Technische Universität Berlin

Fakultät I

Institut für Sprache und Kommunikation

Fachgebiet Audiokommunikation

Diplomarbeit

Implementierung einer mobilen Applikation für raumakustische Messungen

Prüfung und Betreuung:
Prof. Dr. Stefan Weinzierl
David Ackermann
Markus Hädrich

Autor:
Andreas Rosenkranz
[redacted]

Berlin, 6. August 2018

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Andreas Rosenkranz,

Berlin, 6. August 2018

Danksagung

An dieser Stelle möchte ich meinen Dank denen gegenüber aussprechen, deren Unterstützung und Motivation mir beim Verfassen dieser Arbeit eine große Hilfe waren.

Zunächst möchte ich mich bei Prof. Dr. Stefan Weinzierl bedanken, der diese Diplomarbeit betreut, begutachtet und begleitet hat.

Ebenfalls großer Dank gebührt den beiden fachlichen Betreuern Markus Hädrich und David Ackermann, die mir während der gesamten Bearbeitungszeit stets mit hilfreichen Ideen, Anregungen und Antworten zur Seite standen.

Von außerordentlicher Relevanz war die ausgesprochen gute Zusammenarbeit mit Ralf Burgmayer bei der Anfertigung dieser Arbeit, ohne dessen Signalverarbeitungsalgorithmen die in dieser Arbeit beschriebene Implementierung ihre hohe Genauigkeit nicht hätte erreichen können. Auch dafür möchte ich mich sehr herzlich bedanken.

Abschließend möchte ich mich aber vor allem bei meinen Eltern, sowie meiner Freundin für ihre Unterstützung und nie endende Geduld während der Erstellung dieser Arbeit und meines gesamten Studiums bedanken.

Andreas Rosenkranz,

Berlin, 6. August 2018

Zusammenfassung

A. Rosenkranz

Traditionell werden zur Bestimmung raumakustischer Parameter computerbasierte Software oder speziell dafür entwickelte Messgeräte herangezogen. Aufgrund der stetigen Weiterentwicklung mobiler Kommunikationsgeräte, mit immer leistungsfähigeren Prozessoren, grafischen Nutzeroberflächen und schnellem Netzwerkzugang, sind diese mobilen Geräte theoretisch sehr gut geeignet, raumakustische Messgrößen zu bestimmen. Dank der weiten Verbreitung und Verfügbarkeit könnte ihre Transformation in ein Messgerät zusätzliches Messwerkzeug für schnelle, spontane raumakustische Messungen überflüssig machen. Diese Arbeit stellt die Entwicklung einer plattformübergreifenden, leicht zu nutzenden, DIN-konformen und quelloffenen Applikation vor, die die Bestimmung raumakustischer Gegebenheiten und der Qualtität der Sprachübertragung in einem Raum ermöglicht.

Abstract

A. Rosenkranz

The assessment of room acoustical parameters is traditionally conducted with PC-based software or specifically designed measurement devices. Recently, however, mobile devices offer ever-increasing processing power, a graphical user interface and network connections. Thus, they are perfectly suited for the determination of room acoustical properties. Since these smart devices are also commonplace, turning them into measurement tools could make additional tools unnecessary for quick and spontaneous room acoustical assessments. This thesis presents a multi-platform, easy to use, ISO compliant open-source measurement system to assess room acoustical properties and quality of speech transmission in a room. The measurement system is implemented as an installable hybrid mobile application.

Inhaltsverzeichnis

Glossar

Matlab

kommerzielle Software von MathWorks zur Lösung und Visualisierung mathematischer Probleme 15, 23

Analog-Digital-Wandler

elektronisches Gerät zur Umwandlung analoger Eingangsdaten in eine Folge digitaler Daten 16, 55

Android

von Google entwickeltes Betriebssystem für Smart Devices wie Smartphones, Tablets und Fernsehgeräte diverser Hersteller vii, xvi, 26, 31, 32, 40, 44, 45, 47, 49–51, 54–58, 61–64, 66, 68–70

Angular

Erweiterung für JavaScript um Importfunktionen und Modularisierungsmöglichkeiten, entwickelt von Google 31, 32, 68, 69

Apple

Apple Inc., US-amerikanischer Hard- und Softwarehersteller und Dienstleister 43, 46

Benutzungserlebnis

auch Benutzungserfahrung, genderneutrale Formulierung für das Benutzererlebnis aus DIN 9241 28–30, 58

Chrome

von Google entwickelter Internetbrowser 40

CPU

Central Processing Unit, zentrale Prozessoreinheit eines Rechensystems 36–38

Dirac-Impuls

auch Einheitsimpuls, unendlich schmale Funktion mit Integralwert 1 5, 8

Engine

auch Rendering-Engine, Ebene einer Entwicklungsumgebung für Skriptsprachen, die für die Interpretation des Quellcodes verantwortlich ist 31

Framework

Entwicklungsumgebung xviii, 30–33

Google

US-amerikanischer Softwarehersteller und Webdienstleister 55

Hüllkurve

auch Einhüllende, Verlauf der Amplitude 10

Intersection Time

Zeitpunkt im Verlauf einer Impulsantwort, zu dem sich späte Abklingkurve und abgeschätztes Hintergrundrauschen schneiden xviii, 23, 24, 96, 97

iOS

von Apple Inc. entwickeltes Betriebssystem für Apple Inc. Smart Devices wie iPhone und iPad vii, xvi, 4, 15, 26, 31, 32, 40, 43, 45, 47, 49–51, 54, 58, 61–64, 66, 68, 69, 92

ITA Toolbox

MATLAB-basierte, quelloffen und frei verfügbare Umgebung des Instituts für Technische Akustik der RWTH Aachen zur Analyse von Audiodaten xviii, xix, 23–25, 96–104

iteratives Verfahren

Verfahren, das sich durch die Wiederholung von Arbeitsschritten mit zunehmender Genauigkeit einer Lösung annähert 22

Java

klassenbasierte, objektorientierte Programmiersprache von Sun Microsystems der Oracle Corporation, Basis der Android Entwicklung 26

JavaScript

Skriptsprache in der Webentwicklung 23, 27, 30–33, 68, 69

Kernel

zentrales Schnittstelle eines Betriebssystems zur Kommunikation von Hard- und Software 40, 41, 57

Kompilieren

Übersetzung von Quelltext in maschinenlesbaren Text 39

Linux

von Linus Torvald entwickeltes Betriebssystem, kommerziell und auch frei in vielen Varianten verfügbar 40, 41

macOS

von Apple Inc. entwickeltes Betriebssystem für Apple Inc. Computer 31, 40

nativ

in plattformspezifischer Programmiersprache 26–32, 45, 47, 48

Objective-C

objektorientierte Programmiersprache von Apple Inc., mögliche Basis der iOS Entwicklung 26, 69

Plugin

Zusatzmodul zur Erweiterung des Leistungsumfangs einer Entwicklungsumgebung 31, 69

Prozess

Ausführung eines Programmes vii, 35–41

Regressionsanalyse

hier: lineares Ausgleichsverfahren mit Hilfe der Methode der kleinsten Quadrate 22

Safari

von Apple Inc. entwickelter Internetbrowser 40

Samplerate

Abtastrate bei der Digitalisierung eines analogen Audiosignals 23

Sample

einzelnes Element eines digitalisierten Audiosignals einer gewissen Samplerate 23

Skript

hier: zusammengefasste Programmierlogik 40

Skriptsprache

Sprache, deren Skripte erst bei der Ausführung in Maschinensprache übersetzt werden, statt wie bei Programmiersprachen vor der Ausführung übersetzt vorzuliegen 27, 39, 69

Swift

klassenbasierte, objektorientierte Programmiersprache, mögliche Basis der iOS Entwicklung 26

Systemtheorie

Interdisziplinäre Theorie zur Modellierung und Beschreibung von Systemen 1, 5

T_{10}

Zeit, die nach abschalten einer Geräuschquelle vergeht, bis der Pegel in einem Raum um 10 dB abgefallen ist 1, 6, 22

T_{20}

Zeit, die nach abschalten einer Geräuschquelle vergeht, bis der Pegel in einem Raum um 20 dB abgefallen ist 1, 6, 22, 61, 66, 96, 112

T_{30}

Zeit, die nach abschalten einer Geräuschquelle vergeht, bis der Pegel in einem Raum um 30 dB abgefallen ist 1, 6, 22

Thread

Bearbeitungsstrang in der Verarbeitung von Arbeitsschritten eines Prozesses 31, 35, 40

TypeScript

sogenanntes „Superset“ für JavaScript, ergänzt JavaScript um Funktionen, Klassen und Methoden 31, 32, 68, 69, 79, 81

UI

User Interface oder Nutzeroberfläche eines bedienbaren Systems, im Kontext der mobilen Appikation ist stets eine grafische Nutzeroberfläche (engl.: Graphical User Interface, GUI) gemeint 31, 32, 42, 45

Web View

Container, der Internetseiten fassen und lokal mit Hilfe einer Rendering-Engine kompilieren kann 40, 68

Windows

von Microsoft entwickeltes Betriebssystem für Computer und Smart Devices wie Smartphones und Tablets 13, 31

Abkürzungsverzeichnis

$\varepsilon_{\mathsf{android}}$

Abweichung zwischen NTi XL2 und Android Gesamtsystem 104, 105, 107–109

$\mathsf{f_m}$

Mittenfrequenz des betrachteten Frequenzbandes 96–109

$\Delta\varepsilon$

Abweichung zwischen implementierten Algorithmen und implementierter Applikation 20, 98–100, 102, 103

$\varepsilon_{\mathsf{app}}$

Abweichung zwischen ITA Toolbox und implementierter Applikation 97–104

$\varepsilon_{\mathsf{imp}}$

Abweichung zwischen ITA Toolbox und implementierten Algorithmen 24, 25, 96–104

$\varepsilon_{\mathsf{ios}}$

Abweichung zwischen NTi XL2 und iOS System 104–109

$\varepsilon_{\mathsf{max}}$

maximale Abweichung 15

$\varepsilon_{\varnothing}$

durchschnittliche Abweichung 15

$\mathsf{IT_{imp}}$

Intersection Time, bestimmt mit eigenen implementierten Algorithmen 24, 96, 97

$\mathsf{IT_{ita}}$

Intersection Time, bestimmt mit der ITA Toolbox 24, 96, 97

$\mathsf{T_{xl2}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt mit dem NTi XL2 Messsystem 104–109

$\mathsf{T_{android}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt Android System 104–109

$\mathbf{T_{app}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt mit implementierter Applikation 97–104

$\mathbf{T_{imp}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt mit eigenen implementierten Algorithmen 25, 97–104

$\mathbf{T_{ios}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt mit iOS Gesamtsystem 104–109

$\mathbf{T_{ita}}$

Nachhallzeit aus Extrapolation nach T20-Methode, bestimmt mit der ITA Toolbox 25, 97–104

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

A. Rosenkranz

Die Raumakustik als Teilgebiet der Akustik bietet umfassende Möglichkeiten der Modellierung und Beschreibung von Schallereignissen in einem Raum. Als mächtiges Werkzeug der Systemtheorie bündelt die Impulsantwort in der Raumakustik alle Informationen, die die akustischen Eigenschaften eines Raumes als lineares, zeitinvariantes System beschreibbar machen. Auf ihrer Basis werden normgerechte Berechnungen ausgeführt, die im Kontext der Planung und Optimierung von Gebäuden und Räumen großen Einfluss auf deren akustische und psychoakustische Eigenschaften haben können. Je nach Nutzungszweck eines Raumes können infolge dieser Berechnungen Entscheidungen getroffen werden, die nachhaltigen Einfluss auf die Qualität der von Menschen in diesen Räumen verbrachten Zeit haben. Diese Diplomarbeit verfolgt das Ziel, eine Applikation zu entwickeln, die sich die Leistungsfähigkeit mobiler smarter Geräte zunutze macht, um die akustischen Eigenschaften eines Raumes zu bestimmen. Zu diesem Zweck soll die Impulsantwort eines Raumes messtechnisch erfasst und daraus für den Raum charakteristische akustische Kenngrößen abgeleitet werden. Die Berechnung der raumakustischen Kenngrößen soll auf Grundlage der in DIN 3382-1 (Deutsches Institut für Normung [1]), DIN 3382-2 (Deutsches Institut für Normung [2]) und DIN 18041 (Deutsches Institut für Normung [3]) festgehaltenen Berechnungsmethoden und Toleranzen realisiert werden. Nach Berechnung der Parameter sollen die Ergebnisse visuell präsentiert und nach Bedarf in Form eines Messprotokolls versendet werden können. Aufgrund des großen Gesamtumfangs, bietet dieses Thema ausreichend Inhalte für zwei separate Abschlussarbeiten. In dieser Arbeit stehen die Implementierung der Aufnahme und deren Auslösemechanismus, die nach DIN 3382-1 normgerechte Beschneidung der Impulsantwort, die Gestaltung der Benutzungsoberfläche sowie die Evaluation des Messsystems im Vordergrund. Für ausführliche Informationen zur Implementierung der Parameter und Filterbank sei auf die parallel entstandene Masterarbeit mit dem Titel „Implementierung einer mobilen Applikation zur Bestimmung raumakustischer Parameter“ von Ralf Burgmayer [58] verwiesen.

1.1. Zielsetzung

Die Raumimpulsantwort und aus ihr abgeleitete Parameter, die in der Applikation implementiert werden sollen, sind Grundlage des 2. Kapitels. Sie umfassen die Nachhallzeit auf Basis von T_{10}, T_{20} und T_{30}, die frühe Abklingzeit EDT, die Klarheits- und Deutlichkeitsmaße C_{50}, C_{80}, D_{50} und D_{80}, die Schwerpunktzeit T_S, das Bassverhältnis sowie den Sprachübertragungsindex STI. Die messtechnische Erfassung soll anhand einer aus impulsartiger Anregung gewonnenen Raumimpulsantwort erfolgen und wird in den Kapiteln 3 und 4 behandelt. Des Weiteren beschäftigt sich Kapitel

3 mit weiteren Formen der Anregung, deren Einbindung in die Applikation jedoch den Umfang der hier angestrebten Implementierung übersteigt. Für die leichte Bedienbarkeit der Applikation ist eine vollautomatische Endpunkteschneidung der Impulsantwort notwendig und ist Gegenstand des 5. Kapitels. Bei der Versendung der Ergebnisse als Messprotokoll wird ein Export mit Hilfe moderner Kommunikationswege angestrebt. Neben dem Ergebnisprotokoll als vorformatierte Textdatei, die leicht in Tabellenkalkulationsprogramme importiert werden kann, soll auch die aufgezeichnete Impulsantwort im WAVE-Dateiformat exportierbar sein. Zusätzliche Anforderungen an die Implementierung der Applikation sind, neben Plattformunabhängigkeit, eine quelloffene Struktur und eine frei verfügbare Veröffentlichung als erweiterbares Projekt. Diese Kriterien erfordern eine Beschränkung auf Hilfsmittel, die ebenfalls auf den genannten Entwicklungsanforderungen beruhen. So sollten nur Entwicklungsumgebungen in Betracht gezogen werden, die dem Kerngedanken der Quelloffenheit und freien Verfügbarkeit ebenfalls folgen. Auch eventuell aus fremden Quellen stammende Programmstrukturen oder Quelltexte müssen diese Kriterien erfüllen. Ein Überblick und eine Evaluation von Entwicklungswerkzeugen, die diese Anforderungen erfüllen, ist deshalb als gesonderter Teil dieser Arbeit in Kapitel 6 festgehalten. Weiterhin ist es wichtig, eine positive Erfahrung in der Interaktion mit der Applikation zu bieten. Um dies zu erreichen, muss die Applikation stets auf Eingaben reagieren und eine angenehme, zweckdienliche grafische Oberfläche bieten. Diese Problematiken werden in den Kapiteln 7 und 8 besprochen. Kapitel 9 und 10 sollen sicherstellen, dass die zu implementierende Applikation in Verbindung mit einem Smartphone und einem transportablen kleinen Messmikrofon ein Messsystem bilden, dass die Möglichkeit bietet, jederzeit mitgeführt zu werden.

1.2. Überblick über mobile raumakustische Messapplikationen

Die nachfolgende Tabelle 1.1 bietet einen Überblick über zur Zeit der Erstellung dieser Arbeit verfügbare mobile raumakustische Messapplikationen. Aufgelistet sind Applikationen, deren Leistungsumfang sich teilweise mit dem der geplanten Implementierung der Applikation deckt. Die in Tabelle 1.1 aufgelisteten Features decken in diesem Zusammenhang alle Funktionen der geplanten Implementierung ab. Die Mindestanforderung bei der Auswahl der Applikationen war dabei die Fähigkeit, Impulse im Raum über ein Mikrofon zu registrieren und daraus die Nachhallzeit des Raumes bestimmen zu können. Die Applikationen Vescom Acoustics Check (Vescom B.V. [63]) und Impulso (Artnovion LDA [64]) dienen der raumakustischen Messung zur anschließenden Anwendung von Verbesserungsmöglichkeiten, die dem jeweiligen Nutzungszweck des Raumes entsprechen. Beide Applikationen werden kostenfrei zur Verfügung gestellt und bieten nach der Messung direkte Auswahlmöglichkeiten von Absorbermaterialien der jeweiligen Hersteller. Der Funktionsumfang beider Applikationen beschränkt sich auf die Bestimmung der Nachhallzeit. Für Impulso ist mit Impulso Pro eine kostenpflichtige Version der Applikation verfügbar, die die zusätzliche Bestimmung der Parameter C_{50}, D_{80}, dem Bassverhältnis und dem Sprachübertragungsindex (siehe Kapitel 2) ermöglicht. Raumakustische Messungen sind mit diesen Applikationen prinzipiell möglich, ihre tatsächliche Nutzung als Messapplikation ist jedoch aufgrund der ent-

haltenen Produktempfehlungen in Frage zu stellen. Aus diesem Grund sind die beiden Applikationen nicht in der nachfolgenden Tabelle aufgeführt. Wie in Tabelle 1.1 er-

Tabelle 1.1.: Übersicht: Raumakustische Messapplikationen

	Room-Scope [65]	APM Tool [66]	Sound Balance Assistant [67]	ClapIR [68]	RevMeter Pro [69]	Nachhall-zeit (Pro[1]) [70],[71]
Features						
iOS	ja	ja	ja	ja	ja	nein
Android	nein	ja	nein	ja	nein	ja
Kostenfrei	nein	ja[1]	ja	ja	nein	nein
Quelloffen	nein	nein	nein	ja	nein	nein
Konform[4]	ja	ja	k.A.	k.A.	k.A.	k.A
Filterart	spektral	spektral	spektral	spektral	spektral	spektral
Breitband	ja	ja	ja	nein	nein	ja
Oktavfilter	ja	ja	ja	ja	ja	ja[2]
Terzfilter	ja	ja[2]	ja	ja	ja	ja[2]
Nachhallzeit	ja	ja	ja	ja	ja	ja
T_{10}	nein	nein	k.A.	ja	nein	k.A.
T_{20}	ja	ja	k.A.	nein	ja	k.A.
T_{30}	ja	nein	k.A.	nein	ja	k.A.
EDT	ja	ja[2]	nein	nein	nein	nein
C_{50}	ja	ja[2]	nein	nein	nein	nein
C_{80}	ja	ja[2]	nein	nein	nein	nein
D_{50}	ja	ja[2]	nein	nein	nein	nein
D_{80}	ja	ja[2]	nein	nein	nein	nein
Bassverhältnis	nein	nein	nein	nein	nein	nein
T_S	nein	nein	nein	nein	nein	nein
STI	nein	nein	nein	nein	nein	nein
Endpunkt-beschneidung	manuell	k.A.	k.A.	auto	k.A.	k.A.
Visualisierung	ja	ja	ja	ja	ja	ja[3]
Ergebnis-Export	ja	ja	ja	ja	ja	k.A.
Audioexport	ja	nein	nein	nein	nein	nein

sichtlich ist, erfüllt keine der verfügbaren Applikationen alle Anforderungen, die an die in dieser Arbeit vorgestellte Implementierung gestellt wurden. Der Großteil der Applikationen verzichtet auf die Implementierung einer kompletten Filterbank, wie sie durch Ralf Burgmayer [58] verwirklicht wird, und wertet die zu bestimmenden Parameter lediglich an den Stützstellen der jeweiligen Mittenfrequenzen der betrachteten Frequenzbänder aus. Ein Export aufgezeichneter Impulsantworten wird nur von RoomScope (Faber Acoustical, LLC [65]) ermöglicht. Bassverhältnis sowie Sprachübertragungsindex werden, abgesehen von der eingangs erwähnten Applikation Impulso, von keiner der Applikationen zu Berechnung angeboten. Die Bestimmung der Schwerpunktzeit (siehe Kapitel 2) gehört bei keiner der genannten Applikationen zum Funkti-

onsumfang. Die Applikation, die die größte Deckungsgleichheit mit den Funktionen der geplanten Implementierung aufweist, ist RoomScope. Dabei handelt es sich um eine kostenpflichtige Applikation mit umfangreichen Visualisierungs- und Exportmöglichkeiten, deren Anschaffung allerdings auch mit den mit Abstand höchsten Kosten (zur Zeit der Recherchen 109,99 €) verbunden ist. RoomScope ist nur für die iOS-Plattform verfügbar. Mit Ausnahme von ClapIR (Seetharaman und Tarzia [68]) sind die Quellen der aufgelisteten Applikationen nicht einsehbar, dadurch können keine Aussagen über die exakten Arbeitsweisen der Applikationen getroffen werden. Hinweise auf eine automatisierte Endpunktbeschneidung (siehe Kapitel 5) sind bis auf ClapIR in keiner der Applikationen zu finden. Während ClapIR den optimalen Schnittpunkt mit Hilfe einer Abstandsminimierung zwischen der Abklingkurve und der linearen Regressionslinie 100 ms nach deren letztem Schnittpunkt sucht [14], müssen in RoomScope die Grenzen zur Berechnung grafisch per Hand bestimmt werden (siehe [65]). Nach Betrachtung der verfügbaren mobilen Applikationen zur Ausführung raumakustischer Messungen lässt sich feststellen, dass bisher keine plattformübergreifende, quelloffene und kostenfrei zu verwende Applikation zu finden ist, die annähernd den geplanten Funktionsumfang der in dieser Arbeit vorgestellten Applikation erreicht.

[1] Für die Applikation ist eine kostenfreie Basis oder Lite Version mit beschränktem Funktionsumfang verfügbar.

[2] Diese Funktion wird ausschließlich von der kostenpflichtigen Version der Applikation unterstützt.

[3] Visualisierung und/oder Export beschränken sich auf die gemittelte Nachhallzeit als Einzahlwert und Grafiken zum Frequenzspektrum

[4] Normkonformität der Applikation zu DIN EN ISO 3382 laut Herstellerangabe.

2. Die Raumimpulsantwort und raumakustische Parameter

A. Rosenkranz

In der Systemtheorie können lineare, zeitinvariante Systeme (LTI Systeme) durch die sogenannte Stoßantwort oder Impulsantwort beschrieben werden. Dabei beinhaltet die Impulsantwort alle Reaktionen des betrachteten LTI Systems auf eine Anregung in Form eines Dirac-Impulses. DIN 3382-2 [2] beschreibt die Impulsantwort im raumakustischen Kontext als „zeitliche Entwicklung des Schalldrucks, der an einem Ort in einem Raum als Ergebnis der Emission eines Dirac-Impulses an einem anderen Ort im Raum beobachtet wird“. In diesem Kapitel soll die Bedeutung der Impulsantwort in der Raumakustik skizziert und die aus ihr abgeleiteten raumakustischen Parameter, die in der Applikation implementiert werden sollen, kurz erwähnt werden.

2.1. Die Raumimpulsantwort und ihre Bedeutung in der Raumakustik

In der Raumakustik beinhaltet die Raumimpulsantwort die Summe aller Reflexionen und Absorptionen von Schall, die in einem Raum nach Anregung durch ein Schallereignis stattfinden. Sie charakterisiert somit die akustischen Eigenschaften an einem Ort eines Raumes in seinem aktuellen Zustand. Verschiedene Methoden zur Gewinnung von Raumimpulsantworten werden im Kapitel 3.1 auf Seite 9 unter „Anregungssignale“ beschrieben.

2.2. Ableitung raumakustischer Parameter aus der Raumimpulsantwort

Aus dem durch die Raumimpulsantwort dargestellten Druckverlauf lassen sich verschiedene raumakustische Messgrößen ableiten. Als Grundlage der Berechnung dient die sogenannte Abklingkurve, die sich aus der Integration des Schalldruckquadrates ergibt. Eine kurze Übersicht der in der Applikation zu implementierenden Parameter bieten die nachfolgenden Abschnitte.

2.2.1. Nachhallzeit

Die Nachhallzeit (engl.: reverberation time) beschreibt die Zeit, die vergeht, bis der Pegel in einem Raum nach Abschalten eines Anregesignals um 60 dB abgefallen ist.

Oft besteht kein ausreichend hoher Abstand zwischen dem zu untersuchenden Signal und dem im Raum vorhandenen Störpegel. Deshalb haben sich Methoden etabliert, die Zeiten geringerer Pegelabfälle bestimmen und die Nachhallzeit daraus extrapolieren. Die bekanntesten Methoden heißen T_{10}-, T_{20}- und T_{30}-Methode und nehmen einen Pegelabfall um 10, 20, beziehungsweise 30 dB als Grundlage der Extrapolation. Dabei wird der Pegelabfall ab einem Punkt betrachtet, der 5 dB unter dem Maximum der Abklingkurve liegt.

2.2.2. Frühe Abklingzeit

Die frühe Abklingzeit (engl.: early decay time, EDT) ergibt sich ähnlich zur Nachhallzeit aus der Betrachtung eines Pegelabfalls im Verlauf der Abklingkurve. Der untersuchte Bereich der frühen Abklingzeit liegt dabei bei 0 dB bis −10 dB, beginnt also mit dem Maximum der Abklingkurve und endet 10 dB darunter. Während die Nachhallzeit ein genaueres Maß für die physikalischen Eigenschaften eines Raumes darstellt, bildet die frühe Abklingzeit die subjektive Empfindung des Nachhalls durch den Menschen besser ab.

2.2.3. Klarheits- und Deutlichkeitsmaße

Die Maße C_{50} und C_{80} beschreiben das Verhältnis zwischen früh und spät eintreffender Energie am Messpunkt. Dabei wird die früh eintreffende Energie als der Energieanteil der Abklingkurve betrachtet, der in den ersten 50 ms für C_{50} oder 80 ms für C_{80} eintrifft und mit der Energie des nachfolgenden Bereiches der Abklingkurve ins Verhältnis gesetzt. C_{80} wird dabei als Klarheitsmaß bezeichnet. Wird anstelle der verbleibenden Energie die gesamte Energie der Abklingkurve als Bezug gewählt, ergeben sich die Maße D_{50} und D_{80}. D_{50} wird als „Tonschärfe “ oder „Deutlichkeit “ bezeichnet. Die Maße C_{50}, C_{80} (engl.: clarity), D_{50} und D_{80} (engl.: definition) werden als Bedingung für Sprache und auch Musik herangezogen.

2.2.4. Schwerpunktzeit

Die Schwerpunktzeit (engl.: center time, T_S) ergibt sich als Verhältnis aus der Integration über das integrierte Produkt aus Zeit und Energie und dem Integral über die Energie als Quadrat des Schalldruckverlaufes. Sie beschreibt das Gleichgewicht zwischen Klarheit und Nachhall.

2.2.5. Bassverhältnis

Das Bassverhältnis (engl.: bass ratio) ergibt sich aus dem Verhältnis der Summe der Nachhallzeiten bei 125 Hz und 250 Hz zur Summe der Nachhallzeiten bei 500 Hz und 1000 Hz einer oktav- oder terzgefilterten Raumimpulsantwort. Es ist ein Maß für die Wärme des Klangbildes in einem Raum.

2.2.6. Sprachübertragungsindex

Der Sprachübertragungsindex (engl.: speech transmission index, STI) ist ein Maß zur Beschreibung der Übertragungsqualität von Sprache in einem Raum. Er kann aus den relativen Pegeln einer oktavgefilterten Impulsantwort bestimmt werden. Da die zu implementierende Applikation zur Berechnung relative Schalldruckpegel nutzt, können zusätzliche Maskierungseffekte, die die Verwendung absoluter Pegel voraussetzen, nicht bestimmt werden.

Eine genaue Beschreibung der Ableitung der genannten Parameter ist in DIN 3382-1 [1], für Nachhallzeiten in DIN-EN-ISO 3382-2 [2], für den Sprachübertragungsindex in DIN 60268-16 [4], für das Bassverhältnis in Ahnert und Tennhardt [7, Kap.5, S. 191] sowie in der Arbeit zur Parameterimplementierung von Ralf Burgmayer [58] zu finden.

3. Onset-Erkennung

A. Rosenkranz

Als Onset bezeichnet man allgemein das Einsetzen eines akustischen Ereignisses. Die automatische Onset-Erkennung (engl.: onset detection) soll ermöglichen, das Einsetzen eines solchen Ereignisses automatisch zu erkennen. Nachfolgend wird ein Überblick über Anregungssignale, die Anwendung in der Raumakustik finden, gegeben und anschließend auf die Implementierung in der Applikation eingegangen.

3.1. Anregungssignale

Da für die Implementierung der Applikation bereits im Voraus eine Beschränkung auf impulsartige Signale als Anregung vorgenommen wurde, bietet dieser Abschnitt einen kurzgefassten Überblick verschiedener Anregesignale (Stimuli) in der Akustik. Aufgrund des Anwendungsbereiches der Akustik in geschlossenen Räumen werden nachfolgend lediglich die gängigen Methoden zur Bestimmung von Raumimpulsantworten aufgezeigt. Für eine genauere Erläuterung technischer und mathematischer Hintergründe sei auf die Bücher „Messtechnik der Akustik“ [8, Kap. 3] und „Handbuch der Audiotechnik“ [9, Kap. 21] verwiesen.

3.1.1. Impulsartige Anregung

Bei der Verwendung von impulsartigen Quellen wie dem Schuss einer Pistole oder dem Platzen eines Ballons wird der zu untersuchende Raum durch ein zeitlich stark begrenztes Schallereignis angeregt. Die zeitlich starke Begrenzung entspricht dabei der Real-Approximation an einen Dirac-Impuls. Das Schallereignis und sein anschließendes Abklingen im Raum wird als Impulsantwort direkt aufgezeichnet und anschließend analysiert. Der Vorteil dieser Art der Anregung besteht in ihrer Einfachheit und damit guten Transportierbarkeit der Schallquellen. Es muss keine elektronische Schallquelle eingesetzt werden, um ein solches Signal zu erzeugen und somit wird auch die Nichtlinearität dieser Schallquellen umgangen. Nachteile sind die nicht-eindeutige Reproduzierbarkeit des Schallereignisses, sowie die geringe Schallenergie in niedrigen Frequenzbereichen.

3.1.2. Anregung durch Rauschen

Bei stochastischem breitbandigem Rauschen als Messsignal wird zunächst der zu untersuchende Raum stationär über einen Lautsprecher angeregt, um anschließend das Rauschen abrupt abzuschalten und den Abklingvorgang aufzuzeichnen. Vorteile der Anregung durch Rauschen sind das durch die stochastische Zusammensetzung des

Anregesignals resultierende abgedeckte breite Frequenzspektrum und die Reproduzierbarkeit bei der Verwendung sogenannten Pseudozufallsrauschens. Von Nachteil ist die psychoakustische Belastung für Messausführende und deren Umgebung durch Rauschsignale mit hohen Wiedergabepegeln über mehrere Messungen. Dies trifft insbesondere auf weißes Rauschen zu, dessen Pegel mit jeder Oktave weiter ansteigt und schon im niedrigen Frequenzbereich einen hohen Grundpegel benötigt, um ausreichend Abstand zu den Umgebungsgeräuschen aufzuweisen. Auch die Notwendigkeit von Audiotechnik zur Einspeisung des Rauschens in den Raum kann Probleme mit sich führen. So werden durch sie möglicherweise bestimmte Frequenzbereiche nichtlinear wiedergegeben, wenn ein ausreichender Energiepegel im Rauschen erreicht werden soll [9, Kap. 21, S. 1099]. Des Weiteren verteilt sich eine nichtlineare Wiedergabe des Rauschens durch die genutzte Audiotechnik über den gesamten Verlauf der aufgezeichneten Impulsantwort und verschlechtert somit das Signal-Rausch-Verhältnis (siehe Kapitel 5 im Abschnitt 5.1 auf Seite 23) [9, Kap. 21, S. 1122].

Sowohl die Anregung durch impulsartige Quellen als auch durch Rauschen setzt einen ausreichend großen Störabstand zwischen Signal und Hintergrundrauschen voraus, um die Nachhallzeit, sowie daraus abgeleitete raumakustische Maße zu bestimmen. Dieser Abstand ergibt sich bei beiden zuvor genannten Methoden allein aus der aufgezeichneten Raumimpulsantwort nach Abschalten der Anregung.

3.1.3. MLS-Technik

Bei der Nutzung von Maximallängenfolgen (engl.: Maximum Length Sequences, MLS) werden konstruierte Pseudozufallsfolgen als Anregesignal mit der Antwort des zu untersuchenden Systems korreliert. Dabei wird die Messfolge in das System eingespeist und dessen Antwort aufgezeichnet, anschließend wird deren Korrelationsfunktion berechnet und daraus die Impulsantwort bestimmt. Aufgrund spezieller Eigenschaften der Maximallängenfolgen vereinfacht sich die Korrelationsberechnung stark und die Impulsantwort kann mit Hilfe der sogenannten Hadamard-Transformation mit verringertem Aufwand auch auf weniger leistungsstarken Rechensystemen ausgeführt werden. Von Vorteil bei der MLS-Technik sind ein hoher Signal-Rausch-Abstand und gleiche Amplituden in allen enthaltenen Frequenzbereichen. Allerdings kann sich der schnelle Wechsel zwischen Maxima und Minima im Signalverlauf negativ auf die Linearität des Lautsprechers, durch den das Signal in das zu messende System eingespeist wird, auswirken [8, Kap. 3, S. 120-121], [9, Kap. 21, S. 1103-1108]. Bei vorhandenen nichtlinearen Anteilen im Signal treten ähnliche Probleme wie bei der Anregung durch Rauschen auf, da die Verzerrungen mit dem Signal überlagert sind [7, Kap. 21, S 1122].

3.1.4. Anregung durch Gleitsinus Signale

Gleitsinus Signale (Sweeps, Chirps) sind sinusförmige Signale, die in einem vordefinierten Bereich alle Frequenzen in einem bestimmten Zeitbereich nacheinander durchlaufen. Die mathematische Form, in der diese Frequenzen durchlaufen werden, bestimmen das Spektrum des Gleitsinus. Lineare, exponentielle, sowie auch logarithmische Erhöhungen der Frequenz sind üblich. Wird die obere Grenze des abgebildeten Frequenzbereiches erreicht, so beginnt das Gleitsinus Signal wieder bei der niedrigsten Frequenz.

Wie im Fall des Rauschens erfolgt die Anregung des zu untersuchenden Raumes durch Gleitsinus Signale mit Hilfe eines Lautsprechers. Die Messung mit Gleitsinus Signalen gehört zur sogenannten Klasse der Korrelationsverfahren und bietet den Vorteil, bei Messungen über einen längeren Zeitraum die Messdauer in höhere Pegel umrechnen zu können. Somit ist es möglich, das Messergebnis durch eine längere Messdauer zu verbessern. Akustische Störungen können somit durch Korrelationsmessverfahren unterdrückt werden, während die zuvor geschilderten Messverfahren durch Störungen verfälscht werden. Die Impulsantwort wird bei der Anregung durch Gleitsinus Signale durch anschließende Entfaltung (Dekonvolution) des aufgezeichneten Signals erhalten. Diese Entfaltung erfolgt aus Effizienzgründen häufig im Spektralbereich als sogenannte spektrale Entfaltung [8, Kap. 3, S. 122-123], [9, Kap. 21, S. 1114-1121].

3.2. Modellierung des Onsets

Um eine automatische Onset-Erkennung im jeweiligen Kontext zu ermöglichen, wird das akustische Ereignis meist in Form seiner Hüllkurve abstrahiert und mit Hilfe von Modellen wie dem ADSR-Modell in mehrere Phasen zerlegt. ADSR steht dabei für **A**ttack, **D**ecay, **S**ustain, **R**elease und unterteilt somit die Hüllkurve in 4 Phasen. Den Anstieg des Ereignisses, den anschließenden Abfall, das Halten über eine Zeitspanne und das Freigeben in Form des endgültigen Abklingens (siehe Abbildung 3.1). Diese Art von Modellbildung ist wichtig für die genaue Erkennung musikalischer Klangereignisse wie dem Einsetzen von Instrumenten innerhalb eines Orchesters oder sogar einzelner Noten in einem Musikstück. Für diese Modelle existieren viele mathematisch aufwendige Algorithmen, die auf den jeweiligen Nutzungszweck angepasst verwendet werden (siehe Bello et al. [15]). Im Zusammenhang mit der zu implementierenden Applikation vereinfacht sich die Onset-Erkennung aufgrund der zu erkennenden impulsartigen Anregung jedoch erheblich. Die Erkennung des Onsets selbst beschränkt sich im betrachteten Fall auf die einfache Entscheidung, ob ein Signal mit ausreichend hohem Pegel registriert wird.

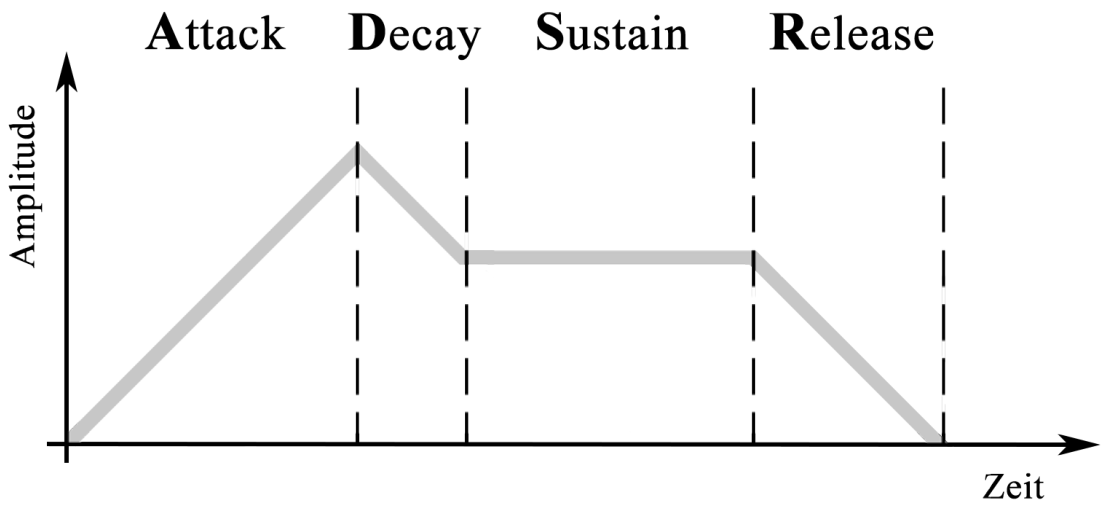


Abbildung 3.1.: ADSR-Modell zur Onset-Erkennung

3.3. Implementierung

Aufgrund der Beschränkung auf impulsartige Anregesignale bei der Definition des Leistungsumfangs der Applikation ist die Implementierung eines mathematisch aufwendigen Algorithmus zur Erkennung der Anregung nicht notwendig. Die Anforderung an die „Onset Detection“ der Applikation ist lediglich, das zu messende Signal zu erkennen und die Aufnahme auszulösen. Dieses Verhalten wird mit Hilfe einer Schätzung des Hintergrundrauschens beim Start eines Messvorgangs erreicht. Hierbei wird über einen vorher definierten Zeitraum das breitbandige Hintergrundrauschen des Raumes im nicht-angeregten Zustand aufgezeichnet und anschließend in einen Geräuschpegelverlauf umgerechnet. Dieser Pegelverlauf wird daraufhin über den vordefinierten Zeitraum gemittelt und als untere Grenze für eine Pegelüberschreitung gesetzt. Wird der gesetzte Grenzpegel im Anschluss von einem Signal im Raum überschritten, so startet der Aufnahmevorgang der Applikation.

4. Aufnahme der Audiodaten

A. Rosenkranz

Um eintreffende Audiodaten zum Zeitpunkt des in Abschnitt 3.3 auf Seite 12 beschriebenen Onsets aufzeichnen zu können, ist eine Zwischenspeicherung dieser Audiodaten notwendig. Üblicherweise erfolgt diese Zwischenspeicherung in einem sogenannten Buffer. Buffer dienen der temporären Aufbewahrung von Daten, deren Weiterverarbeitung nicht sofort geschehen kann oder soll. Im Kontext der zu implementierenden Applikation wird ein solcher Zwischenspeicher benötigt, um eine finite Menge nahezu in Echtzeit eintreffender Mikrofondaten vor dem Start eines Aufnahmevorgangs zu speichern. Dabei erfolgt die Beschränkung des Zwischenspeichers auf eine spezifische Länge (Bufferlänge), um zum Zeitpunkt einer gestarteten Aufnahme genügend Audiodaten gespeichert zu haben, die den Signalverlauf im Raum vor dem Zeitpunkt des Aufnahmestarts beschreiben können. Gleichzeitig soll die Bufferlänge gering gehalten werden, um auch den daraus resultierenden Speicherbedarf zu minimieren. In den nachfolgenden Abschnitten werden klassische Methoden zur Zwischenspeicherung von Daten sowie die Art der Implementierung in der Applikation vorgestellt. Anschließend wird die gewählte Implementierung evaluiert.

4.1. Buffermethoden

In diesem Abschnitt werden typische Varianten der Zwischenspeicherung von Daten besprochen, die in bedienbaren Systemen Anwendung finden.

4.1.1. First In, First Out

Bei der „First In, First Out“-Methode der Bufferung werden die Daten in der Reihenfolge abgespeichert, in der sie im Buffer eintreffen. Der Abruf der Daten zur weiteren Verarbeitung erfolgt von Bufferanfang zu Bufferende (Tanenbaum und Bos [10, Kap. 3, S. 211]). Typischerweise wird diese Art der Bufferung als Ringbuffer implementiert.

4.1.2. Last In, First Out

Wie bei der zuvor beschriebenen Methode werden bei „Last In, First Out“ die eintreffenden Daten in der Reihenfolge, in der sie eintreffen, im Buffer abgelegt. Ihr Abruf zur weiteren Verarbeitung erfolgt in entgegengesetzter Reihenfolge, so dass das zuletzt eingetroffene Element zuerst verarbeitet wird. Üblicherweise angewendet wird diese Methode beispielsweise bei der Speicherung aktueller Arbeitsschritte in Textverarbeitungsprogrammen, um den letzten Status der bearbeiteten Daten abrufbar zu haben,

falls es zu einem Absturz oder ungewollten Änderungen am Dokument kommt ([10, Kap. 3, S. 212]).

4.1.3. Cache

Beim „Cache“ werden Daten in einem Zwischenspeicher abgelegt, auf die ein häufiger Zugriff zu erwarten ist. Somit soll schnellere Verfügbarkeit von oft benötigten Daten gewährleistet werden. Typisch ist die Anwendung eines Caches in Internetbrowsern zur Zwischenspeicherung häufig besuchter Internetseiten. Auch in Betriebssystemen wie Windows werden Abbilder zuletzt verwendeter Daten im Cache zwischengespeichert [10, Kap. 11, S. 942-943].

4.1.4. Virtueller Speicher

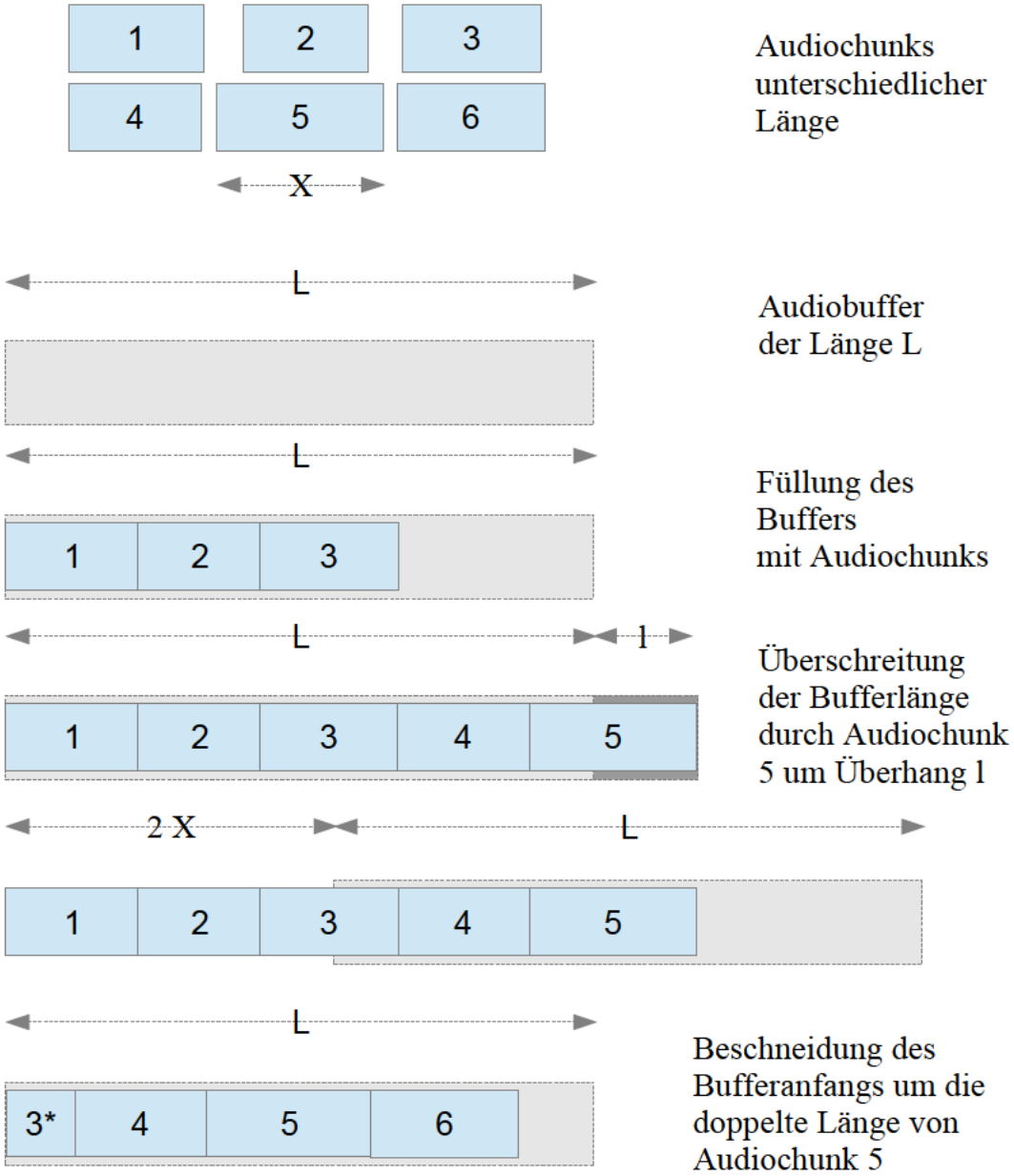
Virtuelle Speicher erlauben die Auslagerung aktuell nicht verwendeter Daten von einem schnellen Speichermedium in ein langsameres Speichermedium. Ein typisches Beispiel ist virtueller Arbeitsspeicher, der genutzt wird, um in einem Betriebssystem Daten aus dem tatsächlichen Arbeitsspeicher eines Computers in den wesentlich langsameren Festplattenspeicher auszulagern [10, Kap. 3, S. 194-195].

4.1.5. Zwischenablage

Die Zwischenablage ist ein aus Betriebssystemen bekannter Zwischenspeicher. Sie ermöglicht die systemweite Ablage eines einzelnen Datensatzes. Wird ein neuer Datensatz in die Zwischenablage geschrieben, so geht der zuvor gespeicherte Datensatz verloren [10, Kap. 5, S. 412].

4.2. Implementierung

Die Implementierung des Buffervorgangs wird durch einen nach der „First In, First Out“-Methode funktionierenden Ringbuffer definierbarer Länge erreicht. Der Buffervorgang läuft kontinuierlich sobald in der Applikation ein Messvorgang begonnen wurde und startet, sobald die Schätzung des Hintergrundrauschens abgeschlossen ist. Die Bufferlänge stellt sicher, dass nach Auslösen des Aufnahmevorgangs durch die implementierte Onset-Erkennung (Abschnitt 3.3, Seite 12) auch der Signalanteil in der Aufzeichnung beinhaltet ist, der unmittelbar vor der Erkennung des Onsets im zu untersuchenden Raum vorhanden war. Bei den vom Mikrofon im Smart Device eingehenden Daten handelt es sich nicht um einzelne Samples, sondern eine Menge an Samples in Form eines Samplepaketes (engl.: chunk). Daher ist die Implementierung eines „echten“ Ringbuffers, der nach Eintreffen des aktuellsten Samples das jeweils älteste Sample aus dem Buffer entfernt, nicht zweckmäßig. Stattdessen wird der Buffer so implementiert, dass er nach vollständiger Befüllung an seinem Anfang um die doppelte Länge des neu eingehenden Samplepaketes beschnitten wird, um anschließend neu eintreffende Samplepakete wieder am Ende des Buffers anfügen zu können. Eine schematische Darstellung des Buffervorgangs beginnend mit leerem Audiobuffer bis nach der ersten Beschneidung ist in Abbildung 4.1 zu sehen.



L – Bufferlänge
l – Bufferüberhang
X – Länge des überhängenden Audiochunks
3* – Audiochunk 3 nach Beschneidung

Abbildung 4.1.: Schematische Darstellung des Buffervorgangs

4.3. Evaluation

4.3.1. Einfluss der Aufnahme auf die Impulsantwort

Die Evaluation der Aufnahme erfolgt durch direktes Einspielen drei verschiedener Impulsantworten in ein iOS Gerät über ein vierpoliges 3,5 mm Klinken-Anschlusskabel. Anschließend werden Original und Aufnahme miteinander verglichen. Zum Vergleich werden beide Impulsantworten zunächst nach DIN 3382-1 [1] am Signalanfang beschnitten und die Maximalamplitude jeweils auf 1 normiert. Anschließend werden beide Impulsantworten am Ende auf gleiche Länge beschnitten. Die zur Evaluation genutzten Impulsantworten sowie MATLAB-Programme befinden sich im digitalen Anhang der Arbeit. Die Bezeichnungen der Impulsantworten lauten „Impulsantwort1.wav“ [21], nachfolgend „Impulsantwort 1“, „Impulsantwort2.wav“ [22], nachfolgend „Impulsantwort 2“ und „Impulsantwort3.wav“ [23], nachfolgend „Impulsantwort 3“. Beim Einspielen der Impulsantworten wurde zusätzlich darauf geachtet, alle beteiligten Systeme im Akkubetrieb, getrennt vom Stromnetz zu betreiben, um eine mögliche Überlagerung der aufzuzeichnenden Signale durch unerwünschte Schwingungen aufgrund der Netzspannung zu vermeiden. Tabellen 4.1, 4.2 und 4.3 zeigen die jeweiligen prozentualen

Tabelle 4.1.: Impulsantwort 1 - Abweichung zwischen Original und Aufzeichnung

Regressionsbereich	$\varepsilon_{\max}$ in %	$\varepsilon_{\varnothing}$ in %
Gesamte Impulsantwort	$17,48$	$0,05$
T10 (-5 bis -15 dB)	$5,57$	$0,8$
T20 (-5 bis -25 dB)	$5,57$	$0,49$
T30 (-5 bis -35 dB)	$5,57$	$0,33$

Tabelle 4.2.: Impulsantwort 2 - Abweichung zwischen Original und Aufzeichnung

Regressionsbereich	$\varepsilon_{\max}$ in %	$\varepsilon_{\varnothing}$ in %
Gesamte Impulsantwort	$20,95$	$0,01$
T10 (-5 bis -15 dB)	$5,6$	$0,63$
T20 (-5 bis -25 dB)	$5,6$	$0,4$
T30 (-5 bis -35 dB)	$5,6$	$0,17$

Tabelle 4.3.: Impulsantwort 3 - Abweichung zwischen Original und Aufzeichnung

Regressionsbereich	$\varepsilon_{\max}$ in %	$\varepsilon_{\varnothing}$ in %
Gesamte Impulsantwort	$21,8$	$0,03$
T10 (-5 bis -15 dB)	$3,65$	$0,64$
T20 (-5 bis -25 dB)	$3,65$	$0,46$
T30 (-5 bis -35 dB)	$3,65$	$0,36$

maximalen ($\varepsilon_{\max}$) und durchschnittlichen ($\varepsilon_{\varnothing}$) Abweichungen zwischen den Samples der originalen und der aufgezeichneten Impulsantwort. Abbildungen 4.2, 4.4 und 4.6

zeigen die grafische Überlagerung der eingespielten und originalen Impulsantworten. Dargestellt sind Überlagerungen der gesamten Signale sowie eine nähere Betrachtung der ersten 1000 Samples. In den Abbildungen 4.3, 4.5 und 4.7 sind die Differenzen der quadrierten Impulsantworten auf gleiche Weise wie ihre Überlagerungen dargestellt. Hier ist zu erkennen, dass die maximalen Abweichungen von etwa 22 % am Anfang der Signale zu verzeichnen sind und anschließend mit dem Signal abklingen. In den zur Bestimmung der Nachhallzeit relevanten Regressionsbereichen sind die maximalen Abweichungen bereits auf unter 6 % zurückgegangen und weisen ab Beginn der Regressionsbereiche mit einer durchschnittlichen Abweichung von unter 0,8 % insgesamt geringe Unterschiede zur originalen Impulsantwort auf. Diese Unterschiede liegen in der verwendeten Hardware in Form von Analog-Digital-Wandlern und den damit verbundenen Übertragungswegen des Audiosignals begründet und sind keine Folge der implementierten Algorithmen zur Aufzeichnung von Impulsantworten.

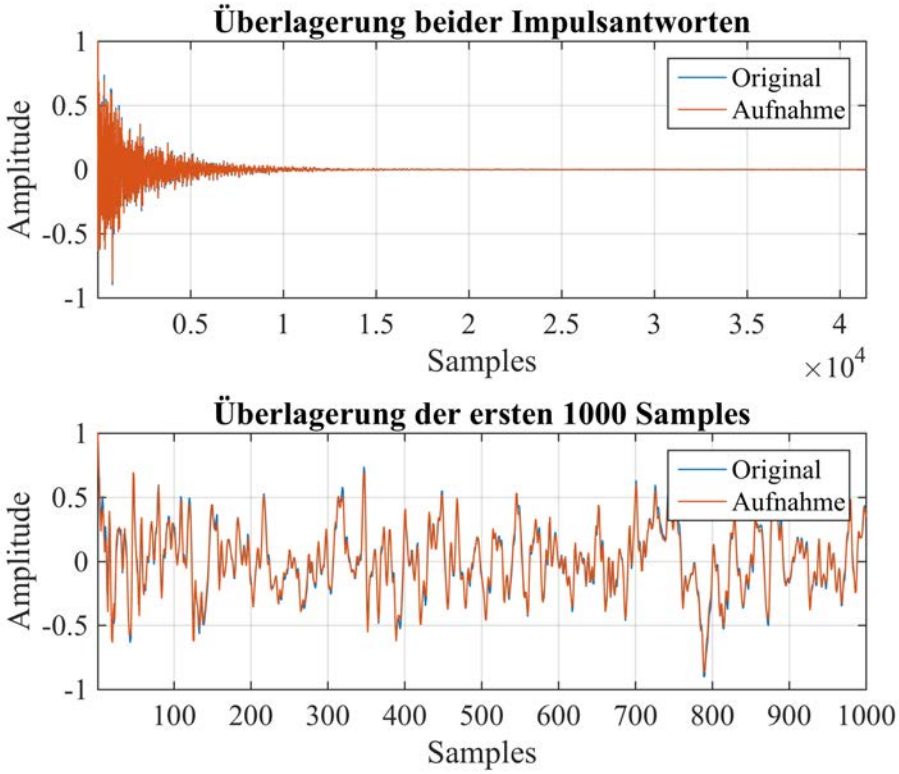


Abbildung 4.2.: Impulsantwort 1: Betrachtung der Impulsantworten

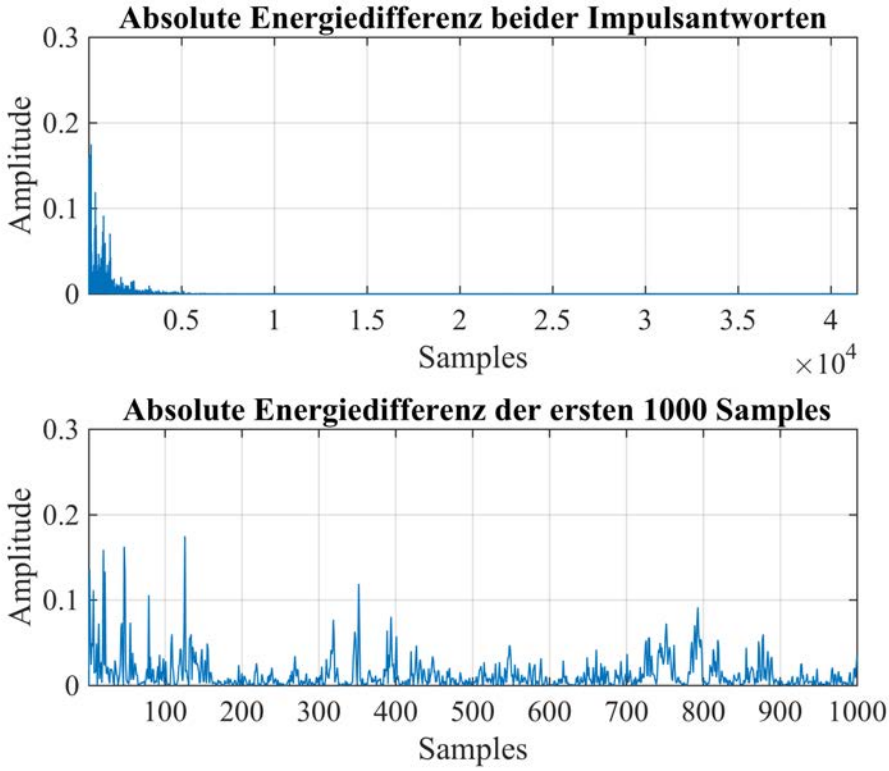


Abbildung 4.3.: Impulsantwort 1: Betrachtung des quadrierten Schalldruckverlaufes

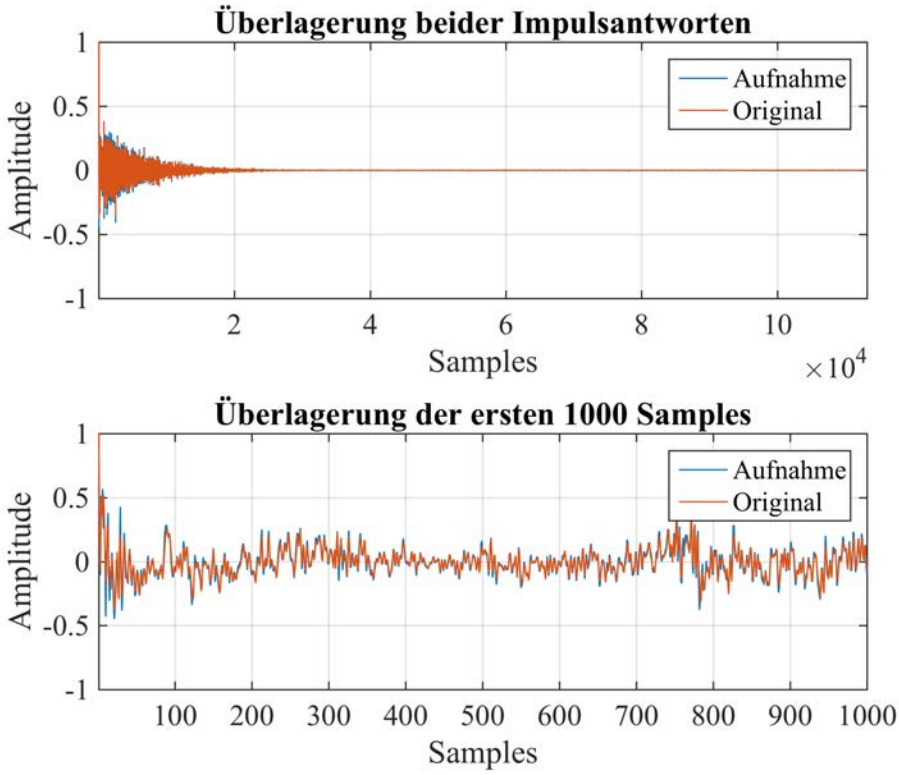


Abbildung 4.4.: Impulsantwort 2: Betrachtung der Impulsantworten

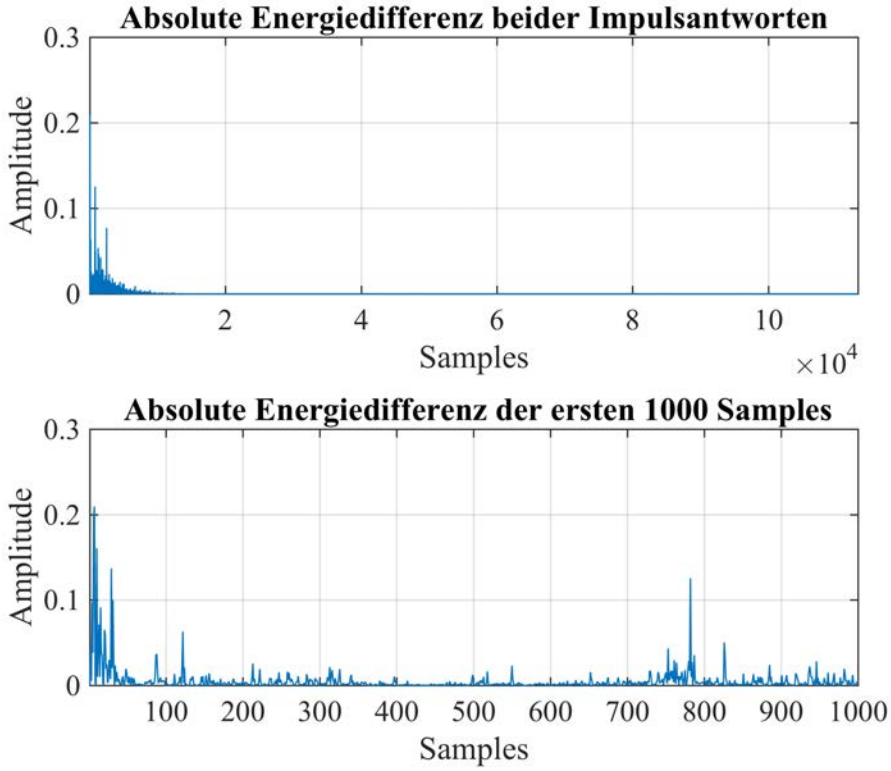


Abbildung 4.5.: Impulsantwort 2: Betrachtung des quadrierten Schalldruckverlaufes

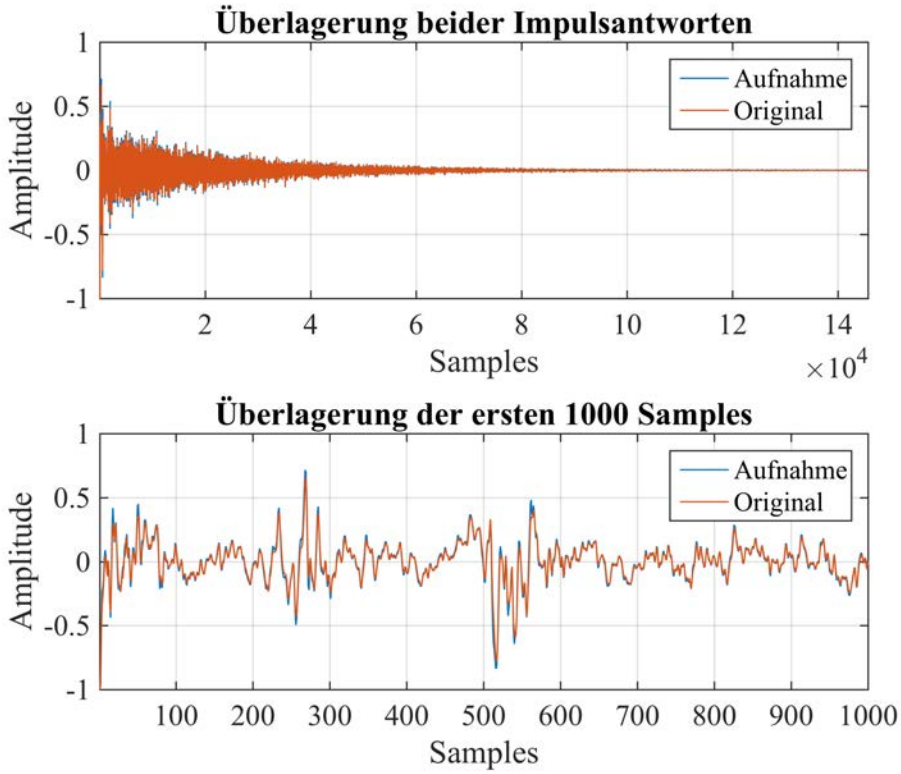


Abbildung 4.6.: Impulsantwort 3: Betrachtung der Impulsantworten

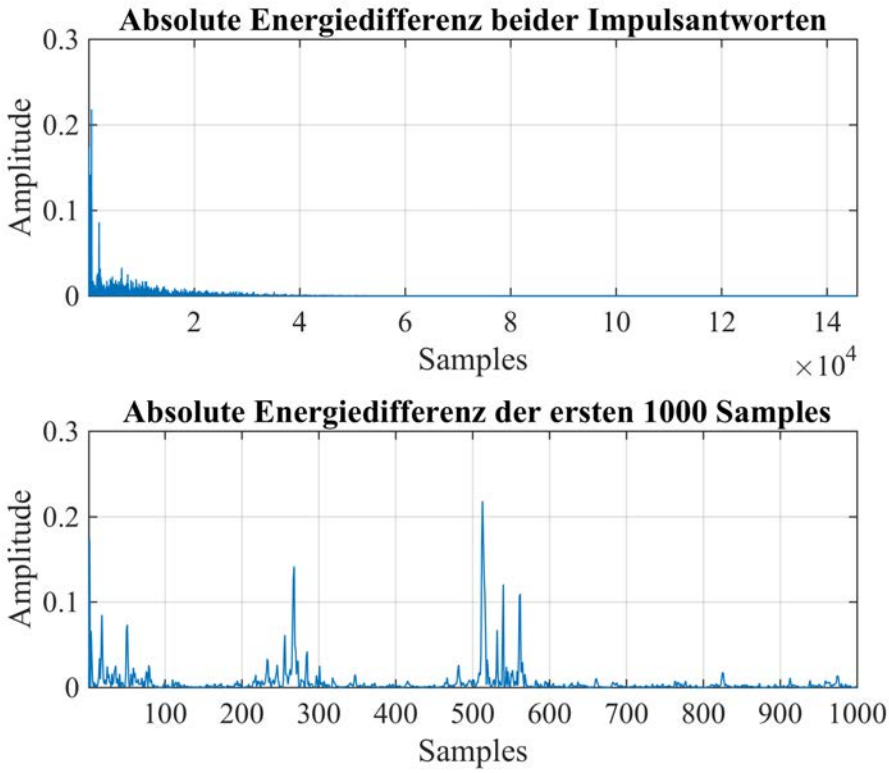


Abbildung 4.7.: Impulsantwort 3: Betrachtung des quadrierten Schalldruckverlaufes

4.3.2. Einfluss auf die Parameterberechnung

Abbildung 4.8 zeigt den Vergleich der berechneten Nachhallzeiten zwischen implementierten Algorithmen, berechnet aus der originalen Impulsantwort 1 und der Applikation selbst mit aufgenommener Impulsantwort 1. Beide Impulsantworten wurden wie zuvor am Anfang beschnitten und erhielten eine Endpunktbeschneidung mit Hilfe des im nachfolgenden Kapitel 5 beschriebenen Algorithmus. Aufgrund der Änderungen in der Impulsantwort in Folge der Aufnahme führen die implementierten Algorithmen in der Applikation zum Teil zu leicht veränderten Ergebnissen bei der Nachhallzeitbestimmung. Der maximale Einfluss der Aufnahmeabweichungen auf die Parameterberechnung am Beispiel der Nachhallzeit beträgt dabei im terzgefilterten Signal im Frequenzband von 250 Hz 2,82 %. Alle weiteren Abweichungen in der Nachhallzeitbestimmung sind in Anhang B.1.2 ab Seite 105 für die drei Impulsantworten mit Endpunktbeschneidung als $\Delta\varepsilon$ tabelliert.

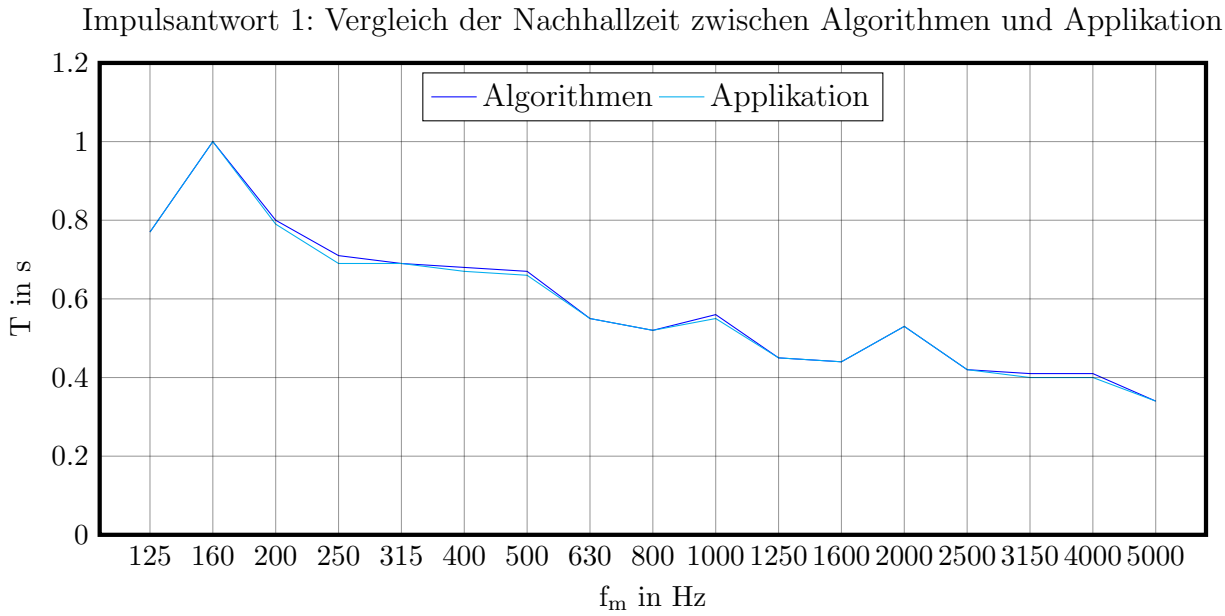


Abbildung 4.8.: Nachhallzeiten Impulsantwort 1, terzgefiltert

5. Automatisierte Endpunktbeschneidung

A. Rosenkranz

Zur korrekten Auswertung aufgezeichneter Impulsantworten ist es notwendig, zu erkennen, an welcher Stelle des Signalverlaufes das zu untersuchende Signal in das vorhandene Störsignal übergeht. Damit wird verhindert, dass vom Störsignal überlagerte Signalanteile eine Berechnung raumakustischer Parameter verfälschen. Eine solche Erkennung kann, mit ausreichender Erfahrung, per Hand erfolgen. Dieses Vorgehen ist jedoch für die Zielsetzung einer leicht zu bedienenden Applikation zur Bestimmung raumakustischer Parameter nicht zweckdienlich. Es existieren verschiedene Verfahren zur automatisierten Erkennung eines Schnittpunktes von Nutz- und Störsignal. Eine Übersicht der verschiedenen Ansätze ist im Artikel von Karjalainen et al. [16] gegeben. Dabei nutzen einige Verfahren, wie das von Ning Xiang, nichtlineare Näherungen an die Abklingkurve [16, S. 869] oder eine Subtraktion des Störsignals vom untersuchten Signal (Guski und Vorländer [17]). Sie versprechen genauere Ergebnisse und robustere Algorithmen in Extremfällen [16, S. 869-872]. Derartige Vorgehensweisen in der Bestimmung der optimalen Endpunktbeschneidung entsprechen jedoch nicht immer den in DIN 3382-1 [1] formulierten Forderungen [17]. Ein schneller und leicht zu implementierender iterativer Algorithmus, der diese Anforderungen erfüllt, soll nachfolgend kurz vorgestellt und seine Implementierung in der Applikation evaluiert werden.

5.1. Der Signal-Rausch-Abstand

5.1.1. Bedeutung des Signal-Rausch-Abstandes

Der Signal-Rausch-Abstand oder das Signal-Rausch-Verhältnis (engl.: Signal-to-Noise-Ratio) ist allgemein ein Maß für die Güte eines übertragenen Signals in der Video-, Bild- und Audioverarbeitung. Definiert als Verhältnis von der mittleren Leistung eines Nutzsignals zur mittleren Leistung des überlagernden Störsignals gibt es einen Anhaltspunkt für die Einschätzung der Verlässlichkeit des Informationsgehaltes des Nutzsignals. Je größer der Signal-Rausch-Abstand ist, desto größer ist die Wahrscheinlichkeit der Unabhängigkeit der übertragenen Informationen vom Hintergrundrauschen. In der Raumakustik spielt der Signal-Rausch-Abstand eine besonders große Rolle bei der Berechnung der Nachhallzeit aus Raumimpulsantworten, sowie den daraus abgeleiteten akustischen Maßen. So ist bei der Extrapolation der Nachhallzeit aus dem Pegelabfall um 10, 20 oder 30 dB ein jeweils 10 dB höherer Signal-Rausch-Abstand notwendig, um zur DIN 3382-1 [1] konforme Ergebnisse zu gewährleisten.

5.1.2. Implementierung

In der zu implementierenden Applikation wird die Bestimmung des Signal-Rausch-Abstandes in dem im nachfolgenden Abschnitt erläuterten Algorithmus von Lundeby et al. [18] enthalten sein. In diesem Algorithmus wird zunächst eine Abschätzung vorgenommen, indem die Energie der gesamten aufgezeichneten Impulsantwort mit der Energie der letzten zehn Prozent der Aufnahme ins Verhältnis gesetzt wird. Unter der Annahme, die Signallänge sei wesentlich größer als die zu bestimmende Nachhallzeit, kann davon ausgegangen werden, dass dieser geschätzte Signal-Rausch-Abstand größer als der tatsächliche Wert ist. Dieser Schätzwert wird daher als Orientierung genommen, um zu entscheiden, ob der nachfolgende Algorithmus ausgeführt werden soll. Erfüllt der geschätzte Signal-Rauschabstand die Anforderungen für eine auf der T_{10}-Methode basierende Bestimmung der Nachhallzeit von mindestens 25 dB, wird der Algorithmus ausgeführt. Nach erfolgreicher Iteration folgt eine neue Bestimmung des Signal-Rausch-Abstandes anhand der erhaltenen Ergebnisse, der anschließend als Mittel zur Prüfung der Verlässlichkeit der zu bestimmenden Nachhallzeiten herangezogen wird.

5.2. Lundeby Algorithmus

Der Algorithmus von Lundeby et al. [18] ist ein iteratives Verfahren zur Bestimmung des Schnittpunktes eines abklingenden Signals mit dem durch ein Störsignal gegebenen Hintergrundrauschen. Dieser Schnittpunkt wird anschließend genutzt, um das Signal zu beschneiden. Von diesem Schnittpunkt an erfolgt eine zu DIN 3382-1 [1] konforme Kompensation der Energie, die sonst durch das vorzeitige Abbrechen der Integration bei der Näherung der Nachhallzeit unberücksichtigt bliebe. Diese Kompensation korrigiert die sonst systematisch zu kurz berechneten Nachhallzeiten bei ihrer Abschätzung aus T_{10}, T_{20} oder T_{30}. Mit seiner iterativen Gestalt ist der implementierte Algorithmus von einer Vielzahl an Variablen und Hilfsalgorithmen abhängig. Algorithmen zur Glättung als auch zur anschließenden Regressionsanalyse der Impulsantwort führen zu einer vereinfachten Signalgestalt, an der eine Iteration der späten Abklingkurve und eine Abschätzung des Hintergrundrauschens stattfinden. Der Glättungsalgorithmus beeinflusst nicht nur durch die Art der Signalglättung den Kurvenverlauf, auf dem die Regressionsanalyse beruht. Auch der Glättungsbereich und die Startwerte, die der Glättung zugrunde liegen, haben dabei Einfluss auf das Ergebnis. Die Implementierung dieses Algorithmus wurde gewählt, da es sich hierbei um ein Verfahren handelt, das automatisiert ohne Eingreifen des Nutzers den optimalen Schnittpunkt zwischen Signal und Rauschen ermittelt. Zusätzlich ermöglicht es, Nachhallzeiten genauer anzugeben und liefert eine Bestimmung des Signal-Rausch-Abstandes als Nebenprodukt. Des Weiteren verringert eine Beschneidung der aufgezeichneten Impulsantworten den Zeitaufwand aller nachfolgenden Rechenoperationen.

5.2.1. Implementierung

Da es sich um ein iteratives Verfahren handelt, in dem lineare Regressionen genutzt werden, gibt es Fälle, in denen der Algorithmus fehlschlägt. Mit Hilfe der linearen Regression wird eine Linie geringsten Abstands durch den Verlauf der Abklingkurve der

aufgezeichneten Impulsantwort gelegt. Ein geringer Signal-Rausch-Abstand, Nichtlinearitäten im Signalverlauf oder energiereiche Reflexionen des Anregesignals können dabei zu zu steilen oder zu flachen Regressionslinien führen. Dieses Verhalten kann durch die weiteren Iterationsschritte des Algorithmus verstärkt werden. In Folge dessen ergeben sich durch den Algorithmus Schnittpunkte, die außerhalb der aufgezeichneten Signallänge liegen können. Für diese Fälle wurden Abfangmechanismen implementiert, die berechnete Ergebnisse auf Kausalität prüfen und sicherstellen, dass bei einem Fehlschlagen des Algorithmus das Signal unverändert bleibt. Eine Beschneidung der Impulsantwort findet somit stets am letzten zuverlässig iterierten Schnittpunkt statt und die letzte verlässliche Schätzung des Signal-Rausch-Abstandes wird an Stelle des iterierten Wertes als Orientierung gewählt. Schlägt der gesamte Algorithmus fehl, erfolgt keine Beschneidung des Signals und als Signal-Rausch-Abstand wird die erste Schätzung des Wertes herangezogen. Die Anpassungen sind im Anhang A.5 dargestellt. Für weitere Einzelheiten der Implementierung sei auf den im digitalen Anhang befindlichen kommentierten Quelltext verwiesen.

5.2.2. Evaluation

Zur Evaluation des implementierten Algorithmus zur Endpunktbeschneidung werden die drei in Kapitel 4.3 auf Seite 16 besprochenen Impulsantworten herangezogen. Die Evaluation erfolgt zunächst anhand eines Vergleiches der sich ergebenden „Intersection Time“ zwischen der ITA Toolbox [72] und dem in der Applikation implementierten Algorithmus. Anschließend wird der Einfluss des angewandten Verfahrens auf raumakustische Parameter anhand einer Betrachtung der Nachhallzeit aufgezeigt. Solche iterativen Algorithmen sind nicht direkt vergleichbar, da im speziellen Fall der Anwendung keine eindeutig korrekte Lösung existiert. Trotzdem werden nachfolgend die Ergebnisse der ITA Toolbox als Referenz gewählt. Die Wahl der Referenz folgt aus der langjährigen Erfahrung der RWTH Aachen mit dem beschriebenen Algorithmus. Alle Impulsantworten wurden zu Beginn der Berechnungen nach DIN 3382-1 [1] am Signalstart beschnitten. Die Anfangsbeschneidung des Signals erfolgte dabei an einer Stelle 20 dB unterhalb und zeitlich vor dem Signalmaximum. Um dies zu erreichen, erfolgt eine Indexsuche im aufgezeichneten Signal, beginnend am ersten Sample des Signals und endend am Signalmaximum. Die Indizes der Beschneidung sind in Tabelle 5.1 dargestellt. Die geringen Abweichungen in den Indizes sind auf die unterschiedliche Indexzählweise von MATLAB und JavaScript sowie die unterschiedlich implementierten Indexsuchfunktionen zurückzuführen. Unter Berücksichtigung beider Faktoren ergibt sich der maximale Abstand des bestimmten Indizes zur Anfangsbeschneidung bei den betrachteten Impulsantworten zu 4 Samples. Dies entspricht bei einer Samplerate von 44 100 Hz etwa dem Elf-Tausendstel einer Sekunde. Bei der „Intersection

Tabelle 5.1.: Vergleich der Anfangsbeschneidung zwischen ITA Toolbox und eigener Implementierung

Impulsantwort	ITA Toolbox Startindex	Implementierung Startindex
Impulsantwort 1	57	58
Impulsantwort 2	772	773
Impulsantwort 3	86	89

Tabelle 5.2.: Impulsantwort 1 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

Frequenzband in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	$0,751$	$0,754$	$0,40$
250	$0,835$	$0,789$	$5,51$
500	$0,756$	$0,791$	$4,63$
1000	$0,626$	$0,651$	$3,99$
2000	$0,577$	$0,592$	$4,60$
4000	$0,471$	$0,483$	$2,55$

Tabelle 5.3.: Impulsantwort 2 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

Frequenzband in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	$1,086$	$1,157$	$6,54$
250	$0,986$	$1,008$	$2,23$
500	$0,766$	$0,807$	$5,35$
1000	$0,851$	$0,885$	$4,00$
2000	$0,835$	$0,87$	$4,19$
4000	$0,753$	$0,763$	$1,33$

Tabelle 5.4.: Impulsantwort 3 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

Frequenzband in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	$2,671$	$2,62$	$1,91$
250	$2,798$	$2,677$	$4,32$
500	$2,945$	$2,755$	$6,45$
1000	$2,886$	$2,741$	$5,02$
2000	$2,915$	$2,728$	$6,42$
4000	$2,916$	$2,732$	$6,31$

Time“ sind zwischen beiden Implementierungen maximale Abweichungen von unter 7 % zu verzeichnen.[1] Tabellen 5.5, 5.6 und 5.7 zeigen beispielhaft die Nachhallzeiten der oktavgefilterten Impulsantwort 2 im Vergleich zwischen ITA Toolbox und den implementierten Algorithmen.
Die angegebenen Abweichungen beziehen sich auf die Differenz zwischen den Ergebnissen der ITA Toolbox und der Implementierung, wobei wie zuvor die ITA Toolbox als Referenz angenommen wurde. Ausführliche Ergebnistabellen zu allen drei Impulsantworten, breitbandig, oktav- sowie terzgefiltert, sind im Anhang B.1.2 zu finden. Alle Ergebnisse wurden auf die DIN-konforme Darstellung der ersten zwei Nachkommastellen gerundet. Kompensationen die keinen Einfluss auf diesen Bereich des Ergebnisses

[1] Abweichungen zur Arbeit von Ralf Burgmayer [58] beruhen auf Parameteränderungen zur besseren Vergleichbarkeit, die dort referenzierten Ergebnisse sind im Anhang B.1.1 zu finden.

Tabelle 5.5.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox und eigener Implementierung - ohne Endpunktbeschneidung

Frequenzband in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %
125	$1,27$	$1,26$	$0,79$
250	$1,31$	$1,31$	0
500	$1,41$	$1,41$	0
1000	$1,16$	$1,16$	0
2000	$1,11$	$1,11$	0
4000	$1,08$	$1,08$	0

Tabelle 5.6.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox und eigener Implementierung - mit Endpunktbeschneidung

Frequenzband in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %
125	$1,26$	$1,26$	0
250	$1,27$	$1,27$	0
500	$1,17$	$1,18$	$0,85$
1000	$1,10$	$1,11$	$0,91$
2000	$1,08$	$1,08$	0
4000	$1,05$	$1,05$	0

Tabelle 5.7.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox und eigener Implementierung - mit Endpunktbeschneidung und Kompensation

Frequenzband in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %
125	$1,26$	$1,26$	0
250	$1,27$	$1,27$	0
500	$1,18$	$1,19$	$0,85$
1000	$1,11$	$1,11$	0
2000	$1,08$	$1,08$	0
4000	$1,05$	$1,05$	0

haben, sind in dieser Form der Darstellung somit nicht zu erkennen. Die maximalen Abweichungen in den Nachhallzeiten der drei Impulsantworten nach Endpunktbeschneidung und Energiekompensation sind dabei 2,43 % und treten in der terzgefilterten Impulsantwort 3 im Frequenzband 125 Hz auf. Der Großteil der Abweichungen liegt zwischen 0 und 1 %. Unter Berücksichtigung der in Abschnitt 5.2 auf Seite 24 erwähnten Eigenschaften des Algorithmus und des beispielhaft gezeigten Einflusses auf die Nachhallzeit bietet die Implementierung des Algorithmus von Lundeby et al. eine verlässliche und genaue Möglichkeit, eine Endpunktbeschneidung der aufgenommenen Signale vor der Berechnung der akustischen Parameter auszuführen. Somit können erhebliche Zeiteinsparungen und ein Genauigkeitsgewinn bei der Parameterberechnung erzielt werden. Zusätzlich ermöglicht der Algorithmus eine Bestimmung der Nachhallzeit bei geringerem Signal-Rausch-Abstand, als in DIN 3382-1 [1] gefordert ist [18].

6. Auswahl einer Entwicklungsumgebung

R. Burgmayer, A. Rosenkranz

6.1. Ansätze zur App Entwicklung

Bevor mit der Implementierung der Applikation begonnen werden kann, muss entschieden werden, auf welche Weise die Applikation entwickelt werden soll. Verschiedene Ansätze zur Entwicklung mobiler Applikationen können verfolgt werden. Nachfolgend sollen die Vor- und Nachteile dieser Ansätze aufgezeigt werden.

6.1.1. Native Entwicklung

Die native Entwicklung mobiler Applikationen verfolgt den Ansatz, möglichst systemnah in der nativen Sprache der jeweiligen Plattform zu entwickeln. Für die Android-Entwicklung erfordert dieser Ansatz Kenntnisse in Java (siehe Google [24]), für die iOS Plattform sind Kenntnisse in Objective-C oder Swift (siehe Apple [25]) notwendig, um eine Applikation nativ implementieren zu können. Der Vorteil der nativen Herangehensweise liegt vor allem in der Entwicklung in einer Programmiersprache, die die jeweiligen Plattformhersteller für ihr System vorgesehen haben. Dies resultiert in einer optimierten Kommunikation zwischen Applikation und Hardware. Das Ergebnis sind schnelle Applikationen mit Zugriff auf alle von den Plattformherstellern zur Verfügung gestellten Programmierschnittstellen sowie systemweiten Dienste und plattformspezifisch optimierte Benutzungsoberflächen. Gegen native Entwicklung spricht ein höherer zeitlicher oder personeller Aufwand verschiedener Systeme und deren nativen Programmiersprachen, wenn mehrere Zielplattformen durch die Entwicklung fokussiert werden sollen. Die native Entwicklung von Applikationen wäre somit im Zusammenhang dieser Arbeit der beste Ansatz, wenn nur eine Zielplattform erreicht werden soll oder ein größeres Entwicklungsteam verfügbar wäre.

6.1.2. Plattformübergreifende Entwicklung

Eine plattformübergreifende Entwicklung mobiler Applikationen wird von einer Vielzahl an Entwicklungsumgebungen ermöglicht und verfolgt den Ansatz einer einheitlichen Programmierbasis für mehrere Plattformen. Die Implementierung einer Applikation erfolgt bei diesem Ansatz in einer einzigen Programmiersprache, die anschließend auf unterschiedliche Weise den jeweiligen Plattformen verständlich gemacht wird. Dabei unterscheidet man zwischen der Idee des sogenannten „Cross Compilers“, einem hybriden Ansatz der Applikationsentwicklung und dem webbasierten Ansatz.

Cross Compiler

Mit Hilfe eines „Cross Compilers“ wird die einheitliche Programmiersprache in die jeweils native Sprache der Plattformen übersetzt. Ein Nachteil dieser Herangehensweise ist, dass Quelltext häufig nicht direkt ineinander übersetzt werden kann und somit der Umfang des übersetzten Quelltextes oft den Umfang einer nativen Implementierung übersteigt. Eine größere Menge an Quelltext führt somit auch zu längeren Bearbeitungszeiten für die implementierte Applikation bei der Ausführung des übersetzten Quelltextes. Die Güte des Cross Compilers zeichnet sich somit dadurch ab, wie viel zusätzlicher Quelltext beim Übersetzungsvorgang produziert wird. Nach der Übersetzung in nativen Quellcode kann eine installierbare Applikation für die Endgeräte zur Verfügung gestellt werden (Linus Öberg [59, S. 7]). Ein Beispiel für eine Entwicklungsumgebung, die sich eines Cross Compilers bedient, ist Microsofts kostenpflichtiges Xamarin (Microsoft [73]).

Web Applikationen

Im Web implementierte Applikationen sind Applikationen, die mit den Mitteln moderner Webentwicklung programmiert werden. Sie sind nicht installierbar, sondern stehen in Form von Internetseiten zur Verfügung. Somit benötigen sie fast immer eine Internetverbindung und bieten sehr begrenzte Offline-Nutzbarkeit. Durch die Online-Verfügbarkeit der Applikation als Webseite entfällt die Notwendigkeit des Vertriebs über die plattformeigenen Vertriebswege. Der große Nachteil dieser Art der Implementierung ist die beschränkte Funktionalität einer Webseite ohne Zugriff auf plattformspezifische Hard- und Softwareschnittstellen [59, S. 8].

Hybrider Ansatz

Der Hybride Ansatz zur Entwicklung einer Applikation benutzt als einheitliche Basis oft JavaScript, eine Skriptsprache der Webentwicklung. Die in JavaScript implementierte Applikation wird von der verwendeten Entwicklungsumgebung in einen sogenannten „Web View“ verpackt, der die Interpretation des Quellcodes übernimmt. Gleichzeitig dient dieser „Web View“ als eine Art Container, um die Applikation auf der jeweiligen Plattform zu installieren. Damit stehen Applikationen, die mit einem hybriden Ansatz entwickelt wurden, die plattformspezifischen Vertriebswege zur Verfügung. Viele der hybriden Entwicklungsumgebungen bieten zudem verschiedene Möglichkeiten, mit der plattformspezifischen Hardware zu kommunizieren und erweitern somit den potenziellen Leistungsumfang einer hybrid implementierten Applikation im Vergleich zur reinen Web Applikation enorm. Eine Anpassung an plattformspezifische Regulierungen bezüglich der Benutzungsoberflächen von Applikationen gestaltet sich bei diesem Ansatz schwieriger als bei nativer Entwicklung [59, S. 9]. Einige bekannte Entwicklungsumgebungen, die dem hybriden Ansatz folgen, werden im Abschnitt 6.2.5 vorgestellt.

6.2. Auswahl einer Entwicklungsumgebung

6.2.1. Evaluationskriterien

Aufgrund der hohen Anzahl existierender Entwicklungswerkzeuge für mobile Applikationen ist eine Evaluation der Möglichkeiten im Hinblick auf die spezifischen Anforderungen an die Applikation notwendig. Heitkötter et al. [11] haben 14 Kriterien aus Experteninterviews, Literatur und Entwicklerforen im Internet zusammengetragen. Dabei wurde eine mobile Applikation als Web-, Hybrid- und native Anwendung implementiert und hinsichtlich dieser Kriterien evaluiert. Tabelle 6.1 nennt die Kriterien und deren Übersetzung in die deutsche Sprache. Für detaillierte Erläuterungen sei auf Heitkötter et al. [11] verwiesen.

Tabelle 6.1.: Kriterien nach Heitkötter et al.

Kriterium	Deutsche Entsprechung
License and Costs	Lizenz und Kosten
Supported Platforms	Unterstützte Plattformen
Access to advanced device-specific features	Hardwarezugriff
Long-term feasibility	Beständigkeit der Entwicklungsumgebung
Look and Feel	Benutzungserlebnis
Application Speed	Geschwindigkeit der Applikation
Distribution	Vertriebswege
Development Environment	Entwicklungsumgebung
GUI Design	GUI Design
Ease of Development	Schweregrad der Entwicklung
Maintainability	Instandhaltung der Applikation
Scalability	Skalierbarkeit
Opportunities for further Development	Erweiterungsmöglichkeiten
Speed and Cost of Development	Geschwindigkeit der Entwicklung

6.2.2. Spezielle Anforderungen an die Applikation

Die Ergebnisse der Arbeit von Heitkötter et al. [11] werden in Kombination mit den spezifischen Anforderungen an die Applikation zur Auswahl eines Entwicklungsansatzes genutzt. Die Anforderungen resultieren aus Vorgaben des Lehrstuhls, sowie für die Implementierung unbedingt nötigen Funktionalitäten. Die spezifischen Anforderungen lauten:

- Quelloffenheit, Lizenzfreiheit: Die Applikation soll als quelloffenes Projekt öffentlich zugänglich gemacht und erweitert werden können. Daher müssen alle verwendeten Softwarepakete und Bibliotheken kostenfrei verwendbar und frei einzusehen sein.

- Plattformunabhängigkeit: Die Applikation soll auf möglichst vielen Plattformen nutzbar sein.

- Möglichst geringer Aufwand der Entwicklung, Erweiterung und Instandhaltung: Hier wird vor allem der Programmieraufwand im Hinblick auf das Erreichen möglichst vieler Plattformen mit möglichst einheitlicher Code-Basis betrachtet, um Entwicklung, Erweiterung und Instandhaltung des Projektes am Lehrstuhl effizient zu gestalten.

- Hardwarezugriff: Zur Aufnahme von Impulsantworten ist der Zugriff auf das Mikrofon sowie die unbearbeiteten Audiodaten unabdingbar.

6.2.3. Vorauswahl der Entwicklungsumgebungen

Aufgrund der speziellen Anforderungen können einige Entwicklungsumgebungen schon im Vorfeld ausgeschlossen werden. Da die Applikation quelloffen sein, und möglichst viele Plattformen erreichen soll, scheidet die native Implementierung aus. Nativ müsste die Applikation für jedes Betriebssystem von Grund auf in verschiedenen Sprachen implementiert werden. Diese Möglichkeit wird ausgeschlossen, da die Implementierung der Applikation im zeitlich begrenzten Rahmen einer Abschlussarbeit durch nur einen Programmierer verwirklicht wird. Auch für die anschließende Instandhaltung und Erweiterung durch den Lehrstuhl bedeute dies einen erheblichen Mehraufwand. In diesem Sinne ist eine für alle Plattformen identische Code-Basis vorteilhaft. Wie in Heitkötter et al. [11] beschrieben, bietet die native Entwicklung hinsichtlich Geschwindigkeit und Benutzungserfahrung sowie einer guten Dokumentation viele Vorteile. Zur Bestimmung raumakustischer Parameter sind jedoch keine Berechnungen in Echtzeit notwendig. Die Vorteile, die eine native Entwicklung in Bezug auf die Geschwindigkeit der Applikation mit sich bringt, überwiegen somit für den betrachteten Anwendungszweck nicht die höhere Geschwindigkeit der Entwicklung plattformübergreifender Ansätze für ein kleines Entwicklerteam in zeitlich beschränktem Rahmen. Auch hinsichtlich des Benutzungserlebnisses und GUI Designs befriedigt ein solcher Entwicklungsansatz die Anforderungen von [11] vollkommen, da ein schlichtes, funktionelles Design mit diesen Mitteln ebenfalls erreicht werden kann. Aus diesen Gründen fällt die Entscheidung für die Implementierung der Applikation zugunsten eines plattformübergreifenden Entwicklungswerkzeugs.

Bei reinen webbasierten Entwicklungswerkzeugen wurde zum Zeitpunkt der Recherchen der Zugriff auf das Mikrofon von iOS Geräten aus Sicherheitsgründen noch untersagt (siehe Mozilla Developer Network [26]). Daher scheiden auch die reinen Webentwicklungsumgebungen aus. Da die meisten Cross Compiler, wie beispielsweise Microsofts Xamarin [73] kostenpflichtig sind, werden diese ebenfalls ausgeschlossen. In Anbetracht der zuvor getroffenen Aussagen bezüglich einer schnellen Entwicklung in kleinem Entwicklerteam für eine quelloffene und kostenfreie Applikation erfüllen die hybriden Ansätze Ionic, React Native und NativeScript alle gestellten Anforderungen.

6.2.4. Wichtung der Kriterien nach Relevanz

Die Auswahl einer geeigneten Entwicklungsumgebung erfolgt durch Vergabe von Punkten für die Qualität mit der die Kriterien erfüllt werden. Die Punkteskala reicht von 1 bis 5, je höher die erreichte Punktzahl, desto besser wird das Kriterium bewertet. Aus

den geschilderten Anforderungen lassen sich für die Kriterien nach Heitkötter et al. [11] unterschiedliche Relevanz für die zu implementierende Applikation ableiten. Aus diesem Grund werden die Kriterien nach Relevanz mit Multiplikatoren gewichtet. Die Gewichtung erfolgt in drei Kategorien:

- Hohe Relevanz (Multiplikator: 1,5 Punkte): Als sehr wichtig werden die Kriterien eingestuft, die den speziellen Anforderungen entsprechen oder den Entwicklungsprozess positiv beeinflussen werden.

- Relevanz (Multiplikator: 1,2 Punkte): Als relevante Kriterien werden Faktoren, die das Benutzungserlebnis positiv beeinflussen, definiert, wobei stilistische Feinheiten im Vergleich zu einer guten Bedienbarkeit eine untergeordnete Rolle spielen.

- Geringe Relevanz (Multiplikator: 1,0 Punkt): Als von geringer Relevanz werden Kriterien eingestuft, die weder mit den Vorgaben in Verbindung stehen, noch den Entwicklungsprozess erleichtern oder das Benutzungserlebnis verbessern.

Die Kriterien Lizenz und Kosten, sowie Hardwarezugriff werden unter der Annahme, dass alle verbleibenden Frameworks diese gleichermaßen erfüllen, aus der Evaluation ausgeschlossen. In Tabelle 6.3 sind den Kriterien ihre entsprechenden Wichtungsfaktoren zugeordnet.

6.2.5. Vorstellung der geeigneten Entwicklungsumgebungen

Die für die Implementierung in Frage kommenden Frameworks und deren Funktionsweise werden in diesem Abschnitt vorgestellt.

React Native

React Native ist eine quelloffene Entwicklungsumgebung zur plattformübergreifenden Entwicklung und folgt dem hybriden Entwicklungsansatz. React Native wurde von Facebook entwickelt und im Januar des Jahres 2015 veröffentlicht. Die Entwicklungsumgebung nutzt JavaScript als Basis-Sprache. Im Gegensatz zum klassischen hybriden Ansatz wird jedoch keine Webseite in einem Web View ausgeführt und als installierbare Applikation aufbereitet. Zwischen der Programmlogik und den nativen Programmierschnittstellen zur Steuerung der Geräte-Hardware und Elementen der Benutzungsoberfläche liegt lediglich eine zusätzliche Schicht, die für die Verbindung der Programmlogik mit nativen Funktionen und Objekten sorgt. Diese Schicht wird Bridge genannt. Aus dieser Struktur resultiert eine im Vergleich zu gewöhnlichen hybriden Architekturen verbesserte Verarbeitungsgeschwindigkeit. Auch nativer Programmcode kann in die Applikationen eingearbeitet werden. Um selbstdefinierte Komponenten zu entwerfen, müssen Kenntnisse in den nativen Programmiersprachen für iOS und Android Geräte vorhanden sein. Nähere Informationen können der Übersichtsseite der Entwicklungsumgebung von Facebook [27] entnommen werden.

NativeScript

NativeScript ist ebenfalls eine auf JavaScript basierende hybride quelloffene Entwicklungsumgebung. Entwickelt von Telerik zur plattformunabhängigen Entwicklung von iOS und Android Applikationen, wurde es im März des Jahres 2015 veröffentlicht. Als Basissprache kann wahlweise JavaScript oder TypeScript eingesetzt werden. NativeScript folgt wie React Native dem Prinzip des Zugriffs auf die nativen Funktionalitäten über eine zusätzliche Schicht zwischen Programmlogik und nativen Programmierschnittstellen. Eine Typumwandlung zwischen Programmlogik und nativen Programmierschnittstellen von JavaScript auf nativ verständlichen Quellcode der jeweiligen Plattformen ermöglicht dabei die Kommunikation. Im Gegensatz zur asynchronen Befehlsübermittlung in React Native werden die Befehle zur Erstellung und Änderung der Benutzungsoberfläche in NativeScript wie bei der nativen Applikationsentwicklung direkt im dafür verantwortlichen Bearbeitungsstrang (UI-Thread) ausgeführt. Diese und weitere Informationen sind auf der Entwicklerseite von NativeScript (Progress Software Corporation [28]) zu finden.

Apache Cordova und Ionic

Ionic ist eine hybride quelloffene Entwicklungsumgebung von Drifty Co., die im Jahr 2013 veröffentlicht wurde. Ionic verfolgt einen klassischen hybriden Entwicklungsansatz und beruht auf der Apache Cordova Engine. Somit sind alle in Apache Cordova bereits vorhandenen Plugins, die native Funktionalitäten ansprechen, in Ionic verwendbar. Apache Cordova ist die quelloffene Variante von Adobe Systems PhoneGap und ermöglicht die Umwandlung webbasierter Applikationen in ausführbare Applikationen für Plattformen wie Windows, macOS, Android, iOS und mehr. Ionic baut auf dieser von Cordova gebotenen Möglichkeit der Bündelung nativer Funktionalitäten mit webbasiertem Inhalt auf und erweitert das Konzept um TypeScript und Angular. Während TypeScript JavaScript um Klassen und Methoden erweitert, bietet Angular eine Importfunktion für einzelne Funktionen und Methoden eines Skripts und erspart somit das Laden gesamter Skripte, wie es in den momentan unterstützten Webstandards noch üblich ist. Ionic bietet demzufolge mit Hilfe von TypeScript, Angular, sowie nativer Plugins und der Bündelung und Übersetzung des Quellcodes für ein breites Spektrum an Plattformen die Möglichkeit, sowohl hybride als auch progressive, auf Web-Code basierende Applikationen zu erstellen. Dabei werden, ohne auf native Programmierung angewiesen zu sein, die Defizite eines plattformübergreifenden Ansatzes im Vergleich zur nativen Implementierung verringert. Eine Ausführliche Dokumentation der Entwicklungsumgebung kann auf der Webseite von Drifty Co. [29] gefunden werden.

6.2.6. Evaluation

Bei der Evaluation der verbleibenden Entwicklungswerkzeuge ist die Anzahl der unterstützten Plattformen von großer Bedeutung, hier zeigt Ionic die besten Voraussetzungen. Mit einer Codebasis lassen sich theoretisch mobile iOS- und Androidendgeräte und sämtliche gängigen mobilen Browser erreichen. Mit React Native und NativeScript lassen sich ebenfalls die großen mobilen Betriebssysteme iOS und Android erreichen. Im Hinblick auf die Beständigkeit der Frameworks in der Zukunft zeichnet sich ebenfalls Ionic aus. Große Firmen wie EA, Diesel und CAT nutzen die Umgebung zur

Erstellung mobiler Applikationen, was ein Fortbestehen wahrscheinlicher macht. Zur Abschätzung der Größe der aktiven Entwicklergemeinde werden deren Hauptanlaufstellen im Internet, GitHub (GitHub Inc. [30]) und Stack Overflow (Stack Exchange Inc. [31]) (siehe Tabelle 6.2), hinzugezogen.

Ionic hat laut einer Stichwortsuche und der Anzahl der Bewertungen auf GitHub die größte Entwicklergemeinde hinter sich. React Native zeigt hinsichtlich der Unterstützung durch Firmen und den Entwickler des Frameworks ebenfalls eine gute Zukunftsperspektive, jedoch ist die aktive Entwicklergemeinde etwas kleiner. NativeScript folgt bezüglich der aktiven Entwickler auf dem dritten Platz. Auch bei der Unterstützung durch große Firmen reiht sich NativeScript an dritter Stelle ein. Jedes der Entwicklungswerkzeuge in Tabelle 6.2 bietet einen Zugriff auf die Audiodaten des Mikrofons der mobilen Endgeräte.

In der Kategorie Look and Feel sind laut diverser Blogeinträge die Frameworks (siehe [32], [33], [34]) React Native und NativeScript gegenüber Ionic überlegen. Seit dem Update auf Ionic 2.0 wird der Abstand jedoch geringer, da eine Vielzahl an UI Elementen hinzugefügt wurde, die den Unterschied zu nativer Optik und Bedienbarkeit deutlich verringern [29]. Bezüglich der Geschwindigkeit der Applikationen sind React Native und NativeScript ebenfalls Ionic etwas überlegen. Das nativ-ähnlichere Verhalten und Aussehen bei React Native und NativeScript resultiert aus den zugrunde liegenden Architekturen der Frameworks [27], [28]. Bei der Qualität der Entwicklungsumgebungen sind alle Frameworks gleichwertig. Zur Erstellung der grafischen Oberfläche nutzt Ionic eigene Angular-Komponenten, die fortlaufend in Umfang und Funktionalität erweitert werden. Bei React Native und NativeScript werden native UI-Elemente angesprochen. Hinsichtlich der zukünftigen Erweiterung und Pflege der Applikation nach Abschluss dieser Arbeit eignet sich Ionic am besten. Hier sind lediglich Kenntnisse der Webentwicklung mit JavaScript beziehungsweise TypeScript und ein Verständnis für Angular notwendig. Dahingegen sind bei den anderen genannten Frameworks unter Umständen zusätzlich Kenntnisse in den nativen Programmiersprachen von Android und iOS und ihren Programmierschnittstellen Voraussetzung. Die Evaluation des Schweregrads der Entwicklung ist subjektiv. Hier scheint das Framework Ionic am besten dokumentiert zu sein.

Da bei Ionic der geringste Programmieraufwand vorliegt und sowohl Signalverarbeitung als auch grafische Oberfläche auf einer Code-Basis beruhen können, bietet das Framework die besten Voraussetzungen zur Instandhaltung und Erweiterung der Applikation. React Native erfordert bei der Implementierung für mehrere Plattformen einen erheblichen Mehraufwand bei der Programmierung. Der Hardwarezugriff erfordert oftmals Kenntnisse der Programmiersprachen der nativen Applikations-Entwicklung und die Verknüpfung von Logik und grafischer Oberfläche erscheint komplex und plattformspezifisch. Somit ist React Native kein auf einer einheitlichen Codebasis basierender Ansatz (siehe [35]). Auch unter NativeScript muss ein erheblicher Mehraufwand hinsichtlich der Benutzungsoberfläche pro zusätzlich zu erreichender Plattform betrieben werden.

Die Modularisierung des Projekts erfordert im Fall der zu implementierenden Ap-

Entwicklungsumgebung	Stack Overflow Einträge mit ≥ 3 Antworten / Bewertung ≥ 3	Interesse bei Git Hub (Sterne)
NativeScript	81 / 385	10400
Ionic	1620 / 4597	29900
React Native	862 / 3454	49300

Tabelle 6.2.: Einträge und Interesse an den Frameworks am 08.06.2017

plikation lediglich eine Trennung von Signalverarbeitung, Signalaufnahme und Interfacedesign – hier sind alle Frameworks gleichwertig. Auch bei der Erweiterbarkeit der Applikationen und Übersetzung in andere Frameworks im Falle einer Nichtfortführung bieten die Werkzeuge gleiche Möglichkeiten. Alle drei Frameworks nutzen JavaScript, somit könnte beispielsweise der Programmcode zur Berechnung der raumakustischen Parameter problemlos übertragen werden. Eine neue Implementierung der Benutzungsoberfläche hingegen ist bei Wechsel des Frameworks nicht zu umgehen. Ionic bietet den Vorteil, dass die grafische Oberfläche auch als gewöhnliche Webseite weiterverwendet werden kann. In der Abschätzung der Geschwindigkeit des Entwicklungsprozesses stellt sich Ionic wegen des geringsten Programmieraufwands und der umfangreichen interaktiven Dokumentation als geeignetste Entwicklungsumgebung dar.

Tabelle 6.3 fasst die Evaluationsergebnisse anhand der in Abschnitt 6.2.4 beschriebenen Punktevergabe zusammen. Das Framework, das die höchste Punktzahl erzielt, wird für die Entwicklung verwendet. Die in Klammern angegebenen Punktzahlen in Tabelle 6.3 stellen die vergebenen Punkte vor der Multiplikation mit dem Gewichtungsfaktor dar. Eine Gewichtung von 0 wurde auf die unabdingbaren Kriterien angewandt, die von allen betrachteten Entwicklungsumgebungen erfüllt werden.

Nach Auswertung der Evaluation fällt die Entscheidung zu Gunsten der hybriden Entwicklungsumgebung Ionic in Kombination mit dem Framework Apache Cordova. Das entscheidende Kriterium war, neben der besten Gesamtbewertung und des vergleichsweise geringeren Programmieraufwandes, die Beständigkeit der Entwicklungsumgebung. Die umfangreiche Dokumentation, regelmäßige Updates sowie die aktive Gemeinde nutzender Personen hinter Ionic, können sich als sehr positiv für die Implementierung erweisen.

Kriterium	Gewichtung	NativeScript	Ionic	React Native
Lizenz und Kosten	0	-	-	-
Hardwarezugriff	0	-	-	-
Beständigkeit der Entwicklungsumgebung	1,5	(2) 3	(4) 6	(3) 4,5
User Experience	1,2	(4) 4,8	(3) 3,6	(4) 4,8
Geschwindigkeit der Applikation	1,2	(4) 4,8	(3) 3,6	(4) 4,8
Vertriebswege	1,0	(3) 3	(4) 4	(4) 4
Qualität der Entwicklungsumgebung	1,5	(3) 4,5	(3) 4,5	(3) 4,5
GUI Design	1,2	(2) 2,4	(4) 4,8	(2) 2,4
Schweregrad der Entwicklung	1,5	(3) 4,5	(4) 6	(3) 4,5
Instandhaltung der Applikation	1,5	(3) 4,5	(4) 6	(3) 4,5
Skalierbarkeit	0	-	-	-
Erweiterungsmöglichkeiten	1,0	(4) 4	(5) 5	(4) 4
Geschwindigkeit des Entwicklungsprozesses	1,5	(3) 4,5	(4) 6	(3) 4,5
$\sum$	-	40	49,5	42,5

Tabelle 6.3.: Kriterien und Punktevergabe

7. Parallele Prozesse

A. Rosenkranz

Im Kontext von Betriebssystemen ist der Begriff „Prozess“ von großer Bedeutung und bezeichnet die Ausführung eines Programmes. Moderne Betriebssysteme sind in der Lage, eine Vielzahl von Prozessen gleichzeitig zu bearbeiten. Da eine positive Erfahrung bei der Benutzung essenziell für jede Applikation ist, müssen Prozesse in den Arbeitsschritten der Applikation parallelisiert werden, um stets eine responsive Umgebung zu bieten und zu keinem Zeitpunkt das Gefühl eines „eingefrorenen“ Systems zu vermitteln. Aus diesem Grund soll dieses Kapitel einen Überblick über die Gestalt paralleler Prozesse geben und darauf eingehen, wie sie im Bereich der Webentwicklung verarbeitet werden.
„Echte“ Parallelität von Prozessen kann nur erreicht werden, wenn ein Betriebssystem Zugriff auf mehrere Prozessoren eines Rechnersystems hat. Werden die nachfolgend erläuterten Vorgehensweisen auf ein System mit nur einem Prozessor oder Prozessorkern angewandt, handelt es sich streng betrachtet nur um „pseudo“-parallele Prozesse, da pro Prozessor immer nur ein einzelner Prozess zum gegebenen Zeitpunkt bearbeitet werden kann. Ein Prozess kann wiederum in einzelne Arbeitsschritte, sogenannte „Threads“ oder „Arbeitsstränge“ zerlegt werden. Nachfolgende Erläuterungen sind auch auf Threads anwendbar.

7.1. Prozessplanung

Abhängig von der Arbeitsweise eines Programmes unterscheiden sich die Anforderungen an die Parallelisierung der auszuführenden Prozesse. Als „Prozessplanung (engl.: scheduling)“ bezeichnet man dabei die Methode zur Planung der Abarbeitung dieser Prozesse. Die Prozessplanung gehört zum Kern jedes Betriebssystems. Die wichtigsten Unterschiede in der Arbeitsweise von Programmen bestehen zwischen Stapelverarbeitungssystemen, interaktiven Systemen und Echtzeit-Systemen. Die gemeinsamen Anforderungen an die Parallelisierung in den drei Systemformen sind Fairness (engl.: fairness), die Durchsetzung der Richtlinien (engl.: policy enforcement) und die Ausgeglichenheit des Systems (engl.: balance). Fairness beschreibt dabei die, im Kontext des Programmes, ausgewogene Zuweisung von Bearbeitungszeit im Prozessor des Systems. Die Durchsetzung der Richtlinien soll dabei sicherstellen, dass die für das System aufgestellten Regeln zur Prozessbearbeitung auch angewandt werden. Die Ausgeglichenheit des Systems sorgt dafür, dass alle Systemkomponenten möglichst gleichmäßig ausgelastet sind. Als primär betrachtete Systemkomponenten sollen Prozessor, Arbeitsspeicher und Speicher parallel Arbeitsschritte erledigen, ohne längere Zeit inaktiv zu sein. Nachfolgend soll ein Überblick über die drei erwähnten Arbeitsweisen, deren spezielle

Anforderungen und typische Ansätze zur Parallelisierung gegeben werden. Für detaillierte Ausführungen sei als Quelle für die folgenden Abschnitte auf „Modern Operating Systems“ (Tanenbaum und Bos [10, Kap. 2, S. 85-165]) verwiesen. Da es sich bei der zu implementierenden Applikation nicht um ein Echtzeit-System handelt, wird diese Klasse von Systemen nur kurz beschrieben.

7.1.1. Prozessplanung in Stapelverarbeitungssystemen

Stapelverarbeitungssysteme arbeiten die ihnen gegebenen Aufgaben ab, ohne dabei direkt mit der ausführenden Person eines Programmes interagieren zu müssen. In der Stapelverarbeitung sind die Kriterien Durchsatz (engl.: throughput), Verweildauer (engl.: turnaround time) und Prozessorausnutzung (engl.: CPU utilization) von besonderer Bedeutung. Mit „Durchsatz“ wird die Anzahl an erledigten Arbeitsaufträgen pro Stunde bezeichnet, ein hoher Durchsatz wird in der Stapelverarbeitung angestrebt. Die Verweildauer, also die Zeit zwischen dem Bearbeitungsbeginn und dem Abschluss eines Prozesses, soll minimiert werden. Dabei soll der Prozessor möglichst durchgehend ausgelastet sein [10, Kap. 2, S. 154].

First-Come, First-Served

„First-Come, First-Served“ ist die einfachste Art, in Stapelverarbeitungssystemen Prozesse zu planen. Wie der Name bereits andeutet, werden Prozesse in der Reihenfolge zur Abarbeitung an den Prozessor übergeben, in der sie eintreffen. Die zu bearbeitenden Prozesse werden dabei in eine Warteschlange aufgenommen und nacheinander ohne Unterbrechung bearbeitet. Vorteile dieser Methode sind die leichte Verständlichkeit und leichte Implementierung. Nachteilhaft ist die fehlende Unterscheidung in Prozessarten, wie Ein- und Ausgabeprozesse mit Speicherzugriff und CPU-intensive Prozesse, die einen geringen Durchsatz und hohe Verweildauern zur Folge haben kann [10, Kap. 2, S. 156-157].

Shortest Job First

Bei dieser Methode wird angenommen, dass die Laufzeiten der unterschiedlichen Prozesse im Voraus bekannt sind, um die Prozesse anschließend in Abhängigkeit ihrer Laufzeit abzuarbeiten. Am besten funktioniert diese Methode, wenn alle Arbeitsaufträge zur gleichen Zeit eintreffen, treffen sie zeitversetzt ein, können immer nur die Laufzeiten der zuerst eingetroffenen Aufträge geschätzt und bearbeitet werden [10, Kap. 2, S. 157-158].

Shortest Remaining Time Next

„Shortes Remaining Time Next“ wandelt die zuvor beschriebene „Shortes Job First“-Methode ab, indem immer der Prozess bearbeitet wird, dessen verbleibende Laufzeit am geringsten ist. Immer, wenn neue Prozesse zur Bearbeitung eintreffen, findet ein Vergleich der verbleibenden Bearbeitungszeit des aktuellen Prozesses mit dem neuen Prozess statt. Ist die Laufzeit des neuen Prozesses geringer, wird der aktuelle Prozess pausiert und der neue Prozess bearbeitet. Vorteilhaft ist die Bevorzugung neuer, schnell

abzuarbeitender Aufträge. Von Nachteil ist die Notwendigkeit, Laufzeiten der Prozesse im Voraus zu kennen [10, Kap. 2, S. 158].

7.1.2. Prozessplanung in interaktiven Systemen

Interaktive Systeme, also Systeme, die mit Menschen interagieren, fordern geringe Reaktionszeiten (engl.: response time) auf Anfragen durch Menschen und eine gute Proportionalität (engl.: proportionality) darin, wie sich die Zeit zur Erledigung von Arbeitsschritten mit dem subjektiven Eindruck des Menschen, wie lange ein solcher Arbeitsschritt dauern sollte, decken [10, Kap. 2, S. 154].

Round-Robin Scheduling

Die „Round-Robin“-Methode ordnet jedem Prozess ein zu ihm gehöriges Zeitintervall zu, ein sogenanntes „quantum“. Dieses Zeitintervall unterteilt die Laufzeit des Prozesses und erlaubt der CPU, den jeweiligen Prozess für die Dauer seines „quantums“ zu bearbeiten, bevor der Prozess pausiert und der nächste für die Dauer seines „quantums“ bearbeitet wird. Der pausierte Prozess wird an das Ende einer Warteschlange von Prozessen gesetzt und die Bearbeitung aller Prozesse läuft nach diesem Schema weiter. Die Länge des Zeitintervalls wird dabei im Voraus gewählt, ohne die tatsächliche Laufzeit eines Prozesses zu kennen, es besteht also die Möglichkeit, einem Prozess ein Zeitintervall zuzuordnen, das seine Laufzeit überschreitet und somit nach Beendigung eines Prozesses unnötige Wartezeiten entstehen. Demnach ist es bei dieser Methode wichtig, das „quantum“ auf einen Wert zu setzen, der eine gute Balance zwischen CPU-Auslastung und Reaktionszeit gewährleistet. Bei der einfachen „Round Robin“-Methode wird angenommen, dass alle Prozesse die gleiche Priorität besitzen [10, Kap. 2, S. 158-159].

Priority Scheduling

Bei der Methode des „Priority Scheduling“ wird jedem Prozess eine Priorität zugeordnet und der Prozess mit der höchsten Priorität wird zuerst bearbeitet. Anschließend gibt es verschiedene Möglichkeiten, mit der Prozessbearbeitung fortzufahren. So kann ein Algorithmus die Priorität des aktuell laufenden Prozesses während der Bearbeitung kontinuierlich herabsetzen, bis seine Priorität die eines anderen Prozesses unterschreitet. Der aktuelle Prozess wird dann pausiert und der neue Prozess bearbeitet. Auch die „Round-Robin“-Methode kann um eine Priorisierung von Prozessen erweitert werden [10, Kap. 2, S. 159-160].

Multiple Queues

„Multiple Queues “ beschreibt die Idee, Prozesse in Prioritätsklassen zu unterteilen und jeder Klasse ein Kontingent an Zeitintervallen zuzuordnen. Dieses Zeitintervall steht zur Bearbeitung der Prozesse in ihrer jeweiligen Klasse zur Verfügung, bevor sie pausiert und für die Bearbeitung im Prozessor gegen einen anderen Prozess ausgetauscht werden. Dabei steht einem Prozess höchster Priorität zunächst die geringste Anzahl an Zeitintervallen zu. Wenn sein Kontingent an Zeitintervallen aufgebraucht wurde, wird ein Prozess in die nächstniedrigere Prioritätsklasse versetzt. Anschließend erhält

er ein höheres Kontingent an Zeitintervallen für den nächsten Bearbeitungsdurchlauf [10, Kap. 2, S. 161].

Shortest Process Next

Um die Reaktionszeit gegenüber äußeren Eingaben zu minimieren, arbeitet „Shortest Process Next“ wie „Shortes Job First“, indem der Prozess mit der geringsten Laufzeit zuerst bearbeitet wird. Bei interaktiven Prozessen kann die Laufzeit der jeweiligen Prozesse, die durch die Eingabe zur Bearbeitung abgeschickt werden, nicht im Voraus bekannt sein. Aus diesem Grund werden Algorithmen implementiert, die zunächst die Laufzeit schätzen und anhand bereits abgearbeiteter Prozesse diese Schätzung optimieren [10, Kap. 2, S. 162].

Guaranteed Scheduling

Beim „Guaranteed Scheduling“ wird eine Regel zur Verteilung der Bearbeitungszeit (CPU-Zeit) für die jeweiligen Prozesse festgelegt. Im einfachsten Fall steht jedem Prozess die gleiche Bearbeitungszeit zur Verfügung. Ist ein neuer Prozess bereit, bearbeitet zu werden, wird ab diesem Zeitpunkt gemessen, wie viel Bearbeitungszeit er bereits in Anspruch genommen hat. Anschließend wird bestimmt, wie viel Bearbeitungszeit allen Prozessen zusteht, indem die Zeit, die seit der Prozesserstellung vergangen ist durch die Anzahl der Prozesse geteilt wird. Aus bereits in Anspruch genommener Menge an Bearbeitungszeit und Bearbeitungszeit, die einem Prozess zusteht, kann ein Verhältnis bestimmt werden. Dieses Verhältnis trifft eine Aussage darüber, welche Prozesse als nächste abzuarbeiten sind. Ein Verhältnis geringer als eins gibt dabei an, dass einem Prozess noch Bearbeitungszeit zusteht. Ein Verhältnis über eins sagt aus, dass der Prozess bereits mehr Bearbeitungszeit in Anspruch genommen hat, als er sollte. Dieses Verhältnis wird zur Priorisierung der Prozesse verwendet. Die Prozesse werden dementsprechend im Prozessor pausiert und durchgewechselt, dass möglichst ein Verhältnis von eins erhalten wird. Dabei entspricht das geringste Verhältnis der höchsten Priorität [10, Kap. 2, S. 162].

Lottery Scheduling

Diese Prozessplanungsmethode teilt jedem Prozess einen „Lottoschein“ zu und jedem dieser Scheine steht eine gewisse Bearbeitungszeit im Prozessor zu. Wenn eine Entscheidung getroffen werden muss, welcher Prozess als nächster vom Prozessor bearbeitet werden soll, wird zufällig einer dieser „Lottoscheine“ gezogen und der dazu passende Prozess bearbeitet. Diese Idee kann leicht um ein Priorisierungssystem erweitert werden, indem bestimmten Prozessen eine größeres Kontingent an „Lottoscheinen“ zugeordnet wird, dadurch wird die Methode des „Lottery Scheduling“ sehr flexibel [10, Kap. 2, S. 163].

Fair-Share Scheduling

„Fair-Share Scheduling“ ist ein Verfahren, das relevant für die Prozessplanung von Mehrnutzer-Systemen ist. Dabei werden Ressourcen wie die Bearbeitungszeit im Prozessor jedem Nutzer und auch jeder Nutzerin nach vordefinierten Regeln zugewiesen,

so dass ihnen, unabhängig von der Anzahl der durch sie gestarteten Prozesse, ihr angemessener Anteil an Ressourcen zur Verfügung steht. Diese Methode erweitert alle zuvor beschriebenen Methoden um die Möglichkeit, ein System von mehreren Nutzern und Nutzerinnen gleichzeitig bedienbar zu machen. Dabei ist anzumerken, dass in den meisten modernen Betriebssystemen sowohl das System selbst als auch tatsächlich Nutzende des Computers oder Smart Devices als separate Nutzende im Sinne der Prozessplanung anzusehen sind. Somit findet das „Fair Share"-Modell auch in Systemen Anwendung, die von nur einem „realen" Nutzer beziehungsweise einer „realen" Nutzerin bedient werden [10, Kap. 2, S. 163-164].

7.1.3. Prozessplanung in Echtzeit-Systemen

Systeme, deren Arbeitsschritte in Echtzeit verarbeitet werden sollen, haben besondere Ansprüche an die Prozessplanung. Bei der Abarbeitung der Prozesse müssen Fristen eingehalten werden (engl.: meeting deadlines), um Datenverlust zu vermeiden. Die Berechenbarkeit (engl.: predictability) der zur Bearbeitung benötigten Ressourcen und Zeiten muss zusätzlich gewährleistet sein [10, Kap. 2, S. 154]. So soll die Berechenbarkeit beispielsweise bei Videostreams absichern, dass keine Qualitätsverluste wie Asynchronitäten zwischen Bild und Ton stattfinden. Echtzeit-Systeme werden in „hard real time" und „soft real time" unterteilt, wobei im Fall „harter" Echtzeitbedingungen jede Abarbeitungsfrist als absolut anzusehen ist. Im Fall „weicher" Echtzeitbedingungen kann jedoch gelegentliches Verpassen von Fristen toleriert werden. Die Aufgabe der Prozessplanung in Echtzeit-Systemen ist es, die Abarbeitung der Prozesse so zu planen, dass nach Möglichkeit alle Fristen eingehalten werden. Die Prozessplanung kann statisch oder dynamisch erfolgen, wobei statische Planungsentscheidungen vor dem Start des Systems getroffen werden, dynamische hingegen noch nach dem Start getroffen werden können [10, Kap. 2, S. 164-167].

Planbarkeit von Echtzeit-Systemen

Es ist nicht immer möglich, die Abarbeitung aller Prozesse in einem Echtzeit-System fristgerecht zu gewährleisten. Besteht die Möglichkeit, alle Prozesse fristgerecht abzuschließen, so wird das entsprechende Echtzeit-System als „planbar (engl.: schedulable)" bezeichnet. Ist ein Echtzeit-System „planbar", so kann es real implementiert werden [10, Kap. 2, S. 165].

7.2. Parallele Prozesse im Web

Da die Implementierung der Applikation größtenteils in JavaScript erfolgt und es sich dabei um eine „single threaded" Skriptsprache handelt, Prozesse also in einem einzigen „Strang" abgearbeitet werden, besteht innerhalb von JavaScript keine Möglichkeit zur „echten" Parallelisierung von Prozessen. Es besteht lediglich die Möglichkeit, abzuarbeitende Befehle und Funktionen mit Hilfe eines „timeouts" asynchron auszuführen und somit die Reihenfolge der ablaufenden Prozesse zu manipulieren.
„Skriptsprache" bedeutet im Vergleich zu einer Programmiersprache, dass JavaScript Quellcode erst bei der Ausführung kompiliert wird, statt bereits kompiliert vorzuliegen. Das Kompilieren übernimmt bei gewöhnlichen Web-Anwendungen der Internet-

browser. Zur Ausführung einer Web-Anwendung als Applikation dient auf den Smart Devices ein auf den Internetbrowsern basierender „Web View“. Im Fall von iOS basiert dieser Web View auf dem Browser Safari, im Fall von Android auf Chrome. Browser und Web Views bieten eine Möglichkeit zur Parallelisierung von JavaScript Prozessen in Form sogenannter „Web Worker“.
Ein Web Worker beschreibt ein separates Skript, das durch ein Nachrichtensende- und empfangssystem mit einem anderen Skript kommunizieren kann. Das Deklarieren eines solchen Skriptes als Web Worker sagt dem Browser, dass der gesamte Inhalt des angegebenen Skriptes in einem separaten „Thread“ oder Bearbeitungsstrang ausgeführt werden soll. Ähnlich wie verschiedene Tabs („Reiter“) in modernen Browsern als einzelne Bearbeitungsstränge ausgeführt werden, kommuniziert der Browser dem Betriebssystem gegenüber, dass das im Web Worker befindliche Skript als separater Strang bearbeitet werden soll (siehe Green [12]). Die eigentliche Prozess- und Threadplanung übernimmt dann das jeweilige Betriebssystem in der nachfolgend beschriebenen Weise.

7.2.1. Parallele Prozesse im iOS Betriebssystem

Die Prozessplanung in iOS und macOS erfolgt laut Apple [36], [37] mit Hilfe einer Prozess-Priorisierung durch „Lottery Scheduling“ und Abarbeitung durch die „Round Robin“-Methode, wobei der Wechsel von Prozessen zwischen unterschiedlichen Prioritätsstufen beschränkt ist. Diese Beschränkung geschieht in „Bändern“ um zu gewährleisten, dass ein anfangs hoch priorisierter Prozess nicht in die niedrigste Prioritätsstufe gelangen kann und umgekehrt. Grundsätzlich läuft zunächst jede Applikation als eigener Prozess mit einem einzelnen Thread ab und kann je nach Implementierung um Threads erweitert werden. Deren weitere Bearbeitung läuft wiederum nach den oben genannten Planungsalgorithmen ab. Unter iOS und macOS bekommt somit jeder Prozess eine Priorität zugeordnet. Diese Priorität wird durch ein bestimmtes Kontingent an „Lottoscheinen“ ausgedrückt, die eine prozentuale Wahrscheinlichkeit widerspiegeln, zu der der jeweilige Prozess eine gewisse Bearbeitungszeit im Prozessor zugewiesen bekommt. Hat ein Prozess die einem „Lottoschein“ entsprechende Bearbeitungszeit erhalten und wurde nicht abgeschlossen, so gelangt er zurück in die Warteschlange aller anderen Prozesse, bis sein „Lottoschein“ ein weiteres Mal gezogen wird. Dieses Vorgehen wiederholt sich über die gesamte Bearbeitungsdauer aller Prozesse. Apple bezeichnet diese Methode als „Mach Scheduling“.

7.2.2. Parallele Prozesse im Android Betriebssystem

Android basiert auf dem Linux Kernel 2.6.25 (siehe Bird [38]) und nutzt standardmäßig die dort implementierte Prozessplanung „Completely Fair Scheduler (CFS)“ (siehe Jones [39]). Da jeder Hersteller von Android Geräten die Möglichkeit hat, die verwendete Android Version nach seinen Wünschen anzupassen, steht ihm auch die Manipulation des Kernels und somit der darin implementierten Algorithmen zur Prozessplanung frei. An dieser Stelle soll aus diesem Grund auf den Standard-Algorithmus „CFS“ eingegangen werden.

Completely Fair Scheduler (CFS)

CFS ist als Weiterentwicklung des „Fair-Share Scheduling“ eine Methode zur Prozessplanung, deren Ziel die Modellierung eines „idealen multi-tasking Prozessors“ ist. Dieses theoretische Prozessormodell erreicht eine ideale Aufteilung der Prozessorleistung auf alle parallel laufenden Aufgaben. Umgesetzt wird dieses Modell mit der Zuordnung einer für jeden Prozess spezifischen Zeit, ähnlich dem „Guaranteed Scheduling“-Modell in Abschnitt 7.1.2 auf Seite 42. Im CFS-Modell entspricht diese Zeit der Wartezeit, die vergeht, wenn ein Prozess bereit ist, bearbeitet zu werden, während ein anderer Prozess im Prozessor bearbeitet wird. Der nächste Prozess, den die Prozessplanung anschließend zur Bearbeitung auswählt, ist immer der Prozess, der die längste Wartezeit aufweist. Dessen Wartezeit verringert sich anschließend um die Zeit, die der Prozess im Prozessor bearbeitet wird. Die Prozesse befinden sich dabei in keiner einfachen Warteschlange, sondern sind in einem nach Wartezeit sortierten sogenannten „red-black-tree“ (siehe Bayer [13]) oder „Rot-Schwarz-Baum“ organisiert. Im Rot-Schwarz-Baum werden Daten in einer Baumstruktur abgelegt. Im Falle des CFS-Modells entspricht dabei das am weitesten links im Baum befindliche Blatt dem Prozess mit der höchsten Wartezeit, während am äußeren rechten Rand neue Prozesse hinzugefügt werden. Die Abarbeitung der Prozesse erfolgt somit von der linken Seite des Baumes aus. Neuere Prozesse gelangen mit zunehmender Bearbeitungszeit der linken Baumseite immer näher. Diese Baumstruktur der Daten wurde gewählt, um aufwendige Suchverfahren zur Bestimmung des nächsten Prozesses innerhalb von Warteschlangen zu vermeiden und somit die Prozessplanung zu beschleunigen. Linux erhielt bereits in Kernel Version 2.6.24 eine Erweiterung des CFS um gruppenspezifische Prozesse, um auch die Ressourcen-Bedürfnisse mehrerer Nutzer und Nutzerinnen eines Systems ausgeglichen zu versorgen.

8. Benutzungsoberfläche

A. Rosenkranz

Die grafische Gestaltung und Planung der Interaktionsmöglichkeiten bilden einen wichtigen Bestandteil der Entwicklung einer Applikation. Nicht nur die aus technischer Sicht notwendigen Bedienelemente müssen in diesem Zusammenhang bedacht werden. Auch gestalterische Richtlinien sollten beachtet werden, um eine Applikation zu entwickeln, die von der gewünschten Zielgruppe angenommen wird. Die Bedienung soll zugänglich und die grafische Oberfläche ansprechend sein. In den nachfolgenden Abschnitten werden die gestalterischen Richtlinien der für die Implementierung relevanten Plattformen zusammengefasst. Anschließend wird die Vereinbarkeit der beiden Konzepte ergründet und in Bezug darauf die grafische Oberfläche der Applikation vorgestellt. Aufgrund des technischen Fokus dieser Arbeit kann in diesem Zusammenhang nur ein Überblick über die gestalterischen Richtlinien und die Orientierung daran gegeben werden.

8.1. Gestalterische Richtlinien

In der Gestaltung von Benutzungsoberflächen (engl.: user interface, UI) bedienbarer Systeme spielen auch die Begriffe des Benutzererlebnisses (engl.: user experience, UX) und der Gebrauchstauglichkeit (engl.: usability) eine große Rolle. Diese Begriffe sollen deshalb hier kurz betrachtet werden, bevor auf die plattformspezifischen gestalterischen Richtlinien eingegangen wird.

Benutzererlebnis

DIN 9241 (Deutsches Institut für Normung [5, S. 7]) beschreibt das Benutzererlebnis oder die User Experience als „Wahrnehmungen und Reaktionen einer Person, die aus der tatsächlichen und/oder der erwarteten Benutzung eines Produkts, eines Systems oder einer Dienstleistung resultieren“. Es handelt sich damit um eine subjektive Bewertung der Erfahrung bei der Nutzung einer mobilen Applikation als Folge ihrer Gestaltung in Kombination mit der Wahrnehmung der nutzenden Person.

Gebrauchstauglichkeit

Als Gebrauchstauglichkeit oder Usability wird das „Ausmaß, in dem ein System [...] durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um festgelegte Ziele effektiv, effizient und zufriedenstellend zu erreichen“ definiert [5, S. 7]. In Bezug auf mobile Applikationen wird damit also ein Maß beschrieben, das zusammenfasst, wie gut eine Applikation durch Benutzung geeignet ist, ihren angedachten Zweck zu erfüllen.

Sowohl das Benutzererlebnis, als auch die Gebrauchstauglichkeit einer Applikation sind demnach Aspekte, die bei der Entwicklung berücksichtigt werden müssen. Bei der Entwicklung der hier betrachteten Applikation wurde sich in erster Linie an den Bedürfnissen der Messausführenden orientiert. Aufgrund des Fokus auf die technische Realisierung einer Messapplikation konnte nicht in gleichem Maße auf die Erforschung der Bedürfnisse und Erwartungen der tatsächlichen zukünftigen Nutzenden eingegangen werden, wie es bei der Entwicklung kommerzieller Anwendungen geschieht. Eine solche Untersuchung der Zielgruppe, anschließende Testreihen und Auswertung der Testergebnisse würden einen gleichen oder sogar größeren Umfang als diese Arbeit erreichen und müssen aus diesem Grund außen vorgelassen werden.

8.1.1. iOS Richtlinien

Vor allen anderen gestalterischen Richtlinien definiert Apple für iOS zunächst drei Designthemen, um das iOS System von anderen Plattformen unterscheiden zu können. Diese Themen nennt Apple „Clarity“ (deutsch: Klarheit), „Deference“ (deutsch: Rücksichtnahme) und „Depth“ (deutsch: Tiefe). Clarity, Deference und Depth sind Themen, die grob zusammenfassen, welche Idee der visuellen Inhaltsvermittlung zwischen System und Nutzer oder Nutzerin das gesamte iOS System umfassen soll (siehe Apple [40]).

Clarity fordert eine visuelle Konsistenz von Texten, Symbolen und Verzierungen, sowie einen starken Fokus des Designs auf die Funktion. Der Einsatz grafischer Elemente zur Interaktion soll dabei subtil wichtige Inhalte betonen.

Deference bezieht sich darauf, dass im gesamten iOS System die nutzende Person im Fokus steht. Klar definierte Benutzungsoberflächen und flüssige Animationen sollen der Person stets vermitteln, dass sie das System kontrolliert. Bildschirmfüllende Inhalte, minimale Verwendung von Umrahmungen, Farbverläufen und Schlagschatten sollen eine unbeschwert gestaltete Benutzungsoberfläche gewährleisten, deren Inhalt im Vordergrund steht.

Depth soll zusammenfassend ausdrücken, wie sich voneinander abhebende visuelle Ebenen und realistische Animationen Lebendigkeit vermitteln und unterstützenden Einfluss auf das Verständnis des Systems haben.

Die nachfolgenden, allgemeineren Prinzipien sollen laut Apple dazu beitragen, Eindruck und Reichweite einer Applikation zu maximieren.

Ästhetische Integrität (Aesthetic Integrity)

Die „aesthetic integrity“ einer Applikation soll repräsentieren, wie gut Erscheinungsbild und Verhalten mit der Funktion der Applikation zusammenpassen.

Konsistenz (Consistency)

Konsistenz einer Applikation beschreibt das Einfügen der Applikation in die vom System vorgegebenen Standards und Richtlinien. Häufig erfolgt dies durch Nutzung systemweit bekannter Interaktionselemente, Symbole, Textformatierungen sowie einheitlicher Bezeichnungen. Allgemein soll die Applikation Funktionen und Verhalten so widerspiegeln, wie sie von der nutzenden Person erwartet werden.

Direkte Beeinflussbarkeit (Direct Manipulation)

Die Interaktion mit der Applikation soll die nutzende Person einbinden, zum Verständnis der Applikation beitragen und ihr direkte Ergebnisse ihrer Interaktion vermitteln.

Rückmeldung (Feedback)

Rückmeldungen über aktuelle Vorgänge im Kontext der Applikation soll die nutzende Person informieren und visuelles Verhalten interaktiver Bildschirmelemente zeigt ihr Reaktionen auf ihre Interaktion an.

Metaphern (Metaphors)

Die Integration bereits bekannter Erfahrungen und Verhaltensweisen erleichtern der nutzenden Person das Erlernen des Umgangs mit der Applikation.

Kontrolle durch Nutzende (User Control)

Der nutzenden Person soll stets vermittelt werden, dass sie die Applikation kontrolliert. So soll sie beispielsweise vor fehlerhaften Verhaltensweisen gewarnt, jedoch nicht aktiv ausgesperrt werden. Es soll eine möglichst gute Balance dazwischen gefunden werden, Freiheiten im Nutzverhalten zu gestatten und dabei ungewolltes Verhalten zu umgehen.

8.1.2. Android Richtlinien

Androids Oberflächengestaltung zugrunde liegende Technologie wird als „Material Design“ bezeichnet (siehe Google [41]). „Material Design“ verfolgt das Ziel, eine „visuelle Sprache“ zu entwickeln, die klassische Prinzipien guten Designs mit der Innovation und den Möglichkeiten von Technologie und Wissenschaft in Einklang bringt. Dabei sollen die nachfolgenden drei wesentlichen Prinzipien berücksichtigt werden (siehe Google [42]).

Material is the Metaphor

Das „Material“ soll als Metapher einer Vereinheitlichung des Raumes mit einem System aus Bewegungen verstanden werden. Oberflächen und Kanten des Materials bieten visuelle Hinweise, die Nutzung bekannter Touch-Interaktionen soll der nutzenden Person schnell den Zusammenhang zwischen Funktion und Design verständlich machen. Licht, Oberfläche und Bewegung sollen vermitteln, wie Objekte sich bewegen, interagieren und sich relativ zueinander im Raum befinden.

Bold, Grapic, Intentional

Die Nutzung der Grundelemente klassischen Designs wie Schriftsetzung, Farben, Bilder und deren Anordnung sollen die visuelle Gestaltung anleiten. Mit ihrer Hilfe werden Hierarchie, Bedeutung und Fokus vermittelt. Farben sollen bewusst gewählt werden. Bilder sollen bildschirmfüllend sein. Große Schriftsetzung und bewusste Verwendung von Leerräumen sollen eine mutige grafische Benutzungsoberfläche bilden, die die nutzende Person einbezieht. Eine Betonung der Interaktion mit der nutzenden Person soll Kernfunktionalitäten sofort ersichtlich machen und Orientierungspunkte für sie bieten.

Motion Provides Meaning

Bewegungen sollen der nutzenden Person vermitteln, dass sie hauptsächlich von ihr bestimmt werden. Alle Interaktionen sollen in einer einzigen Umgebung stattfinden. Elemente der Benutzungsoberfläche sollen der nutzenden Person präsentiert werden, ohne die Kontinuität der Erfahrung in der Benutzung zu unterbrechen. Bewegung soll bedeutungsvoll und angemessen verwendet werden, um Aufmerksamkeit zu bündeln und Kontinuität aufrecht zu erhalten. Rückmeldungen sollen subtil aber klar sein.

8.2. Vereinbarkeit bei plattformübergreifender Entwicklung

Da die Applikation den Design Richtlinien von Android und iOS gleichermaßen genügen soll, ohne dabei unnötigen Mehraufwand für eine der Plattformen zu bedeuten, ist es zunächst relevant, einen Konsens zwischen beiden Gestaltungsphilosophien zu finden.

Die wesentlichen Prinzipien der Designrichtlinien beider Plattformen lassen sich in guter Vereinbarung miteinander zusammenfassen. So steht in beiden Systemen die nutzende Person im Mittelpunkt und soll sich in ihrem Nutzverhalten immer als bestimmender Faktor in der Interaktion mit der Applikation bestätigt sehen. Eine leichte Verständlichkeit der grafischen Oberfläche mit einem Fokus auf die Funktion der Applikation soll sicher gestellt werden. Dabei soll eine visuelle Konsistenz unter subtilem Einsatz von Grafiken, Bedienelementen und Schriftsetzung erzielt werden.

Im nachfolgenden Abschnitt soll die Benutzungsoberfläche der Applikation vorgestellt und ihre Kompatibilität mit diesen Richtlinien aufgezeigt werden.

8.3. Benutzungsoberfläche der Applikation

Die Komponenten der Benutzungsoberfläche zur Interaktion mit der Applikation bestehen fast ausschließlich aus sogenannten „UI Komponenten“ der zur Implementierung verwendeten Entwicklungsumgebung. Alle darin vorkommenden Elemente sind so angelegt, dass sie den nativen grafischen Elementen der Plattform, auf der sie dargestellt werden, entsprechen. Dies bedeutet „Material Design“ Elemente im Kontext des Android Systems und die plattformweit verfügbaren Bedienelemente im Kontext

des Apple Systems. Für Symbole, sogenannte „Ionic Icons“ der Entwicklungsumgebung trifft dieses Verhalten ebenso zu. Das Bedienelement zur Aufnahmesteuerung, eine Grafik sowie die grafische Darstellung der Messergebnisse sind die einzigen visuellen Komponenten der Applikation, die nicht Teil der von beiden Betriebssystemen zur Verfügung gestellten Elemente sind.

Die grafische Oberfläche der Applikation besteht aus drei separat dargestellten, bildschirmfüllenden Ansichten. Ein Menü am unteren Bildschirmrand unterteilt die drei Ansichten in sogenannte „Tabs“. Zwischen allen drei Tabs kann jederzeit durch die Nutzung dieses Menüs oder „wischen“ des Bildschirms in die entsprechende Richtung gewechselt werden. Die Tabs erfüllen getrennt voneinander ihre jeweilige Funktion und sind zur Orientierung innerhalb der Applikation am Kopf der Ansicht mit einer Bezeichnung versehen.

8.3.1. Aufnahmesteuerung

Der erste Bildschirm zur Bedienung der Applikation, den die nutzende Person nach dem Start zu sehen bekommt, ist als „Measurement Tab“ für die Steuerung der Aufnahme zuständig. Zentrales Element ist in dieser Ansicht der über dem Tab-Menü befindliche Button zur Aufnahmesteuerung. Zuerst präsentiert er sich der nutzenden Person als kreisrunder Button in hellem und dunklen grau, mit einem roten Kreis in seiner Mitte, der von einem weißen Ring mit schwarzer Umrahmung umfasst ist (Abbildung 8.1). Dieses Symbol ist plattform- und geräteübergreifend als „Record“-Symbol bekannt und vermittelt der nutzenden Person, dass durch Interaktion mit ihm ein Aufnahmevorgang gestartet werden kann. Wird der Button betätigt, wechseln sowohl Farbschema des Buttons als auch das Symbol in seiner Mitte. Das Symbol wechselt zu einem schwarzen Viereck, dem bekannten Symbol zum Stoppen eines Vorgangs, und soll der nutzenden Person vermitteln, dass sie den gestarteten Vorgang durch ein wiederholtes Betätigen des Buttons jederzeit beenden kann. In Tabelle 8.1 sind die verschiedenen Zustände des Buttons abgebildet. Beim Start einer Aufnahme färbt sich der Button gelb mit orangem Rand und signalisiert damit, wie auch die Farbgebung einer Verkehrsampel, einen Zwischenzustand. Im Kontext der Applikation zeigt die gelbe Färbung eine Schätzung des in einem Raum befindlichen Hintergrundrauschens an, bevor der eigentliche Aufnahmevorgang gestartet werden kann. Ist dieser Vorgang abgeschlossen, wechselt der Button zur grünen Farbe, die vermitteln soll, dass die Applikation zur Aufnahme bereit ist. Wird der Aufnahmevorgang durch einen Stimulus ausgelöst, der einen bestimmten Pegel überschreitet, wechselt der Button seine Farbe zur Signalfarbe rot und zeigt damit an, dass eine Aufnahme stattfindet. Ist die Aufnahme abgeschlossen, wird erneute Aufnahmebereitschaft mit einem Wechsel zur grünen Farbgebung signalisiert. Hinweise zur Bedienung werden der nutzenden Person durch ein halb transparentes Overlay beim ersten Start der Applikation präsentiert und können auch jederzeit durch die Betätigung des „Help-Buttons“ am oberen rechten Rand der Ansicht angezeigt werden. Während im iOS System Buttons auf beiden Seiten des Kopfes der dargestellten Ansicht platziert werden können, da sich der Bezeichner „Measurement“ der Ansicht in der Mitte des Kopfes befindet, wird der Bezeichner der Ansicht des Android Betriebssystems an der linken Seite des Kopfes platziert. Somit ist es für das Android System unüblich, Buttons in der linken oberen Ecke des

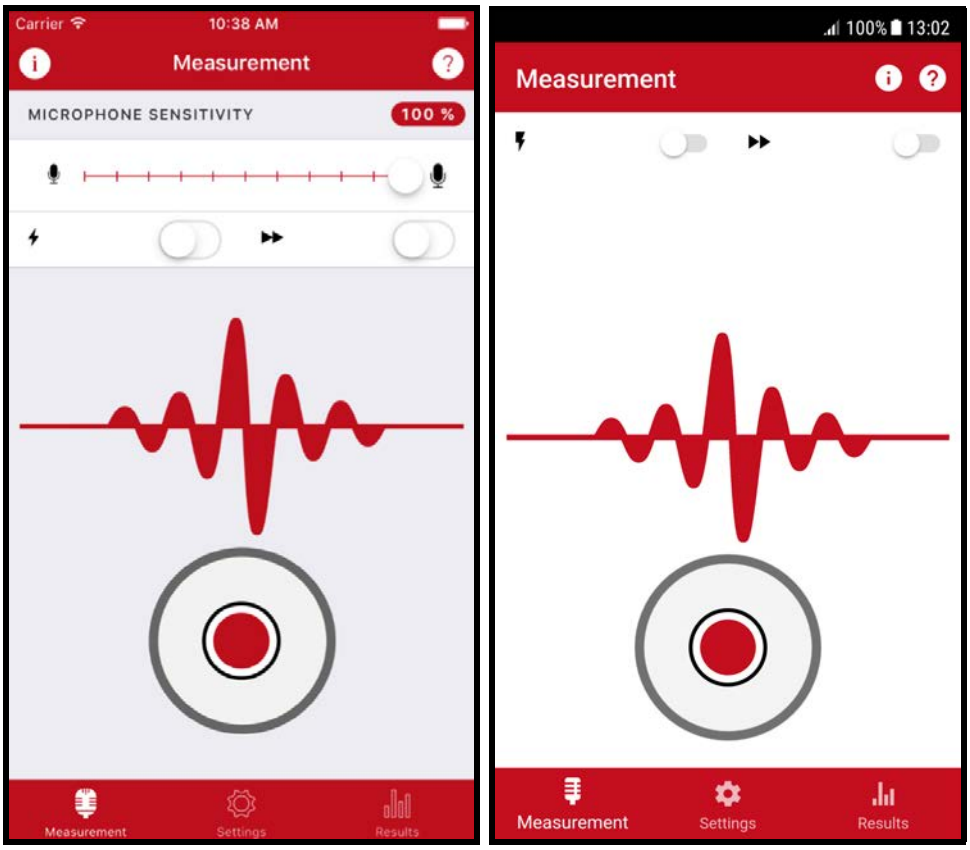


Abbildung 8.1.: Measurement Tab in iOS (links) und Android (rechts).

Seitenkopfes darzustellen. Aus diesem Grund befindet sich der mit einem „i“ versehene Button zum Aufruf der Informationsseite zur Applikation im iOS System am linken oberen Rand, während er im Android System am rechten oberen Rand, links des „Help-Buttons“ zu finden ist. Für eine genauere Beschreibung der Bedienung der Applikation sei auf die im digitalen Anhang befindliche Bedienungsanleitung verwiesen. Ein weiteres grafisches Element befindet sich in Form eines textfreien Logos in der Bildschirmmitte des „Measurement Tabs“. Das Logo ist als wiederkehrendes Element sowohl im Icon als auch im Startbildschirm der Applikation zu finden (siehe A.3). Es dient zusätzlich in der Ansicht dem Zweck, die große einfarbige Fläche zwischen Aufnahmebutton am unteren und den restlichen Bedienelementen am oberen Bildschirmrand aufzubrechen. Alle weiteren Bedienelemente dieser Ansicht entsprechen den nativen Standardelementen der jeweiligen Plattform. Ihre Funktion wird durch Text und Symbole vermittelt. So dient der am oberen Bildschirmrand befindliche Schieberegler zur Einstellung der Mikrofonvorverstärkung. Der aktuelle Wert der Einstellung wird durch eine Prozentzahl hinter dem Schriftzug „Microphone Sensitivity“ dargestellt. Zusätzlich wird die Funktion des Schiebereglers durch ein verkleinertes und ein gewöhnlich großes Mikrofonsymbol angezeigt. Da das Android Betriebssystem keine Manipulation des Mikrofonvorverstärkers erlaubt, ist dieser Schieberegler und die zugehörige Beschriftung nur in der iOS Applikation verfügbar. Darunter befinden sich zwei Schalter. Der linke der beiden Schalter wird durch ein Blitzsymbol beschrieben und dient zur Aktivierung des Blitzes am Smart Device zur zusätzlichen Signalisierung des aktuellen Aufnahmezustands. Der rechts davon befindliche Schalter

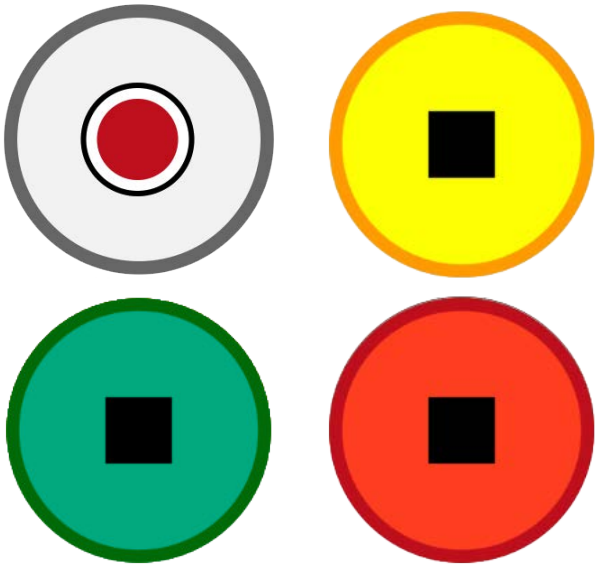


Tabelle 8.1.: Visuelle Hinweisgebung durch die Aufnahmesteuerung

neben dem ebenfalls von Audio-Abspielgeräten bekannten „Fast Forward“-Symbol aktiviert einen Modus der Applikation, in dem die Parameterberechnung direkt nach dem Abschluss des Aufnahmevorgangs ausgeführt wird. Animationen und Symbole vermitteln der nutzenden Person somit auf leicht verständliche Weise die Arbeitsschritte und Funktionen der Aufnahmesteuerung und überlassen ihr dabei jederzeit die Kontrolle über alle Vorgänge. Nach erfolgreicher Beendigung eines Aufnahmevorgangs wird die nutzende Person durch automatischen Wechsel zur nächsten Ansicht weitergeführt.

8.3.2. Aufnahme- und Parametereinstellungen

Die Aufnahme- und Parametereinstellungen erfolgen im „Settings Tab“ (Abbildung 8.2). Diese Ansicht besteht ausschließlich aus nativen Bedienelementen, deren Funktion durch Text beschrieben ist. Die verschiedenen Einstellungen sind thematisch gruppiert und als Liste angeordnet. Am oberen Rand der Ansicht befindet sich ein Button mit der Aufschrift „Calculate“, der der nutzenden Person immer an dieser Stelle der Ansicht präsentiert wird, egal wo sie sich in der Auswahlliste befindet. Mit diesem Button kann, nach erfolgreicher Aufnahme, jederzeit die Parameterberechnung gestartet werden. Wurde bis zur Betätigung des Buttons keine Aufnahme getätigt, wird die nutzende Person darauf hingewiesen. Während der Berechnung wird in Form eines halb transparenten Overlays signalisiert, dass ein Berechnungsvorgang läuft. Ist die Berechnung abgeschlossen, erfolgt eine automatische Weiterleitung zur nächsten Ansicht.

8.3.3. Ergebnisanzeige

In der nächsten Ansicht „Results Tab“ (Abbildung 8.3) werden nach erfolgreicher Parameterberechnung die Ergebnisse der Messung visualisiert. Umrahmt vom nativen Grafikelement der „Karten (engl.: cards)“ werden die unterschiedlichen Ergebnisse

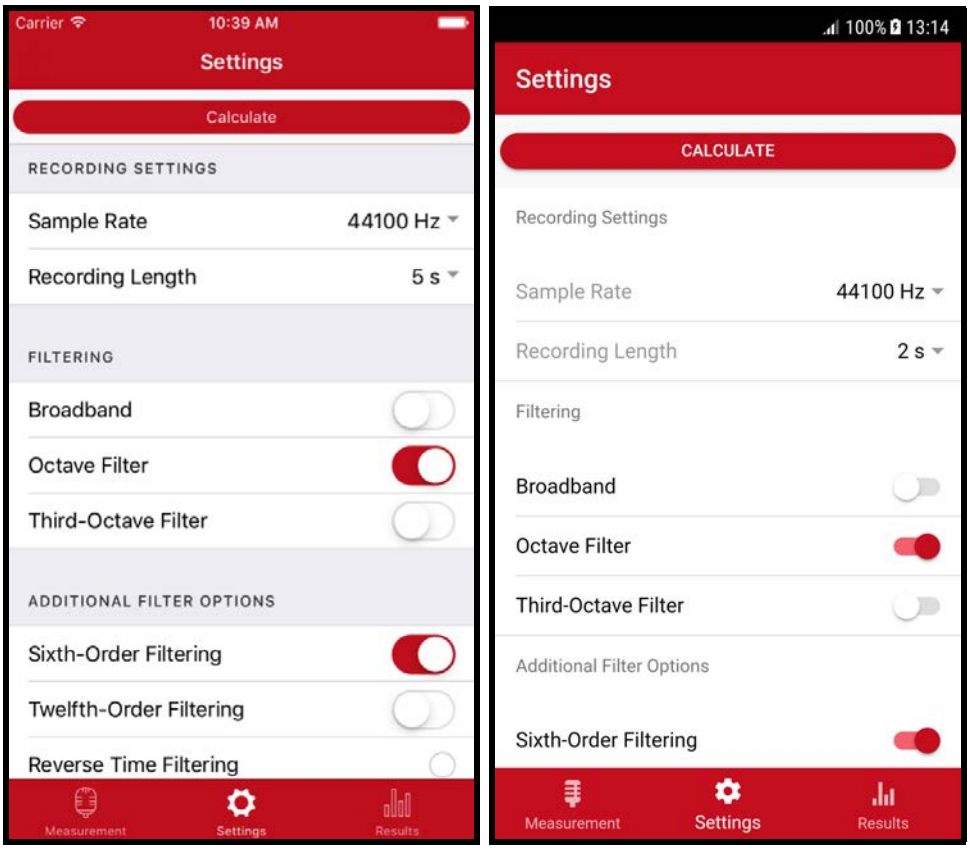


Abbildung 8.2.: Settings Tab in iOS (links) und Android (rechts).

voneinander getrennt in Listenform dargestellt. Wird die Nachhallzeit berechnet, so erfolgt im oberen Bereich der Ansicht die Darstellung der Ergebnisse in Form eines Diagramms, anschließend in Form einer Tabelle. Alle anderen ausgewählten Parameter werden ebenfalls in Tabellenform angezeigt. Jede „Ergebnis-Karte“ ist mit einem Titel versehen, um der nutzenden Person eindeutig zu vermitteln, welcher Parameter darin dargestellt ist. Die Auflistung der Parameter in Tabellen erfolgt in grafisch schlichter Form im Hauptfarbschema der Applikation, bestehend aus rot, schwarz, weiß und grau, den Farben der Technischen Universität Berlin. Die gesamte Darstellung ist schlicht und sachlich gehalten, um dem wissenschaftlichen Anspruch gerecht zu werden und die Messergebnisse im Mittelpunkt stehen zu lassen. Weitere interaktive Elemente im „Results Tab“ sind im Kopf der Ansicht zu finden. Das jeweilige plattformspezifische Symbol zum Teilen von Daten kann durch Antippen genutzt werden, um die Ergebnisse einer Messung mit Hilfe anderer Applikationen vom Smart Device aus zu teilen. Das ebenfalls bekannte Papierkorb Symbol befähigt die nutzende Person, lokal gespeicherte Daten zu löschen.

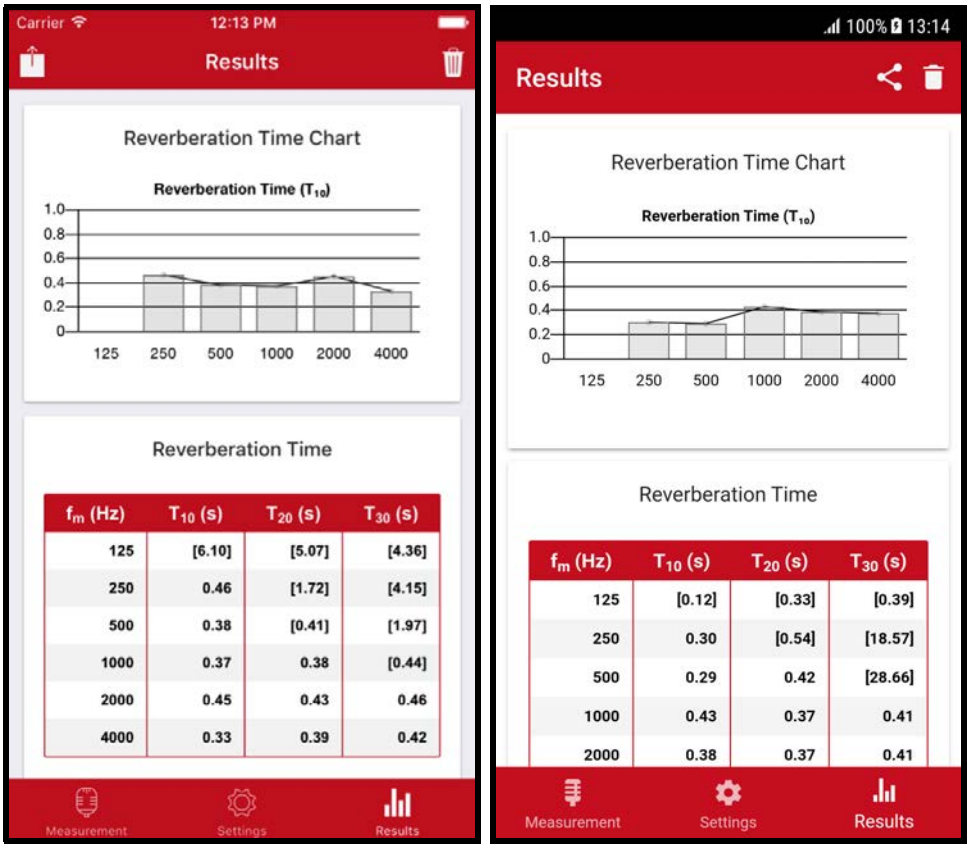


Abbildung 8.3.: Results Tab in iOS (links) und Android (rechts).

9. Android Limitationen und Lösungsansätze

R. Burgmayer, A. Rosenkranz

In diesem Kapitel soll aufgezeigt werden, wie fehlende Schnittstellen des Android Betriebssystems die Nutzung von Android-basierten Geräten für akustische Messungen erschwert und welche Lösungsansätze dabei Abhilfe schaffen können. Zunächst erfolgt eine Betrachtung der Messergebnisse, die für ein iOS Gerät und ein Android Gerät in gleicher Messumgebung erhalten werden. Als Referenz gilt das mobile Messgerät NTi XL2 (NTi Audio AG [62]). Der Messaufbau gleicht dem im nachfolgenden Kapitel 10.1 auf Seite 67 beschriebenen.

9.1. Android Messergebnisse

Die nachfolgenden Messergebnisse stammen aus dem Hörsaal MA 041 der Technischen Universität Berlin. Bei den Messergebnissen ist zunächst auffällig, dass das Android System stets in jedem Frequenzband Ergebnisse liefert (Abbildungen 9.2, 9.5), obwohl das iOS System unter Verwendung des gleichen Mikrofons die Ergebnisse der niedrigsten Frequenzbänder aufgrund eines zu geringen Signal-Rausch-Abstands ausschließt (Abbildungen 9.1, 9.3). Details können den Messwerttabellen B.27, B.28, B.29 und B.30 im Anhang B entnommen werden. Die vom Android System bestimmten Nachhallzeiten in den niedrigen Frequenzbereichen bis 315 Hz liegen zum Teil 50 % unter den vom NTi System angegebenen Ergebnissen. Dieses Verhalten ist auf die Nachbearbeitung des Audiosignals durch Hard- und Software des Android Gerätes zurückzuführen. Die Einflüsse der speziellen Nachbearbeitung des verwendeten Testgerätes sind im Vergleich der Abbildungen 9.4 und 9.8 zu erkennen. Gezeigt werden die ersten 4410 Samples einer mit der Applikation im Android System aufgezeichneten Impulsantwort. Abbildung 9.4 entspricht einer mit dem Gesamtsystem aus Applikation, Android Gerät und micW i436 Mikrofon (mic W [60]) aufgezeichneten Impulsantwort. Beim micW i436 Mikrofon handelt es sich um ein nach IEC-61672 Klasse 2 (Deutsches Institut für Normung [6]) zertifiziertes Messmikrofon mit omnidirektionaler Charakteristik. Besonders am Signalanfang ist eine starke Verfälschung des Signalverlaufs zu erkennen. Somit kann das Android Gerät nicht ohne Weiteres als das beschriebene Gesamtsystem eingesetzt werden. Im Folgenden sollen Einschränkungen und Lösungsansätze für die Nutzung des Android Systems aufgezeigt werden.

MA 041: Vergleich der Nachhallzeiten zwischen iOS und NTi XL2, oktavgefiltert

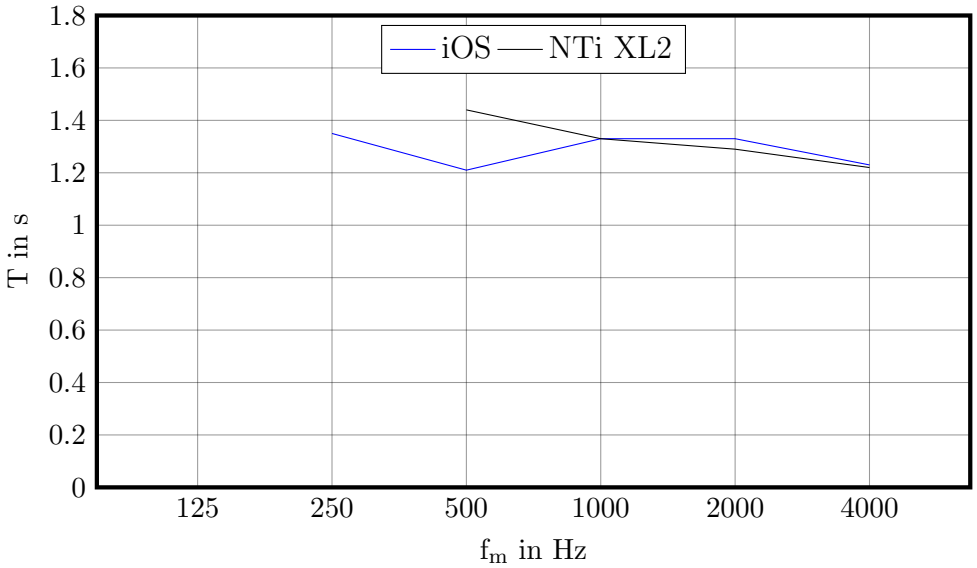


Abbildung 9.1.: Nachhallzeiten im MA 041, oktavgefiltert.

MA 041: Vergleich der Nachhallzeiten zwischen Android und NTi XL2, oktavgefiltert

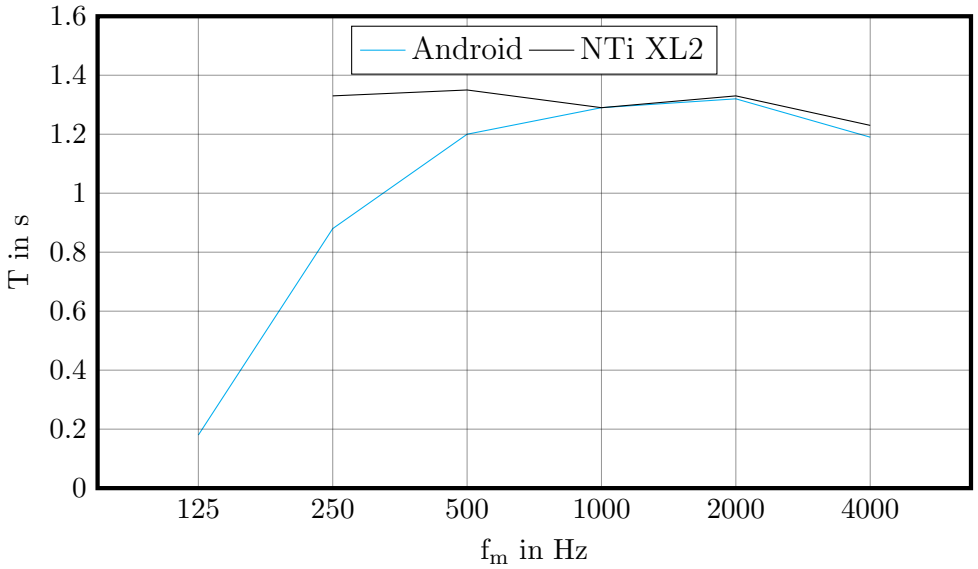


Abbildung 9.2.: Nachhallzeiten im MA 041, oktavgefiltert.

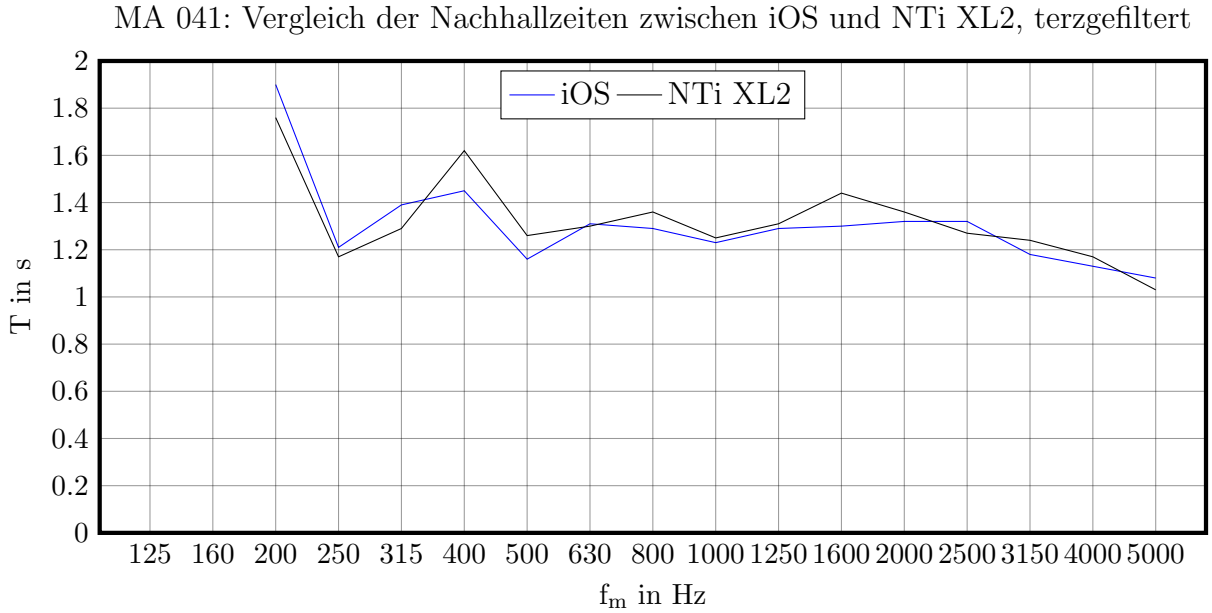


Abbildung 9.3.: Nachhallzeiten im MA 041, terzgefiltert

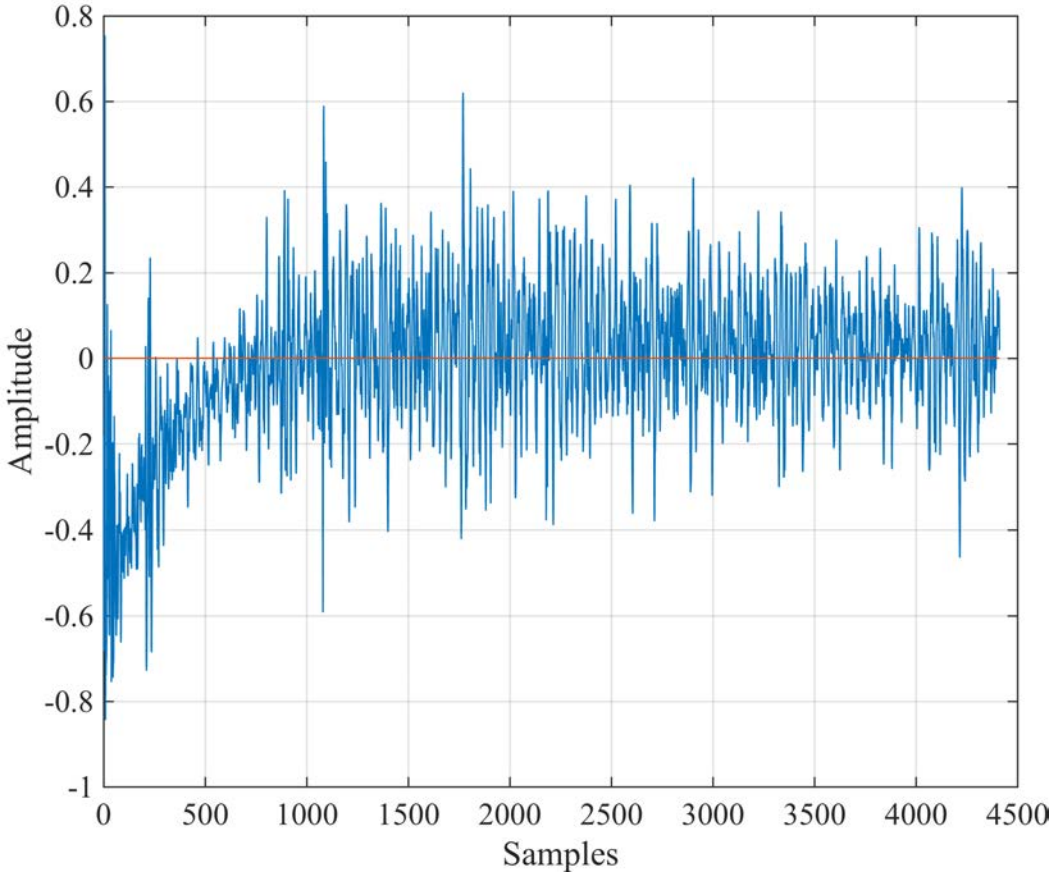


Abbildung 9.4.: Android Impulsantwort mit micW i436, verfälscht

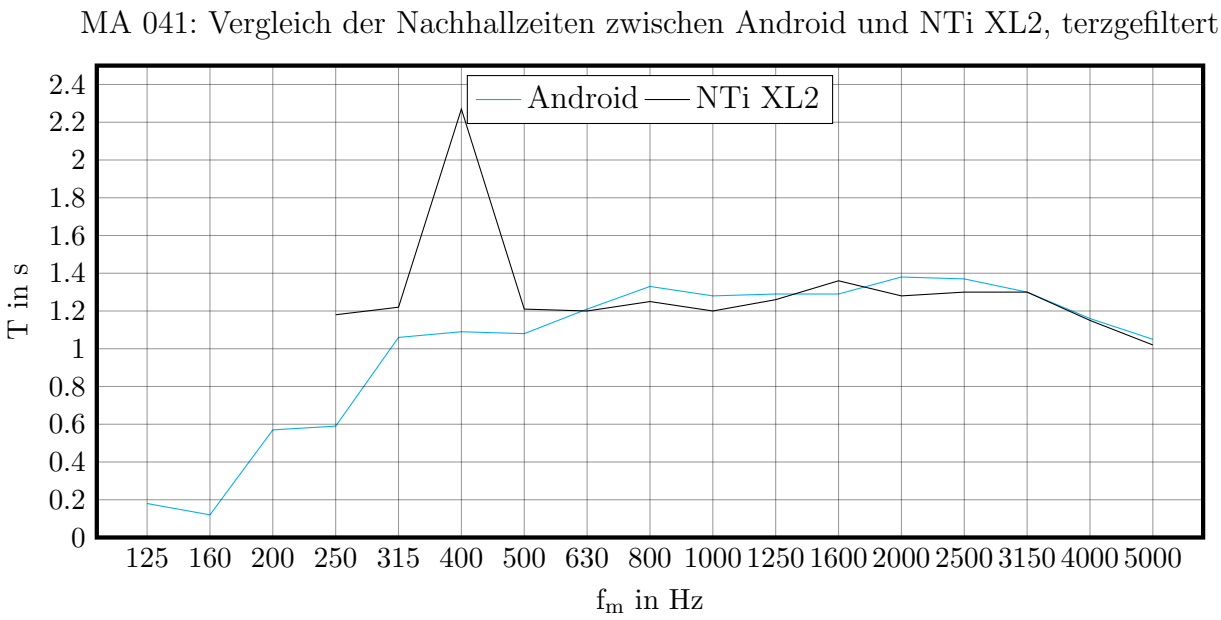


Abbildung 9.5.: Nachhallzeiten im MA 041, terzgefiltert

9.2. Android Limitationen

9.2.1. Vorverarbeitung der Audiosignale

Während für das iOS-Betriebssystem jeweils nur wenige aktuelle Geräte von einem gemeinsamen Hersteller existieren, gibt es für das Android-System eine Vielzahl verschiedener Smart Devices von einer großen Anzahl an Herstellern. Die Effekte zur Vorverarbeitung der aufgenommenen Audiosignale unterscheiden sich je nach Hersteller. Laut der Android Dokumentation (Google [43]) existiert seit der Android Systemversion 7.0 eine Programmierschnittstelle zum Zugriff auf die unbearbeiteten Audiodaten des Mikrofons. Jedoch ist es den Geräteherstellern freigestellt, diesen Aufnahmemodus auch tatsächlich in ihren Android Geräten zur Verfügung zu stellen (siehe Google [44]). Wird der Aufnahmemodus für unbearbeitete Audiodaten vom Hersteller nicht zur Verfügung gestellt, so verhält sich die Aufnahme wie der vom Hersteller definierte Standard für die Aufnahme, ohne eine Rückmeldung an die nutzende Person möglich zu machen. Daraus folgt zumindest für die aktuelle Version des Betriebssystems, dass jeder Hersteller anders mit der Weiterverarbeitung der vom Mikrofon eingehenden Audiosignale verfahren kann und kein allgemein gültiges Vorgehen zur Beseitigung der Vorverarbeitung existiert. Eine Programmierschnittstelle zur Überprüfung aller aktivierten Audioeffekte ist bisher ebenfalls nicht vorhanden. Für jedes Gerät bedarf es daher einer umfassenden Analyse, ob auf unbearbeitete Audiodaten zugegriffen werden kann.

9.2.2. Mikrofon-Vorverstärkung

Eine weitere Einschränkung bei der Verwendung von Android-Endgeräten ist der fehlende Zugriff auf die Mikrofonvorverstärkung. Die Android-Programmierschnittstelle erlaubt keine direkte Manipulation des Vorverstärkerwertes (siehe Google [45] und

Audio Evolution [46]). Diese Einschränkung ist seit längerem bekannt, aber wird seitens Google nicht behoben, da laut Android Support bei entsprechender Kalibrierung theoretisch keine Regelung der Mikrofonvorverstärkung notwendig ist (siehe Pfitzinger Voice Design [47]). Laut Android Kompatibilitätsrichtlinien [44] werden Mikrofon und Analog-Digital-Wandler (ADC) als eine Einheit definiert und sollten dementsprechend kalibriert werden. Die Kalibrierung nach den Richtlinien von Android, Source Tuning genannt, sollte je nach gewähltem Audio-Aufnahme-Modus entsprechend kalibriert sein [48]. Auf der Webseite des App Herstellers Pfitzinger Voice Design [47] wird der Sachverhalt anhand eines konkreten Beispiels erläutert. Diese Richtlinien sind in einigen Fällen nicht bindend und müssen von den Herstellern der Android Endgeräte nicht befolgt werden. Zusätzlich führt dieser Ansatz bei der Verwendung von externen Mikrofonen zu Problemen, da die Vorkalibrierung durch den Hersteller auf die intern verbauten Mikrofone beschränkt ist. Somit bilden empfindlichere externe Mikrofone und Analog-Digital-Wandler keine kalibrierte Einheit mehr und der Zugriff auf den Mikrofonvorverstärker ist wieder notwendig, um ein Übersteuern unterbinden zu können.

9.2.3. Einflüsse auf die Signalgestalt

Die zuvor erwähnten möglichen Einschränkungen bei der Aufnahme von Audiosignalen durch Android Geräte können die Gestalt des aufgezeichneten Signals beeinflussen. Die im Android System verwendeten Audioeffekte wie Rauschunterdrückung, akustische Echoauslöschung und automatische Kontrolle der Eingabelautstärke werden angewendet, um der nutzenden Person bei der Sprachkommunikation möglichst gute Verständlichkeit zu garantieren. Für raumakustische Messungen wird jedoch ein möglichst unverfälschtes Signal benötigt, um genaue Berechnungen daran ausführen zu können. Sind diese Audioeffekte nicht deaktivierbar, so können sie einen ungewollten Einfluss auf das zu messende Signal ausüben. Die Kombination mehrerer Audioeffekte kann ungewollte Einflüsse noch verstärken. Im Vergleich der Abbildungen 9.4 und 9.8 sind für das beispielhaft untersuchte Android Gerät (Samsung Galaxy S6, Android 7) Einflüsse von nicht zu deaktivierender Audio-Nachbearbeitung auf die Signalgestalt zu erkennen.

Einfluss fehlender Regelung der Mikrofonvorverstärkung

Kann der im Gerät eingebaute Mikrofonvorverstärker nicht geregelt werden, so ist eine Übersteuerung des eingehenden Signals möglich. Dabei ist das anregende Signal so laut, dass es vom Mikrofonvorverstärker über einen Zeitraum von mehreren Audiosamples als Maximalausschlag an die nachfolgende Audioverarbeitung weitergegeben wird. Im Fall einer aufgezeichneten Impulsantwort und anschließender Bestimmung einer Nachhallzeit kann dies bedeuten, dass die Regressionsanalyse über einen falschen Bereich der Abklingkurve durchgeführt wird und somit zu falschen Ergebnissen führt. Dies kann je nach Startpunkt der Regression zu längeren oder kürzeren Nachhallzeiten führen. Die frühe Abklingkurve wird vor allem bei starker Übersteuerung über einen Bereich zu hoher Werte summiert und kann ebenfalls zu falschen Ergebnissen führen. Auch für das Signal-Rausch-Verhältnis führt Übersteuerung zu falschen Resultaten.

Einfluss automatischer Eingabelautstärke

Das Ziel einer automatischen Steuerung der Eingabelautstärke (Automatic Gain Control [49]) ist es, bei einem Audiosignal einen gleichbleibenden Geräuschpegel zu erzielen. In der praktischen Anwendung entspricht das einer Normalisierung eines Audiosignals durch Anheben oder Senken des Eingangspegels während der Aufnahme. Dieser Effekt ist sinnvoll für eine gute Sprachverständlichkeit bei schwankender Lautstärke. Bei raumakustischen Messungen, die mit der Applikation durchgeführt werden sollen, ist jedoch das Abklingen - also die Pegelminderung im Verlauf des Signals - wichtig für die nachfolgenden Berechnungen. Ein abklingendes Signal wird durch die Anhebung des Pegels über seine Laufzeit stark verfälscht und wirkt sich negativ auf die Genauigkeit der Parameterberechnung aus.

Einfluss einer Rauschunterdrückung

Die Rauschunterdrückung (Noise Supression [50]) ist ein Effekt, der als ungewollt angesehene Signalanteile aus dem aufgezeichneten Signal entfernt. Diese Signalanteile können das Hintergrundrauschen in einem Raum sein, aber auch andere Geräusche im Hintergrund, die vom System als „ungewollt“ interpretiert werden. Für die raumakustische Messung bedeutet dies unter anderem, dass das Hintergrundrauschen aus dem Signal entfernt wird, aber aufgrund überlappender spektraler Anteile auch andere Signalanteile aus dem Signal gelöscht werden. Das Signal und somit auch die Ergebnisse der Messung werden verfälscht.

Einfluss einer akustischen Echoauslöschung

Aufgabe der akustischen Echoauslöschung (Acoustic Echo Cancellation [51]) ist es, ein Signal möglichst schnell und effektiv von Echos in Form von sich ungewollt wiederholenden Signalanteilen wie dem Feedback zwischen Lautsprecher und Mikrofon aus dem Audiosignal zu entfernen. Bei raumakustischen Messungen kann dies ebenfalls frühe energiereiche Reflexionen des Anregesignals als ungewolltes Echo erkennen, bedämpfen und somit den Signalverlauf verfälschen.

9.3. Lösungsansätze

In diesem Abschnitt werden mögliche Lösungsansätze zur Umgehung der Einschränkungen des Android-Systems aufgezeigt und beschrieben.

9.3.1. Root-Zugriff

Der Root-Zugriff bezeichnet das Erstellen eines Administrator-Kontos mit uneingeschränktem Dateizugriff auf das Android-Betriebssystem. Nach Beschaffung der zusätzlichen Rechte können die gerätespezifischen Effekte zur Audioverarbeitung für jeden Aufnahmemodus aktiviert bzw. deaktiviert werden. Zur Bearbeitung der Aufnahmemodi gibt es dann die Möglichkeit, eine zusätzliche Konfigurationsdatei (siehe Google [50]) im Android System anzulegen. Mit diesen Zugriffsrechten kann auch der Vorverstärker für einzelne Applikationen auf einen festen Wert gesetzt werden. Die

Manipulation des Vorverstärkers ist durch die Änderung entsprechender Dateien ebenfalls möglich (siehe [52]). Zusätzlich kann die Veränderung der Mikrofonvorverstärkung durch Bearbeitung des Kernels des Android Betriebssystems erreicht werden. Im Kernel der Android-Endgeräte können umfangreiche Einstellungen vorgenommen werden, die einer nutzenden Person mit Standardrechten nicht zur Verfügung stehen. Durch die Root Rechte erhält die nutzende Person vollen Lese- und Schreibzugriff auf alle Dateien des Android Systems und kann in den entsprechenden Dateien den Verstärkungsfaktor des Mikrofons anpassen sowie die Vorverarbeitung des Audiodatenstroms vom Mikrofon deaktivieren.

Im Google Playstore - Googles eigener Vertriebsplattform für Applikationen - finden sich Applikationen, die unter Voraussetzung eines Root-Zugangs und Verwendung eines entsprechend angepassten Kernels den Zugriff auf die Mikrofon-Vorverstärkung erlauben. Ein konkretes Beispiel ist die Applikation HD2 Mic Control (SimpleMobDev [53]). Im XDA Entwicklerforum [54] finden sich Anleitungen und bereits bearbeitete Kernel-Dateien, die zu diesem Zweck genutzt werden können.

Das Erstellen eines Root Zugangs und die Veränderung des Kernels stellen in der Regel einen vom Hersteller nicht vorgesehenen Eingriff in das System dar. Neben dem damit verbundenen Aufwand, sowie eventuellem Garantieverlust, besteht die Gefahr, dass das Gerät beim Root-Vorgang Schaden nimmt. Der Schaden kann sowohl software- als auch hardwareseitig sein. Im Extremfall können die Schäden irreversibel sein und das Smart Device funktionsunfähig machen. Die Garantiebestimmungen variieren von Hersteller zu Hersteller. Ein solcher Eingriff in das Android System sollte nur von erfahrenen Nutzenden durchgeführt werden und bietet somit eine mögliche, jedoch umständliche Lösung zur Beseitigung ungewünschter Bearbeitung des Mikrofonsignals. Auch wenn eine Manipulation des internen Vorverstärkers möglich ist, müssen die entsprechenden Werte immer vor der Inbetriebnahme einer Applikation gesetzt werden und erlauben keine Anpassung im laufenden Betrieb.

9.3.2. Externer Mikrofonvorverstärker

Wenn der Hersteller eine Aufnahme ohne Vorverarbeitung ermöglicht oder die Vorverarbeitung mittels Root-Zugriff deaktiviert wurde, kann ein externer Mikrofon-Vorverstärker für Smart Devices zur Justierung der Mikrofon-Verstärkung verwendet werden. Der Anschluss des Vorverstärkers erfolgt über den vierpoligen Klinkeneingang des mobilen Endgerätes. Viele der erhältlichen Vorverstärker verfügen über eine optionale Phantomspeisung. Somit können auch professionelle Messmikrofone verwendet werden.

Da die mobile Applikation für schnelle, einfach auszuführende Messungen gedacht ist, wirkt sich das Mitführen eines Vorverstärkers negativ auf die Mobilität und das damit einhergehende Erlebnis bei der Nutzung aus. Jedoch können mit Verwendung hochwertiger Mikrofone so auch genauere raumakustische Analysen durchgeführt werden.

9.3.3. USB Audio-Interface

Die Verwendung eines USB Audio-Interfaces stellt einen weiteren Lösungsansatz dar. Die Vorverarbeitung durch die Geräte wird umgangen und der Wert der Vorverstärkung kann direkt am Interface gesteuert werden. Generell sind Geräte ab Android 3.1 fähig, Audio über USB zu nutzen (siehe USB_CLASS_AUDIO in Google [55]), jedoch gibt es auch hier geräteabhängig Ausnahmen. Laut anderen Quellen (siehe Extream SD [56]) wird USB-Audio sogar erst ab Android Version 5 unterstützt. Für aktuelle Versionen des Betriebssystems ist die Kompatibilität gewährleistet, wenn das Endgerät eine USB-Host-Fähigkeit besitzt. Zum Anschluss eines regulären USB Audio-Interfaces wird lediglich ein USB Adapter mit sogenannter „On-The-Go“-Funktionalität benötigt, um ein USB Kabel mit dem Micro-USB Eingang des Android Gerätes zu verbinden. Der größte Vorteil dieser Lösung ist das vollständige Umgehen jeglicher Audioeffekte des Android Systems.

Im Hinblick auf die Transportabilität stellt das Mitführen eines USB-Audio-Interfaces und eines Messmikrofons einen negativen Einfluss auf die angestrebte Mobilität der Applikation dar und verschlechtert somit auch das Benutzungserlebnis. Jedoch können auf diese Weise auch qualitativ hochwertige Messungen mit Mikrofonen ausgeführt werden, die in Genauigkeit und Leistungsumfang einem direkt an das Smart Device angesteckten Mikrofon überlegen sind. Nach Evaluation der Signalverarbeitung durch Ralf Burgmayer [58] kann gefolgert werden, dass die implementierte Applikation mit Hilfe eines USB-Audio-Interfaces in Kombination mit dem Android Gerät zu einer genauen und umfassenden Evaluation raumakustischer Parameter in der Lage ist.

Abbildungen 9.6 und 9.7 stellen gemessene Nachhallzeiten eines iOS Systems mit micW i436 Mikrofon [60], eines AndroidSystems mit USB-Audio-Interface und Messmikrofon und dem NTi XL2 [62] Messsystem gegenüber. Im Vergleich zu den Ergebnissen des vorangegangen Abschnitts 9.1 für das untersuchte Android System zeigt sich eine gute Übereinstimmung der gemessenen Nachhallzeiten über alle Systeme. Somit kann die Applikation durch Nutzung eines externen Interfaces als hochwertiges Messsystem angesehen werden. Die Kombination von Applikation und externem Interface ist seit iOS 7 ebenfalls für das iOS System möglich. Im Hinblick auf eine zukünftige Erweiterung der Applikation eröffnet die Verwendung eines USB-Audio-Interfaces so ebenfalls zusätzliche Optionen für einen Ausbau ihrer Funktionalität. Die unter Abbildung 9.8 dargestellte Impulsantwort wurde mit diesem Lösungsansatz aufgezeichnet. Im Vergleich zu Abbildung 9.4 sind die Unterschiede in der Signalgestalt deutlich zu erkennen. Als Audio-Interface wurde ein Focusrite Scarlett 2i4 (Focusrite [74]) und als Messmikrofon ein Behringer ECM 8000 (Behringer Music Group [61]) genutzt. Die im nachfolgenden Kapitel zur Evaluation verwendeten Messergebnisse des Android Systems wurden ebenfalls auf diese Weise gewonnen.

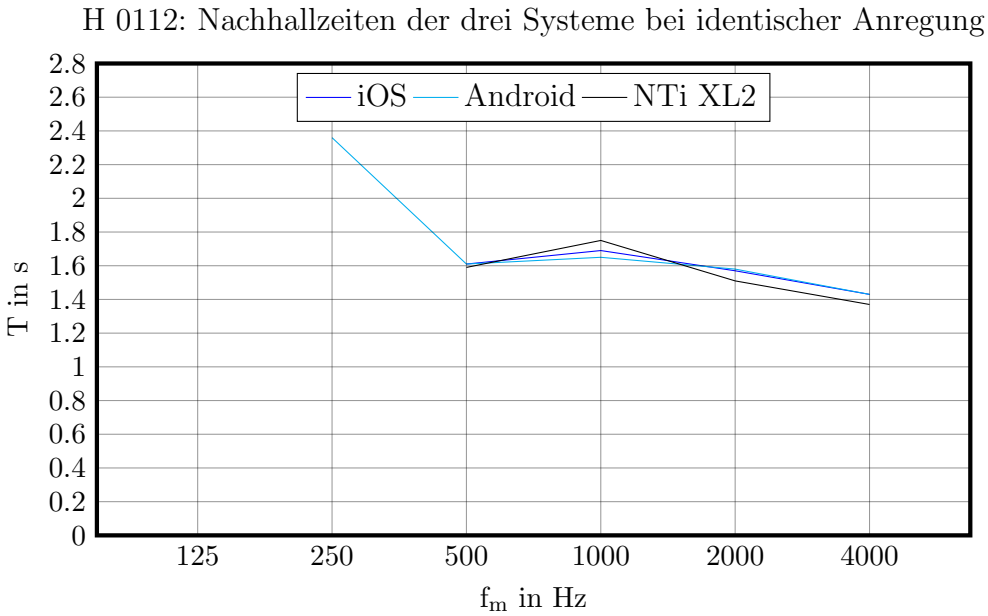


Abbildung 9.6.: Nachhallzeiten im H 0112, oktavgefiltert.

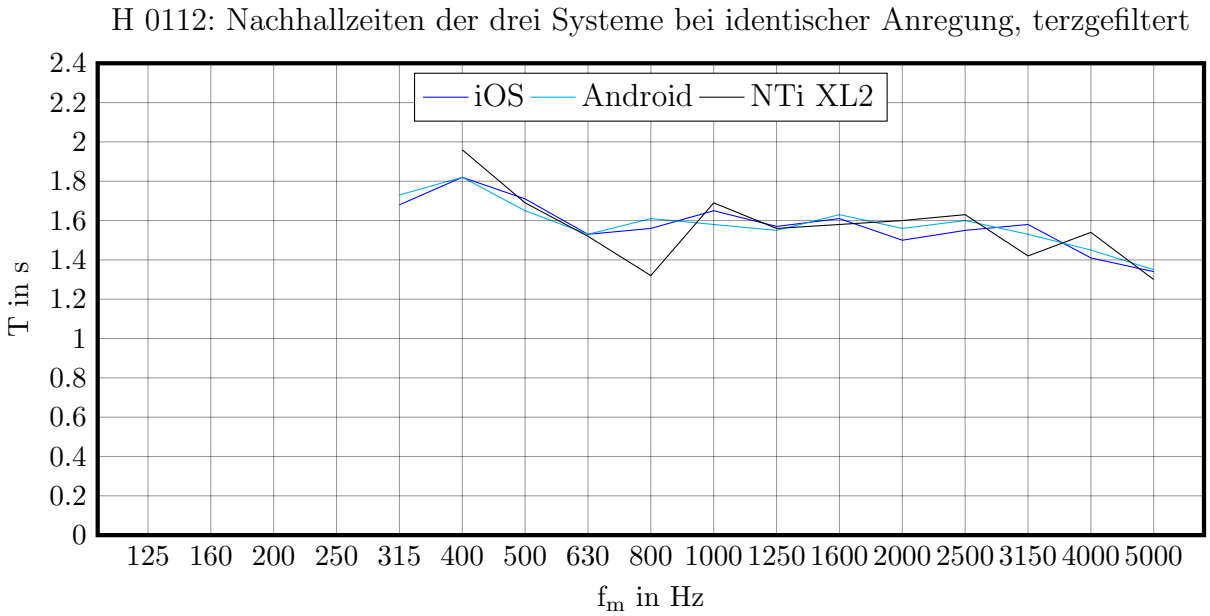


Abbildung 9.7.: Nachhallzeiten im H 0112, terzgefiltert

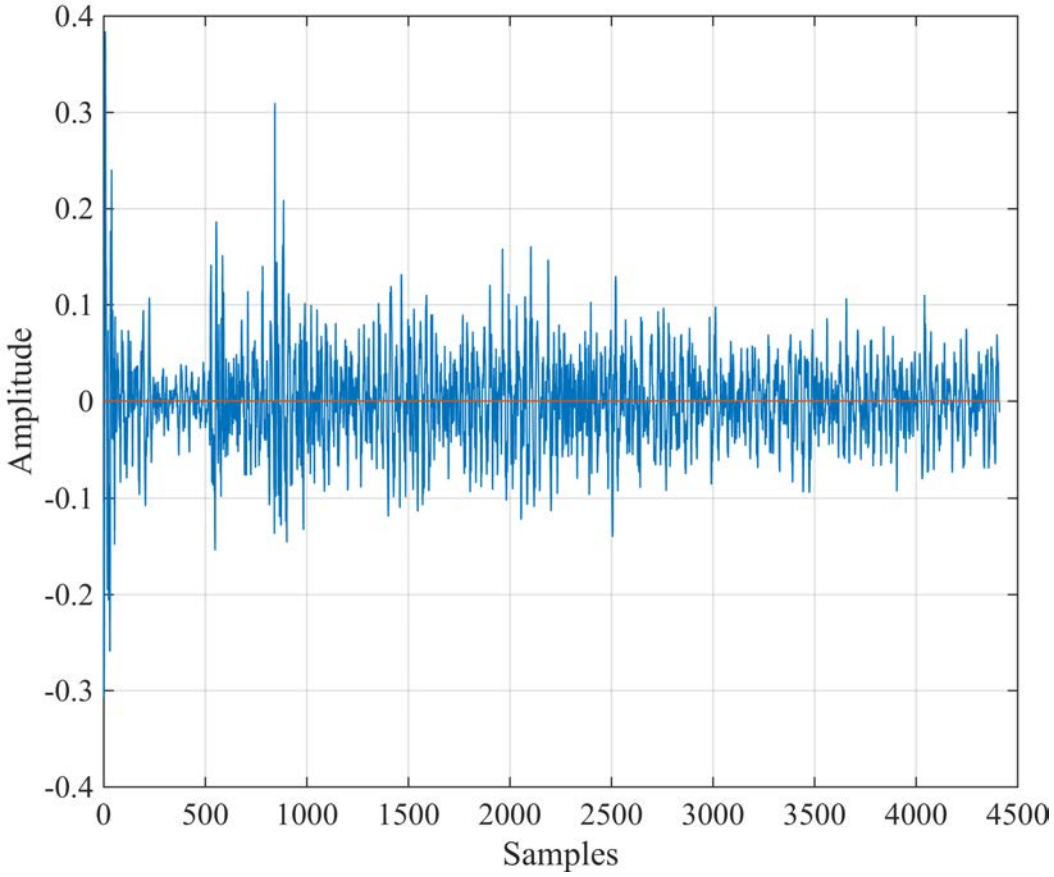


Abbildung 9.8.: Android Impulsantwort mit externem Audio-Interface, unverfälscht

10. Evaluation des Gesamtsystems

R. Burgmayer, A. Rosenkranz

Dieses Kapitel dient der Evaluation des Gesamtsystems, bestehend aus Applikation, Smart Device und dem micW i436 Messmikrofon [60]. Beispielhaft wird für das iOS-System ein iPhone 5s (Apple [75]) und für das Android-System ein Samsung Galaxy S6 (Samsung [76]) mit jeweils zum Zeitpunkt der Messung aktuellem Betriebssystem (iOS 10, Android 7) betrachtet. Da das Android System, wie im vorangegangenen Kapitel 9 beschrieben, für die Verwendung in Form des angestrebten Gesamtsystems ungeeignet ist, wird der dort vorgeschlagene Lösungsweg unter Verwendung eines USB-Audio-Interfaces gewählt, um die Applikation für Android zu evaluieren.

10.1. Versuchsbeschreibung

Für die Evaluation des gesamten Systems werden die Hörsäle H 0112 und MA 005 der Technischen Universität Berlin durch eine impulsartige Signalquelle in Form einer Starterklappe angeregt. Die sich in den angeregten Räumen ergebenden Signalverläufe werden sowohl mit einem iOS- als auch einem Android-Testgerät aufgezeichnet. Aus den Impulsantworten werden die Nachhallzeiten nach der T_{20}-Methode in Oktav- und Terzbändern in den nach DIN 3382-1 [1] relevanten Frequenzbereichen errechnet. Parallel werden zum gleichen Anregungssignal ebenfalls auf T_{20} basierende Nachhallzeiten mit dem NTi XL2 Messgerät [62] bestimmt. Die Messwerte des NTi XL2 dienen dabei als Referenz. Die errechneten Nachhallzeiten werden verglichen und die Abweichungen im Nachhinein diskutiert.

Beim NTi XL2 handelt es sich um ein kommerzielles Messsystem der Firma NTi Audio [62]. Es wird als Referenz für die Vergleichsmessung gewählt, da es sich dabei um ein präzises, professionell eingesetztes mobiles Messgerät handelt, das als leistungsfähiger Schallpegelmesser und umfangreicher Akustik-Analysator weit verbreiteten Einsatz findet. Die zum Vergleich der Messsysteme herangezogene Nachhallzeitmessung erfolgt dabei im NTi XL2 nahezu in Echtzeit und bietet eine gute und schnelle Möglichkeit zur Validierung der nachfolgenden Messungen. Zusätzlich bilden Mobilität, schnelle Einsetzbarkeit und die Orientierung an DIN 3382-1 [1] des NTi XL2 Systems drei der wesentlichen Faktoren, die auch für die Applikation angestrebt werden. In allen nachfolgenden Messungen wird das NTi XL2 in Kombination mit dem NTi M2230 Messmikrofon genutzt.

Technische Spezifikationen der verwendeten Mikrofone sind im Anhang A.4 und in größerem Detail in den angegebenen Quellen zu finden.

Für die Aufnahme selbst gelten die in Abschnitt 4.3 auf Seite 21 aufgezeigten Abweichungen. Zur Filterung der Impulsantworten werden die von Ralf Burgmayer [58] beschriebenen Oktav- und Terzbandfilter sechster Ordnung verwendet. Eine Start- und Endpunktbeschneidung (siehe Kapitel 5) findet vor der Nachhallzeitbestimmung statt.

Die Messsysteme werden in beiden Räumen etwa in der Raummitte positioniert. Bei der Positionierung der Mikrofone wird besonders darauf geachtet, die Membranen möglichst nahe aneinander zu platzieren um positionsbezogene Schallfeldunterschiede zu minimieren. Die Anregung findet tafelseitig vor der ersten Stuhlreihe mithilfe der Starterklappe statt. Bei Stimuli dieser Art gilt es, den beschränkten Abstrahlbereich, sowie eine geringe Energie im tiefen Frequenzbereich zu beachten (siehe Seetharaman und Tarzia [14]).

10.2. Ergebnisse

Die nachfolgenden Ergebnisse wurden vor den in Kapitel 5 erwähnten Anpassungen des Algorithmus zur Endpunktbeschneidung bestimmt. Größere Abweichungen insbesondere um das Frequenzband bei 500 Hz bei Oktavmessungen, sowie im Bereich von 315 bis 1000 Hz bei Terzmessungen sind zum Teil auf die frühere Implementierung zurückzuführen. Zur Übereinstimmung mit der Arbeit von Ralf Burgmayer werden diese Ergebnisse hier trotzdem diskutiert. Am Ende des Kapitels erfolgt eine gesonderte Betrachtung mit aktueller Implementierung des Algorithmus.

10.2.1. Hörsaal H 0112

Abbildungen 10.1 und 10.2 zeigen die Ergebnisse der Nachhallzeitmessung im Hörsaal H 0112. In Abbildung 10.1 sind die entsprechenden Ergebnisse in Oktavbändern dargestellt. Für die Frequenzbänder $f_m = 125\,\text{Hz}$ und $f_m = 250\,\text{Hz}$ ist aufgrund eines zu geringen Signal-Rausch-Verhältnisses keine zu DIN 3382-1 [1] konforme Nachhallzeitbestimmung möglich. Der geringe Signal-Rausch-Abstand in den tiefen Frequenzbändern ist bedingt durch den Störgeräuschpegel, so wie die impulsartige Signalquelle. In den restlichen Frequenzbändern erzielen beide Testgeräte sehr gute Übereinstimmungen der Ergebnisse im Vergleich mit dem Referenzgerät. Die maximale Abweichung liegt für iOS bei 4,38 % und für Android bei 5,71 %. Abbildung 10.2 stellt die Ergebnisse der Terzbandrechnung dar. Auch im terzgefilterten Signal können aufgrund des beschränkten Signal-Rausch-Verhältnisses erst Ergebnisse ab $f_m = 400\,\text{Hz}$ verglichen werden, wobei sowohl iOS- als auch Android-Testgerät bereits im Terzband $f_m = 315\,\text{Hz}$ Ergebnisse liefern. Im Vergleich zur Oktavbandmessung sind die Abweichungen für Terzen wesentlich größer. Bezieht man das Terzband $f_m = 315\,\text{Hz}$ aufgrund des geringen Signal-Rausch-Abstandes nicht in die Evaluation mit ein, so liegt die maximale Abweichung für iOS bei 28,03 % und für Android bei 21,97 %. Bis auf das Terzband mit $f_m = 800\,\text{Hz}$ liegen in allen Frequenzbändern Ergebnisse mit sehr geringer Abweichung vor. Für das Android-Testgerät liegen bis auf das genannte Band alle weiteren Abweichungen unter 10 %, für das iOS-Gerät liegen in den Frequenzbändern 500 Hz mit 14,79 % und 3150 Hz mit 13,38 % noch zwei weitere Abweichungen über 10 % vor.

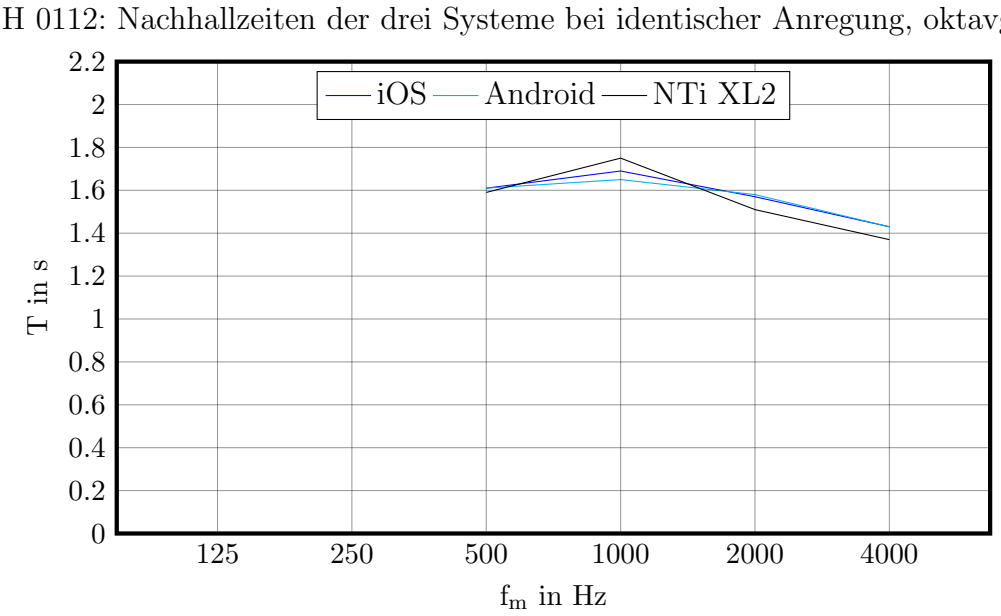


Abbildung 10.1.: Nachhallzeiten im H 0112, oktavgefiltert.

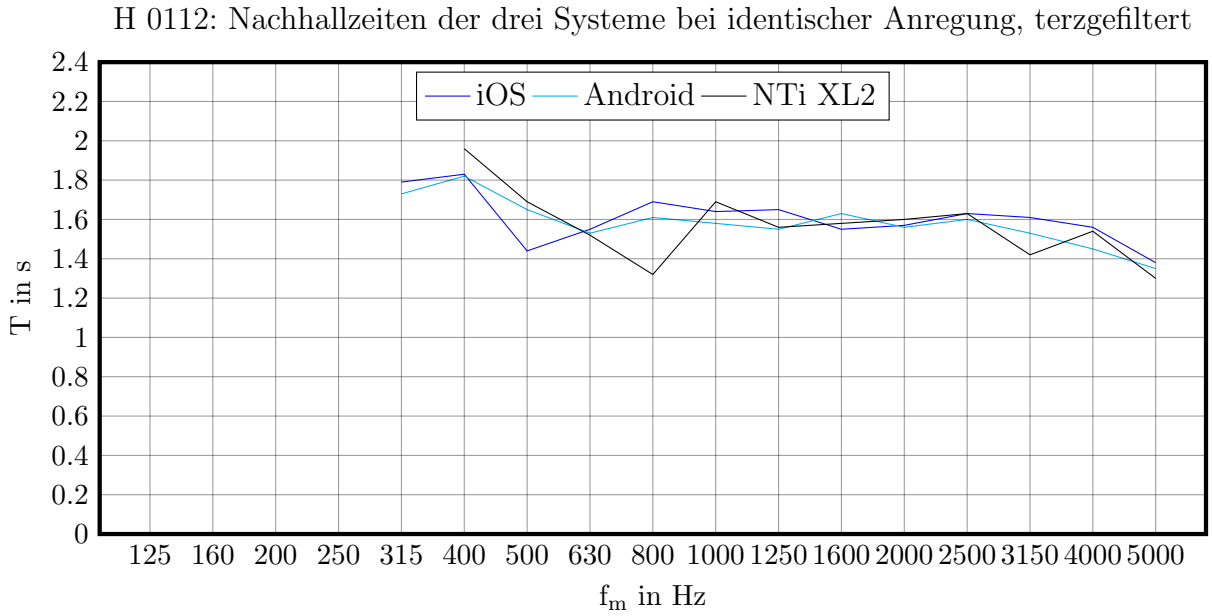


Abbildung 10.2.: Nachhallzeiten im H 0112, terzgefiltert

10.2.2. Hörsaal MA 005

Abbildungen 10.3 und 10.4 zeigen die Ergebnisse für den Hörsaal MA 005. Abbildung 10.3 stellt die in Oktavbändern gemessenen Nachhallzeiten dar. Aufgrund des geringen Signal-Rausch-Verhältnisses ist die Nachhallzeitbestimmung erneut erst für Oktavbänder ab $f_m = 500\,\text{Hz}$ möglich. Sowohl die mit iOS als auch Android bestimmten Nachhallzeiten sind in guter Übereinstimmung mit dem Referenzgerät. Die maximale Abweichung liegt für iOS bei 5,69 % und für Android bei 3,76 %. Abbildung 10.4 zeigt die Nachhallzeiten in Terzbändern. Bei dieser Messung ist der Signal-Rausch-Abstand

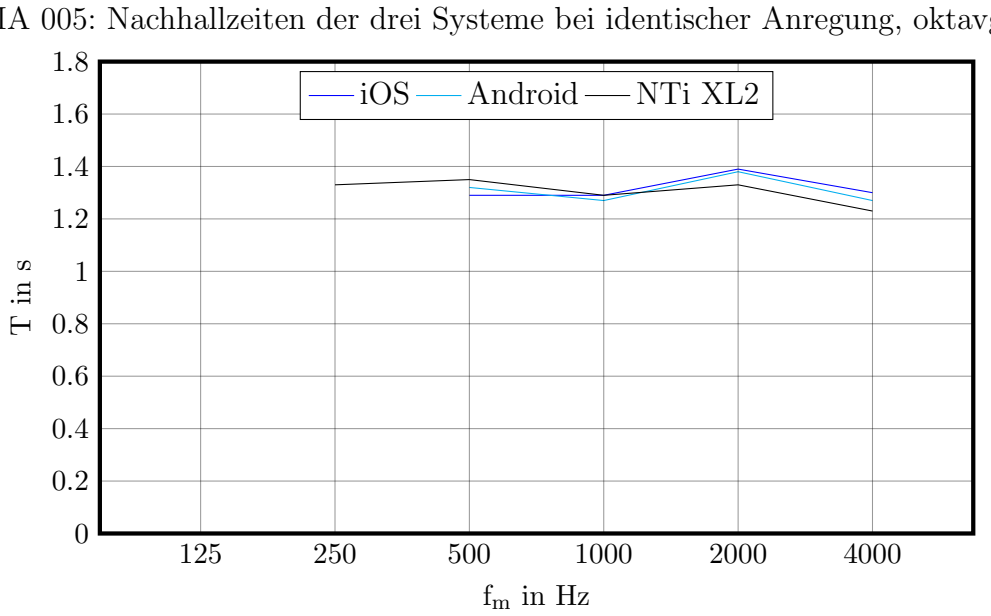


Abbildung 10.3.: Nachhallzeiten im MA 005, oktavgefiltert.

für alle drei Messgeräte erst im Terzband mit Mittenfrequenz $f_m = 400\,\text{Hz}$ ausreichend. In den Terzbändern $f_m = 500\,\text{Hz}$ und $f_m = 630\,\text{Hz}$ war der Signal-Rausch-Abstand wiederum für das NTi XL2 nicht ausreichend. In den übrigen Frequenzbändern erzielt die mobile Applikation auf beiden Systemen wiederum Ergebnisse mit guter Übereinstimmung im Vergleich zur Referenz. Unter Ausschluss der Terzbänder bei 500 Hz und 630 Hz beträgt die maximale Abweichung für iOS 10,48 % bei $f_m = 5000\,\text{Hz}$ und für Android 10,67 % bei $f_m = 2500\,\text{Hz}$. Für das iOS- und das Android-Gerät liegen alle weiteren Abweichungen zwischen 0 % und 8,4 %. Die Abweichungen zwischen den iOS- und Android-Ergebnissen sind dabei auf die Verwendung der unterschiedlichen Mikrofone zurückzuführen. Detaillierte Messwerttabellen sind im Anhang B.4 zu finden.

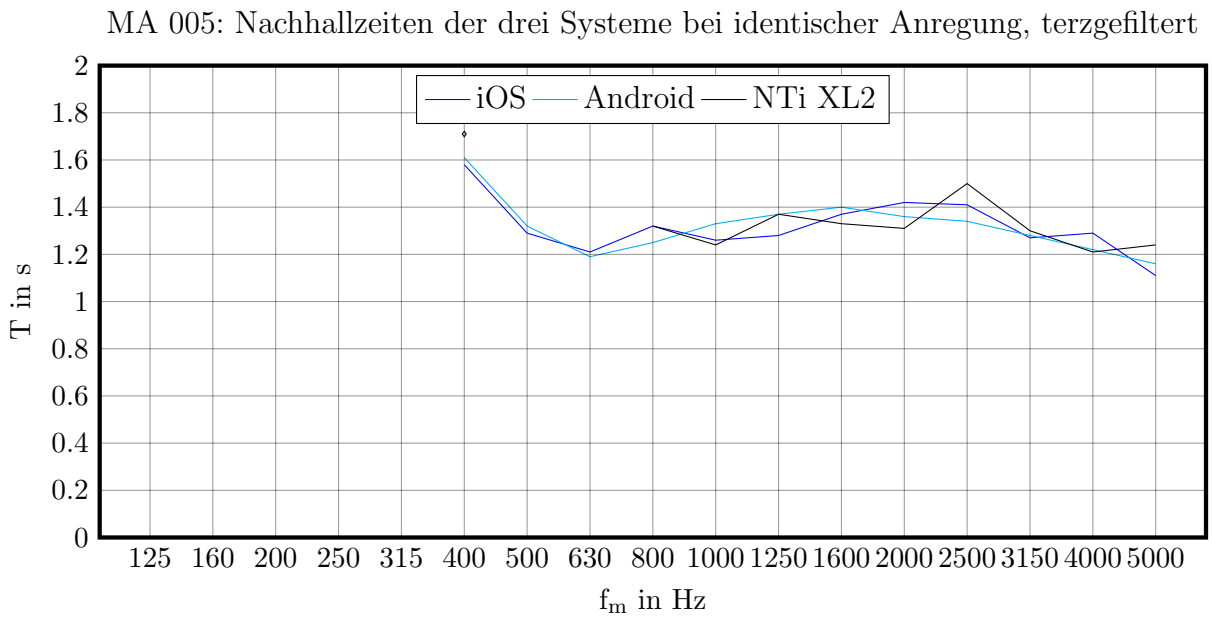


Abbildung 10.4.: Nachhallzeiten im MA 005, terzgefiltert

10.3. Zusammenfassung und Diskussion der Ergebnisse

Der geringe Signal-Rausch-Abstand bei den Messungen stellt das Hauptproblem der Evaluation dar. Vor allem der Frequenzbereich unterhalb von 500 Hz stellt sich als problematisch heraus. Dieser Umstand resultiert hauptsächlich aus der impulsartigen Raumanregung durch die Starterklappe. Diese Form der Anregung wurde gewählt, um die Anwendung der Applikation im realen Messfall mit der mobilsten Form der Anregung zu untersuchen. Wie bei der Evaluation der Signalverarbeitung nehmen die Abweichungen der Ergebnisse im Vergleich zur Referenz mit steigender Anzahl an Frequenzbändern pro Oktave aufgrund unterschiedlicher Filteralgorithmen zu.

Für Oktavbänder werden im Vergleich zur Referenz für beide Räume sehr ähnliche Ergebnisse erzielt. Die maximale Abweichung ist stets geringer als 6 %. Die maximalen Abweichungen für die Evaluation der Terzbänder sind unter Betrachtung der Ergebnisse der Evaluation der Signalverarbeitung ebenfalls tolerabel. So beträgt die maximale Abweichung bei Verwendung identischer Impulsantworten des AARAE-Systems (Carbrera et al. [77]) zur ITA-Toolbox für die Nachhallzeitbestimmung in Terzbändern bereits 13 %. Bezieht man noch die unterschiedlichen Messmikrofone, leicht verschiedene Mikrofonpositionen sowie den geringen Signal-Rausch-Abstand mit ein, lässt sich erkennen, dass Abweichungen dieser Größenordnung durchaus vorkommen können. Ferner kann die im Referenzgerät anders realisierte Filterbank und Signalverarbeitung zu weiteren Abweichungen führen. Die Filterevaluation in der Arbeit von Ralf Burgmayer [58] zeigt, wie sehr sich unterschiedliche Realisierungen der Filterbank auf die Nachhallzeitbestimmung auswirken. Im Messbetrieb zeigten sich auch Unsicherheiten in den Ergebnissen des Referenzsystems. Im Verlauf der Messungen sind die teils starken Schwankungen des NTi-Messsystems in der Nachhallzeitbestimmung aufgefallen, während die mobile Applikation stets Ergebnisse mit geringer Streuung lieferte. Vergleicht man die Abweichungen in beiden Räumen, so fällt auf, dass die Abweichungen

im Hörsaal MA 005 größer als im Hörsaal H 0112 sind. Die vergleichsweise höheren Messfehler resultieren aus höheren Lärmpegeln im Hörsaal MA 005.

Es lässt sich festhalten, dass die Applikation in Kombination mit einem externen Messmikrofon zur verlässlichen Bestimmung der Nachhallzeit geeignet ist. Die Abweichungen in einigen wenigen Terzbändern können unter Annahme funktionierender Algorithmen auf eine Kombination aus unterschiedlichen Algorithmen des NTi XL2, schlechten Messbedingungen, sowie unterschiedlichen Messmikrofonen zurückgeführt werden. Dies bestätigt sich vor allem bei Betrachtung der Studien von Carbrera [19] und Tronchin [20], die Abweichungen in ähnlicher Größenordnung bei bloßer Evaluation der Signalverarbeitung beobachten. Die in Abschnitt 4.3 beschriebenen Abweichungen der Aufnahme mit einem Einfluss auf die Nachhallzeitbestimmung von knapp unter 3 % im relevanten Regressionsbereich sollten ebenfalls als zusätzlicher Faktor miteinbezogen werden.

10.4. Messergebnisse nach Optimierung der Endpunktbeschneidung

In diesem Abschnitt findet die Betrachtung von Messergebnissen nach der in Kapitel 5 besprochenen Anpassung des Algorithmus zur automatisierten Endpunktbeschneidung statt.

Abbildungen 10.5 und 10.6 zeigen die Ergebnisse einer weiteren Vergleichsmessung zur Nachhallzeitbestimmung mit optimierter Endpunktbeschneidung. Die angegebenen Fehlerbalken entsprechen den nach DIN 3382-1 [1] bestimmten Standardabweichungen für die Messung von Nachhallzeiten nach der T_{20}-Methode in den jeweiligen Frequenzbändern. Zugehörige Messwerttabellen sind im Anhang B.5 zu finden. Neben den maximalen Abweichungen im Oktavband von 500 Hz von 6,77 % für das iOS-Gerät und 7,52 % für das Android-Gerät sind bei den in Abbildung 10.5 dargestellten Messergebnisse keine großen Unterschiede zu den zuvor beschriebenen Ergebnissen festzustellen. Eine deutliche Verbesserung ist in Abbildung 10.6 in den Terzbändern ab 400 Hz zu erkennen. Während zuvor zum Teil Unterschiede von nahezu 30 % in den Nachhallzeiten der Terzbänder zu verzeichnen waren, liegen die maximalen Differenzen zum Referenzgerät nach Optimierung bei maximal 12,59 %. Diese maximalen Abweichungen liegen dabei hauptsächlich im niedrigen Frequenzbereich. Die Messergebnisse aller drei Messgeräte befinden sich dabei im gegenseitig dargestellten Toleranzbereich der Standardabweichungen in den jeweiligen Frequenzbändern.

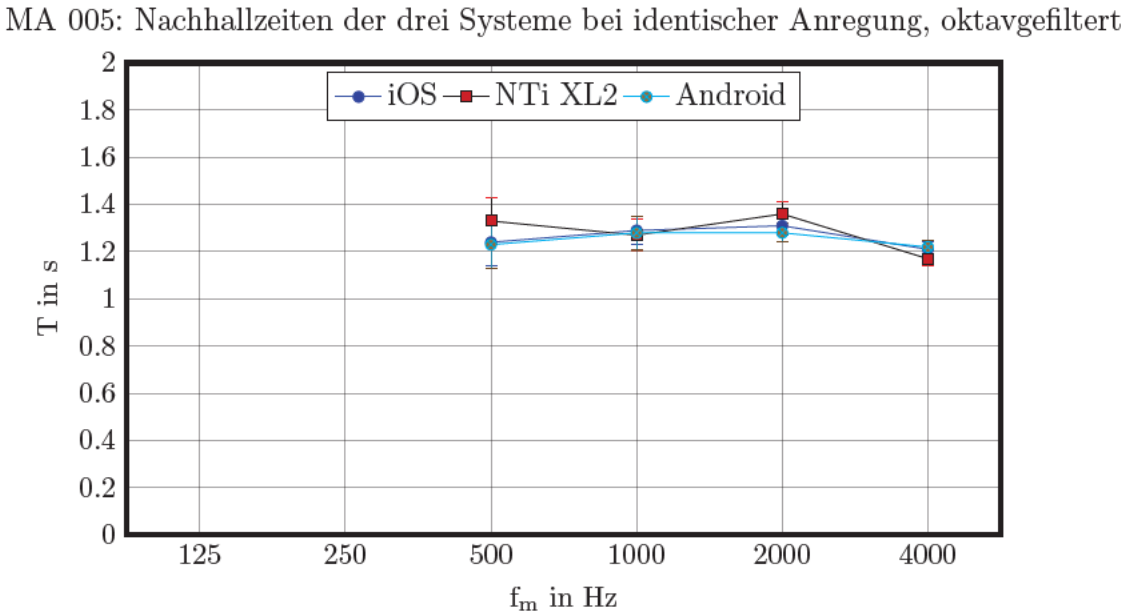


Abbildung 10.5.: Nachhallzeiten im MA 005, oktavgefiltert

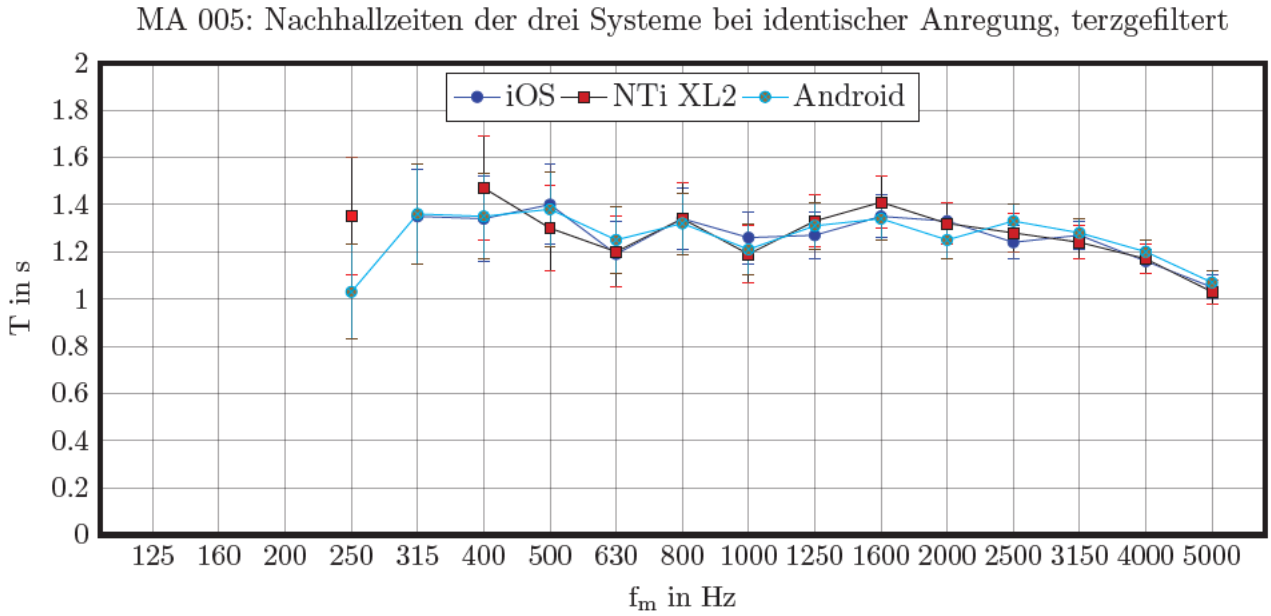


Abbildung 10.6.: Nachhallzeiten im MA 005, terzgefiltert

11. Fazit und Ausblick

A. Rosenkranz

11.1. Fazit

Nach Evaluation der implementierten Algorithmen zur Aufnahme und Signalverarbeitung stellt die entwickelte Applikation ein fähiges Mittel zur Verwendung eines Smart Devices mit externem Messmikrofon als zuverlässiges normkonformes raumakustisches Messgerät dar. Die Einschränkungen, die mit der für die Evaluation genutzten Anregung durch eine Starterklappe einhergehen, können durch die Nutzung von Anregungen wie Ballonplatzen oder Pistolenschuss ausgeglichen werden. Diese Anregungen können auch in den niedrigen Frequenzbereichen ausreichend Energie liefern, um eine Analyse über das gesamte zu betrachtende Frequenzspektrum zu ermöglichen. Mit den in Kapitel 6 besprochenen Mitteln der plattformübergreifenden Entwicklung, ist es gelungen, eine responsive und gut bedienbare Applikation zu entwickeln. Sie genügt allen Ansprüchen, die vor der Entwicklung an sie gestellt wurden. Die Einschränkungen der Android Plattform konnten erst nach intensiver Recherche in ihrem vollen Ausmaß gefasst werden. Sie erschweren die Nutzung der Applikation mit Android-Geräten in der gleichen mobilen Weise, wie es für iOS-Geräte möglich ist. Allerdings lieferte die Recherche über die Limitationen des Android-Systems die Möglichkeit, auch externe Interfaces und in diesem Zusammenhang genauere Messmikrofone mit der Applikation zu testen und so auf erweiterte Anwendungsmöglichkeiten schließen zu können. Über die gesamte Anzahl der Vergleichsmessungen zur Evaluation des Gesamtsystems lieferte die implementierte Applikation Ergebnisse, die eine geringere Streuung als die Ergebnisse des Referenzsystems aufwiesen.

Während der Implementierung der Applikation traten immer wieder Entwicklungshürden auf, die teilweise trotz gründlicher vorangehender Recherche im Voraus nicht überblickt werden konnten. Neben der bereits zuvor bekannten Einsträngigkeit von JavaScript als Basissprache, die mit Hilfe der Web Worker (siehe Kapitel 7, Abschnitt 7.2, Seite 43) umgangen werden konnte, stellten auch die Web Views mit ihrer Verwandtschaft zu Web Browsern immer wieder kleine Hindernisse dar. So mussten beispielsweise eigene Index-Suchfunktionen implementiert werden, um das sogenannte „Stack Size Limit“ der Browser zu umgehen. Diese Begrenzung erlaubt dem Browser oder Web View nur eine bestimmte Menge an rekursiven Operationen und führte zu Problemen bei der Suche nach der maximalen Amplitude in langen Impulsantworten. Im Verlauf der Implementierung fand auch ein Wechsel der Entwicklungsumgebung Ionic auf Version 2.0 statt. Dieser Wechsel machte die Verwendung von TypeScript und Angular 2.0 notwendig, da mit dem finalen Wechsel auf Ionic 2.0 die Möglichkeit entfiel, Applikationen vollständig in JavaScript zu implementieren. Um trotzdem auf

die Vorteile von Ionic 2.0, wie einen enormen Geschwindigkeitsgewinn und modernere grafische Oberflächen, zurückgreifen zu können, war es notwendig, auch TypeScript und Angular in die Entwicklung mit einzubeziehen. So mussten nach Fertigstellung der Applikation in ihrer Testversion zur Ergänzung einer ansprechenden Benutzungsoberfläche zusätzlich TypeScript und Angular 2.0 eingebunden werden. Dies brachte jedoch auch einen großen Vorteil in der Fehlersuche während des Entwicklungsprozesses mit sich. Da es sich bei JavaScript um eine Skriptsprache handelt, die erst während der Laufzeit kompiliert wird, wurden eventuelle Fehler erst während der Programmlaufzeit sichtbar. TypeScript bietet die Möglichkeit, Fehler im Programmcode schon vor der Übersetzung in JavaScript sichtbar zu machen und erleichterte somit die weitere Programmierung. Zusätzlich beinhaltet TypeScript ein strengeres System von Variablentypen, dass die teilweise unübersichtliche Konvertierung von Datentypen, wie sie in JavaScript stattfindet, besser überwacht.

Das Ziel, auf native Programmierung zu verzichten und die Applikation vollständig in der gemeinsamen Codebasis JavaScript zu implementieren, konnte bis auf eine kleine Korrektur in nativer Sprache des iOS-Systems erreicht werden. Um Kontrolle über den Mikrofonvorverstärker von iOS zu erlangen, musste das Plugin zur Audioaufnahme in Objective-C erweitert werden. Die vorgenommenen Anpassungen sind im Anhang A.6 aufgezeigt.

Die größte Herausforderung stellten die in Kapitel 9 besprochenen Limitationen des Android-Betriebssystems dar. Als dem System mit Android 7.0 die Möglichkeit des Zugriffs auf unbearbeitete Audiodaten gegeben wurde, schien es zunächst möglich, auch Android-Geräte für akustische Messungen nutzbar zu machen. Erst bei der nativen Nachrüstung der Vorverstärkerkontrolle für das iOS-System fiel auf, dass Android einen solchen Zugriff für Entwickler nicht vorgesehen hat. Auch die den Android-Systementwicklern überlassene Implementierung für den Zugriff auf unbearbeitete Audiodaten erschwerte die Möglichkeit, einen allgemeingültigen Lösungsweg für das System zu finden. Eine aufwendige Recherche schloss sich diesen Erkenntnissen an. Die verlässliche Messung mit Android Geräten erforderte letztendlich die Nutzung zusätzlicher externer Hardware und bedeutete somit einen großen Verlust in Bezug auf die Mobilität des Android Systems.

11.2. Ausblick

11.2.1. Zur Nutzung interner Mikrofone

Im Zusammenhang mit der Evaluation der Applikation wurden auch Messungen ausgeführt, bei denen die Ergebnisse der Nachhallzeitbestimmung unter Verwendung der internen Mikrofone untersucht wurden. Der Messaufbau glich dem im zuvor beschriebenen Kapitel, die Mikrofone der Smart Devices waren dabei zur Quelle des Anregesignals gerichtet. Besonders in den Frequenzbändern unter 1000 Hz für Oktavbandmessungen und unter 800 Hz für Terzbandmessungen waren Unterschiede von bis zu 31,88 % für das iOS- und bis zu 49,28 % für das Android-Gerät zu verzeichnen. Dies lässt vermuten, dass die internen Mikrofone neben ihrer oft nicht omnidirektionalen

Richtcharakteristik auch andere Defizite aufweisen, die sie zur genauen Bestimmung raumakustischer Gegebenheiten ungeeignet machen. Weitere Abweichungen im Fall des Android-Gerätes treten durch die in Kapitel 9 beschriebenen Limitationen des Android Gerätes auf. Oberhalb von 500 Hz können die Ergebnisse jedoch durchaus als grobe Orientierung für die Nachhallzeit eines Raumes genutzt werden. Die Messergebnisse sind beispielhaft im Anhang B.2 tabellarisiert. Insgesamt erfordert die Nutzung der internen Mikrofone jedoch eine genauere Untersuchung des jeweiligen verwendeten Gerätes.

Interessante Verwendungsmöglichkeiten im Bereich der Akustik könnten sich aus der Verwendung der Gesamtanzahl der im Smart Device verbauten Mikrofone ergeben, da viele moderne Smartphones bereits über mindestens zwei interne Mikrofone verfügen.

11.2.2. Potenzielle Erweiterungsmöglichkeiten der Applikation

In diesem Abschnitt wird auf Ausbaumöglichkeiten des Leistungsumfangs der Applikation eingegangen.

Weitere Anregesignale

Die Erweiterung der Applikation um zusätzliche Formen der Anregung stellt eine gute Möglichkeit dar, ihren Anwendungsbereich und ihre Genauigkeit zu erhöhen. Während eine Erweiterung um die Anregung durch abklingendes Rauschen technisch leicht realisiert werden kann, würden sich keine merklichen Vorteile durch die Implementierung dieser Anregungsform ergeben. Eine Erweiterung um die Anregung durch die in Kapitel 3 im Abschnitt 3.1.4 erwähnten Gleitsinus Signale würde durch die Möglichkeit der Verbesserung des Signal-Rausch-Verhältnisses einen großen Vorteil in der Genauigkeit der bestimmten Parameter bedeuten. Diese Genauigkeit geht allerdings auf Kosten der Mobilität, da externe Audiotechnik zum Senden des Anregesignals benötigt wird. Hierbei bietet sich die in Kapitel 9 angesprochene Verwendung eines externen USB Audio-Interfaces an.

Erweiterung um Kalibrierung

Eine Erweiterung der Applikation um die Kalibrierung des Messmikrofons ermöglicht den Einsatz des Messsystems zur Bewertung von Schallpegeln. Zusätzlich bietet eine Kalibrierung des Systems die Möglichkeit, auch die Maskierungseffekte des Sprachübertrangsindexes zu bestimmen.

Erweiterte Nutzung der grafischen Darstellung

Die großen Displays und Interaktionsmöglichkeiten moderner Smart Devices sind gut geeignet, um Messergebnisse oder Impulsantworten grafisch und interaktiv darstellen zu können. Visualisierungen der Impulsantwort als Pegelverlauf oder in Form eines Wasserfalldiagramms könnten für spezifische Anwendungsfälle wie die Studioakustik oder allgemein zur Frequenzanalyse genutzt werden.

Literaturverzeichnis

Normen und Weisungen

[1] Deutsches Institut für Normung (2009): *DIN EN ISO 3382-1: Akustik – Messung von Parametern der Raumakustik – Teil 1: Aufführungsräume.* Berlin: Beuth Verlag.

[2] Deutsches Institut für Normung (2008): *DIN EN ISO 3382-2: Akustik – Messung von Parametern der Raumakustik – Teil 2: Nachhallzeit in gewöhnlichen Räumen.* Berlin: Beuth Verlag.

[3] Deutsches Institut für Normung (2016): *DIN 18041:2004-05: Hörsamkeit in kleinen bis mittelgroßen Räumen - Anforderungen, Empfehlungen und Hinweise für die Planung.* Berlin: Beuth Verlag.

[4] Deutsches Institut für Normung (2012): *DIN EN 60268-16: Elektroakustische Geräte – Teil 16: Objektive Bewertung der Sprachverständlichkeit durch den Sprachübertragungsindex.* Berlin: Beuth Verlag.

[5] Deutsches Institut für Normung (2011): *DIN EN ISO 9241-210:2010: Ergonomie der Mensch-System-Interaktion – Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme.* Berlin: Beuth Verlag.

[6] Deutsches Institut für Normung (2014): *Elektroakustik - Schallpegelmesser - Teil 1: Anforderungen (IEC 61672-1:2013); Deutsche Fassung EN 61672-1:2013.* Berlin: Beuth Verlag.

Monografien und Sammelbänder

[7] Ahnert, W. und H.-P. Tennhardt (2008): „Raumakustik." In: Stefan Weinzierl (Hrg.) *Handbuch der Audiotechnik*, Kap. 5. Berlin, Heidelberg, New York: Springer, S. 182–263.

[8] Ahnert, W. und S. Feistel (2010): „Einmessung und Verifizierung raumakustischer Gegebenheiten von Beschallungsanlagen." In: M. Möser (Hrg.) *Messtechnik der Akustik*, Kap. 3. Berlin, Heidelberg, London, New York: Springer, S. 115–183.

[9] Müller, S (2008): „Messtechnik." In: Stefan Weinzierl (Hrg.) *Handbuch der Audiotechnik*, Kap. 21. Berlin, Heidelberg, New York: Springer, S. 1087–1170.

[10] Tanenbaum, A.S. und H. Bos (2015): *Modern Operating Systems.* 4. New Jersey: Pearson Education, Inc.

[11] Heitkötter, H.; S. Hanschke und T.A. Majchrzak (2012): „Comparing cross-plattform development approaches for mobile applications." In: *Lecture Notes in Business Information Processing*, vol. 140. Berlin, Heidelberg, New York: Springer, S. 120–138.

[12] Green, I. (2012): *Web Workers*. Sebastopol: O'Reilly Media, Inc. URL `http://my.safaribooksonline.com`.

[13] Bayer, R. (1972): „Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms." In: *Acta Informatica*, 1 S. 290–306.

Artikel

[14] Seetharaman, P. und S. P. Tarzia (2012): „The Hand Clap as an Impulse Source for Measuring Room Acoustics." In: *Audio Engineering Society Convention 132*. URL `http://www.aes.org/e-lib/browse.cfm?elib=16223`.

[15] Bello, J.P. et al. (2005): „A Tutorial on Onset Detection in Music Signals." In: *IEEE Transactions on Speech and Audio Processing*, 13(5) S. 1035–1047.

[16] Karjalainen, M.; Poju Antsalo und Timo Peltonen (2002): „Estimation of modal decay parameters from noisy response measurements." In: *Journal of the Audio Engineering Society*, 50 S. 867–878.

[17] Guski, M. und M. Vorländer (2013): „Noise compensation methods for room acoustical parameter evaluation." In: *International Symposium on Room Acoustics*.

[18] Lundeby, A.; T. E. Vigran; H. Bietz und M. Vorländer (1995): „Uncertainties of measurements in room acoustics." In: *Acta Acustica united with Acustica*, 81 S. 344–355.

[19] Carbrera, D.; J. Xun und M. Guski (2016): „Calculating Reverberation Time from Impulse Responses: A Comparison of Software Implementations." In: *Acoustics Australia*, 44(2) S. 369–378.

[20] Tronchin, L. (2013): „On the Uncertainties of Calculations of Acoustical Parameters in Room Acoustics Measurements." In: *Acoustical Society of America*.

Internet Ressourcen

[21] Murphy, D. und S. Shelley (2014): „Chiesa die San Biagio Calimera." Online: `http://www.openairlib.net/auralizationdb/content/chiesa-di-san-biagio-calimera`. Zugriff: 06.08.2018.

[22] Murphy, D. und S. Shelley (2012): „Cafeteria Universidad San Buenaventure Bogota." Online: `http://www.openairlib.net/auralizationdb/content/cafeteria-universidad-san-buenaventura-bogot%C3%A1`. Zugriff: 06.08.2018.

[23] Murphy, D. und S. Shelley (2012): „Chiesa di San Lorenzo Parco Archeologico di Apigliano Martano." Online: http://www.openairlib.net/auralizationdb/content/chiesa-di-san-lorenzo-%E2%80%93-parco-archeologico-di-apigliano-martano. Zugriff: 06.08.2018.

[24] Google (2017): „Develop Apps | Android Developers." Online: https://developer.android.com/develop/index.html. Zugriff: 06.08.2018.

[25] Apple (2017): „Develop - Bring Your Ideas to Life." Online: https://developer.apple.com/develop/. Zugriff: 06.08.2018.

[26] Mozilla Developer Network (2017): „MDN Web Docs: Navigator.getUserMedia()." Online: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/getUserMedia. Zugriff: 06.08.2018.

[27] Facebook (2017): „React Native - Learn once, write anywhere: Build mobile apps with React." Online: https://facebook.github.io/react-native/. Zugriff: 06.08.2018.

[28] Progress Software Corporation (2017): „NativeScript: Build amazing iOS and Android apps with technology you already know." Online: https://www.nativescript.org/. Zugriff: 06.08.2018.

[29] Drifty Co. (2017): „Ionic: The top open source framework for building amazing mobile apps." Online: http://ionicframework.com/. Zugriff: 06.08.2018.

[30] GitHub Inc. (2017): „GitHub - The world's leading software development platform." URL https://github.com/.

[31] Stack Exchange Inc. (2017): „Stack Overflow - Where Developers Learn, Share, & Build Careers." URL https://stackoverflow.com/.

[32] Massart, Frédéric (2016): „React Native vs Ionic: A Side-by-Side Comparison." Online: https://www.codementor.io/fmcorz/react-native-vs-ionic-du1087rsw. Zugriff: 06.08.2018.

[33] Aggarwal, Ankush (2017): „Ionic vs React Native." Online: https://medium.com/@ankushaggarwal/ionic-vs-react-native-3eb62f8943f8. Zugriff: 06.08.2018.

[34] Kumar, Ajay (2017): „React Native vs Ionic 2 framework: Side-by-Side Comparison." Online: http://www.techexceed.com/react-native-vs-ionic-2-framework/. Zugriff: 06.08.2018.

[35] Bouquet, Dash (2017): „Building an app: React Native vs Ionic." Online: https://hackernoon.com/is-it-possible-to-save-money-and-still-deliver-good-experience-to-a-user-45ea2bae1f83. Zugriff: 06.08.2018.

[36] Apple (2017): „Kernel Programming Guide." Online: https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/scheduler/scheduler.html. Zugriff: 06.08.2018.

[37] Apple (2017): „Threading Programming Guide." Online: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/CreatingThreads/CreatingThreads.html. Zugriff: 06.08.2018.

[38] Bird, T. (2011): „Android Kernel Versions." Online: http://elinux.org/Android_Kernel_Versions. Zugriff: 06.08.2018.

[39] Jones, M. (2009): „Inside the Linux 2.6 Completely Fair Scheduler." Online: https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/. Zugriff: 06.08.2018.

[40] Apple (2017): „Human Interface Guidelines." Online: https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/. Zugriff: 06.08.2018.

[41] Google (2017): „Up and running with material design." Online: https://developer.android.com/design/index.html. Zugriff: 06.08.2018.

[42] Google (2017): „Material Design - Principles." Online: https://material.io/guidelines/material-design/introduction.html#introduction-principles. Zugriff: 06.08.2018.

[43] Google (2017): „Android MediaRecorder." Online: https://developer.android.com/guide/topics/media/mediarecorder.html. Zugriff: 06.08.2018.

[44] Google (2017): „Android 8.0 Compatibility Definition: Raw Audio Capture." Online: https://source.android.com/compatibility/android-cdd#5_4_audio_recording. Zugriff: 06.08.2018.

[45] Google (2017): „MediaRecorder.Audiosource." Online: https://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html. Zugriff: 06.08.2018.

[46] Audio Evolution (2016): „Audio Evolution Manual: Android Limitations." Online: http://www.audio-evolution.com/manual/doku.php?id=android_limitations. Zugriff: 06.08.2018.

[47] Pfitzinger Voice Design (2016): „Audio Evolution Manual: Android Limitations." Online: http://www.pfitzingervoicedesign.com/field_recorder/aussteuerung.html. Zugriff: 06.08.2018.

[48] Google (2017): „Configuring Pre-Processing Effects." Online: https://source.android.com/devices/audio/implement-pre-processing. Zugriff: 06.08.2018.

[49] Google (2017): „AutomaticGainControl." Online: https://developer.android.com/reference/android/media/audiofx/AutomaticGainControl.html. Zugriff: 06.08.2018.

[50] Google (2017): „NoiseSupressor." Online: https://developer.android.com/reference/android/media/audiofx/NoiseSuppressor.html. Zugriff: 06.08.2018.

[51] Google (2017): „AcousticEchoCanceler." Online: https://developer.android.com/reference/android/media/audiofx/AcousticEchoCanceler.html. Zugriff: 06.08.2018.

[52] thejester123, XDA Developers (2016): „(How To / MOD) Microphone Gain Increase on All S5 Variants + Mod." Online: https://forum.xda-developers.com/galaxy-s5/development/how-to-microphone-gain-increase-s5-t3190613. Zugriff: 06.08.2018.

[53] SimpleMobDev (2011): „HD2 Mic Control." Online: https://play.google.com/store/apps/details?id=org.hd2.util.audio. Zugriff: 06.08.2018.

[54] XDA Developers (2017): „XDA Developers (Forum)." Online: https://forum.xda-developers.com. Zugriff: 06.08.2018.

[55] Google (2017): „UsbConstants." Online: https://developer.android.com/reference/android/hardware/usb/UsbConstants.html/index.html. Zugriff: 06.08.2018.

[56] Extream SD (2017): „USB audio driver in USB Audio Player/Recorder PRO and Audio Evolution Mobile." Online: http://www.extreamsd.com/index.php/technology/usb-audio-driver. Zugriff: 06.08.2018.

[57] edimuj (2018): „cordova-plugin-audioinput." URL https://github.com/edimuj/cordova-plugin-audioinput.

Abschlussarbeiten

[58] Ralf Burgmayer (2017): *Entwicklung einer mobilen Applikation zur Bestimmung raumakustischer Parameter*. Masterarbeit, Technische Universität, Fakultät 1, Fachgebiet Audiokommunikation, Berlin.

[59] Linus Öberg (2015): *Evaluation of Cross-Platform Mobile Development Tools: Development of an Evaluation Framework*. Masterarbeit, UmeåUniversity, Umeå.

Hard- und Software

[60] mic W (2013): „i436 Mini Ansteckmikrofon." URL http://www.mic-w.com.

[61] Behringer Music Group (2000): „Behringer ECM8000 Ultra-Linear Measurement Condenser Microphone." URL http://www.music-group.com/Categories/Behringer/Microphones/Condenser-Microphones/ECM8000/p/P0118.

[62] NTi Audio AG (2016): „NTi Audio XL2: Tragbarer Audio- und Akustik-Analysator." URL http://www.nti-audio.com.

[63] Vescom B.V. (2016): „Vescom Acoustics Check." Online: https://itunes.apple.com/US/app/id747745892?mt=8. Zugriff: 06.08.2018.

[64] Artnovion LDA (2017): „Impulso." Online: https://www.artnovion.com/impulso/acoustics. Zugriff: 06.08.2018.

[65] Faber Acoustical, LLC (2017): „RoomScope." Online: https://www.faberacoustical.com/apps/ios/roomscope/. Zugriff: 06.08.2018.

[66] Suono e Vita (2014): „APM Tool full use your smart phone to control your room acoustics." Online: http://www.suonoevita.it/en/apm_tool_full_use_your_smart_phone_control_your_room_acoustics. Zugriff: 06.08.2018.

[67] Studio Six Digital (2017): „Sound Balance Assistant." Online: http://studiosixdigital.com/. Zugriff: 06.08.2018.

[68] Seetharaman, P. und S. Tarzia (2017): „Clap IR." Online: https://github.com/starzia/ClapIR. Zugriff: 06.08.2018.

[69] Schosoft (2013): „RevMeter Pro." Online: http://www.schosoft.com/page5/page5.html. Zugriff: 06.08.2018.

[70] Kröber (2017): „Nachhallzeit." Online: https://play.google.com/store/apps/details?id=com.kroeber.reverberation_time. Zugriff: 06.08.2018.

[71] Kröber (2017): „Nachhallzeit Pro." Online: https://play.google.com/store/apps/details?id=com.kroeber.reverberation_time_pro. Zugriff: 06.08.2018.

[72] Dietrich, P. et al. (2012): „ITA- Toolbox - An Open Source MATLAB Toolbox for Acousticians." In: H. Hanselka (Hrg.) *Fortschritte der Akustik: DAGA 2012 Tagungsband*. Deutsche Gesellschaft für Akustik e.V. (DEGA), Deutsche Gesellschaft für Akustik e.V. (DEGA), S. 151–152.

[73] Microsoft (2017): „Xamarin: Everything you need to deliver great mobile apps." Online: https://www.xamarin.com/. Zugriff: 06.08.2018.

[74] Focusrite (2016): „Focusrite Scarlett 2i4 Audio-Interface mit USB-Stromversorgung." URL https://focusrite.de/usb-audio-interfaces/scarlett-2i4.

[75] Apple (2013): „iPhone 5s - Technische Daten." URL https://support.apple.com/kb/sp685?locale=de_DE.

[76] Samsung (2015): „Samsung Galaxy S6." URL https://www.samsung.com/de/smartphones/galaxy-s6-g920f/SM-G920FZWEDBT.

[77] Carbrera, D.; D. Jimenez und W.L. Martens (2014): „Audio and Acoustical Response Analysis Environment (AARAE): a tool to support education and research in acoustics." In: *inter.noise 2014*.

Anhang

A. Zusätzliche Informationen

A.1. Ordnerstruktur des digitalen Anhangs

Dieser Arbeit liegt ein Anhang in digitaler Form bei. Er enthält alle Daten des Ionic Projekts, eine Bedienungsanleitung für die Applikation, alle angegebenen Quellen, die keine Internetquellen sind, sowie sonstige zur Evaluation genutzten Daten. Eine digitale Kopie dieser Arbeit ist ebenfalls enthalten.

Abbildung A.1 zeigt die Ordnerstruktur des Anhangs.

```
-Anhang D                    <-- Digitaler Anhang
|-- -Algorithm Testing Environment <-- Testumgebung für Berechnungsalgorithmen
|-- -App Daten              <-- Ordner mit Applikation als Ionic Projekt
     |-- -roomAcoustics
|-- -Bedienungsanleitung     <-- Bedienungsanleitung der Applikation im PDF Format
|-- -Digitale Kopie          <-- Digitale Kopie dieser Arbeit
|-- -Evaluation Gesamtsystem<-- Messwerttabellen und Evaluationsdaten
|-- -Matlab Skripte          <-- Ordner mit Evaluationsskripten
|-- -Messwerte               <-- Hauptordner für App Dateien
|-- -Sortierte Quellen       <-- Quellen im PDF Format
```

Abbildung A.1.: Ordnerstruktur des digitalen Anhangs

A.2. Übersicht über die Applikationsstruktur

Abbildung A.2 zeigt eine Übersicht über die Module der Applikation, wie sie miteinander verknüpft sind und welche Aufgaben in ihnen abgearbeitet werden. Hier soll nur ein kurzer Überblick über die Funktionsweise der Applikation gegeben werden, genauere Informationen können dem Quelltext der Applikation entnommen werden. In Abbildung A.3 ist die Ordnerstruktur des Projektordners der Applikation dargestellt.

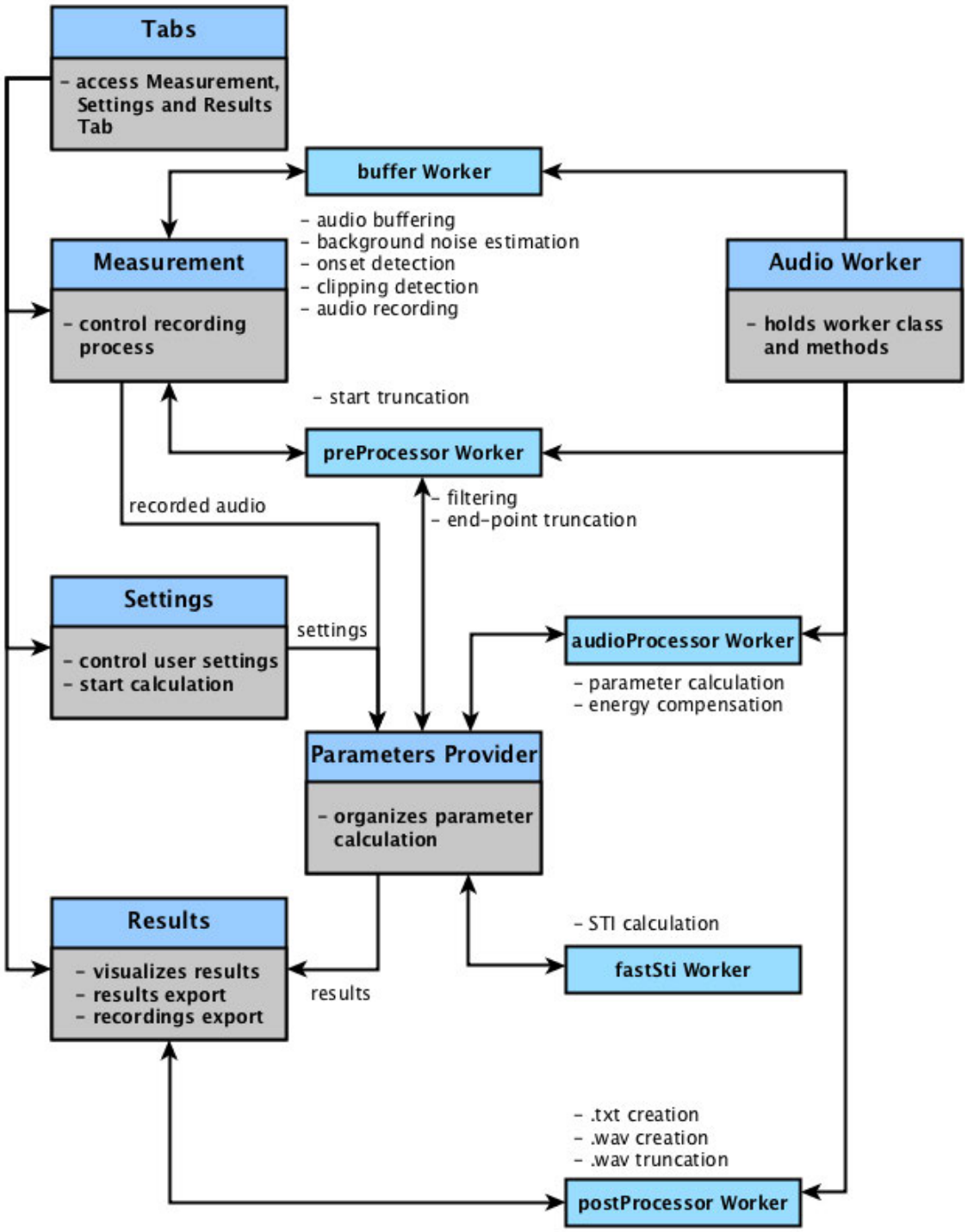


Abbildung A.2.: Struktur der Applikation

```
-roomAcoustics            <-- Projektordner
|-- -hooks
|-- -node_modules
|-- -platforms
|-- -plugins
|-- -resources
   |-- -android
      |-- -icon
      |-- -splash
   |-- -ios
      |-- -icon
      |-- -splash
|-- -src                   <-- Hauptordner für App Dateien
   |-- -app
   |-- -assets
      |-- -icon
      |-- -images
         |-- -overlay
      |-- -js
         |-- -helper
         |-- -workerfiles
   |-- -pages
      |-- -measurement
      |-- -results
      |-- -settings
      |-- -tabs
   |-- -providers
   |-- -services
   |-- -theme
|-- -www
   |-- -assets
      |-- -fonts
      |-- -icon
      |-- -images
         |-- -overlay
      |-- -js
         |-- -helper
         |-- -workerfiles
   |-- -build
```

Abbildung A.3.: Struktur des Projektordners

A.2.1. Hauptansichten der Applikation

Die mit „Measurement“, „Settings“ und „Results“ bezeichneten Ansichten der Applikation bilden die Hauptansichten der Benutzungsoberfläche. Ihre grafische Repräsentation wurde bereits in Kapitel 8 im Abschnitt 8.3 besprochen. Zwischen den drei Ansichten kann jederzeit mit Hilfe der „Tabs“-Leiste am unteren Bildschirmrand oder durch „wischen“ nach links oder rechts gewechselt werden. Alle diese Ansichten sind sogenannte „Komponenten“ der Entwicklungsumgebung Ionic und bestehen, ähnlich wie gewöhnliche Internetseiten, aus einer grafischen Ansicht, dem sogenannten „View“, einem „Style Sheet“ sowie einer TypeScript-Datei, die die Programmlogik der jeweiligen

Ansicht enthält.

Measurement

Dateipfad: `/roomAcoustics/src/pages/measurement`

Die Messansicht der Applikation dient zur Steuerung des Aufnahmevorgangs von Impulsantworten. Ihre Programmlogik steuert dabei die Kommunikation mit den Web Workern „buffer Worker“ und „preProcessor Worker“, die durch das in Kapitel 7 in Abschnitt 7.2 erwähnte Nachrichtensystem ihnen zugewiesene Aufgaben parallel zur Aufnahme abarbeiten. Ist der Aufnahmevorgang abgeschlossen, werden alle aufgezeichneten Impulsantworten in einem Aufnahmeobjekt zwischengespeichert und für die weitere Verarbeitung bereitgestellt.

Settings

Dateipfad: `/roomAcoustics/src/pages/settings`

In der Einstellungsansicht hat die nutzende Person die Möglichkeit, alle Einstellungen bezüglich der raumakustischen Berechnungen auf Basis der aufgenommenen Impulsantworten zu tätigen. Dort kann auch manuell eine Berechnung der Parameter gestartet werden. Wird eine Berechnung gestartet, werden die ausgewählten Einstellungen an den „Parameters Provider“ weitergegeben. Dort werden die eigentlichen Berechnungen der Parameter koordiniert.

Results

Dateipfad: `/roomAcoustics/src/pages/results`

Ist eine Berechnung abgeschlossen, werden die berechneten Parameter auf der Ergebnisansicht in tabellarischer Form dargestellt. Im Fall der Nachhallzeit erfolgt auch eine grafische Präsentation der zu DIN 3382-1 [1] konformen Ergebnisse. Zusätzlich besteht die Möglichkeit, Ergebnisse zu exportieren und lokal gespeicherte Daten zu löschen. Der Export aufgezeichneter Impulsantworten erfolgt dabei im WAVE-Dateiformat, die berechneten Parameter werden in Form einer vorformatierten Textdatei im Speicher abgelegt. Für die Funktionen der parallelen Erstellung der Dateien zum Export kommuniziert die Programmlogik der Ergebnisansicht mit dem „postProcessor Worker“ Die lokale Speicherung der Daten erfolgt in einem eigens für die Applikation angelegten Ordner, der leicht mit dem Cloud-Dienst der jeweiligen Plattform synchronisiert werden kann.

A.2.2. Parameters Provider

Dateipfad: `/roomAcoustics/src/providers/`

Ein Provider entspricht einer TypeScript-Komponente der Ionic Entwicklungsumgebung, die nur Programmlogik enthält. Im „Parameters Provider“ findet die Koordination der Kommunikation mit dem „preProcessor Worker“, dem „audioProcessor Wor-

ker“ und dem „fastSTI Worker“ statt. Der Parameters Provider ist zuständig für das Auslösen des Filtervorgangs der aufgezeichneten Impulsantworten, die iterative Endpunktbeschneidung (siehe Kapitel 5) und die anschließende Parameterberechnung. Ist die Parameterberechnung abgeschlossen, werden die erhaltenen Ergebnisse der Ergebnisansicht zur Verfügung gestellt.

A.2.3. Audio Worker

Dateipfad: `/roomAcoustics/src/services/`

Der Audio Worker ist als sogenannter „Service“ implementiert und macht die in ihm enthaltene Klassendefinition der verwendeten Web Worker in jeder Komponente der Applikation verfügbar. Durch ihn wird die Erstellung der Web Worker und die Kommunikation mit ihnen erleichtert.

buffer Worker

Dateipfad: `/roomAcoustics/src/assets/js/workerfiles/bufferWorker.js`

Während des Aufnahmevorgangs werden eintreffende Pakete mit Audiodaten, sogenannte „Audio Chunks“ von der Messansicht an den „buffer Worker“ weitergegeben. Dort erfolgt die Schätzung des Hintergrundrauschens, das Buffern der Audiodaten, das Erkennen eines Stimulus zum Auslösen der Aufnahme einer Impulsantwort, sowie die Erkennung eventueller Übersteuerung des Audiosignals. Wurde ein ausreichender Stimulus wahrgenommen und es wird keine Übersteuerung festgestellt, so zeichnet der buffer Worker die entsprechende Impulsantwort auf und gibt sie nach Abschluss des Aufnahmevorgangs zurück an die Messansicht.

preProcessor Worker

Dateipfad: `/roomAcoustics/src/assets/js/workerfiles/preProcessorWorker.js`

Wurde eine Impulsantwort aufgezeichnet und zurück an die Messansicht gesendet, wird sie an den „preProcessor Worker“ weitergeleitet. Dort erfolgt zunächst die Anfangsbeschneidung der Impulsantwort. Die am Anfang beschnittenen Impulsantworten werden über die Dauer des Aufnahmevorgangs im preProcessor Worker zwischengespeichert und nach Abschluss der Aufnahme zurück an die Messansicht gesendet. Dort erfolgt eine Speicherung der Aufnahmedaten in einem Aufnahmeobjekt, das für die anderen Ansichten, sowie den Parameters Provider zur weiteren Verarbeitung zur Verfügung gestellt wird.

Wird eine Berechnung der Parameter ausgelöst, erhält der preProcessor Worker vom Parameters Provider die Anweisung, den Filtervorgang und die Endpunktbeschneidung auszuführen. Die gefilterten und am Ende beschnittenen Audiodaten werden im Anschluss zurück an den Parameters Provider übergeben.

audioProcessor Worker

Dateipfad: `/roomAcoustics/src/assets/js/workerfiles/audioProcessorWorker.js`

Sind die gefilterten und beschnittenen Audiodaten im Parameters Provider verfügbar, werden sie von dort zusammen mit den Einstellungen zur Berechnung und Parameterauswahl an den „audioProcessor Worker“ weitergegeben. Hier erfolgt die eigentliche Parameterberechnung und Energiekompensation der Endpunktbeschneidung. Ist die Berechnung der Parameter abgeschlossen, werden die Ergebnisse zurück an den Parameters Provider übergeben.

fastSTI Worker

Dateipfad: `/roomAcoustics/src/assets/js/workerfiles/fastStiWorker.js`

Der „fastSTI Worker“ dient ausschließlich zur Bestimmung des Sprachübertragungsindexes, da seine Berechnung mit wesentlich höherem Aufwand verbunden ist, als es für die restlichen Parameter der Fall ist. Die Kommunikation mit dem fastSTI Worker erfolgt ebenfalls ausgehend vom Parameters Provider.

postProcessor Worker

Dateipfad: `/roomAcoustics/src/assets/js/workerfiles/postProcessorWorker.js`

Im „postProcessor Worker“ finden alle parallelen Bearbeitungen statt, die nach dem Abschluss der Audioverarbeitung notwendig sind. Nachdem die Ergebnisse der Berechnung an die Ergebnisansicht weitergegeben wurden, erhält der postProcessor Worker den Auftrag, die aufgezeichneten Audiodaten in das WAVE-Dateiformat umzuwandeln und die berechneten Ergebnisse in Form einer Textdatei zum leichten Import in Tabellenkalkulationsprogramme abzuspeichern.

A.3. Grafiken

Zusätzlich zur Implementierung der Applikation wurden Grafiken wie ein Icon so wie ein Startbildschirm für die Applikation angefertigt. Diese Grafiken sollen hier gezeigt werden.

A.3.1. Icon der Applikation

Abbildung A.4 zeigt das das für die Applikation erstellte Logo, das in den jeweiligen Plattformen als Icon zum Start der Applikation zu sehen ist.

Abbildung A.4.: Icon der Applikation

A.3.2. Startbildschirm der Applikation

In Abbildung A.5 ist der Startbildschirm oder „Splash Screen“ der Applikation zu sehen. Er erscheint beim Start der Applikation bis zum Abschluss ihres Ladevorgangs.

Abbildung A.5.: Startbildschirm der Applikation

A.4. Technische Spezifikationen verwendeter Messmikrofone

Die folgenden Abschnitte zeigen die Frequenzgänge der verwendeten Messmikrofone, wie sie auf den Herstellerseiten zu finden sind.

A.4.1. Frequenzgang des micW i436 Mikrofons

Das folgende Bild zeigt den durch den Hersteller angegebenen Frequenzgang des micW i436 Ansteckmikrofons.

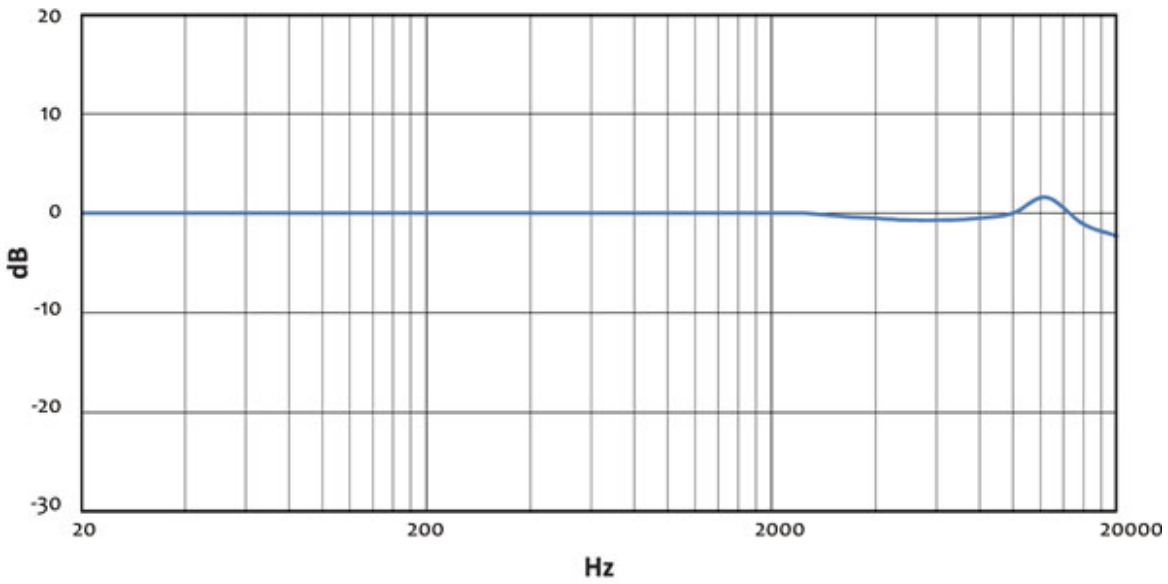


Abbildung A.6.: Frequenzgang des micW i436 Mikrofons [60]

A.4.2. Frequenzgang des Behringer ECM8000 Mikrofons

Das folgende Bild zeigt den durch den Hersteller angegebenen Frequenzgang des in Kombination mit dem USB Interface verwendeten Behringer ECM8000 Messmikrofons.

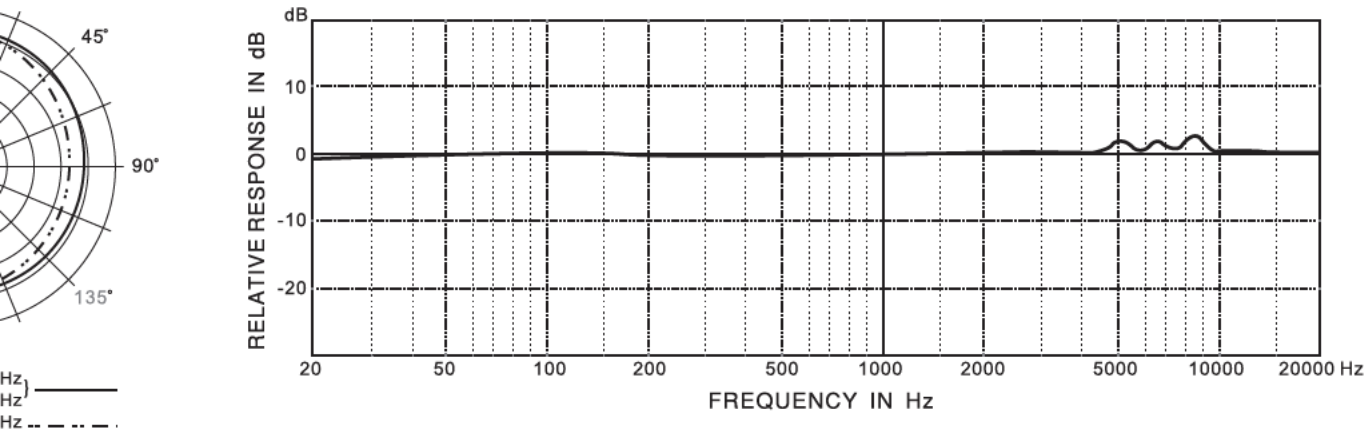


Abbildung A.7.: Frequenzgang des Behringer ECM8000 Mikrofons [61]

A.4.3. Frequenzgang des NTi M2230 Mikrofons

Das folgende Bild zeigt die durch den Hersteller angegebenen Frequenzgänge der Messmikrofone. Die in schwarzer Farbe dargestellte Kurve beschreibt dabei das verwendete NTi M2230 Messmikrofon.

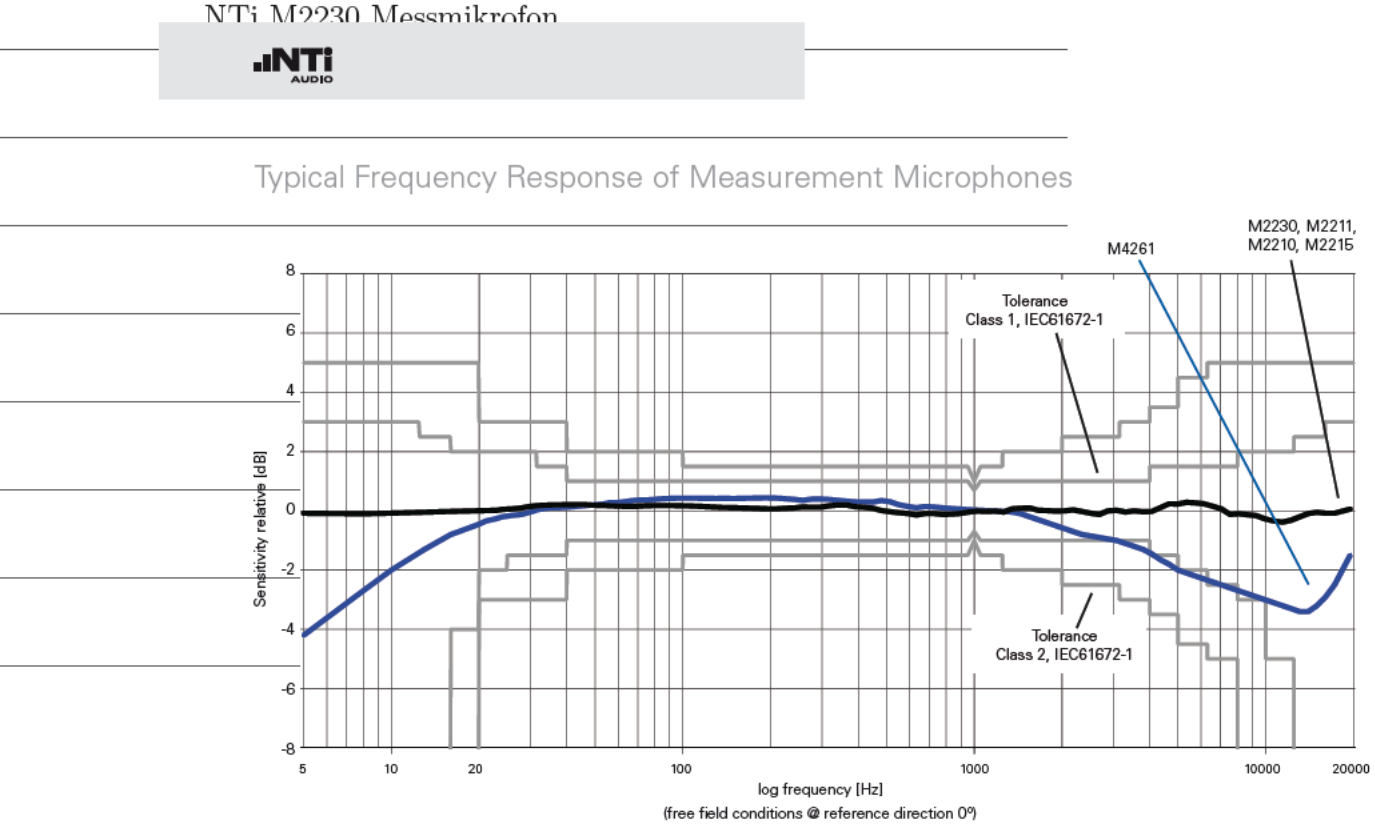


Abbildung A.8.: Frequenzgänge der NTi Messmikrofone [62]

A.5. Anpassungen an den Algorithmus nach Lundeby et al.

Um sicher zu stellen, dass die Berechnung der raumakustischen Parameter stets zu darstellbaren Ergebnissen in der Applikation führt, waren Anpassungen an den Algorithmus zur automatischen Endpunktbeschneidung nach Lundeby et al. [18] notwendig. Diese Anpassungen wurden in Kapitel 5.2.1 erläutert und sollen hier dargestellt werden. Die Bezeichnung der einzelnen Schritte des Algorithmus erfolgt dabei analog zur Bezeichnung in [18]. Die Implementierung des Algorithmus ist unter dem Pfad „`/roomAcoustics/src/assets/js/lundebyMethod.js`“ im Projektordner zu finden.

A.5.1. Anpassungen in Schritt 4

Im vierten Schritt des Algorithmus wird mit Hilfe linearer Regression eine erste Schätzung für den Schnittpunkt (im Quelltext: cpIdx) von später Abklingkurve und Störpegel bestimmt. Nach dieser Bestimmung wurde eine zusätzliche Abfrage des Wertes des bestimmten Schnittpunktes hinzugefügt. Wichtige weitere Größen für diese Abfrage sind der Störbeginn (im Quelltext: startNoise) und der Sicherheitsabstand (im Quelltext: noiseIndex). Der Störbeginn beschreibt den in Schritt 2 des Algorithmus geschätzten Übergang des Signals in den Bereich des Störsignals. Der Sicherheitsabstand aus Schritt 3 des Algorithmus beschreibt die Stelle im Signal, die 10 dB über dem gemittelten Schalldruckpegel des Restsignals nach dem Störbeginn liegt.

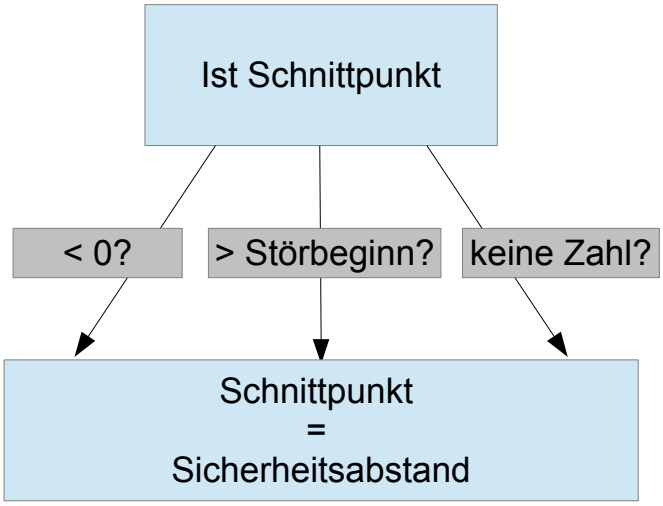


Abbildung A.9.: Anpassungen in Schritt 4

A.5.2. Anpassungen in Schritt 7

Im siebten Schritt des Algorithmus wurden mehrere Abfangmechanismen hinzugefügt. Schritt 7-9 enthalten die iterative Bestimmung des Schnittpunktes zwischen später

Abklingkurve und Störpegel auf Basis der in Schritt 1-6 bestimmten Größen. Der Sicherheitsabstand aus Schritt 3 wird nun mit jeder Iteration des Algorithmus neu bestimmt. In der Anpassung des Algorithmus wird zusätzlich der letzte Wert des Sicherheitsabstands (im Quelltext: rmsNoiseIdx) als alter Sicherheitsabstand (im Quelltext: rmsNoiseIdxOld) gespeichert. Es erfolgt eine Abfrage wie in Abbildung A.10.

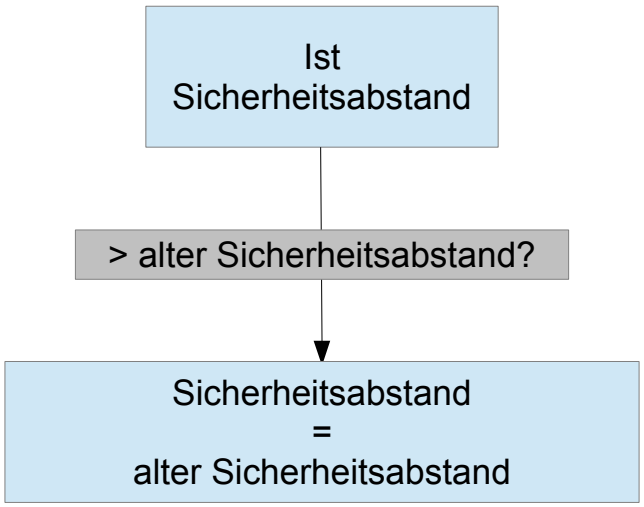


Abbildung A.10.: Anpassungen in Schritt 7

Eine weitere Abfrage erfolgt wie in Abbildung A.11 um sicher zu stellen, dass die Iteration des Sicherheitsabstandes nicht den zu untersuchenden Bereich des Signals verlässt.

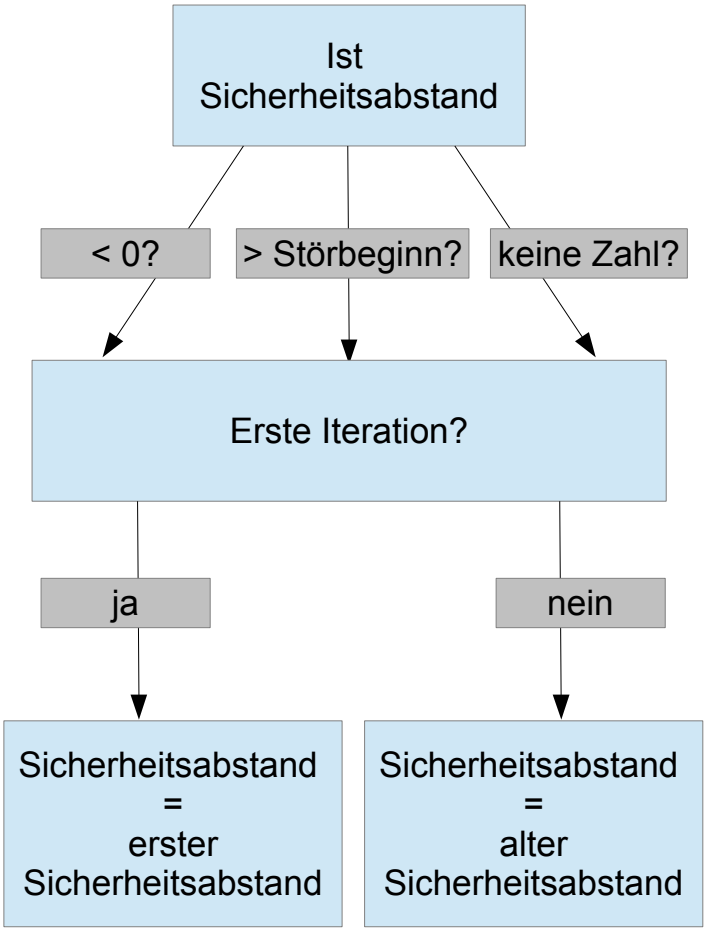


Abbildung A.11.: Anpassungen in Schritt 7

Zuletzt wird in Schritt 7 überprüft, ob der Sicherheitsabstand im zu untersuchenden Bereich zwischen 0 und dem zuvor bestimmten Störbeginn liegt (siehe Abbildung A.12.

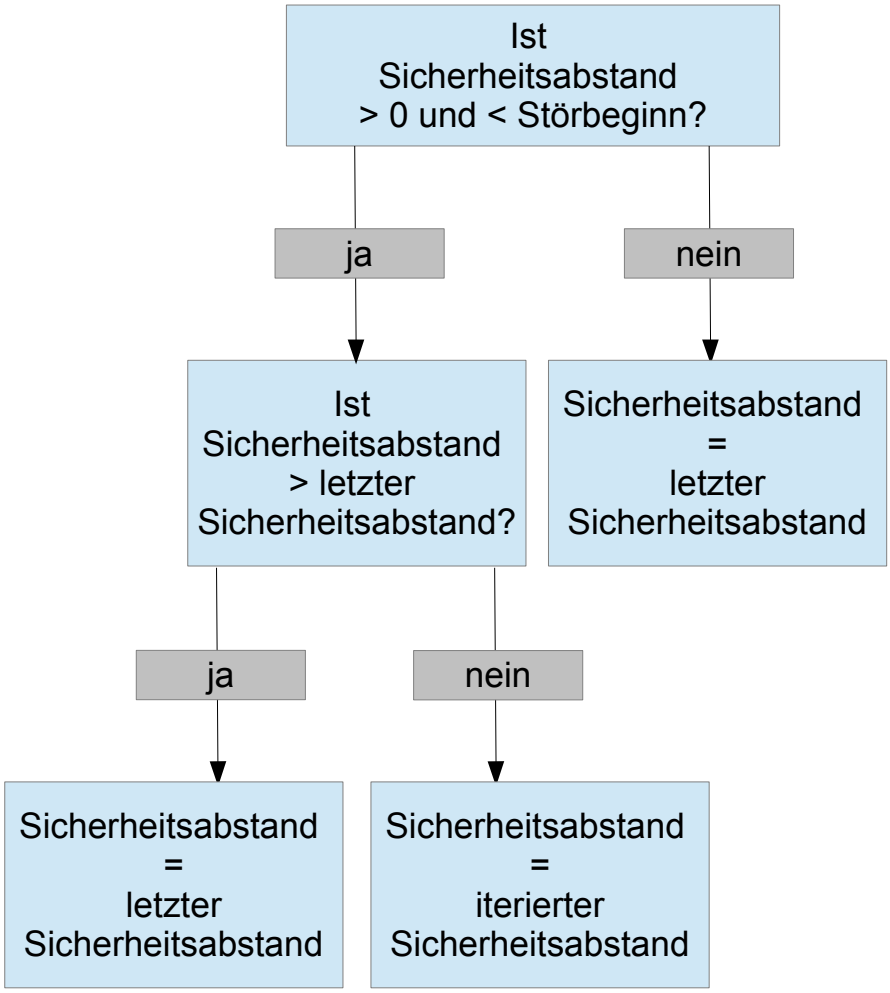


Abbildung A.12.: Anpassungen in Schritt 7

A.5.3. Anpassungen in Schritt 8

Schritt 8 des Algorithmus schätzt aus den zuvor iterierten Größen die späte Abklingkurve neu ab. In jedem Iterationsschritt wird ein neuer Start- und Endwert (im Quelltext: startIdx, endIdx) für die Regressionsgerade durch die späte Abklingkurve bestimmt. Der Start der Gerade wird anschließend auf den Spitzenwert des untersuchten Signals (im Quelltext: peak) gesetzt, falls der iterierte Startwert die Spitze unterschreitet, den Störbeginn überschreitet oder das Ergebnis keine Zahl ist (siehe Abbildung A.13).

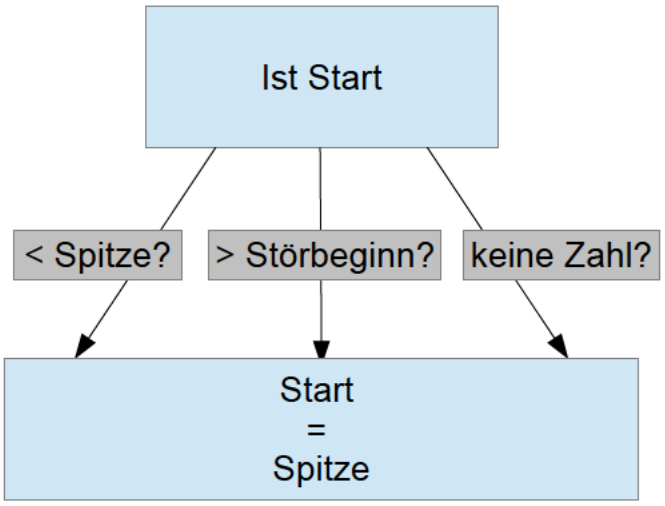


Abbildung A.13.: Anpassungen in Schritt 8

In Abbildung A.14 ist dargestellt, wie auf ähnliche Weise der Endwert überprüft wird. Ist dieser kleiner als 0, größer als der Störbeginn oder das Ergebnis keine Zahl, so wird er auf den iterierten Sicherheitsabstand (im Quelltext: rmsNoiseIdx) gesetzt.

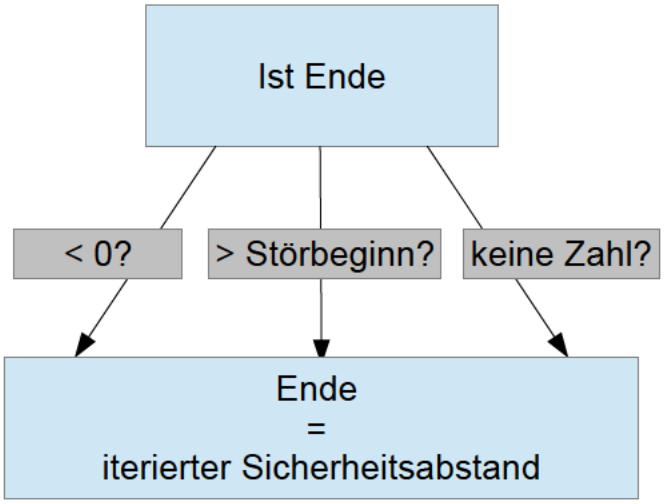


Abbildung A.14.: Anpassungen in Schritt 8

A.5.4. Anpassungen in Schritt 9

Schritt 9 beinhaltet die letztendliche Berechnung des Schnittpunktes zwischen Nutz- und Störanteil des aufgezeichneten Signals. In Abbildung A.15 erfolgen die gleichen Abfragen wie in Abbildung A.14 für den aktuell iterierten Schnittpunkt. Sind die Voraussetzungen der Abfrage erfüllt, so wird der aktuelle Schnittpunkt auf den zuvor iterierten Schnittpunkt zurückgesetzt.

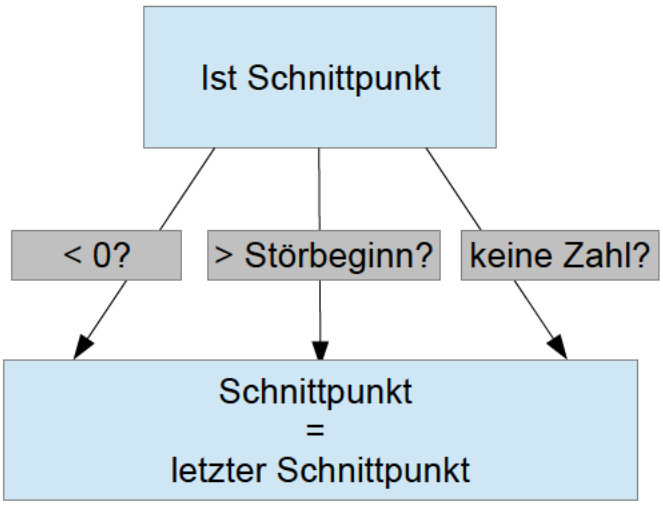


Abbildung A.15.: Anpassungen in Schritt 9

A.6. Anpassung des Audio Input Plugins

Die Manipulation des Vorverstärkerwertes des iOS-Systems erforderte seine manuelle Anpassung des Cordova AudioInput Plugins des GitHub Nutzers „edimuj“ [57] durch den Autor dieser Arbeit. Die Änderungen erfolgten in allen drei zugehörigen Dateien des Plugins. Anpassungen, die lediglich einzelne Zeilen Quelltextes betreffen, sind in den folgenden Quelltextauszügen in roter Farbe hervorgehoben. Die drei Dateien sind unter dem Pfad „`/roomAcoustics/plugins/cordova-plugin-audioinput/src/ios`“ im Projektordner zu finden.

A.6.1. Anpassungen in CDVAudioInputCapture.m

Zeilen 52-57:

```
52 - int audioSourceType =
       [[command.arguments objectAtIndex:4] intValue];
53 - float gainValue = [[command.arguments objectAtIndex: 5] floatValue];
54 -
55 - self.audioReceiver =
```

```
        [[AudioReceiver alloc] init:sampleRate
        bufferSize:bufferSizeInBytes
        noOfChannels:channels
        audioFormat:format
        sourceType:audioSourceType
        gain:gainValue];
56 -
57 - self.audioReceiver.delegate = self;
```

A.6.2. Anpassungen in Audioreceiver.m

Zeilen 44-65:

```
44 -
45 - CGFloat gain;
46 -
47 - /**
48 - AudioReceiver class implementation
49 - */
50 - @implementation AudioReceiver
51 -
52 - @synthesize mySampleRate, myBufferSize, myChannels,
       myBitRate, myFormat, myGain, delegate;
53 -
54 - /**
55 - Init instance
56 - */
57 - - (AudioReceiver*)init:(int)sampleRate
          bufferSize:(int)bufferSizeInBytes
          noOfChannels:(short)channels
          audioFormat:(NSString*)format
          sourceType:(int)audioSourceType
          gain:(float) gainValue
58 - {
59 -   static const int maxBufferSize = 0x100000;
60 -
61 -   if (self) {
62 -       OSStatus status = noErr;
63 -       gain = gainValue;
64 -       AVAudioSession* avSession =
             [AVAudioSession sharedInstance];
```

Zeilen 105-121 enthalten die Funktion zur Änderung des Vorverstärkerwertes. Die gesamte Funktion wurden zum originalen Plugin hinzugefügt.

```
105 - /**
106 -    Function for Input Gain Setting
107 -  */
```

```
108 - - (void)setInputGain:(CGFloat)gain
109 - {
110 -  AVAudioSession *avSession =
          [AVAudioSession sharedInstance];
111 -  NSLog(@"Gain settable in method?: %hhd",
                        avSession.inputGainSettable);
112 -  if (avSession.inputGainSettable){
113 -   BOOL success = [avSession setInputGain:gain error:nil];
114 -   NSLog(@"Gain Value: %.1f ", avSession.inputGain);
115 -   if (!success) {NSLog(@"Error setting gain.");
          //error  handling
116 -   }
117 -  }
118 -  else {
119 -   NSLog(@"Cannot set input gain");
120 -  }
121 - }
```

Zeilen 123-130:

```
123 - /**
124 -    Start Audio Input capture
125 -  */
126 -
127 - - (void)start {
128 -    [self startRecording];
129 -    [self setInputGain:gain];
130 - }
```

A.6.3. Anpassungen in Audioreceiver.h

Zeilen 35 - 40:

```
35 - @property (nonatomic) int mySampleRate;
36 - @property (nonatomic) int myBufferSize;
37 - @property (nonatomic) short myChannels;
38 - @property (nonatomic) short myBitRate;
39 - @property (nonatomic) NSString* myFormat;
40 - @property (nonatomic) float myGain;
```

Zeilen 42-49:

```
42 - (void)start;
43 - (void)stop;
44 - (void)pause;
45 - (void)dealloc;
46 - (void)setInputGain:(CGFloat)gain;
47 - (AudioReceiver*)init:(int)sampleRate
        bufferSize:(int)bufferSizeInBytes
        noOfChannels:(short)channels
```

```
        audioFormat:(NSString*)format
        sourceType:(int)audioSourceType
        gain:(float) gainValue;
48 - (void)didReceiveAudioData:(short*)samples
                dataLength:(int)length;
49 - (void)hasError:(int)statusCode:(char*)file:(int)line;
```

B. Tabellen

Falls nicht anders angegeben, gilt für alle Berechnungen, die mit Hilfe der implementierten Algorithmen ausgeführt wurden, die jeweils angegebene Art der Filterung für die Filterordnung 6. Alle angegebenen Abweichungen ε sind immer als Abweichungen zum angegebenen Referenzsystem zu verstehen. Alle betrachteten Ergebnisse der Nachhallzeit wurden mit Hilfe der T_{20}-Methode bestimmt. Der Versuchsaufbau bei Messungen mit der Applikation gleicht dem in Kapitel 10 in Abschnitt 10.1 beschriebenen.

B.1. Evaluation des Algorithmus von Lundeby et al.

B.1.1. Ohne Anpassung der Parameter

Tabelle B.1.: Impulsantwort 1 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

f_m in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	$0,7505$	$0,8492$	$13,15$
250	$0,8346$	$0,8479$	$1,59$
500	$0,7562$	$0,7991$	$5,67$
1000	$0,48$	$0,6003$	$4,03$
2000	$0,46$	$0,5971$	$3,41$
4000	$0,39$	$0,5143$	$9,24$

Tabelle B.2.: Impulsantwort 2 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

f_m in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	$1,0858$	$1,1754$	$8,25$
250	$0,9861$	$0,9904$	$0,44$
500	$0,7655$	$0,7668$	$0,17$
1000	$0,8508$	$0,9226$	$8,44$
2000	$0,8346$	$0,8762$	$4,98$
4000	$0,7526$	$0,7878$	$4,68$

B.1.2. Mit Anpassung der Parameter

Tabelle B.3.: Impulsantwort 3 - Vergleich der „Intersection Time“ zwischen ITA Toolbox und eigener Implementierung

f_m in Hz	IT_{ita} in s	IT_{imp} in s	ε_{imp} in %
125	2,671	2,9745	11,36
250	2,798	2,9745	6,31
500	2,9451	2,9744	0,99
1000	2,8856	2,9745	3,08
2000	2,9148	2,9745	2,05
4000	2,9158	2,9735	1,98

Tabelle B.4.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem

Endpunktbeschneidung	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
nein	0,56	0,56	0	0,55	1,79
ja	0,56	0,56	0	0,55	1,79
ja, mit Kompensation	0,56	0,56	0	0,55	1,79

Tabelle B.5.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem

Endpunktbeschneidung	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
nein	1,15	1,15	0	1,15	0
ja	1,10	1,10	0	1,09	0,91
ja, mit Kompensation	1,10	1,10	0	1,09	0,91

Tabelle B.6.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem

Endpunktbeschneidung	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
nein	4,57	4,57	0	4,53	0,88
ja	4,54	4,52	0,44	4,46	1,76
ja, mit Kompensation	4,56	4,55	0,22	4,54	0,44

Tabelle B.7.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	0,88	0,88	0	0,89	0,89
250	0,71	0,71	0	0,70	1,41
500	0,63	0,63	0	0,62	1,59
1000	0,48	0,48	0	0,48	0
2000	0,46	0,46	0	0,46	0
4000	0,39	0,39	0	0,39	0

Tabelle B.8.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	$0,88$	$0,88$	0	$0,87$	$1,14$	1,14
250	$0,71$	$0,71$	0	$0,70$	$1,41$	1,41
500	$0,63$	$0,63$	0	$0,62$	$1,59$	1,59
1000	$0,48$	$0,48$	0	$0,48$	0	0
2000	$0,46$	$0,46$	0	$0,46$	0	0
4000	$0,39$	$0,39$	0	$0,39$	0	0

Tabelle B.9.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	$0,88$	$0,88$	0	$0,88$	0
250	$0,71$	$0,71$	0	$0,70$	$1,41$
500	$0,63$	$0,63$	0	$0,62$	$1,59$
1000	$0,48$	$0,48$	0	$0,48$	0
2000	$0,46$	$0,46$	0	$0,46$	0
4000	$0,39$	$0,39$	0	$0,39$	0

Tabelle B.10.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	$1,27$	$1,26$	$0,79$	$1,28$	$0,79$
250	$1,31$	$1,31$	0	$1,30$	$0,76$
500	$1,41$	$1,41$	0	$1,38$	$2,13$
1000	$1,16$	$1,16$	0	$1,15$	$0,86$
2000	$1,11$	$1,11$	0	$1,15$	$0,90$
4000	$1,08$	$1,08$	0	$1,07$	$0,93$

Tabelle B.11.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	$1,26$	$1,26$	0	$1,25$	$0,79$	0,79
250	$1,27$	$1,27$	0	$1,27$	$0,00$	0
500	$1,17$	$1,18$	$0,85$	$1,18$	$0,85$	0
1000	$1,10$	$1,11$	$0,91$	$1,10$	$0,00$	0,90
2000	$1,08$	$1,08$	0	$1,07$	$0,93$	0,93
4000	$1,05$	$1,05$	0	$1,05$	$0,00$	0

Tabelle B.12.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	1,26	1,26	0	1,25	0,79
250	1,27	1,27	0	1,27	0,00
500	1,18	1,19	0,85	1,19	0,85
1000	1,11	1,11	0	1,11	0,00
2000	1,08	1,08	0	1,07	0,93
4000	1,05	1,05	0	1,05	0,00

Tabelle B.13.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	5,60	5,58	0.36	5,60	0
250	5,02	5,01	0,20	5,00	0,40
500	4,71	4,71	0	4,70	0,21
1000	4,24	4,24	0	4,22	047
2000	3,76	3,76	0	3,74	0,53
4000	2,71	2,72	0,37	2,70	0,37

Tabelle B.14.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	5,42	5,38	0,74	5,35	1,29	0,56
250	4,93	4,88	1,01	4,85	1,62	0,61
500	4,69	4,65	0,85	4,62	1,49	0,65
1000	4,22	4,22	0	4,19	0,71	0,71
2000	3,76	3,76	0	3,73	0,80	0,80
4000	2,71	2,72	0,37	2,69	0,74	1,10

Tabelle B.15.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	5,53	5,50	0,54	5,48	0,90
250	5,00	4,96	0,80	4,95	1,00
500	4,73	4,69	0,85	4,67	1,27
1000	4,24	4,23	0,24	4,21	0,71
2000	3,76	3,76	0	3,74	0,53
4000	2,71	2,72	0,37	2,69	0,74

Tabelle B.16.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	0,77	0,77	0	0,77	0
160	1,00	1,00	0	1,01	1,00
200	0,81	0,80	$1,23$	0,80	1,23
250	0,71	0,71	0	0,70	1,41
315	0,69	0,69	0	0,69	0
400	0,68	0,68	0	0,68	0
500	0,67	0,67	0	0,66	1,49
630	0,55	0,55	0	0,55	0
800	0,52	0,52	0	0,52	0
1000	0,56	0,56	0	0,56	0
1250	0,45	0,45	0	0,45	0
1600	0,44	0,44	0	0,44	0
2000	0,53	0,53	0	0,53	0
2500	0,42	0,42	0	0,42	0
3150	0,41	0,41	0	0,40	2,44
4000	0,41	0,41	0	0,40	2,44
5000	0,34	0,34	0	0,34	0

Tabelle B.17.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	0,77	0,77	0	0,77	0	0
160	1,00	1,00	0	1,00	0	0
200	0,81	0,80	$1,23$	0,79	2,47	1,25
250	0,71	0,71	0	0,69	2,82	2,82
315	0,69	0,69	0	0,69	0	0
400	0,68	0,68	0	0,67	1,47	1,47
500	0,67	0,67	0	0,66	1,49	1,49
630	0,55	0,55	0	0,55	0	0
800	0,52	0,52	0	0,52	0	0
1000	0,56	0,56	0	0,55	1,79	1,79
1250	0,45	0,45	0	0,45	0	0
1600	0,44	0,44	0	0,44	0	0
2000	0,53	0,53	0	0,53	0	0
2500	0,42	0,42	0	0,42	0	0
3150	0,41	0,41	0	0,40	2,44	2,44
4000	0,41	0,41	0	0,40	2,44	2,44
5000	0,34	0,34	0	0,34	0	0

Tabelle B.18.: Impulsantwort 1 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	0,77	0,77	0	0,78	1,30
160	1,00	1,00	0	1,00	0
200	0,81	0,80	1,23	0,80	1,23
250	0,71	0,71	0	0,70	1,41
315	0,69	0,69	0	0,69	0
400	0,68	0,68	0	0,67	1,47
500	0,67	0,67	0	0,66	1,49
630	0,55	0,55	0	0,55	0
800	0,52	0,52	0	0,52	0
1000	0,56	0,56	0	0,55	1,79
1250	0,45	0,45	0	0,45	0
1600	0,44	0,44	0	0,44	0
2000	0,53	0,53	0	0,53	0
2500	0,42	0,42	0	0,42	0
3150	0,41	0,41	0	0,40	2,44
4000	0,41	0,41	0	0,40	2,44
5000	0,34	0,34	0	0,34	0

Tabelle B.19.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	1,34	1,35	0,75	1,34	0
160	1,21	1,20	0,83	1,24	2,48
200	1,28	1,29	0,78	1,29	0,78
250	1,48	1,48	0	1,48	0
315	1,30	1,30	0	1,27	2,31
400	1,26	1,27	0,79	1,21	3,97
500	1,49	1,50	0,67	1,47	1,34
630	2,05	2,07	0,98	1,73	15,61
800	1,28	1,28	0	1,29	0,78
1000	1,09	1,09	0	1,08	0,92
1250	1,15	1,15	0	1,14	0,87
1600	1,08	1,08	0	1,07	0,93
2000	1,12	1,12	0	1,11	0,89
2500	1,16	1,15	0,86	1,14	1,72
3150	1,12	1,12	0	1,11	0,89
4000	1,08	1,08	0	1,07	0,93
5000	0,95	0,95	0	0,95	0

Tabelle B.20.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	1,34	1,35	0,75	1,34	0	0,74
160	1,20	1,20	0	1,19	0,83	0,83
200	1,27	1,28	0,79	1,28	0,79	0
250	1,45	1,45	0	1,44	0,69	0,69
315	1,05	1,06	0,95	1,06	0,95	0
400	1,07	1,07	0	1,07	0	0
500	1,25	1,26	0,80	1,26	0,80	0
630	1,21	1,23	1,65	1,22	0,83	0,81
800	1,20	1,20	0	1,21	0,83	0,83
1000	1,02	1,03	0,98	1,02	0	0,97
1250	1,11	1,11	0	1,11	0	0
1600	1,05	1,06	0,95	1,05	0	0,94
2000	1,08	1,08	0	1,07	0,93	0,93
2500	1,12	1,12	0	1,11	0,89	0,89
3150	1,10	1,09	0,91	1,09	0,91	0
4000	1,07	1,07	0	1,07	0	0
5000	0,90	0,91	1,11	0,89	1,11	2,20

Tabelle B.21.: Impulsantwort 2 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	1,34	1,35	0,75	1,34	0
160	1,20	1,20	0	1,19	0,83
200	1,27	1,28	0,79	1,28	0,79
250	1,45	1,45	0	1,45	0
315	1,05	1,07	1,90	1,06	0,95
400	1,08	1,07	0,93	1,07	0,93
500	1,26	1,27	0,79	1,28	1,59
630	1,22	1,24	1,64	1,24	1,64
800	1,20	1,20	0	1,22	1,67
1000	1,02	1,03	0,98	1,02	0
1250	1,11	1,11	0	1,11	0
1600	1,06	1,06	0	1,05	0,94
2000	1,08	1,08	0	1,08	0
2500	1,12	1,12	0	1,11	0,89
3150	1,10	1,10	0	1,09	0,91
4000	1,07	1,07	0	1,07	0
5000	0,90	0,91	1,11	0,90	0

Tabelle B.22.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - ohne Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	6,13	6,09	0,65	6,10	0,49
160	5,34	5,33	0,19	5,37	0,56
200	5,34	5,34	0	5,34	0
250	5,00	5,00	0	4,99	0,20
315	4,95	4,96	0,20	4,94	0,20
400	4,81	4,80	0,21	4,78	0,62
500	4,70	4,70	0	4,69	0,21
630	4,57	4,57	0	4,55	0,44
800	4,48	4,48	0	4,48	0
1000	4,29	4,28	0,23	4,27	0,47
1250	4,00	4,00	0	3,98	0,50
1600	3,98	3,98	0	3,96	0,50
2000	3,68	3,68	0	3,66	0,54
2500	3,24	3,25	0,31	3,24	0
3150	2,92	2,92	0	2,90	0,68
4000	2,59	2,59	0	2,58	0,39
5000	2,17	2,17	0	2,16	0,46

Tabelle B.23.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %	$\Delta\varepsilon$ in %
125	5,93	5,90	0,51	5,80	2,19	1,69
160	5,24	5,15	1,72	5,14	1,91	0,19
200	5,21	5,20	0,19	5,16	0,96	0,77
250	4,92	4,88	0,81	4,92	0	0,82
315	4,87	4,82	1,03	4,81	1,23	0,21
400	4,75	4,72	0,63	4,70	1,05	0,42
500	4,63	4,62	0,22	4,58	1,08	0,87
630	4,56	4,56	0	4,54	0,44	0,44
800	4,46	4,45	0,22	4,42	0,90	0,67
1000	4,27	4,27	0	4,24	0,70	0,70
1250	3,99	3,98	0,25	3,97	0,50	0,25
1600	3,97	3,97	0	3,95	0,50	0,50
2000	3,67	3,68	0,27	3,65	0,54	0,82
2500	3,24	3,24	0	3,23	0,31	0,31
3150	2,92	2,92	0	2,90	0,68	0,68
4000	2,58	2,59	0,39	2,57	0,39	0,77
5000	2,17	2,18	0,46	2,16	0,46	0,92

Tabelle B.24.: Impulsantwort 3 - Vergleich der Nachhallzeit zwischen ITA Toolbox, Implementierung und Gesamtsystem - mit Endpunktbeschneidung und Kompensation

f_m in Hz	T_{ita} in s	T_{imp} in s	ε_{imp} in %	T_{app} in s	ε_{app} in %
125	6,17	6,02	2,43	6,03	2,27
160	5,34	5,23	2,06	5,26	1,50
200	5,31	5,34	0,56	5,29	0,38
250	5,00	4,95	1,00	4,99	0,20
315	4,93	4,89	0,81	4,88	1,01
400	4,81	4,78	0,62	4,75	1,25
500	4,69	4,67	0,43	4,65	0,85
630	4,57	4,56	0,22	4,55	0,67
800	4,48	4,47	0,22	4,45	0,70
1000	4,28	4,28	0	4,25	0,25
1250	3,99	3,99	0	3,98	0,50
1600	3,97	3,97	0	3,95	0,50
2000	3,68	3,68	0	3,65	0,82
2500	3,24	3,24	0	3,23	0,31
3150	2,92	2,92	0	2,90	0,68
4000	2,58	2,59	0,39	2,58	0
5000	2,17	2,18	0,46	2,16	0,46

B.2. Messungen mit internen Mikrofonen

Tabelle B.25.: H 0112: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	0,07	-	-	-	-
250	1,45	1,07	1,82	20,33	41,21
500	1,65	1,52	1,33	24,06	14,29
1000	1,63	1,63	1,61	1,24	1,24
2000	1,61	1,61	1,61	0	0
4000	1,55	1,55	1,51	2,65	2,65

Tabelle B.26.: H 0112: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\boldsymbol{\varepsilon}_{android}$ in %
125	0,18	0,28	-	-	-
160	-	0,08	-	-	-
200	0,56	0,16	-	-	-
250	0,94	0,70	1,38	31,88	49,28
315	1,74	1,85	1,67	4,19	10,78
400	1,82	1,79	1,61	13,04	11,18
500	1,54	1,75	1,67	7,78	4,79
630	1,38	1,50	1,62	14,81	7,41
800	1,57	1,49	1,76	10,80	15,34
1000	1,68	1,79	1,63	3,07	9,82
1250	1,66	1,68	1,69	1,78	0,59
1600	1,64	1,64	1,29	3,14	3,14
2000	1,67	1,65	1,54	8,44	7,14
2500	1,80	1,84	1,92	6,25	4,17
3150	1,74	1,84	1,90	8,42	10,00
4000	1,55	1,56	1,61	3,73	3,11
5000	1,38	1,45	1,32	4,55	9,85

B.3. Messungen mit micW i436

Tabelle B.27.: MA 041: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	T_{xl2} in s	ε_{ios} in %
125	-	-	-
250	1,35	-	-
500	1,21	1,44	15,97
1000	1,33	1,33	0
2000	1,33	1,29	3,10
4000	1,23	1,22	1,82

Tabelle B.28.: MA 041: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %
125	0,18	-	-
250	0,88	-	-
500	1,20	1,47	18,37
1000	1,29	1,41	8,51
2000	1,32	1,32	0
4000	1,19	1,19	0

Tabelle B.29.: MA 041: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	T_{xl2} in s	ε_{ios} in %
125	-	-	-
160	-	-	-
200	1,90	1,76	7,95
250	1,21	1,17	3,42
315	1,39	1,29	7,75
400	1,45	1,62	10,49
500	1,16	1,26	7,94
630	1,31	1,30	0,77
800	1,29	1,36	5,15
1000	1,23	1,25	1,60
1250	1,29	1,31	1,53
1600	1,30	1,44	9,72
2000	1,32	1,36	2,94
2500	1,32	1,27	3,94
3150	1,18	1,24	4,84
4000	1,13	1,17	3,42
5000	1,08	1,03	4,85

Tabelle B.30.: MA 041: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %
125	0,18	-	-
160	0,12	-	-
200	0,57	-	-
250	0,59	1,18	50,00
315	1,06	1,22	13,11
400	1,09	2,27	51,98
500	1,08	1,21	10,74
630	1,21	1,20	0,83
800	1,33	1,25	6,40
1000	1,28	1,20	6,67
1250	1,29	1,26	2,38
1600	1,29	1,36	5,15
2000	1,38	1,28	7,81
2500	1,37	1,30	5,38
3150	1,30	1,30	0
4000	1,16	1,15	0
5000	1,05	1,02	2,94

B.4. Messungen mit micW i436 (iOS) und Interface (Android)

Tabelle B.31.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\boldsymbol{\varepsilon}_{android}$ in %
125	-	-	-	-	-
250	-	-	1,33	-	-
500	1,29	1,32	1,35	4,44	2,22
1000	1,29	1,27	1,29	0	1,55
2000	1,39	1,38	1,33	4,51	3,76
4000	1,30	1,27	1,23	5,69	3,25

Tabelle B.32.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\boldsymbol{\varepsilon}_{android}$ in %
125	-	-	-	-	-
160	-	-	-	-	-
200	-	-	-	-	-
250	-	-	-	-	-
315	-	-	-	-	-
400	1,58	1,61	1,71	7,6	5,85
500	1,29	1,32	-	-	-
630	1,21	1,19	-	-	-
800	1,32	1,25	1,32	0	5,30
1000	1,26	1,33	1,24	1,61	7,26
1250	1,28	1,37	1,37	6,57	0
1600	1,37	1,40	1,33	3,01	5,26
2000	1,42	1,36	1,31	8,40	3,82
2500	1,41	1,34	1,50	6,00	10,67
3150	1,27	1,28	1,30	2,31	1,54
4000	1,29	1,22	1,21	6,61	0,83
5000	1,11	1,16	1,24	10,48	6,45

Tabelle B.33.: H 0112: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\boldsymbol{\varepsilon}_{android}$ in %
125	-	-	-	-	-
250	-	2,36	-	-	-
500	1,61	1,61	1,59	1,26	1,26
1000	1,69	1,65	1,75	3,43	5,71
2000	1,57	1,58	1,51	3,97	4,64
4000	1,43	1,43	1,37	4,38	4,38

Tabelle B.34.: H 0112: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	-	-	-	-	-
160	-	-	-	-	-
200	-	-	-	-	-
250	-	-	-	-	-
315	1,79	1,73	-	-	-
400	1,83	1,82	1,96	6,63	7,14
500	1,44	1,65	1,69	14,79	2,37
630	1,55	1,53	1,52	1,97	0,66
800	1,69	1,61	1,32	28,03	21,97
1000	1,64	1,58	1,69	2,96	6,51
1250	1,65	1,55	1,56	5,77	0,64
1600	1,55	1,63	1,58	1,90	3,16
2000	1,57	1,56	1,60	1,88	2,50
2500	1,63	1,60	1,63	0	1,84
3150	1,61	1,53	1,42	13,38	7,75
4000	1,56	1,45	1,54	1,30	5,84
5000	1,38	1,35	1,30	6,15	3,85

B.5. Messergebnisse nach Optimierung der Endpunktbeschneidung

Tabelle B.35.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	-	-	-	-	-
250	-	-	-	-	-
500	1,24	1,23	1,33	6,77	7,52
1000	1,29	1,28	1,27	1,57	0,79
2000	1,31	1,28	1,36	3,68	5,88
4000	1,21	1,22	1,17	3,42	4,27

Tabelle B.36.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	-	-	-	-	-
160	-	-	-	-	-
200	-	-	-	-	-
250	-	1,03	1,35	-	-
315	1,35	1,36	-	-	-
400	1,34	1,35	1,47	8,84	8,16
500	1,40	1,38	1,40	7,69	6,15
630	1,19	1,25	1,20	0,83	4,17
800	1,34	1,32	1,34	0	1,49
1000	1,26	1,21	1,19	5,88	1,68
1250	1,27	1,31	1,33	4,51	1,50
1600	1,35	1,34	1,41	4,26	4,96
2000	1,33	1,25	1,32	0,76	5,30
2500	1,34	1,33	1,28	4,69	3,91
3150	1,27	1,28	1,24	2,42	3,23
4000	1,16	1,20	1,17	0,85	2,56
5000	1,05	1,07	1,03	1,94	3,88

Tabelle B.37.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Oktavbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	-	-	-	-	-
250	-	-	-	-	-
500	1,23	1,21	1,27	3,15	4,72
1000	1,25	1,26	1,32	5,30	4,55
2000	1,35	1,37	1,30	3,85	5,38
4000	1,22	1,17	1,17	4,27	0

Tabelle B.38.: MA 005: Messergebnisse der Nachhallzeitmessung und zugehörige Abweichungen ε in Terzbändern. Filterordnung $N = 6$

f_m in Hz	T_{ios} in s	$T_{android}$ in s	T_{xl2} in s	ε_{ios} in %	$\varepsilon_{android}$ in %
125	-	-	-	-	-
160	-	-	-	-	-
200	-	-	-	-	-
250	-	-	-	-	-
315	-	-	-	-	-
400	1,34	-	-	-	-
500	1,33	1,26	1,20	10,83	5,00
630	1,18	1,23	1,35	12,59	8,89
800	1,39	1,35	1,28	8,59	5,47
1000	1,34	1,30	1,25	7,20	4,00
1250	1,36	1,40	1,30	4,62	7,69
1600	1,38	1,41	1,27	8,66	11,02
2000	1,35	1,32	1,39	2,88	5,04
2500	1,33	1,33	1,31	1,53	1,53
3150	1,21	1,24	1,32	8,33	6,06
4000	1,19	1,17	1,17	1,71	0
5000	1,06	1,05	1,05	0,95	0

C. Diagramme

Falls nicht anders angegeben, gilt für alle Berechnungen, die mit Hilfe der implementierten Algorithmen ausgeführt wurden, die jeweils angegebene Art der Filterung für die Filterordnung 6. Alle angegebenen Abweichungen ε sind immer als Abweichungen zum angegebenen Referenzsystem zu verstehen. Alle betrachteten Ergebnisse der Nachhallzeit wurden mit Hilfe der T_{20}-Methode bestimmt. Der Versuchsaufbau bei Messungen mit der Applikation gleicht dem in Kapitel 10 in Abschnitt 10.1 beschriebenen.

C.1. Messungen mit internen Mikrofonen

H 0112: Nachhallzeiten der drei Systeme bei identischer Anregung, oktavgefiltert

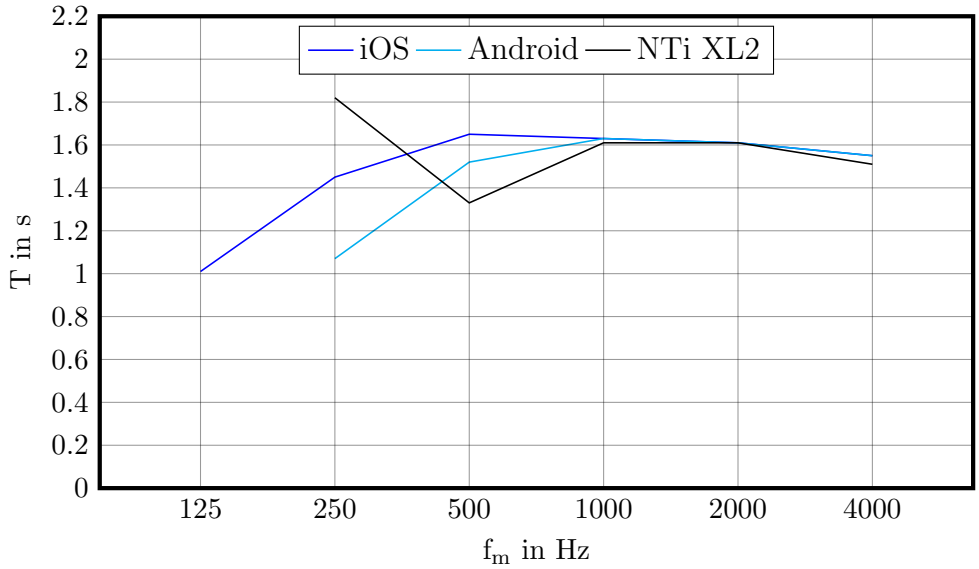


Abbildung C.1.: Nachhallzeiten im H 0112, oktavgefiltert.

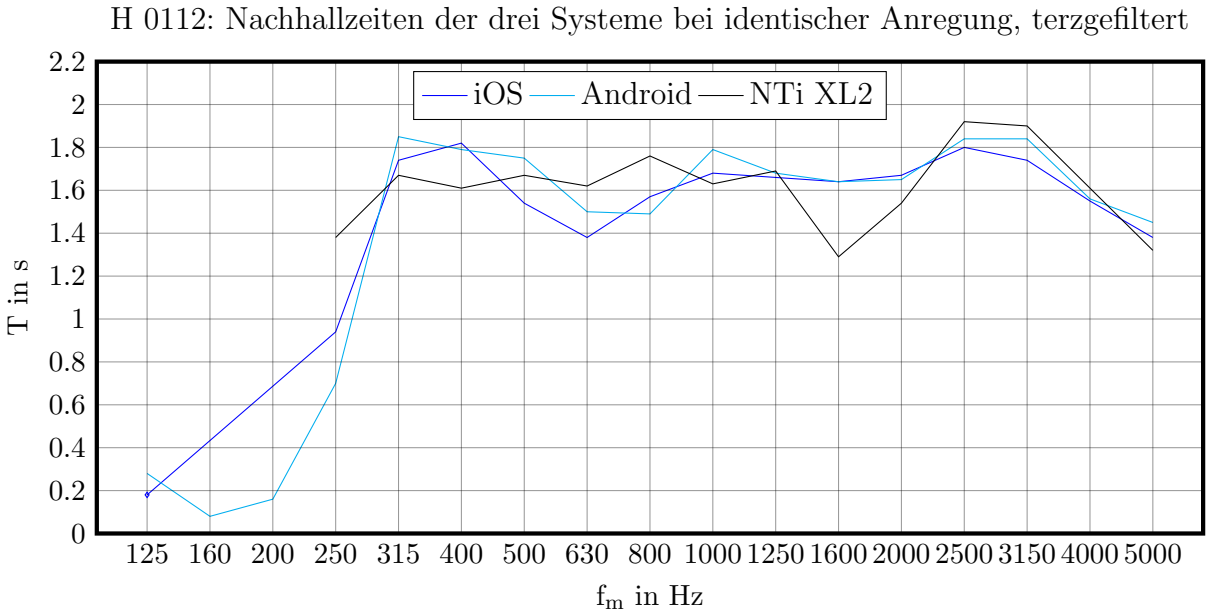


Abbildung C.2.: Nachhallzeiten im H 0112, terzgefiltert

C.2. Messungen mit micW i436

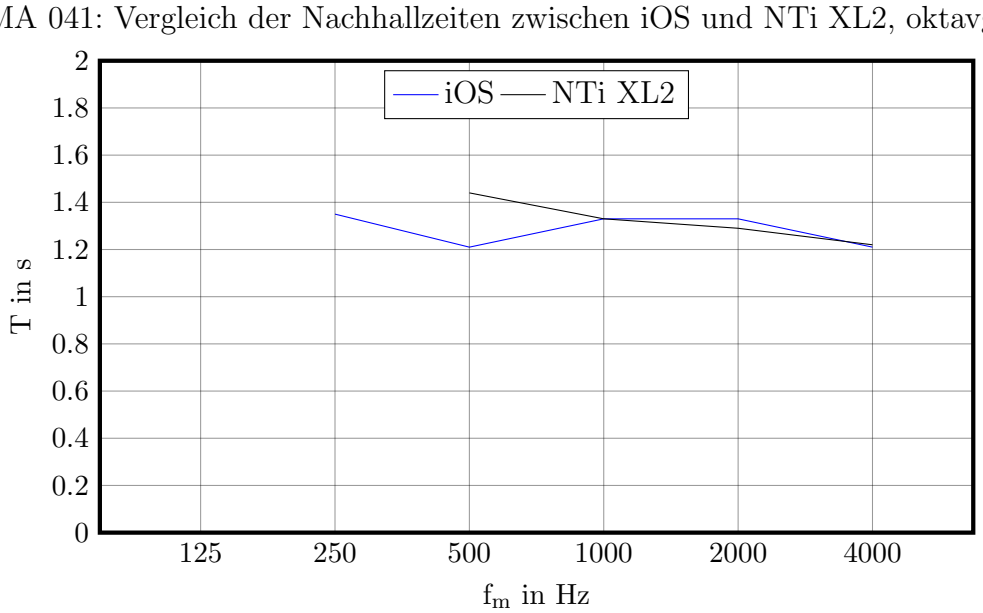


Abbildung C.3.: Nachhallzeiten im MA 041, oktavgefiltert.

MA 041: Vergleich der Nachhallzeiten zwischen Android und NTi XL2, oktavgefiltert

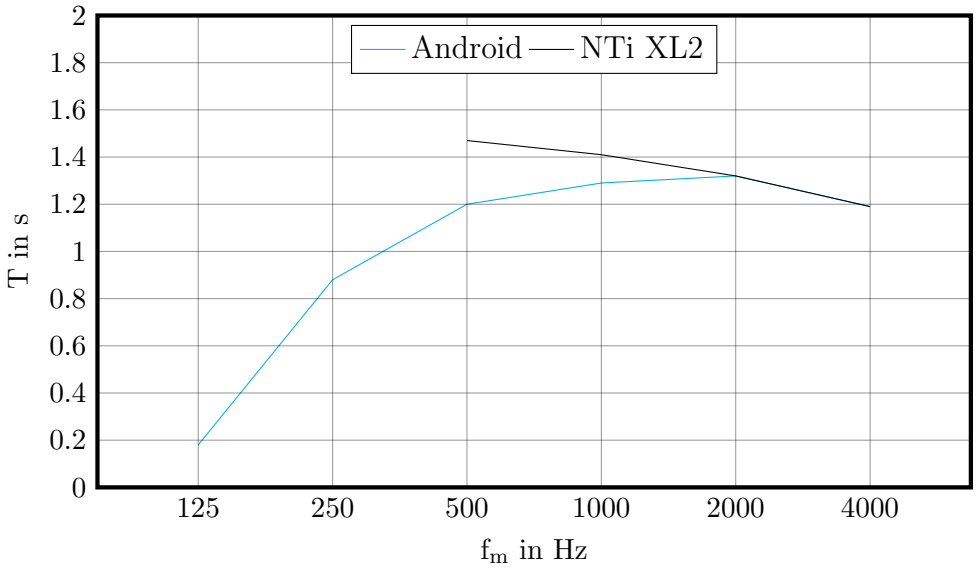


Abbildung C.4.: Nachhallzeiten im MA 041, oktavgefiltert.

MA 041: Vergleich der Nachhallzeiten zwischen iOS und NTi XL2, terzgefiltert

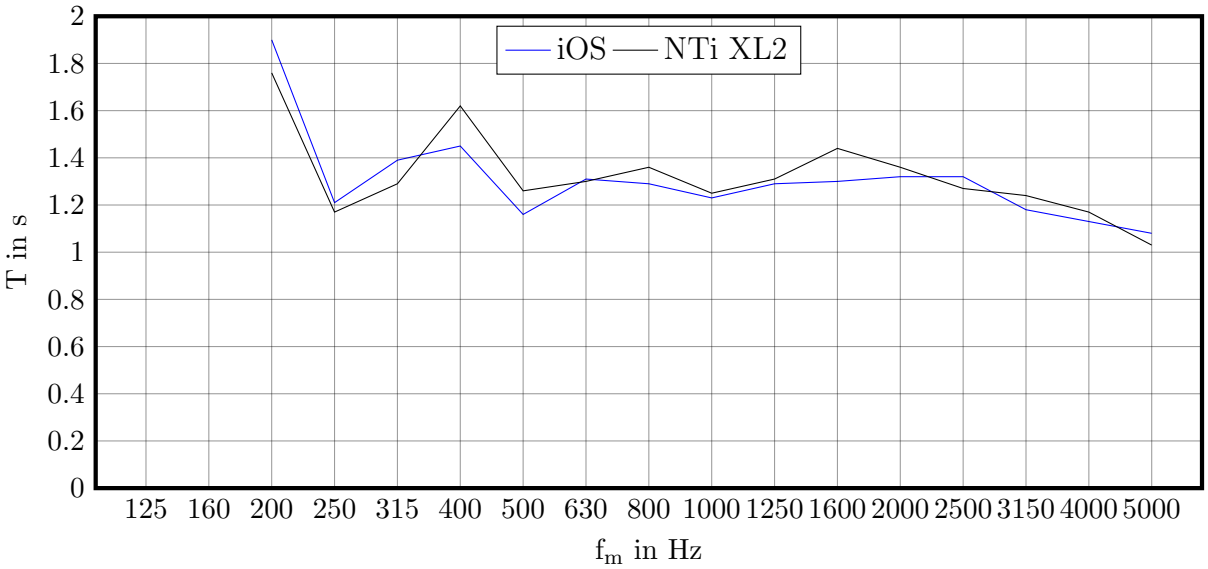


Abbildung C.5.: Nachhallzeiten im MA 041, terzgefiltert

MA 041: Vergleich der Nachhallzeiten zwischen Android und NTi XL2, terzgefiltert

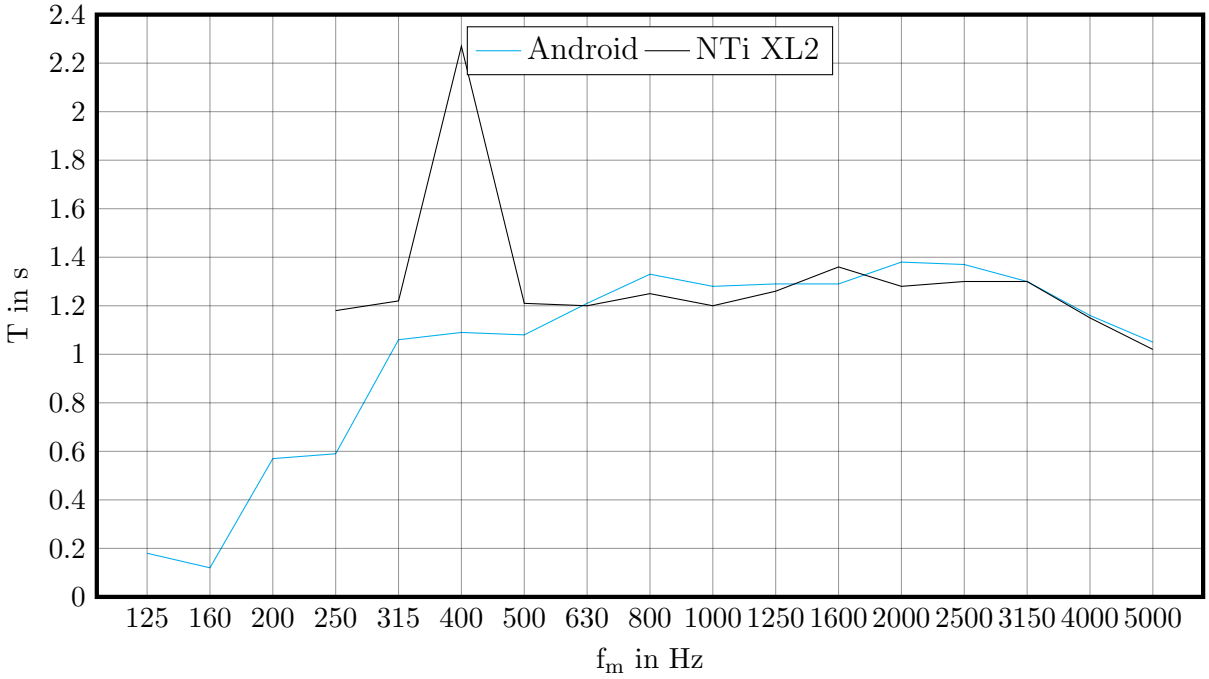


Abbildung C.6.: Nachhallzeiten im MA 041, terzgefiltert

C.3. Messungen mit micW i436 (iOS) und Interface (Android)

C.3.1. Messungen vor Optimierung der Endpunktbeschneidung

MA 005: Nachhallzeiten der drei Systeme bei identischer Anregung, oktavgefiltert

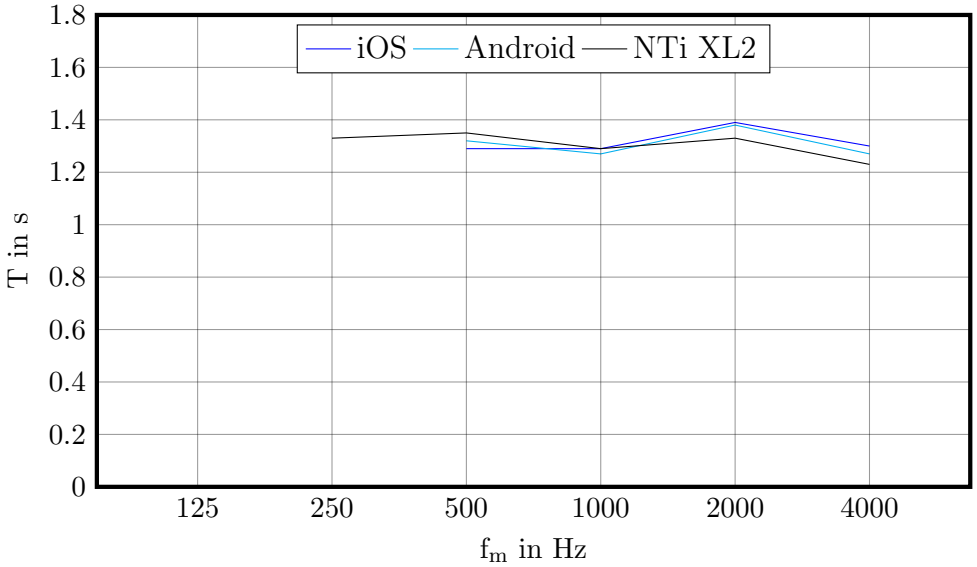


Abbildung C.7.: Nachhallzeiten im MA 005, oktavgefiltert.

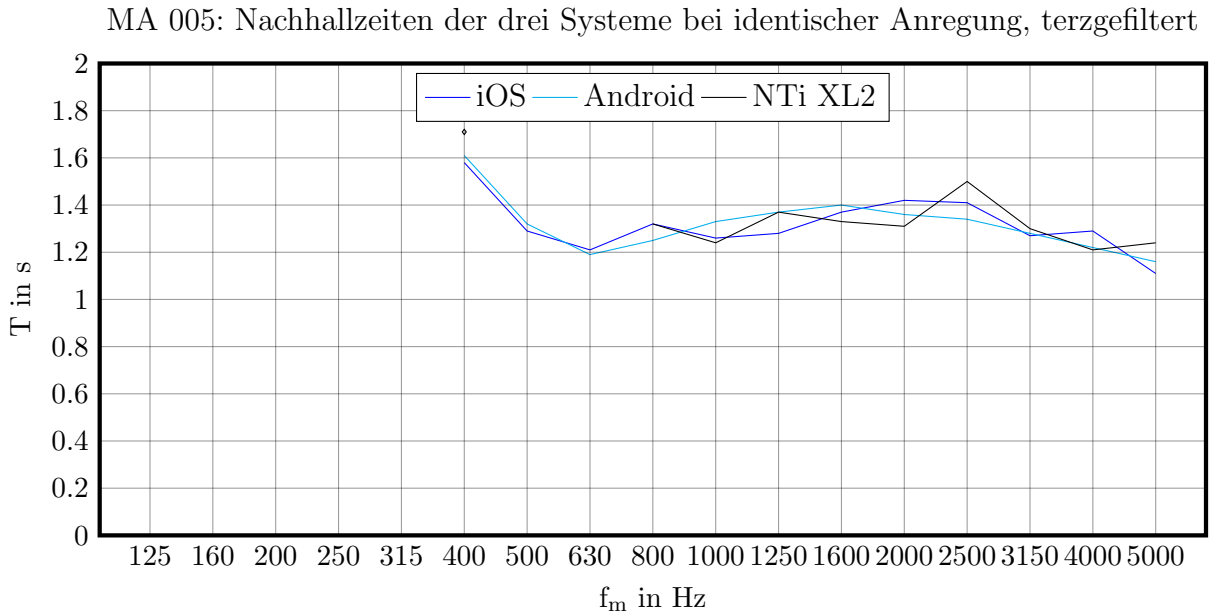


Abbildung C.8.: Nachhallzeiten im MA 005, terzgefiltert

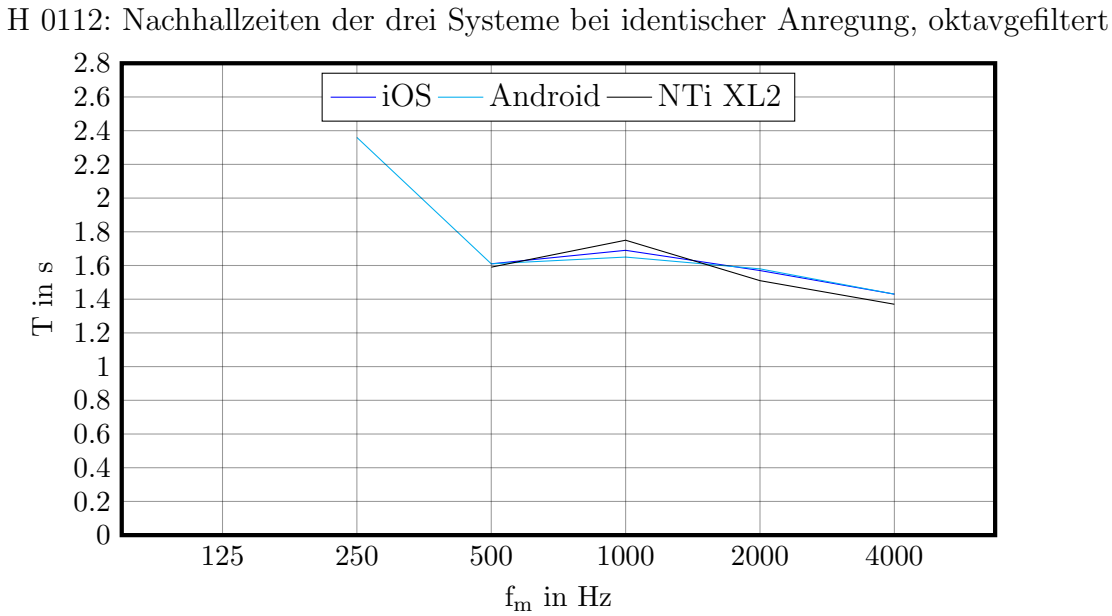


Abbildung C.9.: Nachhallzeiten im H 0112, oktavgefiltert.

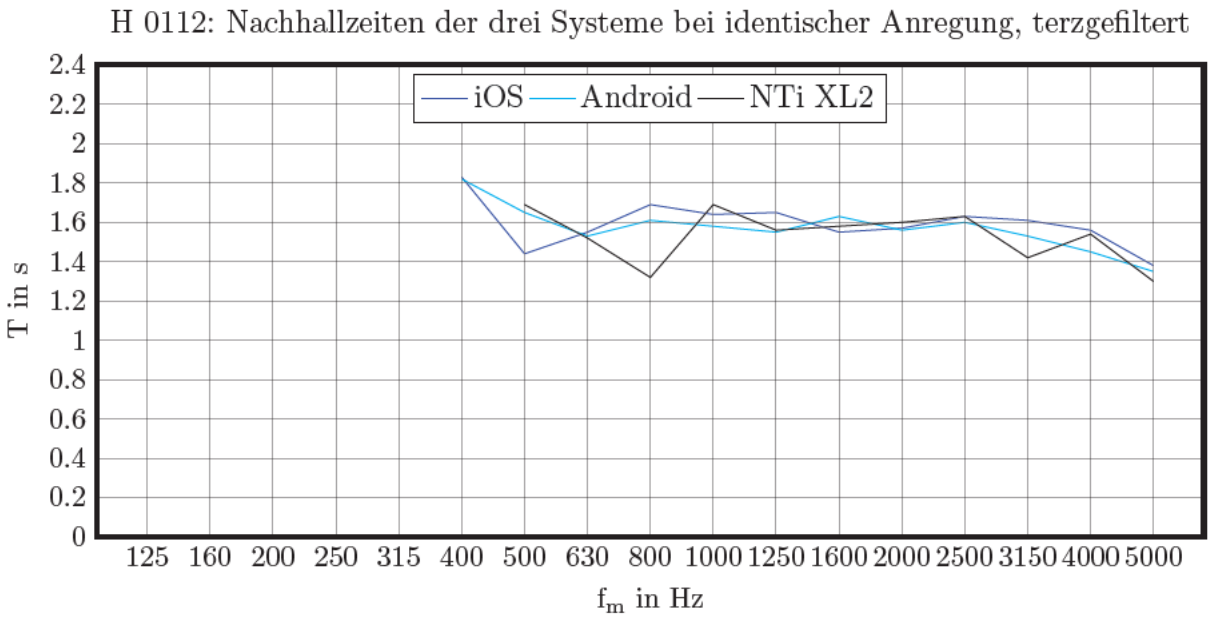


Abbildung C.10.: Nachhallzeiten im H 0112, terzgefiltert

C.3.2. Messungen nach Optimierung der Endpunktbeschneidung

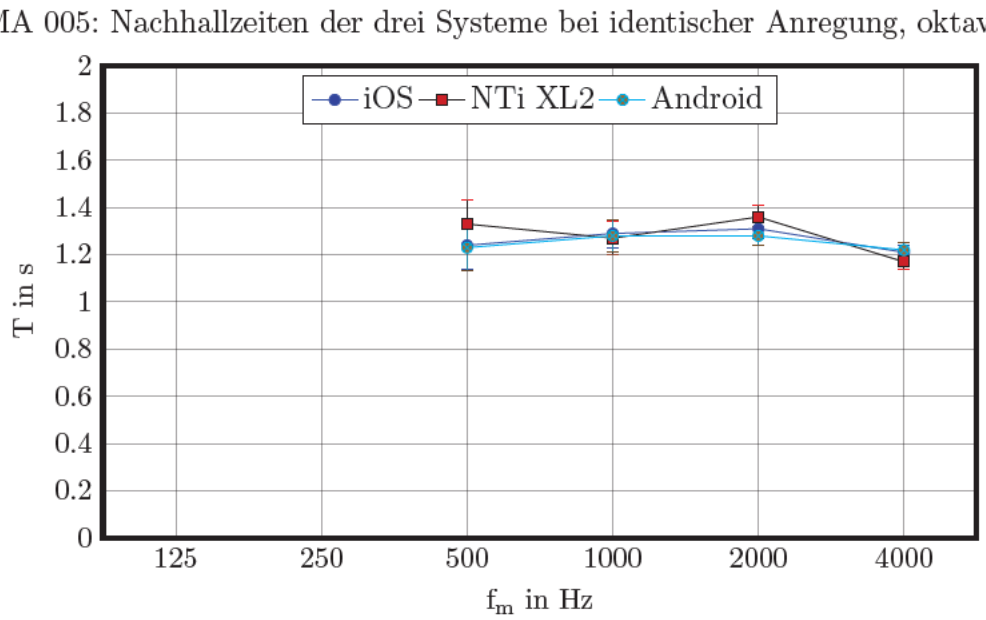


Abbildung C.11.: Nachhallzeiten im MA 005, oktavgefiltert

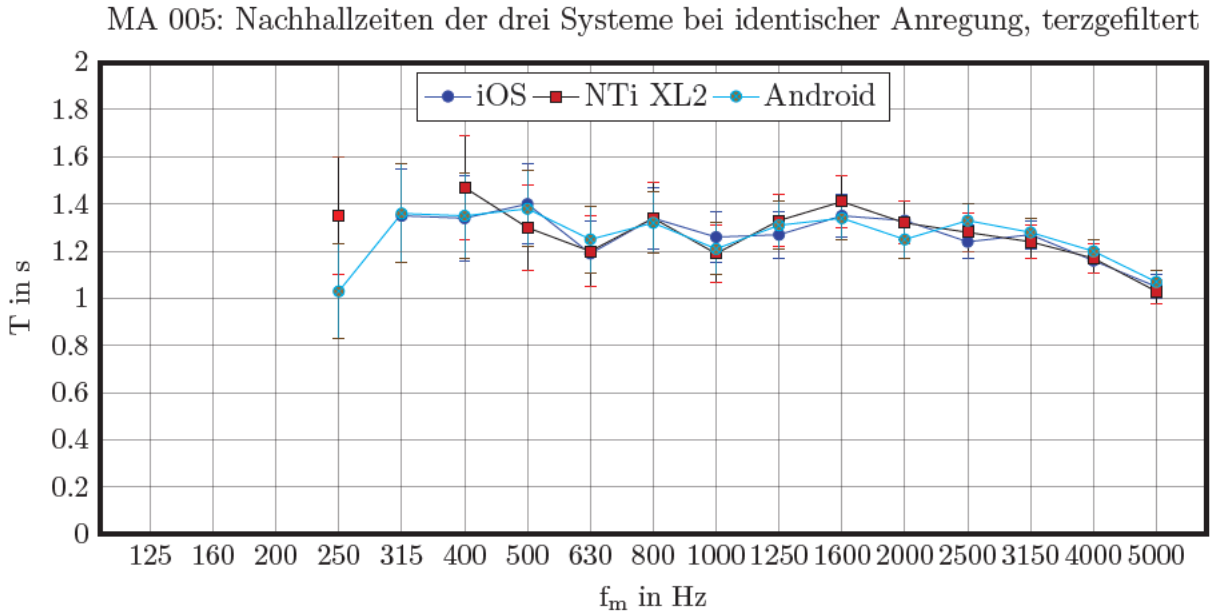


Abbildung C.12.: Nachhallzeiten im MA 005, terzgefiltert