Trennung von Sprache und Hintergrundgeräuschen

Bachelorarbeit aus Aufnahmetechnik 1, SE

Lorenz Häusler

Betreuung: Univ.Prof. DI Dr. Alois Sontacchi

Graz, 26. Juni 2018

institut für elektronische musik und akustik

Trennung von Sprache und Hintergrundgeräuschen

Bachelorarbeit aus Aufnahmetechnik 1, SE

Lorenz Häusler

Betreuung: Univ.Prof. DI Dr. Alois Sontacchi

Graz, 26. Juni 2018

institut für elektronische musik und akustik

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 26. Juni 2018

. . . . . .

(Unterschrift)

Zusammenfassung

In dieser Bachelorarbeit sollte eine Methode für das Herausfiltern von Sprache aus einem 2-Kanal-Audiosignal für Telefonieandwendung definiert und ausgearbeitet werden. Hierfür wurde die Kohärenzfunktion zweier Kanäle gebildet und mithilfe spektraler Filterung bzw. Manipulation eine Maske generiert, welche die Signalanteile, die vorwiegend Sprache enthalten, isoliert aus dem Eingangssignal extrahiert. Der in Matlab ausgearbeitete Algorithmus wurde in JUCE als VST-Plugin implementiert und gestestet.

Abstract

In this bachelor thesis, a method was to be found, to filter noise from a two-channel audio-speech-signal for the use with telephone communication. To achieve this, the coherence function of those two channels was formed and modified in the frequency domain to create a mask. This mask was then applied to said signal to extract the voice precisely. The proposed algorithm was tested in Matlab and implemented as a VST-Plugin in JUCE.

Inhaltsverzeichnis

1 Einleitung

Die Verbesserung der Sprachverständlichkeit von digitalen Audiosignalen ist schon lange ein Thema vieler Publikationen, weshalb es eine Vielzahl von Verfahren für ein- und mehrkanalige Aufnahmen gibt. Bei der Trennung von Sprache und Rauschen von mehrkanaligen Signalen, bedienen sich diese Verfahren meist der Kreuzkorrelations-Methode, bei welcher mithilfe der Kohärenzfunktion eine spektrale Maske Φ generiert wird, die kohärente Anteile (Sprache) von unkorrelierten Anteilen (Rauschen) trennt [TSBDA17]. Die erstellte Maske muss noch nachbearbeitet werden, da sie meist das Nutzsignal nicht perfekt isoliert. In so gut wie jeder Anwendung die sich dieser Methode bedient, wird die Maske zeitlich geglättet, was zu schnelles Aus- und Eingreifen der Maske verhindert. Die Methode hat jedoch auch seine Schwächen. Eine davon sind die zufälligen Kohärenzen im Rauschen, die u.A. zu Musical-Noise führen können(siehe Kapitel 2.4). Um diese zu unterdrücken, gibt es z.B. den Ansatz der Einführung einer Gewichtsfuntkion $\Gamma(\Phi)$, welche mithilfe einer variablen Tangens-Hyperbolikus Funktion kleine Werte der Maske, welche durch Rauschen entstehen gegen 0 und große Werte, die das Nutzsignal repräsentieren, gegen 1 streben lässt [CJM02]. Dieses Problem kann auch mithilfe von spektraler Glättung behoben werden, welche spektrale Spitzen der Maske ein- und ausfadet. Ein weiterer Punkt in der Verbesserung dieser Methode ist die zeitliche Anpassung der einzelnen Kanäle. Das Sprachsignal kommt in realen Messumgebungen nie exakt gleichzeitig an den verwendeten Mikrofonen an. Deswegen ist es notwendig, dass die Verzögerung ermittelt und ausgeglichen wird. Hierfür wird die sogenannte TDOA Methode wie in [TSBDA17] beschrieben verwendet.
Der Aufbau für die Erfassung der Testdaten für den Algorithmus bestand aus einer Audioquelle und mehreren Mikrofonen. Durch die verschiedenenn Lagen der Sensoren kann der Sprecher als konkrete Quelle im Raum mit definierter Richtung geortet werden. Dadurch ergeben sich neben den kohärenten Anteilen in den Sensorsignalen zusätzlich die definierte räumliche Information. Zusammenfassend beschäftigt sich diese Arbeit nun mit folgenden Punkten:

1. Erstellung eines Algorithmus, welcher eine Maske mithilfe der Kreuzkorrelations-Methode generiert.
2. Manipulation bzw. Anpassung der Maske, um ein besser verständliches Sprachsignal zu erhalten
3. Implementierung als VST-Plugin, um die Steuerung der Parameter intuitiver zu gestalten und eine bessere Visualisierung zu ermöglichen.

2 Mathematische Grundlagen

Da es sich um eine Echtzeitanwendung handelt, in der spektrale Manipulationen durchgeführt werden, wird vorausgesetzt, dass die Eingangsdaten in Frames behandelt werden. In folgendem Kapitel wird ein Signal in Abhängigkeit seiner Frequenzbins k, und des aktuellen Frames l betrachtet. Komplexe Variablen werden unterstrichen gekennzeichnet.

$$\underline{X}(k,l) = FFT\left\{x_l[n]\right\},\ \underline{X}(k,l) \in \mathbb{C} \tag{1}$$

$x_l[n]$ entspricht hierbei einem Ausschnitt des Eingangssignals.

2.1 Kohärenzfunktion

Sie beschreibt die lineare Abhängigkeit zweier Signale über die Frequenz. Für ihre Berechnung werden die Auto- bzw. Kreuzleistungsspektren der Signale benötigt:

$$P_{11}(k,l) = \underline{X_1}(k,l) \cdot \underline{X_1}^*(k,l) \tag{2}$$

$$P_{22}(k,l) = \underline{X_2}(k,l) \cdot \underline{X_2}^*(k,l) \tag{3}$$

$$\underline{P_{12}}(k,l) = \underline{X_1}(k,l) \cdot \underline{X_2}^*(k,l) \tag{4}$$

Die Kohärenzfunktion ist wie folgt definiert:

$$\underline{\Phi_{12}}(k,l) = \frac{\langle \underline{P_{12}}(k,l) \rangle}{\sqrt{\langle P_{11}(k,l) \rangle \cdot \langle P_{22}(k,l) \rangle}} \tag{5}$$

Der Zähler ist in der Regel komplexwertig, der Nenner immer reell, was eine komplexwertige Maske zur Folge hat. Im Falle von linearer Abhängigkeit, entspricht der Wert des Kreuzleistungsspektrums dem der Wurzel des elementweisen Produkts der Autoleistungsspektren und der Wert beim betrachteten Frequenznbin strebt betragsmäßig gegen 1. Falls die Signale jedoch linear unabhängig sind, hat das Kreuzleistungsspektrum einen betragsmäßig kleineren Wert als der Nenner und in weiterer Folge wandert die Kohärenzfunktion in Richtung 0. Die verschiedenen Parameter für die Maskenberechnung müssen zeitlich geglättet werden(mit $\langle\rangle$ gekennzeichnet), da es durch zu große Phasensprünge im schlimmsten Fall zu einer Auslöschung der Zeiger kommen könnte.

2.2 Zeitverzögerungsabschätzung

Damit der Phasenwinkel der Kohärenzfunktion genau ist, müssen beide Signale zeitgleich an den Mikrofonen ankommen. Da das in der Realität selten der Fall ist, muss ein Weg gefunden werden, die Zeitverzögerung abzuschätzen und auszugleichen. Praktischerweise ergibt die inverse Fouriertransformation(IFFT) des Kreuzleistungsspektrums die Kreuzkorrelationsfunktion, wessen Maximum der Zeitverzögerung der beiden Signale entspricht.

$$\tau = index(max(IFFT\left\{P_{12}(k,l)\right\})) \tag{6}$$

In der digitalen Signalvearbeitung, kann die Zeitverzögerung auf diese Art und Weise nur auf ein Sample genau berechnet werden, und ist damit abhängig von der Samplefrequenz. Um die Genauigkeit zu erhöhen, kann zusätzlich eine parabolische Interpolation um den Bereich des Maximums durchgeführt werden, welche es ermöglicht eine Zeitverzögerung im sub-Samplebereich zu berechnen.

$$\Delta_{sub} = \frac{x[n-1] - x[n+1]}{2 \cdot (x[n-1] - 2 \cdot x[n] + x[n+1])} \tag{7}$$

Um die berechnete Verzögerung auszugleichen, muss abhängig vom Vorzeichen der Verzögerung eines der beiden Eingangssignale zeitlich verschoben werden.

$$\underline{X'}(k,l) = \underline{X}(k,l) \cdot e^{j \cdot 2 \cdot \pi \cdot \frac{k}{fs} \cdot \tau} \tag{8}$$

Mit der bestimmten Zeitverzögerung und dem Abstand der Mikrofone lässt sich der Einfallswinkel der Quelle berechnen (siehe Abbildung 1).

$$\Theta = cos^{-1}\left(\frac{\tau \cdot c}{d}\right) \tag{9}$$

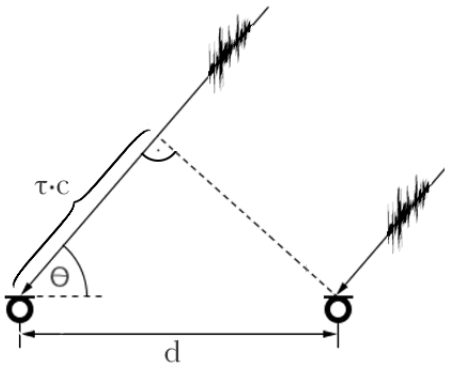

Abbildung 1 – Skizze zur geometrischen Anordnung der beiden Mikrofone. Definition des Einfallswinkels Θ und Angabe der relativen geometrischen Abmessungen bei Annahme eines ebenen Wellenfeldes.

2.3 Zeitliche Glättung

Die berechnete Maske soll zeitlich geglättet werden, um bei zu schnellem Aus- bzw Eingreifen der Maske Fauchen zu verhindern. In dieser Methode wird ein simpler Tiefpass erster Ordnung verwendet, dessen Glättung mithilfe eines Faktors fgf gesteuert wird. Der Wert des Faktors liegt zwischen 0 (keine Glättung) und 1(Wert bleibt konstant).

$$\underline{\Phi_{12,sm}}(k,l) = (1 - fgf) \cdot \underline{\Phi_{12}}(k,l) + fgf \cdot \underline{\Phi_{12,sm}}(k,l-1) \tag{10}$$

Diese Art der Glättung wird auch für andere Größen wie z.B die Änderung der Zeitverzögerung verwendet.

2.4 Spektrale Glättung

Ein Nachteil der Kohärenzfunktion ist, dass das unkorrelierte Rauschen beider Kanäle zufällig in Phase sein kann, was zu vereinzelten Spitzen in der Maske führt. Wenn die Maske nun auf das zu verarbeitende Signal multipliziert wird, ergibt sich sogenannter "Musical Noise", welcher sich im Ausgangssignal als zufällig verteilte Sinuskomponenten bemerkbar macht. Diese Spitzen können mithilfe spektraler Glättung gemindert bzw. beseitigt werden. Um diese Glättung zu erreichen, wird der Realteil der berechneten Maske mit einem Gauß-Fenster $W(k)$ gefalten.

$$\Re\left\{\underline{\Phi_{12,sm}}(k,l)\right\} = \Re\left\{\underline{\Phi_{12,sm}}(k,l)\right\} \circledast W(k) \tag{11}$$

Der Grad der Glättung kann hierbei durch die Länge des Fensters bestimmt werden. Bei der Faltung muss zusätzlich die Energie des Fensters E_W berücksichtigt werden um die richtige Skalierung des Ergebnisses zu garantieren.

$$E_W = \sum_{i=0}^{N_W} W(i)^2 \tag{12}$$

Um das Ergebnis in der Größenordnung der Maske vor der Faltung zu halten, wird noch eine Zusätzliche Konstante k_{Gauss} berechnet, welche anschließend mit dem Ergebnis multipliziert wird.

$$k_{Gauss} = \sqrt{\frac{\sum W(i)^2}{\sum W(i)}} \tag{13}$$

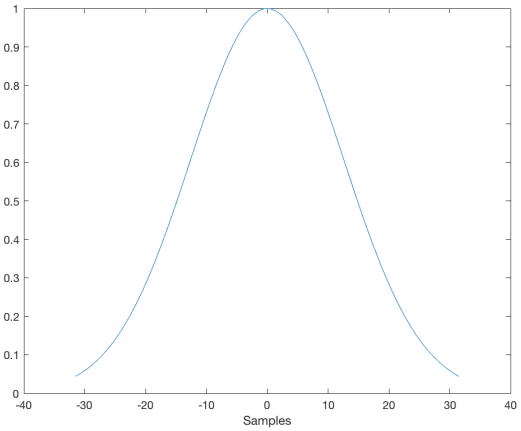

Abbildung 2 – Darstellung eines Gauss-Fensters der Länge N=64.

2.5 Zusammensetzung der Maske

Da die Verzögerungszeit ausgeglichen ist, kann davon ausgegangen werden, dass die Nutzsignale auf den Kanälen annähernd in Phase auf der reellen Achse liegen. Dadurch kann die Maske noch strenger definiert werden:

$$\underline{\Phi_{ges}}(k,l) = \Re\left\{\underline{\Phi_{12,sm}}(k,l)\right\} \cdot (1 - |\Im\left\{\underline{\Phi_{12,sm}}(k,l)\right\}|) \tag{14}$$

Zusätzlich gilt: Falls $\Re\left\{\underline{\Phi_{12,sm}}(k,l)\right\} < 0 \Longrightarrow \Re\left\{\underline{\Phi_{12,sm}}(k,l)\right\} = 0 \ \forall k$

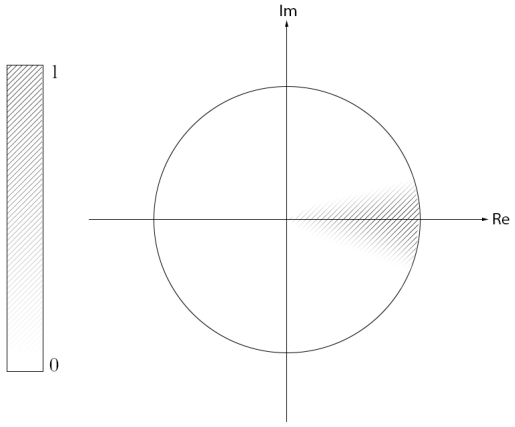

Abbildung 3 – Maskengewicht in Abhängigkeit der Lage des detektierten Phasenwinkels der Kohärenzfunktion des zeit-synchronen Eingangssignals

3 JUCE

Die im vorherigen Kapitel beschriebenen Funktionen werden nun wie folgt in JUCE als Plugin implementiert. Daher erfolgt in diesem Teil vor eine kleine Einführung der wichtigsten Bestandteile eines Audio-Plugin Projekts bevor die tatsächliche Implementierung des Plugins erläutert wird. Wenn in JUCE ein Audio-Plugin erstellt wird, enthält der automatisch generierte Prototyp des Programms die zwei folgenden Bestandteile:

1. Plugin-Processor
2. Plugin-Editor

3.1 Plugin-Processor

Der Prozessor wird beim Anlegen eines Plugins automatisch generiert und erbt von der sogenannten AudioProcessor Klasse die notwendigen Funktionen um Audio-Daten zu verarbeiten. Die zwei wichtigsten dieser Funktionen sind prepareToPlay(), und processBlock().

prepareToPlay(): Diese Funktion liefert dem Plugin die zwei wichtigsten Parameter des Audio-Buffers. Diese Parameter sind einerseits die Samplerate und andererseits die Anzahl der Samples, die der Buffer enthält. Diese Funktion wird vor dem Verarbeiten der Daten aufgerufen und sollte für etwaige Initialisierungen verwendet werden.

processBlock(): In dieser Funktion findet die tatsächliche Datenverarbeitung statt. Sie enthält als Übergabeparameter einen Zeiger auf den zu bearbeitenden Buffer($\&Buffer$). Das heißt, dass man die Daten nur für eine begrenzte Zeit zur Verfügung hat und aufpassen muss, dass exakt gleichviele Daten in den Buffer geschrieben werden, wie eingelesen wurden, da sonst in der Ausgabe unerwünschten Artefakte entstehen können. JUCE übernimmt das Speicherhandling und es kann immer auf die gleiche Adresse zugegriffen werden um die Audiodaten zu bearbeiten.

3.2 Plugin-Editor

Der Editor ist für die Erstellung und das Handling des User Interfaces zuständig. Hier werden alle graphischen Elemente, wie z.B. Textfelder, Buttons und Bilder definiert und verarbeitet. Allen Bedienelementen (Buttons, Slider, etc.) werden hier Listener zugeteilt, die dem Plugin Auskunft darüber geben, wenn sich der Zustand dieser Elemente verändert hat, damit in der Verarbeitung der Audiodaten darauf Rücksicht genommen werden kann. In unserem Fall wurde hier zusätzlich ein Timer definiert, welcher in regelmäßigen Abständen bestimmte Operationen aufruft, wie zum Beispiel das Rendern der Grafiken und das Anfordern der anzuzeigenden Werte. Zusammenfassend kann gesagt werden, dass dieser Teil des Plugins die Schnittstelle zwischen dem Benutzer und dem Plugin mit seinen Funktionen darstellt.

3.3 Bearbeitung des Buffers

In Abbildung 4 wird der Zyklus des Buffers verdeutlicht. Im implementierten Plugin wird hauptsächlich das Spektrum der Eingangsdaten über die myCoherenceCalc Klasse verarbeitet. In der process Funktion dieser Klasse wird zuerst die FFT der aktuellen Frames der Eingangskanäle berechnet.
Für die Berechnung einer FFT n'ter Ordnung werden $N = 2^n$ Samples benötigt. Deshalb muss das Programm warten bis der FFT-Buffer komplett mit Daten befüllt ist, bevor die Transformation in den Frequenzbereich durchgeführt werden kann. Anschließend wird die processFrameInBuffer() Methode aufgerufen(genaue Beschreibung in Kapitel 4) , welche die Maske berechnet, bearbeitet und auf die Eingangssignäle legt. Die bearbeiteten Signale der verschiedenen Kanäle werden rücktransformiert und mittels Overlap & Add in einen seperaten Ausgabebuffer gespeichert, welcher in weiterer Folge die richtige Anzahl an Samples in den Buffer der processBlock() Funktion des Audio Processors schreibt, worauf der Zyklus von vorne beginnt.
Es ist zu beachten, dass die Eingangsdaten vom Buffer eingelesen und nach der Bearbeitung bei der selben Adresse auch wieder abgelegt werden. Im nächsten Zyklus stehen neue Daten im Buffer, die Adresse ändert sich jedoch nicht.

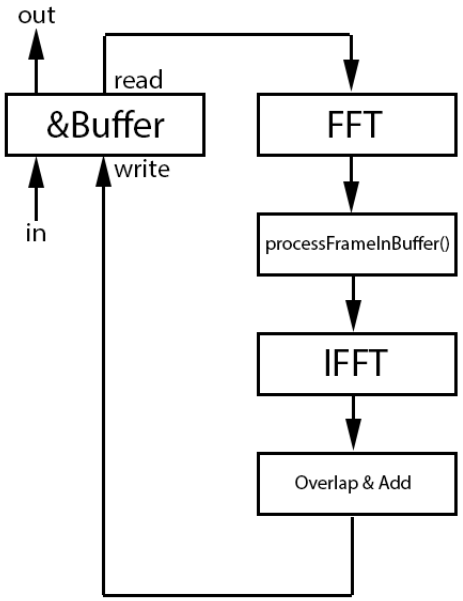

Abbildung 4 – Lese- bzw. Schreibzyklus des Ein- bzw. Ausgangsbuffers

3.4 Die Fast Fourier Transformation in JUCE

Wenn mithilfe des performRealOnlyForwardTransform Befehls eine FFT berechnet wird, muss darauf geachtet werden, wie die Daten vor dem Speichern formatiert werden. JUCE verwendet ein Interleaved Format (siehe Abbildung 5), in welchem die Real- und Imaginärteile der komplexen FFT Koeffizienten aller Frequenzen abwechselnd nacheinander in ein Array geschrieben werden. Es ist zu beachten, dass beim Berechnen einer N-Punkte FFT deshalb ein Array mit 2N Speicherstellen benötigt wird.

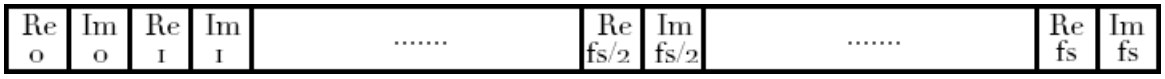

Abbildung 5 – Format der FFT Koeffizienten in JUCE

Um mit diesem Format die notwendigen Berechnungen durchzuführen, werden einige mathematische Funktionen definiert:

1. myComplexMultiplication():
   Berechnet die komplexe Multiplikation.
2. myComplexDivision():
   Berechnet die komplexe Division.
3. CalcAmpSpectrum():
   Berechnet das Amplitudenspektrum.
4. Seperate():
   Schreibt entweder Real- oder Imaginärteil in einen seperates Array.
5. Accumulate():
   kopiert Real- bzw Imaginärteil in ein Interleaved-Array.

4 Die processFrameInBuffer() Funktion

In diesem Abschnitt wird die Implementierung der wichtigsten Funktionen innerhalb der processFrameInBuffer() Routine erläutert, welche das eigentliche Herzstück des Plugins darstellt. In den verwendeten Strukturbildern sind Variablen und Größen, die mit dem User-Interface steuerbar sind unterstrichen.

4.1 Strukturbild

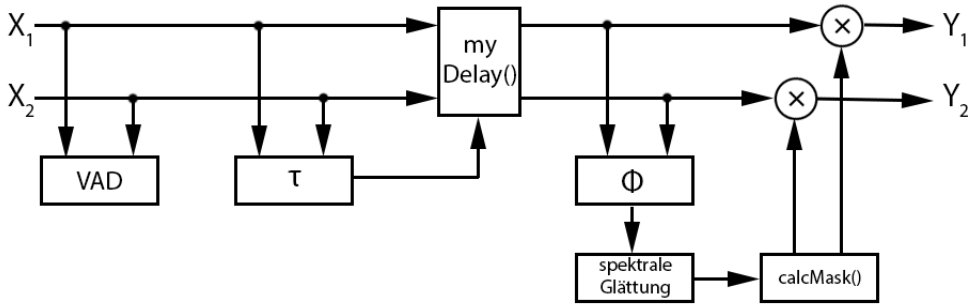

Abbildung 6 – vereinfachtes Strukturbild der gesamten Berechnung

In Abbildung 6 ist die Struktur von processFrameInBuffer() veranschaulicht. Zuerst wird die Voice-Activity-Detection berechnet, dann die Zeitverzögerung abgeschätzt. Mit letzterer wird durch die myDelay() Funktion jeweils der voreilende Kanal verzögert und damit das gleichzeitige Eintreffen der Sprachsignalen gewährleistet, bevor die Maske Φ berechnet und geglättet wird. Abschließend wird in calcMask() die Maske mit den Einschränkungen aus Gleichung 14 gefiltert, bevor sie letztendlich auf die Eingangskanäle aufmultipliziert wird.

4.2 zeitliche Glättung

Die zeitliche Glättung wird in der Funktion smoothOverTime() realisiert. Sie glättet das Signal mit der in Gleichung 10 definierten Formel mithilfe eines Faktors fgf. Da nicht nur die Maske, sondern auch andere Werte auf diese Art geglättet werden, ist in Abbildung 7 der Eingang als allgemeine Größe zu betrachten.

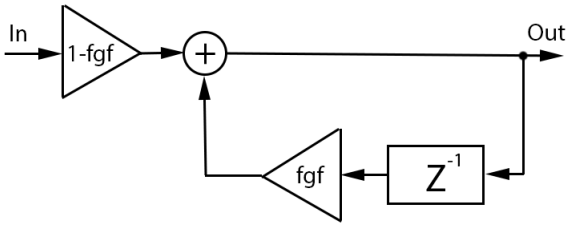

Abbildung 7 – Schematische Darstellung der zeitlichen Glättung

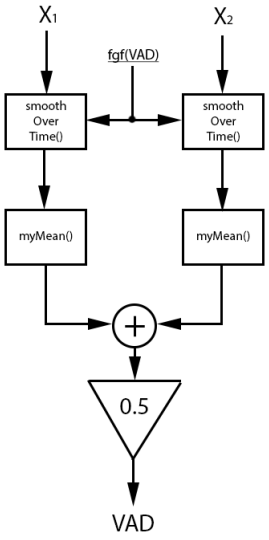

Abbildung 8 – Berechnung der Voice-Activity-Detection

4.3 Voice-Activity-Detection

Für die Berechnung der VAD werden die Eingangskanäle mit der smoothOverTime() Funktion geglättet (siehe Abbildung 8). Die myMean() Funktion quadriert erst den Eingang, mittelt ihn, und zieht abschließend die Wurzel des Mittelwerts. Der VAD-Wert kann als Mittelung der spektralen Kanalmittelwerte betrachtet werden und gibt Auskunft über die Energie in beiden Kanälen.

4.4 Zeitverzögerung τ

Der Aufbau dieses Programmteils wird in Abbildung 9 dargestellt. Einer der Eingangskanäle wird mit der konjugiert Komplexen des zweiten Kanals mithilfe von myComplexMultiplication() multipliziert, wodurch sich P_{12} ergibt (siehe Gleichung 4). Dem Ergebnis wird ein Fenster aufgeprägt, welches ihre Stützstellen aus dem tauWindowParams Array ausliest, um die Berechnung der Verzögerungszeit auf den Bereich zu begrenzen indem das Nutzsignal vermutet wird. Die resultierende Größe wird über die IFFT zurück in den Zeitbereich transformiert und um N/2 geshiftet um die Berechnung negativer Werte zu vereinfachen, da sie sich statt in der Mitte, sonst am Ende des Arrays befinden würden. Nun wird innerhalb der myTauCalc() Funktion der Index des Maximums ermittelt und anhand einer Parabolinterpolation auf sub-Samples genau approximiert. Mit der bekannten Verzögerungszeit, kann abhängig vom Vorzeichen einer der beiden Kanäle zeitlich angeglichen werden. Die zeitliche Verzögerung erfolgt wie in Gleichung 8 beschrieben.

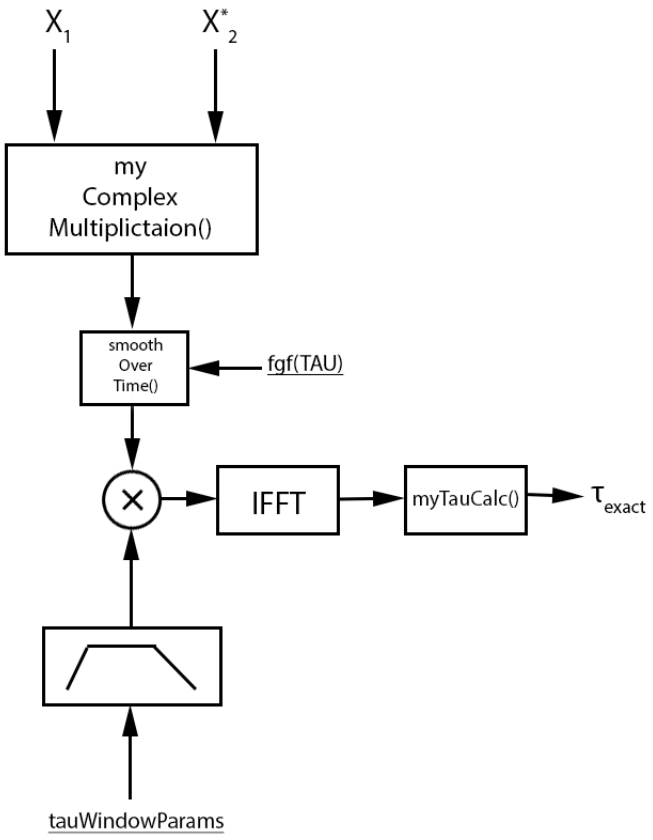

Abbildung 9 – Ermittlung der Zeitverzögerung zwischen den Kanälen

4.5 Maske Φ

Die Maske wird wie in Gleichung 5 beschrieben durch eine komplexe Division berechnet. Um eine Division durch 0 der Imaginärteile zu vermeiden wurde der Realteil des Nenners in den Imaginärteil kopiert, wobei auf die richtige Skalierung des Ergebnisses geachtet wurde. Zuätzlich werden P_{xx}, P_{yy} und P_{xy} mit dem Faktor fgf(PHI) geglättet.

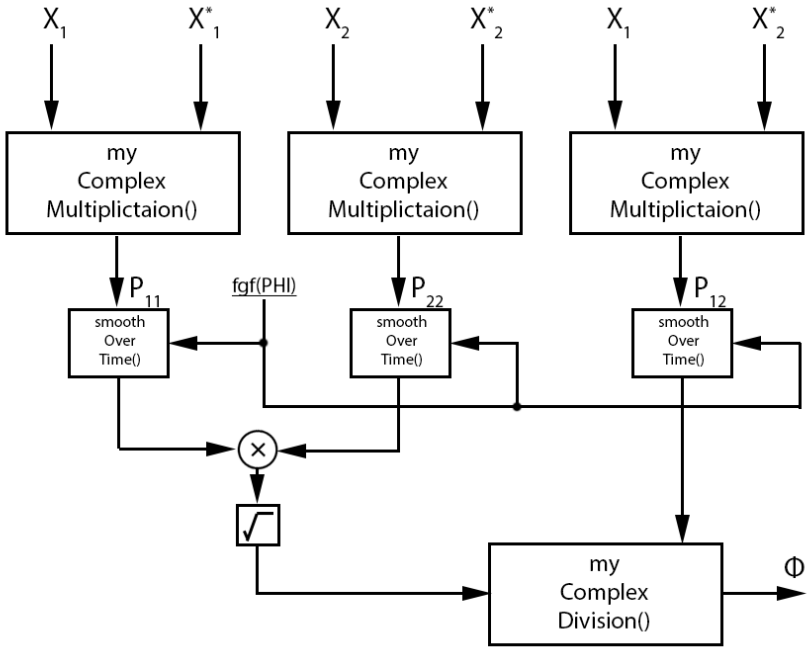

Abbildung 10 – Berechnung der Maske Φ

4.6 Spektrale Glättung

Um die spektrale Glättung durchzuführen, muss erst ein Gauss-Fenster mit der gaussWindow() Funktion erstellt werden. Sie liefert abhängig von der gewünschten Länge neben der Fensterfunktion auch die Fensterenergie. Die Fouriertransformierte des Fensters wird durch diesen Wert skaliert und mit der FFT des Quardats des Realteils der Maske multipliziert. Mit dem Fenster und seiner Länge wird in myConstCalc() zusätzlich ein Normierungsfaktor berechnet, um das rücktransformierte Faltungsergbnis der Eingangsmaskezwischen 0 und 1 zu halten.
Anschließend wird das Ergebnis durch die inverse FFT rücktransformiert. Diese Vorgehensweise greift somit auf die zirkuläre Faltung zu um die Glättung im Frequenzbereich zu realisieren.

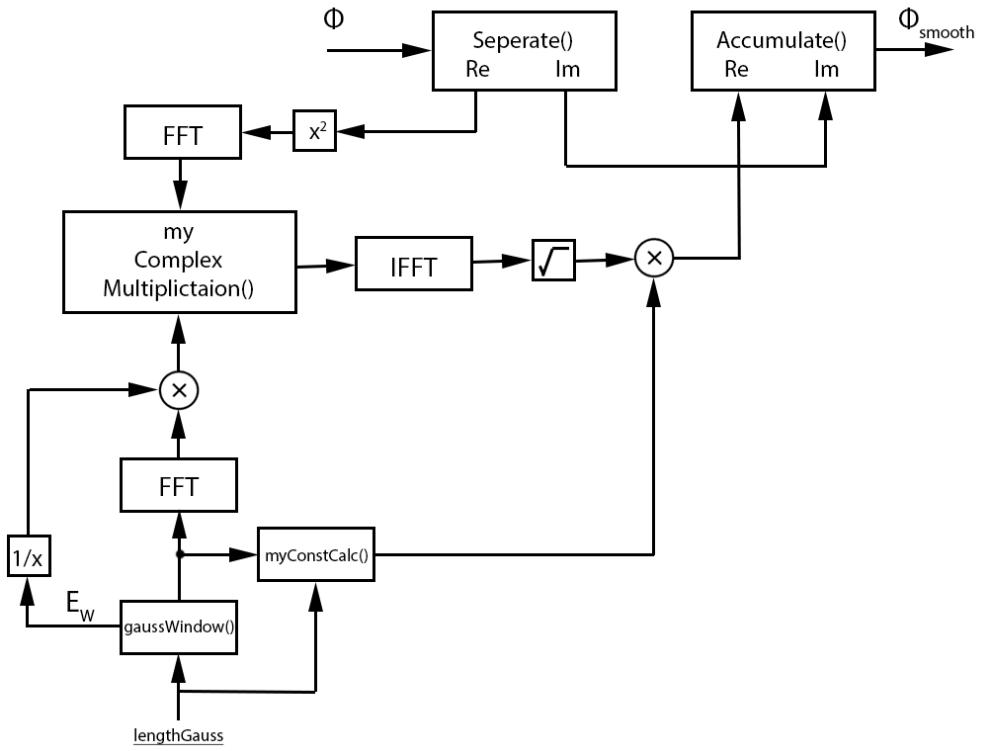

Abbildung 11 – die spektrale Glättung der Maske durch Faltung mit einem Gauss-Fenster

5 SpEx Plugin

Der Name des Plugins steht für Speech Extraction und lässt sich in jeder DAW verwenden, die VST-Plugins unterstützt. Es wurde mit JUCE v.5.3.2 erstellt und mit macOS High Sierra v.10.13.4 getestet.

5.1 Oberfläche und Bedienung

In Abbildung 12 ist die Bedienoberfläche dargestellt. Die einzelnen Bereiche sind der Projektstruktur nachempfunden.

Abschnitt 1: Dieser Bereich enthält alle Möglichkeiten für das Aufbringen der Maske. Hier kann die Maske auch invertiert angewandt werden um die Residuen zu überprüfen. Zuletzt kann entweder die Mono- oder Stereoausgabe des maskierten Signals aktiviert werden.

Abschnitt 2: Hier wird das Fenster für die Begrenzung des Auswertungsbereichs der Zeitverzögerungsabschätzung und der Maske definiert. Die 4 Textfelder stehen für die Stützstellen des Fensters.

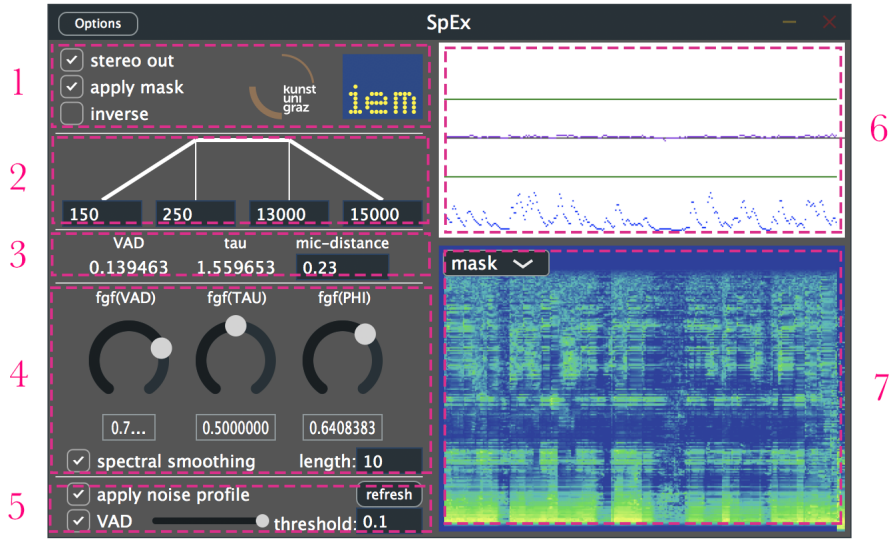

Abbildung 12 – die Bedienoberfläche im Überblick

Abschnitt 3: Im dritten Bereich werden die Werte der VAD, Zeitverzögerung und des Mikrofonabstands angezeigt, wobei Letzterer einstellbar ist.

Abschnitt 4: Hier sind alle Glättungsparameter untergebracht. Die 3 Drehknöpfe steuern die verwendeten zeitlichen Glättungen, der ToggleButton aktiviert die Spektrale Glättung. Darüber hinaus, kann im Textfeld die Länge des Gauß-Fensters definiert werden.

Abschnitt 5: In diesem Bereich wird das Rausch-Profil aktiviert und eingelesen. Zusätzlich ist eine primitive Steuerung des Profils über die VAD implementiert (siehe Ausblick).

Abschnitt 6: Im Ersten der beiden Diagramme(rechts oben) wird die Verzögerungszeit (lila) und die VAD(blau) dargestellt. Zusätzlich sind die Grenzen der Verzögerungszeit, welche sich durch den Mikrofonabstand ergeben in grün veranschaulicht.

Abschnitt 7: in der zweiten Grafik(rechts unten) werden alle spektralen Größen dargestellt. In einem Dropdown-Menü können verschiedene Spektren(Maske, In, Out, P_{12}, P_{12} bandbegrenzt) ausgewählt werden.

6 Schlussfolgerung und Ausblick

Das in dieser Arbeit erstellte und beschriebene Plugin funktioniert, hat jedoch einige Schwächen. Es ist es sehr rechenintensiv, was einerseits darauf zurückzuführen ist, dass der Algorithmus ohne viel Rücksicht auf Performance implementiert wurde, andererseits wurde für die Darstellung der Diagramme nicht OpenGL, sondern die JUCE-eigene paint() Funktion verwendet, welche hauptsächlich für die hohe CPU Auslastung zuständig ist. Weiters ist die Art wie der Plugin-Editor und Plugin-Prozessor zusammenarbeiten, nicht optimal umgesetzt. Die steuerbaren Variablen könnten mithilfe eines AudioProcessorValueTreeState besser referenziert werden, anstatt sie mit vielen set/get Methoden zu verarbeiten.Ein weiteres Problem ist, dass die gefilterte Stimme oft sehr tonal und dumpf klingt, und in den Residuen oft in etwa gleich gut verständlich ist. Mit den Testdaten, die mir zur Verfügung standen, ist mir auch aufgefallen, dass das tieffrequente Rauschen auch mehr in der Maske enthalten ist als in den Residuen, da es relativ hochenergetisch im Vergleich zum Sprachsignal ist. Um diesem Problem entgegenzuarbeiten, habe ich gegen Ende der Arbeit noch probiert ein simples Rauschprofil zu implementieren, welches recht gute Ergebnisse in Sachen Rauschunterdrückung lieferte, das Sprachsignal jedoch auch etwas beeinträchtigte. Zusammenfassend funktioniert das Plugin gut, wenn man genug Zeit hat alle Einstellungen perfekt an das Signal anzupassen, es ist jedoch meiner Meinung nach nicht ausreichend flexibel und nicht auf jedes Signal anwendbar.
Für die eventuelle Fortsetzung dieses Projekts wird vorgeschlagen die Voice-Activity-Detection mehr in manche Glättungen/Parametersteuerungen einfließen zu lassen, da sie an sich sehr gut funktioniert. Sie ist beim jetzigen Stand des Plugins nur in der Erfassung des Rauschprofils im Einsatz. Wie vorher erwähnt sollte auch an der Performance gearbeitet werden. Um das Problem der Dämpfung der Zischlaute/höherfrequenten Anteile zu umgehen, könnte versucht werden die Transienten der Maske zusätzlich hervorzuheben. Die Implementation eines Rauschprofils, scheint bei dieser Anwendung auch sehr nützlich zu sein, falls es in Passagen in denen keine Stimme detektiert wird, automatisch aktualisiert bzw. geglättet wird. Das Plugin könnte noch mit der PESQ-Methode(perceptual evaluation of speech quality) bewertet werden, um die bestmöglichen Einstellungen zu evaluieren. Abschließend sollte noch die Verarbeitung mehrkanäliger Datensätze ermöglicht werden, da das Plugin in seiner jetzigen Form nur für zweikanalige Aufnahmen ausgelegt ist. Einige der Bestandteile sind aber schon für die Anwendung mit mehreren Eingangskanälen ausgelegt, jedoch bedarf es einer Logik für die Hierachie der verschiedenen Kohärenzberechnungen.

Danksagungen: Mein Dank gilt dem gesamten Institut für Elektronische Musik und vor allem meinem Betreuer Dipl.-Ing. Dr.techn. Alois Sontacchi, der sich immer Zeit nahm um mir mit meinen unzähligen Fragen und Problemen zu helfen, ohne die Geduld zu verlieren. Ganz besonders möchte ich mich auch bei Dipl.-Ing. BSc Daniel Rudrich bedanken, der mir stundenlang meine Fragen bezüglich JUCE und C++ beantwortet hat, was die Fertigstellung des Plugins in diesem Umfang überhaupt möglich gemacht hat.

Literatur

[CJM02] A. Carlos and J. Jean-Marc, "Ambience extraction and synthesis from stereo signals for mutli-channel up-mix." IEEE, 2002.

[TSBDA17] B. Tanmay, M. Sudhindu Bikash, S. Debasri, and C. Amlan, "Coherence based dual microphone speech enhancement technique using fpga," *Microprocessors and Microsystems*, vol. 55, no. 2, 2017.

JUCE-Tutorials: https://juce.com/learn/tutorials
JUCE-Dokumentation: https://docs.juce.com/master/index.html