Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Multimediakommunikation und Signalverarbeitung

Prof. Dr.-Ing. Walter Kellermann

Forschungspraktikum

Messung von Head-Related Transfer Functions und Raumimpulsantworten im Audiolabor

von Nils Ballmann

Februar 2014

Betreuer: Zhang, Mengqiu, PhD und Bürger, Michael, M.Sc.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

_____ _____

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abkürzungsverzeichnis

LMS Lehrstuhl für Multimediakommunikation und Signalverarbeitung

LNT Laboratorium für Nachrichtentechnik

LRC Low Reverberation Chamber

HRTF Head Related Transfer Function

IR Impulse response - Impulsantwort

MLS Maximum-Length-Sequenzen

MADI Multi Channel Audio Digital Interface

RIR Room impulse response

GUI Graphical User Interface

Kapitel 1

Einleitung

In dieser Arbeit ist das Forschungspraktikum am Lehrstuhl für Multimediakommunikation und Signalverarbeitung dokumentiert.

Im Rahmen dieses Forschungspraktikums wurde mit vielfältigem Messequipment gearbeitet und für verschiedene Projekte Messungen durchgeführt.

Zunächst wird kurz auf das verwendete Messequipment eingegangen und danach die einzelnen Projekte, sowie die dazugehörigen Messungen vorgestellt.

Kapitel 2

Messequipment

Im Folgenden werden das bei den Messungen verwendete Equipment und die dazugehörigen Räume kurz vorgestellt.

2.1 Räumlichkeiten

In diesem Abschnitt wird auf die Räume eingegangen, in denen gemessen wurde. Außerdem werden kurz die Erfahrungen und Besonderheiten im Umgang mit diesen Räumen beschrieben.

2.1.1 Low Reverberation Chamber

Die *Low Reverberation Chamber (LRC)* ist am Lehrstuhl für Multimediakommunikation und Signalverarbeitung (LMS) im Raum 5.14 untergebracht. Die Kammer ist auf Gummifüßen stehend, gegen die umgebenden Räumlichkeiten entkoppelt. Außerdem ist darin neben schalldämpfendem Material ein metallenes Lochraster eingebaut, welches als Diffusor dient.

Neben einem Kabelkanal, hat die Kammer eine Tür mit einem Verriegelungshebel, wobei die Tür erst dann für akustische Messungen relevant abdichtet, wenn der Ver-

riegelungshebel komplett um 90° gedreht wurde. Auch bei dem Kabelkanal ist auf Abdichtung zu achten. Die Kammer ist in ihrem Inneren 2,74m breit, 2,53m tief und 2,36m hoch.

Bei längeren Messungen oder vielen kurzen Messungen hintereinander, kann es notwendig werden, dass die Messungen zu Zeitpunkten durchgeführt werden, in denen wenig oder kaum störende menschliche Einflüsse vorhanden sind, da insbesondere die umgebenden Türen auf dem selben Stockwerk, sowie dem Stockwerk darüber, bei hartem Schließen die Messungen beeinflussen. Es hat sich als hilfreich erwiesen Nachts zu messen.

2.1.2 Audiolabor

Das Audiolabor ist im Raum G0.25 untergebracht. Für das Audiolabor ist eine Betriebssicherheitseinweisung notwendig, da der Raum keine Belüftung hat und verschiedene potentielle Gefahrenquellen vorhanden sind. Es kann eine Freischaltung des Studenten- oder Mitarbeiterausweises für die Türschließanlage stattfinden.

Die Wände des Labors sind mit Absorberkästen gedämmt, welche man umgedreht auch als Reflektoren benutzen kann. Diese Reflektoren sind mit einem leichten Winkel ausgestattet, so dass stehende Wellen nur in geringem Maße auftreten. Im Boden sind Tanks verbaut, in denen die Signale aus dem Vorraum an Geräte innerhalb des Labors weiter gegeben werden können.

Des Weiteren ist ein 128 Lautsprecher-Array an der Wand ringsum verbaut, welches sich durch den Computer im Vorraum ansprechen lässt. Hierbei ist zu beachten, dass der Benutzer-Account, der das Array über das MATLAB-Tool `playrec` ansprechen soll, entweder ein auf diesem Computer (`lnt217`) lokaler Account und dort in der Gruppe `audio` (eventuell auch `video`) ist, oder der Account innerhalb der LNT-Domain in der

Gruppe `audio` (eventuell auch `video`) ist.

2.2 Hardware

In diesem Abschnitt wird auf die Hardware eingegangen, mit der gemessen wurde. Außerdem werden kurz die Erfahrungen und Besonderheiten im Umgang mit dieser Hardware beschrieben.

2.2.1 X-Y-Tisch und Rotationsschrittmotor

Bei den Messungen kam ein *X-Y-Tisch*, auf dem noch ein *Rotationsschrittmotor* verbaut war, zum Einsatz.

Der *X-Y-Tisch* kann eine Fläche von 20cm × 20cm abdecken. Es ist zu beachten, dass die Schrittmotoren am oberen Ende der Skala verbaut sind, d.h. der Ursprung des Tisches liegt jeweils gegenüber den Motoren. Die Schrittweite-Auflösung des Tisches wurde in diesen Messungen nicht bis an ihre Grenzen getestet, da ein Ein-Zentimeter-Raster völlig ausreichte.

Bei dem *Rotationsschrittmotor* war es notwendig auf etwa Ein-Drittel-Grad-Schritte zurück zu greifen. Genauer wurde die Schrittweite-Auflösung auch hier nicht getestet. Pro Konstruktor-Aufruf des MATLAB-Steuer-Tools des Rotationsschrittmotors wechselte der Ursprung, dies machte es notwendig einen Korrektur-Bias einzuführen.

2.2.2 Vision Lasertechnik Nanostep Smart

Um die Schrittmotoren des in Abschnitt 2.2.1 genannten X-Y-Tisches anzusteuern, wurde ein *Nanostep Smart* der Firma *Vision Lasertechnik* benutzt. Aufgrund der Austauschbarkeit der Verkabelung von der Schrittmotorsteuerung zu den Motoren ist darauf zu achten, dass Kabel mit entsprechendem Querschnitt benutzt werden, da diese

auch für den Leistungstransport zu den Motoren benutzt werden. Des Weiteren ist darauf zu achten, dass auch wirklich der X-Motor mit dem X-Anschluss des Nanostep Smarts verbunden ist, analog in Y-Richtung. Wenn das Gerät mit Strom versorgt, eingeschaltet und bereit für Steuerbefehle ist, dann leuchtet die eingebaute LED blau.

2.2.3 LMS Schrittmotorsteuerung

Der Rotationsschrittmotor, des in Abschnitt 2.2.1 genannten Tisches, wurde über eine Eigenentwicklung des LMS angesteuert. Aufgrund der Länge des Kabels des Schrittmotors zur Schrittmotorsteuerung, war es möglich die Steuerung außerhalb der Messkammer zu platzieren.

2.2.4 HEAD acoustics Kunstkopf HMS II.3

Für die Aufnahme der Signale wurde der Kunstkopf *HMS II.3* der Firma *HEAD acoustics* verwendet. Dieser verfügt neben den Mikrofonen in den Ohren auch noch über einen Mund, über den auch Signale wieder gegeben werden können. In den Messungen wurden jedoch nur die Ohrmikrofone benutzt.

2.2.5 HEAD acoustics BEQ-II

Als Vorverstärker für die Mikrofonsignale des in Abschnitt 2.2.4 genannten Kunstkopfes kam ein *BEQ-II* der Firma *HEAD acoustics* zum Einsatz. Dieser scheint speziell für den genannten Kunstkopf entwickelt worden zu sein.

2.2.6 RME Advanced Digital Interface ADI-8 DD

Zur Analog-Digital-Wandlung der in Abschnitt 2.2.5 genannten, vorverstärkten Mikrofonsignale wurde ein *Advanced Digital Interface ADI-8 DD* der Firma *RME* benutzt. Mit diesem Gerät sind die Grenzen der Wandlung bei 24Bit Auflösung, 96kHz Abtastrate für 16 Kanäle. In den getätigten Messungen wurde jedoch nur 44,1kHz Abtastrate

und 2 Kanäle zur Wandlung nach ADAT benutzt.

2.2.7 RME Hammerfall DSP Multiface

Als Soundkarte für den Computer wurde eine *Hammerfall DSP Multiface* der Firma *RME* benutzt. Die Verbindung der Break-Out-Box dieser Soundkarte und der Karte selbst im Computer wurde FireWire (IEEE-1394) verwendet. Hierbei wurde sowohl für die Wiedergabe, als auch für die Aufnahme das ADAT-Interface eingesetzt.

2.2.8 RME Hammerfall DSP Digiface

Alternativ wurde auch ein *Hammerfall DSP Digiface* des gleichen Herstellers an Stelle des in 2.2.7 genannten Multiface verwendet.

2.2.9 Lautsprecher-Array (8)

Für die Wiedergabe von Signalen wurde das *Prototyp-Array* benutzt. Dieses besteht aus acht Einheiten in einer Linie nebeneinander, wobei jede dieser Einheiten aus einem Verstärker und einem Lautsprecher aufgebaut sind. Die Lautsprecher sind *CANTON*-Lautsprecher und die Verstärker sind wiederum eine Eigenentwicklung des LMS. Dieser Klasse-D Digitalverstärker hat den Namen *SMD-Dig*. Sie werden über optisches ADAT angesteuert und schleifen die Signale zu ihren Nachbareinheiten durch, sodass nur ein ADAT-Anschluß notwendig ist.

2.2.10 LMS DICIT-Mikrofon-Array

Zur Aufnahme der Signale wird das *DICIT-Mikrofon-Array* benutzt. Dieses Array ist eine Eigenentwicklung des LMS und beinhaltet 15 Mikrofone.

Zur Pegelmessung der Mikrofone wurde folgendes Equipment benutzt.

2.2.10.1 Ferrofish A16 MK-II

Für die Digital-Analog-Wandlung wurde ein *A16 MK-II* der Firma *Ferrofish* verwendet. Hierbei wurde das Multi Channel Audio Digital Interface (MADI) und XLR als Ausgang eingesetzt.

2.2.10.2 Genelec 8020C Lautsprecher

Die Wiedergabe erfolgte über einen einzelnen *8020C Lautsprecher* der Firma *Genelec*. Das Signal wurde über XLR eingespeist und die Verstärkung durch den Lautsprecher konstant gehalten.

2.2.11 LMS Studer Mic24ADAT

Für die Analog-Digital-Wandlung der Mikrofonsignale des in Abschnitt 2.2.10 genannten Mikrofon-Arrays wird der *Studer Mic24ADAT* benutzt, welcher auch eine Eigenentwicklung des LMS ist. Von den 24 möglichen Kanälen wurden nur 15 verwendet. Das Gerät gibt die Messdaten in ADAT aus.

2.2.12 Omni-Mikrofon-Array

Das Omni-Mikrofon-Array ist aus acht Mikrofonen der *Blue Line Series* der Firma *AKG* aufgebaut. Jedes dieser Mikrofone besteht aus dem *SE300B* Vorverstärker und dem omnidirektionalen Mikrofon-Kopf *CK92*.

Zur Pegelmessung der Mikrofone wurden wiederum die aus Abschnitt 2.2.10.1 und 2.2.10.2 bekannten Geräte benutzt.

Für die Pegelmessung wurden die Mikrofone in relativ kleinem Abstand zueinander auf eine Stange montiert. Für die eigentlichen Messungen wird der Abstand zwischen den Mikrofonen erweitert werden.

2.2.13 RME Micstasy

Das *RME Micstasy* wird als Analog-Digital-Wandler benutzt. Es ist damit möglich die acht Mikrofon-Kanäle mit 24 Bit Auflösung und 192kHz Abstastrate zu wandeln und jedem Kanal unabhängig von den Anderen ein 'Kalibrierungs'-Gain zu geben.

2.2.14 RME Advanced Digital Interface ADI-648 MK II

Zur Wandlung von ADAT nach MADI wurde ein *Advanced Digital Interface ADI-648 MK II* der Firma *RME* benutzt. Dessen Grenzen bei der Wandlung liegen bei 24Bit Auflösung und 192kHz Abtastrate. In den Messungen wurde jedoch nur 44,1kHz Abstastrate benutzt.

2.2.15 Lautsprecher-Array (128)

Im Audiolabor (2.1.2) ist ein Array mit 128 Lautsprechern verbaut, welches genauso wie der in Abschnitt 2.2.9 genannte Prototyp aufgebaut ist, jedoch mehr Einheiten beinhaltet. Außerdem ist dieses Array nicht in einer Linie aufgebaut, sondern im Audiolabor ringsum an der Wand verbaut.

2.3 Software

In diesem Abschnitt wird auf die Software eingegangen, mit der gemessen wurde. Außerdem werden kurz die Erfahrungen und Besonderheiten im Umgang mit dieser Software beschrieben.

2.3.1 MATLAB

Die Basis aller Messungen war die Software *MATLAB* der Firma *The MathWorks*. Ursprünglich zur Lösung numerischer Probleme anhand von Matrizen entwickelt, ist MATLAB heutzutage ein Programm mit dem auch Tools mit grafischer Oberfläche,

zur automatisierten Messung und Darstellung der Messdaten komfortabel realisierbar sind. Außerdem gibt es ganze Toolboxen mit denen Modelierungen sehr einfach zu machen sind, wie zum Beispiel *Simulink*. Diese wurden jedoch in diesen Messungen nicht verwendet.

2.3.2 MeasurementToolRIR

Gemessen an sich wurde mit dem sogenannten *MeasurementToolRIR*, welches eine Eigenentwicklung des LMS darstellt. Dies ist ein Tool für MATLAB, welches eine grafische Benutzer-Oberfläche hat, mit dem Raum-Impulsantworten in Szenarien mit mehreren Quellen und mehreren Mikrofonen anhand von Maximum-Length-Sequenzen (MLS) mit diverser Ordnungen gemessen werden können.

2.3.3 Nanostep

Die in MATLAB-Sprache geschriebene Klasse *Nanostep* ist wiederum eine Eigenentwicklung des LMS und ist dazu gedacht den in Abschnitt 2.2.2 beschriebenen Nanostep Smart anzusteuern. Nachdem die anfänglichen Geschwindigkeits- und Timeout-Probleme behoben waren, ließ sich die Klasse sehr gut verwenden.

2.3.4 Stepper

Die in MATLAB-Sprache geschriebene Klasse *Stepper* ist auch eine Eigenentwicklung des LMS und ist dazu gedacht die in Abschnitt 2.2.3 beschriebene, ebenfalls vom LMS entwickelte, Schrittmotorsteuerung anzusprechen.

Die Funktionalität der Klasse an sich war gegeben, jedoch galt es die Fehler und Eigenheiten der Klasse zu umgehen oder auszugleichen.
Wie schon im Abschnitt 2.2.3 beschrieben, war der Ursprung nach nahezu jeder Kalibrierung an einem anderen Ort, sodass es notwendig war eine Kompensations-Variable einzuführen und darüber eine händische Ausrichtung auf den geforderten Ursprung

auszuführen. Es ließ sich nicht feststellen, ob dies ein Software- oder ein Hardware-Fehler ist.

Des Weiteren ist bei der Kalibrierung darauf zu achten, dass der Schrittmotor im schlimmsten Fall sich einmal komplett um die eigene Achse dreht und somit Kabel, die eventuell an einem Gerät hängen, welches durch den Schrittmotor gedreht wird, ebenso um diese Achse gedreht werden. Wenn dies der Fall ist, sollte man den Schrittmotor einmal komplett in Gegenrichtung drehen lassen um ein Aufwickeln der Kabel zu vermeiden. Dabei ist zu beachten, dass der Klasse nur absolute Positionen übergeben werden können und die Klasse sich selbst den kürzesten Weg dorthin sucht. Das heißt eine komplette Drehung muss in drei Schritten ausgeführt werden, wobei im zweiten Schritt der gegenüber liegende Punkt der ursprünglichen Position durch einen kürzesten Weg überschritten werden muss.

Der für die automatisierten Messungen kritischste Punkt war jedoch, dass die Klasse nicht so geschrieben ist, dass nach einem Funktionsaufruf an eine bestimmte Position zu fahren, die Funktion nicht solange wartet, bis der Schrittmotor an dieser Position angekommen ist. Als Folge davon war es notwendig, in das Script für die automatisierten Messungen nach jeder Anweisung für eine Drehung eine Wartezeit einzubauen. Hierbei haben sich 0,4 Sekunden pro Grad Drehung als ausreichend erwiesen.

Kapitel 3

Projekt: HRTFs - Head Related Transfer Functions

In diesem Kapitel wird näher auf die Messungen für das Projekt *Head Related Transfer Functions (HRTFs)* eingegangen.

In bisherigen Simulationen wurde der Kopf eines Menschen als Kugel modelliert, ohne die eigentliche Kopfform, den Hals oder den Brust-Schulter-Bereich mit einzubeziehen. Um diesem Umstand zu begegnen sollte ein Kunstkopf vermessen werden und HRTFs erzeugt werden. HRTFs, im Deutschen *Kopfbezogene Transferfunktionen*, beschreiben, wie der Name schon sagt, die Impulsantwort eines Kopfes. Da es bei den HRTFs auf die Richtung und Distanz ankommt sind mehrere Messungen notwendig. Daraus lässt sich für zukünftige Simulationen ein Modell erzeugen, welches in Abhängigkeit von(Richtung und Distanz, eine bessere Approximation des menschlichen Kopfes darstellt.

Zuerst wurde ein Messaufbau ausgearbeitet. Dafür wurden Änderungen und Erweiterungen der vorhandenen Software notwendig. Außerdem wurde eine Checkliste für die Messungen erstellt.

3.1 Messaufbau

Die Messungen der HRTFs wurden in der LRC, bekannt aus 2.1.1, durchgeführt. In Abbildung 3.1 ist der Aufbau zu sehen. Dort ist die Audio-Hardware in der Farbe Schwarz dargestellt, also das in 2.2.9 beschriebene Lautsprecher-Array und der aus 2.2.4 bekannte Kunstkopf.

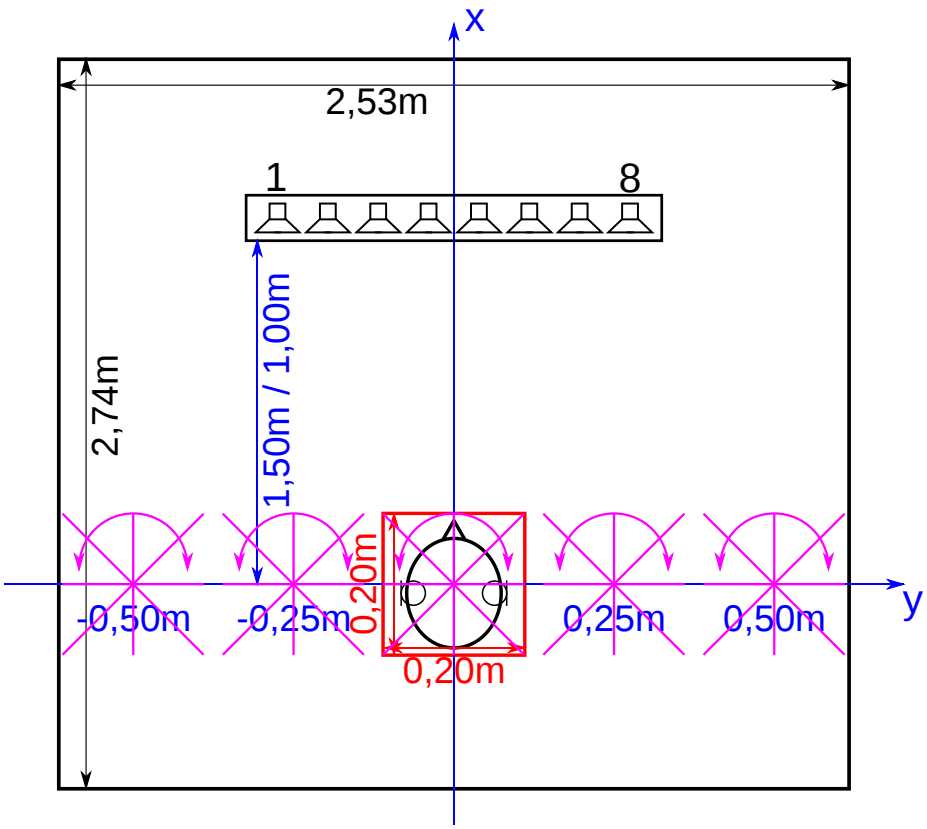

Abbildung 3.1: Raum Aufbau

Das Raum-Koordinatensystem ist in der Farbe Blau dargestellt. Der Mittelpunkt des in 2.2.1 vorgestellte X-Y-Tisches, hier in der Farbe Rot dargestellt, wurde in zwei Distanzen, $1,00$m und $1,50$m, zum Lautsprecher-Array auf verschiedenen Positionen, 0m, $\pm 0,25$m und $\pm 0,50$m in y-Richtung vom Mittelpunkt des Lautsprecher-Arrays, aufgestellt.

Auf jeder dieser Positionen wurden jeweils die Hauptachsen des X-Y-Tisch-Koordinatensystems in einem $0,01$m Raster durchschritten, sowie die Diagonalen in einem $\sqrt{2} \cdot 10^{-2}$m Raster, sodass die Reichweite des X-Y-Tisches voll ausgenutzt wurde. Außerdem wurde in der Mitte des Tisches der Kopf durch den Rotationsschrittmotor von $-90°$ bis $90°$ in $10°$ Schritten gemessen. Die doppelte Messung des Mittelpunktes mit $0°$ Ausrichtung war nicht beabsichtigt. Diese Raster sind in der Abbildung durch die Farbe Pink angedeutet.

Pro Schritt der genannten Raster wurden auf beiden Ohr-Mikrofonen des Kunstkopfes gleichzeitig alle acht Lautsprecher einzeln nacheinander vermessen. Diese Funktionalität stellt das in 2.3.2 vorgestellte Tool zur Verfügung, sodass nur die Steuerbefehle an den X-Y-Tisch oder den Rotationsschrittmotor gesendet und danach die Messung ausgelöst werden musste.

Die Verkabelung der Komponenten ist dem Punkt C.1.1 *Cabeling* zu entnehmen.

3.2 Software Änderungen

Die in 2.3.2 vorgestellte Software ist an sich ein sehr brauchbares Tool um Room impulse responses (RIRs) mit M Mikrofonen und N Lautsprechern zu messen. Im Zuge dieses Forschungspraktikums wurde es notwendig den Funktionsumfang zu erweitern.

Einerseits wurde eine Kohärenz-Visualisierung eingebaut, um herauszufinden, ob die Messdaten überhaupt plausibel sind.

Außerdem wurde um das Tool herum ein Set von MATLAB-Skripten geschrieben, welche viele Einzelmessungen koordinieren und zwischen den Messungen die Tisch-Bewegungen durchführt.

Dabei wurden ein paar Workarounds entdeckt, durch welche die Arbeit mit dem Tool vereinfacht wurde.

3.2.1 Kohärenzmessung

Das Skript zur Kohärenzmessung findet sich im selben Ordner wie das in 2.3.2 beschriebene MeasurementToolRIR unter dem Namen `measureCoherence.m`. Alternativ ist es im Anhang A.1 zu finden.

Das Skript basiert auf dem `measureImpulseResponse.m`-Skript, jedoch ist es sehr spezifisch für das HRTFs Projekt angepasst und somit auf ein Zwei-Mikrofon-Szenario ausgelegt. Für ein N-Mikrofon-Szenario müsste erst spezifiziert werden, welche Kohärenzen von Interesse wären.

In dem Skript wird zuerst eine Fensterung aufgrund der Signallänge abgeschätzt und die Signale durch die MATLAB-Funktion `mscohere` ausgewertet. Nach erfolgreichem Durchlauf werden die Ergebnisse in einem Plot visualisiert. Das dazugehörige Skript ist im Anhang A.2 zu finden.

3.2.2 Automatisierung

Um den Zeitaufwand der Messungen deutlich zu reduzieren und von den in 2.1.1 genannten Problemen zu entkoppeln, wurde der Messablauf einer X-Y-Tisch-Position im Raum vollständig automatisiert. Damit wurde es möglich zu Nachtzeiten zu messen, ohne den Messvorgang selbst zu überwachen.

Die Automatisierung der Messung gliedert sich in sechs Einzelskripte, die dazugehörigen Skripte sind im Anhang B zu finden. Die sechs Einzelskripte sind:

- `automatedMeasurement.m`

- `measureTablePosition.m`
- `moveToPositionsAndMeasure.m`
- `measureRotation.m`
- `checkAutomatedMeasurement.m`
- `strendswith.m`

3.2.2.1 automatedMeasurement.m

Das Skript `automatedMeasurement.m` ist der Startpunkt der Messung. Hier wird als Erstes die Hardware initialisiert und kalibriert. Danach werden alle notwendigen Parameter eingestellt. Da die Messung selten und aus unerklärlichen Gründen abstürzte, wird zu Debugging-Zwecken die `diary` Funktion von MATLAB benutzt, welche alle Ein-, Ausgaben und Exceptions mitprotokolliert. Nach einer Wartezeit wird dann der erste Teil der Messung ausgeführt. Dieser ist in 3.2.2.2 zu finden. Danach wird die Mitte des X-Y-Tisches angefahren und die Messungen durchgeführt, zwischen denen Rotations-Bewegungen ausgeführt werden. Dieser Teil ist in 3.2.2.4 zu finden. Danach wird noch die benötigte Zeit ausgegeben und das Diary beendet.

3.2.2.2 measureTablePosition.m

Hier wird die Funktion `measureTablePosition` beschrieben. Der Parameter `trans` muss ein initialisiertes und kalibriertes Objekt der Klasse Nanostep (2.2.2) sein, mit dem der X-Y-Tisch bewegt werden kann. Der Parameter `tablePosition` stellt die Position des X-Y-Tisches in Raumkoordinaten dar. Die Parameter `measureMainAxes` und `measureDiagonals` sind Wahrheitswerte und legen fest, ob das Raster an den Hauptachsen bzw. den Diagonalen während der Messungen durchlaufen werden soll. Der Parameter `positions` ermöglicht es noch einzelne Positionen in Tischkoordinaten zu übergeben, die in gegebener Reihenfolge angefahren und gemessen werden.

Zuerst werden in Abhängigkeit von den o.g. Wahrheitswerten die Raster erzeugt. In der Mitte des Tisches werden, um höhere Genauigkeit zu erhalten, zusätzliche Punkte mit halbem Rasterabstand eingefügt. Anschließend wird sowohl für die erzeugten Rasterpunkt-Arrays, wiederum in Abhängigkeit von o.g. Wahrheitswerten, sowie das übergebene `positions`-Array die Funktion `moveToPositionsAndMeasure` aufgerufen.

3.2.2.3 moveToPositionsAndMeasure.m

Hier wird die Funktion `moveToPositionsAndMeasure` beschrieben. Die Position des X-Y-Tisches in Raumkoordinaten wird in dem Parameter `tablePosition` abgelegt. Der Parameter `trans` muss ein initialisiertes und kalibriertes Objekt der Klasse Nanostep (2.3.3) sein, mit dem der X-Y-Tisch bewegt werden kann. Der Parameter `positions` ist eine N×2-Matrix, welche 2D-Punkte enthält, die in gegebener Reihenfolge angefahren und gemessen werden.

Zuerst wird ein Handle auf die GUI des aus 2.3.2 bekannten Tools ermittelt. Darüber können dann die notwenigen Metainformationen dem Tool übergeben werden. Für jede Zeile der `positions`-Matrix werden folgende Vorgänge ausgeführt. Es wird die beschriebene Tischposition angefahren und darauf gewartet, dass das System zur Ruhe kommt. Die für diese Einzelmessung relevanten Metadaten werden geschrieben und eine Messung ausgelöst. Jeglicher Fehler der währendessen auftritt, wird in eine eigene Fehlerdatei geschrieben, zusammen mit Hinweisen, wann und wo der Fehler aufgetreten ist, danach wird mit der nächsten Einzelmessung fortgefahren. Da das Tool Visualisierungen pro Messung öffnet, wird jede `figure` bis auf die GUI selbst geschlossen. Abschließend werden die Metatdaten für die nächste Messung vorbereitet.

3.2.2.4 measureRotation.m

Hier wird die Funktion `measureRotation` beschrieben. Der Parameter `rot` muss ein initialisiertes und kalibriertes Objekt der Klasse Stepper (2.3.4) sein, mit dem der Rotationsschrittmotor bewegt werden kann. Die Position des X-Y-Tisches in Raumko-

ordinaten wird in dem Parameter `tablePosition` abgelegt. Der Parameter `rotCalib` ist der Kalibierungswert aus 3.2.2.1, er wird benötigt um die Metadaten in der Messung richtig einzufügen. Die Parameter `rotStart`, `rotStop` und `rotStop` legen Start- und Stop-Wert und Schrittweite des Rotations-'Rasters' in Grad fest.

Wie schon in 3.2.2.3 wird zuerst ein Handle auf die GUI des aus 2.3.2 bekannten Tools ermittelt. Danach werde die für alle Einzelmessungen relevanten Metadaten gesetzt. Bei jeder Einzelmessung werden folgende Vorgänge ausgeführt. Die neue Rotationsposition wird angefahren, danach wird pro gefahrenem Grad eine gewisse Zeit gewartet, da die Stepper-Klasse das nicht von sich aus tut. Danach wird noch gewartet bis das System wieder zur Ruhe kommt. Nachdem die Metadaten für die Einzelmessung gesetzt wurden, wird die Messung an sich ausgelöst, wiederum jegliche Fehler abfangend und protokollierend. Wiederum werden die Visualisierungen der Einzelmessung geschlossen und die Metadaten für die nächste Messung vorbereitet.

3.2.2.5 checkAutomatedMeasurement.m

Nachdem eine Messung durchgeführt wurde, und keine Error-Report-Files in dem Ordner `automatedMeasurements` zu finden sind. Lässt sich noch mit der Funktion `checkAutomatedMeasurement` ein Ordner überprüfen und visualisieren, welche Messungen dort vorgefunden wurden.

Dazu wird zuerst eine Liste der in dem Ordner vorhandenen Dateien erzeugt. Für jede MATLAB-Datei darin werden folgende Vorgänge ausgeführt. Es wird sicher gestellt, dass keine Variable `measurements` existiert. Danach wird die Datei geladen und sichergestellt, dass jetzt eine Variable `measurements` existiert. Danach wird ein Index hochgezählt und mit dem in der Datei abgelegten `measurementIndex` verglichen. Außerdem wird noch die absolute Position gespeichert und danach die Variable `measurement` wieder gelöscht, für den nächsten Einzelcheck. Nachdem alle Einträge abgearbeitet wurden, wird die absolute Position in einem 3D-Plot dargestellt, wobei die Z-Achse im Plot je-

doch Rotationswerte darstellt. So ist eine einfache visuelle Überprüfung möglich, ob noch Werte fehlen.

3.2.2.6 strendswith.m

Für die in 3.2.2.5 beschriebene Funktion war eine weitere kurze Funktion `strendswith` notwendig, welche überprüft ob ein String mit einem Muster endet. Hier wurde keine eigene Implentierung benutzt, sondern auf eine Implementierung von Dahua Lin [1] aus dem MATLAB CENTRAL FileExchange zurück gegriffen.

3.2.3 Workarounds

An sich ist das MeasurementToolRIR ein brauchbares Tool um RIRs zu Messen. Daher würde es sich lohnen das Tool weiter zu ausgereifterer Software zu entwickeln. Hierbei sind die gefundenen Workarounds ein guter Indikator wo in erster Linie angesetzt werden sollte.

Es hat sich bei automatisierten Messungen als hilfreich erwiesen, den einzelnen Messaufruf dieses Tools in einen try-catch-Block einzufassen, da in manchen Fällen Programmabstürze auftraten, deren Ursache nicht identifiziert werden konnte.So geht bei einem Fehler nur eine Einzelmessung verloren, aber nicht der ganze Messdurchlauf. Eine Einzelmessung kann auch im Nachhinein nachgeholt werden.

Auch die Speicher- und Lade-Funktion der aktuellen Einstellungen sollte untersucht werden und dabei vor allem Multi-Benutzer-Szenarien bedacht werden. Hierbei sind vor allem Probleme mit relativen und absoluten Pfaden und Schreibrechten aufgetreten. Workaround war hierbei, nur einen Account zu benutzen.

[1] http://www.mathworks.com/matlabcentral/fileexchange/21710-string-toolkits/content/strings/strendswith.m

Auch die Wahl des Speicherortes der Messungen und die dabei vorhandenen Restriktionen sollten überdacht werden, oder zumindest kommuniziert werden. Die Checkliste in 3.3 gibt auch nochmal Hinweise wo Probleme versteckt sein könnten. Vermutlich sind auch hier relative und/oder absolute Pfade das Problem. Dies konnte nicht umgangen werden, das Output-Directory musste bei `../MeasurementToolRIR_measurements/measurements/` belassen werden.

Bei den Pfadeingaben verhält sich außerdem das Verlassen des Textfeldes nach manueller Pfadeingabe und die Pfadeingabe über den Button unterschiedlich, was nicht unbedingt der Benutzer-Intuition folgt.

Des weiteren wäre es allgemein wünschenswert, wenn im Code die Darstellung von Werten besser von der Berechnung von Werten getrennt wäre. Dies würde die Fehleranalyse und vor allem zukünftige Weiterentwicklung vereinfachen.

3.3 Checkliste

Nachdem die Messung automatisiert wurde und damit die Probleme der LRC umgangen werden konnten, gab es Überlegungen das Starten und Stoppen der automatisierten Messungen in dritte Hände abzugeben. Hierfür wurde eine Checkliste erarbeitet, welche im Anhang C zu finden ist. Um Sprachunabhängig zu sein, wurde diese Checkliste in Englisch verfasst.

Kapitel 4

Projekt: Impulsantwort des Audiolabors

In diesem Kapitel wird näher auf die Messungen des Projekts *Impulsantwort des Audiolabors* eingegangen.

In bisherigen Simulationen, zum Beispiel für Präsentationen, wurde der Raum des Audiolabors durch vereinfachende Modelle angenähert. Durch Vermessung der tatsächlichen Impulsantwort sollen die Ergebnisse der Simulationen verbessert werden.

Zuerst wurde ein Messaufbau ermittelt, welcher dann in den tatsächlichen Messungen umgesetzt werden sollte.

4.1 Ermittlung des Messaufbau

Es sollte das DICIT-Array für die Messungen benutzt werden um die IR eines relativ großen Bereichs innerhalb des Audiolabors ermitteln zu können. Da die Eigenschaften des DICIT-Arrays im Hinblick auf IR-Messungen nicht bekannt waren, musste erst die Tauglichkeit des DICIT-Arrays und, nachdem dieses als untauglich befunden wurde, die des Omni-Mikrofon-Arrays festgestellt werden.

Darauf aufbauend soll dann ein Messaufbau erarbeitet werden.

4.1.1 Tauglichkeit des DICIT-Array

Um vergleichbare Werte zu bekommen, musste als erstes festgestellt werden, ob die einzelnen Mikrofone des Arrays annähernd gleiche Pegel liefern. Außerdem sollte ermittelt werden, ob die Richtung des DICIT-Arrays maßgeblichen Einfluss auf die gemessene IR hat.

4.1.1.1 Pegelmessung

Die Pegelmessung wurde mit folgenden Geräten durchgeführt:

- Genelec 8020C Lautsprecher
- Ferrofish A16 MK-II
- RME Advanced Digital Interface ADI-648 MK II
- LMS Studer Mic24ADAT
- DICIT-Array

Der Aktiv-Lautsprecher wurde als Punktquelle benutzt und war über ein XLR auf 6,3-Klinke Kabel an den A16 MK-II angeschlossen, wo die Digital-Analog-Wandlung stattfand. Gespeist wurde dieser über ein optisches Kabel über ADAT vom ADI-648, welcher die Wandlung von MADI nach ADAT übernahm. Hier wurden auch die DICIT-Array Mikrofonsignale, welche vom Studer Analog-Digital gewandelt wurden, von ADAT wiederum nach MADI übersetzt, sodass dass MADI-Interface des Messcomputers angesprochen werden konnte.

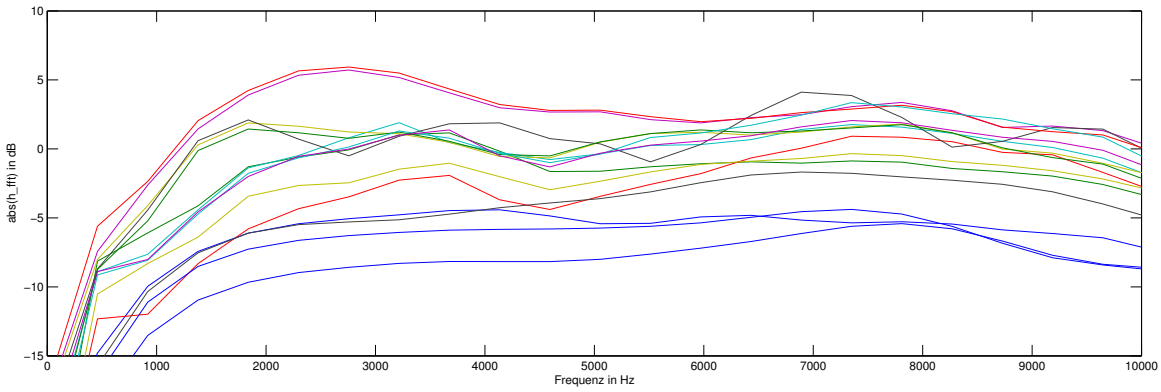

Abbildung 4.1: Betragsfrequenzgang DICIT, mittlere Position

In einem ersten Versuch wurde das DICIT-Array an seiner Position belassen und der Lautsprecher gegenüber jedem Mikrofon in $\sim 1,5$m Abstand aufgestellt. Für jede dieser Positionen wurden alle Mikrofone vermessen. Aufgrund einer Art logarithmischer Verteilung der einzelnen Mikrofone im Array zur Mitte hin, war die Verschiebung nicht genau genug, um Pegelgleichheit erkennen zu können. Außerdem werden hier eigentlich unterschiedliche RIRs verglichen, sodass eine Vergleichbarkeit von vorneherein nicht gegeben war.

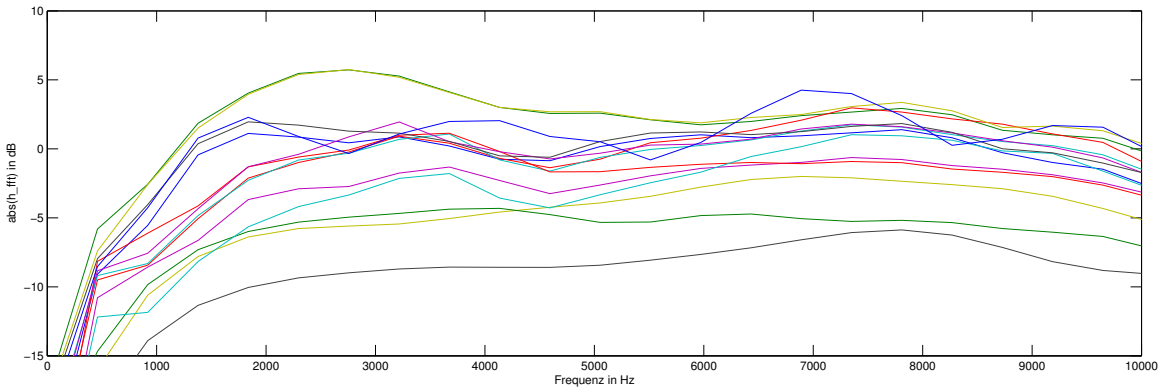

Abbildung 4.2: Betragsfrequenzgang DICIT, mittlere Position, Stecker vertauscht

Deswegen wurde in einem zweiten Versuch der Lautsprecher an seiner Position belassen und das Array verschoben. Hierbei stellte sich außerdem noch heraus, dass wenn schon jeweils jedes Mikrofon gemessen wird, es eigentlich ausreicht, die beiden Enden und die

Mitte des Arrays gegenüber dem Lautsprecher zu haben. Zur Sicherheit wurde noch jede Messung mit den beiden Stecker des Arrays am Studer vertauscht durchgeführt um Aussagen über die Symmetrie machen zu können. Die Ergebnisse hierzu sind in den Abbildungen 4.1 und 4.2 zu sehen.

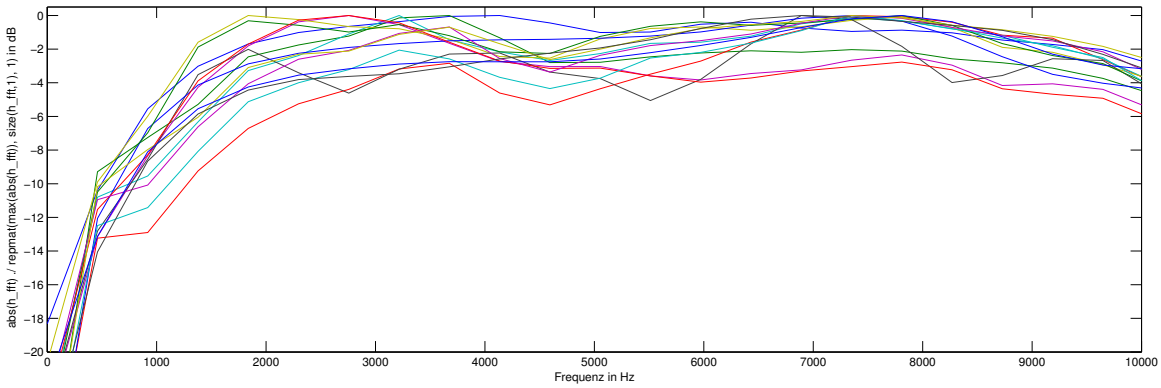

Abbildung 4.3: Maximalwert bereinigter Betragsfrequenzgang DICIT, mittlere Position

Wie in Abbildung 4.3 zu sehen, sorgen die je nach Frequenz unterschiedlichen Abweichungen der Pegel der einzelnen Mikrofone des Arrays dafür, dass das DICIT-Array für die vorliegende Problemestellung nicht als Messmikrofon-Array brauchbar ist.

4.1.1.2 Richtungsabhängigkeit

Es war geplant die Abhängigkeit der IR-Messung von der Richtung des Arrays festzustellen. Hierfür sollte mit dem Lautsprecher-Array des Audiolabors eine komplette RIR-Messung in Nord-Süd-Richtung an derselben Stelle mit einer Messung in Süd-Nord-Richtung verglichen werden und die Abschattung durch das Array selbst ermittelt werden. Jedoch schon die Pegelmessung in Abschnitt 4.1.1.1 zeigt, dass die Verwendung des DICIT-Arrays hinfällig ist, somit auch der Test der Richtungsabhängigkeit.

Norden und Süden entsprechen hier nicht den tatsächlichen Himmelsrichtungen, sondern den in der Abbildung 4.6 Eingezeichneten.

4.1.2 Tauglichkeit des Omni-Mikrofon-Array

Dank der omnidirektionalen Mikrofon-Köpfe der Mikrofone ist dieses Array Richtungsunabhängig und somit entfällt dieser Test bei diesem Mikrofon-Array.

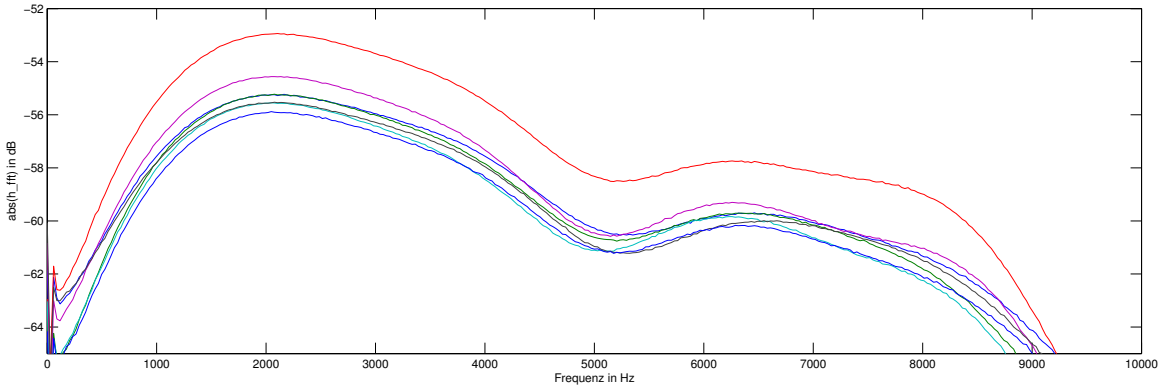

Abbildung 4.4: Betragsfrequenzgang Omni-Array

Für die Pegelmessung wurden die Mikrofone in einem $\sim 4{\cdot}10^{-2}$m Raster auf einer Stange montiert. Sie sollten relativ nah bei einander sein, damit die Laufzeitunterschiede zu dem $\sim 2,5$m entfernten Lautsprecher nicht zu groß werden, aber jedoch nicht so nah, dass sich die Mikrofone gegenseitig beeinflussen.

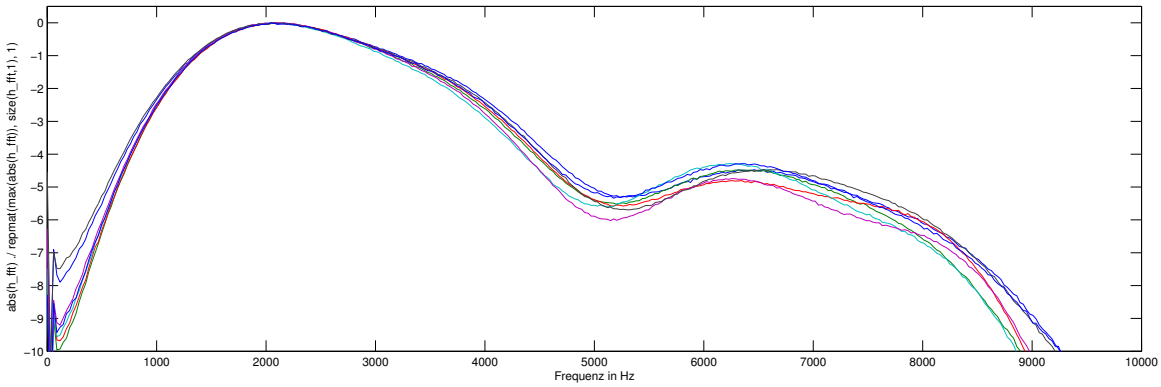

Abbildung 4.5: Maximalwert bereinigter Betragsfrequenzgang Omni-Array

Abbildung 4.4 zeigt die Abweichung der Mikrofone untereinander. Hierbei ist gut zu sehen wie der Betragsfrequenzgang der Mikrofone (in dem für diese Messungen relevanten Bereich) nahezu parallel verlaufen. Um dies zu verdeutlichen ist in Abbildung

4.5 zu sehen, wie der Betragsfrequenzgang von seinem Maximalwert abweicht. Es zeigt dass ideale Vorraussetzungen vorliegen, mit einer (nachträglichen) Kalibierung die Mikrofone auf gleiche Pegel zu bekommen. Insofern können die Messungen mit diesen Mikrofonen gemacht werden.

4.1.3 Geplanter Messaufbau

Das Lautsprecher-Array wird direkt vom Computer aus angesprochen. Die Signale des Omni-Mikrofon-Arrays werden zuerst im RME Micstasy analog digital gewandelt und dann wiederum über MADI an den Computer übertragen.

Die Planung sieht vor, das Omni-Mikrofon-Array an zwei Positionen im Audiolabor auf Nord-Süd- und Ost-West-Achse aufzustellen. Dazwischen sollte noch eine Position mit Ost-West-Achse sein und daneben zwei weitere Positionen mit Nord-Süd-Achse. Die genauen Positionen sind der Abbildung 4.6 zu entnehmen. Die Abbildung ist jedoch noch auf das DICIT-Array ausgelegt, die Positionen sind aber analog auf das Omni-Mikrofon-Array anzuwenden.

Über die Gesamtheit der Messungen an diesen Positionen soll versucht werden, eine positionsabhängige RIR für einen relativ großen Bereich zu erzeugen. Deswegen sollte auch, im Gegensatz zu der Pegelmessung, für den eigentlichen Messaufbau der Abstand zwischen den Mikrofonen größer gewählt werden, um mehr Raum abzudecken.

4.2 Tatsächliche Messung

Da die tatsächlichen Messungen nicht mehr im Zeitrahmen dieses Forschungspraktikums durchführbar sind, sollen hier die Erfahrungen, die während der Testmessungen gemacht wurden, dokumentiert werden.

Es wurde festgestellt, dass mit dem Computer des Audiolabors, nicht alle 128 Laut-

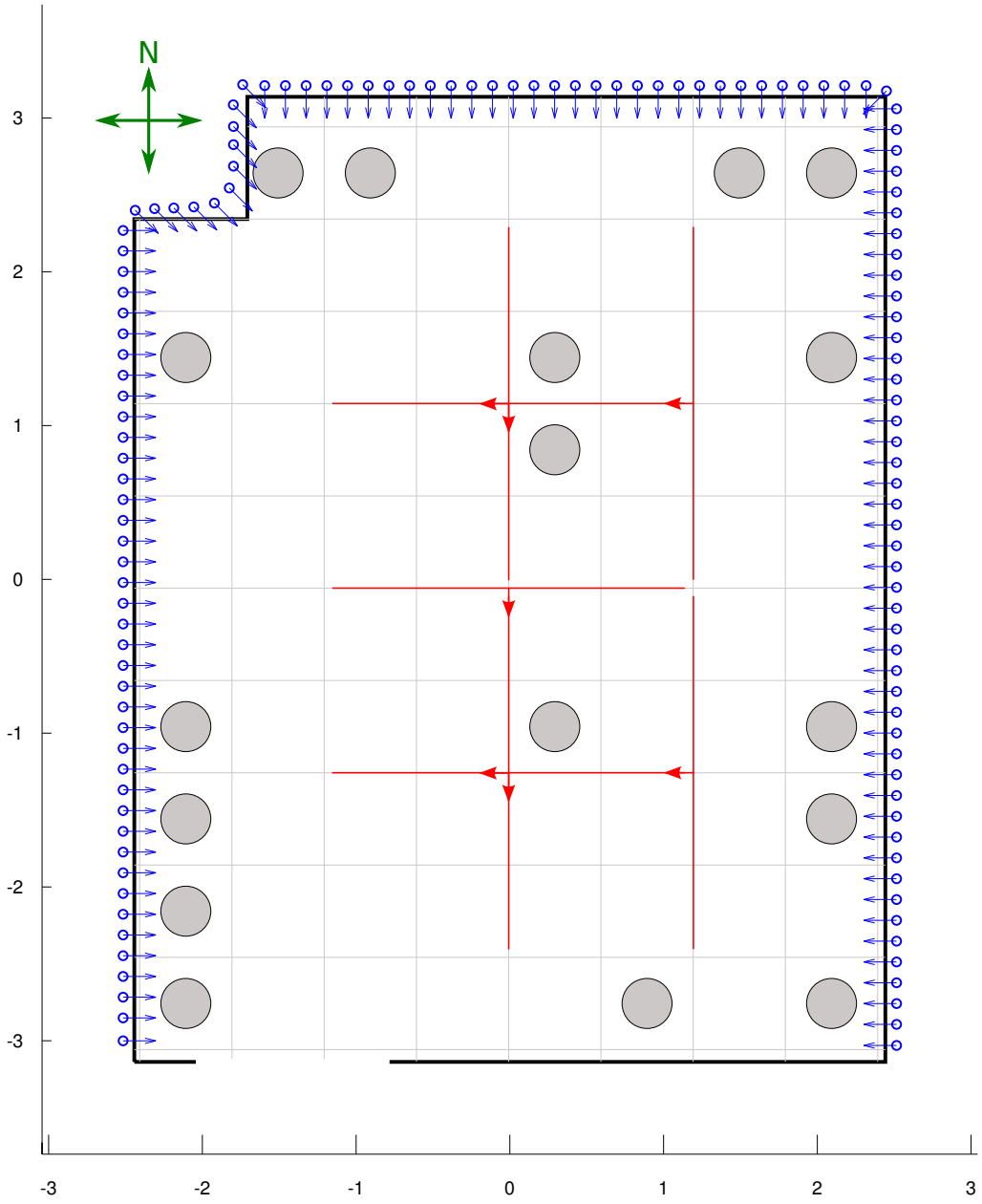

Abbildung 4.6: Array Positionen

sprecher in einer Messung gemessen werden können. MATLAB stößt hier scheinbar mit seiner umfangreichen Speicherbenutzung an die Grenzen des Arbeitsspeichers des Computers. Deswegen wird der Messvorgang in vier Einzelmessungen aufgeteilt. Leider lässt sich dieser Vorgang aufgrund eines `uiwait` in der GUI des MeasurementToolRIR nicht oder nur mit erheblichem Aufwand automatisieren.

Anhang A

Kohärenzmessung

A.1 measureCoherence.m

```
function measureCoherence (parameters)
  import structs.*;
  listOfParameters = fieldnames(parameters);
  for i = 1:length(listOfParameters)
    unbundleStruct(parameters.(listOfParameters{i}));
  end
  % Create timestamp
  dateVector = datevec(now);
  timestamp = datestr(dateVector, 'yyyy-mm-dd_HH-MM-SS');
  outFilePrefix = [dirs.output '/' timestamp];

  % Ensure numbers as specifiers
  [devOut, devIn] = playrec_devicestring2ID(devOut, devIn); %#ok<NODEF>
  % Ensure correct initialization
  if ~playrec_validateInit(samplerate, devOut, devIn,...
      nrAllocatedOutChannels, nrAllocatedInChannels, framesize)
    if playrec('isInitialised');
      playrec('reset');
```

```
19        pause(0.5);
20      end
21      playrec('init', samplerate, devOut, devIn, nrAllocatedOutChannels,...
22          nrAllocatedInChannels, framesize); pause(0.15);
23    end
24    playrec ('resetSkippedSampleCount');
25
26    % Pre-allocate arrays
27    x = zeros(lenPeriod, nrUsedInChannels); % The final signals
28    pageNr = zeros(nrPeriods,1);
29    P_silenceDigital = zeros(1,nrUsedInChannels);
30    P_silenceDigital_dB = zeros(1,nrUsedInChannels);
31    P_inDigital = zeros(1,nrUsedInChannels);
32    P_inDigital_dB = zeros(1,nrUsedInChannels);
33    tableP_inDigital = zeros(nrSequences,nrUsedInChannels);
34    tableP_inDigital_dB = zeros(nrSequences,nrUsedInChannels);
35    tableSNR_recording_dB = zeros(nrSequences,nrUsedInChannels);
36    skippedSampleCount = zeros(nrSequences,nrPeriods);
37
38    %% var init
39    Cxy = cell(nrUsedOutChannels);
40    F = cell(nrUsedOutChannels);
41
42    % Allocate temporary files
43    tmpFileNames = cell(1,nrSequences);
44    for iSeq = 1:nrSequences
45      tmpFileNames{iSeq} = [dirs.tmp '/rec_sequence' num2str(iSeq) '.'...
46          num2str(nrUsedInChannels) 'ch.single.pcm'];
47    end
48
49    %% Audio IO
50
51    % Initiate output of Silence period
52    lenSilence = initialSilence * samplerate;
53    s_silence = zeros(lenSilence, nrUsedInChannels);
```

```
54  silenceMeasured = false;
55  pause(initialPause);
56  iPageSilence = playrec('playrec', zeros(lenSilence,...
57      nrUsedOutChannels), listOutChannels, -1, listInChannels.');
58
59  % Initiate output and recording of test sequence
60  SNR_aliasingMLS_dB = 20*log10(2^orderMLS-1);
61  fprintf(1,'SNR_aliasingMLS_dB = % 7.3f \n', SNR_aliasingMLS_dB);
62
63  % Initialize struct accomodating series measurements
64  coherenceMeasurement = struct([]);
65
66  for iPlayDevCh = 1 : nrUsedOutChannels
67    hWaitbar = waitbar(0.0,'Measurement Progress');
68    fhTmpFiles = fopen_multi(tmpFileNames,'w');
69
70    for iSeq = 1:nrSequences
71      fprintf(1,'\n');
72      s_out = s_base{iSeq};
73      P_silenceDigital(1:end) = 0;
74      P_inDigital(1:end) = 0;
75      P_inDigital_dB(1:end) = 0;
76
77      % Initiate output and recording of test sequence
78      for iPage = 1:nrPeriods
79        pageNr(iPage) = playrec('playrec', s_out(1:lenPeriod, 1),...
80            listOutChannels(iPlayDevCh), -1, listInChannels.');
81        if(iPage) == 1
82          playrec('resetSkippedSampleCount');
83        end
84      end
85      pageNumberPostSilence = playrec('playrec', zeros(2*framesize,...
86          1), listOutChannels(iPlayDevCh), -1, listInChannels.');
87
88      if (~silenceMeasured)
```

```
89          % Receive background-noise measurement
90          playrec('block', iPageSilence); % wait for page to be ready
91          s_silence(:,:) = playrec('getRec', iPageSilence); % get silence
92          if forceMeanFreeSignals % remove DC component (conditional)
93            for iCh = 1:nrUsedInChannels
94              s_silence(:,iCh) = s_silence(:,iCh) - mean(s_silence(:,iCh));
95            end
96          end
97          % remove read-out page from memory
98          playrec('delPage', iPageSilence);
99
100         % Determine background-noise level
101         P_silenceDigital = sum(abs(s_silence).^2,1)/lenSilence;
102         P_silenceDigital_dB = 10*log10(P_silenceDigital);
103         fprintf(1,'Sequence   1  silence  levels [dB]: %s\n\n',...
104             sprintf('% 7.3f ',P_silenceDigital_dB));
105         silenceMeasured = true;
106       end
107
108       % Receive input & on-the-fly averaging
109       % guard periods
110       for iPage = 1:nrGuardPeriodsBeginning
111         playrec('block', pageNr(iPage)); % wait for page to be ready
112         playrec('getRec', pageNr(iPage)); % throw away guard data
113         playrec('delPage', iPage); % remove read-out page from memory
114         waitbar(((iSeq-1)*nrPeriods+iPage)/(nrSequences*nrPeriods),...
115             hWaitbar);
116       end
117
118       % useful data
119       for iPage = (1+nrGuardPeriodsBeginning):(nrPeriods)
120         playrec('block', pageNr(iPage)); % wait for page to be ready
121         skippedSampleCount(iSeq, iPage) =...
122             playrec('getSkippedSampleCount');
123         % fetch recorded data
```

```
124         [s_in, ~] = playrec('getRec', pageNr(iPage));
125         fwrite(fhTmpFiles(iSeq),s_in.','single');
126         for iCh = 1:nrUsedInChannels
127           s_in(:,iCh) = s_in(:,iCh) - mean(s_in(:,iCh));
128           if exist('noninterfleavedfilewriting','var')
129             fwrite(fhTmpFiles(iSeq),s_in(:,iCh),'single');
130           end
131         end
132         playrec('delPage', iPage); % remove read-out page
133         % accumulate average power
134         P_inDigital = P_inDigital + sum(abs(s_in).^2,1) / lenPeriod;
135         waitbar(((iSeq-1)*nrPeriods+iPage)/(nrSequences*nrPeriods),...
136             hWaitbar);
137       end
138
139       P_inDigital = P_inDigital / (nrUsefulPeriods);
140       P_inDigital_dB = 10*log10(P_inDigital);
141       SNR_recording_dB = P_inDigital_dB - P_silenceDigital_dB;
142       fprintf(1,'Sequence %3.0f recording levels [dB]: %s\n' , iSeq,...
143           cell2mat({sprintf('% 7.3f ',P_inDigital_dB)}));
144       fprintf(1,'Sequence %3.0f recording SNR  [dB]: %s\n' , iSeq,...
145           cell2mat({sprintf('% 7.3f ',SNR_recording_dB)}));
146
147       tableP_inDigital(iSeq, :) = (P_inDigital);
148       tableP_inDigital_dB(iSeq, :) = (P_inDigital_dB);
149       tableSNR_recording_dB(iSeq, :) = (SNR_recording_dB);
150
151       playrec('delPage');
152       pause (0.1);
153       % whos
154     end
155
156     %% Evaluate Recordings
157     % Average Levels
158     P_inDigital = mean(tableP_inDigital,1);
```

```
159     % geometric mean, here!
160     % P_inDigital_dB = mean(tableP_inDigital_dB,1);
161     % ALTERNATIVE: arithmetic mean, here!
162     P_inDigital_dB = 10*log10(P_inDigital);
163     SNR_recording_dB = mean(tableSNR_recording_dB,1);
164
165     close (hWaitbar);
166     fclose_multi(fhTmpFiles); % close written files
167     % reopen files in read mode
168     fhTmpFiles = fopen_multi(tmpFileNames,'r');
169
170     fseek_multi(fhTmpFiles,0,1);
171     fileLengths = ftell_multi(fhTmpFiles);
172     hWaitbar = waitbar(0,'Processing recorded data ...');
173     nrSamples = round(fileLengths(1)/4/nrUsedInChannels);
174     if ( nrSamples ~= lenPeriod*(nrPeriods-nrGuardPeriodsBeginning))
175       error('unexpected length');
176     end
177     for iCh = 1:nrUsedInChannels
178       waitbar((iCh-1)/nrUsedInChannels, hWaitbar,...
179           'Processing recorded data: reading from disk' );
180       fseek_multi(fhTmpFiles,(iCh-1)*4,-1);
181       data = fread_multi(fhTmpFiles, [nrSamples, 1],'single',...
182           4*(nrUsedInChannels-1));
183       if exist('noninterfleavedfilewriting','var')
184         fseek_multi(fhTmpFiles,(iCh-1)*4*lenPeriod,-1);
185         data = fread_multi(fhTmpFiles,Inf,'single');
186       end
187       waitbar((iCh-1)/nrUsedInChannels, hWaitbar,...
188           'Processing recorded data: estimating word length' );
189       tmp = data{1};
190       wordLengthADC(iCh) = estimateWordLength(tmp(1:min(length(tmp),...
191           1000)));
192
193       data = cell2mat(data);
```

```
194       for o = 1:trimFromEnsemble
195         data = removeOutliers(data, 1);
196       end
197       x(:,iCh) = trimmedMean(data,acceptQuantTrimmedMean);
198
199       for iSeq = 1:nrSequences
200         frewind(fhTmpFiles(iSeq));
201       end
202       waitbar(iCh/nrUsedInChannels, hWaitbar);
203     end
204
205     if (cutToLengthInSeconds > 0)
206       x = x(1:min(max(1,round(cutToLengthInSeconds*samplerate)),...
207           lenPeriod),:);
208     end
209     fclose_multi(fhTmpFiles); % close written files
210     close (hWaitbar);
211     delete (tmpFileNames{1:end});
212
213     % ruff estimation of window size (power of 2 the quarter of the
214     % signal length fits in)
215     window = bitshift(1, ceil(log2(length(x(:,1)))) - 2);
216     %window = 1024 * 8;
217     noverlap = window/2; % 50% overlap
218     nfft = window;
219
220     [Cxy{iPlayDevCh}, F{iPlayDevCh}] = mscohere(x(:,1), x(:,2),...
221         window, noverlap, nfft, samplerate);
222     %% Write measurements to file
223     % Preparing
224     usedRecDeviceChannels = listInChannels;
225     usedRecDevice = devs(devIn + 1).name;
226     usedPlayDeviceChannels = listOutChannels;
227     usedPlayDevice = devs(devOut + 1).name;
228     usedInputArrays = inputs(useOfArrays(listInChannels));
```

```
229     usedInputArrayChannels = useOfArrayChannels(listInChannels);
230     usedConvertersAD = convertersAD(useOfConvertersAD(listInChannels));
231     usedConvertersDA = convertersDA(converterDA_index);
232     usedOutputArrays = outputs(outArrayIndex);
233     usedOutputArrayChannels = 1; %useOfArrayChannels(usedIOChannels);
234
235     metadata = repmat(struct(),nrUsedInChannels,1);
236     for iCh = 1:nrUsedInChannels
237       recDeviceChannel = listInChannels(iCh);
238       arrayIndexOfChannel = useOfArrays(recDeviceChannel);
239       % Setup-Specific
240       metadata(iCh).playDevice = usedPlayDevice;
241       metadata(iCh).playDeviceChannel =...
242           usedPlayDeviceChannels(iPlayDevCh);
243       metadata(iCh).converterDA = usedConvertersDA(1);
244
245       metadata(iCh).inputArray = usedInputArrays(iCh);
246       metadata(iCh).inputArrayPosition =...
247           inputArrayTranslation{arrayIndexOfChannel,:};
248       metadata(iCh).inputArrayRotation =...
249           inputArrayRotation{arrayIndexOfChannel};
250       metadata(iCh).converterAD = usedConvertersAD(iCh);
251       metadata(iCh).recDevice = usedRecDevice;
252       metadata(iCh).recDeviceChannel = usedRecDeviceChannels(iCh);
253       metadata(iCh).inputArrayChannel = usedInputArrayChannels(iCh);
254       metadata(iCh).timestamp = timestamp;
255       metadata(iCh).projectName = projectName;
256       metadata(iCh).sessionName = sessionName;
257       metadata(iCh).samplerateADC = samplerate;
258       metadata(iCh).samplerateDAC = samplerate;
259       metadata(iCh).comment = comment;
260       metadata(iCh).originPositionForOutput = originPositionForSource;
261       metadata(iCh).originRotationForOutput = originRotationForSource;
262       metadata(iCh).outputArray = usedOutputArrays(1);
263       % array-channel is always 1...8 % iPlayDevCh;
```

```
264         metadata(iCh).outputArrayChannel =...
265             mod(listOutChannels(iPlayDevCh) - 1, 8) + 1;
266         metadata(iCh).outputArrayPosition= outputArrayPosition;
267         metadata(iCh).outputArrayRotation= outputArrayRotation;
268         metadata(iCh).outputArrayCoordinateSystem =...
269             outputArrayCoordinateSystem;
270         metadata(iCh).roomBounds = roomBounds;
271         metadata(iCh).measurementIndex = measurementIndex;
272         metadata(iCh).ambientNoiseLevel = ambientNoiseLevel;
273         metadata(iCh).temperature = temperature;
274         metadata(iCh).T60 = T60;
275         metadata(iCh).inputArray = inputs(arrayIndexOfChannel);
276         metadata(iCh).inputArrayChannel =...
277             useOfArrayChannels(recDeviceChannel);
278         metadata(iCh).outputArray = outputs(outArrayIndex);
279         metadata(iCh).outputType = 'loudspeaker';
280         metadata(iCh).outputSignalType = 'MLS';
281         nrGuardPeriods = nrPeriods-nrUsefulPeriods;
282         tmp = attenuationMLS;
283         attenuationMLS = sprintf('%6.2f dB', attenuationMLS);
284         metadata(iCh).outputSignalDetails= bundleToStruct('orderMLS',...
285             'nrUsefulPeriods', 'nrGuardPeriods', 'nrSequences',...
286             'attenuationMLS');
287         attenuationMLS = tmp;
288
289         % metadata(iCh).outputArray = 1;
290
291         % Recording-Specific
292         metadata(iCh).x = x(:, iCh);
293         metadata(iCh).SNR_recording_dB = SNR_recording_dB(iCh);
294         metadata(iCh).P_inDigital_dB = P_inDigital_dB(iCh);
295         metadata(iCh).P_silenceDigital_dB= P_silenceDigital_dB(iCh);
296         % metadata(iCh).PNR = PNR(iCh);
297         % metadata(iCh).averageGroupDelay = averageGroupDelay(iCh);
298         metadata(iCh).wordLengthADC = wordLengthADC(iCh);
```

```
299         metadata(iCh).synchronizationOfChannels =...
300             'same AD converter / word clock';
301 
302       end
303       % Concatenate measurements
304       coherenceMeasurement = [coherenceMeasurement; metadata];
305     end
306 
307     coherenceMeasurement = orderfields(coherenceMeasurement);
308     % save(outFilePrefix, 'coherenceMeasurement');
309 
310     visualizeMSCohere(Cxy, F);
311 end
```

A.2 visualizeMSCohere.m

```
1  % function [] visualizeMSCohere(Cxy, F)
2  %
3  % Inputs:
4  %   Cxy      result from mscohere
5  %   F        frequency from mscohere
6  %
7  % Outputs:
8  %   plot
9  %
10 % Author: Nils Ballmann, Erlangen, 2013
11 %
12 
13 function [] = visualizeMSCohere(Cxy, F)
14   hFig = subfig();
15   set(hFig,'Name', 'mscohere visualisation' ,'DefaultTextInterpreter',...
16       'none');
17 
```

```
18    sizeCxy = length(Cxy);
19
20    rows = ceil(sizeCxy/2);
21    if rows > 1 || sizeCxy == 2
22      cols = 2;
23    else
24      cols = 1;
25    end
26
27    for i = 1:sizeCxy
28      plotting = subplot(rows, cols, i);
29      plot(plotting, F{i}, Cxy{i});
30      title(sprintf('channel %i', i));
31      axis([0 max(F{i}) -0.02 1.02]);
32      xlabel('Hz');
33      ylabel('magnitude squared coherence');
34    end
35  end
```

Anhang B

Automatisierte Messung

B.1 automatedMeasurement.m

```
%% hardware init trans
% trans=Nanostep('/dev/ttyUSB0');
trans=Nanostep('COM5');
trans.set_speed(5); % 5 mm/s max
trans.calibrate();
trans.rangemeasure();
%% goto initial position
trans.goto([100 100]);

%% hardware init rot
% rot=Stepper('/dev/ttyS0');
rot=Stepper('COM1');
rot.set_speed(10); % 10 is slowest
rot.calibrate();
% calibration pulls cable around pole, so turn that back
rot.goto(150);
rot.goto(-150);
rot.goto(0);
```

```
19
20 %% rot setup calibration
21 rotCalib = 112;
22 rot.goto(rotCalib);
23
24 %% rot range
25 rotStart = -90;
26 rotStop = 90;
27 rotStep = 10;
28
29 %% init trans values
30 positions = []; % table coord
31 tablePosition = [0 -0.50]; % room coord
32
33 %% init pause
34 waitingTime = 46800; % wait from 11 to 23 o'clock
35
36 %% actual measurement
37
38 % start diary
39 timestamp = datestr(datevec(now), 'yyyy-mm-dd_HH-MM-SS');
40 fname = fullfile('automatedMeasurements',...
41     strcat('diary_', timestamp, '.log'));
42 diary(fname);
43 clear 'timestamp';
44 clear 'fname';
45
46 tool = findall(0, 'Type', 'Figure');
47 handles = guidata(tool);
48 set(handles.hMeasurementIndex, 'String', '1');
49 set(handles.hMeasurementIndexInc, 'String', '1');
50 clear 'tool';
51 clear 'handles';
52
53 pause(waitingTime);
```

```
54 tic; % start stopwatch
55
56 % measurement xy-table
57 measureTablePosition(trans, tablePosition, true, true, positions);
58
59 % move back to the center
60 trans.goto([100 100]);
61 % measurement rotation
62 measureRotation(rot, tablePosition, rotCalib, rotStart, rotStop,...
63     rotStep);
64
65 elapsedTime = toc %#ok<NOPTS> % stop stopwatch, print time
66
67 % turn diary off
68 diary off;
```

B.2 measureTablePosition.m

```
1  function [ ] = measureTablePosition( trans, tablePosition,...
2      measureMainAxes, measureDiagonals, positions )
3  %MEASURETABLEPOSITION Summary of this function goes here
4  %   Detailed explanation goes here
5
6    if measureMainAxes
7      mainAxes = zeros(45, 2);
8      i = 1;
9      for x = 0:10:200
10       mainAxes(i, :) = [x 100];
11       i = i + 1;
12       if (x == 90 || x == 100)
13         xx = x + 5;
14         mainAxes(i, :) = [xx 100];
15         i = i + 1;
```

```
16        end
17      end
18
19      for y = 0:10:200
20        if y ~= 100
21          mainAxes(i, :) = [100 y];
22          i = i + 1;
23        end
24        if (y == 90 || y == 100)
25          yy = y + 5;
26          mainAxes(i, :) = [100 yy];
27          i = i + 1;
28        end
29      end
30    end
31
32    if measureDiagonals
33      diagonalAxes = zeros(44, 2);
34      i = 1;
35      for x = 0:10:200
36        if x ~= 100
37          diagonalAxes(i, :) = [x x];
38          i = i + 1;
39        end
40        if (x == 90 || x == 100)
41          xx = x + 5;
42          diagonalAxes(i, :) = [xx xx];
43          i = i + 1;
44        end
45      end
46
47      for y = 0:10:200
48        if y ~= 100
49          diagonalAxes(i, :) = [y (200-y)];
50          i = i + 1;
```

```
51        end
52        if (y == 90 || y == 100)
53          yy = y + 5;
54          diagonalAxes(i, :) = [yy (200-yy)];
55          i = i + 1;
56        end
57      end
58    end
59
60    if measureMainAxes
61      moveToPositionsAndMeasure(tablePosition, trans, mainAxes);
62    end
63    if measureDiagonals
64      moveToPositionsAndMeasure(tablePosition, trans, diagonalAxes);
65    end
66    if size(positions, 2) == 2
67      moveToPositionsAndMeasure(tablePosition, trans, positions);
68    end
69  end
```

B.3 moveToPositionsAndMeasure.m

```
1  function [ ] = moveToPositionsAndMeasure( tablePosition, trans,...
2      positions )
3  %MEASURE Summary of this function goes here
4  %   Detailed explanation goes here
5
6    tool = findall(0, 'Type', 'Figure');
7    handles = guidata(tool);
8
9    set(handles.hSourceCoord3, 'String', '0');
10
11   for line = 1:size(positions, 1)
```

```
12      x = positions(line, 1);
13      y = positions(line, 2);
14
15      % Move head into position...
16      trans.goto([x y]);
17      % ... and wait for settlement of the setup.
18      pause(5);
19
20      % Update metadata: set coordinates
21      x = ((x - 100) / 1000) + tablePosition(1);
22      y = ((y - 100) / 1000) + tablePosition(2);
23      set(handles.hSourceCoord1, 'String', x);
24      set(handles.hSourceCoord2, 'String', y);
25
26      % Measure
27      %msgbox(sprintf('x = %d y = %d', x, y));
28      try
29        MeasurementToolRIR('hMeasure_Callback', tool, [], handles)
30      catch err
31        timestamp = datestr(datevec(now), 'yyyy-mm-dd_HH-MM-SS');
32        fname = fullfile('automatedMeasurements',...
33            strcat('error_', timestamp, '.log'));
34        %open file
35        fid = fopen(fname,'a+');
36        % write the timestamp to file
37        fprintf(fid, 'Timestamp: "%s"\n', timestamp);
38        measurementIndex  = readNumericFromEdit(...
39            handles.hMeasurementIndex,  0);
40        fprintf(fid,...
41            'Position: x = "%s", y = "%s"\nMeasurement index: "%s"\n',...
42            x, y, measurementIndex);
43        % write the error to file
44        fprintf(fid, 'Error: "%s"\n', err.identifier);
45        fprintf(fid, 'Report:\n%s\n', err.getReport('extended',...
46            'hyperlinks','off'));
```

```
47
48        % close file
49        fclose(fid);
50      end
51
52      % Close visualisations (in fact, every other figure)
53      openFigures = findall(0, 'Type', 'Figure');
54      for idx = 1:numel(openFigures)
55        element = openFigures(idx);
56        if element ~= tool
57          close(element);
58        end
59      end
60
61      % Update metadata again: increment measurement index
62      measurementIndex  = readNumericFromEdit(...
63          handles.hMeasurementIndex,  0);
64      measurementIndexInc = readNumericFromEdit(...
65          handles.hMeasurementIndexInc, 0);
66      measurementIndex = measurementIndex + measurementIndexInc;
67      set(handles.hMeasurementIndex, 'String', measurementIndex);
68
69      % Short guard period waiting
70      pause(1);
71    end
72
73  end
```

B.4 measureRotation.m

```
1  function [ ] = measureRotation( rot, tablePosition, rotCalib,...
2      rotStart, rotStop, rotStep )
3  %MEASUREROTATION Summary of this function goes here
```

```
4  %   Detailed explanation goes here
5
6    tool = findall(0, 'Type', 'Figure');
7    handles = guidata(tool);
8    x = tablePosition(1);
9    y = tablePosition(2);
10   set(handles.hSourceCoord1, 'String', x);
11   set(handles.hSourceCoord2, 'String', y);
12
13   for d = rotStart:rotStep:rotStop
14     % Get current position
15     position = rot.getpos();
16     % Move head into position...
17     rot.goto(d + rotCalib);
18     % Wait for rotation to end: 4 sec / 10 degree
19     pause(abs(mod(d + rotCalib, 360)-position) * 0.4);
20     % ... and wait for settlement of the setup.
21     pause(5);
22
23     % Update metadata: set coordinates
24     set(handles.hSourceCoord3, 'String', d);
25
26     % Measure
27     %msgbox(sprintf('d = %d', d));
28     try
29       MeasurementToolRIR('hMeasure_Callback', tool, [], handles)
30     catch err
31       timestamp = datestr(datevec(now), 'yyyy-mm-dd_HH-MM-SS');
32       fname = fullfile('automatedMeasurements',...
33           strcat('error_', timestamp, '.log'));
34       %open file
35       fid = fopen(fname,'a+');
36       % write the timestamp to file
37       fprintf(fid, 'Timestamp: "%s"\n', timestamp);
38       measurementIndex  = readNumericFromEdit(...
```

```
39            handles.hMeasurementIndex, 0);
40        fprintf(fid,...
41            'Position: z(rotation) = "%s"\nMeasurement index: "%s"\n',...
42            d, measurementIndex);
43        % write the error to file
44        fprintf(fid, 'Error: "%s"\n', err.identifier);
45        fprintf(fid, 'Report:\n%s\n', err.getReport('extended',...
46            'hyperlinks','off'));
47
48        % close file
49        fclose(fid);
50      end
51
52      % Close visualisations (in fact, every other figure)
53      openFigures = findall(0, 'Type', 'Figure');
54      for idx = 1:numel(openFigures)
55        element = openFigures(idx);
56        if element ~= tool
57          close(element);
58        end
59      end
60
61      % Update metadata again: increment measurement index
62      measurementIndex = readNumericFromEdit(...
63          handles.hMeasurementIndex, 0);
64      measurementIndexInc = readNumericFromEdit(...
65          handles.hMeasurementIndexInc, 0);
66      measurementIndex = measurementIndex + measurementIndexInc;
67      set(handles.hMeasurementIndex, 'String', measurementIndex);
68
69      % Short guard period waiting
70      pause(1);
71    end
72  end
```

B.5 checkAutomatedMeasurement.m

```
function [ ] = checkAutomatedMeasurement( path )
%CHECKAUTOMATEDMEASUREMENT Summary of this function goes here
%   Detailed explanation goes here

  % get all the files in given path
  files = dir(path);

  index = 0;
  sizeDir = length(files);
  points = zeros(sizeDir, 3);

  for i = 1:sizeDir
    %get filename
    name = files(i).name;

    % filter certain files
    if strcmp(name, '.') || strcmp(name, '..') ||...
        files(i).isdir || ~strendswith(name, '.mat')
      continue;
    end

    % negative checks
    % no variable 'measurement' present
    assert(exist('measurement', 'var') == 0,...
        'no variable "measurement" present');

    % load file
    pathName = fullfile(path, name);
    load(pathName);

    % positive checks
```

```
32      % contains variable 'measurement'
33      assert(exist('measurement', 'var') == 1);
34      index = index + 1;
35      measurementIndex = measurement(1,1).measurementIndex;
36      assert(measurementIndex == index,...
37          '%s - measurementIndex %d index %d', name,...
38          measurementIndex, index);
39      points(index, :) = measurement(1,1).outputArrayPosition;
40
41      % clear for next loop
42      clear 'measurement';
43    end
44
45    points = points(1:index, :);
46
47    plot3(points(:,1), points(:,2), points(:,3), 'rx');
48    grid on;
49  end
```

B.6 strendswith.m

```
1  function b = strendswith(s, pat)
2  %STRENDSWITH Determines whether a string ends with a specified pattern
3  %
4  % b = strstartswith(s, pat);
5  % returns whether the string s ends with a sub-string pat.
6  %
7  % History % -------
8  % - Created by Dahua Lin, on Oct 9, 2008
9  %
10 %% main
11 sl = length(s);
12 pl = length(pat);
```

```
13 b = (sl >= pl && strcmp(s(sl-pl+1:sl), pat)) || isempty(pat);
```

Anhang C

Checklist „automated Measurement“

C.1 Hardware setup

C.1.1 Cabeling

- PC → Silver firewire → Hammerfall DSP Digiface

- PC → Serial cable → LMS stepmotorcontrol

- PC → USB → x-y-table stepmotorcontrol

- X-Y-table stepmotorcontrol → x serial cable (thick, yellow marking) → x-stepmotor (see drawing)

- X-Y-table stepmotorcontrol → y serial cable (thick, yellow marking) → y-stepmotor (see drawing)

- LMS stepmotorcontrol → round connector → serial cable → rotational stepmotor

- Hammerfall DSP → word clock out → green BNC cable → word clock in → RME ADI-8

- Hammerfall DSP → ADAT IN → thin optical cable → ADAT OUT → ADI-8

- Hammerfall DSP → ADAT OUT → thin optical cable → speaker array IN (on module 8)

- ADI-8 → Output (1/2) → XLR-cable → AES IN → Head BEQ-II
- ADI-8 → Input (1/2) → XLR-cable → AES OUT → Head BEQ-II

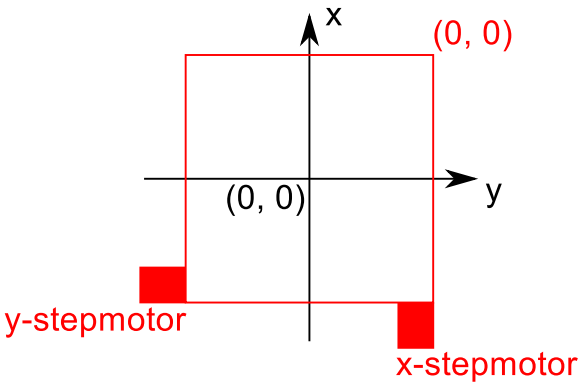

Abbildung C.1: Table setup

C.1.2 Power

C.1.2.1 Switches

- Turn on switch on black multiple socket outlet
- Turn on Head BEQ-II
- Turn on RME ADI-8
- Turn on LMS stepmotorcontrol (next to PC)

C.1.2.2 Checks

- Check speaker array powering (switch on multiple socket outlet illuminted, in the left corner)
- Check if x-y-table stepmotorcontrol has power (blue leds)

C.1.3 x-y-table

- moved x-y-table into position

C.2 Software setup

C.2.1 Preconditions

- „RIR_Measurements“-folder anywhere on C:\with write permission

C.2.2 MATLAB

- „clear all force hidden“
- „clc“
- Goto folder „RIR_Measurements“
- Add folder „myLib“ and „Steppercontrol“ and subfolders to path
- Goto folder „MeasurementToolRIR“
- Add folder „automatedMeasurements“ to path
- Mark „run_main_GUI.m“ and hit F9 to run it
- In MeasurementToolRIR GUI load config „hrtf_lrc_hats.guistate.fig“ from „RIR_Measurements“ (needs to be in this folder) → Load → Ok (ignore errors)
- Click on „Sound Devices“ and close window again with OK (sets audio interfaces up)
- Click on „Route Channels“ and close window again with OK (sets audio interfaces up)
- Open „automatedMeasurement.m“
- Run cell „hardware init trans“ (x-y-table calibration and range check needed), wait to finish
- Run cell „goto initial position“, wait to finish
- Run cell „hardware init rot“, wait to finish

C.3 Measurement

C.3.1 Preconditions

- Minimum 1.7 GB space on C:\

C.3.1.1 MeasurementToolRIR GUI

- Do not change „Output Directory“, else measurement won't work. Must be pointing to „../MeasurementToolRIR_measurements/measurements/“
- Change „Project Name“ according to needs (no points in folder name, breaks tool)
- Change „Session Name“ according to needs (no points in folder name, breaks tool)
- Change „Comment“ according to situation. Here the distance to speaker array and the center of the x-y-table is mentioned for the metadata.

C.3.1.2 automatedMeasurement.m

- Find value for „rotCalib“ in cell „rot setup calibration“ by re-running cell until head is parallel to speaker array.
- Change values of „rotStart“ and „rotStop“ in cell „rot range“ according to needs and run cell.
- Change values of „tablePosition“ in cell „init trans values“ according to situation (room coordinates, unit: meter) and run cell.
- Change value „waitingTime“ in cell „init pause“ according to needs (unit: seconds) and run cell.

C.3.2 Measurement

- Run cell „actual measurement“, wait to finish
- Lock PC (Ctrl, Alt and Delete – Enter)

C.3.3 Post processing

- Unlock PC
- Check if in the latest „automatedMeasurements/diary_***.log“ the file contains errors.
- Check if in the folder „automatedMeasurements“ there are files „error_***.log“. Those contain points in room coordinates (x, y and z for rotation) which to re-measure

on that table position, because something went wrong. This can be done with the positions-array in the cell „init trans values“ in „automatedMeasurement.m“ and in case of rotational erros with „rotStart“ and „rotStop“ in cell „rot range“.
- Copy the folder „RIR_Measurements\MeasurementToolRIR_measurements\measurements\<Project Name>\<Session Name>“ to a remote location (e.g. lnt410).
- Delete folder „RIR_Measurements\MeasurementToolRIR_measurements\measurements\<Project Name>\<Session Name>“ to free space.

C.4 Further measurements

- Move x-y-table to new position
- Repeat C.3 Measurement