Bachelorarbeit

im Studiengang
Audiovisuelle Medien

Implementierung eines Decodierungstools des MHV-Stützmikrofons für 3D-Audio-Anwendungen

vorgelegt von
Hannes Kieselbach
an der
Hochschule der Medien, Stuttgart
am 30. August 2023

zur Erlangung des akademischen Grades
Bachelor Of Engineering

Erstprüfer:
Prof. Dr. Frank Melchior
Zweitprüfer:
Prof. Oliver Curdt

Eidesstattliche Erklärung

„Hiermit versichere ich, Hannes Kieselbach, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Implementierung eines Decodierungstools des MHV-Stützmikrofons für 3D-Audio-Anwendungen“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.“
Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO, § 23 Abs. 2 Master-SPO (Vollzeit)) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Stuttgart, 28. August 2023 – Hannes Kieselbach

Zusammenfassung

Diese Arbeit erläutert die Konzeption und Implementierung eines amplitudenbasierten Decodierungstools für dreidimensionale Mischungen. Dabei werden dreikanalige Signale des MHV-Stützmikrofons in vier virtuelle Schallquellen decodiert, die anschließend in eine Mischung integriert werden können. Dem Nutzer* wird die Möglichkeit gegeben, die Eingangssignale zu manipulieren. Es wird eine Umgebung vorgestellt, die mithilfe von Testsignalen die Signalverarbeitung verifiziert. Ziel dieser Arbeit ist es ein universell nutzbares VST-Plugin zu entwickeln, das in eine 3D-Audio-Umgebung insertiert werden kann. Dabei wird der Workflow in der Programmierumgebung Rainbow der Firma Cycling'74 analysiert und bewertet. Vernachlässigt werden weitere Schritte, wie die Entwicklung eines Benutzeroberflächenkonzepts und Studien zur Usability.

Abstract

This work explains the design and implementation of an amplitude-based decoding tool for three-dimensional mixes. A three-channel signal of the MHV microphone array is decoded into four virtual sound sources, which can then be integrated into a mix. The user is given the possibility to manipulate the input signal. An environment is presented that uses test signals to verify the signal processing. The goal of this work is to develop a universally usable VST plugin that can be inserted into a 3D-audio environment. The workflow of the programming tool Rainbow by Cycling'74 is analyzed and evaluated. Unattended are further steps, such as the development of a user interface concept and studies on usability.

Inhaltsverzeichnis

Inhaltsverzeichnis I

Abbildungsverzeichnis III

Tabellenverzeichnis V

Abkürzungsverzeichnis VI

Formelzeichenverzeichnis VIII

1 Einleitung 1

2 Psychoakustik 4

2.1 Räumliches Hören 4

2.1.1 Das menschliche Ohr 5

2.1.2 Ortung von Schallereignissen 6

2.2 Summenlokalisation 9

2.2.1 Summenlokalisation für Pegelunterschiede 9

2.2.2 Summenlokalisation für Laufzeitunterschiede 11

2.3 Ausgedehnte Schallquellen 12

3 Technische Grundlagen 14

3.1 Stützmikrofone 14

3.1.1 Blumlein Pair 15

3.1.2 Natives MS-Stützmikrofon 16

3.1.3 MHV-Stützmikrofon 17

3.2 Ambisonics 19

3.2.1 Two-Dimensional First Order Ambisonics 21

3.2.3 Three-Dimensional First Order Ambisonics 22

4 Konzept 24

4.1 Parameter der Benutzeroberfläche 24

4.2 Steuersignal- und Audiosignalfluss 26

4.2.1 Überblick 26

4.2.2 Einzelne Programmabschnitte 27

4.3 Testumgebung 30

5 Implementierung 32

5.1 MAX/MSP 32

5.2 RNBO-Patch 33

5.3 Verwendete Objekte und Subpatches 35

6 Verifizierung 41

6.1 MAX-Projekt 41

6.2 Externe 3D-Audio Anwendung 43

7 Diskussion und Ausblick 47

Literaturverzeichnis 50

A Digitaler Anhang 52

Abbildungsverzeichnis

2.1	Schnitt durch das menschliche Ohr	5
2.2	Übertragungsfunktion des Außenohrs für dem Schalleinfall von vorne, oben und unten	6
2.3	Kopfbezogenes Koordinatensystem	7
2.4	Lokalisationsschärfe in der horizontalen Ebene	8
2.5	Lokalisationsschärfe in der vertikalen Ebene	9
2.6	Ein Lautsprechertripple in der horizontalen Ebene	10
2.7	Die schematische Empfindung des Korrelationsgrades *k*	12
3.1	Anordnung des Blumlein XY-Paars	16
3.2	Native MS-Mikrofonierung	17
3.3	Anordnung des MHV-Stützmikrofons	29
3.4	Sphärische Harmonische, Reihen beschreiben die n-Ordnung von Ambisonics	20
3.5	Natives Aufnahmeformat für 2D-FOA	21
3.6	Natives Aufnahmeformat für 3D-FOA	23
3.7	A-Format des 3D-FOA	23
4.1	Steuersignale und Audiosignale im Rainbow-Patch	26
4.2	Audio- und Steuersignalfluss von *editMHV*	27
4.3	Audio- & Steuersignalfluss von *decoder*	29
4.4	Audio- und Steuersignalfluss von *editFOA*	29
4.5	Audiosignalfluss der Testumgebung	31

5.1	Einbindung eines Rainbow-Patches in ein MAX-Projekt	34
5.2	Programmierung von Subpatch p *editMHV*	38
5.3	Programmierung von Subpatch p *decoder*	39
5.4	Programmierung von Subpatch p *editFOA*	39
5.5	Programmierung von Subpatch p *switchFour*	40
6.1	Eingangs- und Ausgangssignale des Rainbow-Patches	42
6.2	Programmierung von Subpatch p *offset*	42
6.3	Programmierung von Subpatch p *phaseCouples*	42
6.4	Manipulation der Testsignale	43
6.5	Skizzierung des Reaper-Projekts	45
6.6	Vergleich der Eingangs- und Ausgangssignale des Decoders (*LRBT*) bei gleicher Phasenlage der Eingangssignale	46
6.7	Vergleich der Eingangs- und Ausgangssignale des Decoders (*LRBT*) bei invertiertem horizontalen Signalanteil des Eingangssignals	46

Tabellenverzeichnis

4.1	Parameter der Benutzeroberfläche	25
5.1	Unterschiede zwischen MAX/MSP und Rainbow	34

Abkürzungsverzeichnis

Hörer*	Diese Kurzschreibweise steht stellvertretend für eine genderneutrale Bezeichnung von Versuchspersonen eines Hörversuchs	4, 6, 9, 12
Nutzer*	Diese Kurzschreibweise steht stellvertretend für eine genderneutrale Bezeichnung von Personen, die eine Software nutzen	24, 26, 47, 48, 49
DAW	Digital Audio Workstation	1, 40, 43, 44
VST	Virtual Studio Technology	1, 2, 31, 32, 33, 35, 40, 43, 44, 47
HRTF	Head Related Transfer Function	2, 5
FOA	First Order Ambisonics	19, 20, 21, 22, 23, 24, 25, 28, 44, 48
HOA	Higher Order Ambisonics	19
MS	Mid-Side	14, 16, 17, 18, 19
MHV	Mid-Horizontal-Vertical	1, 2, 17, 19, 24, 26, 27, 28, 30, 41, 44, 48, 49
LRBT	Left, Right, Bottom, Top	1, 24, 25, 44, 46, 48

ILD	Interaural Level Differences	6, 7
ITD	Interaural Time Differences	6, 7
MAX	Grafische Programmierumgebung MAX/MSP der Firma Cycling ‘74	3, 32, 33, 34, 35, 38, 41, 43, 44, 48
RNBO	Rainbow-Objekt in MAX/MSP	3, 33, 47

Formelzeichenverzeichnis

φ	Azimutwinkel im kopfbezogenen Koordinatensystem [Grad]	5, 6, 7, 8, 15, 16, 22,23, 24
δ	Elevationswinkel im kopfbezogenen Koordinatensystem [Grad]	5, 6, 7, 8, 24
f	Frequenz [Hz]	7, 8, 30
g	Gainfaktor	9, 10, 13, 15, 16, 30, 31
Δt	Laufzeitunterschied [ms]	11,
k	Korrelationsgrad	12. 13
$x[n]$	Eingangs- bez. Ausgangssignal einer digitalen Signalverarbeitung	13, 27, 28, 31, 37
a	Skalierungsfaktor der horizontalen Ausbreitung der MHV-Signalverarbeitung	16, 18, 37
b	Skalierungsfaktor der vertikalen Ausbreitung der MHV-Signalverarbeitung	18, 37
S	Virtuelle Schallquelle	21, 22

1 Einleitung

Ausgangslage dieser Bachelorarbeit sind zwei Masterarbeiten, verfasst an der Hochschule der Medien in Stuttgart. Sie dienen als Inspirationsquelle für die Themenfindung dieser Bachelorarbeit und werden in den beiden folgenden Absätzen zusammenfassend vorgestellt:

Die Masterarbeit von Leon Hofmann (2020) entwickelt und erprobt ein Stützmikrofonverfahren, welches neben einem Mittensignal auch Informationen über die horizontale und vertikale Ausbreitung der Schallquelle einfängt und somit ein dreidimensionales Abbild der Schallquelle aufzeichnet. Hofmann (2020) analysiert in dieser Arbeit die Schallabstrahlung unterschiedlicher natürlicher Schallquellen und fertigt Testaufnahmen im Freifeldraum an. Das Mikrofonierungsverfahren ist ausführlich in Abschnitt 3.1 erläutert.

Die Masterarbeit von Maurice Strobel (2021) greift dieses Mikrofonierungsverfahren auf und entwickelt ein Postproduktionstool für die DAW Ableton Live. Dabei entwickelt Strobel einen Algorithmus, der Decodierung und Panning innerhalb dieser Software ermöglicht. Diesen Decoder möchte ich in dieser Arbeit aufgreifen und als VST-Plugin entwickeln.

Meine Arbeit ist also von diesen beiden vorangegangenen Abschlussarbeiten inspiriert und greift die Thematiken auf und widmet sich der Programmierung eines VST-Plugins, das die dreikanaligen Signale des MHV-Stützmikrofons von Hofmann (2020) in zwei unterschiedliche Formate decodiert. Zum einen in das B-Format des 1st Order Ambisonics Formates und zum anderen in das von Hofmann (2020) vorgeschlagene LRBT-Format.
Dadurch möchte ich die Verwendung des MHV-Stützmikrofons in gängigen 3D-Audio-fähigen Digital Audio Workstations (DAW) ermöglichen, wie beispielsweise Nuendo oder der Alternative Reaper. Als Entwicklungsumgebung dient hierzu die grafische Programmier-

umgebung MAX/MSP der Firma Cycling ’74, welche seit dem Jahr 2022 über das Objekt Rainbow den Export als VST-Plugin mit generischer Benutzeroberfläche ermöglicht.

In Kapitel 2 werden grundlegende psychoakustische Phänomene erläutert. Zuerst wird in diesem Kapitel die Anatomie des Ohres erläutert, aus der sich die Übertragungsfunktion HRTF ableiten lässt. Darauf aufbauend werden die Gesetze des räumlichen Hörens in Bezug auf einzelne und mehrere Schallquellen vorgestellt und das Hören in vertikaler und horizontaler Ebene verglichen.

Kapitel 3 widmet sich den technischen Grundlagen. Seit geraumer Zeit gibt es unterschiedlichste Mikrofonierungsverfahren, die ein dreidimensionales Abbild einer Schallquelle einzufangen, allerdings werde ich nur eine kleine Auswahl von Mikrofonierungsverfahren vorstellen. Diese Verfahren dienen Hofmann (2020) als Grundlage zum Entwurf des entwickelten Stützmikrofons. Ein weiterer Teil dieses Abschnittes beschäftigt sich mit diesem MHV-Stützmikrofonarray von Hofmann (2020), das als Grundlage dieser Arbeit dient. Zusätzlich wird in diesem Abschnitt auf die Motivation von Ambisonics und dessen technische Funktionsweise eingegangen und besonders die erste Ordnung dieser virtuellen Szene beschrieben, die unter anderem als Zielformat des Decodierungstools dient.

Kapitel 4 befasst sich mit der mathematischen Beschreibung des Decodierungstools und der Manipulation der Eingangssignale auf Grundlage der Ausgangssignale des Mikrofonarrays von Hofmann (2020). Ziel ist es, ein Konzept für zwei unabhängig voneinander auswählbare Decodierungen zu erstellen, damit das Stützmikrofon flexibel in unterschiedlichen 3D-Umgebungen integriert werden kann. Es wird auf die Konzeption des Steuer- und Audiosignalflusses eingegangen, die anhand von Abbildungen grafisch aufbereitet sind. Im letzten Teil dieses Abschnittes gehe ich auf die Entwicklung einer Testumgebung ein, die die Signalverarbeitung verifiziert.

In Kapitel 5 wird die Programmierumgebung MAX/MSP von der Firma Cycling’74 vorgestellt, dessen besondere Arbeitsweise

beschrieben und die verwendeten Objekte aufgelistet. Anschließend wird der Workflow des Objekts RNBO erläutert, dessen besonderen Möglichkeiten vorgestellt und die Eingliederung in einem regulären MAX/MSP-Patch erklärt. Darauf aufbauend werden die unterschiedlichen Subpatches vorgestellt, die das Programm in logische Abschnitte unterteilen. Ein Subpatch ist vergleichbar mit einer Funktion in der konventionellen Programmierung.

Kapitel 6 stellt eine Verifizierungsumgebung in MAX vor und zeigt an einem praktischen Beispiel die Insertierung des Plugins in eine 3D-Audio-Umgebung am Beispiel der Software Reaper. Die zuvor in Abschnitt 4.3 beschriebenen Testsignale werden in einer Mehrkanal Audiospur aufgezeichnet und das Verhalten des Plugins anhand der aufgezeichneten Ausgangssignale überprüft.

Das letzte Kapitel beinhaltet Diskussion und Ausblick der vorangegangenen Arbeit. Hier werden Erfahrungen und Eindrücke der Programmierumgebung MAX/MSP und dem neuartigen Objekt RNBO geschildert. Es werden Möglichkeiten für folgende Entwicklungsschritte formuliert und die Arbeit eingeordnet.

2 Psychoakustik

Die Psychoakustik kombiniert physikalische Phänomene der Akustik mit der subjektiven Wahrnehmung des Menschen. Hierbei wird zwischen einem physikalischen Schallereignis und einer subjektiven Empfindung des Hörers*, dem sogenannten Hörereignis, unterschieden. Die Untersuchungen der Psychoakustik widmen sich diesen wahrgenommenen Hörereignissen. Im ersten Abschnitt beschreibe ich zuerst die Anatomie des menschlichen Ohres um anschließend die Gesetze des räumlichen Hörens zu erläutern. Der zweite Abschnitt widmet sich der Phantomschallquelle, also einer wahrgenommenen Schallquelle, welche sich zwischen mehreren physikalischen Schallquellen befindet. Im letzten Abschnitt dieses Kapitels gehe ich auf ein besonderes Phänomen von mehrkanaligen Lautsprechersystemen ein, der Empfindung einer ausgedehnten Schallquelle, sie kann nicht an einem bestimmten Ort im Raum lokalisiert werden kann und wird deshalb als umhüllend empfunden. (Weinzierl, 2008)

2.1 Räumliches Hören

Um ein Hörereignis in unserer dreidimensionalen Welt zuverlässig zu lokalisieren ist es notwendig, interaural, also mit beiden Ohren, zu hören. In der Realität helfen uns bei der Lokalisation außerdem unsere anderen Sinne, beispielsweise der Sehsinn. (Weinzierl, 2008) Diese anderen Sinne sind jedoch nicht Teil dieses Kapitels und werden vernachlässigt. Der folgende Ansatz widmet sich nun kurz der Anatomie unseres Ohres und fasst dessen Funktionsweise zusammen. Im Anschluss wird die Ortung von Schallereignissen erläutert.

2.1.1 Das menschliche Ohr

Das Außenohr fängt den Schall richtungsabhängig ein und leitet ihn durch den Ohrkanal an das Trommelfell weiter. Dort wird der Schall im Mittelohr in eine mechanische Bewegung gewandelt und an das ovale Fenster weitergeleitet. Im Innenohr befindet sich die flüssigkeitsgefüllte Hörschnecke, dessen Haarzellen nehmen den gewandelten Körperschall auf und leiteten ihn an den Hörnerv weiter. (Dickreiter et al. 2023)

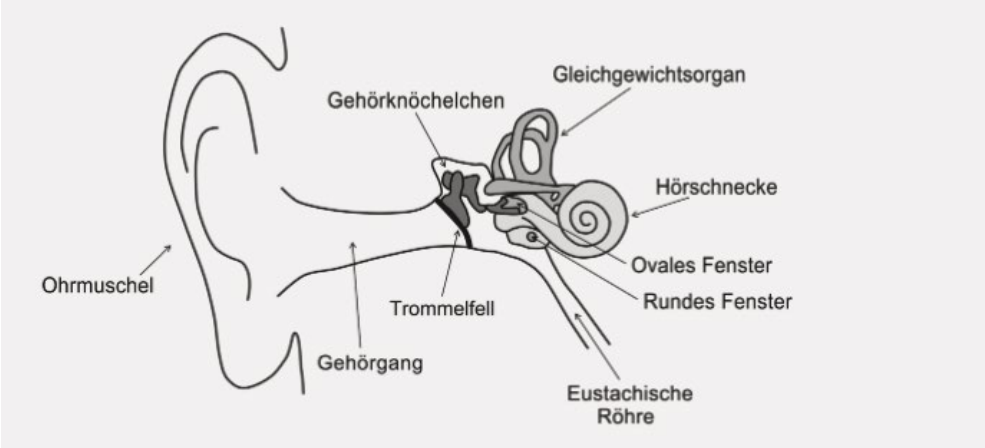


Abb. 2.1: Schnitt durch das menschliche Ohr (Grzesinski and Smyrek, 1973)

Die Form unseres Außenohrs wirkt wie ein richtungsabhängiger Filter, der uns bei der Ortung eines Schallereignisses hilft. Die sogenannte Head Related Transfer Function (HRTF), also die kopfbezogene Übertragungsfunktion, beschreibt diesen Filter in Abhängigkeit der Raumkoordinaten φ und δ. Durch Reflektionen innerhalb des Außenohrs trifft der Schall zu unterschiedlichen Zeiten am Ohrkanal an. Es bildet sich ein linear verzerrter Frequenzgang mit Kammfilterstruktur, also Anhebungen und Absenkungen in unterschiedlichen Frequenzbereichen. (Dickreiter et al. 2023) Durch unterbewusste Peilbewegungen entstehen feine Abwandlungen dieser Kammfilterstruktur und helfen unserem Gehirn dabei Schallquellen zu orten. (Mackensen, 2004)

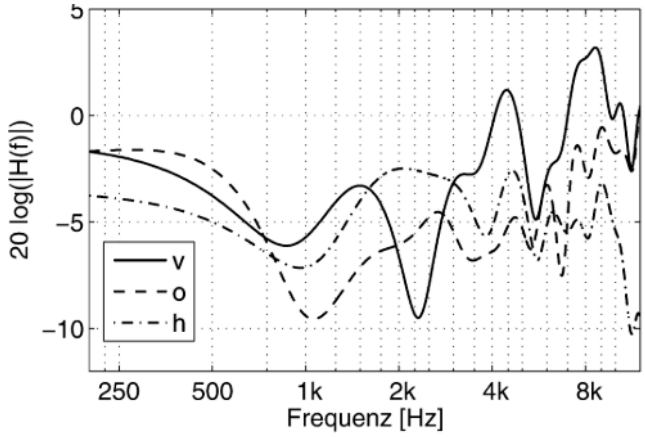


Abb. 2.2: Übertragungsfunktion des Außenohrs für dem Schalleinfall von vorne, oben und unten (Weinzierl, 2008)

2.1.2 Ortung von Schallereignissen

Um die dreidimensionale Position einer Schallquelle zu beschreiben, wird das kopfbezogene Koordinatensystem verwendet. Dessen Ursprung ist im Kopf des Hörers* in der Mitte der gedachten Achse zwischen beiden Ohrkanälen. Das Koordinatensystem wird in drei Ebenen aufgeteilt: die horizontale Ebene, die Medianebene und die Frontalebene. Durch zwei Polarkoordinaten wird die Position auf der Sphäre beschrieben: der Winkel φ beschreibt die Position auf der Horizontalebene, auch Azimut genannt und der Winkel δ die Position auf der Medianebene, auch Elevation genannt. Durch den Faktor r wird die Entfernung zum Hörer* angegeben. (Weinzierl, 2008)

Neben den spektralen Änderungen bedingt durch die Ohrmuscheln beider Ohren aus Abschnitt 2.1.1 sind für die Ortung einer Schallquelle noch die Interaural Level Difference (ILD), also die frequenzabhängigen Pegelunterschiede zwischen beiden Ohren und die Interaural Time Difference (ITD), also die zeitliche Differenz zwischen beiden Ohren, entscheidend. (Weinzierl, 2008)

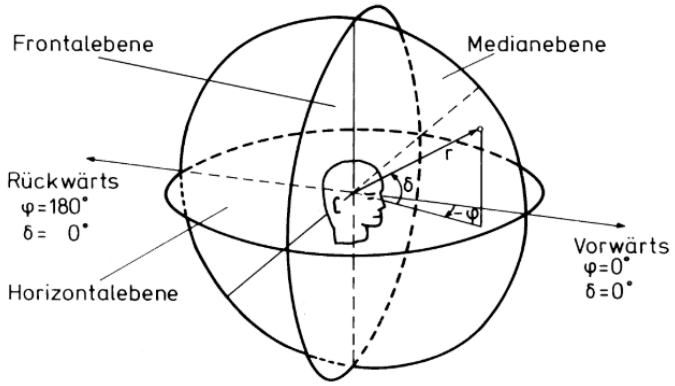


Abb. 2.3: Kopfbezogenes Koordinatensystem (Weinzierl, 2008)

In der Horizontalebene können interaurale Laufzeitdifferenzen (ITD) und frequenzabhängige Pegeldifferenzen (ILD) die Ortung beeinflussen, einzeln oder in Kombination miteinander. (Grzesinski and Smyrek, 1973) Laufzeitdifferenzen können anhand der Phasenlage und der Hüllkurve bis zu einer Frequenz $f \leq 1{,}6\ kHz$ verglichen werden, für höhere Frequenzen dient nur noch die Hüllkurve des Signals als Unterscheidungsmerkmal. Laufzeitdifferenzen können in einem Bereich von -0.7 ms und 0,7 ms wahrgenommen werden. Frequenzabhängige Pegelunterschiede hingegen können erst ab einer Frequenz von $f \geq 300\ Hz$ unterschieden werden, da sich größere Wellenlängen um Kopf und Ohrmuscheln herum beugen und somit keine Differenz verursachen. Die Ohrsignale werden bei der frequenzabhängigen Pegeldifferenz in schmalbandige Segmente zerlegt und einzeln analysieret. (Dickreiter et al. 2023)

Die Lokalisationsschärfe ist in der horizontalen Ebene am genauesten und beträgt bei einem Winkel $\varphi = 0°$ eine Auflösung von $\pm 3{,}6\,°$. Die Auflösung nimmt zu den Seiten hin ab und beträgt bei einem Winkel von $\varphi = 90°$ beziehungsweise $\varphi = 270°$ nur noch $\pm 9.2\,°$ (Pulkki and Karjalainen, 2015)

Die Medianebene verläuft genau zwischen beiden Ohren und hat deswegen nahezu identische Ohrsignalmerkmale, weswegen weder die ITD

noch die ILD zur Ortung beitragen können. (Weinzierl, 2008) Durch die Ohrmuscheln werden jedoch bestimmte Frequenzbereiche angehoben oder abgesenkt. Die Ortung ist dadurch deutlich gröber als in der horizontalen Ebene. Beispielsweise beträgt die Lokalisationsschärfe bei einem Winkel der Schallquelle von $\delta = 36°$ eine Lokalisationsschärfe von $\pm 30\,°$. Bei einem Winkel von $\delta = 90°$ können nicht einmal zuverlässig vorne oder hinten unterschieden werden. (Pulkki and Karjalainen, 2015)

Blauert (1997) unternimmt hierzu ein Experiment, bei dem er Probanden schmalbandiges Rauschen an unterschiedlichen Positionen auf der Medianebene vorspielt und diese die Position mit den Parametern vorne, hinten und oben angeben. Es stellt sich über das statistische Mittel heraus, dass nicht das eigentliche Schallereignis, sondern die spektralen Anteile des Testsignals ausschlaggebend für deren vermutete Position entscheidend sind. Frequenzen von $250\,Hz \leq f \leq 500\,Hz$ und $3\,kHz \leq f \leq 5\,kHz$ werden als richtungsrelevant für vorne empfunden während Frequenzen von $700\,Hz \leq f \leq 1{,}5\,kHz$ für die Ortung von Signalen entscheidend sind, die hinten geortet werden. Ein schmales Band um $f = 8\,kHz$ ist entscheidend für die Ortung von oben.

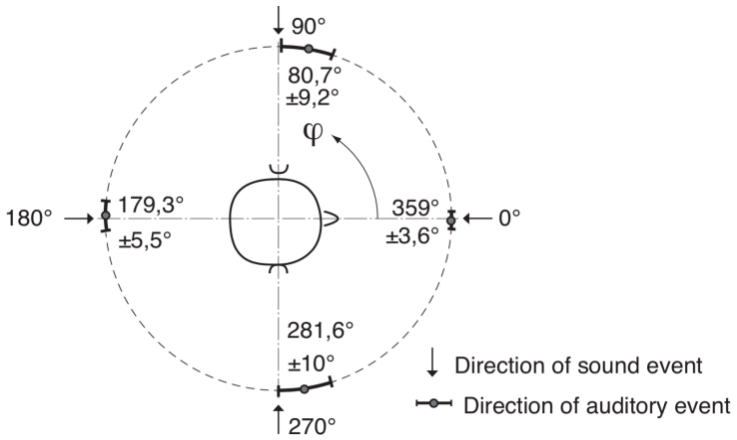


Abb. 2.4: Lokalisationsschärfe in der horizontalen Ebene (Pulkki and Karjalainen, 2015)

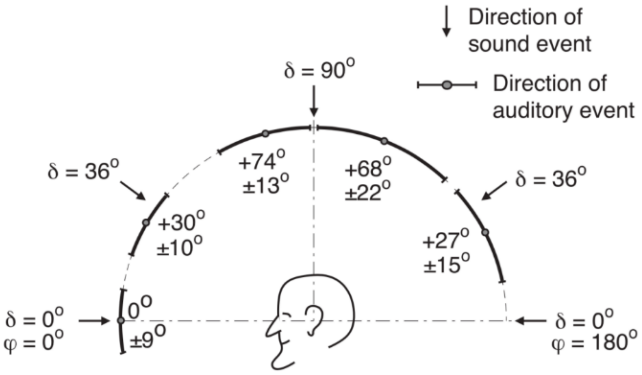


Abb. 2.5: Lokalisationsschärfe in der vertikalen Ebene (Pulkki and Karjalainen, 2015)

2.2 Summenlokalisation

Eine Phantomschallquelle beschreibt in der Psychoakustik das Empfinden eines Hörereignisses an einer Stelle im Raum, an der keine physikalische Schallquelle vorhanden ist. Ist das über die Lautsprecher abgespielte Signal identisch können Pegel- und Laufzeitdifferenzen die Phantomschallquelle in eine bestimmte Richtung zwischen den Lautsprecherachsen verschieben. Im folgenden Abschnitt wird die pegel- und laufzeitbasierende Summenlokalisation erläutert.

2.2.1 Summenlokalisation für Pegelunterschiede

Am besten lässt sich die Summenlokalisation anhand eines Stereolautsprechersetups beschreiben. Werden über den linken und den rechten Lautsprecher ein identisches Signal mit gleichem Pegel $g_{links} = g_{rechts} = 1$ und gleicher Phasenlage $\Delta t = 0\ ms$ abgespielt und sitzt der Hörer* dabei in der Mitte zwischen beiden Lautsprechern im gleichschenkligen Dreieck so entsteht eine Phantomschallquelle zwischen beiden Lautsprechern

in der Mitte der Stereobasis. Bei einem Pegelverhältnis $g_{links} = 0\,dB$ und $g_{rechts} = -15\,dB$ wird das Signal bereits vollständig von links geortet. (Dickreiter et al. 2023)

Erweitert man nun dieses Lautsprecherpaar um einen weiteren Lautsprecher in der vertikalen Ebene, erhält man ein Lautsprechertripple. Sind auch hier die Energien der Lautsprecher $g_1 = g_2 = g_3 = 1$ und die Phasenlage der Signale identisch, so wird auch hier das Schallereignis in der Mitte des Dreiecks geortet. (Pulkki, 2001)

Zu beachten sind richtungsabhängige Lokalisationsunschärfen in der horizontalen und vertikalen Ebene aus Kapitel 2.1.2.

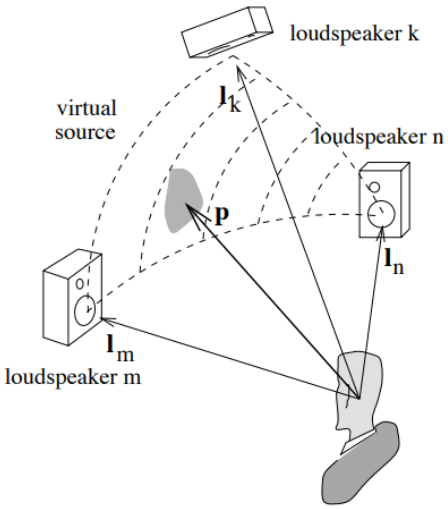


Abb. 2.6: Ein Lautsprechertripple in der horizontalen und vertikalen Ebene (Pulkki, 2001)

2.2.2 Summenlokalisation für Laufzeitunterschiede

Neben Pegelunterschieden können auch Laufzeitunterschiede zu einer Verschiebung der Phantomschallquelle führen. Hierbei wird generell unterschieden zwischen Laufzeitänderungen, die eine Phantomschallquelle verschieben, dem Haas-Effekt und der Echogrenze. Eine Richtungsdifferenzierung der Schallereignisse kann nur dann unterschieden werden, wenn zwischen den beiden Ereignissen eine Stille liegt oder der Direktschall transientenreich ist, also kurzzeitige Pegelspitzen im Signal auftreten. (Pulkki, 2001)

Laufzeitunterschiede von bis zu $\Delta t \leq 0{,}6\,ms$ führen zu einer Verschiebung der wahrgenommen Phantomschallquelle auf der horizontalen Lautsprecherachse beziehungsweise auf der Achse des Lautsprechertripples. (Dickreiter et al. 2023)

Laufzeitunterschiede zwischen $3\,ms \leq \Delta t \leq 30\,ms$ beschreiben den Präzedenzeffekt, also das Gesetz der ersten Wellenfront. Dieses Gesetz beschreibt die Ortung von Schallquellen in einem geschlossenen Raum. Treffen also Reflektionen einer schallharten Wand an unserem Ohr auf, so nehmen wir diese Reflektion nicht als zusätzliches Schallereignis war, sondern als Rauminformation, also beispielsweise Raumhall. Somit bestimmt die zuerst eintreffende Schallwelle die Richtung, während die zweite Schallwelle dem Gehirn als Rauminformation dient. (Weinzierl, 2008) Eine Sonderform des Präzedenzeffekts ist der Haas-Effekt. Er beschreibt das Phänomen, dass zwei kurz aufeinander auftreffende Schallquellen einen Pegelunterschied von bis zu $10\,dB$ haben können und die Ortung trotzdem bei der zuerst eintreffenden Wellenfront lokalisiert wird. (Grzesinski and Smyrek, 1973)

Laufzeitunterschiede von $\Delta t > 30\,ms$ werden von unserem Gehirn als zusätzliche Schallquelle wahrgenommen, also einem Echo.

2.3 Ausgedehnte Schallquellen

Bisher wurden im Abschnitt 2.2 korrelierte Signale untersucht. Das heißt, dass die Ursprungssignale, die auf die Lautsprecher gesendet wurden, identisch waren. Es wurden die Fälle der Pegelunterschiede und Laufzeitunterschiede der Signale untersucht. In diesem Abschnitt wird nun das Phänomen der ausgedehnten Schallquellen untersucht.

Es wird dann von einer ausgedehnten Schallquelle gesprochen, wenn der Hörer* das Gefühl von Einhüllung empfindet. Es handelt sich hier also um eine Art undefinierte Klangwolke. Dieses Phänomen tritt auf, wenn sich die Signale, die auf die jeweiligen Lautsprecher gesendet werden, unterscheiden. Der Grad der Unterscheidung lässt sich mathematisch als Korrelationsgrad k beschreiben. Hierbei werden beide Signale miteinander verglichen. Ist $k = 1$, so spricht man von korrelierten Signalen, somit ist das Resultat in einer Stereomischung also ein Monosignal. Ist $k = 0$, so spricht man von dekorrelierten Signalen, die sich gar nicht ähneln, das Stereobild fällt hier gänzlich auseinander und der Hörer* kann Schallquellenaus unterschiedlichen Richtungen orten. (Weinzierl, 2008) Ein Beispiel für ein dekorreliertes Signal ist ein Nachhall. Ob mit Raummikrofonen aufgenommen oder im Nachhinein über Hallgeräte zu einer Mischung hinzugefügt, verbreitert dieses dekorrelierte Signal die trockene Mischung und verbreitert das Stereobild.

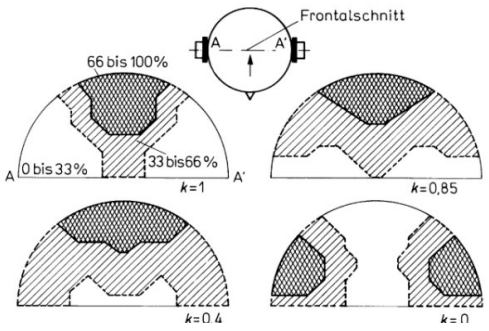


Abb. 2.7: Die schematische Empfindung des Korrelationsgrads k (Weinzierl, 2008)

Technisch lassen sich dekorrelierte Signalgruppenmit n Kanälen über eine Schaltung mit $n + 1$ Rauschgeneratoren erzeugen. Im Beispiel eines Stereopaars werden also drei Rauschgeneratoren benötigt. Es kann für dieses Beispiel somit folgende Formel aufgestellt werden:

$$L[n] = \frac{1}{2}(g_a * generator_1[n] + g_b * generator_3[n])$$

$$R[n] = \frac{1}{2}(g_a * generator_2[n] + g_b * generator_3[n])$$

Für den Fall $g_a = 1$ und $g_b = 0$ gilt, dass beide Kanäle unterschiedliche Signale enthalten und somit ein dekorreliertes Signal $k = 0$ vorliegt.

Für den Fall $g_a = 0$ und $g_b = 1$ liegen identische Signale $k = 1$ an beiden Kanälen an und somit korrelieren beide Signale. (Weinzierl, 2008)

3 Technische Grundlagen

Um das Decodierungs-Plugin zu verstehen ist es notwendig die in diesem Kapitel beschriebenen technischen Begriffe zu erläutern. Sie sind die Ausgangslage für die mathematische Konzeption und Implementierung des Decoders. Im Folgenden werde ich die Mikrofonierungsverfahren beschreiben, die Hofmann (2020) als Grundlage für das Konzept des HMV-Mikrofons dienen. Anschließend wird die virtuelle Szene Ambisonics vorgestellt und auf die 1. Ordnung dieses Standards eingegangen.

3.1 Stützmikrofone

Generell werden in der Tontechnik Haupt- und Stützmikrofone unterschieden. Ein Hauptmikrofon übernimmt bei einer Aufnahme die Aufgabe ein möglichst natürliches Klangbild und eine natürliche Repräsentation der Schallquellen abzubilden. Es ist weit von den aufzunehmenden Schallquellen entfernt, also im Raum positioniert und beinhaltet somit einen großen Raumanteil im Signal. Ein Stützmikrofon hingegen ist deutlich näher an der abzubildenden Schallquelle positioniert und beinhaltet dadurch deutlich mehr Direktschall als Reflektionen oder Übersprechen anderer Schallquellen. Es gibt zahlreiche Mikrofonarrays für Stereo-, Surround- und 3D-Abbildungem. In diesem Abschnitt gehe ich jedoch nur auf drei ausgewählte Mikrofonierungsverfahren ein, die für das Verständnis dieser Arbeit und der entwickelten Software notwendig sind.

Allgemein wird bei der Stützmikrofonierung unterschieden zwischen Intensitäts- und Laufzeitstereofonie. Die Stereomikrofonierungsverfahren XY und MS haben beide gemeinsam, dass deren Anordnung koinzident ist, sich also die Mikrofonkapseln physikalisch am selben Ort befinden. Das bietet eine verlustfreie Abwärtskompatibilität zu Mono, da es zu keinen Laufzeitverzögerungen kommt, welche in einem Downmix Kamm-filterstrukturen hervorrufen können. Das Stereobild bildet sich hier durch

Pegelunterschiede, die durch die Charakteristik der Mikrofone und deren Ausrichtung bedingt ist. Die AB-Mikrofonierung beinhaltet Laufzeitunterschiede und bietet daher keine Abwärtskompatibilität zu Mono, da hier beim Übereinanderlegen der Spuren Kammfilter entstehen können. Auf die Laufzeitstereofonie werde ich jedoch nicht weiter eingehen, da sie nicht Basis für die hier vorliegende Arbeit ist. (Dickreiter et al. 2023)

3.1.1 Blumlein Pair

Diese koinzidente Mikrofonanordnung besteht aus zwei Mikrofonen mit der Charakteristik Acht, die in einem Winkel $\varphi = \pm 45°$ zur Schallquelle ausgerichtet sind, mit $\varphi = 0°$ als Mittelachse. Der linke Kanal wird als X und der rechte Kanal als Y definiert. Durch die Änderung des Öffnungswinkels kann die Breite des aufgezeichneten Stereobildes verbreitert oder verringert werden. (Zotter and Frank, 2019) Die Empfindlichkeit der beiden Mikrofone über den Verlauf des Winkels φ lässt sich in einer Matrix beschreiben, wobei $0 \leq g \leq 1$ dem normalisierten Signalpegel des Mikrofons entspricht. (Zotter and Frank, 2019)

$$\text{Rechts: } g_{xy}(-45°) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \text{ Links: } g_{xy}(45°) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$, \text{ Mitte: } g_{xy}(-45°) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Dieses Mikrofonierungsverfahren hat den Nachteil, dass Schallereignisse, die aus Richtung $-135° \leq \varphi \leq -45°$ und $45° \leq \varphi \leq 135°$ auf das Mikrofonpaar auftreffen phasenverschoben auf dem jeweils gegenüberliegenden Kanal auftreffen, deswegen hat sich in der Praxis eine Alternative mit zwei Nieren anstatt Achten etabliert. (Zotter and Frank, 2019)

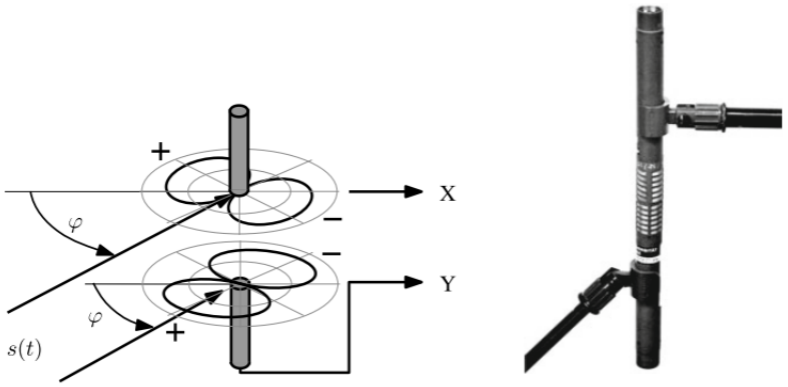


Abb. 3.1: Anordnung des Blumlein XY-Paars (Zotter and Frank, 2019)

3.1.2 Natives MS-Stützmikrofon

Die native MS-Mikrofonierung besteht aus einer Kugel und einer Acht. Die Kugel liefert das Mittensignal W und die Acht das Seitensignal Y, mit den empfindlichen Seiten nach $\varphi = 90\,° \, und \, \varphi = -90\,°$, wobei $\varphi = 0°$ als Mittelachse definiert ist. Alternative zu einer Kugel kann für das Mittensignal W auch eine Niere verwendet werden.

Auch hier lässt sich die Empfindlichkeit der beiden Mikrofone über den Verlauf des Winkels φ in einer Matrix beschreiben, wobei $0 \leq g \leq 1$ der normalisierten Amplitude des Mikrofons entspricht. (Zotter and Frank, 2019)

$$\text{Links: } g_{xy}(90°) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \text{ Rechts: } g_{xy}(-90°) = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

$$\text{Mitte: } g_{xy}(0°) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Durch die Matrizierung der Signale W und Y wird das Stereosignalpaar L und R gewonnen, das nun auf die Lautsprecher geroutet werden kann. Diese Decodierung ist verlustfrei und kann in beide Richtungen beliebig oft wiederholt werden.

Fügt man dem Seitensignal Y einen Faktor a, mit $0 \leq a \leq 1$ hinzu, so lässt sich die Breite des matrizierten Stereobildes im Nachhinein verändern, wobei $a = 1$ die maximale Breite definiert, hingeben $a = 0$ das Seitensignal aus der Formel nimmt und somit nur noch das Mittensignal W übrig bleibt. Somit wird auf den beiden Kanälen L und R dasselbe Signal übertragen, das Signal ist also mono.

$$L = \frac{W + (a * Y)}{2}$$

$$R = \frac{W + ((a * (-Y))}{2}$$

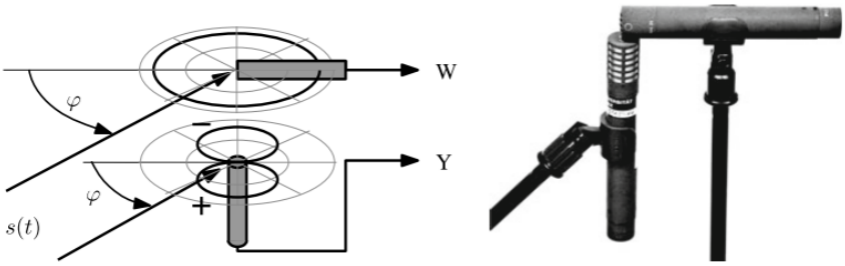


Abb. 3.2: Native MS-Mikrofonierung (Zotter and Frank, 2019)

3.1.3 MHV-Stützmikrofon

Wie bereits in Kapitel 1 erwähnt, greife ich in dieser Arbeit das MHV-Stützmikrofonierungsverfahren von Hofmann (2020) auf. Dieses beruht auf dem Prinzip der MS-Stereofonie (Kapitel 3.1.2) und erweitert dieses Mikrofonarray um eine weitere Achse, die das vertikale Abstahlverhalten der Schallquelle aufzeichnet. Das Array besteht aus einem Mikrofon mit der Charakteristik Niere, welche die Mitteninformationen der Schallquelle aufzeichnet und aus zwei orthogonal zueinander angeordneten Mikrofonen mit der Charakteristik Acht, welche das horizontale und vertikale

Abstrahlverhalten der Schallquelle einfangen. Da es sich hierbei um ein koinzidentes Mikrofonierungsverfahren ohne Lautzeitunterschiede handelt ist es notwendig, dass die Kapseln der einzelnen Mikrofone möglichst am selben Ort den Schall aufzeichnen. Hierzu schlägt Hofmann (2020) folgende Anordnung vor: Zunächst wird, wie bei der MS-Anordnung, die Niere und die horizontale Acht, beispielsweise über eine geeignete MS-Klemme, bündig positioniert und anschließend die vertikale Acht vor der unempfindlichen Seite der horizontalen Acht angebracht.

Durch Matrizierung der einzelnen Mikrofonkanäle und deren Mischverhältnis zueinander kann in der Postproduktion eine Änderung in der abgebildeten Breite beziehungsweise der Höhe in der Mischung realisiert werden. Bei der Matrizierung entstehen aus drei Mikrofonsignalen die vier Monokanäle (*L, R, B, T*).

$$L = M + (H * a) \qquad B = M + (V * b)$$
$$R = M + ((-H) * a) \qquad T = M + ((-V) * b)$$

Die Parameter a und b dienen als Faktoren um die horizontale und vertikale Breite des Signals zu manipulieren (Vergleich Abschnitt 3.1.2).
Beim Positionieren dieses Stützmikrofons ist es wünschenswert, dekorrelierte Signale zu erhalten (Kapitel 2.3). Das bedeutet im Falle eines Stützmikrofons, dass das gesamte Array möglichst nahe an der Schallquelle positioniert werden muss, da so die unterschiedlichen Klangfarben und frequenz- und pegelabhängigen Abstrahlungen der Schallquelle optimal eingefangen werden können. Allerdings kann es bei sehr naher Mikrofonierung zu in der Regel nicht erwünschten Klangverfärbungen kommen, auch als Nahbesprechungseffekt bekannt. Deswegen ist es notwendig den für sich geeigneten individuellen Abstand für das Stützmikrofon zu finden, abhängig von der Art der Schallquelle und der Färbung durch den Nahbesprechungseffekt. (Hofmann, 2020)

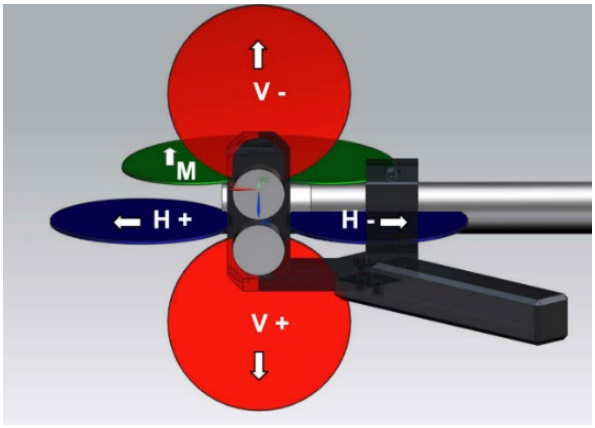


Abb. 3.3: Anordnung des MHV-Stützmikrofons (Hofmann, 2020)

3.2 Ambisonics

Um frei von dezidierten Lautsprechersetups eine zwei- oder dreidimensionale Szene aufnehmen zu können, zu speichern und zu distribuieren wurde der Ambisonics-Standard entwickelt. Er beschreibt Ort und Intensität der Schallquellen anhand von virtuellen Mikrofonen, die gemeinsam die Ortsbeschreibung eines Schallereignisses in einer Sphäre beschreiben. Der mathematische Hintergrund von First Order Ambisonics (FOA) ist die Matrizierung der MS-Mikrofonierung, wie in Abschnitt 3.1.2 beschrieben. Ein genereller Vorteil von Ambisonics ist die verlustfreie Abwärtskompatibilität, beispielsweise zu Stereo (Zotter and Frank, 2019).

Da die Ortsauflösung von FOA auf die Achsen *links-rechts, oben-unten* und *vorne-hinten* begrenzt ist, wurde als Nachfolger Higher Order Ambisonics (HOA) entwickelt. Das Prinzip beruht hier auf den sphärischen Harmonischen der Kugelflächenfunktion (Wittek, Haut and Keinath, 2006) . Je nach Ordnung wird die Anzahl von virtuellen Mikrofonen vergrößert und damit eine höhere Ortsauflösung erreicht. Ziel ist es, differenziertere räumliche Abbildungen zu erreichen. Vorteil von höheren Ordnungen ist, dass Signale mit vielen unterschiedlichen

Richtungsinformationen dargestellt und abgespeichert werden können. Somit kann eine ausgedehnte Schallquelle dargestellt werden, das kann bei einer Raumsimulation zu dem Gefühl von Einhüllung entscheidend beitragen, da viele dekorrelierte Signale aus vielen unterschiedlichen Richtungen dargestellt werden können. (Zotter and Frank, 2019)

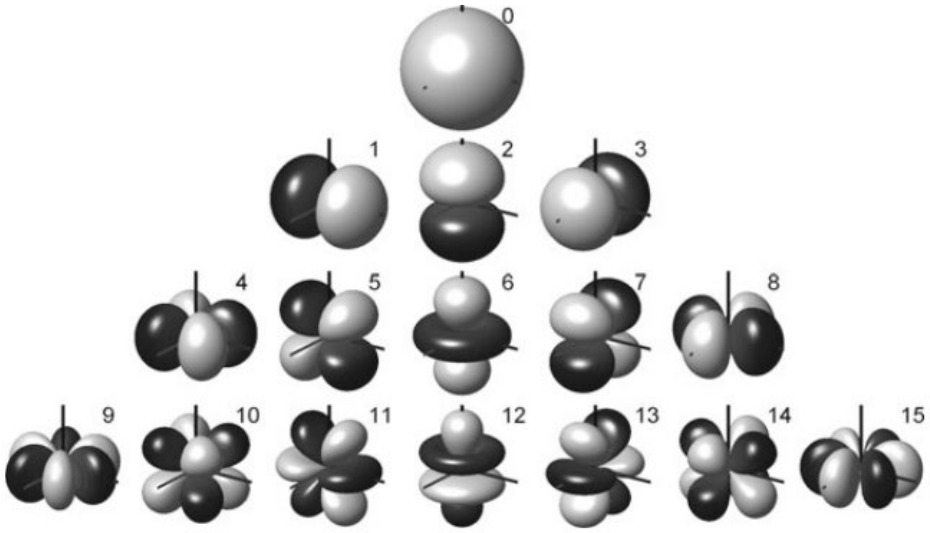


Abb. 3.4: Sphärische Harmonische, Reihen beschreiben die n- Ordnung von Ambisonics (Zotter and Frank, 2019)

Allgemein werden bei Ambisonics verschiedene Formate und Kanalanordnungen unterschieden. Im folgenden Absatz werden drei dieser Formate kurz erläutert.

Das A-Format bezeichnet das Aufnahmeformat. Dieses Format muss zur Weiterverarbeitung in das B-Format codiert werden, dieses dient als Postproduktionsformat und ist am weitesten verbreitet. Das B-Format stellt die oben beschrieben sphärischen Harmonischen in einzelnen Monokanälen dar. (Sontacchi *et al.*, 2011) Um 2D-FOA direkt ohne Decoder abspielen zu können wurde das G-Format entwickelt, welches die Signale auf typische und verbreitete Lautsprecheranordnungen anpasst. (Elen and Carbines, 2006)

3.2.1 Two-Dimensional First Order Ambisonics

Dieses zweidimensionale Format besteht aus den Kanälen (W, Y, X), wobei W das Mittensignal ist. Die beiden Seitensignale werden mit je einem Mikrofon der Charakteristik Acht abgebildet. Y steht für die *links-rechts* Achse und X steht für die *vorne-hinten* Achse. Es wird auch als natives B-Format bezeichnet, da die Kanalstruktur den sphärischen Harmonischen entspricht.

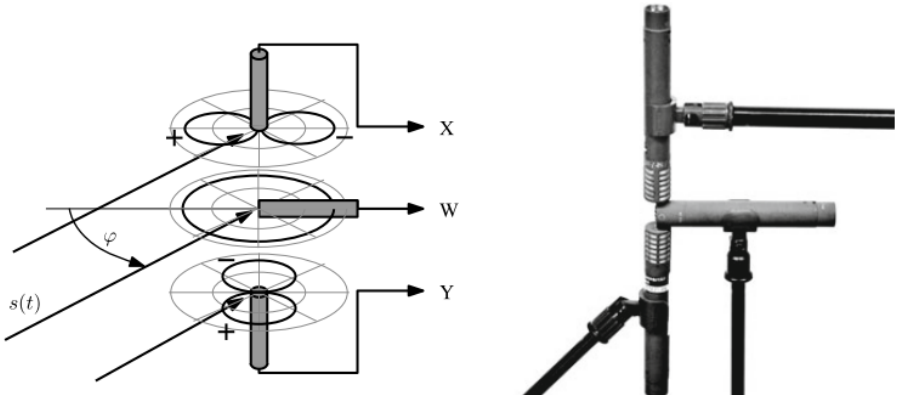


Abb. 3.5: Natives Aufnahmeformat für 2D-FOA (Zotter and Frank, 2019)

Nach demselben Prinzip, wie in Abschnitt 3.1.2 beschrieben, werden auch hier Signale miteinander mariziert, um die virtuellen Schallquellen Links, Rechts, Vorne und Hinten zu erhalten. (Zotter and Frank, 2019)

$$S_{links} = W + Y$$
$$S_{rechts} = W + (-Y)$$
$$S_{vorne} = W + X$$
$$S_{hinten} = W + (-X)$$

Neben dem nativen B-Format Aufnahmeverfahren gibt es mehrere Möglichkeiten im A-Format aufzunehmen und anschließend in das eben erläuterte B-Format zu konvertieren. Davon werde ich nun eines dieser Mikrofonierungsverfahren genauer beleuchten.

Dieses Mikrofonierungsverfahren besteht aus drei Mikrofonen mit der Charakteristik Niere, welche in den Winkeln $\varphi = 120°$, $\varphi = -120°$ und $\varphi = 0°$ angeordnet werden. Entsprechend zu den Ausrichtungswinkeln der Mikrofone werden die Kanäle des A-Formats C_{120}, C_{-120} und C_0 benannt. Durch die folgende Matrizierung dieser Kanäle kann auch hier das B-Format (W, Y, X) gewonnen werden. (Zotter and Frank, 2019)

$$W = C_{120} + C_{-120} + C_0$$
$$Y = \left(-\sqrt{3} * C_{-120}\right) - \left(\sqrt{3 * C_{120}}\right)$$
$$X = (2 * C_0) - (-C_{-120}) - (-C_{120})$$

3.2.3 Three-Dimensional First Order Ambisonics

Um das oben beschriebe 2D-FOA-Format um eine Höheninformation zu erweitern wird ein weiterer Kanal hinzugefügt. Dieses dreidimensionale Format besteht aus den Kanälen (W, Y, Z, X). Dem 2D-FOA-B-Format wird also der Kanal Z hinzugefügt, der die vertikale Ausdehnung einer Schallquelle darstellt.

Um die virtuellen Schallquellen des 2D First Order Ambisonics um die Kanäle *Oben* und *Unten* zu erweitern, wird auf Basis der Matrizierung aus Abschnitt 3.2.1 folgende Formel aufgestellt.
(Zotter and Frank, 2019)

$$S_{links} = W + (Y);$$
$$S_{rechts} = W + (-Y)$$
$$S_{vorne} = W + X$$
$$S_{hinten} = W + (-X)$$
$$S_{oben} = W + Z$$
$$S_{unten} = W + (-Z)$$

Das native Aufnahmeformat des 3D-FOA beinhaltet also alle sphärischen Harmonischen der ersten Ordnung der Kugelfunktion (Abschnitt 3.2). (Wittek, Haut and Keinath, 2006)

$$W = 1$$
$$X = \cos(\theta) * \sin(\varphi)$$
$$Y = \sin(\theta) * \cos(\varphi)$$
$$Z = \sin(\varphi)$$

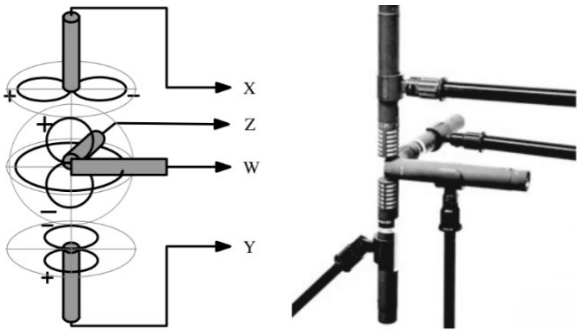
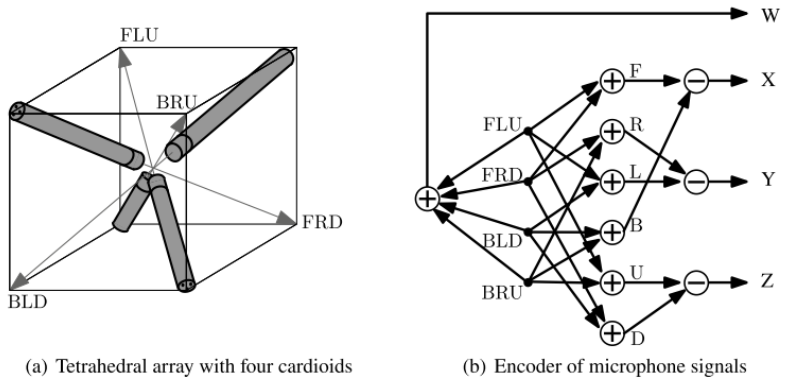


Abb. 3.6: Natives Aufnahmeformat für 3D-FOA (Zotter and Frank, 2019)

Ein mögliches A-Format des 3D-FOA ist die Anordnung von vier Nierenmikrofonen in einer tetraederförmigen Form. Die Mikrofonkanäle *FLU, FRD, BLD* und *BRU* werden anschließend in das B-Format (*W,X,Y,Z*) codiert.



(a) Tetrahedral array with four cardioids (b) Encoder of microphone signals

Abb. 3.7: A-Format des 3D-FOA (Zotter and Frank, 2019)

4 Konzept

Diese Arbeit entwickelt einen Decoder, der die dreikanaligen Signale eines MHV-Stützmikrofons decodiert, um sie in 3D-Audio-Anwendungen als Plugin zu insertieren. Hierbei hat der Nutzer* die Auswahl zwischen der Decodierung in das von Hofmann (2020) vorgeschlagene LRBT-Format oder in FOA (siehe Abschnitt 3.1.3). Zusätzlich wird dem Nutzer* die Möglichkeit gegeben, das Eingangssignal zu manipulieren. In diesem Abschnitt werden neben entwickelten Parametern der Benutzeroberfläche auch das Design von Steuer- und Audiosignaldatenfluss vorgestellt. Im letzten Abschnitt dieses Kapitels wird eine Testumgebung entworfen, mit der der Signalfluss überprüft werden kann.

4.1 Parameter der Benutzeroberfläche

Die verwendete Entwicklungsumgebung (siehe Kapitel 5) lässt es nicht zu, eine individuelle grafische Benutzeroberfläche zu entwerfen. Es wird eine generische Oberfläche erzeugt, welche lediglich aus Schiebereglern besteht. Das beschränkt die Entwicklung des Designs einer Nutzeroberfläche auf Zahlenwerte, die vom Nutzer* verändert werden können. Da es in dieser Art von Nutzeroberfläche keine Schalter und Auswahlknöpfe gibt werden diese über Regler mit einem ganzzahligen Wert von 0 oder 1 dargestellt, was in der Programmierung auch als `boolean` bekannt ist. Dem Nutzer* wird ermöglicht vor der Decodierung das Eingangssignal des MHV-Stützmikrofons zu manipulieren. Hierbei kann die vertikale und horizontale Ausbreitung des Signals verändert werden und die Polarität der Achsen H und V vertauscht werden, das entspricht einer Änderung der Winkel $\varphi = 0° \rightarrow \varphi = 180°$ beziehungsweise $\delta = 0° \rightarrow \delta = 180°$ (siehe Abschnitt 2.1.2). Außerdem ist eine Schaltfläche für die Wahl des Decoders vorgesehen. Für den Fall, dass FOA als Zielformat gewählt ist gibt es zusätzlich die Möglichkeit, die vertikale oder horizontale Achse auf die X-Achse des FOA-Formats zu verschieben.

Name	Datentyp	Beschreibung
H-Width	`float [1;0]`	Steuert die horizontale Ausdehnung des Signals
H-Polarity	`boolean {0,1}`	Vertauscht die Ortsinformationen *Links-Rechts*
V-Width	`float [1;0]`	Steuert die vertikale Ausdehnung des Signals
V-Polarity	`boolean {0;1}`	Vertauscht die Ortsinformationen *Links-Rechts*
HV-Switch	`boolean {0;1}`	Vertauscht die horizontalen und vertikalen Ortsinformationen
Decoder	`boolean {0;1}`	Wählt zwischen den beiden Decodierungen *(LRBT)* und FOA
YtoX	`boolean {0;1}`	Nur für den Fall FOA: Sendet die horizontalen Ortsinformationen an die X-Achse des FOA-Formats
ZtoX	`boolean {0;1}`	Nur für den Fall FOA: Sendet die vertikalen Ortsinformationen an die X-Achse des FOA-Formats

Tabelle 4.1: Parameter der Benutzeroberfläche

4.2 Steuersignal- und Audiosignalfluss

Dieser Abschnitt teilt das Programm in einzelne logische Einheiten auf, welche Audio- und Steuersignale untereinander austauschen können. Zuerst wird ein Entwurf auf Top-Level-Niveau vorgestellt.

4.2.1 Überblick

Das Plugin nimmt vom Host ein dreikanaliges Audiosignal des MHV-Mikrofonarrays mit den Kanälen (*M, H, V*) entgegen und zusätzlich sechs Steuersignale (siehe Abschnitt 4.1). Abhängig von diesen Steuersignalen wird ein vierkanaliges Audiosignal mit den Kanälen *(L, R, B, T)* oder *(W, Y, Z, X)* an den Host zurückgegeben. Diese Steuersignale können von dem Nutzer* in der Benutzeroberfläche innerhalb eines festgelegten Wertebereichs eingestellt werden.

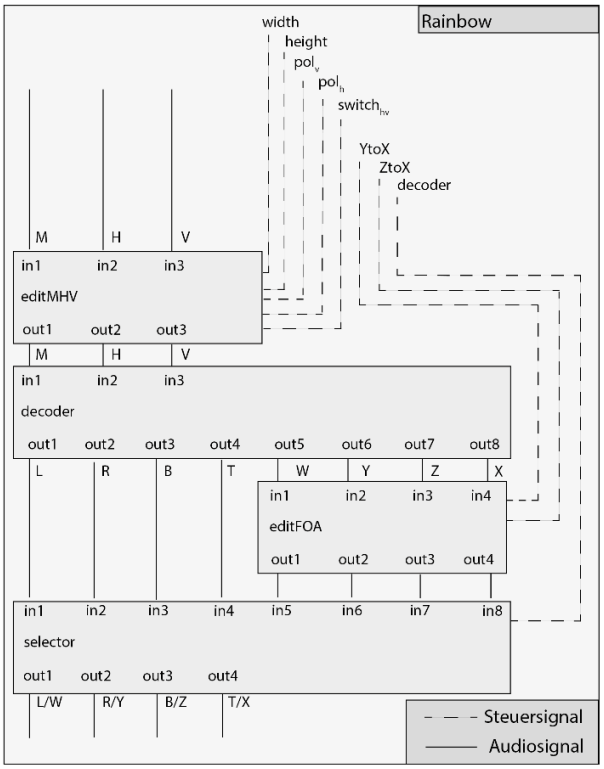


Abb. 4.1: Steuersignale und Audiosignale im Rainbow-Patch

4.2.2 Einzelne Programmabschnitte

Der erste Abschnitt *editMHV* manipuliert das Eingangssignal *(M,H,V)*. Hierzu werden die Steuersignale *width, height,* pol_h , pol_v und $switch_{hv}$ entgegengenommen und mit den Audiosignalen wie folgt verrechnet:

$$H[n] = H[n] * width * pol_h$$
$$V[n] = H[n] * height * pol_v$$
$$\text{mit } width \in \{0;1\};\ pol_h \in [0;-1];\ height \in \{0;1\};\ pol_v \in [0;-1]$$

Der Parameter $switch_{HV}$ steuert einen Switch, welcher die horizontalen und vertikalen Informationen des Signals miteinander tauscht. Die Parameter *width* und *height* entsprechen den Gainfaktoren *a* und *b* aus Abschnitt 3.1.3.

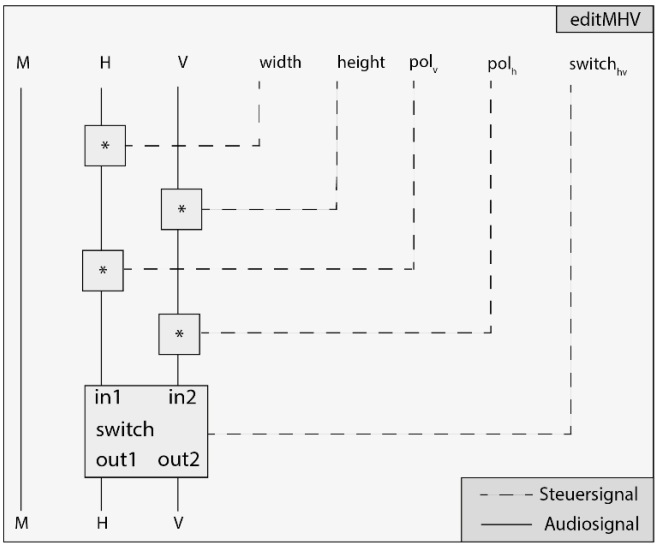


Abb. 4.2: Audio- und Steuersignalfluss von *editMHV*

Der Abschnitt *decoder* (siehe Abbildung 4.X) widmet sich der Umwandlung der manipulierten MHV-Signale. Es werden die Kanäle *(M,H,V)* entgegengenommen und die Signalpaare *(L,R,B,T)* und *(W,Y,Z,X)* ausgegeben. Hierzu wird zunächst das Multikanalsignal in zwei Stränge aufgeteilt und verrechnet. Im Folgenden wird die Matrizierung des ersten Strangs in das Zielformat *(L,R,B,T)* beschrieben:

$$L[n] = \frac{1}{2}(M[n] * H[n])$$
$$R[n] = \frac{1}{2}(M[n] * (-1 * H[n]))$$
$$B[n] = \frac{1}{2}(M[n] * V[n])$$
$$T[n] = \frac{1}{2}(M[n] * (-1 * V[n]))$$

Der zweite Strang konvertiert das MHV-Signal in das FOA. Um das Format FOA zu erreichen ist lediglich nötig, den vertikalen Kanal zu invertieren. Eine weitere Matrizierung ist nicht nötig, da das MHV-Stützmikrofon bereits die Kanäle *W*, *Y* und *Z* nativ aufzeichnet, lediglich die Informationen des Kanals *X* werden bei diesem Mikrofonierungsverfahren nicht aufgezeichnet. (siehe Abschnitt 3.2.3)

$$W[n] = M[n]$$
$$Y[n] = H[n]$$
$$Z[n] = (-1) * V[n]$$
$$X[n] = null$$

Um diesen Kanal mit Informationen zu füllen, gibt es nun zwei Möglichkeiten. Zum einen kann auf diesen Kanal eine Matrizierung der Signale *Y* und *X* gegeben werden, dadurch werden allerdings keine zusätzlichen Informationen generiert, da zur Matrizierung nur bereits verwendete vertikale und horizontale Signalanteile dienen. Dieser Ansatz wird daher nicht weiter verfolgt. Die zweite Möglichkeit besteht darin, eine Umsortierung der Kanäle vorzunehmen. Dies geschieht im Programmabschnitt *editFOA*.

Dieses Objekt hat vier Audioeingänge und die zwei Steuersignaleingänge *YtoX* und *ZtoX*. Diese Signale steuern jeweils einen Switch, der die Signale *Y* beziehungsweise *Z* auf den Kanal *X* routen. Über die Änderung der Parameter pol_h beziehungsweise pol_v des Programmabschnitts *editMHV* kann nun auf dem neuen Kanal *X* die Umpolung vorne-hinten gesteuert werden.

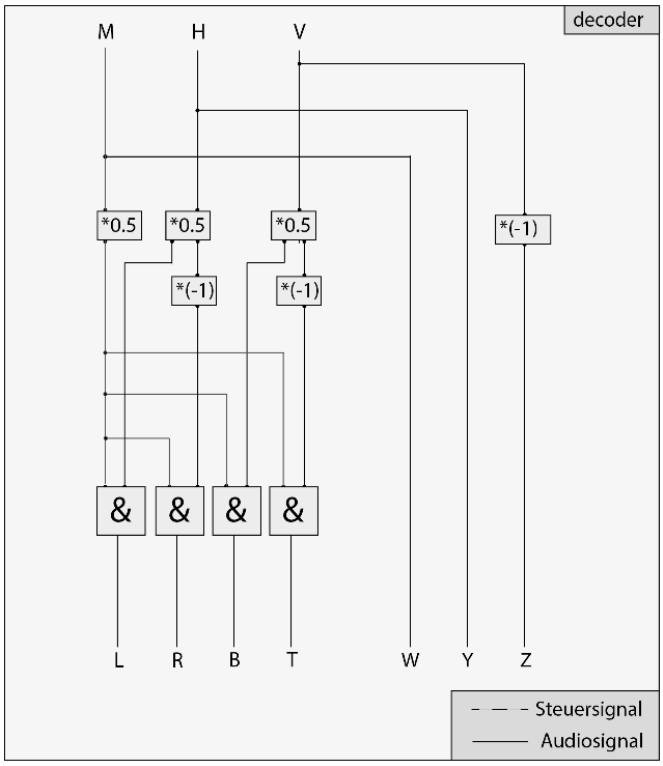


Abb. 4.3: Audio- und Steuersignalfluss von *decoder*

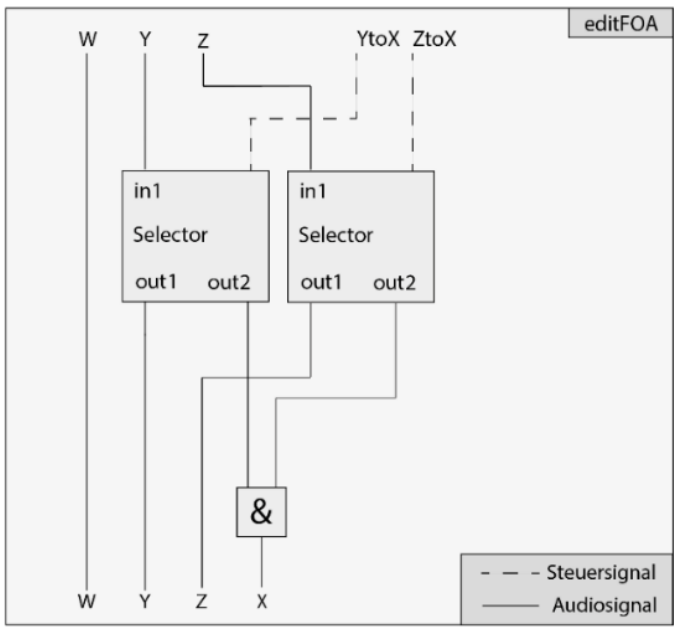


Abb. 4.4 Audio- und Steuersignalfluss von *editFOA*

4.3 Testumgebung

Um die Signalverarbeitungskette des Plugins zu überprüfen, wird eine Testumgebung konzipiert. Sie wird als interaktive Umgebung konzipiert und beinhaltet das zu überprüfende Objekt (siehe Abbildung 4.5). Alle Testsignale bestehen aus einer dreikanaligen Audiodatei. Folgend werden zwei Testsignale vorgestellt.

Das Signal *sinewave* besteht aus drei phasengleichen Sinustönen mit der Frequenz $f = 440\ Hz$. Es wird über einen einzigen Sinusgenerator erzeugt, der auf die drei Kanäle geroutet ist.

Das Signal *randomNoise* beinhaltet Zufallswerte zwischen $-1 \leq n \leq 1$, sogenanntes Rauschen. Hierbei werden drei voneinander unabhängige Rauschgeneratoren auf die drei Kanäle der Datei geroutet.

Nach der Auswahl des bevorzugten Testsignals können die Einzelkanäle des Signals manipuliert werden. Im ersten Schritt kann ein Offset auf das Testsignal errechnet werden, bei dem die Werte des Signals in den positiven Bereich $0 \leq n \leq 1$ verschoben werden.

Im zweiten Schritt werden die Werte der H- und V-Signale aufgeteilt und invertiert. Diese Signale laufen anschließend einen Mischer mit den Ausgängen (M, H, V). Im Mischer werden zuerst die jeweils positiven und negativen H- und V-Anteile miteinander gemischt und im Anschluss die Seiten- und Höheninformation mit dem Mittensignal M in ein Verhältnis gebracht (siehe Abbildung 4.5).

Das geschieht über den jeweiligen Faktor g, mit $0 \leq g \leq 1$. Ist also beispielsweise $g_{21} = 1$ und $g_{22} = 0$ so wird die horizontale Information des Audiosignals vom Decoder als links interpretiert.

Das resultierende Testsignal simuliert also das Signal des MHV-Mikrofonarrays und wird als Eingangssignal für das zu überprüfende Objekt genutzt.

$$M[n] = g_1 * gen[n]$$
$$H_{positive}[n] = g_{21} * gen[n]$$
$$H_{negative}[n] = (-1) * g_{22} * gen[n]$$
$$V_{positive}[n] = g_{31} * gen[x]$$
$$V_{negative}[n] = (-1) * g_{32} * gen[n]$$

Die vier Ausgangssignale des Objekts werden in Abhängigkeit des ausgewählten Decoders berechnet (siehe Abschnitt 4.1). Die Signale (*L, R, B, T*) und *(W, Y, Z, X)* können im Oszilloskop betrachtet und interpretiert werden. Des Weiteren laufen die Ausgangssignale in ein Objekt, das ein VST-Plugin hosten kann. Das ist als Alternative für die grafische Darstellung der Ausgangssignale gedacht. Hierzu ist im Projektordner ein Vorschlag hinterlegt. Es handelt sich hierbei um das VST-Plugin Energy Visualizer (*IEM Plug-in Suite*, 2023) des Instituts für Elektronische Musik und Akustik aus Graz. In Kapitel 6 werden die Ein- und Ausgangssignale des Rainbow-Objekts beziehungsweise des kompilierten VST-Plugins verglichen.

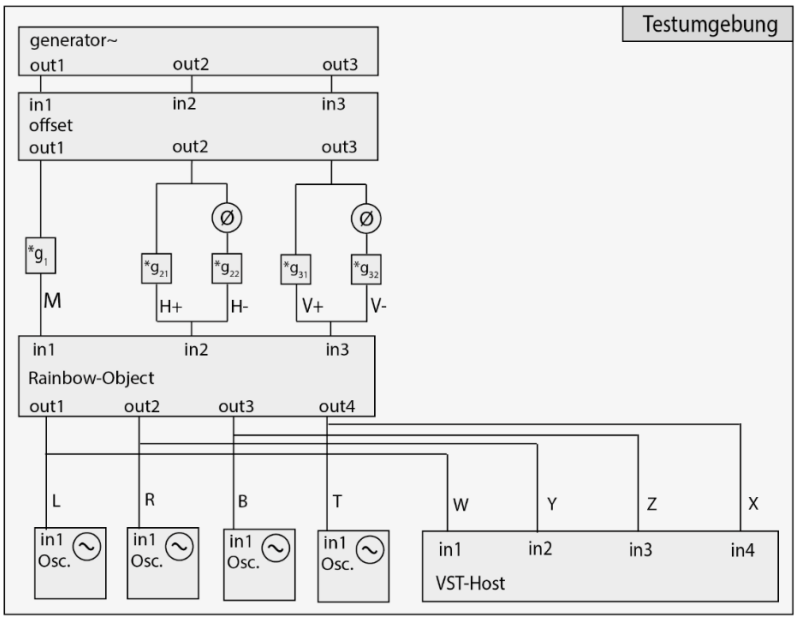


Abb. 4.5: Audiosignalfluss der Testumgebung

5 Implementierung

Dieses Kapitel widmet sich der Entwicklungsumgebung, in der das VST-Plugin programmiert wurde. Im Anschluss wird der Aufbau Projekts anhand der Subpatches beschrieben. Sie gliedern das Programm in logische Abschnitte. Die verwendeten Objekte werden vorgestellt.

5.1 MAX/MSP

Die Entscheidung der Entwicklungsumgebung für das VST-Plugin ist auf die grafische Programmierumgebung MAX/MSP der Firma Cycling ‘74 gefallen. Sie ist speziell auf Bedürfnisse der Audio- und Videosignalverarbeitung angepasst und unterscheidet sich in der Nutzung grundlegend von codebasierten Programmiersprachen, wie Java oder Python. MAX verwendet einen leeren Schreibtisch, in dem sogenannte Objekte mit Patchkabeln verbunden werden, die Audio- und Steuersignale übertragen können. Somit ähnelt der Workflow eher einer Entwicklung eines elektronischen Schaltplans als an eine klassische Entwicklungsumgebung. Sie wird von Ingenieuren genauso wie von Musikern und Medienkünstlern verwendet, um individuelle Signalverarbeitungsketten und Instrumente unabhängig von Programm-erweiterungen von Drittanbietern zu realisieren. Es ist möglich einen Audiodatenstrom direkt über ein Audiointerface abzugreifen oder zu senden oder über Generatoren zu erzeugen. MAX wird sowohl zur Entwicklung des VST-Plugins als auch zur Erstellung einer Testumgebung zum Verifizieren des Algorithmus verwendet.(*What is Max? | Cycling ’74*, 2023)

5.2 RNBO-Patch

MAX/MSP ist eine eigenständige Programmiersprache und kann erst einmal nicht in gängigen Code übersetzt werden. Seit dem Jahr 2022 ist es jedoch möglich über das Rainbow-Objekt (`rnbo~`) den generierten Patch in verschiedene Zielformate zu exportieren. Dazu zählt der Export als Audio Unit (AU) oder VST-Plugin für die Plattformen Microsoft Windows, Apple OSX und GNU/Linux. Des Weiteren kann Quellcode für den Microcontroller Raspberry Pi, JavaScript und C++ erzeugt werden. (CDM Create Digital Music, 2023)

Um einen Rainbow-Patch zu erstellen, wird innerhalb eines MAX-Patches das Objekt `rnbo~` erzeugt. Dieses Objekt verhält sich wie ein Subpatch. Das Objekt wird also in einen MAX-Patch integriert. Die innerhalb des Rainbow-Objekts erzeugte Signalverarbeitung kann dann in die oben erläuterten Zielformate exportiert werden. Damit ein erzeugter Patch exportiert werden kann werden die lokalen Daten an einen Server von Cycling ’74 hochgeladen. In der Cloud wird der Quellcode für das entsprechende Format kompiliert und anschließend auf den lokalen Rechner wieder heruntergeladen.

Es gibt einige Unterschiede zwischen einem herkömmlichen MAX/MSP-Patch und einem Rainbow-Patch, die in folgender Tabelle stichwortartig gegenübergestellt werden. (*Key Differences | Cycling ’74*, 2023)

MAX/MSP-Patch	Rainbow-Patch
Ganzzahlen **`integer`** & Fließkommazahlen **`float`**	~~Ganzzahlen **`integer`** &~~ Fließkommazahlen **`float`**
Zeichenwerte in Messages Numerische Werte in Messages	~~Zeichenwerte in Messages~~ Numerische Werte in Messages
Statische Objekte über die Zeit *t* ändern	~~Statische Objekte über die Zeit *t* ändern~~

Dynamische Objekte über die Zeit *t* ändern	Dynamische Objekte über die Zeit *t* ändern
Audioprocessing kann ein- oder ausgeschaltet werden	~~Audioprocessing kann ein- oder ausgeschaltet werden~~
`send~` und **`receive~`** Objekte sind global miteinander verbunden	**`send~`** und **`receive~`** Objekte sind innerhalb einer **`rnbo~`** Instanz miteinander verbunden

Tabelle 5.1: Unterschiede zwischen MAX/MSP und Rainbow

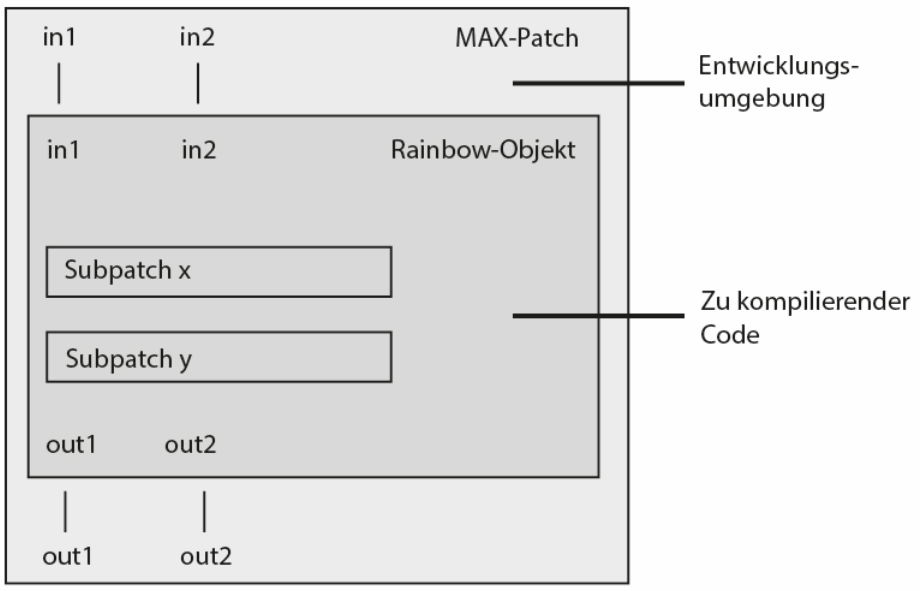


Abb. 5.1: Einbindung eines Rainbow-Patches in ein MAX-Projekt

5.3 Verwendete Objekte und Subpatches

Die folgende Aufzählung beinhaltet alle verwendeten Objekte ab und listet die erstellten Subpatches auf, die zur Gliederung des Programms erstellt wurden:

rnbo ~ Das Rainbow-Objekt wird innerhalb eines MAX-Patches insertiert und verhält sich wie ein Subpatch. Jegliche Signalverarbeitung, die innerhalb des Rainbow-Patches erstellt wird kann exportiert werden als VST oder AU-Plugin, als Raspberry Pi Code, als C-Code oder als JavaScript für Webseiten.

param Das param-Objekt definiert Parameter des Rainbow-Objekts. Es wird als Steuersignal in der MAX-Ebene definiert und über Patchkabel an das Rainbow-Objekt weitergereicht. Die definierten Parameter werden in der Nutzeroberfläche des VST-Plugins als generisch generierte Benutzeroberfläche dargestellt. Es kann die Parameter über `@name @min @max @value` definieren. `@value` stellt dabei einen Initialwert dar.

attrui Das attrui-Objekt kann als grafische Oberfläche innerhalb eines MAX-Patches definiert werden. Dadurch lassen sich in der Ebene darüber die über param definierten Parameter des Objekts über diese Oberfläche verändern. Es wird über ein Patchkabel mit dem zu steuernden Objekt verbunden.

vst ~ Das vst-Objekt kann innerhalb einer MAX-Umgebung ein VST-Plugin hosten. Das Objekt kann über eine definierte Anzahl an Ein- und Ausgängen individualisiert werden.

gain~ Das gain-Objekt erstellt einen interaktiven Fader mit Pegelanzeige in dB. Dieser kann ein Audiosignal in einer logarithmischen Skala absenken oder verstärken.

toggle Das toggle-Objekt ist ein Schalter, der die Werte 0 und 1 ausgeben kann.

scope~ Das scope-Objekt verfügt über eine grafische Oberfläche, die Audiosignale visuell darstellen kann.

add ~ Das add-Objekt kann innerhalb eines Rainbow-Patches zwei Audiosignale miteinander addieren.

in~ Das in-Objekt erzeugt einen Audioeingang innerhalb eines Subpatches.

in Das in-Objekt (ohne Tilde) erzeugt einen Steuersignaleingang innerhalb eines Subpatches

out~ Das out-Objekt erzeugt einen Audioausgang innerhalb eines Subpatches.

***~** Der *-Operant multipliziert zwei Audiosignale oder ein Audiosignal und ein Steuersignal miteinander. Es kann ein Initialwert definiert werden.

-~ Der *-Operant subtrahiert zwei Audiosignale oder ein Audiosignal und ein Steuersignal voneinander. Es kann ein Initialwert definiert werden.

scale Das scale-Objekt kann den Steuersignalumfang um ein definiertes Verhältnis ändern. (siehe unten)

p Das p-Objekt erstellt einen Subpatch. Das ist eine Art Unterordner, der zur Strukturierung des Programms verwendet wird.

Der Subpatch p `switch` ist ein kleines Hilfsprogramm, das die Daten von Audiokanälen vertauschen kann. Es hat beispielsweise zwei Audioeingänge und einen Steuersignaleingang des Typs `boolean {0,1}`. Ein `scale`-Objekt sendet zum einen ein Signal unverändert und invertiert das andere.

`scale 0. 1. 1. 0. und scale 0. 1. 0. 1.`

Anschließend werden die einzelnen Signale miteinander multipliziert.
Dieser Mechanismus kann für beliebig viele Audiokanäle angewendet werden.
Anschließend gibt das Objekt die vertauschten Audiokanäle aus.

$$out1[n] = a * in1[n] + b * in2[n]$$
$$out2[n] = a * in2[n] + b * in1[n]$$

Der Subpatch p `offset` besitzt drei Audioeingänge und einen Steuersignaleingang des Typs `boolean {0,1}`. Das Steuersignal aktiviert die Signalverarbeitung. In diesem Patch werden die Signale in ihren Zahlenwerten $-1 \leq n \leq 1$ zu den Zahlenwerten $0 \leq n \leq 1$ konvertiert. Aus der Elektrotechnik ist dieser Vorgang auch als DC-Offset bekannt.

$$out[n] = \frac{1}{2}(in[n] + 1)$$

Der Subpatch p `phaseCouples` besteht aus drei Audioeingängen und fünf Audioausgängen. Der erste Kanal wird durchgeleitet. Für den zweiten und dritten Kanal wird je ein weiterer Partnerkanal angelegt, dessen Signal invertiert wird.

$$outX[n] = in[n]$$
$$outY[n] = (-1) * [in]$$

Das Programm ist in vier logischen Programmabschnitte unterteilt. Sie setzen den konzipierten Audio- und Steuersignalfluss aus Abschnitt 4.2 um. Die Steuersignale werden in der obersten Eben des Rainbow-Objekts über das Objekt `param` definiert und in die Subpatches übergeben. Das Objekt definiert Namen in der Nutzeroberfläche, minimal-, maximal- und Initialwerte. Beispielsweise:

```
param "2 Height" @name height
@min 0. @max 100. @value 100.
```

Die folgenden Abbildungen zeigen die Umsetzung der Programmierung in Rainbow. Verwendet wurden die oben erläuterten Multiplikations- und Additions-Operatoren, welche Steuersignale der obersten Patchebene empfangen und an die Subpatches weitergereichen. Die number-Objekte dienen lediglich zur Übersicht und haben keinen Einfluss auf das Audio- oder Steuersignal.

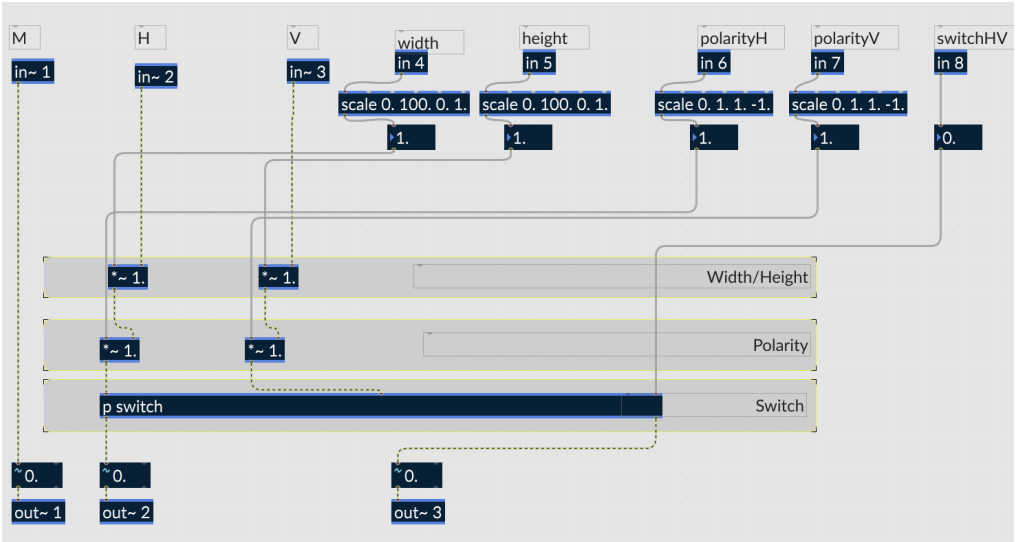


Abb. 5.2: Programmierung von Subpatch `p editMHV`

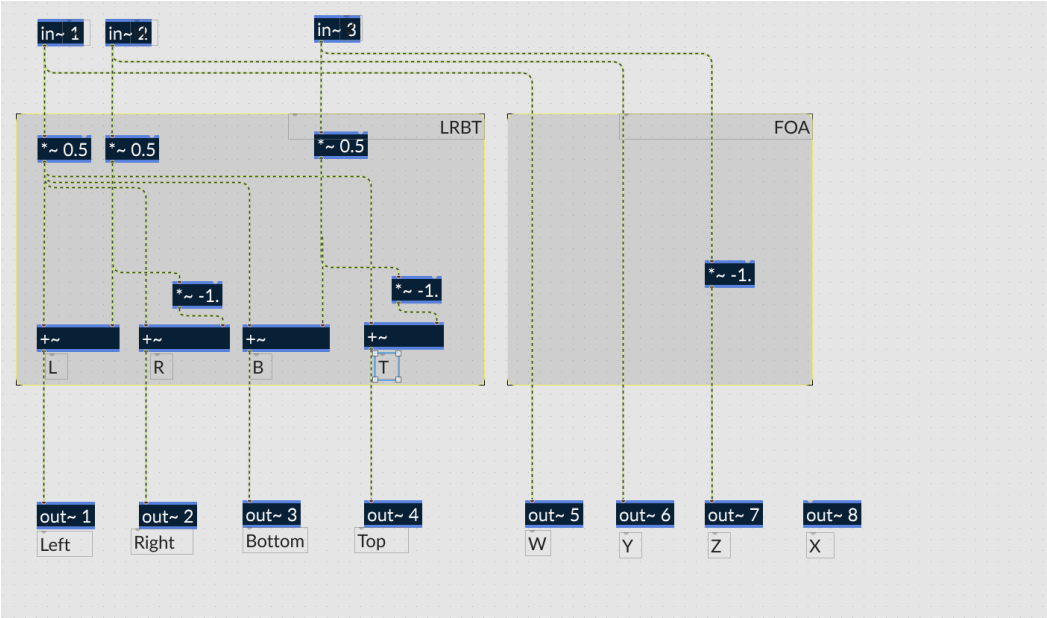


Abb. 5.3: Programmierung von Subpatch `p decoder`

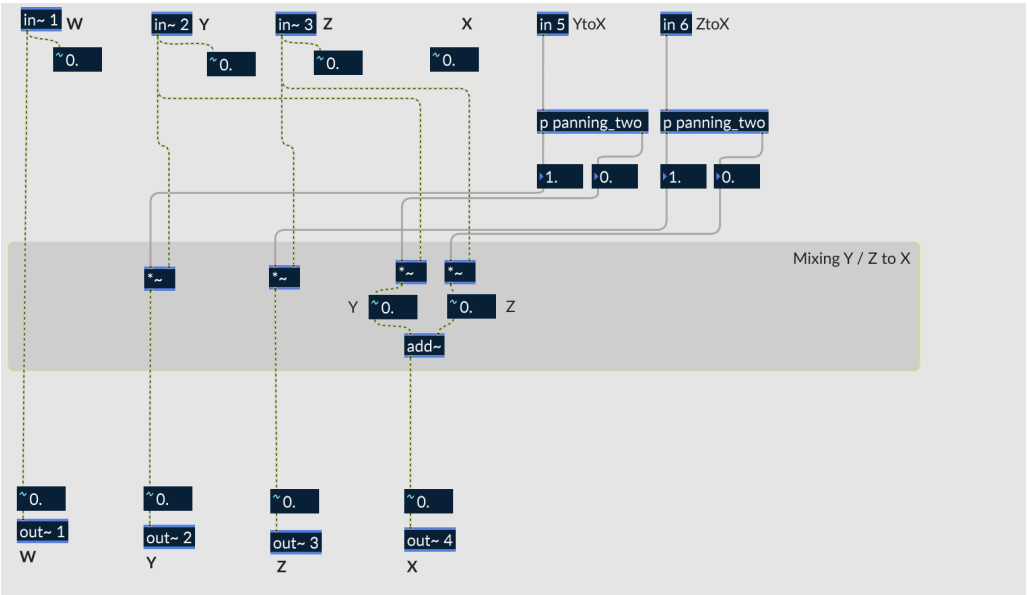


Abb. 5.4: Programmierung von Subpatch `p editFOA`

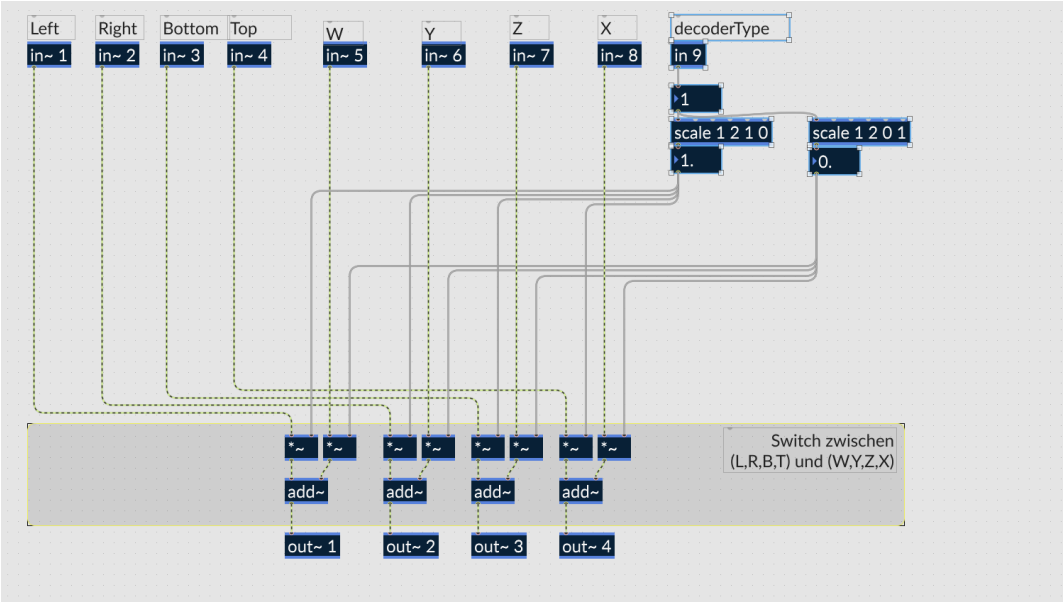


Abb. 5.5: Programmierung von Subpatch `p switchFour`

Der Ausgang des Subpatches `switchFour` ist mit den Ausgängen `out~ 1`, `out~ 2`, `out~ 3` und `out~ 4` des Rainbow-Objekts der obersten Ebene verbunden. Das kompilierte VST-Plugin wird diese Ausgänge als Rückgabewerte dem Host, also beispielsweise einer DAW, übergeben. Um die erläuterten Signalverarbeitungen zu überprüfen, wird das Rainbow-Objekt in eine Testumgebung integriert, welche nun in folgendem Kapitel vorgestellt wird.

6 Verifizierung

In diesem Abschnitt werden zwei Möglichkeiten vorgestellt, die Signalverarbeitung zu verifizieren. Dabei wird zum einen ein MAX-Patch verwendet und zum anderen ein Reaper-Projekt erstellt, in dem die generierten Signale aufgezeichnet werden. Beide Projektdateien liegen dieser Arbeit als Anhang bei.

6.1 MAX-Projekt

Um das Rainbow-Objekt zu überprüfen, wird eine interaktive Testumgebung entwickelt, welche die Signale eines MHV-Stützmikrofons simulieren kann. Es stehen hierbei unterschiedliche Testsignale zu Verfügung. (Abschnitt 4.3) Es können beim Testen auf alle Parameter des Decodierungstools zugegriffen werden, um das Verhalten der Signalverarbeitungskette anhand unterschiedlicher Analysetools zu überprüfen. Um die interaktive Testumgebung zu realisieren, wird auf der obersten Ebene des MAX-Patches eine Verifizierungsumgebung erzeugt, mit der die Manipulation der Testsignale möglich ist. Mithilfe der Subpatches *offset* und *phaseCouples* (siehe Kapitel 5.3) werden die Signale manipuliert und anschließend miteinander gemischt. Die manipulierten Signale dienen als Eingangssignal des Rainbow-Objekts.

Im Anschluss werden die fünf Signale in einem Mischer gegeben. Die Ausgänge des Mischers simulieren die Ausgangssignale eines MHV-Stützmikrofons. Um das Verhalten des Decodierungstools zu verifizieren, werden an den Ein- und Ausgängen des Rainbow-Objekts Oszillatoren implementiert, die das Verhalten des Plugins visualisieren. Hierfür bietet sich das Testsignal *sinewave* (siehe Abschnitt 4.3) besonders gut an, da hier die Phasenlage abgelesen werden kann.

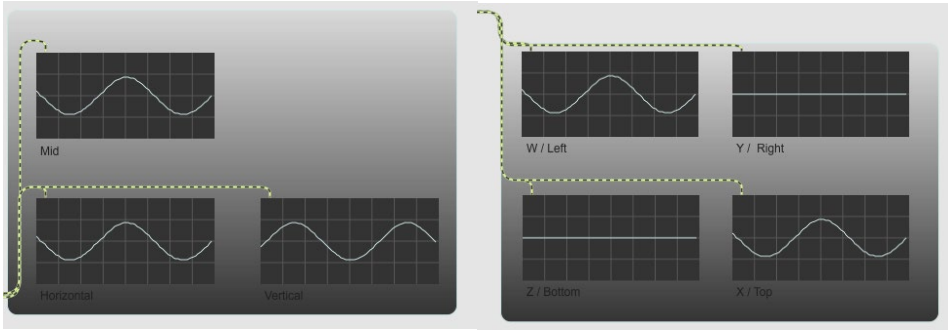


Abb. 6.1: Eingangs- und Ausgangssignale des Rainbow-Patches

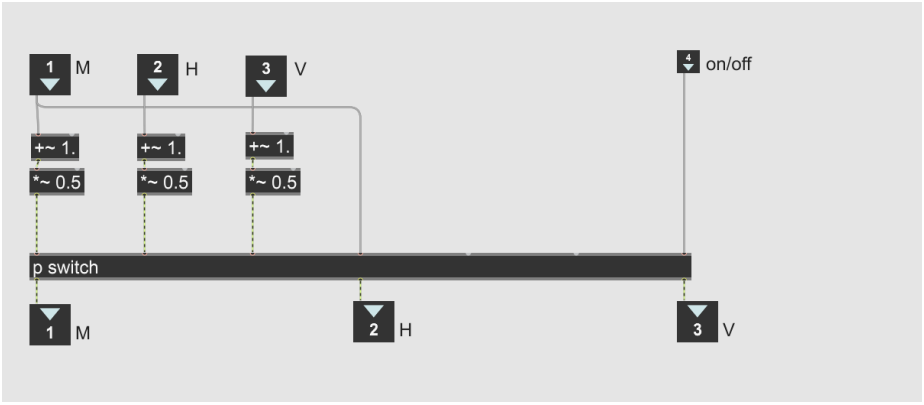


Abb. 6.2: Programmierung von Subpatch *offset*

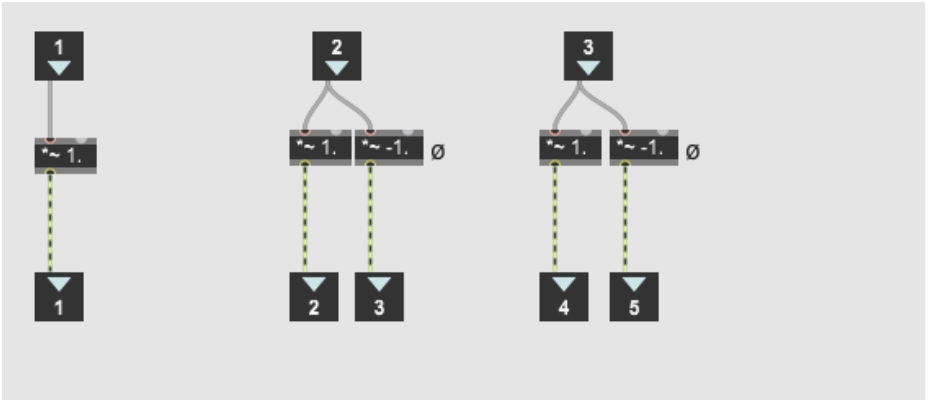


Abb. 6.3: Programmierung von Subpatch *phaseCouples*

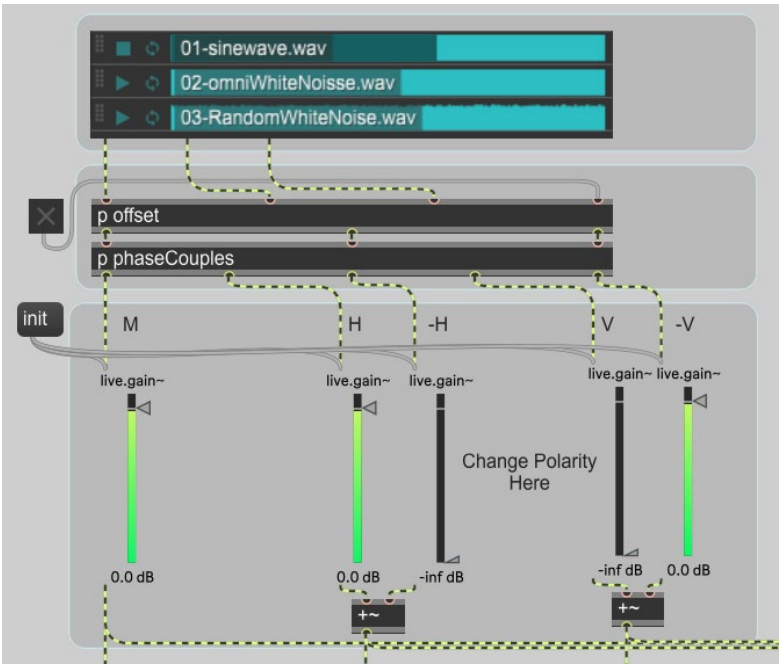


Abb. 6.4: Manipulation der Testsignale

6.2 Externe 3D-Audio Anwendung

Bisher beschäftigte sich die Verifizierung allein mit der implementierten Schaltung im Rainbow-Objekt. Der MAX-Patch dient bei der Programmierung auch als Debugging-Plattform um Fehler während dem Prozess der Programmierung aufzudecken. Zur Verifizierung des VST-Plugins wird eine weitere Testumgebung erstellt. Das stellt sicher, dass auch beim Export des Plugins zu keinen Fehlern kommt. Hierzu dient die DAW Reaper (*REAPER | Download*, 2023).

Zuerst wird ein Monokanal *sigGen* erstellt und ein Sinusgenerator insertiert. Im Anschluss werden fünf weitere Monokanäle (*M*, *V+, V-, H+, H)* erstellt, die *sigGen* empfangen. Über sie kann das Signal manipuliert werden.

Pegel- und Phasenlage der Sinusschwingung kann verändert werden. Die Signale laufen anschließend in einen dreikanaligen Bus *MHV*. In einem weiteren Bus *decoding* wird das VST-Plugin geladen. Es empfängt die Daten des MHV-Busses und sendet sie anschließend an den *LRBT*-Bus (siehe Abschnitt 4.3) Dieser wiederum wird im Anschluss aufgeteilt in vier Monokanäle. Zeichnet man nun während der Änderung der Eingangswerte den *MHV*- und *LRBT*-Bus als Multi-Wave-Datei auf können verschiedene Fälle miteinander verglichen werden. Dies wird nun stellvertretend für zwei Fälle aufgezeigt.

Werden bei gleichem Pegel und gleicher Phasenlage $\varphi = 0°$ der Eingangssignale die Ausgangssignale betrachtet, so werden Pegel im Ausgangsbus in den Kanälen *Left* und *Bottom* festgestellt.

Werden die Eingangskanäle bei gleichem Pegel aber einer Phase $\varphi = 180°$ in Kanal *Horizontal* mit den Ausgangssignalen verglichen, werden Pegel im Ausgangsbus in Kanal *Right* und *Bottom* festgestellt.

Durch parallele Aufzeichnungen der Eingangs- und Ausgangswerte des VST-Plugins in Reaper kann die korrekte Signalverarbeitung der Signale überprüft werden, hierzu werden die Busse *MHV* und *LRBT/FOA* mitgeschnitten. Für die Visualisierung der *FOA*-Codierung kann in dem *FOA*-Bus ein geeignetes Plugin verwendet werden. Im Projektordner des MAX-Patches, welcher dieser Arbeit als Anhang beiliegt, befindet sich ein Vorschlag. Es handelt sich um das VST-Plugin Energy Visualizer des Instituts für elektronische Musik (*IEM Plug-in Suite*, 2023). Die DAW Reaper bietet auch die Möglichkeit der Manipulation der Eingangssignale. In Reaper kann in dem integrierten Mixer die Phase des Signals verändert werden. Für dekorrelierte Signale ist es notwendig in jeden der Eingangskanäle einen eigenen Signalgenerator zu insertieren. In den folgenden beiden Abbildungen sind die Messergebnisse für die oben beschriebenen Testsignale dargestellt.

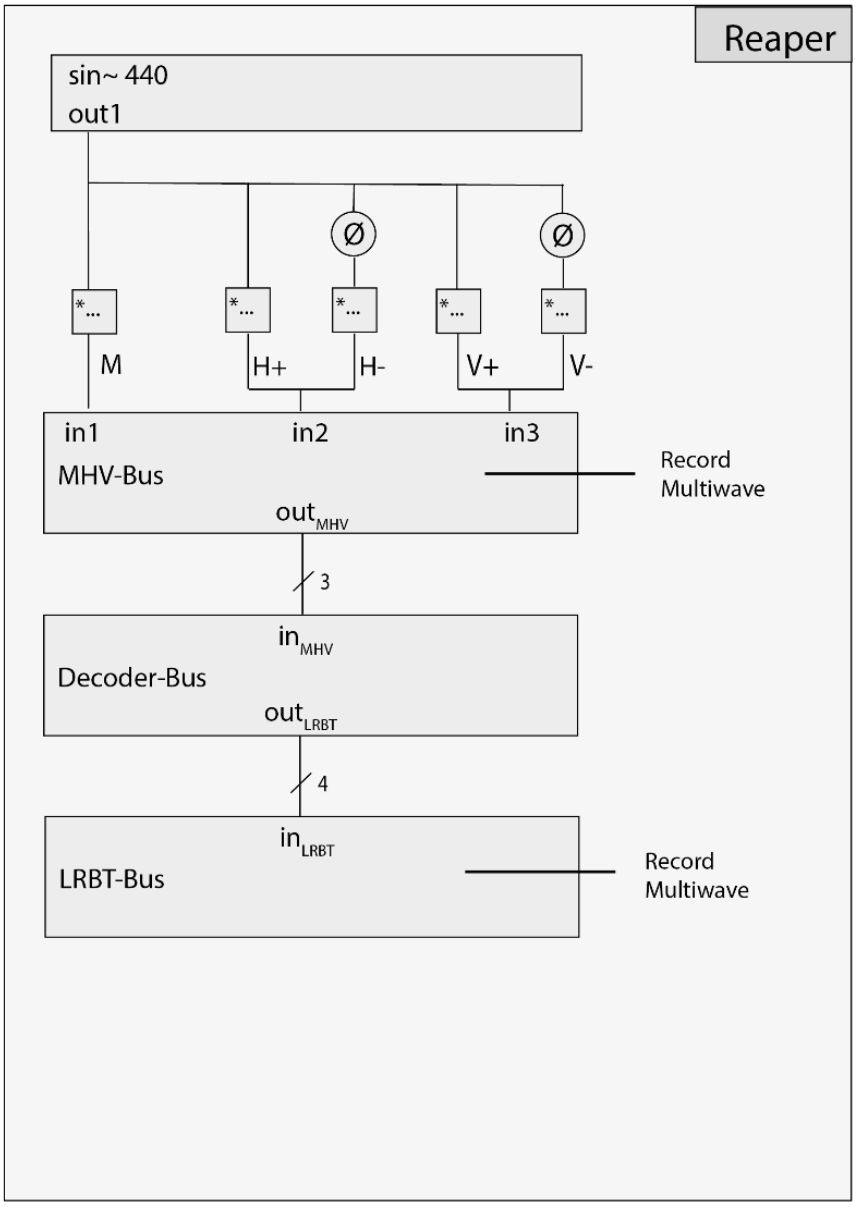


Abb. 6.5: Skizzierung des Reaper-Projekts

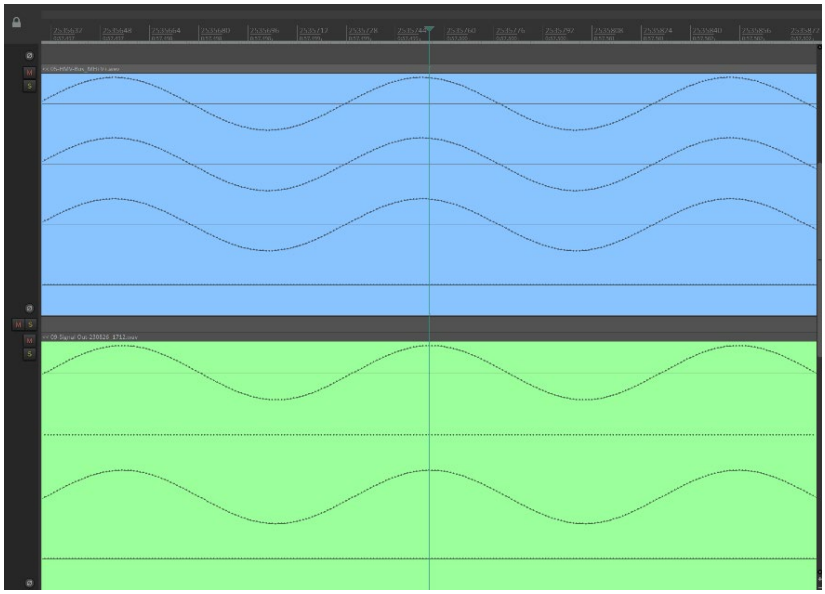


Abb. 6.6: Vergleich der Eingangs- und Ausgangssignale des Decoders (*LRBT*) bei gleicher Phasenlage der Eingangssignale

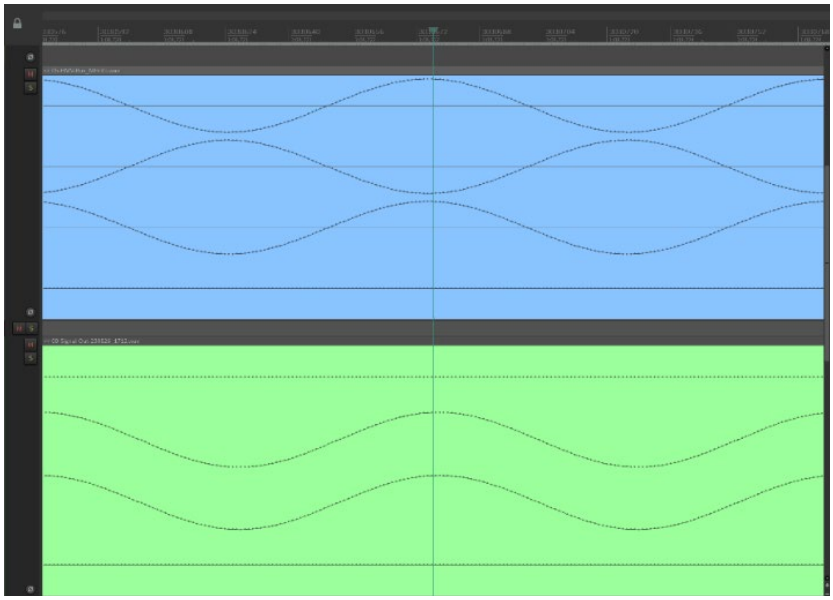


Abb. 6.7: Vergleich der Eingangs- und Ausgangssignale des Decoders (*LRBT*) bei invertiertem horizontalen Signalanteil des Eingangssignals

7 Diskussion und Ausblick

Erklärtes Ziel dieser Arbeit ist es, die Programmierumgebung von Rainbow der Firma Cycling ’74 (*Introducing RNBO | Cycling ’74*, 2023) zu nutzen, um ein plattformübergreifendes VST-Plugin zu entwickeln und zu verifizieren. Die grafische Programmierumgebung MAX/MSP bietet hierbei gegenüber der codebasierten Programmierung deutliche Vorteile, da sie an die Struktur einer elektronischen Schaltung erinnert und somit Datenströme leicht nachzuverfolgen sind. Dies schafft für Medieningenieure eine intuitive und vertraute Arbeitsumgebung. Das neu entwickelte Objekt `rnbo~` schlägt hierbei die Brücke zur plattformübergreifenden Nutzung dieser Patches und ermöglicht dem Programmierer den branchenüblichen VST-Standard zu exportieren. Die Entwicklungsumgebung bietet die Möglichkeit komplizierte und höchst individuelle Instrumente und Signalverarbeitungen, beispielsweise Audioeffekte, Decoder oder Synthesizer zu realisieren und in den bestehenden Workflow einer 3D-Audio-Anwendung ohne Kompatibilitätsprobleme zu integrieren. Das eröffnet dem Nutzer* dieser Software die Möglichkeit unkonventionelle Herangehensweisen und individuelle Problemlösungsstrategien in eine bekannte Umgebung zu integrieren. Hiermit sei festgehalten, dass das gesetzte Ziel einen universell nutzbaren Decoder im VST-Standard zu programmieren mithilfe der Software MAX/MSP und dem Objekt Rainbow erreicht wurde.

Ob und inwieweit es jedoch in Zukunft die Möglichkeit geben wird eine individuell gestaltete Benutzeroberfläche zu implementieren bleibt abzuwarten. Das schränkt aktuell noch die Individualisierung stark ein und ist der größte Nachteil gegenüber der konventionellen codebasierten Programmierung wie C+. Es müssen auf Umwegen Schalter und Auswahlfenster improvisiert werden, die sich von anderen Nutzeroberflächenelementen optisch nicht unterscheiden und das Programm somit unübersichtlich erscheinen lassen.

Hier sehe ich großes Verbesserungspotential bei der zukünftigen Weiterentwicklung von Rainbow. Es bleibt abzuwarten, ob Cycling ’74 mit der weiterführenden Entwicklung dieses Objekts auf den Nachteil der Gestaltung der Benutzeroberfläche eingeht. Beispielsweise wäre denkbar, eine ähnliche Lösung wie bei Max4Live zu realisieren, bei der die Präsentationsebene des MAX-Patches als Nutzeroberfläche für die Ableton-Erweiterung genutzt werden kann. (*Max for Live | Cycling ’74*, 2023)

Des Weiteren ist es fraglich, ob ein automatisiert generierter Quellcode effektiv und somit ressourcenschonender als von Hand programmierter Code ist. Das ist in der Welt der Audiobearbeitung, vor allem bei dem mobilen Einsatz von Plugins, wichtig. Beispielsweise an einem Laptop auf einer mobilen Produktion. Hierzu können fortführend Studien zum Ressourcenverbrauch und zu der generierten Latenz des Plugins durchgeführt werden.

Die hier vorgestellte Arbeit kann als einer von weiteren Schritten zur plattformunabhängigen Nutzung des MHV-Stützmikrofons angesehen werden.
Sie bietet neben der Decodierung in die zwei Zielformate LRBT und FOA dem Nutzer* auch die Möglichkeit, die MHV-Signale vor der Decodierung zu manipulieren und somit die Ausdehnung in horizontaler und vertikaler Achse sowie deren Polarität zu ändern. Zusätzlich wird dem Nutzer* bei Wahl des Zielformats FOA die Möglichkeit gegeben, horizontale und vertikale Informationen auf die X-Achse des Formats zu routen. Weitere Entwicklungsschritte dieses Zwischenstands können beispielsweise die Implementierung einer individualisierten Nutzeroberfläche sein oder die Erweiterung der Signalverarbeitung um ein amplitudenbasiertes Panningtool, welches in den Zielformaten LRBT und FOA eine Anordnung der virtuellen Schallquelle mithilfe von Azimut und Elevation ermöglichen. Auch vordefinierte dezidierte Lautsprecheranordnungen, wie in der Arbeit

von Strobel (2021) sind denkbar. Zusätzliche Decodierungs- und Downmixformate können das universale Einsatzgebiet des Plugins erweitern.

Die Verifizierungsumgebung bietet die Möglichkeit, mithilfe von unterschiedlichen Testsignalen die Signalverarbeitung zu überprüfen. Dem Nutzer* wird dabei die Möglichkeit gegeben die Eingangssignale zu manipulieren und somit unterschiedliche Fälle des MHV-Signals zu simulieren. Stellvertretend wurden hier zwei dieser möglichen Eingangssignale mit den Ausgangssignalen verglichen. Weitergehend könnte diese Auswahl der Testsignale um real aufgenommene Mehrkanalsignale des MHV-Stützmikrofons erweitert werden, welche Hofmann (2020) im Rahmen der Masterarbeit im reflektionsarmen Raum aufgezeichnet hat. Zusätzlich wäre die Ergänzung dieser weiterführenden Umgebung um eine trackingbasierte Binauralisierung denkbar, um raumunabhängig über Kopfhörer das Resultat der Decodierung zu hören und nicht nur anhand von Oszilloskopen abzulesen.

Abschließend möchte ich betonen, dass diese Arbeit auf Entwicklungen und Erkenntnissen von Leon Hofmann (2020) und Maurice Strobel (2021) beruhet und lediglich einen weiteren Schritt in diesem Forschungsgebiet darstellt. Ein daran anschließendes Thema wäre beispielsweise die Nutzung der weiterentwickelten Testumgebung zur Evaluation durch Hörversuche.

Literaturverzeichnis

Dickreiter, M. 1942-, Schule für Rundfunktechnik and ARD.ZDF medienakademie (2023) *Handbuch der Tonstudiotechnik*.

Elen, R. and Carbines, P. (2006) 'Getting Ambisonics Around-Page'. Available at: www.gerzonic.net (Accessed: 24 August 2023).

Grzesinski, C. and Smyrek, V. (1973) *Tontechnik für Veranstaltungstechniker in Ausbildung und Praxis : mit 706 Abbildungen, 71 Tabellen und 152 Übungsaufgaben*. S. Hirzel Verlag Stuttgart.

Hofmann, L. (2020) *Konzeption und Erprobung eines Stützmikrofonverfahrens zur ausgedehnten Abbildung akustischer Instrumente in mehrdimensionalen Audiomischungen*.

IEM Plug-in Suite (2023). Available at: https://plugins.iem.at/ (Accessed: 22 August 2023).

Introducing RNBO | Cycling '74 (2023). Available at: https://cycling74.com/products/rnbo (Accessed: 24 August 2023).

Jens Blauert (1997) *Räumliches Hören*.

Key Differences | Cycling '74 (2023). Available at: https://rnbo.cycling74.com/learn/key-differences (Accessed: 24 August 2023).

Max for Live | Cycling '74 (2023). Available at: https://cycling74.com/products/maxforlive (Accessed: 27 August 2023).

Philip Mackensen aus Berlin, M. (2004) *Auditive Localization. Head movements, an additional cue in Localization vorgelegt von*.

Pulkki, V. and Karjalainen, M. (2015) *Communication Acoustics*. Wiley. Available at: https://doi.org/10.1002/9781119825449.

Pulkki, Ville. (2001) *Spatial sound generation and perception by amplitude panning techniques*. Helsinki University of Technology.

REAPER | Download (2023). Available at: https://www.reaper.fm/download.php (Accessed: 26 August 2023).

RNBO ('rainbow'): Start in Max's UI, deliver to plug-ins, Web browsers, hardware, any OS - CDM Create Digital Music (2023). Available at: https://cdm.link/2022/11/rnbo-max-for-web-hardware-plugin/ (Accessed: 24 August 2023).

Sontacchi, A. *et al.* (2011) *AMBIX-A SUGGESTED AMBISONICS FORMAT*. Available at: http://ambisonics.iem.at/xchange/meeting08/.

Strobel, M. (2021) *Entwicklung eines 3D-Audio-Postproduktionstools für eine dreikanalige koinzidente Stützmikrofonanordnung*.

Weinzierl, S. (2008) *Handbuch der Audiotechnik*, *Handbuch der Audiotechnik*. Springer Berlin Heidelberg. Available at: https://doi.org/10.1007/978-3-540-34301-1.

What is Max? | Cycling '74 (2023). Available at: https://cycling74.com/products/max (Accessed: 29 August 2023).

Wittek, H., Haut, C. and Keinath, D. (2006) 'Doppel-MS-eine Surround-Aufnahmetechnik unter der Lupe Double M/S-a Surround recording technique put to test'. Available at: www.schoeps.de. (Accessed: 28 August 2023).

Zotter, F. and Frank, M. (2019) *Springer Topics in Signal Processing Ambisonics A Practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement, and Virtual Reality, Volume 19*. Available at: http://www.springer.com/series/8109.

A Digitaler Anhang

Dieser Arbeit liegt ein digitaler Anhang in folgender Ordnerstruktur bei:

./PDF	Die hier vorliegende Arbeit als PDF-Dokument
./VST	Exportiertes VST-Plugin für Mac OSX und für Microsoft Windows
./MAX	Testumgebung als MAX-MSP Projektdatei, in dieser befindet sich auch das Rainbow-Objekt
./REAPER	Testumgebung als Reaper-Projektdatei, in der Testaufnahmen vorliegen