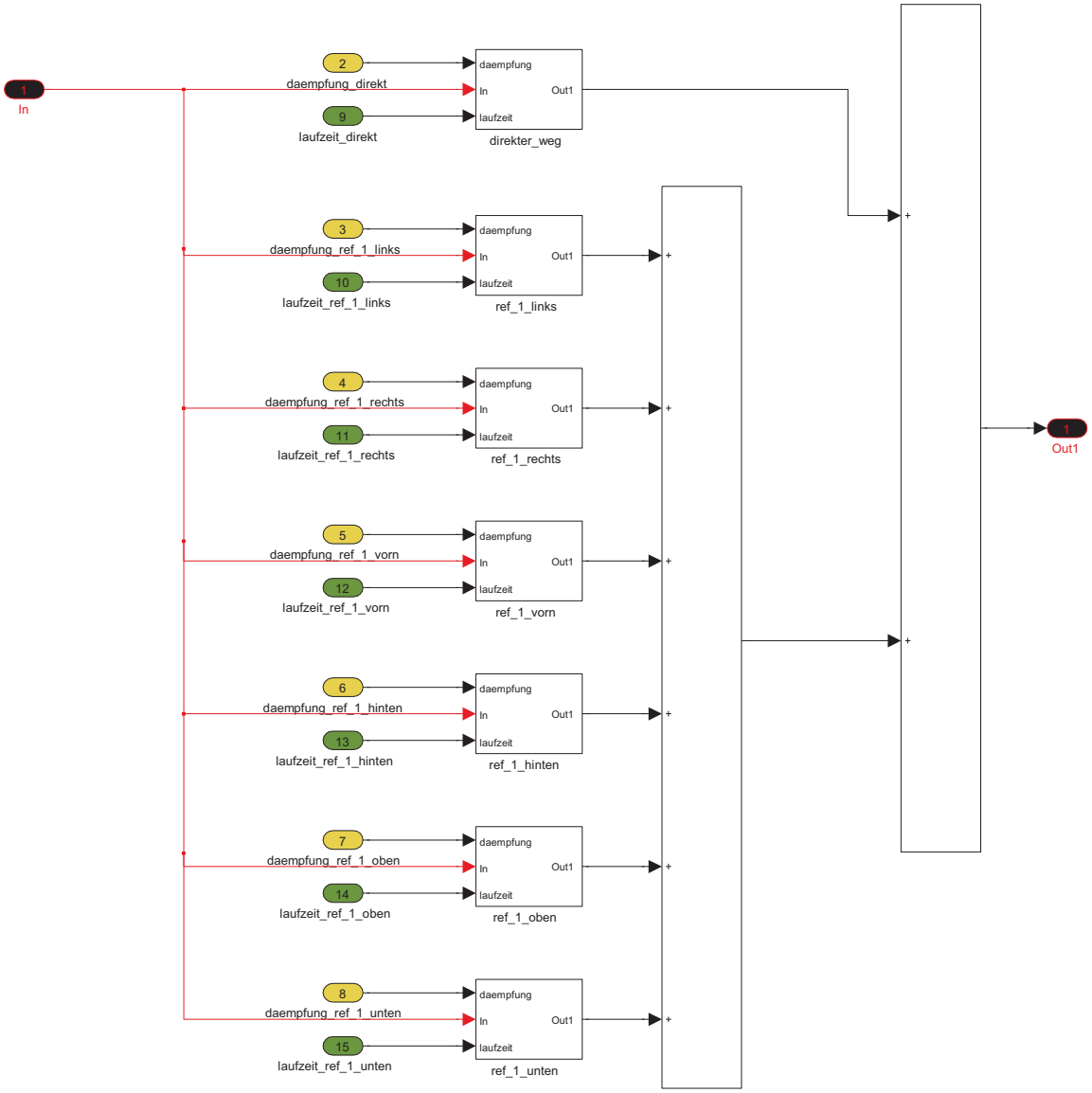


Abbildung 6.3.: SIMULINK-Simulationsmodell

In den Systemblöcken des SIMULINK-Simulationsmodells ist der in Abbildung 6.4 dargestellten Systemaufbau hinterlegt.

Abbildung 6.4.: Einzelne Systemblöcke des SIMULINK-Simulationsmodells

Das SIMULINK-Model wurde nur bis zur Reflexion I.-Ordnung implementiert, da schon bei der geringen Anzahl an Parametern das System sehr langsam war. Für 2 Sekunden Wave-Datei Simulieren benötigte das System mehr als 10 Sekunden und das bei nur einem Mikrofon und einer Audioquelle. Um die Simulation zu beschleunigen wurde ein weiteres Simulationsmodell als MATLAB-Code implementiert.

6.2. Simulation mit MATLAB

Die Funktion 6.1 simuliert mit den übergebenen Parametern den Raum. Als Ausgabe werden Wave-Dateien erzeugt. Bei der Funktion wurde darauf geachtet, das ausschließlich für die Organisation der Simulation *for-Schleifen* verwendet wurden. Alle Berechnungen wurden mit MATLAB-Syntax unternommen. Gerade die Verschiebungsoperationen, die die Laufzeitverzögerung simulieren, würden in einer for-Schleife die Simulation um ein vielfaches verlangsamen. Somit wurden Simulationszeiten von ca. 60 Sekunden bei zwei 2 Sekunden Audiodateien, mit 6 Mikrofonen und Reflexionen bis zur II.-Ordnung schon erreicht.

SIMULINK 10 Sekunden bei 1 Audioquelle mal 1 Mikrofon mal 7 Schallwege ergibt 7 Berechnungen in 10 Sekunden. 0,7 Berechnungen pro Sekunde bei zwei 2 Sekunden Audiodateien.

MATLAB 60 Sekunden bei 2 Audioquellen mal 6 Mikrofonen mal 37 Schallwege ergibt 444 Berechnungen in 60 Sekunden. 7,4 Berechnungen pro Sekunde bei zwei 2 Sekunden Audiodateien.

Listing 6.1: MATLAB-Funktion zum Simulieren des Raumes

```
1 function erfolgreich = simulationsstart(laufzeiten, daempfung,
     fsim, aud_dateien)
2 %Rückgabewert(erfolgreich): Meldet der GUI zurück, ob die Funktion
      erfolgreich ausgeführt wurde
```

```
3  %Ausgabe: Für jedes Mikrofon wird eine Wave-Datei erstellt.
4  %Übergabeparameter(laufzeit): Laufzeitenparameter als Struktur
5  %Übergabeparameter(daempfung): Daempfungsparameter als Struktur
6  %Übergabeparameter(fsim): Simulationsfrequenz
7  %Übergabeparameter(aud_dateien): Audiodateinamen und Pfade
8  [niu mic_anz] = size(laufzeiten); %Anzahl der Mikrofone wird
       ermittelt
9  [daten_laufzeiten zeilennamen spaltennamen] = struktur_zu_array(
       laufzeiten(1));
10 [data_anz niu] = size(daten_laufzeiten); %Anzahl der Audioquellen
       wird ermittelt
11 [data(:,1) fa(1) nbit] = wavread(char(aud_dateien(1))); %Die erste
        Audiodatei wird geladen
12 data = interpft(data, length(data)*fsim/fa(1)); %Die Audiodatei
       wird auf die Simulationsfrequenz interpoliert
13 if data_anz>1 %Wenn mehr als eine Audioquelle vorhanden, dann
14     for i=2:data_anz %Von 2 bis Anzahl der Audioquellen
15         [laenge anz] = size(data); %Länge der Audiodatein
               ermitteln
16         [data_z fa(i) nbit] = wavread(char(aud_dateien(i))); %Die
               i. Audiodatei wird geladen
17         data_z = interpft(data_z,length(data_z)*fsim/fa(i)); %Die
               i. Audiodatei wird auf Simulationsfrequenz interpoliert
18         if length(data_z)<laenge %Wenn die i. Audiodatei kürzer
               als die übrigen Audiodateien ist, dann
19             data_z(end+1:laenge) = 0; %Hängt an die i. Audiodatei
                   Nullen bis sie so lang wie die anderen Audiodateien
                    ist
20         else %Wenn die i. Audiodatei länger als die übrigen
               Audiodateien ist, dann
21             data(end+1:length(data_z),:) = 0; %Hängt an alle
                   Audiodateien Nullen bis sie so lang wie die i.
                   Audiodatei ist
22         end
23         data(:,i) = data_z; %Die i. Audiodatei wird den übrigen
               Audiodateien zugefügt
24     end
25 end
26 for i=1:mic_anz %Von 1 bis Anzahl der Mikrofone
27    [daten_laufzeiten zeilennamen spaltennamen] = struktur_zu_array
          (laufzeiten(i)); %Zerlegt die i. Laufzeitstruktur in
         Datenblöcke
```

```
28    [daten_daempfung zeilennamen spaltennamen] = struktur_zu_array(
          daempfung(i)); %Zerlegt die i. Daempfungsstruktur in
          Datenblöcke
29    clear zwischen2; %zwischen2 wird gelöscht
30    verschieben = fsim.*daten_laufzeiten; %Anzahl der
          Verschiebungstakte ermitteln
31    verschieben = round(verschieben); %Verschiebungstakte runden
32    for j=1:data_anz %Von 1 bis Anzahl der Audiodaten
33        clear zwischen; %zwischen wird gelöscht
34        zwischen = data(:,j)*daten_daempfung(j,:); %Audiodateien
              werden nach den Parametern gedämpft
35        zwischen(end+1:end+max(verschieben),:) = 0; % Die Länge der
              Audiodaten wird um die Länge der größten Verschiebung
              verlängert
36        for k=1:length(verschieben) %Von 1 bis Anzahl der
              Reflexionen
37            zwischen(1+verschieben(k):end-max(verschieben)+
                  verschieben(k),k) = zwischen(1:end-max(verschieben),
                  k); %Verzögert die Audiosignale um die
                  Laufzeitverzögerung
38            zwischen(1:verschieben(k), k) = 0; %Anfang der
                  Audiodaten auf Null setzen (Anlaufverzögerung=
                  Laufzeitverzögerung)
39            sprintf('Mikrofon␣%d␣Audioquelle␣%d␣Reflexion␣%d',i,j,k
                  )
40        end
41        zwischen2(j,:) = sum(zwischen'); %Die Audiodaten werden
              summiert
42    end
43    clear out; %out wird gelöscht
44    if data_anz>1
45      out = sum(zwischen2)'; %zwischen2 wird summiert
46    else
47      out = zwischen2';
48    end
49    out = out./max(out); %out wird auf 1 normiert
50    outfile_name = ['wav\out' num2str(i)]; %Name für die
          Ausgabeaudiodatei wird erstellt
51    wavwrite(out ,fsim, outfile_name); %out wird in eine Audiodatei
          abgespeichert
52  end
53  erfolgreich = 1; %Rückmeldung: Simulation erfolgreich
```

Nachfolgend werden einige Zeilen näher erläutert:

In Zeile 29,

```
verschieben = fsim.*daten_laufzeiten;,
```

wird ermittelt, um wie viele Takte die Audiodaten verschoben werden. Nach der Rechenregel $n_{\text{Takte}} = f_{\text{sim}} \cdot t_{\text{Tot}}$ wird die Anzahl der Takte bestimmt, um die das Signal verschoben wird. Anschließend wird die Anzahl der Taktverschiebungen noch gerundet.

In der Zeile 33,

```
zwischen = data(:,j)*daten_daempfung(j,:);,
```

werden die Audiodaten mit den ermittelten Dämpfungsparametern verrechnet. Dies geschieht mit einer Matritzenmultiplikation, wobei `data(:,j)` ein Spaltenvektor ist und `daten_daempfung(j,:)` ein Zeilenvektor. Die Multiplikation von Spaltenvektor mal Zeilenvektor ergibt eine zweidimensionale Matrix. Im konkreten Fall eine Matrix mit je Spalte entsprechend Audiodaten gedämpft um jeweiligen Dämpfungsparameter und je Zeile entsprechend einen Audiodatenwert. In Formel 6.2 [8, Seite 15] wird der mathematische Zusammenhang dargestellt.

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \tag{6.1}$$

$$c_{ik} = \sum_{j=1}^{n} \{a_{ij} \cdot b_{jk}\} \tag{6.2}$$

Für die Verschiebung der Signale wurden speziell MATLAB-Syntax verwendet, die diese weitaus schneller verarbeitet, als eine Verschiebung des Signal in einer **`for`** -Schleife.

In Zeile 36,

```
zwischen(1+verschieben(k):end-max(verschieben)+ verschieben
(k),k)= zwischen(1:end-max(verschieben),k);,
```

werden die Daten verschoben, wobei gleichzeitig der Datensatz um die längste vorkommende Verschiebung verlängert wird. Der Anfang des Audiodatensatzes wird

in Zeile 37,

```
zwischen(1:verschieben(k), k)= 0;,
```

mit Nullen aufgefüllt.

Für die gesamte Funktion wurde der Übersichtlichkeit halber ein Flussdiagramm in Abbildung 6.5 erstellt.

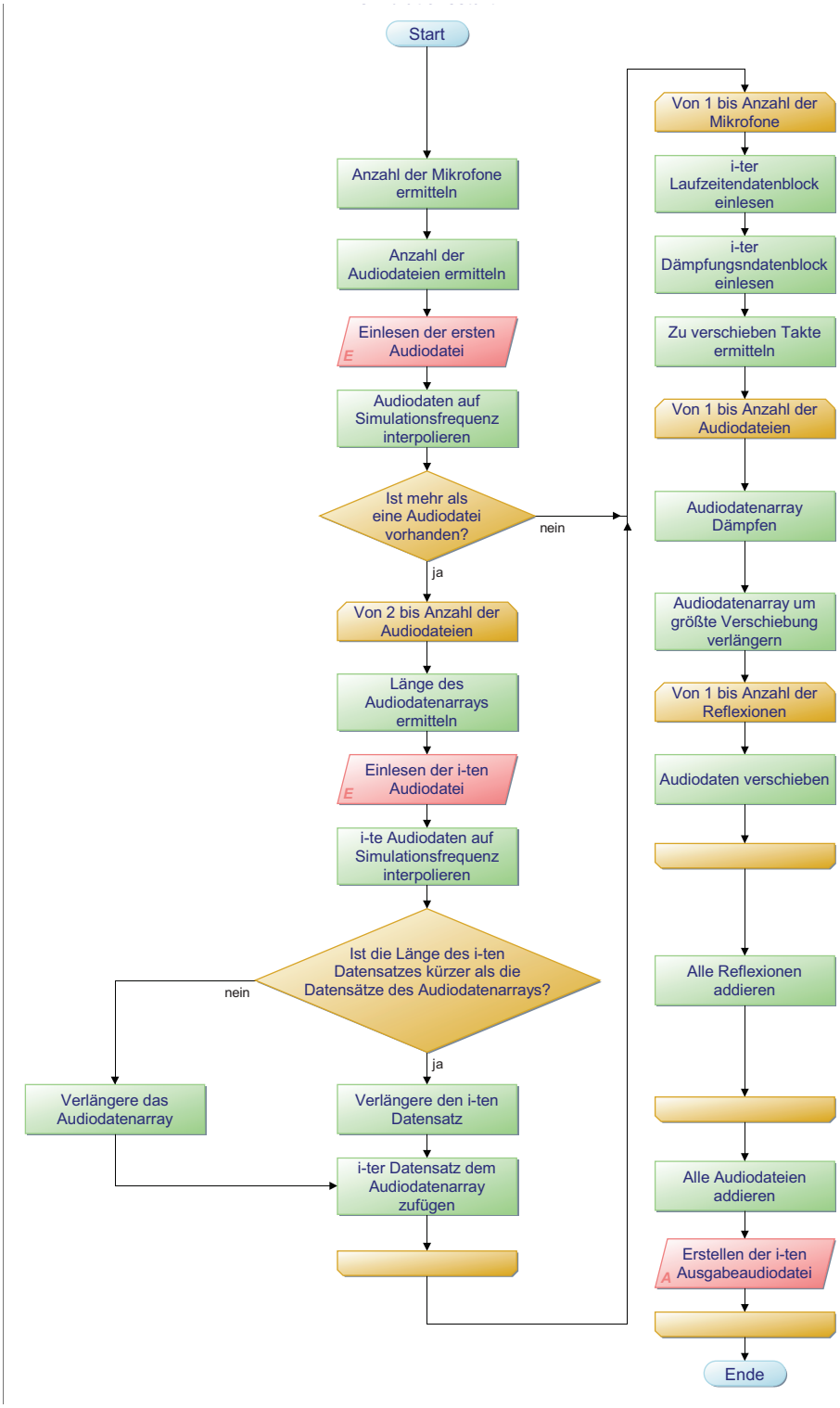


Abbildung 6.5.: Flussdiagramm für die Funktion simulationsstart()

7. Frequenzabhängige Oberflächenreflexionen

Trifft eine Schallwelle auf eine Oberfläche, so wird die Schallwelle an der Oberfläche reflektiert. Dieses Phänomen tritt klar erkenntlich auf, wenn man beispielsweise in einem leeren Raum steht. Im leeren Raum ist die Sprache stark verhallt.

Da die Oberfläche von den Wänden unterschiedlich hart ist, Oberflächenstrukturiert und unterschiedliche Formen hat, reflektiert der Schall je nach Wellenlänge unterschiedlich gut. Angegeben werden die Reflexionseigenschaften meist als Arbsorption α in Abhänigkeit der Frequenz. Tabelle 7.1 listet verschiedene Absorptionsfaktoren für verschiedene Wandtypen. Der Reflexionsfaktor ρ berechnet sich aus $\rho = 1 - \alpha$[14].

Decken	$125Hz$	$250Hz$	$500Hz$	$1000Hz$	$2000Hz$	$4000Hz$
Beton oder Fliesen	0,01	0,01	0,15	0,02	0,02	0,02
Akustische Deckenfliesen	0,70	0,66	0,72	0,92	0,88	0,75
Fiberglas: Spray $5'$	0,05	0,15	0,45	0,70	0,80	0,80
Rigips $1/2'$ $16'$ auf Mitte	0,29	0,10	0,05	0,04	0,07	0,09
Teppich auf Schaumgummi	0,08	0,24	0,57	0,69	0,71	0,73

Fussboden	$125Hz$	$250Hz$	$500Hz$	$1000Hz$	$2000Hz$	$4000Hz$
Linoleum/Vinylfliesen auf Beton	0,02	0,03	0,03	0,03	0,03	0,02
Beton oder Fliesen	0,01	0,01	0,15	0,02	0,02	0,02
Holz auf Balken	0,15	0,11	0,10	0,07	0,06	0,07
Teppich auf Schaumgummi	0,08	0,24	0,57	0,69	0,71	0,73

Wand	$125Hz$	$250Hz$	$500Hz$	$1000Hz$	$2000Hz$	$4000Hz$
Grob-Beton - rau	0,36	0,44	0,31	0,29	0,39	0,25
Grob-Beton - gestrichen	0,10	0,05	0,06	0,07	0,09	0,08
Rigips $1/2'$ $16'$ on center	0,29	0,10	0,05	0,04	0,07	0,09
Teppich auf Schaumgummi	0,08	0,24	0,57	0,69	0,71	0,73

Fensterfront	$125Hz$	$250Hz$	$500Hz$	$1000Hz$	$2000Hz$	$4000Hz$
Vorhang: 470 $\frac{g}{m^2}$ Stoff	0,07	0,31	0,49	0,75	0,70	0,6
Glas: Fenster	0,35	0,25	0,18	0,12	0,07	0,04

Tabelle 7.1.: Frequenzabhänige Absorptionsgrade [11]

Die hier aufgelisteten Absorptionsfaktoren wurden im Simulationsprogramm in der Funktion *filter_coefficienten()* hinterlegt. Bei Aufrufen der Funktion werden Simulationsfrequenz und Filternummer als Übergabeparameter übergeben. Der Rückgabewert sind 256 Filtercoefficienten. Für die Berechnung wird die MATLAB-Funktion *firls()* verwendet, die aus der Filterspezifikation die Filterkoeffizienten berechnet.

Die Funktion für den Simulationsstart wurde für die Simulation mit frequenzabhänigen Oberflächen, wie im Listing 7.1 dargestellt, modifiziert. Wie in Zeile 4 und 10 entnommen werden kann, wurde die MATLAB-Funktion *filter()* verwendet. In einem ersten Versuch wurde der Filter in einer **for** -Schleife implementiert. Die MATLAB eigene Funktion *filter()* ist jedoch um ein vielfaches schneller. Die Verschiebung des Signals durch die Filterung wird in Zeile 5 und Zeile 11 rausgerechnet.

Listing 7.1: Ausschnitt aus Raumsimulations-Funktion mit Frequenzabhängigen Oberflächen

```
1        for k=1:length(verschieben) %Von 1 bis Anzahl der
             Reflexionen
2            if k>1 %erste Reflextion
3                b = filter_coefficienten(ref_I(k-1), fsim); %
                     Filtercoefficienten werden ermittelt
4                zwischen(:,k) = filter(b,1,zwischen(:,k)); %
                     Audiodaten werden gefilterert
5                zwischen(1:end-256/2,k) = zwischen(256/2+1:end,k);
                     %Verschiebung des gefilterten Signals um die
                     hälfte der Filterkoeffizienten
6                zwischen(end-256/2+1:end,k) = 0;
7            end
8            if k>7 %zweite Reflexion
9                b = filter_coefficienten(ref_II(k-7), fsim); %
                     Filtercoefficienten werden ermittelt
10               zwischen(:,k) = filter(b,1,zwischen(:,k)); %
                     Audiodaten werden gefilterert
11               zwischen(1:end-256/2,k) = zwischen(256/2+1:end,k);
                     %Verschiebung des gefilterten Signals um die
                     hälfte der Filterkoeffizienten
12               zwischen(end-256/2+1:end,k) = 0;
13           end
14           zwischen(1+verschieben(k):end-max(verschieben)+
                 verschieben(k),k) = zwischen(1:end-max(verschieben),
                 k); %Verzögert die Audiosignale um die
                 Laufzeitverzögerung
15           zwischen(1:verschieben(k), k) = 0; %Anfang der
                 Audiodaten auf Null setzen (Anlaufverzögerung=
                 Laufzeitverzögerung)
16           sprintf('Mikrofon␣%d␣Audioquelle␣%d␣Reflexion␣%d',i,j,k
                 ) %Anzeige zum aktuellen Simulationsfortschritt
17       end
```

In Abbildung 7.1 und Abbildung 7.2 wird beispielhaft der Filterentwurf für akustische Deckenfliesen bei $88kHz$ und $460kHz$ Simulationsfrequenz dargestellt. In Kapitel 8 wird auf die

Notwendigkeit der hohen Simulationsfrequenzen von im Beispiel verwendeten $460kHz$ eingegangen. Es ist aus den beiden Abbildungen ersichtlich, dass die Genauigkeit des Filters abhängig von der Simulationsfrequenz ist, wie im Amplitudengang des Filters abzulesen ist. Je höher die Simulationsfrequenz, desto höher liegt die für den Filterentwurf verwendete Abtastfrequenz. Dies führt zu einer höheren Steilheit der Flanken relativ zur Abtastfrequenz. Da die Anzahl der Filterkoeffizienten auf 256 festgelegt wurde, verschlechtert sich das Verhältnis von Filterspezifikation zu Filterentwurf. Der Phasengang der beiden Filterentwürfe ist im betrachteten Frequenzbereich annähernd $0°$. Dies ist auch im Ausgangssignal zu erkennen.

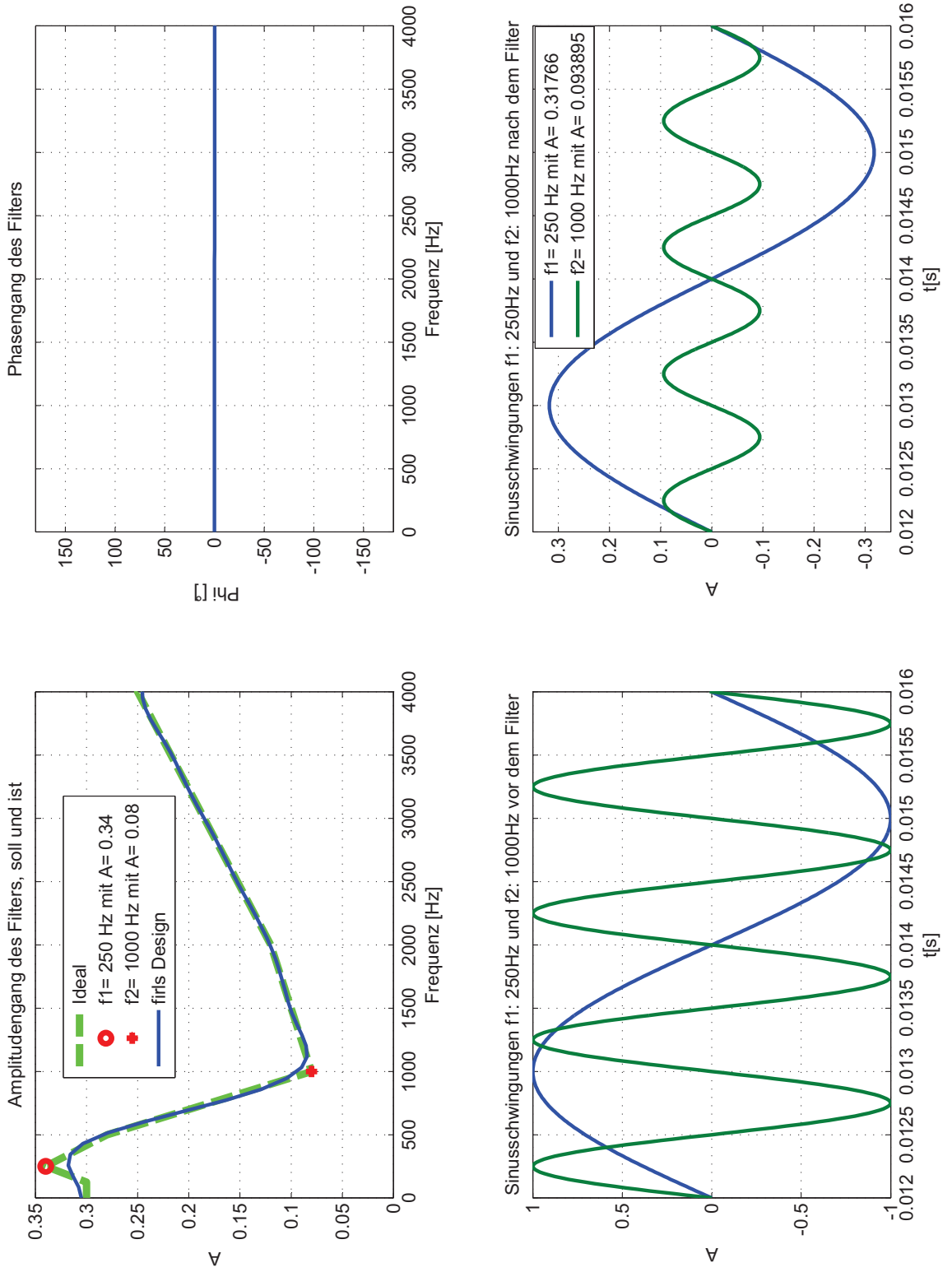


Abbildung 7.1.: Filterentwurf der akustischen Deckenfliesen nach Tabelle 7.1 bei einer Simulationsfrequenz von $88 kHz$

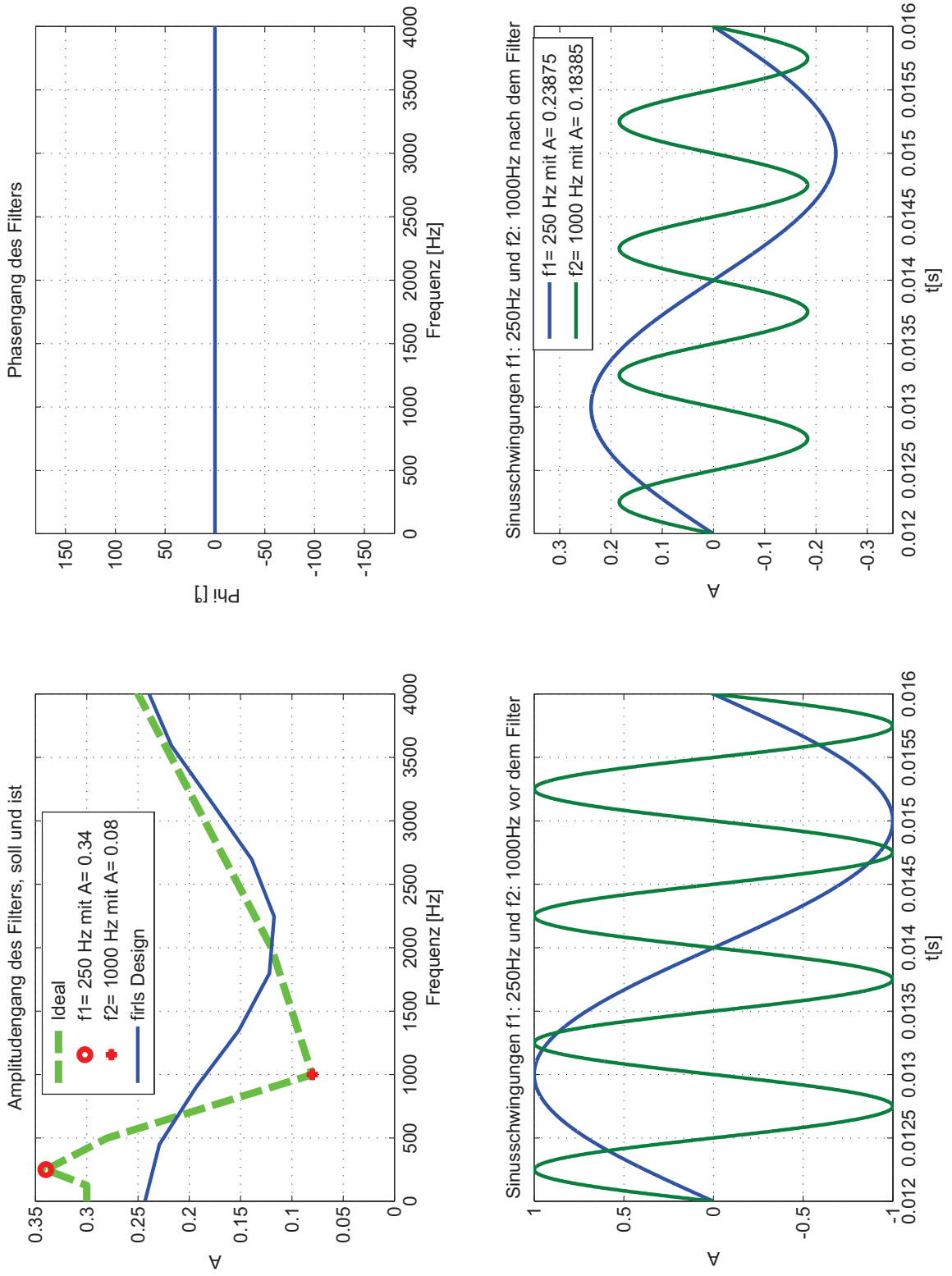


Abbildung 7.2.: Filterentwurf der akustischen Deckenfliesen nach Tabelle 7.1 bei einer Simulationsfrequenz von $460 kHz$

8. Fehler und Genauigkeiten

Bei einer Simulation ist die Genauigkeit immer eine wichtige Größe zur Beurteilung der Simulationsergebnisse. Bei der akustischen Raumsimulation ist die Genauigkeit ausschlaggebend für die Winkelauflösung. Die Winkelauflösung ist die für den Anwender entscheidende Größe. Im Folgenden wird die Winkelauflösung berechnet bzw. die verschiedenen Problematiken bezüglich der Winkelauflösung interpretiert. Eine Größe hat entscheidene Auswirkungen auf die Winkelauflösung, dies ist die zeitdiskrete Auflösung der Simulation. Zum besseren Verständnis wird vorerst das Mikrofonarray erklärt.

8.1. Mikrofonarray

Um eine Sprachquelle mit Hilfe von Unterraumverfahren- Algorithmen zu orten, muss ein Mikrofonarray verwendet werden. Ein Mikrofonarray sind mehrere nebeneinander angeordnete Mikrofone. Die Anordnung kann zunächst beliebig gewählt werden. Am einfachsten ist die Anordnung in einer Reihe. Der Abstand zwischen den Mikrofonen ist dabei ein abhängiger Parameter, wie im Weiteren beschrieben wird.

8.1.1. Abstand zwischen den Mikrofonen

In Abbildung 8.1 ist ein Mikrofonarray systematisch dargestellt. Der Abstand x wird vom Mittelpunkt der Mikrofone aus gemessen.

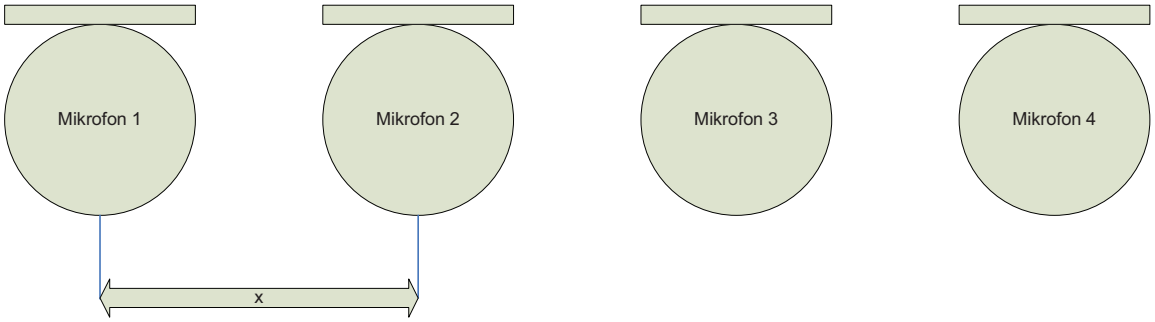


Abbildung 8.1.: Mikrofonarray in einer Reihe

Der maximale Abstand zwischen den Mikrofonen wird durch die höchste betrachtete Frequenz bestimmt, da zwischen zwei Mikrofonen nicht mehr als eine halbe Periode der höchsten betrachteten Frequenz passen darf, um Doppeldeutigkeiten auszuschließen. Soll das gesamte System Sprachquellen orten, ist eine Frequenz von $4kHz$ maximal erforderlich. In Formel 8.1 und Formel 8.2 wird der maximale Abstand x zwischen den Mikrofonen berechnet.

$$x = \frac{v_{\text{Luft}} \cdot 2}{f_{\text{max}}} \tag{8.1}$$

x: maximaler Abstand zwischen zwei Mikrofonen.

v_{Luft}: Schallausbreitungsgeschwindigkeit in der Luft $343\frac{m}{s}$ (Herleitung und Berechnung in Kapitel 5.1 auf Seite 29)

f_{max}: maximale betrachtete Frequenz $4kHz$

$$x = \frac{343\frac{m}{s} \cdot 2}{4kHz} = 0{,}042875m \tag{8.2}$$

8.1.2. Umrechnung von Laufzeiten in Winkel

Um den Winkel zu bestimmen, aus dem die Schallwelle auf das Mikrofonarray trifft, muss zuerst die maximale Zeit t_{max} ermittelt werden, die eine Schallwelle beim Durchlauf längs des Mikrofonarrays benötigt.

$$t_{\text{max}} = \frac{x}{v_{\text{Luft}}} = \frac{0{,}042875m}{343\frac{m}{s}} = 125\mu s \tag{8.3}$$

Die tatsächliche Laufzeitverzögerung t_0, die die Schallwelle bei Eintritt unter einem bestimmsten Winkel benötigt wird schematisch in Abbildung 8.2 dargestellt.

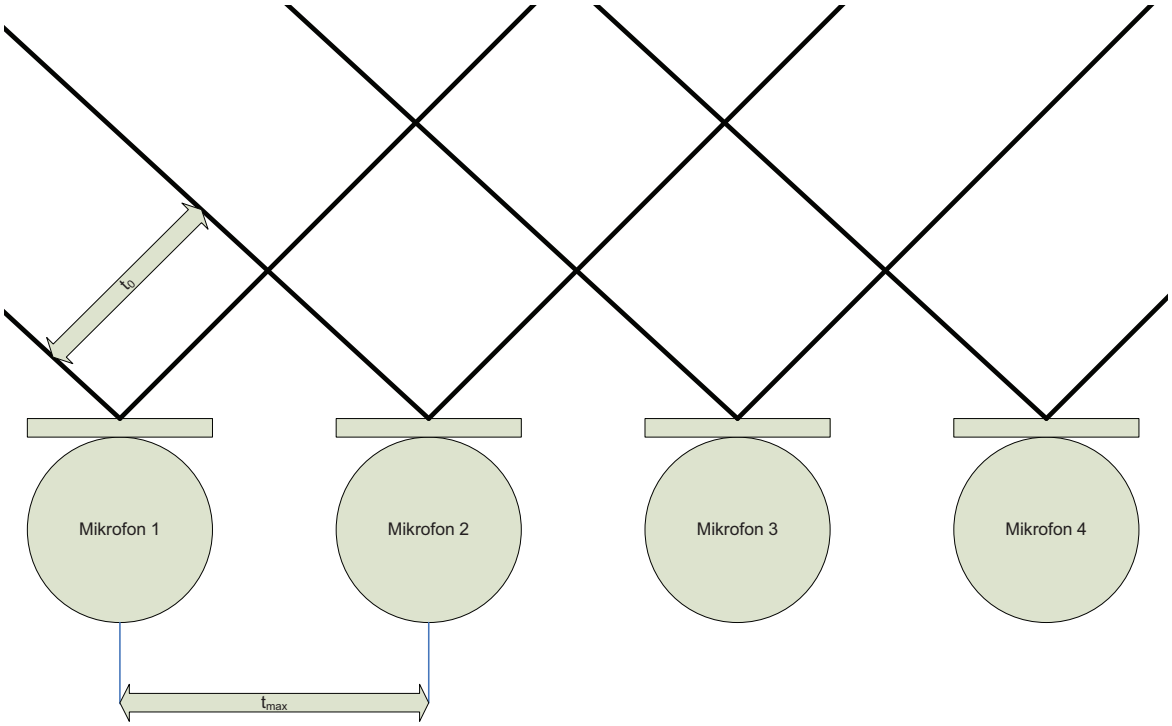


Abbildung 8.2.: Laufzeit, die eine Schallwelle beim Durchlaufen eines Mikrofonarrays benötigt

Der Arkussinus vom Verhältnis von tatsächlicher Laufzeitverzögerung zu maximaler Laufzeitverzögerung ergibt den Winkel θ aus dem die Schallwelle kommt.

$$\theta = \arcsin \frac{t_0}{t_{max}} \tag{8.4}$$

θ: Winkel aus dem die Schallwelle auf das Mikrofonarray auftrift.

t_0: tatsächliche Laufzeitverzögerung der Schallwelle von einem zum nächstem Mikrofon

t_{max}: maximale Laufzeitverzögerung bei Durchlauf längs des Mikrofonarrays

8.2. Winkelauflösung in verschiedenen Winkelbereichen

Betrachtet man den Arkussinus in Abbildung 8.3 sieht man, dass von $-40°$ bis $40°$ ein linearer Verlauf vorhanden ist. Jenseits von $40°$ steigt die kurve immer stärker an. Will man jenseits von $40°$ eine genaue Winkelauflösung muss man dafür eine höhere Abtastrate wählen, als das für eine genaue Winkelauflösung im Bereich von $-40°$ bis $40°$ nötig ist. Winkel höher als $80°$ benötigen eine sehr hohe Abtastrate. An dieser Stelle hat der Arkussinus eine sehr große Steigung. Bei $90°$ ist die Kurve senkrecht.

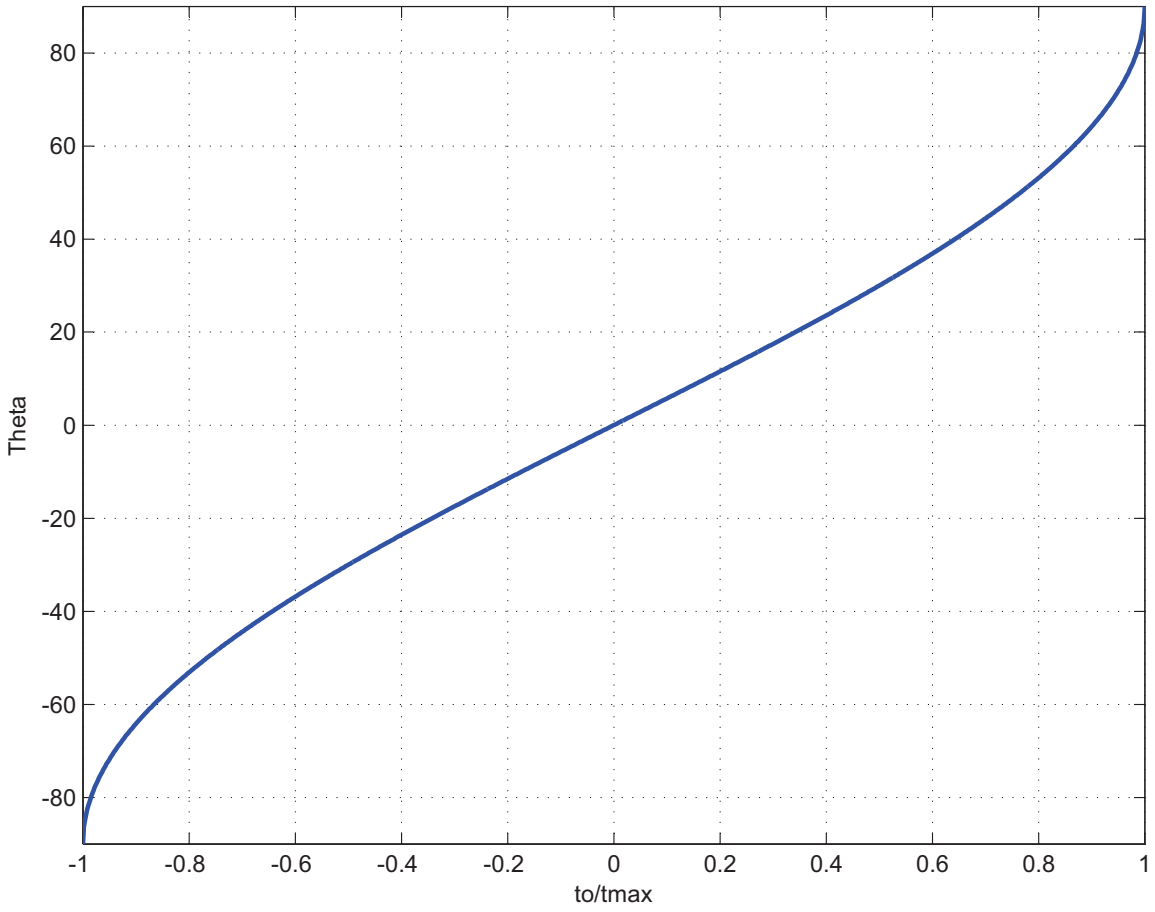


Abbildung 8.3.: Arkussinus vom Verhältnis $\frac{t_0}{t_{max}}$

Tabellarisch wird in Tabelle 8.1 die minimale Abtastfrequenz berechnet, die sich bei verschiedenen Ausgangswinkeln bei 1° Winkelauflösung ergibt. Die berechneten Werte sind in Abbildung 8.4 in Abhänigkeit vom Winkel dargestellt.

$\theta_a[^\circ]$	$\theta_b[^\circ]$	$\frac{t_{0,a}}{t_{\max}}$	$\frac{t_{0,b}}{t_{\max}}$	$\frac{t_{0,b}-t_{0,a}}{t_{\max}}$	$T[s]$	$f_a[kHz]$
0	1	0,000	0,0175	0,0175	0,00000218	458
10	11	0,174	0,191	0,0172	0,00000215	466
20	21	0,342	0,358	0,0163	0,00000204	489
30	31	0,500	0,515	0,0150	0,00000188	532
40	41	0,643	0,656	0,0133	0,00000166	603
50	51	0,766	0,777	0,0111	0,00000139	721
60	61	0,866	0,875	0,00859	0,00000107	931
70	71	0,940	0,946	0,00583	0,000000728	1373
80	81	0,985	0,988	0,00288	0,000000360	2777
89	90	0,999848	1	0,000152	0,0000000190	52526

Tabelle 8.1.: Auflistung der benötigten Abtastfrequenzen für 1° Winkelauflösung bei verschiedenen Winkeln

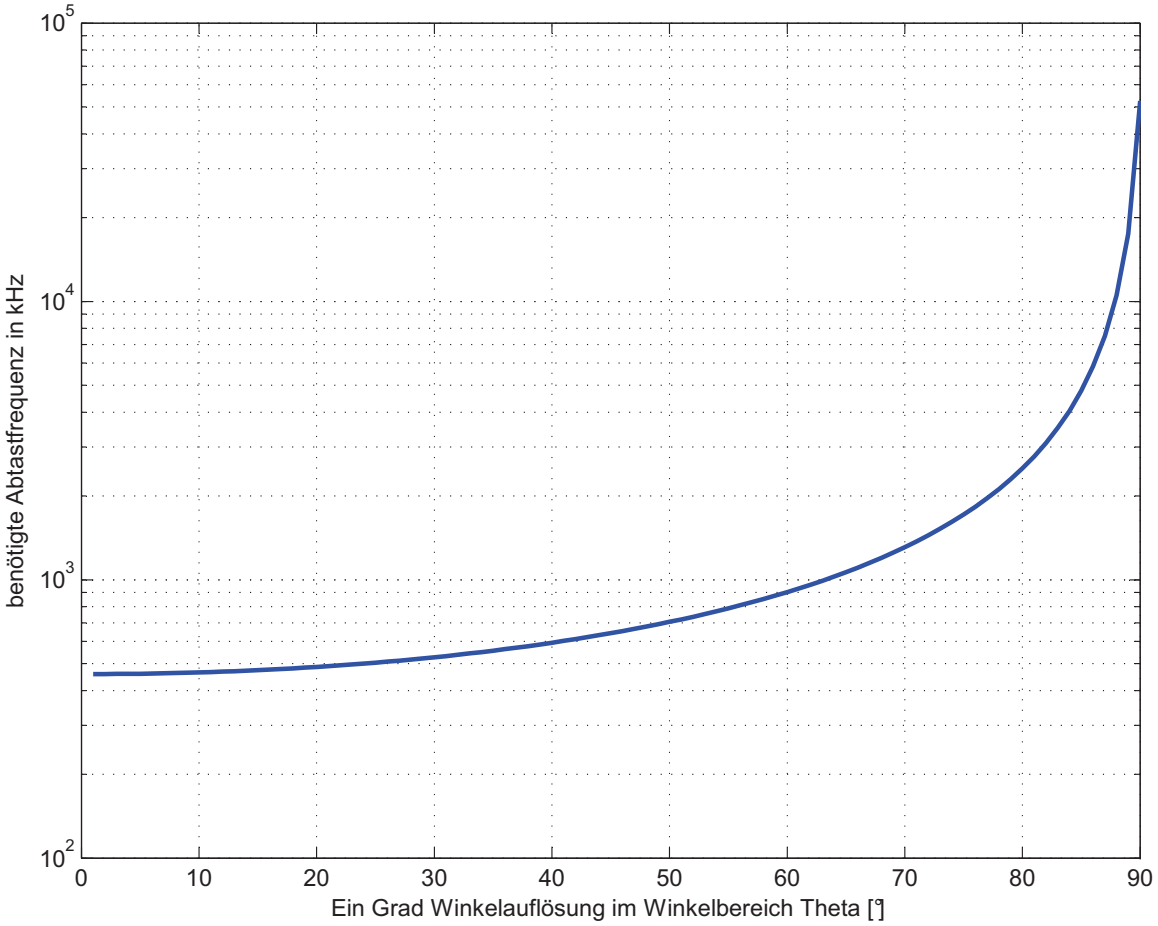


Abbildung 8.4.: Benötigte Abtastfrequenz bei $1°$ Winkelauflösung in Abhängikeit des Winkelbereichs

Die steigende Abtastfrequenz im Winkelrandbereich hat zur Folge, dass die Winkelauflösung nicht an allen Punkten gleich groß ist. Im Bereich von $0°$ ist die Winkelauflösung weitaus höher als am Winkelrandbereich, da die Abtastfrequenz sowohl für die Simulation als auch bei einem Hardwareaufbau fest eingestellt werden muss. Das Simulationprogramm kann eine Winkelauflösung nur in einem definierten Winkelbereich gewährleisten. So kann z.B. eine Winkelauflösung von $1°$ im Bereich von $-40°$ bis $40°$ bei einer Abtastfrequenz von $603 kHz$ gewährleistet werden.

8.2.1. Benötigte Winkelauflösung

Es stellt sich die Frage, welche Winkelauflösung ist nötig. Wie in der Einleitung erwähnt, soll die Sprachlokalisation zum Zweck der Vorlesungsaufnahme realisiert werden. Dabei ist

nicht festgelegt, in welcher Distanz die Kamera zur Aufnahme der Vorlesung zum Professor steht. Denkbar ist die feste Montage an der Decke in unmittelbarer Nähe der Tafel, oder das Aufstellen am anderen Ende des Seminarraums hinter den Studenten. Je weiter die Kamera entfernt vom aufzunehmenden Objekt ist, desto gravierender ist eine Abweichung bedingt durch die Winkelauflösung. Bleibt man bei dem aufgeführten Beispiel, bei dem eine Vorlesung aufgenommen werden soll, sollte der Professor samt Tafel aufgenommen werden. Da die Distanz nicht im Vorwege festzulegen ist, werden drei verschiedene Szenarien aufgeführt.

1. Eine Montage der Kamera an der Decke des Seminarraums in Abstand $r = 3m$,
2. ein Aufstellen der Kamera hinter den Studenten bei einem sehr kurzen Seminarraum im Abstand $r = 7m$ und
3. ein Aufstellen der Kamera hinter den Studenten bei einem mittelgroßen Seminarraum im Abstand $r = 10m$.

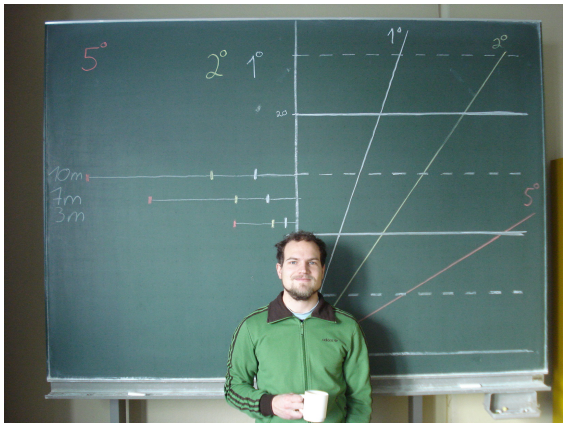
Betrachtet werden $\theta_a = 1°, \theta_a = 2°$ und $\theta_a = 5°$ Winkelauflösung im günstigsten Fall, so dass die Kamera frontal zur Tafel steht. Es wird eine Tafel der Breite $210cm$ zugrunde gelegt, um prozentual ermitteln zu können, wie weit sich ein Sprecher aus der Mitte der Aufnahme entfernt hat.

$$x = \tan(\theta_a) \cdot r \tag{8.5}$$

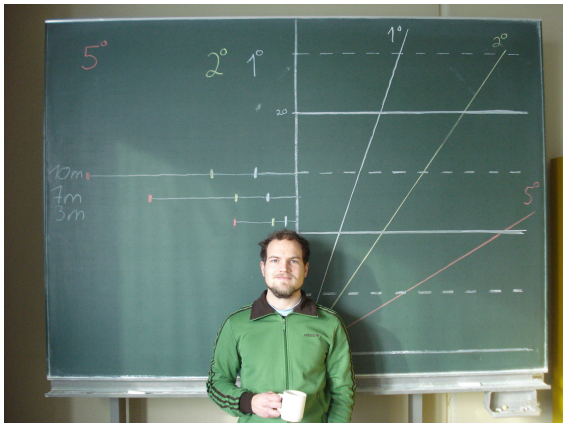
	$r = 300cm$		$r = 700cm$		$r = 1000cm$	
	Abweichung x aus der Mitte der Videoaufnahme					
$\theta_a = 1$	$5,2cm$	$5,0\%$	$12,2cm$	$11,6\%$	$17,5cm$	$16,6\%$
$\theta_a = 2$	$10,5cm$	$10,0\%$	$24,4cm$	$23,3\%$	$34,9cm$	$33,3\%$
$\theta_a = 5$	$26,2cm$	$25,0\%$	$61,2cm$	$58,3\%$	$87,5cm$	$83,3\%$

Tabelle 8.2.: Abweichung vom Mittelpunkt bei Vidoaufnahme

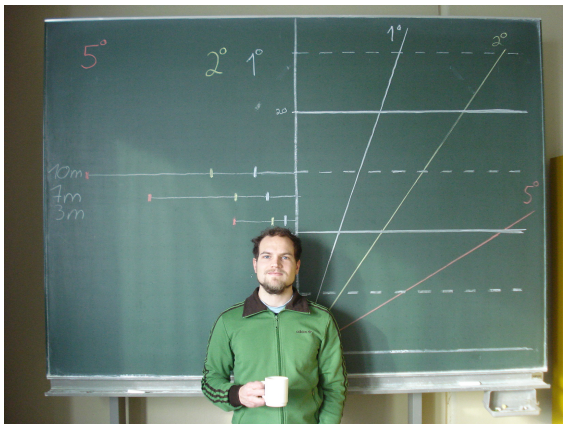
Wie groß die Abweichung des Sprechers aus der Mitte der Videoaufnahme maximal sein darf, kann nicht absolut bestimmt werden. Es obliegt der subjektiven Wahrnehmung des Betrachters. Deshalb sind die berechneten Abweichungen in Bildern dargestellt. Somit kann sich der Leser selbst ein Urteil über die notwendige Winkelauflösung machen. Hier sei noch zu erwähnen, dass die Kamerasteuerung auch nur die Winkelauflösung als Parameter übergeben bekommt, somit wird das Bild ruckartig dem Sprecher folgen.



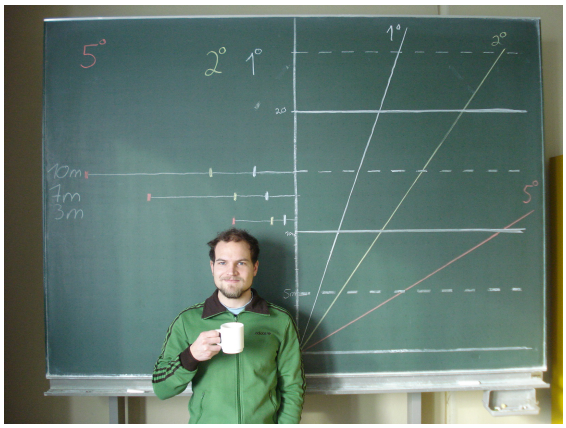
(a) $0°$ Winkelauflösung



(b) $1°$ Winkelauflösung

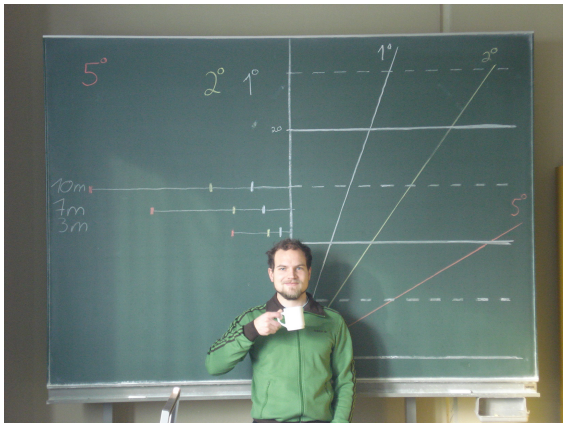


(c) $2°$ Winkelauflösung

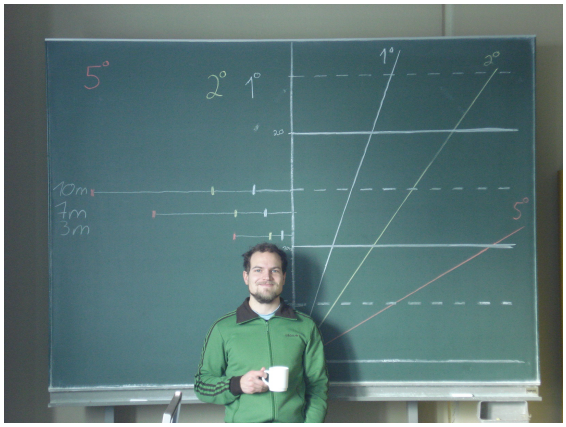


(d) $5°$ Winkelauflösung

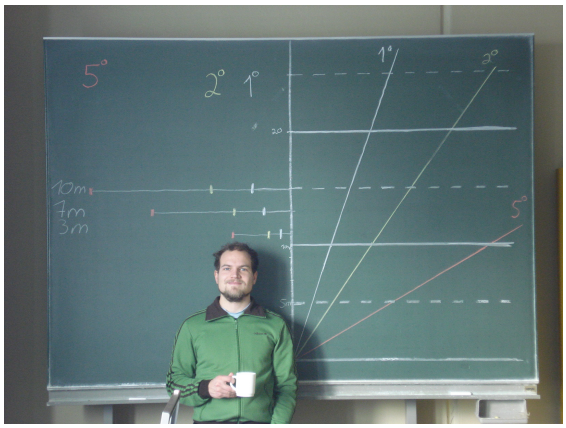
Abbildung 8.5.: Bildaufnahme aus $3m$ Entfernung



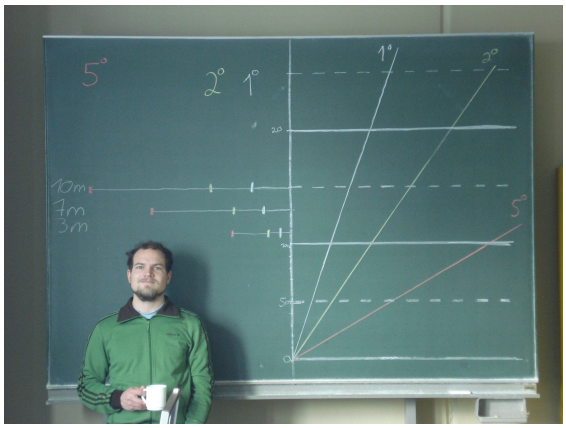
(a) $0°$ Winkelauflösung



(b) $1°$ Winkelauflösung



(c) $2°$ Winkelauflösung

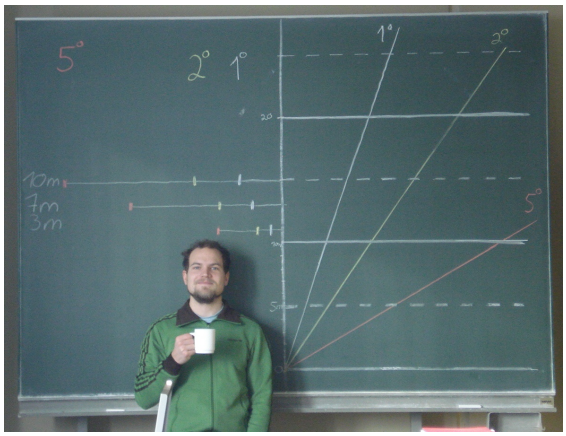


(d) $5°$ Winkelauflösung

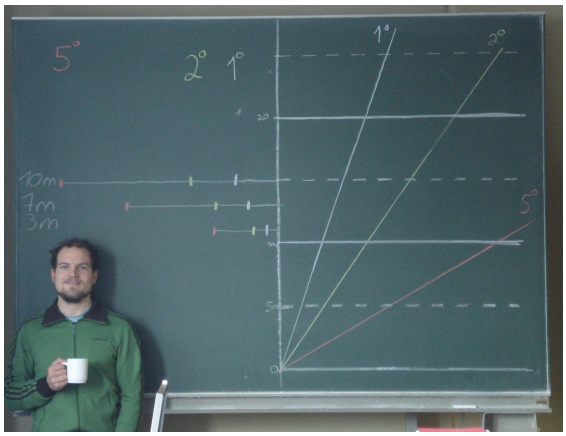
Abbildung 8.6.: Bildaufnahme aus $7m$ Entfernung

(a) $0°$ Winkelauflösung

(b) $1°$ Winkelauflösung



(c) $2°$ Winkelauflösung



(d) $5°$ Winkelauflösung

Abbildung 8.7.: Bildaufnahme aus $10m$ Entfernung

8.2.2. Verschiedene Winkelauflösungen und die Implementierung im Simulationsprogramm

Zum Schluss des Kapitels 8.2 werden in Tabelle 8.3 noch ein paar markante Abtastfrequenzen angegeben, die sich aus verschiedenen Winkelauflösungen und verschiedenen Öffnungswinkel ergeben.

	$\theta = 0°$	$\theta = 5°$	$\theta = 10°$	$\theta = 20°$	$\theta = 40°$
$\theta_a = 0{,}5°$	$f_a = 917kHz$	$f_a = 921kHz$	$f_a = 932kHz$	$f_a = 977kHz$	$f_a = 1201kHz$
$\theta_a = 1°$	$f_a = 458kHz$	$f_a = 460kHz$	$f_a = 466kHz$	$f_a = 489kHz$	$f_a = 603kHz$
$\theta_a = 2°$	$f_a = 229kHz$	$f_a = 230kHz$	$f_a = 233kHz$	$f_a = 246kHz$	$f_a = 304kHz$
$\theta_a = 3°$	$f_a = 153kHz$	$f_a = 154kHz$	$f_a = 156kHz$	$f_a = 164kHz$	$f_a = 204kHz$
$\theta_a = 4°$	$f_a = 115kHz$	$f_a = 115kHz$	$f_a = 117kHz$	$f_a = 124kHz$	$f_a = 154kHz$
$\theta_a = 5°$	$f_a = 92kHz$	$f_a = 92kHz$	$f_a = 94kHz$	$f_a = 99kHz$	$f_a = 124kHz$

Tabelle 8.3.: Abtastfrequenzen in Abhänigkeit der Winkelauflösung und des Öffnungswinkel

Da es sich bei der maximal zulässigen Abweichung des Sprechers aus der Mitte des Bildes um eine subjektive Entscheidung des Anwenders handelt, ist die Winkelauflösung und der Öffnungswinkel im Simulationsprogramm parametrierbar. Aus Winkelauflösung und Öffnungswinkel berechnet das Simulationsprogramm die benötigte Simulationsfrequenz. Folgend wird der zugehörige MATLAB-Code im Listing 8.1 dargestellt.

Listing 8.1: Berechnung der benötigten Simulationsfrequenz

```
1  function fa = Simulationsfrequenz_berechnen(Winkelaufloesung,
       Oeffnungswinkel, micarr)
2  %Rückgabewert(fa): Berechnete Simulationsfrequenz
3  %Übergabeparameter(Winkelaufloesung): gewünschte Winkelauflösung
4  %Übergabeparameter(Oeffnungswinkel): gewünschter Oeffnungswinkel
       in dem die Winkelauflösung garantiert werden soll
5  %Übergabeparameter(micarr): Koordinaten der einzelnen Mikrofone
6  v_luft = 343;
7  P12 = micarr(1,:)-micarr(2,:); % Berechnung der Strecke zwischen
       dem ersten und zweiten Mikrofon
8  P12abs = sqrt(P12(1)^2+P12(2)^2+P12(3)^2); % Berechnung der
       Strecke zwischen dem ersten und zweiten Mikrofon
9  t_max = P12abs/v_luft; % Berechnung der maximalen Laufzeit
       zwischen zwei Mikrofonen
10 % Berechnung der benötigten Abtastfrequenz
11 verhaeltnis1 = sin((Winkelaufloesung+Oeffnungswinkel)/180*pi); %
       Winkelverhältnis am Rand des Öffnungswinkel plus
       Winkelauflösung
```

```
12 verhaeltnis2 = sin(Oeffnungswinkel/180*pi); % Winkelverhältnis am
       Rand des Öffnungswinkel
13 verhaeltnisaenderung = verhaeltnis1 - verhaeltnis2; % Differenz
       zwischen den Winkelverhältnissen
14 Ta = verhaeltnisaenderung * t_max; % Simulationsschrittweite
15 fa = 1/Ta; % Simulationsfreuenz
16 fa_round = fa/1000; % Abtastfrequenz auf kHz umrechnen
17 fa = ceil(fa_round)*1000; %fa aufrunden und auf Hz umrechen
18 if fa<48000 % Wenn fa kleiner als 48kHz, dann 48kHz
19     fa = 48000;
20 end
```

8.3. Phasenfehler durch Wertediskrete Wave- Files

Betrachtet man einen Sinus der nicht nur Zeitdiskret sondern auch Wertediskret ist, dann kann nachgewiesen werden, dass ein Phasenfehler auch durch die begrenzte Bitauflösung zustande kommen kann. Das DSK-Board *TMS320C6713 DSK* hat eine Bitauflösung von $16Bit$. Wave- Files, die zur Simulation verwendet werden, können durchaus nur $8Bit$ haben.

Folgend wird ein Szenario erzeugt, welches den Grenzfall darstellt, ab dem ein Phasenfehler durch ein Quantisierungsfehler erzeugt werden kann. In Abbildung 8.8 sind zwei abgetastete Sinusschwingungen zueinander um $\frac{1}{f_a}$ verschoben. Die rote und grüne Linie zeigt den Toleranzbereich an, um wieviele Quantisierungsstufen die Sinusschwingung fehlerhaft abgetastet werden kann. Im dargestellten Beispiel sind es 4 Quantisierungsstufen. Ein solcher Fehler könnte durch Rauschen oder ein schlecht ausgepegeltes Mikrofon verursacht werden. Auf der Abbildung ist erkennbar, dass die um die Abtastzeit verschobene Sinusschwingung innerhalb des Quantisierungstoleranzbereiches liegt.

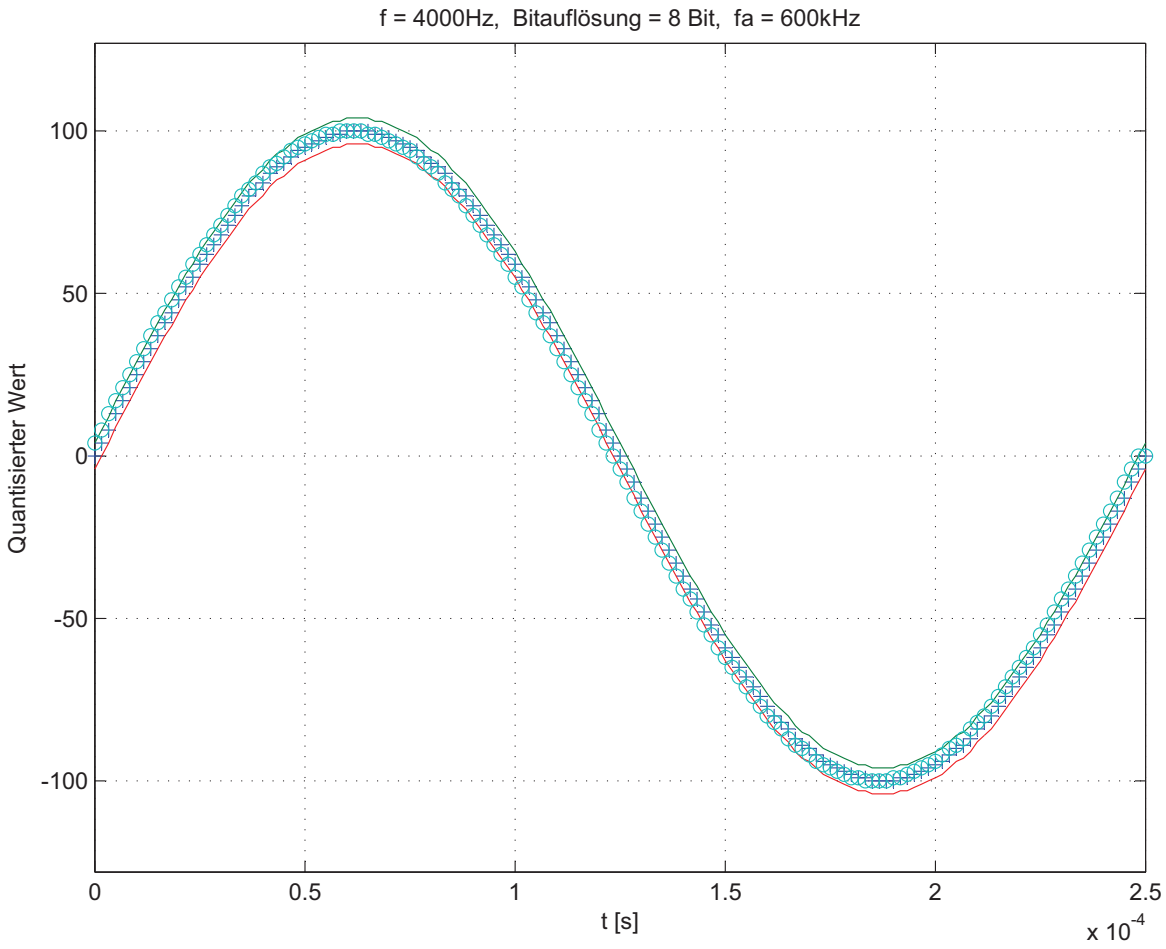


Abbildung 8.8.: Abgetastete Sinusschwingung mit um $\frac{1}{f_a}$ verschobenen Sinusschwingung

Ein Phasenfehler bedeutet, dass der Winkel fehlerhaft berechnet wird. Im konkreten Fall auf um $1°$ bei $40°$ bzw. um $0{,}76°$ bei $0°$.

Abbildung 8.9 zeigt eine genaue Betrachtung des Nulldurchgangs. Die abgetasteten Werte liegen genau auf der roten Toleranzgrenze. An der Abbildung ist zu erkennen, dass selbst bei Erhöhung der Abtastfrequenz der gleiche Fehler auftritt. Es kann keine höhere Winkelgenauigkeit durch eine höhere Abtastrate erreicht werden. Die Winkelgenauigkeit ist in diesem Fall durch die Bitauflösung begrenzt.

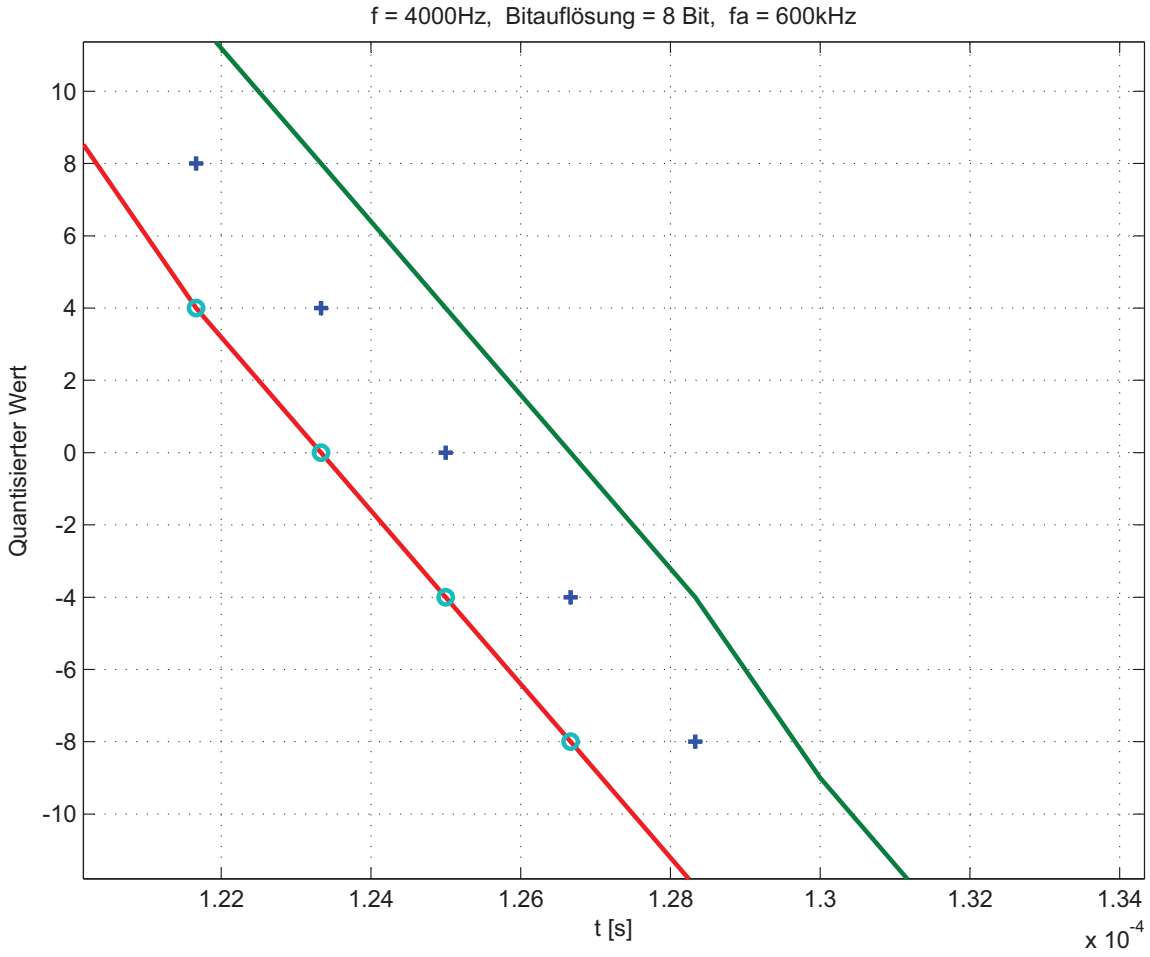


Abbildung 8.9.: Abgetastete Sinusschwingung mit um $\frac{1}{f_a}$ verschobenen Sinusschwingung beim Nulldurchgang

Abbildung 8.10 zeigt eine genaue Betrachtung des Maximums. In diesem Bereich ist aufgrund der geringen Steigung beim Maximum eine Phasenverschiebung von mehr als einer Abtastzeit möglich.

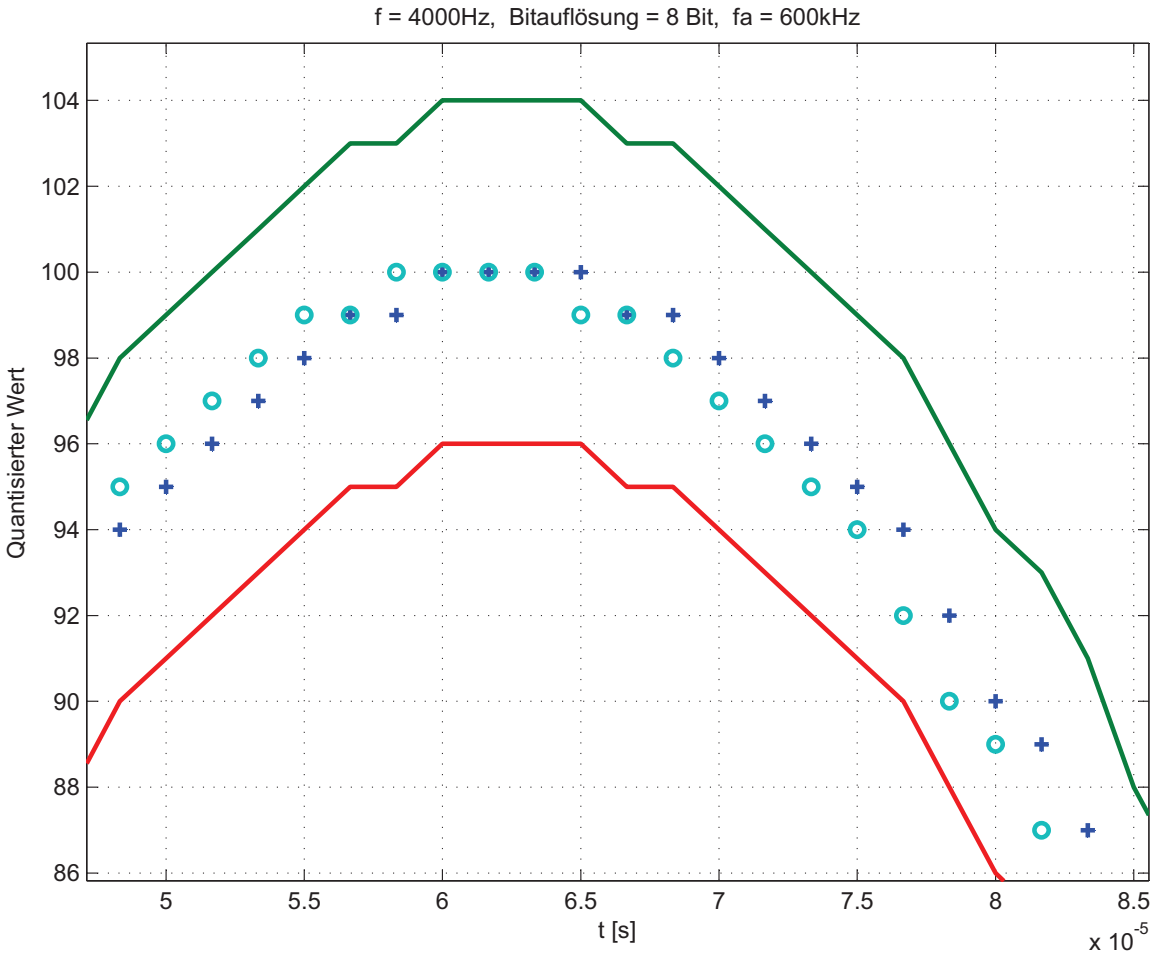


Abbildung 8.10.: Abgetastete Sinusschwingung mit um $\frac{1}{f_a}$ verschobenen Sinusschwingung beim Maximum

Wird ein Breitband Algorithmus getestet, werden auch niederfrequente Sinusschwingen untersucht, die weitaus niedrigere Frequenzen, als die für den Abstand der Mikrofone zugrunde gelegten Frequenz haben. In Abbildung 8.11 ist dieses Szenario aufgezeigt. Hier handelt es sich um eine $100Hz$ Sinusschwingung, bei der der Toleranzbereich auf eine Quantisierungsstufe verringert wurde. Und die Abtastfrequenz nur $90kHz$ beträgt, was einer Winkelauflösung von ca. $5°$ im Winkelbereich von $0°$ entspricht.

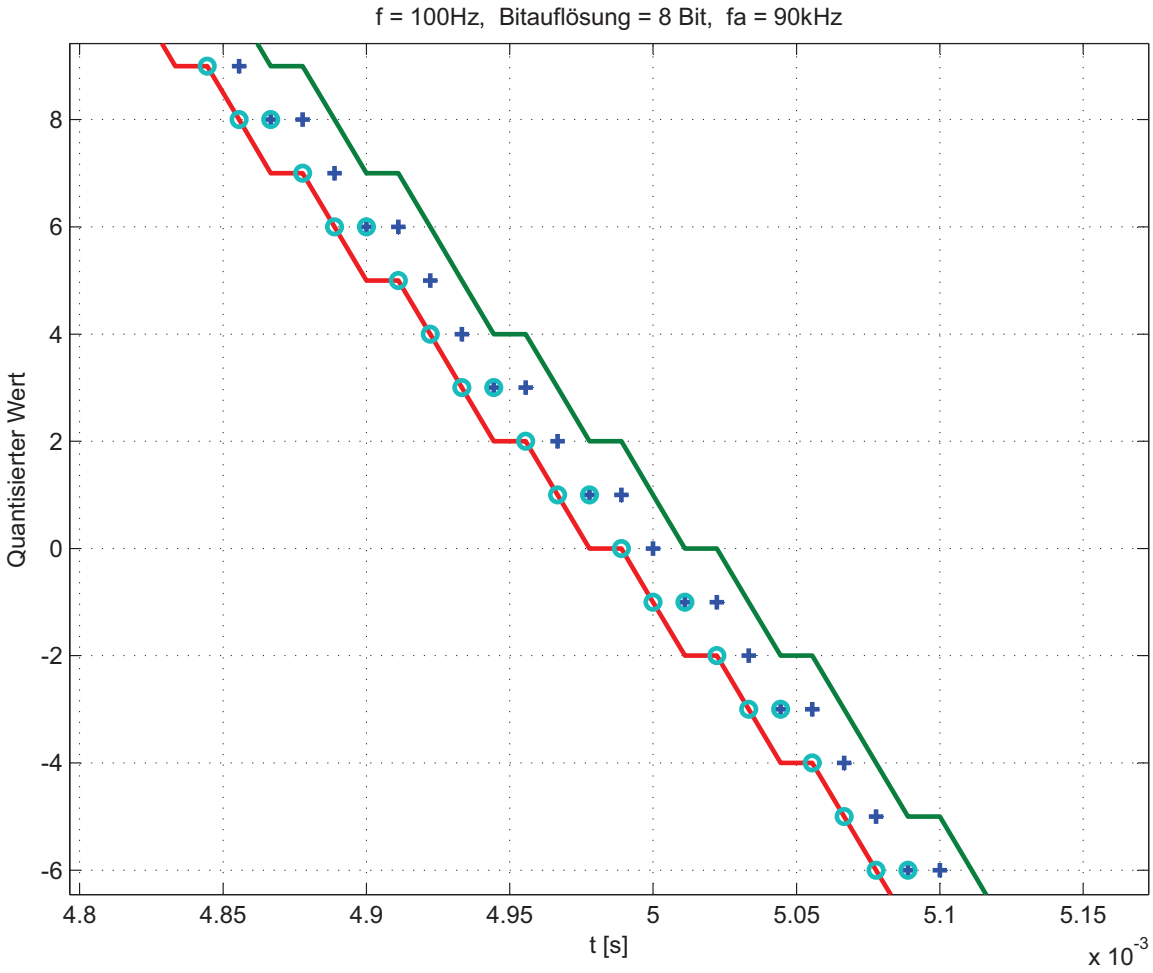


Abbildung 8.11.: Abgetastete Sinusschwingung mit um $\frac{1}{f_a}$ verschobenen Sinusschwingung beim Nulldurchgang bei $f = 100Hz$

Hierbei wird deutlich, dass Signale mit einer geringen Steigung bzw. niedrigen Frequenz aufgrund von Quantisierungsfehlern einen Phasenfehler erzeugen können.

9. Test

In diesem Kapitel wird das Simulationsprogramm mit unterschiedlichen Methoden getestet. Zuerst wurde die Genauigkeit der Simulation überprüft, anschließend wurde die Impulsantwort einer Raumsimulation aufgenommen und zum Schluss wurden die Simulationsausgangsdaten mit einem MUSIC.Algorithmus überprüft.

9.1. Untersuchung der Simulationsgenauigkeit

Zum Schluss wurde untersucht, ob das Programm richtige Ausgangssignale produziert. Der wichtigste Parameter dabei ist die korrekt ermittelte Laufzeitverzögerung. Um zu testen ob die Laufzeitverzögerung richtig simuliert wird, wurden die Wave-Dateien von Mikrofon 1 und Mikrofon 2 untersucht. In Abbildung 9.1 werden die beiden Zeitverläufe von Mikrofon 1 und Mikrofon 2 dagestellt, die sich bei folgenden Koordinaten ergeben haben:

P_{M1} Mikrofon 1 mit $x_{M1} = 3m$, $y_{M1} = 2m$, $z_{M1} = 1m$

P_{M2} Mikrofon 2 mit $x_{M2} = 3{,}045m$, $y_{M2} = 2m$, $z_{M2} = 1m$

P_A Audioquelle mit $x_A = 0{,}3m$, $y_A = 3m$, $z_A = 1{,}7m$

Mit Formel 4.4, Formel 4.5 und Formel 5.3 wurde die Laufzeitverzögerung zwischen den Mikrofonen berechnet.

$$\overrightarrow{a_1} = \overrightarrow{P_{M1}P_A} = \begin{pmatrix} x_A - x_{M1} \\ y_A - y_{M1} \\ z_A - z_{M1} \end{pmatrix} = \begin{pmatrix} 0{,}3m - 3m \\ 3m - 2m \\ 1{,}7m - 1m \end{pmatrix} = \begin{pmatrix} -2{,}7m \\ 1m \\ 0{,}7m \end{pmatrix}$$

$$\overrightarrow{a_2} = \overrightarrow{P_{M2}P_A} = \begin{pmatrix} x_A - x_{M2} \\ y_A - y_{M2} \\ z_A - z_{M2} \end{pmatrix} = \begin{pmatrix} 0{,}3m - 3{,}045m \\ 3m - 2m \\ 1{,}7m - 1m \end{pmatrix} = \begin{pmatrix} -2{,}745m \\ 1m \\ 0{,}7m \end{pmatrix}$$

$$\left|\overrightarrow{a_1}\right| = \sqrt{a_x^2 + a_y^2 + a_z^2} = \sqrt{-2{,}745^2 + 1^2 + 0{,}7^2}m = 2{,}963106m$$

$$\left|\overrightarrow{a_2}\right| = \sqrt{a_x^2 + a_y^2 + a_z^2} = \sqrt{-2{,}7^2 + 1^2 + 0{,}7^2}m = 3{,}004168m$$

$$T_{\text{Tot},1} = \frac{a_1}{v_{\text{Luft}}} = \frac{2{,}963106m}{343\frac{m}{s}} = 0{,}008638793s$$

$$T_{\text{Tot},2} = \frac{a_2}{v_{\text{Luft}}} = \frac{3{,}004168m}{343\frac{m}{s}} = 0{,}008758507s$$

$$T_0 = T_{\text{Tot},2} - T_{\text{Tot},1} = 0{,}008758507s - 0{,}008638793s = 0{,}119714ms$$

Aus Abbildung 9.1 werden die Zeiten des Maximums beider Mikrofone $t_1 = 2{,}7078ms$ und $t_2 = 2{,}8265ms$ entnommen, um daraus wie oben die Laufzeitverzögerung zwischen den Mikrofonen zu berechnen. Aus den beiden Ergebnissen wird der relative zeitliche Fehler berechnet.

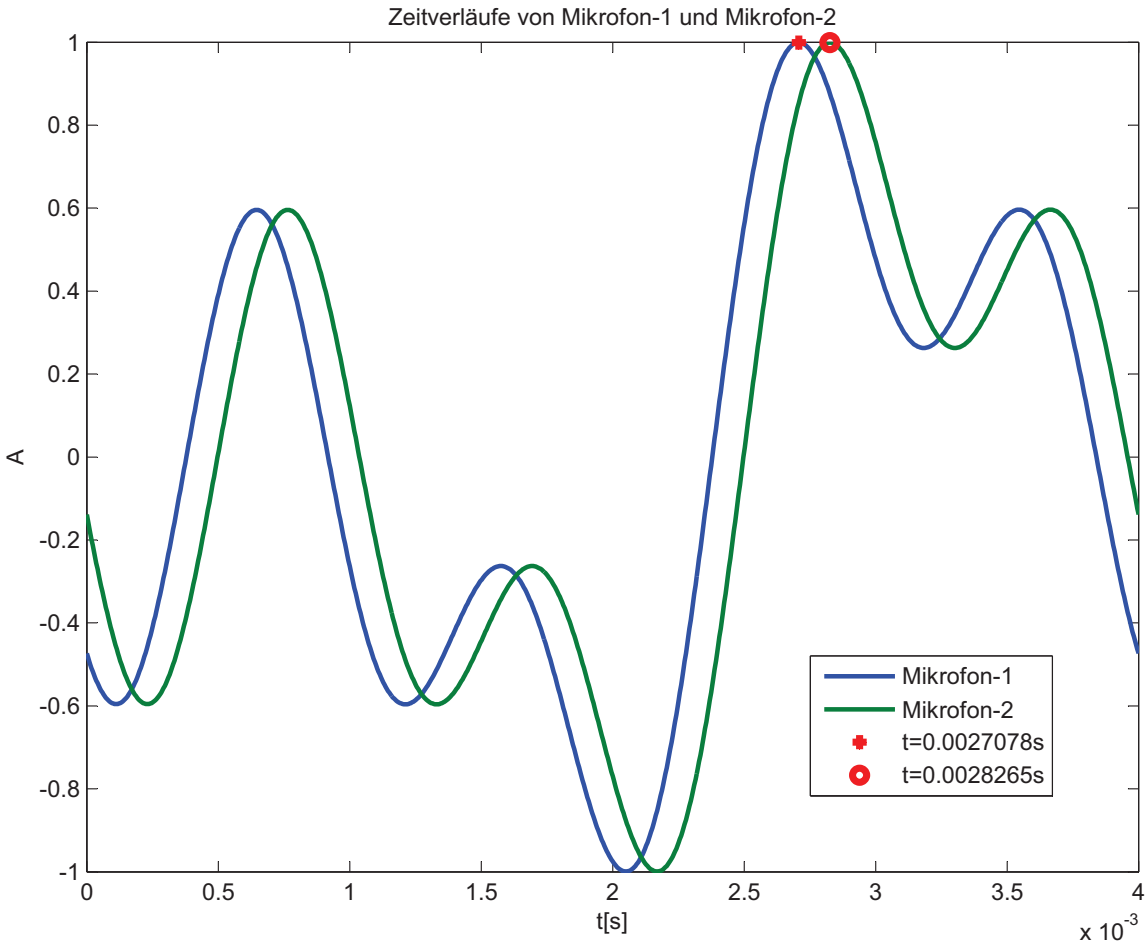


Abbildung 9.1.: Zeitverlauf von Mikrofon 1 und Mikrofon 2

$$t_0 = t_2 - t_1 = 2{,}8265\,ms - 2{,}7078\,ms = 0{,}1187\,ms$$
$$t_{\text{ERROR}} = \frac{T_0 - t_0}{T_0}\frac{0{,}119714\,ms - 0{,}1187\,ms}{0{,}119714\,ms} = 0{,}00847 = 0{,}847\%$$

Interessant ist der Winkelfehler, der sich bei der berechneten Zeitdifferenz ergibt. Die Positionen der Mikrofone und der Audioquelle wurden so gewählt, dass sich ungefähr ein Fehler von 1° ergibt. Es wurde sich bei dem Test dem Winkelfehler von 1° iterativ genähert.

$$\theta_{goe} = \arcsin\left\{\frac{T_\Delta}{t_{max}}\right\} = \arcsin\left\{\frac{0{,}119714\,ms}{0{,}131195\,ms}\right\} = 65{,}85^\circ$$
$$\theta_{sim} = \arcsin\left\{\frac{t_\Delta}{t_{max}}\right\} = \arcsin\left\{\frac{0{,}1187\,ms}{0{,}131195\,ms}\right\} = 64{,}79^\circ$$
$$\theta_{ERROR} = \theta_{goe} - \theta_{sim} = 65{,}85^\circ - 64{,}79^\circ = 1{,}06^\circ$$

Eingestellt wurde bei der Parametrierung eine Winkelauflösung von 1° garantiert über 40°. Wie der Berechnung entnommen werden kann, ist dies sogar bis zu einem Öffnungswinkel von 65° der Fall. Dies ist genauer als nach Einstellung verlangt worden ist. Die höhere Genauigkeit lässt sich dadurch erklären, dass sich nur im ungünstigsten Fall eine Winkelauflösung von 1° über einen Öffnungswinkel von 40° ergibt.

9.2. Aufnahme der Impulsantwort eines simulierten Raumes

Die Verteilung, der an einem Hörerort eintreffenden Reflexionen, kann man mittels der Raumimpulsantwort darstellen. Das ist die zeitliche Folge von Schallrückwürfen nach Anregen eines Raumes mit einem kurzen Schallimpuls.[19]

In Abbildung 9.2 ist die Raumimpulsantwort mit Reflexionen bis zur I.-Ordnung aufgenommen worden. Da bei dieser Aufnahme die Wände mit Vollreflexion betrachtet worden sind, fallen die Impulshöhen mit der Zeit ab.

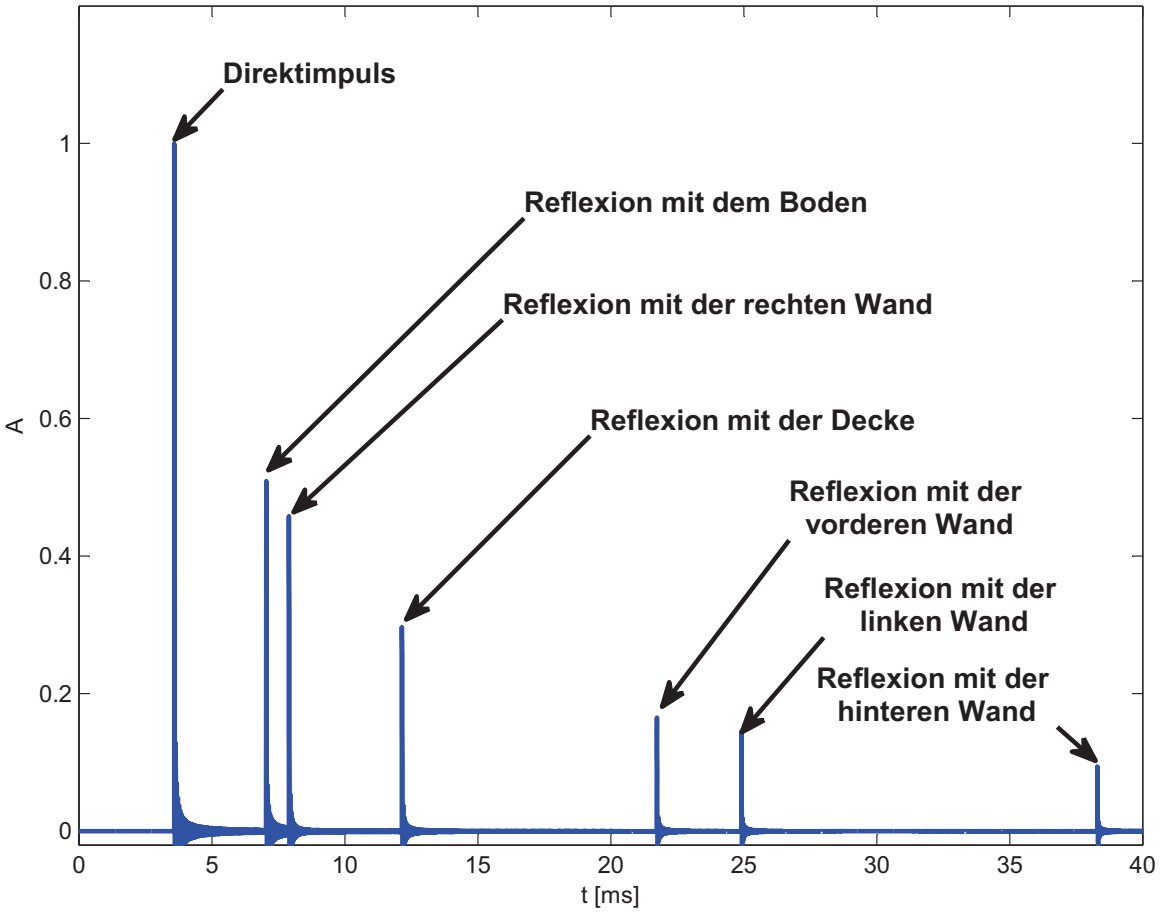


Abbildung 9.2.: Raumimpulsantwort mit Reflexionen bis zur I.-Ordnung

Dem Simulationsprogramm wurde als Signalquelle ein Dirac-Impuls zugeführt. Das Simulationsprogramm interpoliert die Eingangsdaten auf eine höhere Simulationsfrequenz. Das

Interpolieren geschieht mit der MATLAB Funktion *interpft()*. Diese Funktion erzeugt über eine *FFT* aus den Eingangsdaten die Werte einer *DFT*. Das transformierte Signal wird mittels *Zero-padding* auf die gewünschte Simulationsfrequenz erweitert. Danach wird das transformierte Signal mit einer *IFFT* zurück in den Zeitbereich transformiert. Dadurch entstehen *Overshoots* und *ripples*, welche in Abbildung 9.3 zu erkennen sind und durch das Gibbssche Phänomen verursacht werden.

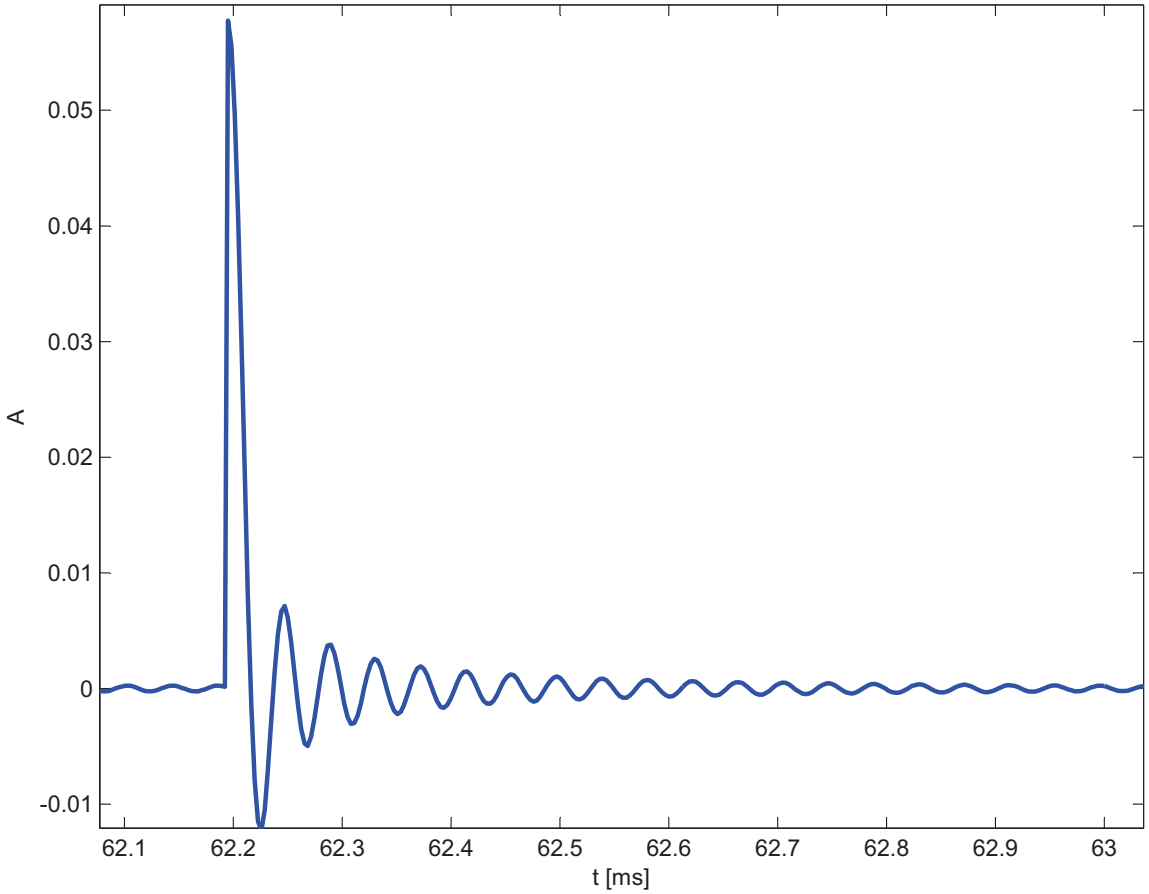


Abbildung 9.3.: Gibbssche Phänomen durch Interpolieren des Dirac-Impulses

Bei der Betrachtung einer Raumimpulsantwort mit Reflexionen bis zur II.-Ordnung fallen die Impulshöhen nicht mehr mit der Zeit ab, wie in Abbildung 9.4 dargestellt wird. Dies liegt daran, dass sich bei der Reflexion II.-Ordnung alle Reflexionen mit gegenüberliegenden Wänden genau überlagern.

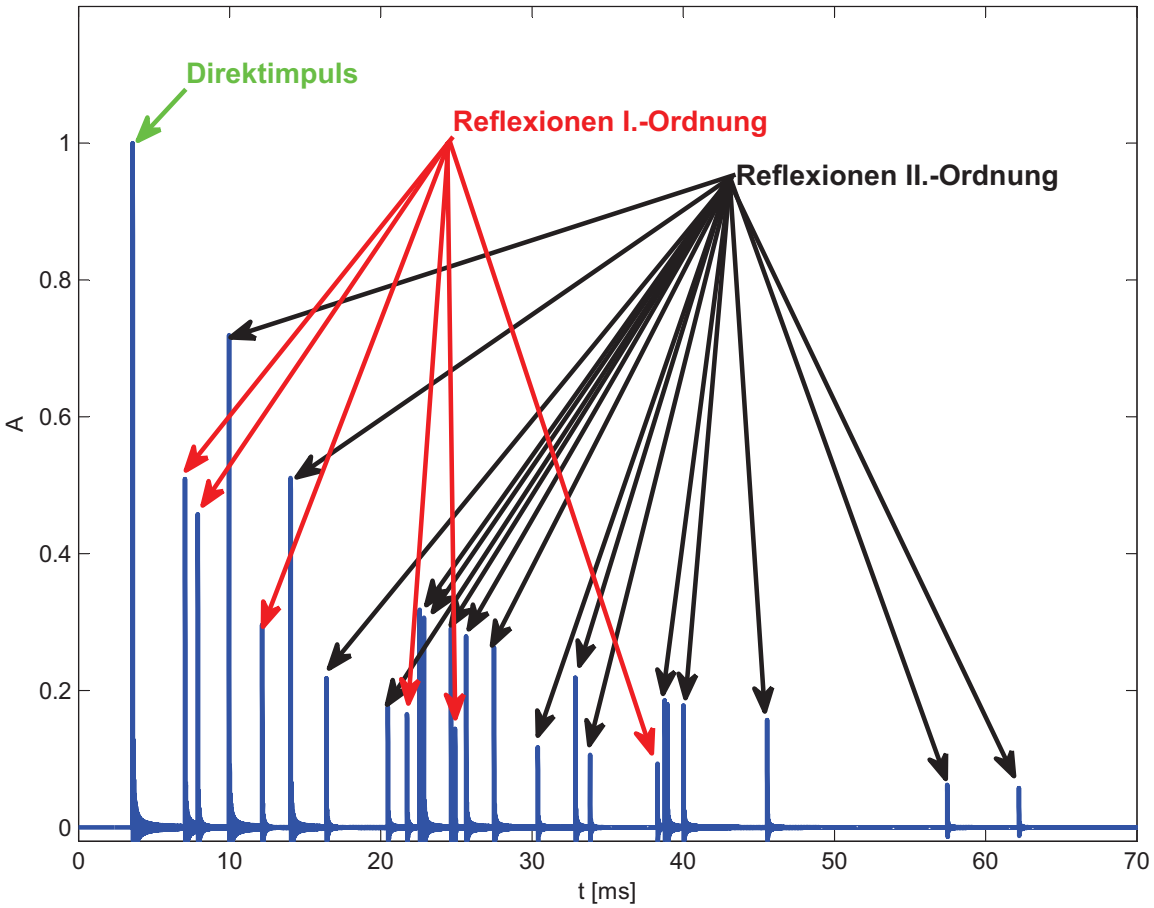


Abbildung 9.4.: Raumimpulsantwort mit Reflexionen bis zur II.-Ordnung

Werden, wie in Abbildung 9.5 dargestellt, die frequenzabhängigen Oberflächen mit in die Simulation integriert, werden die reflektierten Impulse noch stärker gedämpft. Die implementierten Filter dämpfen Frequenzen oberhalb von $8kHz$ mit $40dB$. Da ein Dirac-Impuls einen Frequenzgang von 1 hat und mit einer Simulationsfrequenz von $328kHz$ simuliert wurde, kommt fasst keine Leistung mehr bei den Reflexionen durch die Filter.

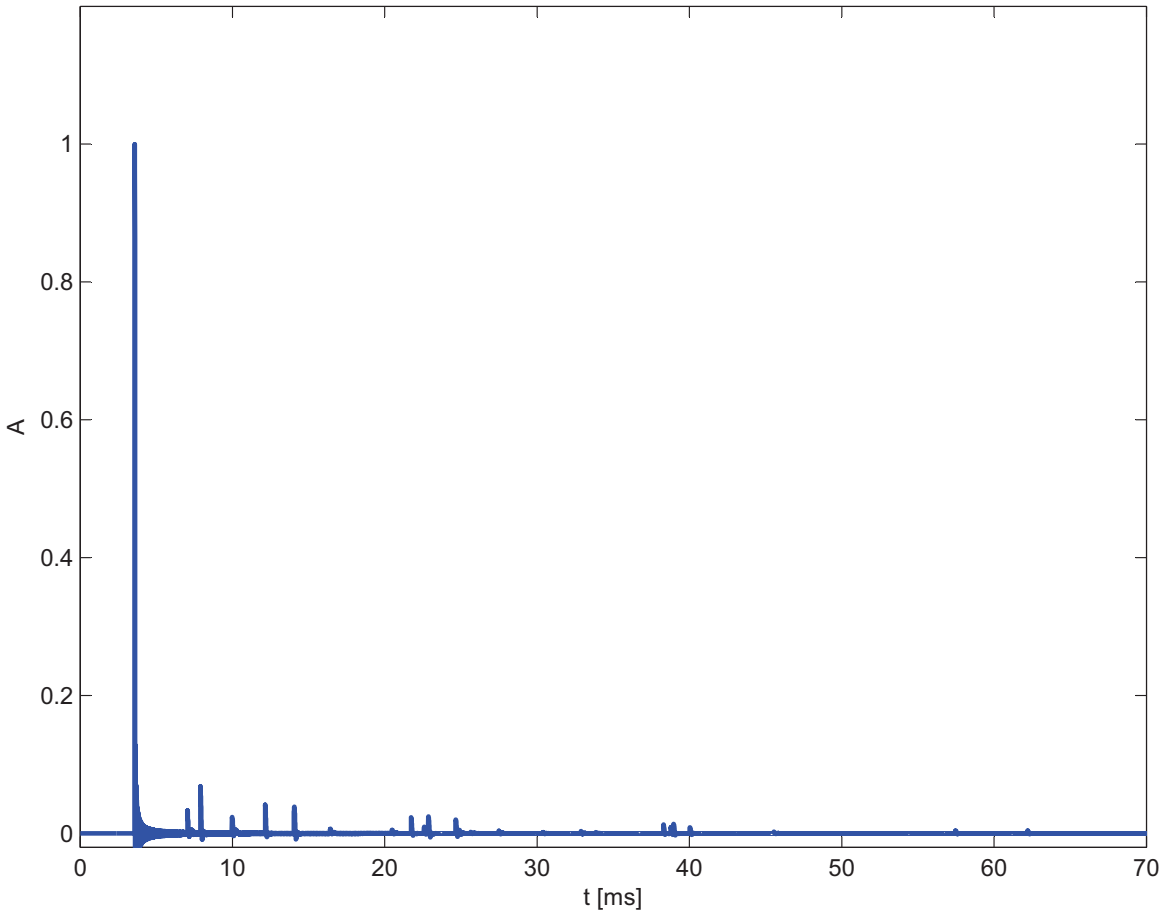


Abbildung 9.5.: Raumimpulsantwort mit Reflexionen bis zur II.-Ordnung mit frequenzabhängigen Oberflächen

Das Simulationsprogramm orientiert sich an einem Hardwareaufbau. Wenn mit einem Mikrofon über einen *ADU* eine Raumimpulsantwort aufgenommen wird, wird das aufgenommene Signal, bevor es auf den ADU geleitet wird, über einen Tiefpass gefiltert. Dem Simulationsprogramm dürfen auch nur Tiefpassgefilterte Signale zugeführt werden, damit das Simulationsverhalten möglichst der Realität entspricht. Wird ein *Dirac-Impuls* mit einem Tiefpass gefiltert, ergibt sich näherungsweise ein Rechteckimpuls. Aus diesem Grunde wurde ein Rechteckimpuls als Signalquelle für die Aufnahme der Impulsantwort bei frequenzabhängugen Oberflächen verwendet. Im Folgenden wird dies als bandbegrenzte Raumimpulsantwort definiert.

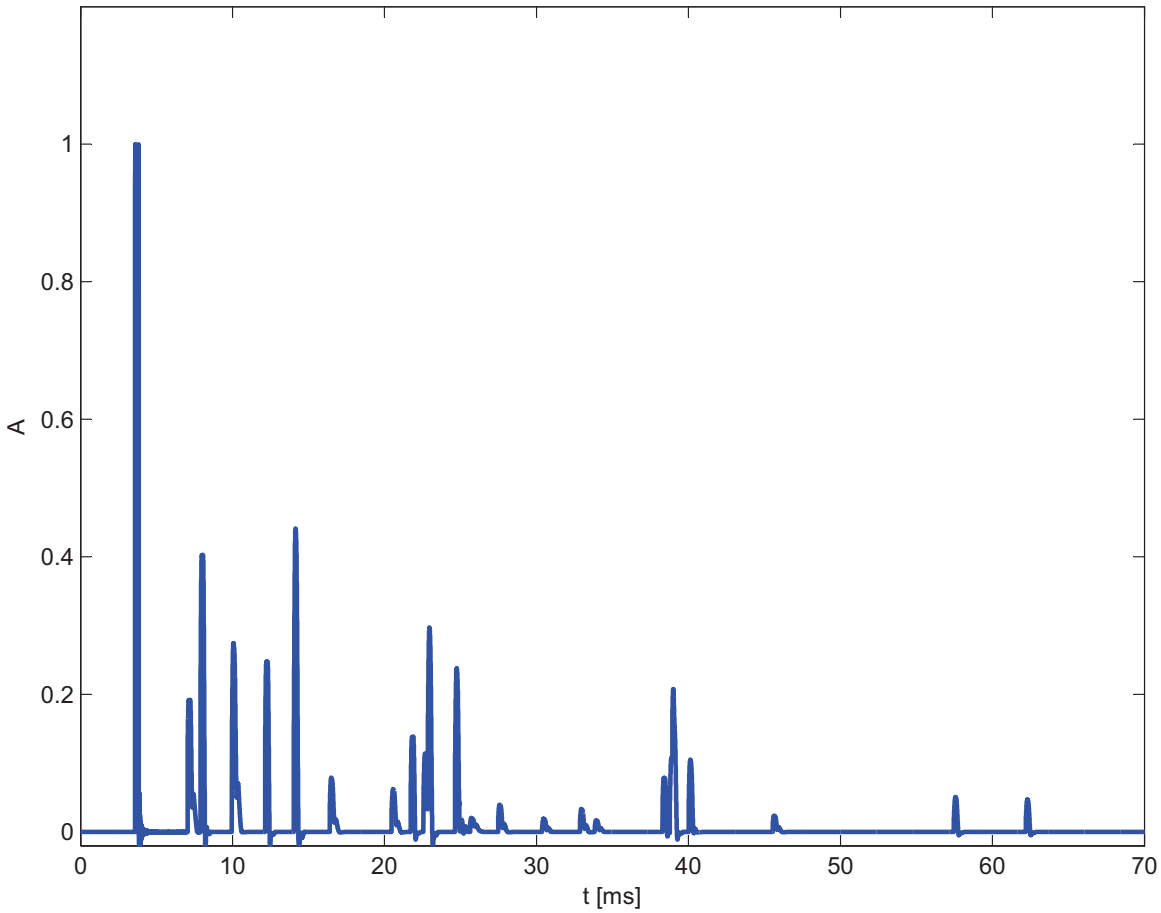


Abbildung 9.6.: Bandbegrenzte Raumimpulsantwort mit Reflexionen bis zur II.-Ordnung mit frequenzabhängigen Oberflächen

Die durch die Filterung geringere Flankensteilheit wurde in Abbildung 9.7 noch mal höher aufgelöst dargestellt.

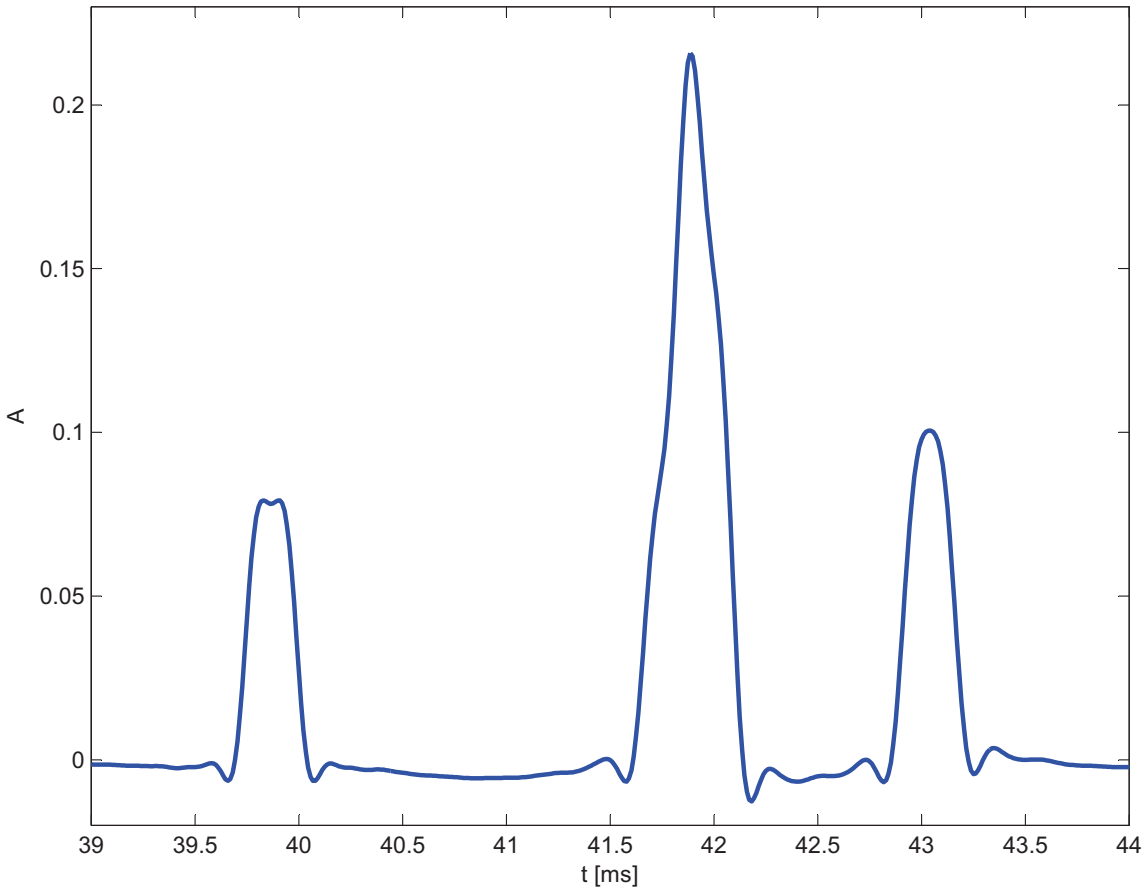


Abbildung 9.7.: Dämpfung des Impulses durch die frequenzabhängigen Oberflächen

9.3. Schallquellen-Lokalisation mittels MUSIC-Algorithmus

Das Simulationsprogramm soll auch für unterraumbasierte Verfahren verwendbar sein. Dies wurde durch eine Simulationsfrequenz, die sich dem Abstand des Mikrofonarrays anpasst, gewährleistet. Weiterhin soll getestet werden, ob ein solches unterraumbasiertes Verfahren zur Schallquellen-Lokalisation mit den Ausgangssignalen der Mikrofone arbeiten kann. Hierzu wurde der *MUSIC-Algorithmus* verwendet. Mit dem Simulationsprogramm wurden drei Mikrofonausgangssignale erzeugt, die dem MUSIC-Algorithmus zugeführt wurden. Aus der Simulationsanordnung wurde vorweg der Winkel der Audioquelle zum Mikrofonarray berechnet, um die Ergebnisse des MUSIC-Algorithmus bewerten zu können. Beim Simulationsprogramm wurde eine Simulationsfrequenz gewählt, die eine Winkelauflösung von 2° gewährleistet.

P_{M1} Mikrofon 1 mit $x_{M1} = 0{,}745\,m$, $y_{M1} = 6{,}145\,m$, $z_{M1} = 0{,}945\,m$

P_{M2} Mikrofon 2 mit $x_{M2} = 0{,}805\,m$, $y_{M2} = 6{,}145\,m$, $z_{M2} = 0{,}945\,m$

P_A Audioquelle mit $x_A = 1{,}452\,m$, $y_A = 6{,}958\,m$, $z_A = 1{,}54\,m$

Mit Formel 4.4, Formel 4.5 und Formel 5.3 wurde die Laufzeitverzögerung zwischen den Mikrofonen berechnet.

$$\vec{a_1} = \overrightarrow{P_{M1}P_A} = \begin{pmatrix} x_A - x_{M1} \\ y_A - y_{M1} \\ z_A - z_{M1} \end{pmatrix} = \begin{pmatrix} 1{,}452\,m - 0{,}745\,m \\ 6{,}958\,m - 6{,}145\,m \\ 1{,}54\,m - 0{,}945\,m \end{pmatrix} = \begin{pmatrix} 0{,}707\,m \\ 0{,}813\,m \\ 0{,}595\,m \end{pmatrix}$$

$$\vec{a_2} = \overrightarrow{P_{M2}P_A} = \begin{pmatrix} x_A - x_{M2} \\ y_A - y_{M2} \\ z_A - z_{M2} \end{pmatrix} = \begin{pmatrix} 1{,}452\,m - 0{,}805\,m \\ 6{,}958\,m - 6{,}145\,m \\ 1{,}54\,m - 0{,}945\,m \end{pmatrix} = \begin{pmatrix} 0{,}647\,m \\ 0{,}813\,m \\ 0{,}595\,m \end{pmatrix}$$

$$\left|\vec{a_1}\right| = \sqrt{a_x^2 + a_y^2 + a_z^2} = \sqrt{0{,}707^2 + 0{,}813^2 + 0{,}595^2}\,m = 1{,}23079m$$

$$\left|\vec{a_2}\right| = \sqrt{a_x^2 + a_y^2 + a_z^2} = \sqrt{0{,}647^2 + 0{,}813^2 + 0{,}595^2}\,m = 1{,}19733m$$

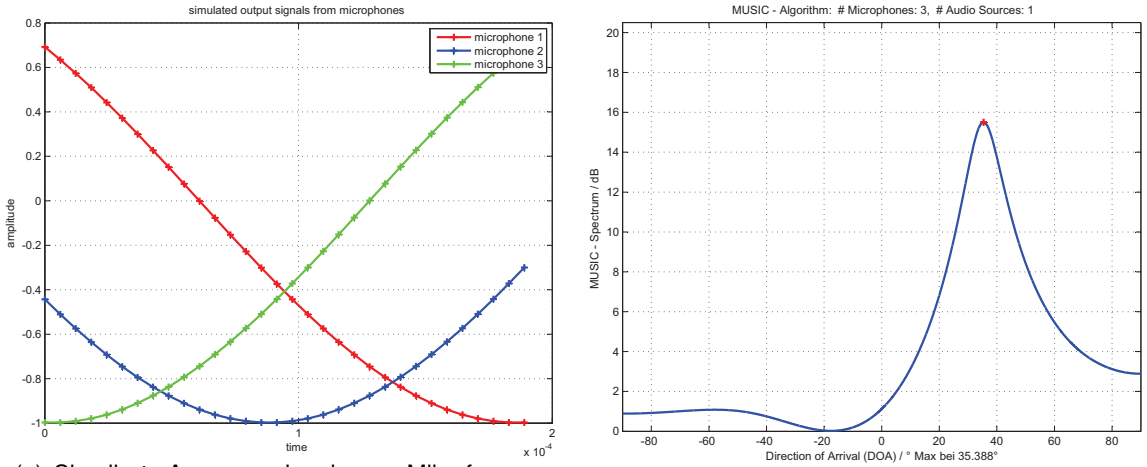
$$T_{\text{Tot},1} = \frac{a_1}{v_{\text{Luft}}} = \frac{1{,}23079\,m}{343\,\frac{m}{s}} = 0{,}0035883s$$

$$T_{\text{Tot},2} = \frac{a_2}{v_{\text{Luft}}} = \frac{1{,}19733\,m}{343\,\frac{m}{s}} = 0{,}0034908s$$

$$T_0 = T_{\text{Tot},1} - T_{\text{Tot},2} = 0{,}0035883s - 0{,}0034908s = 0{,}0975\,ms$$

$$\theta_{\text{goe}} = \arcsin\left\{\frac{T_\Delta}{t_{\text{max}}}\right\} = \arcsin\left\{\frac{0{,}0975\,ms}{0{,}1749\,ms}\right\} = 33{,}88^\circ$$

Das Ergebnis der Schallquellen-Lokalisation vom MUSIC-Algorithmus bei Simulationsausgangswerten ohne Reflexionen ergab $35{,}388^\circ$. Dies entspricht einen Fehler von $1{,}508^\circ$, wie in Abbildung 9.8(b) dargestellt. Damit wird die Schallquelle im durch die Simulation ergebenen Toleranzbereich lokalisiert.

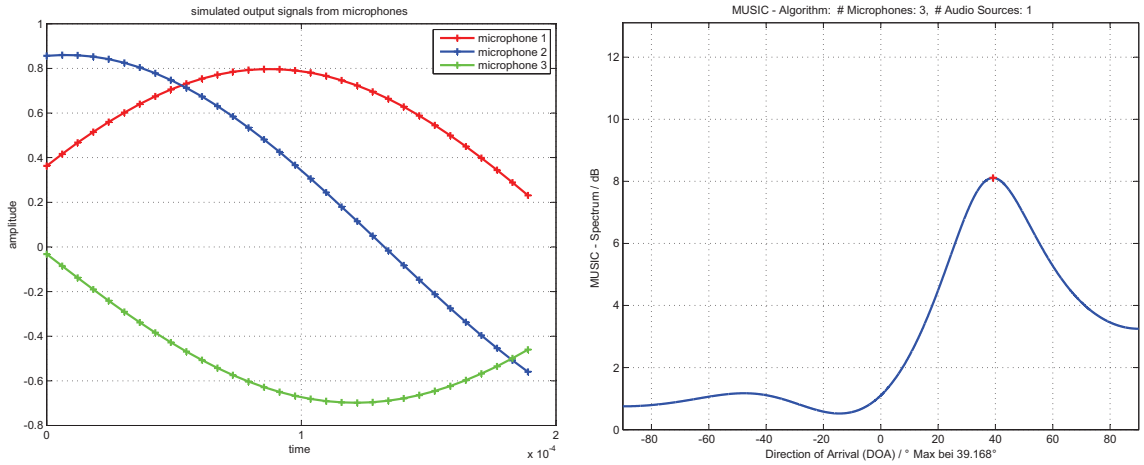


(a) Simulierte Ausgangssignale vom Mikrofonarray

(b) Ergebnis der Schallquellen-Lokalisation

Abbildung 9.8.: Schallquellen-Lokalisation mittels MUSIC-Algorithmus an simulierten Mikrofonausgangssignalen ohne Reflexionen

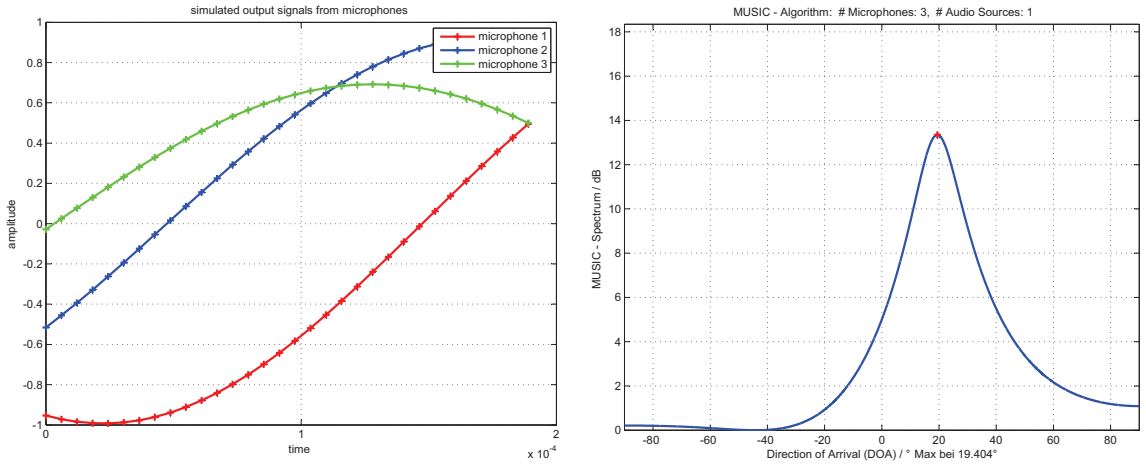
Bei Simulationsausgangswerten mit Reflexionen bis zur I.-Ordnung verschiebt sich der ermittelte Wert auf $39{,}168$. Dies enspricht einem Fehler von $5{,}288°$. Die Reflexionen beeinflussen das Ergebnis. Die Kurve der Schallquellen-Lokalisation ohne Reflexion ist, wie in Abbildung 9.8(b) dargestellt, gegenüber der Schallquellen-Lokalisation mit Reflexionen, wie in Abbildung 9.9(b), deutlich schlanker und höher. Dies kommt durch die Beeinflussung der Reflexionen, die aus einem anderen Winkel auf das Mikrofonarray einfallen.



(a) Simulierte Ausgangssignale vom Mikrofonarray (b) Ergebnis der Schallquellen-Lokalisation

Abbildung 9.9.: Schallquellen-Lokalisation mittels MUSIC-Algorithmus an simulierten Mikrofonausgangssignalen mit Reflexionen bis zur I.-Ordnung

In Abbildung 9.10(b) ist das Ergebnis der Schallquellen-Lokalisation mit Reflexionen bis zur II.-Ordnung dargestellt. Der ermittelte Wert liegt bei $19{,}404°$. Dies ist ein sehr große Fehler von $-14{,}476°$. Die Reflexionen überlagern sehr stark. Somit hat der direkte Schall nicht die höchste Amplitude.

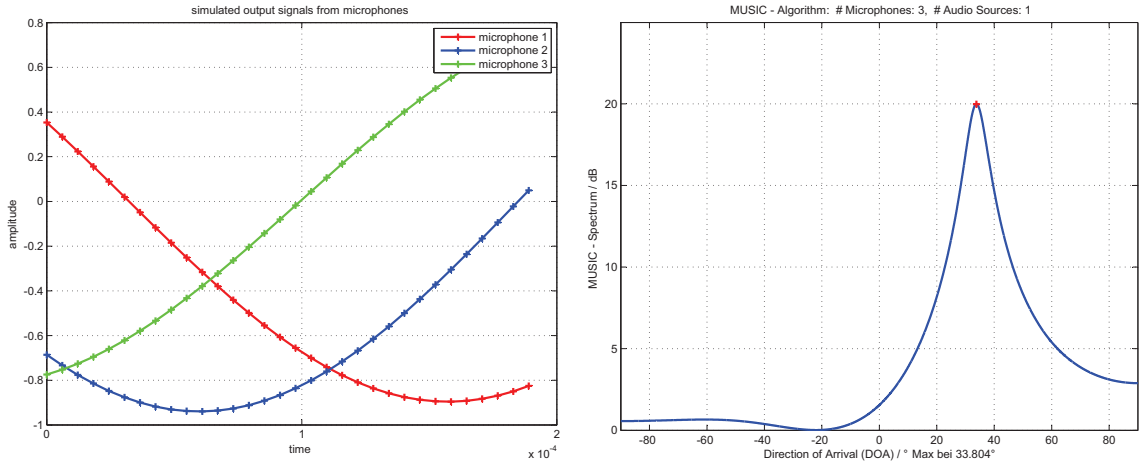


(a) Simulierte Ausgangssignale vom Mikrofonarray

(b) Ergebnis der Schallquellen-Lokalisation

Abbildung 9.10.: Schallquellen-Lokalisation mittels MUSIC-Algorithmus an simulierten Mikrofonausgangssignalen mit Reflexionen bis zur II.-Ordnung

In Abbildung 9.11(b) beeinflussen die Reflexionen die Schallquellen-Lokalisation nur noch gering. Es wurde bei dieser Simulation stark dämpfende Oberflächen ausgewählt. Hierdurch konnte der MUSIC-Algorithmus die Schallquelle sehr gut lokalisieren mit $33{,}804$. Dies entspricht einen Fehler von $-0{,}076°$. Dass der Fehler geringer ist als bei der Schallquellen-Lokalisation ohne Reflexionen, ist möglicherweise einer zufälligen positiven Beeinflussung der schwachen Reflexionen zuzuschreiben.



(a) Simulierte Ausgangssignale vom Mikrofonarray

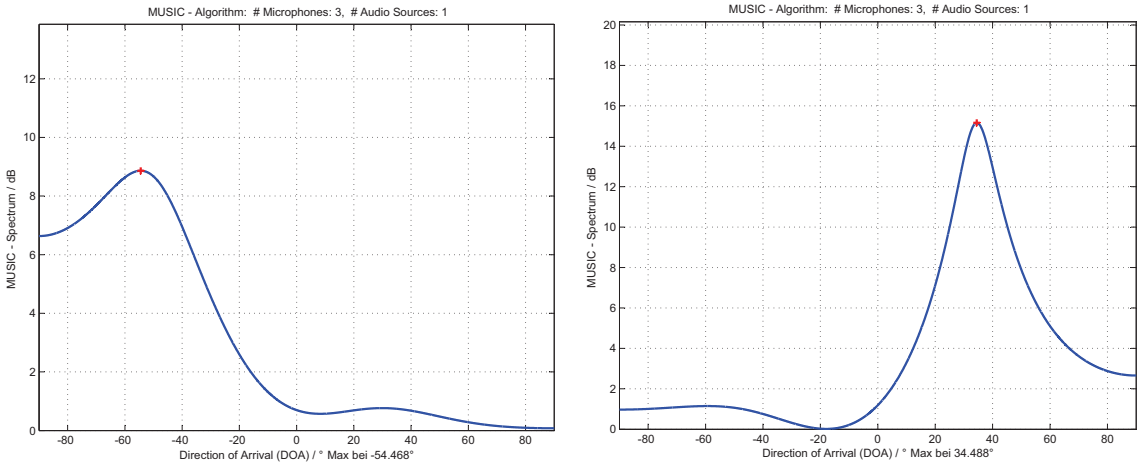
(b) Ergebnis der Schallquellen-Lokalisation

Abbildung 9.11.: Schallquellen-Lokalisation mittels MUSIC-Algorithmus an simulierten Mikrofonausgangssignalen mit Reflexionen bis zur II.-Ordnung und frequenzabhängigen Oberflächen

Um die Möglichkeiten des Programmes weiter darzulegen, werden im Folgenden Ergebnisse bei unterschiedlichen Raumanordnugen dargestellt. Dabei wurden nur die Raumdimensionen geändert und die Ausrichtung der Audioquelle zum Mirkrofonarray beibehalten. Somit steht die Audioquelle weiterhin in einem Winkel von $33{,}88°$ zum Mikrofonarray.

9.3.1. Vergleich zwischen vollreflektierenden und dämpfenden Oberflächen

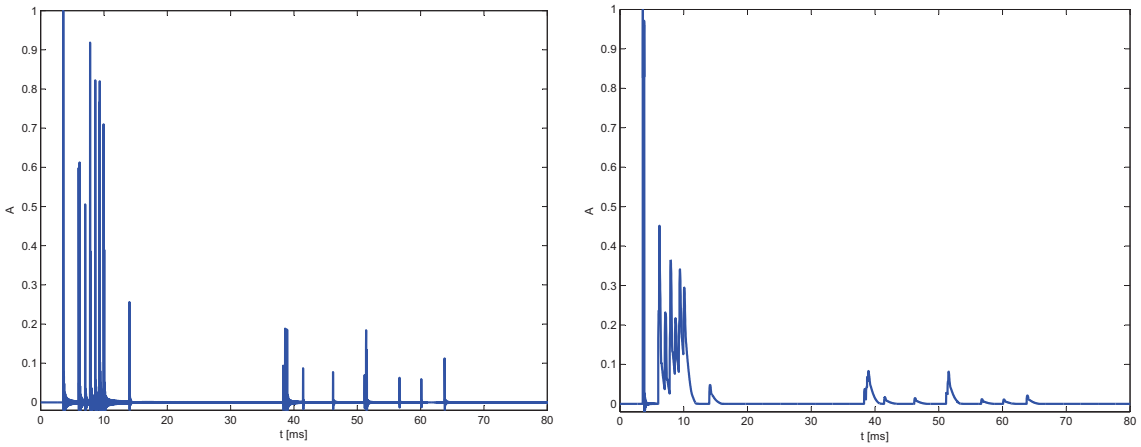
Zunächst wurde eine sehr ungünstige Raumanordnung gewählt, bei der sehr viele Reflexionen nahe beieinander und mit geringer Dämpfung durch Ausbreitung auf das Mikrofonarray vorlagen. Das Ergebnis der Schallquellen-Lokalisation, wie in Abbildung 9.12(a) dargestellt, ergibt einen Winkelfehler von über $80°$. Die Anordnung ist so ungünstig, dass der MUSIC-Algorithmus nicht in der Lage ist, die Schallquelle zu Orten. Bei gleicher Anordnung wurden Oberflächenparameter mit stark dämpfenden Eigenschaften eingestellt. In Abbildung 9.12(b) kann man eine Fehlberechnung des MUSIC-Algorithmus von nur $0,6°$ ablesen. Die Zuverfässigkeit des MUSIC-Algorithmus ist somit von der Oberflächenstruktur der Wände abhängig. In diesem Beispiel wurden nur die beiden Extremsituationen dargestellt. Es ist natürlich auch möglich Wandparameter mit einer mittleren Dämpfung einzustellen.



(a) Vollreflektierende Oberflächen (b) Dämpfende Oberflächen

Abbildung 9.12.: Ergebnisvergleich der Schallquellen-Lokalisation bei unterschiedlich reflektierenden Oberflächen

In Abbildung 9.13 sind die Raumimpulsantworten der beiden Messungen dargestellt, an denen man sehr deutlich die Problematik der dicht beieinander auftretenden Reflexionen anschauen kann. Durch die stark dämpfenden Oberflächen prägen sich die Reflexionen in Abbildung 9.13(b) geringer aus.



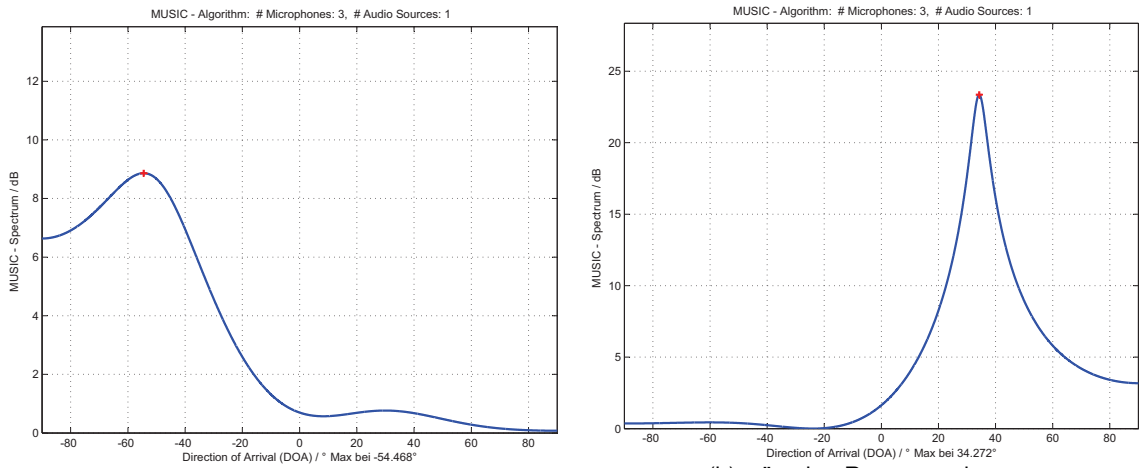
(a) Vollreflektiernden Oberflächen (b) Dämpfenden Oberflächen

Abbildung 9.13.: Raumimpulsantwortvergleich bei unterschiedlich reflektierenden Oberflächen

Es können, wie oben dargestellt, Schallquellen-Lokalisations-Algorithmen in Abhängigkeit der Wandparameter getestet werden.

9.3.2. Vergleich zwischen günstigen und ungünstigen Raumanordungen

Zum Schluss wird hier der Vergleich zwischen einer günstigen Raumanordnung und einer ungünstigen Raumanordnung dargelegt. In Abbildung 9.14(a) wurde die Audioquelle und das Mikrofonarray in eine Ecke eines Raumes platziert. Dabei fallen viele Reflexionen nahe beieinander auf das Mikrofonarray ein. Der MUSIC-Algorithmus kann die Schallquelle nicht Orten. In Abbildung 9.14(b) wurde die Audioquelle und das Mikrofonarray in die Mitte eines Raumes platziert. Somit sind die Reflexionen weit auseinander und durch die langen Wegstrecken bei Ankunft am Mikrofonarray schon stark gedämpft. Die Schallquelle kann vom MUSIC-Algorithmus gut lokalisiert werden. Die Zuverlässigkeit des MUSIC-Algorithmus ist von der Raumanordnung abhängig. Auch hier können verschiedene Situationen, die sich verschieden stark auf die Schallquellen-Lokalisation auswirken, getestet werden.

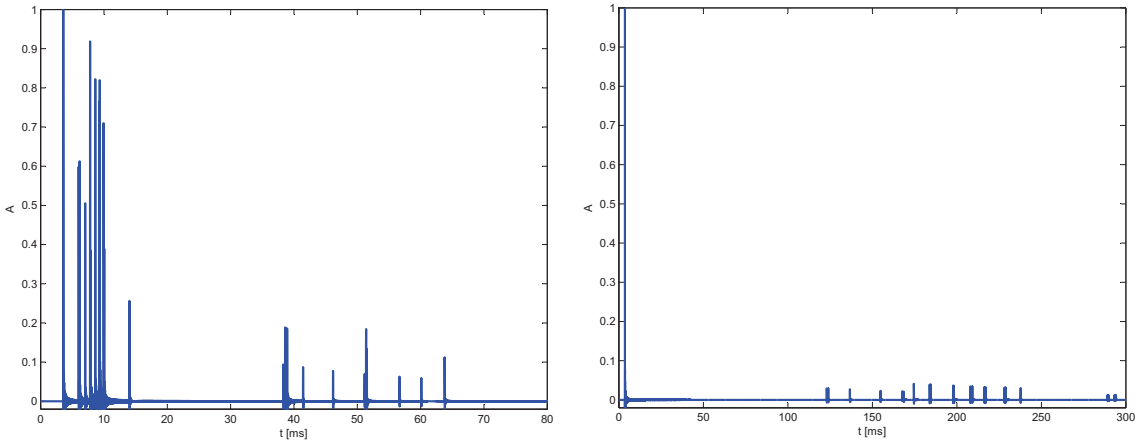


(a) ungünstige Raumanordnung

(b) günstige Raumanordnung

Abbildung 9.14.: Ergebnisvergleich der Schallquellen-Lokalisation bei unterschiedlichen Raumanordnungen

In Abbildung 9.15 sind die Raumimpulsantworten der beiden Messungen dargestellt. In Abbildung 9.15(b) ist zu erkennen, dass die Reflexionen deutlich später und stark gedämpft auf das Mikrofonarray einfallen, als dies in Abbildung 9.15(a) der Fall ist.



(a) ungünstige Raumanordnung (b) günstige Raumanordnung

Abbildung 9.15.: Raumimpulsantwortvergleich bei unterschiedlichen Raumanordnungen

Es können wie oben dargestellt Schallquellen-Lokalisations-Algorithmen in Abhängigkeit der Raumanordnung getestet werden.

10. Ausblick

Das Raumsimulationsprogramm ist beliebig erweiterbar. Nachstehend werden ein paar Beispiele aufgeführt, die als *Add on* zum Simulationsprogramm implementiert werden können.

- bewegte Quellen
- Anzahl der Wandreflexionen
- Auslagern von rechenintensiven MATLAB-code in C-code zur Beschleunigung des Simulationsteiles
- Richtcharakteristik von Schallquellen hinzufügen
- Erweiterung der Wandreflexiondatenbank

Eine akustische Raumsimulation mit MATLAB, speziell für unterraumbasierte Verfahren wurde bis vor dieser Bachelorarbeit nicht öffentlich zugänglich gemacht.[1] Dies war bislang besonders ärgerlich im Bezug auf die M-Files von MATLAB, da bis dato auf keine Vorarbeit zurückgegriffen werden konnte. Mit dieser Bachelorthesis und den damit erstellten M-Files ist eine erste Open Source Basis für eine Weiterentwicklung solcher Raumsimulationen erstellt worden. Ich hoffe an dieser Stelle, dass meine Arbeit aufgegriffen wird und dieses Programm erweitert wird.

Für das Gesamtprojekt, die vollautomatische interaktive Videoaufzeichnung von Vorlesungen, ist ein erster Teile fertig gestellt. Parallel zu dieser Arbeit ist an der HAW Hamburg eine Thesis über das Beschleunigen von FFTs mit dem *DSK TMS320C6713* von *Texas Instruments* entstanden und eine Thesis über Graphikkarten, die zur Verwendung als Signalprozessor herangezogen werden sollen. Beides sind Hardwareumgebungen auf dem die vollautomatische Videoaufzeichnung realisiert werden könnte. In einer folgenden Masterthesis kann die Findung eines optimal geeigneten Algorithmus zur Sprachquellenlokalisation und/oder die Implementierung eines Algorithmus in einer der beiden Hardwareumgebungen realisiert werden.

11. Schluss

Die akustische Raumsimulation wurde erfolgreich unter MATLAB implementiert. Für eine bedienfreundliche Handhabung wurde eine graphische Oberfläche erstellt. Damit das Programm nachvollziehbar ist und damit erweiterbar, wurden über diese Arbeit hinaus reichende Gedanken zur Strukturierung des Programms gemacht. Sowie eine Struktur für die Parameterdatensätze entwickelt. Über die physikalischen Eigenschaften der Schallausbreitung ist ein Konzept zur Implementierung der Simulationsparameter entstanden. Ganz besonders wurde die Genauigkeit der Simulationssoftware untersucht, wodurch eine Parametrierung der Simulationsfrequenz entstanden ist. Für die Simulation wurde das Konzept geändert, nachdem festgestellt wurde, dass die Simulation unter MATLAB-SIMULINK mindestens 10 mal langsamer läuft, als in einem MATLAB-File. Außerdem wurde darauf geachtet, dass alle Funktionen, die direkt mit der Simulationberechnung zu tun haben, als externe MATLAB-File gehandhabt wurden. Somit ist es nachträglich möglich die Simulation zu beschleunigen, indem man die MATLAB-Files für die Simulation optimiert, ohne die graphische Oberfläche verändern zu müssen. Das gleiche gilt für die Parameterberechnung. Sollte es für den Anwender doch interessant sein, wie sich sein Algorithmus bei Einbeziehen der Reflexionen III.- und IV.-Ordnung verhält, kann er ebenfalls die entsprechende M-File ändern.

Die akustische Raumsimulation ist soweit fertig gestellt, dass ein Testen von Schallquellen-Lokalisations-Algorithmen auch mit unterraumbasierten Verfahren möglich ist. Es kann sowohl die Abhängigkeit der Wandparameter als auch die Abhängigkei der Raumanordnung auf die Lokalisationzuverlässigkeit getestet werden. Somit beschleunigt die akustische Raumsimulation die Verbesserung der Algorithmen.

Glossar

ADU Ein **A**nalog-**D**igital-**U**msetzer setzt analoge Eingangssignale in digitale Daten bzw. Datenstrom um, der dann weiterverarbeitet oder gespeichert werden kann.

Callback Function Rückruffunktion (engl. Callback function) ist einer anderen Funktion eine übergebene Parameterfunktion. Unter GUIDE werden diese Funktionen immer dann aufgerufen, wenn ein Ereignis stattgefunden hat. (z.B.: Drücken eines Buttons)

Cellarray ist ein Array aus Daten vom Typ ”Cell” und wird unter MATLAB zum Ansprechen von Strukturen verwendet.

Create Function Ist eine Funktion, die beim Erstellen eines graphischen Objekts ausgeführt wird. Unter GUIDE werden diese Funktionen beim Erstellen eines graphischen Objekts ausgeführt. (z.B.: Definieren der Hintergrundfarbe vom Button)

DFT Die **D**iskrete **F**ourier-**T**ransformation ist die Fourier-Transformation eines zeitdiskreten periodischen Signals.

Dirac-Impuls benannt nach Paul Adrien Maurice Dirac, ist ein unendlich kurzer und unendlich hoher Impuls mit der Gewichtung 1. In dieser Arbeit der am kürzesten mögliche Impuls. (Bei $48kHz$ Abtastfrequenz ein Impuls der Dauer von $20{,}3\mu S$)

DSK TMS320C6713 Entwicklungsboard für den *TMS320C6713 DPS* (engl. DSP Starter Kit, DSK)

DSP Ein Digitaler Signalprozessor (engl. **d**igital **s**ignal **p**rocessor) dient der kontinuierlichen Bearbeitung von digitalen Signalen (z. B. Audio- oder Videosignale) durch die digitale Signalverarbeitung. Zur Verarbeitung von analogen Signalen wird der DSP in Verbindung mit Analog-Digital-Umsetzern und Digital-Analog-Umsetzern eingesetzt.[15]

FFT Schnelle Fourier-Transformation (engl. **f**ast **F**ourier **t**ransform) ist ein Algorithmus zur schnellen Berechnung der Werte einer diskreten Fourier-Transformation.

Gibbssche Phänomen ist die Bezeichnung des typischen Verhaltens von Fourierreihen in der Umgebung von Sprungstellen.[8]

GUIDE Entwicklungsumgebung für eine GUI unter MATLAB (engl. **g**raphical **u**ser **i**nterface **d**evelopment **e**nvironment).

IFFT Die Umkehrfunktion zur schnellen Fourier-Transformation(FFT). (engl. **I**nverse **F**ast **F**ourier **T**ransformation)

M-Files ist ein Dateityp zur Speicherung von Funktionen und Skripten unter MATLAB.

MATLAB ist eine kommerzielle, plattformunabhängige Software des Unternehmens The MathWorks, Inc. zur Lösung mathematischer Probleme und zur grafischen Darstellung der Ergebnisse. MATLAB ist primär für numerische Berechnungen mithilfe von Matrizen ausgelegt, woher sich auch der Name ableitet: **MAT**rix **LAB**oratory.[16]

Mikrofonarray Mehrere nebeneinander angeordnete Mikrofone.

NaN (engl. **N**ot **a** **N**umber) bedeutet ”keine Zahl”

Open Source Lizenzen für Software, deren Quelltext öffentlich zugänglich ist und durch die freie Lizenz Weiterentwicklungen fördert.

Overshoot Ein Überschwinger der nach einer sprunghaften Änderung einer Eingangsgröße nicht den gewünschten Ausgangswert direkt erreicht, sondern über den Sollwert hinausschießt.

ripple Ist die Überlagerung eines konstanten Signalanteils mit einen Wechselanteil mit beliebiger Frequenz und Kurvenform.

SIMULINK Software des Herstellers The MathWorks zur Modellierung von Systemen (technisch, physikalisch, finanzmathematisch, ...). Simulink ist ein Zusatzprodukt zu MATLAB und benötigt dieses zum Ausführen.[17]

Spiegelquellenverfahren ist ein Verfahren mit dessen Hilfe sich die Wegstrecke zwischen zwei Punkten im Raum gespiegelt über eine Wand ermitteln lässt.[1, Seite 13]

Standarddialog sind graphische Dialogoberflächen, die der Benachrichtigung, Abfrage oder Eingabe dienen.

Struktur auch Datenstruktur, ist ein Datenformat zur strukturierten Ablage von Daten.

Unterraum ist eine Teilmenge einer Menge, die mit einer mathematischen Struktur versehen ist, welche bezüglich der mathematischen Struktur im weitesten Sinne abgeschlossen ist. Eine genaue Definition hängt von der Struktur ab.[18]

unterraumbasierten Verfahren sind spezielle Verfahren zur Spektralschätzung.

Zero-padding Ist das Erweitern eines Signals bzw. eines transformierten Signals mit Nullen auf eine bestimmte Länge.

Literaturverzeichnis

[1] BERNDT, Gerson: *3D- Simulation für die Übertragung von Schallsignalen auf Mikrofon-Arrays*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2003

[2] ERBSLAND, Tobias ; NITSCH, Andreas: *Diplomarbeit mit LaTeX* . 2008. – URL http://drzoom.ch/project/dml/

[3] GRUPP, Frieder ; FLORIAN: *Simulink - Grundlagen und Beispiele*. Oldenbourg Wissenschaftsverlag, 2007. – ISBN 978-3-486-58091-4

[4] HOFFMANN, Josef: *MATLAB und SIMULINK - in Signalverarbeitung und Kommunikation*. Addison Wesley Longman Verlag, 1999. – ISBN 3-8273-1454-2

[5] MERZIGER, Gerhard ; MÜHLBACH, Günter ; WILLE, Detlef ; WIRTH, Thomas: *FORMELN + HILFEN zur HÖHEREN MATHEMATIK*. 2. Auflage. Binomi Verlag, 2007. – ISBN 978-3-923923-35-9

[6] MEY, Sven ; CAJINA, Rodolfo: *DSP-gesteuertes Mikrofonarray zur Sprecherlokalisierung mit Hilfe eines breitbandigen MUSIC-Algorithmus*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2006

[7] MÖSER, Michael: *Technische Akustik*. 8. aktualisierte Auflage. Springer-Verlag, 2009. – ISBN 978-3-540-89817-7

[8] PAPULA, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler*. Bd. 2. 12., überarbeitete und erweiterte Auflage. Vieweg+Teubner Verlag, 2009. – ISBN 978-3-8348-0564-5

[9] PAPULA, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler*. Bd. 1. 12., überarbeitete und erweiterte Auflage. Vieweg+Teubner Verlag, 2009. – ISBN 978-3-8348-0545-4

[10] SAXENA, Anshul K.: *Wideband Audio Source Localization using Microphone Array and MUSIC Algorithm*, Hochschule für Angewandte Wissenschaften Hamburg, Master Thesis, 2009

[11] SENGPIEL, Eberhard: *Absorptionsgrad α verschiedener Materialien und Oberflächen*. Januar 2010. – URL http://www.sengpielaudio.com/Rechner-RT60Koeff.htm

[12] STEIN, Ulrich: *Einsteigen in das Programmieren mit MATLAB*. 2., aktualisierte Auflage. Carl Hansen Verlag, 2009. – ISBN 978-3-446-41594-2

[13] TIPLER, Paul A. ; MOSCA, Gene: *Physik - Für Wissenschaftler und Ingeneure*. 2. deutsche Auflage. Elsevier GmbH, 2004. – ISBN 3-8274-1164-5

[14] WIKIPEDIA: *Absorptionsgrad*. Februar 2010. – URL http://de.wikipedia.org/wiki/Absorptionsfaktor

[15] WIKIPEDIA: *Digitaler Signalprozessor*. Februar 2010. – URL http://de.wikipedia.org/wiki/Digitaler_Signalprozessor

[16] WIKIPEDIA: *MATLAB*. Februar 2010. – URL http://de.wikipedia.org/wiki/Matlab

[17] WIKIPEDIA: *SIMULINK*. Februar 2010. – URL http://de.wikipedia.org/wiki/Simulink

[18] WIKIPEDIA: *Unterraum*. Februar 2010. – URL http://de.wikipedia.org/wiki/Unterraum

[19] WISSEN, BauNetz: *Raumimpulsantwort*. Februar 2010. – URL http://www.baunetzwissen.de/glossarbegriffe/Akustik_Raumimpulsantwort_44915.html?index=R

Tabellenverzeichnis

Abbildungsverzeichnis

A. Quellcode der GUI

Listing A.1: M-File für die GUI

```
1
2  %% Funktion zur Verarbeitung von Übergabeparameter
3  % Diese Funktion sollte nicht verändert werden!
4  function varargout = ersteOberflaeche(varargin)
5  % ERSTEOBERFLAECHE M-file for ersteOberflaeche.fig
6  %      ERSTEOBERFLAECHE, by itself, creates a new ERSTEOBERFLAECHE
       or raises the existing
7  %      singleton*.
8  %
9  %      H = ERSTEOBERFLAECHE returns the handle to a new
       ERSTEOBERFLAECHE or the handle to
10 %      the existing singleton*.
11 %
12 %      ERSTEOBERFLAECHE('CALLBACK',hObject,eventData,handles,...)
       calls the local
13 %      function named CALLBACK in ERSTEOBERFLAECHE.M with the
       given input arguments.
14 %
15 %      ERSTEOBERFLAECHE('Property','Value',...) creates a new
       ERSTEOBERFLAECHE or raises the
16 %      existing singleton*.  Starting from the left, property
       value pairs are
17 %      applied to the GUI before ersteOberflaeche_OpeningFcn gets
       called.  An
18 %      unrecognized property name or invalid value makes property
       application
19 %      stop.  All inputs are passed to ersteOberflaeche_OpeningFcn
       via varargin.
20 %
21 %      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
       only one
22 %      instance to run (singleton)".
23 %
24 % See also: GUIDE, GUIDATA, GUIHANDLES
```

```
25
26 % Edit the above text to modify the response to help
       ersteOberflaeche
27
28 % Last Modified by GUIDE v2.5 01-Feb-2010 13:07:35
29
30 % Begin initialization code - DO NOT EDIT
31 gui_Singleton = 1;
32 gui_State = struct('gui_Name',       mfilename, ...
33                    'gui_Singleton',  gui_Singleton, ...
34                    'gui_OpeningFcn', @ersteOberflaeche_OpeningFcn,
                           ...
35                    'gui_OutputFcn',  @ersteOberflaeche_OutputFcn,
                           ...
36                    'gui_LayoutFcn',  [] , ...
37                    'gui_Callback',   []);
38 if nargin && ischar(varargin{1})
39     gui_State.gui_Callback = str2func(varargin{1});
40 end
41
42 if nargout
43     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
44 else
45     gui_mainfcn(gui_State, varargin{:});
46 end
47 % End initialization code - DO NOT EDIT
48
49 %% Funktion die einmalig beim Öffnen des Programmes ausgeführt
       wird
50 function ersteOberflaeche_OpeningFcn(hObject, eventdata, handles,
       varargin)
51 % This function has no output args, see OutputFcn.
52 % hObject    handle to figure
53 % eventdata  reserved - to be defined in a future version of
       MATLAB
54 % handles    structure with handles and user data (see GUIDATA)
55 % varargin   command line arguments to ersteOberflaeche (see
       VARARGIN)
56
57 % Choose default command line output for ersteOberflaeche
58 handles.output = hObject;
59 guidata(hObject, handles);
60 % Update handles structure
61 raum = [6 10 3.5];  %Initialisierung des Raumes
```

```
62 mic1 = [3 2 1];      %Initialisierung der Mikrofone
63 mic2 = [3.045 2 1];
64 mic3 = [3.09 2 1];
65 mic4 = [3.135 2 1];
66 mic5 = [3.18 2 1];
67 mic6 = [3.225 2 1];
68 micarr = [mic1; mic2; mic3; mic4; mic5; mic6];  %Erzeugen eines
      Mikrofonarrays
69 colname = {'Breite','Tiefe','Höhe'};    %Initialisierung eines
      Cellarray zur Beschriftung der Tabellen
70 %Initialisierung eines Cellarray zur Beschriftung der
      Mikrofontabelle
71 %Hierzu wird eine selbsterstellte Funktion aufgerufen
72 rowname = string_array_erstellen(size(micarr), 'Mikrofon');
73 set(handles.uitable_mic,'Data',micarr); %Daten werden an die
      Mikrofontabelle übergeben
74 set(handles.uitable_mic,'ColumnName',colname);  %
      Spaltenbeschriftung wird an die Mikrofontabelle übergeben
75 set(handles.uitable_mic,'RowName',rowname); %Zeilenbeschriftung
      wird an die Mikrofontabelle übergeben
76
77 aud1 = [3 8 1.7];   %Initialisierung der Audioquellen
78 aud2 = [4 8 1.7];
79 audarr = [aud1; aud2];  %Erzeugen eines Audioarray
80 colname = {'Breite','Tiefe','Höhe'};    %Initialisierung eines
      Cellarray zur Beschriftung der Tabellen
81 %Initialisierung eines Cellarray zur Beschriftung der
      Mikrofontabelle
82 %Hierzu wird eine selbsterstellte Funktion aufgerufen
83 rowname = string_array_erstellen(size(audarr), 'Audioquelle');
84 set(handles.uitable_aud,'Data',audarr); %Daten werden an die
      Audiotabelle übergeben
85 set(handles.uitable_aud,'ColumnName',colname);  %
      Spaltenbeschriftung wird an die Audiotabelle übergeben
86 set(handles.uitable_aud,'RowName',rowname); %Zeilenbeschriftung
      wird an die Audiotabelle übergeben
87 set(handles.listbox_audquellen, 'String', rowname); %Beschriftung
      der Listbox
88
89 handles.aud_dateien(1) = {[]};  %Initialisierung des Dateien-
      String
90 guidata(hObject, handles);  %Aktualisieren der Handles
      Fensterstruktur
91
```

```
92  axes(handles.axes1);    %Axes zur Verfügung stellen
93  x1 = micarr(1:size(micarr),1);  %Zerlegen der Mikrofon- und
       Audioarrays in Koordinatenvektoren
94  y1 = micarr(1:size(micarr),2);
95  z1 = micarr(1:size(micarr),3);
96  x2 = audarr(1:size(audarr),1);
97  y2 = audarr(1:size(audarr),2);
98  z2 = audarr(1:size(audarr),3);
99  plot3(x1,y1,z1,'+',x2,y2,z2,'o');   %Plot des Raums mit Mikrofon
       und Audioquellen
100 grid;
101 AXIS([0 raum(1) 0 raum(2) 0 raum(3)])   %Plot an die Abmaße des
       Raumes anpassen
102 pbaspect([raum(1) raum(2) raum(3)])     %Seitenverhältnisse des
       Plot an die Raumabmessungen anpassen
103 xlabel('Breite');
104 ylabel('Tiefe');
105 zlabel('Höhe');
106
107 %Erzeugen eines Stringarray für das Mikrofonpopupmenu
108 %Dazu wird eine selbsterstellte Funktion aufgerufen
109 popstr = string_array_erstellen(size(micarr), 'Mikrofon');
110 set(handles.popupmenu_mic_del,'String',popstr);     %Stringarray
       in das Popupmenu zum Löschen von Mikrofonen laden
111 set(handles.popupmenu_mic_auswahl,'String',popstr); %Stringarray
       in das Popupmenu zum Anzeigen der Distanzen laden
112
113 %Erzeugen eines Stringarray für das Audiopopupmenu
114 %Dazu wird eine selbsterstellte Funktion aufgerufen
115 popstr = string_array_erstellen(size(audarr), 'Audioquelle');
116 set(handles.popupmenu_aud_del,'String',popstr); %Stringarray in
       das Popupmenu zum Löschen von Audioquellen laden
117 handles.fa = 100000;         %Simulationsfrequenz auf 100kHz
       vordefiniert
118 handles.raum = raum;         %Raumabmessung in eine Handlesvariable
        schreiben
119 handles.micarr = micarr;     %Mikrofonarray in eine Handlesvariable
        schreiben
120 handles.audarr = audarr;     %Audioquellenarray in eine
       Handlesvariable schreibe
121 guidata(hObject, handles);  %Aktuallisieren der Handles
       Fensterstruktur
122
123 %% Funktion zum aktualisieren alle Tabellen, Menus und den Plot
```

```
124 %Diese Funktion wird immer aufgerufen nachdem handlesvariablen
    aktualisiert wurden.
125 function update_fig_uitable(hObject, eventdata, handles)
126
127 %update MIC_UITABLE
128 colname = {'Breite','Tiefe','Höhe'};    %
    Spaltenbeschriftungsstring wird erzeugt
129 %Zeilenbeschriftungsstring wird erzeugt, hierzu wird eine
130 %selbsterstellte Funktion aufgerufen
131 rowname = string_array_erstellen(size(handles.micarr), 'Mikrofon')
    ;
132 set(handles.uitable_mic,'Data',handles.micarr); %Daten werden in
    die Tabelle geladen
133 set(handles.uitable_mic,'ColumnName',colname);  %
    Spaltenbeschriftung wird in die Tabelle geladen
134 set(handles.uitable_mic,'RowName',rowname);     %
    Zeilenbeschriftung wird in die Tabelle geladen
135
136 %update AUD_UITABLE
137 rowname = string_array_erstellen(size(handles.audarr), '
    Audioquelle');
138 set(handles.uitable_aud,'Data',handles.audarr); %Daten werden in
    die Tabelle geladen
139 set(handles.uitable_aud,'ColumnName',colname);  %
    Spaltenbeschriftung wird in die Tabelle geladen
140 set(handles.uitable_aud,'RowName',rowname);     %
    Zeilenbeschriftung wird in die Tabelle geladen
141
142 %update LISTBOX
143 test = 'Hallo';
144 listbox_string = {test};    %Damit ein Cellarray in einer Schleife
     erweitert werden kann, muss Sie vordeklariert werden
145 handles.alle_dateien_geladen = 1;   %Merker um festzustellen ob zu
     jeder Audioquelle eine WAV-Datei geladen ist
146 guidata(hObject, handles);          %Aktualisieren der Handles
    Fensterstruktur
147 for i=1:size(handles.audarr)    %Schleife läuft von 1 bis zur
    Anzahl von Audioquellen
148     if isnan(handles.audarr)    %Wenn keine Audioquelle eingegeben
        ist wird die Schleife hier abgebrochen
149         break;
150     end
```

```
151     if length(handles.aud_dateien)>= i  %Wenn die Länge des
           DateienStringarrays länger-gleich dem Schleifendurchlauf
           ist, dann:
152         if iscellstr(handles.aud_dateien(i))    %Wenn
               Dateienstring von i einen Inhalt hat, dann:
153             listbox_char = [char(rowname(i)) ': ' char(handles.
                   aud_dateien(i))];    %Erstelle Chararray mit
                   Audioquelle_i + Dateienstring von i
154             listbox_string(i) = {listbox_char}; %Weise den
                   chararray dem Cellarray von i zu
155         else
156             listbox_char = [char(rowname(i)) ': ' '----keine Datei
                   Ausgewählt'];    %Erstelle Chararray mit
                   Audioquelle_i +  ----keine Datei ausgewählt
157             listbox_string(i) = {listbox_char}; %Weise den
                   chararray dem Cellarray von i zu
158             handles.alle_dateien_geladen = 0;   %Merker um
                   festzustellen ob zu jeder Audioquelle eine WAV-
                   Datei geladen ist, auf 0 setzten
159             guidata(hObject, handles);          %Aktualisieren der
                    Handles Fensterstruktur
160         end
161     else
162         listbox_char = [char(rowname(i)) ': ' '----keine Datei
               Ausgewählt'];    %Erstelle Chararray mit Audioquelle_i
               +  ----keine Datei Ausgewählt
163         listbox_string(i) = {listbox_char}; %Weise den chararray
               dem Cellarray von i zu
164         handles.alle_dateien_geladen = 0;   %Merker um
               festzustellen ob zu jeder Audioquelle eine WAV-Datei
               geladen ist, auf 0 setzten
165         guidata(hObject, handles);          %Aktualisieren der
               Handles Fensterstruktur
166     end
167 end
168 assignin('base', 'sfsdgfn',listbox_string)
169 set(handles.listbox_audquellen,'String',char(listbox_string));  %
       Zuweisung des Cellarray an die Listbox
170
171 %update AXES
172 axes(handles.axes1);    %Axes zur Verfügung stellen
173 %Zerlegen des Mikrofonarray in Koordinaten, wenn es existiert
174 if ~isnan(handles.micarr)
175     x1 = handles.micarr(1:size(handles.micarr),1);
```

```
176     y1 = handles.micarr(1:size(handles.micarr),2);
177     z1 = handles.micarr(1:size(handles.micarr),3);
178 end
179 %Zerlegen des Audioarray in Koordinaten wenn es existiert
180 if ~isnan(handles.audarr)
181     x2 = handles.audarr(1:size(handles.audarr),1);
182     y2 = handles.audarr(1:size(handles.audarr),2);
183     z2 = handles.audarr(1:size(handles.audarr),3);
184 end
185 %Plot der Mikrofone und Audioquellen je nachdem ob welche
    vorhanden sind
186 if ~isnan(handles.micarr)
187     if ~isnan(handles.audarr)
188         plot3(x1,y1,z1,'+',x2,y2,z2,'o');
189     else
190         plot3(x1,y1,z1,'+');
191     end
192 else
193     if ~isnan(handles.audarr)
194         plot3(x2,y2,z2,'o');
195     else
196         plot3(0,0,0);
197     end
198 end
199
200 grid;
201 AXIS([0 handles.raum(1) 0 handles.raum(2) 0 handles.raum(3)])    %
    Plot an die Raumabmessungen anpassen
202 pbaspect([handles.raum(1) handles.raum(2) handles.raum(3)])      %
    Seitenverhältnisse an die Raumabmessungen anpassen
203 xlabel('Breite');
204 ylabel('Tiefe');
205 zlabel('Höhe');
206
207 %update Popup_menü zum Löschen von Mikrofonen
208 %Erzeugen ein Cellarray für die Mikrofonarray
209 %Dazu wird eine selbsterstellte Funktion verwendet
210 popstr = string_array_erstellen(size(handles.micarr), 'Mikrofon');
211 set(handles.popupmenu_mic_del,'String',popstr); %erzeugtes
    Cellarray dem Popupmenu zum Löschen von Mikrofonen zuweisen
212 set(handles.popupmenu_mic_auswahl,'String',popstr); %erzeugtes
    Cellarray dem Popupmenu zum Anzeigen der Distanzen zuweisen
213
214 %update Popup_menü zum löschen von Audioquellen
```

```
215 %Erzeugen eines Cellarray für das Audioquellenarray
216 %Dazu wird eine selbsterstellte Funktion verwendet
217 popstr = string_array_erstellen(size(handles.audarr), 'Audioquelle
    ');
218 set(handles.popupmenu_aud_del,'String',popstr); %erzeugtes
    Cellarray dem Popupmenu zum Löschen von Audioquellen zuweisen
219
220 %Hier werden alle Bedienflächen für die berechneten Parameter
    deaktiviert,
221 %da diese nach dem Verändern eines Mikrofons oder Audioquelle neu
    berechnet
222 %werden müssen.
223 set(handles.popupmenu_mic_auswahl,'Enable','off');       %Popupmenu
     zum Auswählen der Distanzen deaktivieren
224 set(handles.uitable_sim_daten,'Enable','off');           %Tabelle
    mit Distanzen deaktivieren
225 set(handles.pushbutton_start_sim,'Enable','off');        %
    Simulationsstartbutton deaktiveren
226 set(handles.popupmenu_parameter_auswahl,'Enable','off');%Popupmenu
     zum Auswälen der Parameter deaktiveren
227
228 %% Funktion zum Ausgeben von Rückgabewerten
229 % --- Outputs from this function are returned to the command line.
230 function varargout = ersteOberflaeche_OutputFcn(hObject, eventdata
    , handles)
231 % varargout  cell array for returning output args (see VARARGOUT);
232 % hObject    handle to figure
233 % eventdata  reserved - to be defined in a future version of
    MATLAB
234 % handles    structure with handles and user data (see GUIDATA)
235
236 % Get default command line output from handles structure
237 varargout{1} = handles.output;
238
239 %% Funktion zur Initialisierung der Mikrofontabelle
240 function uitable_mic_CreateFcn(hObject, eventdata, handles)
241
242 %% Funktion zur Initialisierung der Graphik
243 function axes1_CreateFcn(hObject, eventdata, handles)
244
245 %% Funktion zur Initialisierung der Audioquelle
246 function uitable_aud_CreateFcn(hObject, eventdata, handles)
247
```

```
248 %% Funktion zum verarbeitet des Callback vom Popupmenu für die
    Auswahl der Mikrofone
249 function popupmenu_mic_auswahl_Callback(hObject, eventdata,
    handles)
250 ersteOberflaeche('popup_menu',gcbo,[],guidata(gcbo));   %Hier wird
     eine Funktion aufgerufen die den Callback mehrerer Popupmenus
    verarbeitet
251
252 %% Funktion zur Initialisierung vom Popupmenu für die Auswahl der
    Mikrofone
253 function popupmenu_mic_auswahl_CreateFcn(hObject, eventdata,
    handles)
254 %Dieser Code wird von Matlab automatisch erzeugt und legt die
255 %Hintergrundfarbe des Popupmenues fest
256 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
257     set(hObject,'BackgroundColor','white');
258 end
259
260 %% Funktion zum verarbeitet der Callbacks von den Popupmenu vom
    Distanzen-Panel
261 function popup_menu(hObject, eventdata, handles) %selbsterstellter
     Funktionskopf,
262 %Mit dieser Funktion werden die Callbacks der Popupmenus
    verarbeitet, je
263 %nach Auswahl werden bestimmte Daten auf der Tabelle ausgegeben
264 %Die Parameter werden in der Callback-Eigenschaft eingestellt.
265 auswahl_mic = get(handles.popupmenu_mic_auswahl,'Value');
266 auswahl_parameter = get(handles.popupmenu_parameter_auswahl,'Value
    '); %Radien oder Laufzeiten oder Dämpfungen
267
268 %Die Funktion struktur_zu_array ist eine selbsterstellte Funktion
    und
269 %wird in dem Quellcode der Funktion erklärt
270 switch auswahl_parameter %Radien oder Laufzeiten oder Dämpfungen
271     case 1  %Radien
272         [datenblock zeilennamen spaltennamen] = struktur_zu_array(
              handles.radien(auswahl_mic));
273     case 2  %Laufzeiten
274         [datenblock zeilennamen spaltennamen] = struktur_zu_array(
              handles.laufzeiten(auswahl_mic));
275     case 3  %Dämpfungen
276         [datenblock zeilennamen spaltennamen] = struktur_zu_array(
              handles.daempfung(auswahl_mic));
```

```
277 end
278 %Hier werden die geladenen Daten in die Tabelle mit Beschriftung
       geladen
279 set(handles.uitable_sim_daten,'Data',datenblock');
280 set(handles.uitable_sim_daten,'ColumnName',zeilennamen);
281 set(handles.uitable_sim_daten,'RowName',spaltennamen);
282
283
284 %% Callback-Funktion vom Button der die Simulation startet
285 function pushbutton_start_sim_Callback(hObject, eventdata, handles
       )
286 erfolgreich = 0;
287 %assignin('base','laufzeiten',handles.laufzeiten);
288 %assignin('base','daempfungen',handles.daempfung);
289 %assignin('base','dateien',handles.aud_dateien);
290 set(handles.pushbutton_start_sim,'Enable','off');
291 %Hier wird die Auswahl der Anzahl der Reflexionen/
       Oberflächenreflexionsfaktor ausgewertet
292 ref_auswahl = get(handles.radiobutton_ohne_ref, 'Value')*1 + get(
       handles.radiobutton_ref_I, 'Value')*2 + get(handles.
       radiobutton_ref_II, 'Value')*3 + get(handles.
       radiobutton_Wandparameter, 'Value')*4;
293 %assignin('base','ref_auswahl',ref_auswahl);
294 tic
295 switch ref_auswahl
296     case 1  %Keine Reflexionen
297       erfolgreich = simulationsstart_ohne_ref(handles.laufzeiten,
             handles.daempfung, handles.fa, handles.aud_dateien);
298     case 2  %Reflexionen bis zur ersten Ordnung
299       erfolgreich = simulationsstart_ref_I(handles.laufzeiten,
             handles.daempfung, handles.fa, handles.aud_dateien);
300     case 3  %Reflexionen bis zur zweiten Ordnung
301       erfolgreich = simulationsstart(handles.laufzeiten, handles.
             daempfung, handles.fa, handles.aud_dateien);
302     case 4 %Reflexionen bis zur zweiten Ordnung und
          Oberflächenreflexionsfaktor
303       %Hier wird die Auswahl der Oberflächenbeschaffenheit
             ausgewertet
304       decke = get(handles.radiobutton_decke_beton, 'Value')*1 +
             get(handles.radiobutton_decke_metall, 'Value')*2 + get(
             handles.radiobutton_decke_raster, 'Value')*3 + get(
             handles.radiobutton_decke_stroh, 'Value')*4 + get(handles
             .radiobutton_decke_teppich, 'Value')*5;
```

```
305         fussboden = get(handles.radiobutton_fuss_linoleum, 'Value')
               *6 + get(handles.radiobutton_fuss_fliesen, 'Value')*7 +
               get(handles.radiobutton_fuss_parkett, 'Value')*8 + get(
               handles.radiobutton_fuss_teppich, 'Value')*9;
306         wand = get(handles.radiobutton_wand_beton, 'Value')*10 + get
               (handles.radiobutton_wand_stein, 'Value')*11 + get(
               handles.radiobutton_wand_hohl, 'Value')*12 + get(handles.
               radiobutton_wand_teppich, 'Value')*13;
307         fenster = get(handles.radiobutton_fenster_gardienen, 'Value'
               )*14 + get(handles.radiobutton_fenster_fenster, 'Value')
               *15 + get(handles.radiobutton_fenster_wand, 'Value')*16;
308         if fenster == 16 %Wenn die Fensterseite wie eine normale
               Wand behandelt werden soll!
309             fenster = wand;
310         end
311         erfolgreich = simulationsstart_oberfl(handles.laufzeiten,
               handles.daempfung, handles.fa, handles.aud_dateien, decke
               , fussboden, wand, fenster);
312 end
313 set(handles.pushbutton_start_sim,'Enable','on');
314 %simulationsmodel
315 toc
316 if erfolgreich==1
317     disp('Raumsimulation␣Erfolgreich␣abgeschlossen!')
318 end
319
320
321 %% Callback-Funktion vom Button der Mikrofone löscht
322 function pushbutton_mic_del_Callback(hObject, eventdata, handles)
323 % Hier wird die selbsterstellte Funktion zeile_aus_array_loeschen
324 % verwendet, die Funktion wird im Quellcode von der Funktion
       erklärt
325 handles.micarr = zeile_aus_array_loeschen(handles.micarr, get(
       handles.popupmenu_mic_del,'Value'));
326 guidata(hObject, handles);  %Handlesvariable wird aktualisiert
327 ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo));   %
       Aktualisierungsfunktion wird aufgerufen
328
329 %% Callback-Funktion vom Button der Mikrofone hinzufügt
330 function pushbutton_mic_new_Callback(hObject, eventdata, handles)
331 breite = get(handles.edit_breite, 'String');     %Textfeld für die
       Breite wird ausgelesen
332 tiefe = get(handles.edit_tiefe, 'String');       %Testfeld für die
       Tiefe wird ausgelesen
```

```
333 hoehe = get(handles.edit_hoehe, 'String');      %Textfeld für die
    Höhe wird ausgelesen
334 %Alle  eingelesenen Werte werden von String zu double konvertiert
335 %Sind keine Zahlen in die Felder eingetragen worden, wird NaN
    zurückgegeben
336 breite = str2double(breite);
337 tiefe = str2double(tiefe);
338 hoehe = str2double(hoehe);
339 if (isnan(breite)) || (isnan(hoehe)) || (isnan(tiefe))  %Abfrage,
    ob eine Angaben eine Zahl war
340     MsgBox('Hier ist wohl ein Janusz am Werk!','Wuaas passiert
        wenn ich daaahs mache!');
341 else
342     zeile = [abs(breite) abs(tiefe) abs(hoehe)];    %Die Eingaben
        werden zu einen Koordinatenvektor zusammengefügt
343     %Hier wird die selbsterstellte Funktion is_array_im_Raum
        verwendet,
344     %diese Funktion wird im Quellcode der Funktion erklärt
345     if is_arrays_im_raum(zeile, handles.audarr, handles.raum)   %
        Überprüffung, ob die Koordinaten in den Raum passen
346         %Hier wird die selbsterstellte Funktion add_zeile_zu_array
            ()
347         %aufgerufen, diese Funktion wird im Quellcode der Funktion
             erklärt
348         handles.micarr = add_zeile_zu_array(handles.micarr, zeile)
            ; %Der Koordinatenvektor wird dem Mikrofonarray
            hinzugefügt
349         guidata(hObject, handles);  %Aktualisierung der
            Handlesvariable
350         ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo
            ));   %Aktualisierungsfunktion wird aufgerufen
351     else
352         MsgBox('Das Mikrofon ist nicht im Raum','Fehler');  %
            Messageausgabe, falls der Koordinatenvektor nicht in
            den Raum passt
353     end
354 end
355
356 %% Callback-Funktion vom Button der Audioquellen hinzufügt
357 function pushbutton_aud_new_Callback(hObject, eventdata, handles)
358 breite = get(handles.edit_breite, 'String');    %Textfeld für die
    Breite wird ausgelesen
359 tiefe = get(handles.edit_tiefe, 'String');      %Textfeld für die
    Tiefe wird ausgelesen
```

```
360 hoehe = get(handles.edit_hoehe, 'String');      %Textfeld für die
    Höhe wird ausgelesen
361 %Alle  eingelesenen Werte werden von String zu double konvertiert
362 %Sind keine Zahlen in die Felder eingetragen worden wird NaN
    zurückgegeben
363 breite = str2double(breite);
364 tiefe = str2double(tiefe);
365 hoehe = str2double(hoehe);
366 if (isnan(breite)) || (isnan(hoehe)) || (isnan(tiefe))   %Abfrage,
     ob eine Angaben eine Zahl war
367     MsgBox('Hier ist wohl ein Janusz am Werk!','Wuaas passiert
        wenn ich daaahs mache!');
368 else
369     zeile = [abs(breite) abs(tiefe) abs(hoehe)];    %Die Eingaben
        werden zu einen Koordinatenvektor zusammengefügt
370     %Hier wird die selbsterstellte Funktion is_array_im_Raum
        verwendet,
371     %diese Funktion wird im Quellcode der Funktion erklärt
372     if is_arrays_im_raum(handles.micarr, zeile, handles.raum)    %
        Überprüfung, ob die Koordinaten in den Raum passen
373         %Hier wird die selbsterstellte Funktion add_zeile_zu_array
                ()
374         %aufgerufen, diese Funktion wird im Quellcode der Funktion
                erklärt
375         handles.audarr = add_zeile_zu_array(handles.audarr, zeile)
            ; %Der Koordinatenvektor wird dem Audioarray
            hinzugefügt
376         guidata(hObject, handles);  %Aktualisierung der
            Handlesvariable
377         ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo
            ));   %Aktualisierungsfunktion wird aufgerufen
378     else
379         MsgBox('Die Audioquelle ist nicht im Raum','Fehler');   %
            Messageausgabe, falls der Koordinatenvektor nicht in
            den Raum passt
380     end
381 end
382
383 %% Callback-Funktion vom Editfeld für die Tiefe
384 function edit_tiefe_Callback(hObject, eventdata, handles)
385
386 %% Funktion zur Initialisierung des Editfeld von der Tiefe
387 function edit_tiefe_CreateFcn(hObject, eventdata, handles)
```

```
388 %Die folgenden Zeilen sind von GUIDE automatisch erstellt worden
    und setzen
389 %die Hintergrundfarbe des Editfeldes fest
390 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
391     set(hObject,'BackgroundColor','white');
392 end
393
394 %% Callback-Funktion vom Editfeldes für die Breite
395 function edit_breite_Callback(hObject, eventdata, handles)
396
397 %% Funktion zur Initialisierung des Editfeld von der Breite
398 function edit_breite_CreateFcn(hObject, eventdata, handles)
399 %Die folgenden Zeilen sind von GUIDE automatisch erstellt worden
    und setzen
400 %die Hintergrundfarbe des Editfeldes fest
401 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
402     set(hObject,'BackgroundColor','white');
403 end
404
405 %% Callback-Funktion vom Editfeldes für die Höhe
406 function edit_hoehe_Callback(hObject, eventdata, handles)
407
408 %% Funktion zur Initialisierung des Editfeld von der Höhe
409 function edit_hoehe_CreateFcn(hObject, eventdata, handles)
410 %Die folgenden Zeilen sind von GUIDE automatisch erstellt worden
    und setzen
411 %die Hintergrundfarbe des Editfeldes fest
412 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
413     set(hObject,'BackgroundColor','white');
414 end
415
416 %% Callback-Funktion vom Popupmenu zum Löschen von Mikrofone
417 function popupmenu_mic_del_Callback(hObject, eventdata, handles)
418
419 %% Funktion zur Initialisierung des Popupmenu zum Löschen von
    Mikrofone
420 function popupmenu_mic_del_CreateFcn(hObject, eventdata, handles)
421 %Die folgenden Zeilen sind von GUIDE automatisch erstellt worden
    und setzen
422 %die Hintergrundfarbe des Popupmenus fest
```

```
423 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
424     set(hObject,'BackgroundColor','white');
425 end
426
427 %% Callback-Funktion des Button zum Löschen von Audioquellen
428 function pushbutton_aud_del_Callback(hObject, eventdata, handles)
429 %Die Funktion zeile_aus_array Löschen ist eine selbsterstellte
    Funktion,
430 %deren Funktionalität im Quellcode der Funktion erklärt wird
431 handles.audarr = zeile_aus_array_loeschen(handles.audarr, get(
    handles.popupmenu_aud_del,'Value'));  %Hier wird eine Zeile die
     vorher ausgewählt wurde aus dem Audioarray gelöscht
432 guidata(hObject, handles);  %Aktualisierung der Handlesvariable
433 ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo));   %
    Aktualisierungsfunktion wird aufgerufen
434
435 %% Callback-Funktion von dem Popupmenu zum löschen von
    Audioquellen
436 function popupmenu_aud_del_Callback(hObject, eventdata, handles)
437
438 %% Funktion zur Initialisierung des Popupmenus zum löschen von
    Audioquellen
439 function popupmenu_aud_del_CreateFcn(hObject, eventdata, handles)
440 %Die folgenden Zeilen sind von GUIDE automatisch erstellt worden
    und setzen
441 %die Hintergrundfarbe des Popupmenus fest
442 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
443     set(hObject,'BackgroundColor','white');
444 end
445
446 %% Callback-Funktion von dem Drop-down-Menü Datei
447 function menu_datei_Callback(hObject, eventdata, handles)
448
449 %% Callback-Funktion vom Drop-down-Menü Datei/Datei speichern
450 function menu_datei_save_Callback(hObject, eventdata, handles)
451 %Die Funktion arrays_speichern() ist eine selbsterstellte Funktion
452 %deren Fuktionalität im Quellcode der Funktion erklärt wird
453 %Hier werden die Raumabmessungen, Mikrofonarrays und Audioquellen
    in eine
454 %Datei abgespeichert
455 erfolgreich = arrays_speichern(handles.micarr, handles.audarr,
    handles.raum);
```

```
456 if erfolgreich == 1 %Wenn das abspeichern erfolgreich war, dann
457     MsgBox('Datei Erfolgreich gespeichert');    %Messagebox
458 else
459     MsgBox('Datei konnte nicht gespeichert werden!');   %
        Messagebox
460 end
461
462 %% Callback-Funktion vom Drop-down-Menü Datei/Datei Laden
463 function menu_datei_load_Callback(hObject, eventdata, handles)
464 %Die Funktion datei_laden() ist eine selbsterstellte Funktion
    deren
465 %Funktionalität im Quellcode der Funktion erklärt wird
466 %Es werden Raumabmessungen, Mikrofonarrays und Audioquellen
    geladen
467 [handles.micarr handles.audarr handles.raum erfolgreich] =
    arrays_laden();
468 if erfolgreich == 1 %Wenn das Laden erfolgreich war, dann
469     guidata(hObject, handles);  %Aktualisierung die
        Handlesvariablen
470     ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo));
          %Aufruf der Aktualisierungsfunktion
471     MsgBox('Datei Erfolgreich geladen');    %Messagebox
472 else
473     MsgBox('Datei konnte nicht geladen werden!');   %Messagebox
474 end
475
476 %% Callback-Funktion vom Button, der die Raumabmessung aufnimmt
477 function pushbutton_raumabmessung_Callback(hObject, eventdata,
    handles)
478 breite = get(handles.edit_breite, 'String');    %Editfeld für die
    Breite wird ausgelesen und in die Variable Breite geschrieben
479 tiefe = get(handles.edit_tiefe, 'String');      %Editfeld für die
    Tiefe wird ausgelesen und in die Variable Tiefe geschrieben
480 hoehe = get(handles.edit_hoehe, 'String');      %Editfeld für die
    Höhe wird ausgelesen und in die Variable Höhe geschrieben
481 breite = str2double(breite);    %Die Breite wird von String in
    einen Double konvertiert
482 tiefe = str2double(tiefe);      %Die Tiefe wird von String in
    einen Double konvertiert
483 hoehe = str2double(hoehe);      %Die Höhe wird von String in einen
     Double konvertiert
484 if (isnan(breite)) || (isnan(hoehe)) || (isnan(tiefe))  %Abfrage:
    Ist eine der eingelesenen Parameter keine Zahl
```

```
485     MsgBox('Hier ist wohl ein Janusz am Werk!','Wuaas passiert
           wenn ich daaahs mache!');     %Messagebox
486 else     %alle Parameter sind Zahlen
487     zeile = [abs(breite) abs(tiefe) abs(hoehe)];     %Erzeuge
           Raumabmessung aus Parametern
488     %Die Funktion is_arrays_im_raum() ist eine selbsterstellte
           Funktion dessen Funktionalität im Quellcode erklärt wird
489     if is_arrays_im_raum(handles.micarr, handles.audarr, zeile);
              %Wenn die Raumabmessung groß genug für die Arrays ist,
           dann
490         handles.raum = zeile;    %Raumabmessung der
               Handlesvariablen Raum zuweisen
491         guidata(hObject, handles);  %Handlesvariable aktualisieren
492         ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo
               ));    %Aktualisierungfunktion aufrufen
493     else     %Ist der Raum zu klein für die Arrays, dann
494         MsgBox('Der Raum ist zu klein für die Audioquellen- und
               Mikrofonpositionen','Fehler!'); %Massagebox
495     end
496 end
497
498 %% Callback-Funktion vom Button, der die Simulationsparameter
       berechnet
499 function pushbutton_berechnung_Callback(hObject, eventdata,
       handles)
500 %Die Funktion berecnung_aller_radien() ist eine selbsterstellte
       Funktion
501 %dessen Funktionalität im Quellcode der Funktion erklärt wird
502 handles.radien = berechnung_aller_radien(handles.micarr, handles.
       audarr, handles.raum); %Berechnung aller Radien  I. II. Ordnung
        Reflektionen
503 guidata(hObject, handles);  %Aktualisierung der Handlesvariable
504 %Die Funktion berechne_laufzeiten() ist eine selbsterstellte
       Funktion, dessen Funktionalität im Quellcode, der Funktion
       erklärt wird
505 handles.laufzeiten = berechne_laufzeiten(handles.radien);   %
       Berechnung der Laufzeiten aus den Radien
506 guidata(hObject, handles);  %Aktualisierung der Handlesvariable
507 %Die Funktion berechne_daempfung() ist eine selbsterstellte
       Funktion
508 %dessen Funktionalität im Quellcode der Funktion beschrieben wird
509 handles.daempfung = berechne_daempfung(handles.radien); %
       Berechnung der Dämpfungen aus den Radien
510 guidata(hObject, handles);  %Aktualisierung der Handlesvariable
```

```
511 set(handles.popupmenu_mic_auswahl,'Enable','on');    %
    Mikrofonauswahl Popupmenu aktivieren
512 set(handles.uitable_sim_daten,'Enable','on');        %
    Simulationsparameter Tabelle aktivieren
513 set(handles.pushbutton_start_sim,'Enable','on');     %
    Simulationsstart Button aktivieren
514 set(handles.popupmenu_parameter_auswahl,'Enable','on'); %
    Parameterauswahl Popupmenu aktivieren
515 auswahl = get(handles.popupmenu_mic_auswahl,'Value');   %
    Mikrofonauswahl abrufen
516 %Die Funktion struktur_zu_array() ist eine selbsterstellte
    Funktion, dessen Funktionalität im Quellcode der Funktion
    beschrieben wird
517 [datenblock spaltennamen zeilennamen] = struktur_zu_array(handles.
    radien(auswahl)); %Radienstruktur auslesen
518 set(handles.uitable_sim_daten,'Data',datenblock');          %
    Datenblock in die Simulationsparameter Tabelle einlesen
519 set(handles.uitable_sim_daten,'ColumnName',spaltennamen);   %
    Spaltenbeschriftung in die Simulationsparameter Tabelle
    einlesen
520 set(handles.uitable_sim_daten,'RowName',zeilennamen);       %
    Zeilenbeschriftung in die Simulationsparameter Tabelle
    einlesen
521
522 %% Callback Funktion vom Popupmenu zur Parameterauswahl
523 function popupmenu_parameter_auswahl_Callback(hObject, eventdata,
    handles)
524 %Hier wird die Funktion popup_menu() aufgerufen
525 ersteOberflaeche('popup_menu',gcbo,[],guidata(gcbo));
526
527 %% Funktion zur Initialisierung das Popupmenu zur Parameterauswahl
528 function popupmenu_parameter_auswahl_CreateFcn(hObject, eventdata,
     handles)
529 %Die folgenden Zeilen sind von MATLAB automatisch erstellt worden
    und
530 %definieren den Hintergrund des Popupmenus "Weiß"
531 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
532     set(hObject,'BackgroundColor','white');
533 end
534
535 %% Callback-Funktion der Listbox für die Audiodateien
536 function listbox_audquellen_Callback(hObject, eventdata, handles)
537
```

```
538 %% Funktion zur Initialsierung der Listbox für die Audiodateien
539 function listbox_audquellen_CreateFcn(hObject, eventdata, handles)
540 %Die folgenden Zeilen sind von Matlab automatisch erstellt worden
    und
541 %definieren den Hintergrund der Listbox als "Weiß"
542 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
543     set(hObject,'BackgroundColor','white');
544 end
545
546 %% Callback-Funktion des Button für das Laden einer Audiodatei
547 function pushbutton_lade_datei_Callback(hObject, eventdata,
    handles)
548 aud_auswahl = get(handles.listbox_audquellen,'Value');  %Die
    Auswahl aus der Listbox wird ausgelesen
549 %Die Funktion string_array_erstellen() ist eine selbsterstellte
    Funktion dessen Funktionalität im Quellcode der Funktion
    beschrieben wird
550 lade_dialog_beschriftungs_string = string_array_erstellen(size(
    handles.audarr), 'Audioquelle'); %Es wird ein
    Beschriftungsstring für den Ladedialog erstellt
551 lade_dialog_beschriftung = lade_dialog_beschriftungs_string(
    aud_auswahl);   %Es wird ein Beschriftungsstring für den
    Ladedialog erstellt
552 %Die Funktion lade_wav() ist eine selbsterstellte Funktion dessen
    Funktionalität im Quellcode der Funktion beschrieben wird
553 datei = lade_wav(lade_dialog_beschriftung); %Es wird ein Dateipfad
     geladen und auf die Variable datei geschrieben
554 handles.aud_dateien(aud_auswahl) = {datei}; %Der geladene
    Dateipfad wird dem Cellarray handles.aud_dateien zugefügt
555 guidata(hObject, handles);  %die Handlesvariable wird aktualisiert
556 ersteOberflaeche('update_fig_uitable',gcbo,[],guidata(gcbo));   %
    Die Aktualisierungfunktion wird aufgerufen
557
558 %% Callback-Funktion vom Popupmenu zur Auswahl der Winkelauflösung
559 % --- Executes on selection change in popupmenu_theta_a.
560 function popupmenu_theta_a_Callback(hObject, eventdata, handles)
561 % hObject    handle to popupmenu_theta_a (see GCBO)
562 % eventdata  reserved - to be defined in a future version of
    MATLAB
563 % handles    structure with handles and user data (see GUIDATA)
564 Winkelaufloesung_cell = str2double(get(handles.popupmenu_theta_a,'
    String')); %Popupmenu für die Winkelauflösung auslesen
```

```
565 Oeffnungswinkel_cell = str2double(get(handles.popupmenu_theta,'String')); % Popupmenu für den Öffnungswinkel auslesen
566 Winkelaufloesung_auswahl = get(handles.popupmenu_theta_a,'Value');
     %Popupmenu für die Winkelauflösung auslesen
567 Oeffnungswinkel_auswahl = get(handles.popupmenu_theta,'Value'); %Popupmenu für den Öffnungswinkel auslesen
568 Winkelaufloesung = Winkelaufloesung_cell(Winkelaufloesung_auswahl);
569 Oeffnungswinkel = Oeffnungswinkel_cell(Oeffnungswinkel_auswahl);
570 fa = Simulationsfrequenz_berechnen(Winkelaufloesung, Oeffnungswinkel, handles.micarr); %Simulationsfrequenz in der Funktion "Simulationsfrequenz_berechnen" berechnen
571 fa_string = ['Simulationsfrequenz: ', num2str(fa/1000), 'kHz']; %String erzeugen mit Simulationsfrequenz um die Simulationsfrequenz im Simulationsprogramm anzuzeigen
572 set(handles.text_fa,'String',fa_string); %erzeugter String fa_string in textfeld text_fa schreiben
573 handles.fa = fa; %Simulationsfrequenz auf die handlesdatei speichern
574 guidata(hObject, handles);  %Aktualisieren der Handles Fensterstruktur
575
576 %% Funktion zur Initialisierung das Popupmenu zur Auswahl der Winkelauflösung
577 % --- Executes during object creation, after setting all properties.
578 function popupmenu_theta_a_CreateFcn(hObject, eventdata, handles)
579 % hObject    handle to popupmenu_theta_a (see GCBO)
580 % eventdata  reserved - to be defined in a future version of MATLAB
581 % handles    empty - handles not created until after all CreateFcns called
582
583 % Hint: popupmenu controls usually have a white background on Windows.
584 %       See ISPC and COMPUTER.
585 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
586     set(hObject,'BackgroundColor','white');
587 end
588
589 %% Callback-Funktion vom Popupmenu zur Auswahl des Öffnungswinkel
590 % --- Executes on selection change in popupmenu_theta.
591 function popupmenu_theta_Callback(hObject, eventdata, handles)
```

```
592 % hObject    handle to popupmenu_theta (see GCBO)
593 % eventdata  reserved - to be defined in a future version of
    MATLAB
594 % handles    structure with handles and user data (see GUIDATA)
595 Winkelaufloesung_cell = str2double(get(handles.popupmenu_theta_a,'
    String')); %Popupmenu für die Winkelauflösung auslesen
596 Oeffnungswinkel_cell = str2double(get(handles.popupmenu_theta,'
    String')); % Popupmenu für den Öffnungswinkel auslesen
597 Winkelaufloesung_auswahl = get(handles.popupmenu_theta_a,'Value');
     %Popupmenu für die Winkelauflösung auslesen
598 Oeffnungswinkel_auswahl = get(handles.popupmenu_theta,'Value'); %
    Popupmenu für den Öffnungswinkel auslesen
599 Winkelaufloesung = Winkelaufloesung_cell(Winkelaufloesung_auswahl)
    ;
600 Oeffnungswinkel = Oeffnungswinkel_cell(Oeffnungswinkel_auswahl);
601 fa = Simulationsfrequenz_berechnen(Winkelaufloesung,
    Oeffnungswinkel, handles.micarr); %Simulationsfrequenz in der
    Funktion "Simulationsfrequenz_berechnen" berechnen
602 fa_string = ['Simulationsfrequenz:␣', num2str(fa/1000), 'kHz']; %
    String erzeugen mit Simulationsfrequenz um die
    Simulationsfrequenz im Simulationsprogramm anzuzeigen
603 set(handles.text_fa,'String',fa_string); %erzeugter String
    fa_string in textfeld text_fa schreiben
604 handles.fa = fa; %Simulationsfrequenz auf die handlesdatei
    speichern
605 guidata(hObject, handles);  %Aktualisieren der Handles
    Fensterstruktur
606
607 %% Funktion zur Initialisierung das Popupmenu zur Auswahl des
    Öffnungswinkel
608 % --- Executes during object creation, after setting all
    properties.
609 function popupmenu_theta_CreateFcn(hObject, eventdata, handles)
610 % hObject    handle to popupmenu_theta (see GCBO)
611 % eventdata  reserved - to be defined in a future version of
    MATLAB
612 % handles    empty - handles not created until after all
    CreateFcns called
613
614 % Hint: popupmenu controls usually have a white background on
    Windows.
615 %       See ISPC and COMPUTER.
616 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
```

```
617     set(hObject,'BackgroundColor','white');
618 end
```

B. Graphische Oberfläche

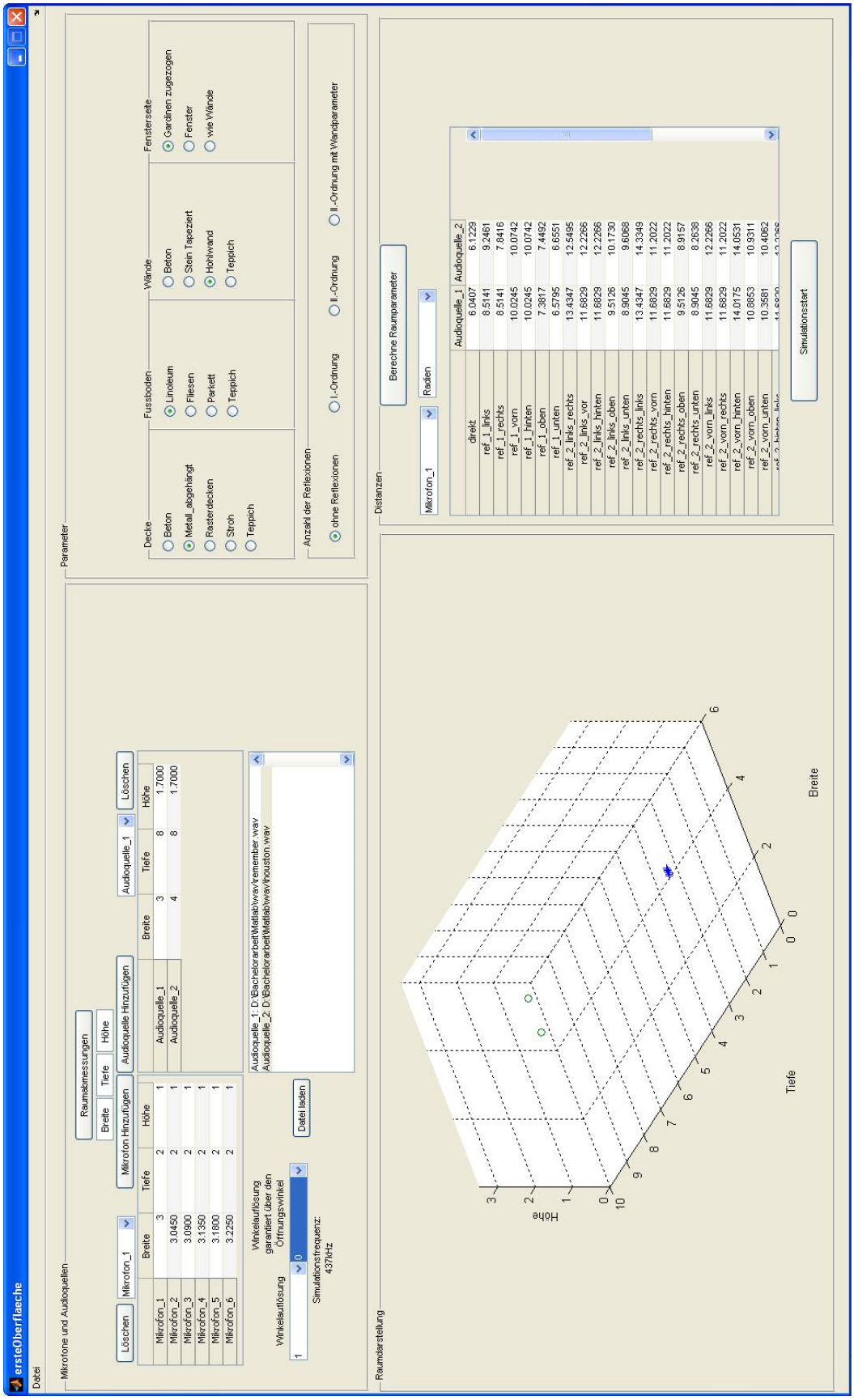


Abbildung B.1.: Graphische Oberfläche

C. Bedienungsanleitung

C.1. Programm starten

Zum starten des Programms führen Sie die Datei *ersteOberflaeche* unter MATLAB aus. Die Datei ist im Ordner *Matlab* auf der beigelegten CD enthalten. Achten Sie darauf, dass Sie vorher den Ordner **komplett** auf ein schreibfähiges Laufwerk kopieren, da die Ausgabedatein in den Unterordner *wav* erstellt werden.

Sollte sich die Datei nicht ausführen lassen überprüfen Sie Ihre MATLAB-Version, das Programm wurde mit MATLAB 7.8.0 erstellt.

C.2. Raumabmessungen und Koordinaten Eingabe

Zum Eingeben der Raumabmessungen und der Koordinaten der Audioquellen und Mikrofone werden die drei Editfelder, die oben in Abbildung C.1 dargestellt werden, verwendet. Bei Programmstart werden Raumabmessungen, Mikrofone und Audioquellen mit Standardwerten vorbelegt. Diese können über die Butten gelöscht werden. Die eingegebenen Abmessungen und Koordinaten können über die Menu Bar *Datei* gespeichert und geladen werden.

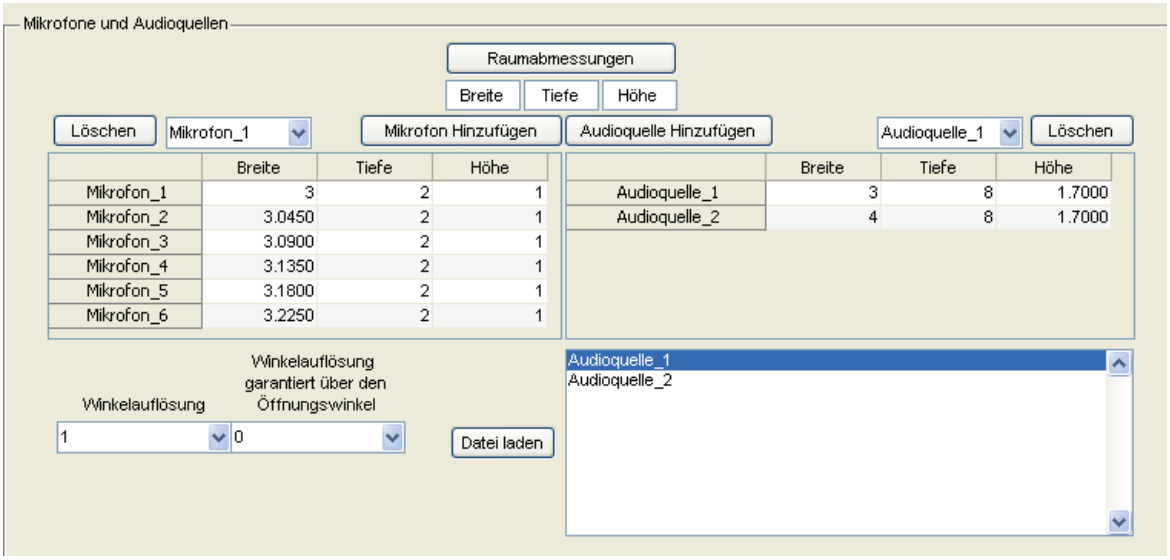


Abbildung C.1.: Raumabmessungen und Koordinaten Eingabe

In Abbildung C.2 wird der eingegebene Raum mit den Audioquellen und den Mikrofonen dargestellt. Die Audioquellen sind grün als Kreise gekennzeichnet und die Mikrofone blau als Kreuze. In dieser Darstellung sind die Mikrofone als Mikrofonarray dicht beieinander angeordnet, so dass sie sich optisch nicht auseinander halten lassen.

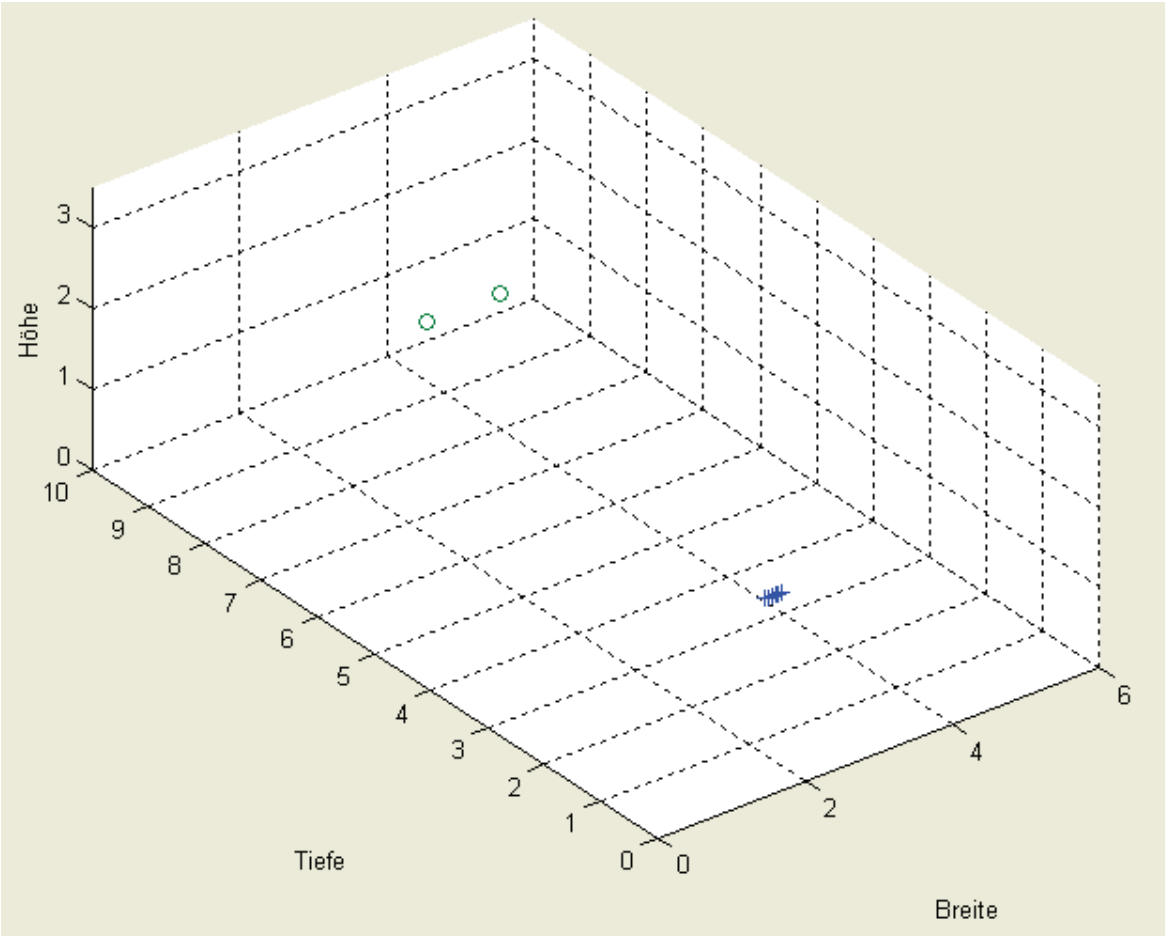


Abbildung C.2.: Darstellung des Raumes mit Audioquellen und Mikrofonarray

C.3. Winkelauflösung und Öffnungswinkel festlegen

Die gewünschte Winkelauflösung, die über einen gewünschten Öffungswinkel garantiert werden soll, kann über die beiden Popup Menus, die ebenfalls in Abbildung C.1 dargestellt sind, eingestellt werden. Werden keine Parameter eingestellt simuliert das Programm mit $100kHz$.

C.4. Audiodateien laden

Audiodateien können, wie ebenfalls in Abbildung C.1 unten links dargestellt, über einen Button *Datei laden* geladen werden. Bei Betätigung des Buttons wird ein Standarddialog zum

laden von Wave-Dateien gestartet. Achten Sie darauf, das für jede Audioquelle eine Audiodatei geladen wird. Die ausgewählten Dateipfade werden im Textfeld angezeigt.

C.5. Wandparameter einstellen

Die Wandparameter lassen sich, wie in Abbildung C.3 dargestellt, über *Radio Buttons* einstellen. Die hinter den Parametern implementierten Filter sind im Kapitel 7 aufgelistet. Die Wandparameter lassen sich auch im unteren Bereich der Abbildung C.3 abschalten. Ebenso lassen sich die Reflexionen I. und II.-Ordnung abschalten.

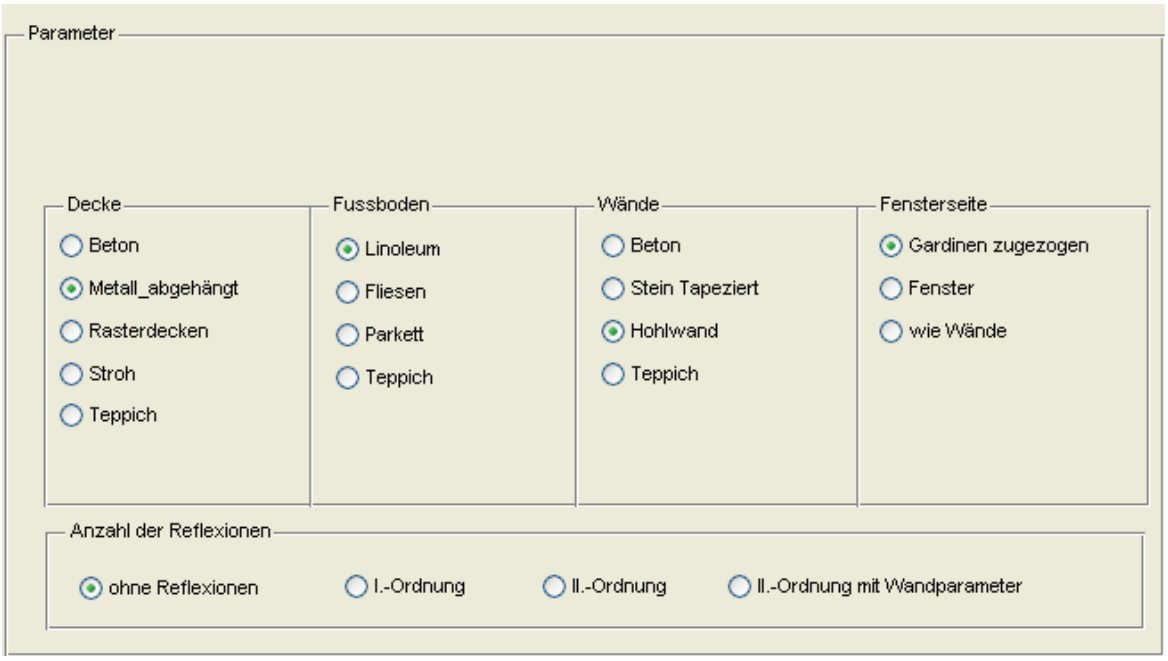


Abbildung C.3.: Eingabefeld für die Wandparameter

C.6. Start der Simulationsparameterberechnung

Die Simulationsparameterberechnung lässt sich über den Button *Berechne Raumparameter* in Abbildung C.4 starten.

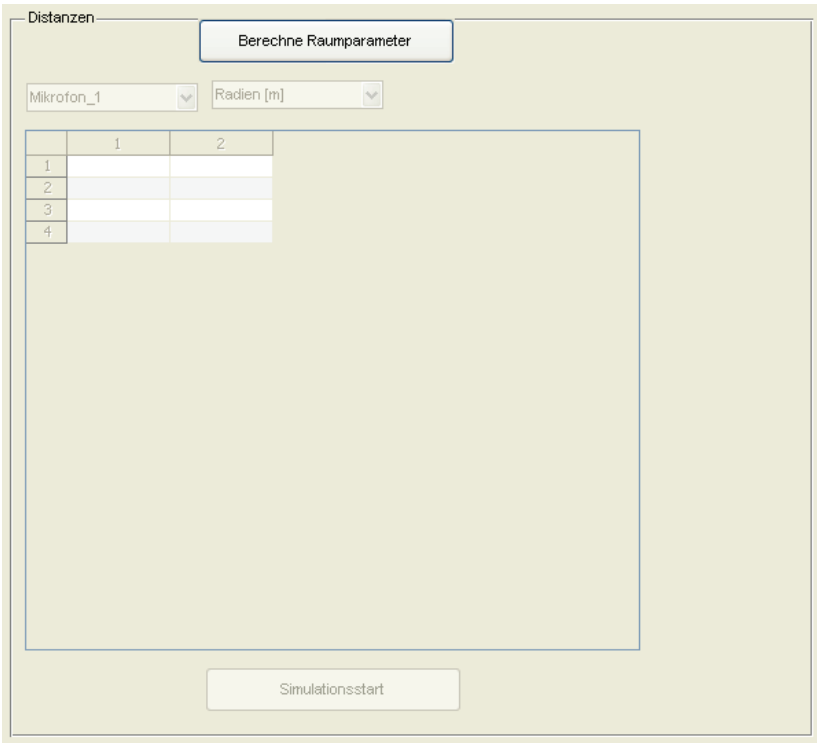


Abbildung C.4.: Starten der Simulationsparameterberechnung

Ist die Simulationsparameterberechnung abgeschlossen, wird folgendes Bild C.5 dargestellt.

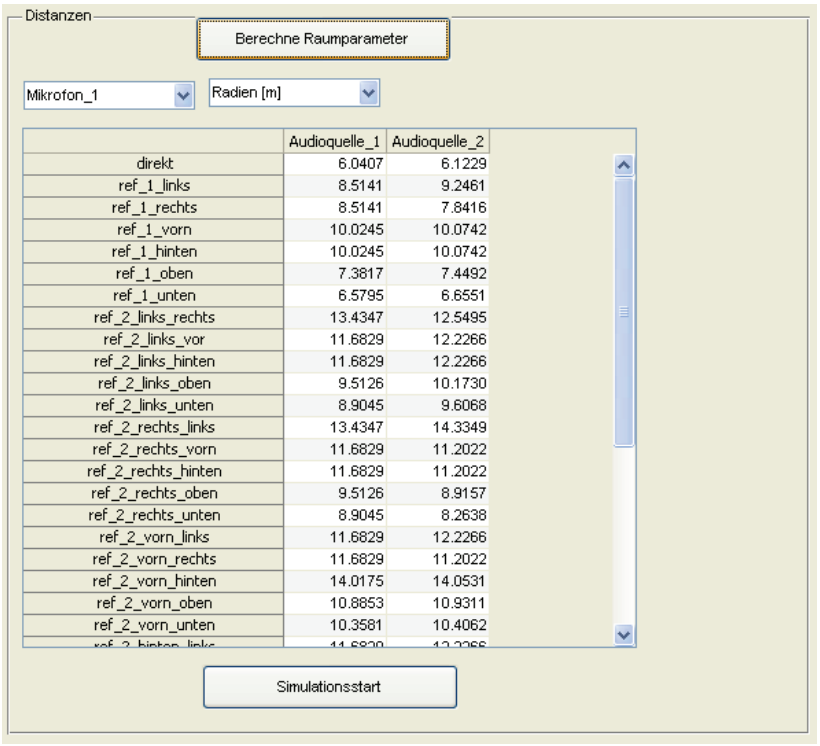


Abbildung C.5.: Berechnete Simulationsparameterberechnung

C.7. Simulationsstart

Nachdem die Simulationsparameter berechnet worden sind, kann die akustische Raumsimulation gestartet werden. Dies geschieht über den Button *Simulationsstart* ebenfalls in Abbildung C.5. Die Simulation wird eine gewisse Zeit in Anspruch nehmen. Den Fortschritt der Simulation kann man im MATLAB *Command Window* verfolgen. Ist die Simulation abgeschlossen wird im *Command Window* das erfolgreiche Abschließen der Simulation benachrichtigt. Die erzeugten Wave-Dateien sind im *Matlab* Unterordner *Wav* mit den Namen *out**1**.wav* bis *out**n**.wav* abgespeichert.

Versicherung über die Selbstständigkeit

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

Hamburg, 26. Februar 2010	
Ort, Datum	Unterschrift