REALISIERUNG EINES SMARTPHONE-BASIERTEN AUDITIVEN FEEDBACKS ZUR UNTERSTÜTZENDEN STOTTERBEHANDLUNG

Jürgen Hock, Stefan Feldes

Institut für Digitale Signalverarbeitung, Hochschule Mannheim

juergen.hock@gmx.com

Kurzfassung: In der Stotterbehandlung wird mitunter therapiebegleitend das auditive Feedback eingesetzt. Dabei wird die Stimme des Patienten verzögert und transponiert über Kopfhörer zurückgekoppelt. Der vorliegende Beitrag stellt ein Projekt vor, bei dem das auditive Feedback auf Android-Smartphones realisiert wurde. Die Anwendung nimmt das Audiosignal vom eingebauten Mikrofon auf, entrauscht es, verfremdet es wahlweise und gibt es mit Verzögerung am angeschlossenen Kopfhörer aus. Zur Stimmverfremdung wurden Effekte wie Transposition, Roboter- und Flüsterstimme auf einer gemeinsamen Implementierungsbasis umschaltbar realisiert. Zur Reduzierung des Grundrauschpegels wurde eine einfache Rauschunterdrückung integriert. Zur Erhaltung der aufgrund der Transposition veränderten Formantenlagen wurde ein LPC-Verfahren untersucht. Beschrieben werden hier die eingesetzten Algorithmen und deren Softwareumsetzung sowie Erkenntnisse bezüglich der Hardwareanforderungen, der vorgenommenen Laufzeitoptimierung und der API-bedingten Verzögerungszeiten bei der Audioausgabe.

1 Einleitung

In diversen empirischen Studien, u.a. [2, 3], stellten Logopäden fest, dass das Stottern durch eine besondere Art und Weise zu Sprechen reduziert werden kann, z.B. beim Singen, Flüstern oder synchronen Sprechen mit einer weiteren Person. Möglicherweise hängt das mit der Wahrnehmung der dadurch verfremdeten Eigenstimme zusammen. Ein vergleichbarer Effekt kann auch mittels *Altered Auditory Feedback* erzielt werden. Dabei wird das Sprachsignal einer Person aufgenommen und während des Sprechens verändert ans Ohr zurückgeführt. Die Verfremdung der Stimme kann dabei durch eine Funktion bzw. mehrere aneinandergereihte Funktionen erfolgen. Zu den meist eingesetzten Funktionen gehören die zeitliche Verzögerung der Ausgabe sowie die Transposition der Stimmlage.

Im Zuge der Stottertherapie werden zum auditiven Feedback bereits apparative Sprechhilfen eingesetzt – kompakte Geräte, die einen Mikrofoneingang, Kopfhörerausgang und benötigte Konfigurationsschnittstellen bereitstellen. Ein Hindernis sind jedoch u. U. die relativ hohen Anschaffungskosten. Alternativ könnte die Funktionalität auch kostengünstiger als Softwareapplikation auf einer vorhandenen Hardware realisiert werden.

Daher wurde hier im Rahmen einer Machbarkeitsstudie der Prototyp einer Software-Sprechhilfe für Android-Smartphones entwickelt. Mit der entwickelten Applikation kann der Nutzer sein Sprechen über Kopfhörer verzögert in Echtzeit mithören und auf Wunsch seine Stimme durch verschiedene Audioeffekte verfremden. Unser Ziel war, die Applikation für derzeit gängige

Geräte umzusetzen und dabei die Realisierungsgrenzen für dieses Anwendungsszenario herauszufinden.

Der Beitrag gliedert sich wie folgt: In Kap. 2 fassen wir wichtige logopädische Studienergebnisse zusammen und leiten Anforderungen für die zu entwickelnde Applikation ab. Kap. 3 beschreibt die realisierten Algorithmen zur Stimmverfremdung. Kap. 4 erläutert die Umsetzung auf das Android-Betriebssystem. Abschließend werden die Grenzen und Perspektiven diskutiert.

2 Behandlung von Stottern mit auditivem Feedback

Beim auditiven Feedback unterscheidet man zwischen *Delayed Auditory Feedback* (DAF) und *Frequency-Shifted Auditory Feedback* (FAF). Das Delayed Auditory Feedback stellt eine zeitliche Verzögerung der Ausgabe dar, wobei die Verzögerung zwischen wenigen Millisekunden bis zu einer halben Sekunde liegt. Beim Frequency-Shifted Auditory Feedback handelt es sich um eine künstliche Transposition der Stimmlage. Transponiert wird es z.B. um kleine Terz, Tritonus bis zur Oktave sowohl aufwärts als auch abwärts.

Die Wirkung von DAF und FAF in der Stotterbehandlung wurde u.a. in den Studien [2, 3] untersucht. Dabei konnte eine allgemeine stotterreduzierende Wirkung bei Verwendung von DAF mit einer Verzögerungszeit von 50 ms festgestellt werden. Der Erfolg von FAF ist dagegen nicht als pauschal einzustufen. Dennoch kann FAF in einzelnen Fällen durchaus zu größerer Stotterreduktion beitragen. Die DAF-Methode hat den Nachteil, dass es bei Verzögerungen ab 100 ms zum Sprechen mit einem charakteristischen Rhythmus kommt (sog. *DAF-Voice*). Im Gegensatz dazu verändert FAF, falls es wirkt, das Sprechen nicht auffällig. Es gibt derzeit keine Erkenntnisse über eine optimale DAF- bzw. FAF-Konfiguration. Diese kann nur individuell durch Ausprobieren herausgefunden werden. Man variiert die Parameter so, dass der Sprecher nicht belastet und die Symptomatik vermindert wird.

Laut Langzeitstudie [3] lag die Zufriedenheitsrate der Probanden bei 50%. Kritikpunkte im Falle der Unzufriedenheit waren zum einem, dass das Doppelt-Hören über Kopfhörer von einigen auf Dauer als Belastung empfunden wurde, zum anderen, beklagten einige eine unzureichende Rauschreduktion. Insbesondere bei höherer Lautstärke und bei Headsets mit im Ohrteil eingebautem Mikrofon störten mitverstärkte und verzögert wiedergegebene Umweltgeräusche.

In Auswertung dieser Studienergebnisse wurden daher für die Realisierung unseres Smartphone-basierten auditiven Feedbacks die Möglichkeit zur individuellen Konfiguration der Audioeffekte sowie eine Rauschreduktion vorgesehen.

3 Algorithmen zur Stimmverfremdung

Die Realisierung von DAF und FAF erfordert neben einem einstellbaren Delay die Implementierung von Algorithmen zur Stimmverfremdung, insb. zur Transposition. Darüber hinaus haben wir zum individuellen Nutzen wahlweise auch einen Flüster- und Roboterisierungseffekt ergänzt. Alle Effekte können nach dem Vocoderprinzip gemäß [4, 1] realisiert werden. In Abbildung 1 ist die gemeinsame Implementierungsbasis schematisch dargestellt.

Anpassungen im Spektrum des Audiosignals werden hier über die *Short-Time Fourier Transform* (STFT) ermöglicht. Hierfür wird das Signal blockweise aufgenommen, mit einer Fensterfunktion gewichtet und Fourier-transformiert. Der Vorgang erfolgt überlappend, indem sich das Analysezeitfenster stets um eine *Hop Size* genannte Anzahl von Abtastwerten H_A vorwärts schiebt. Der Audioeffekt wird durch Verarbeitung der Frequenzkomponenten nach Betrag und Phase erzielt. Das veränderte Zeitsignal wird nach Fourierrücktransformation aus den resultie-

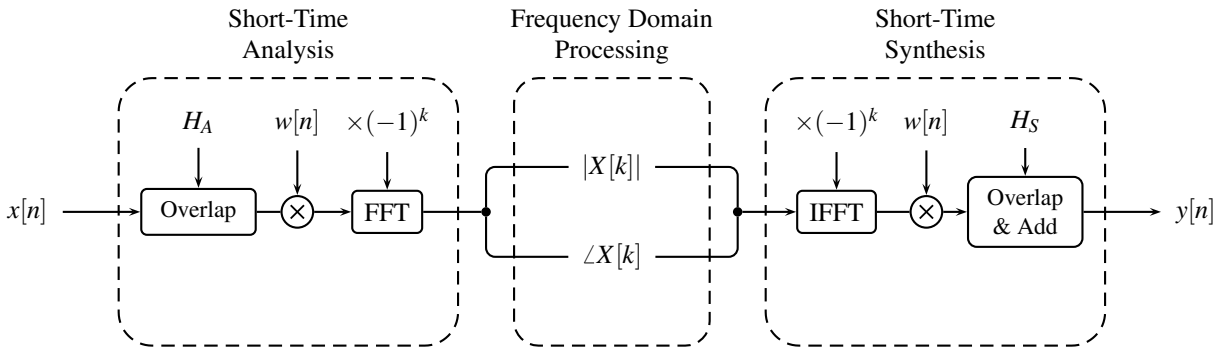


Abbildung 1 - Implementierungsbasis des Vocoders.

renden Signalblöcken durch Überlagerung mit ggf. anderer Hop Size H_S und Addition gewonnen.

Die Signalanalyse und -synthese mittels STFT muss geeignet parametrisiert werden. Bei einer Abtastrate von 44100 Hz verwenden wir ein Hann-Fenster, dessen Größe je nach Anforderung des jeweiligen Effekts 2048, 1024 und 512 Abtastwerte beträgt. Die Hop Size H_A und H_S liegt dann gewöhnlich bei $\frac{1}{4}$ der verwendeten Fenstergröße.

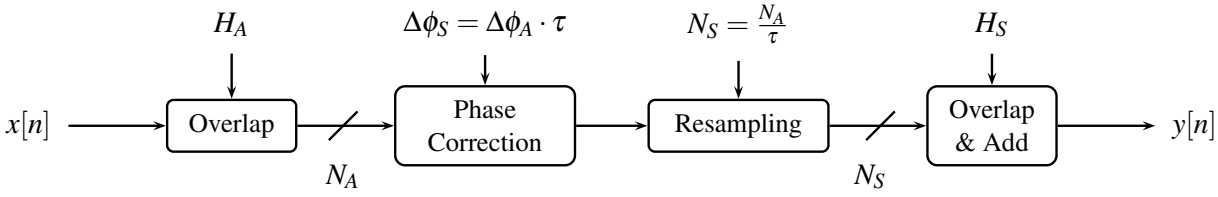
3.1 Transposition

Durch den Transpositionseffekt kann die Stimmgrundfrequenz des Sprechers erhöht oder erniedrigt werden. Die Festlegung des Transpositionsintervalls erfolgt in Halbtonschritten von -12 (Oktave nach unten) bis 12 (Oktave nach oben). Diese für den Endbenutzer intuitive Angabe überführen wir gemäß Gl. 1 in den Transpositionsfaktor τ. Der Transpositionsfaktor gibt die Skalierung der im Signal enthaltenen Frequenzkomponenten an und liegt im Wertebereich von 0,5 (Frequenzhalbierung) bis 2 (Frequenzverdoppelung).

$$\tau = 2^{S/12} \mid_{S=\{-12..12\}} \tag{1}$$

Die Transposition als solche kann schon durch Neuabtastung (oder auch Interpolation) des Signals hervorgerufen werden. Wird mit erhöhter Abtastfrequenz wiedergegeben, so erhöht sich auch die Tonhöhe des Audiosignals und umgekehrt. Eine geänderte Abtastfrequenz führt jedoch auch zur Verkürzung oder Verlängerung der Wiedergabezeit. Um das zu verhindern, sollte zuvor die Signaldauer antiproportional zur Neuabtastung künstlich verkürzt oder verlängert werden. Dieses Vorgehen bezeichnet man auch als *Time Stretching*.

Im Zuge der Echtzeitverarbeitung führen wir diese zwei Schritte mittels Vocoder wie in Abbildung 2 dargestellt für jeden Signalblock kombiniert aus. Dabei sind die FFT-Prozeduren in der Abbildung der Übersichtlichkeit halber ausgespart. Die Phasenanpassung des Time Stretchings erfolgt im Frequenzbereich und die Neuabtastung durch lineare Interpolation im Zeitbereich.



Festzulegende Parameter: $\tau = \{0,5..2\}$ und N_A = Analyse-Fenstergröße.

Abbildung 2 - Schematische Darstellung des Ablaufs zur Transposition.

Um die Zeitdauer eines Audiosignals grundsätzlich zu verkürzen oder zu verlängern, stellt man unterschiedliche Fensterverschiebungen $H_A = \frac{H_S}{\tau}$ für Analyse und Synthese ein. Ist die Fensterverschiebung im Synthese-Schritt größer als die im Analyse-Schritt, wird die gesamte Signaldauer verlängert und umgekehrt. Jedoch werden somit Phaseninterferenzen in den sich überlappenden Signalabschnitten im Synthese-Schritt verursacht. Mit einer Phasenkorrektur gemäß Theorem 2 gelingt es einen Signalabschnitt beliebig korrekt zu verschieben.

$$x[n-\Delta H] \;\circ\!\!-\!\!\bullet\; X[k]\cdot e^{-\Omega_k \Delta H} \tag{2}$$

Jedoch kann dieses Theorem nicht direkt für die Echtzeitverarbeitung eingesetzt werden, da Signalblöcke fortlaufend um $\Delta H = (H_S - H_A)\cdot i$ Abtastwerte versetzt werden sollen und die Zahl der Versetzungsschritte i zu einem beliebigen Zeitpunkt unbekannt ist. Die Lösungsidee nach [4, 1] besteht darin, die Differenz der Phasen zweier Analyseblöcke zu ermitteln, um daraus die notwendige Distanz zum nächsten Synthese-Block zu bestimmen.

Dazu betrachtet man zuerst die aktuelle Phase ϕ_A und die im vorhergehenden Analyse-Schritt ermittelte Phase ϕ_{A-1} für jede Frequenzkomponente. Die Differenz beider Analyse-Phasen ist durch Gl. 3 gegeben:

$$\Delta\phi_A = [\phi_A - (\phi_{A-1} + \Omega_k H_A)]^{+\pi}_{-\pi} \tag{3}$$

Diese Phasendifferenz kann nun je nach Verhältnis $\tau = \frac{H_S}{H_A}$ wie folgt skaliert werden:

$$\Delta\phi_S = \Delta\phi_A \cdot \tau \tag{4}$$

Die aktuelle Synthese-Phase ϕ_S des zu verschiebenden Signalblocks errechnet sich aus der vorhergehendem Wert ϕ_{S-1} gemäß Gl. 5:

$$\phi_S = [(\phi_{S-1} + \Omega_k H_S) + \Delta\phi_S]^{+\pi}_{-\pi} \tag{5}$$

Bei Ω_k handelt es sich um die normierte Kreisfrequenz, die für die k-te von N Frequenzkomponenten wie folgt bestimmt wird:

$$\Omega_k = \frac{2\pi k}{N} \tag{6}$$

Das Symbol $[\,]^{+\pi}_{-\pi}$ steht für die *Principal Phase Argument* Funktion. Damit wird der fortlaufende Phasenwert auf den Wertebereich von $-\pi$ bis π wie in Gl. 7 angegeben normiert:

$$\phi_{out} = mod\,(\phi_{in} + \pi, 2\pi) - \pi \tag{7}$$

Die Notwendigkeit eine der beiden Phasenwerte ϕ_X oder ϕ_{X-1} umzurechnen, ergibt sich aus der Tatsache, dass zu jedem Zeitpunkt lediglich der momentane Wert der Phase (sog. *Instantaneous Phase*) im Bereich von $-\pi$ bis π vorliegt. Durch die Addition von $\Omega_k H_X$ wird hier der vergangene Phasenwert ϕ_{X-1} mit der fehlenden Distanz bis zum momentanen Zeitpunkt ϕ_X erweitert, sodass die Phasendifferenz korrekt bestimmt werden kann.

Durch anschließende Neuabtastung verändert sich die Länge der Signalblöcke von ursprünglichen N_A Abtastwerten auf $N_S = \frac{N_A}{\tau}$. Da Time Stretching und Neuabtastung für den Echtzeitbetrieb nicht sequentiell, sondern verschachtelt im selben Verarbeitungsschritt erfolgen, muss die tatsächliche Synthese- der Analyse-Verschiebungsgröße entsprechen, damit die Gesamtdauer des Signals erhalten bleibt.

Für diesen Audioeffekt verwendeten wir die Fenstergröße $N_A = 2048$ und die dem Hann-Fenster entsprechende Fensterverschiebung $H_S = H_A = N_A \cdot \frac{1}{4}$.

3.2 Roboter- und Flüsterstimme

Zur Erzeugung der Roboter- und Flüsterstimme werden nach [4] ausschließlich die Phasen aller Frequenzkomponenten verändert. Deren Beträge bleiben erhalten. Die Fensterverschiebungen H_A und H_S werden ebenfalls gleich eingestellt.

Die intonationslose Roboterstimme entsteht, wenn die Phasen auf einen konstanten Wert, z.B. Null, gesetzt werden. Dadurch wird dem Audiosignal eine künstliche Periodizität von $H_A = H_S$ Abtastwerten beigefügt. So sinkt oder erhöht sich die Gesamttonhöhe auf eine neue Grundfrequenz. Die künstliche Grundfrequenz ergibt sich aus dem Verhältnis der Abtastrate zur Fensterverschiebung, also F_s/H_A. Bei einer Fenstergröße von 1024 Abtastwerten zeigt der Effekt eine deutliche Wirkung.

Im Falle der Flüsterstimme generiert man für jede Frequenzkomponente einen zufälligen Phasenwert im Wertebereich von $-\pi$ bis π. Der Effekt konnte deutlich bei der Fenstergröße von 512 Abtastwerten beobachtet werden.

3.3 Formantenerhaltung

Um die durch die Transposition verursachte Änderung der Formantenlage zu vermeiden, kann die LPC-Technik eingesetzt werden, siehe Abbildung 3. Dabei wird durch eine blockweise inverse Filterung noch vor dem Analyse-Schritt die spektrale Einhüllende eliminiert. So wird nur das Residuum durch den Vocoder verarbeitet. Abschließend werden die extrahierten Formanten durch eine reziproke Filterung wieder einprägt.

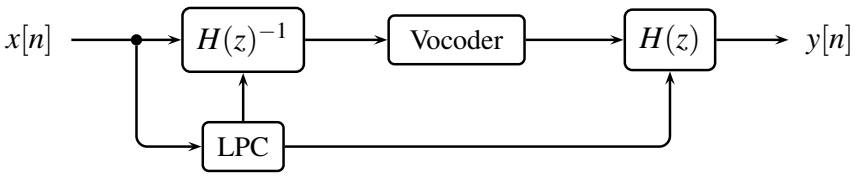


Abbildung 3 - Prinzip des Transpositionsverfahrens mit Formantenerhaltung.

Die Koeffizienten des Filters $H(z)$ müssen stets aus dem Quellsignal bestimmt werden. Hierfür arbeiten wir mit Blöcken von 1024 bis 2048 Abtastwerten, was bei einer Abtastfrequenz von 44100 Hz etwa 23 ms bis 46 ms entspricht. Die Filterordnung legten wir auf 16 Koeffizienten fest.

Durch die beschriebene Formantenerhaltung konnte bei Transposition der sog. *Mickey-Mouse-Effekt* verringert werden. Es kann auch bei anderen Effekten angewandt werden, um z.B. trotz Verfremdung die persönliche Intonation der Stimme beizubehalten.

3.4 Rauschunterdrückung

Zur Unterdrückung des Grundrauschpegels, der z.B. am Mikrofon entsteht, setzen wir die *Non-Linear Spectral Subtraction* Technik [4] ein. Die Idee besteht darin, Beträge der Frequenzkomponenten so zu skalieren, dass relativ große Werte erhalten bleiben und relativ kleine unterdrückt werden. Die Phasen der Frequenzkomponenten bleiben dabei unmodifiziert.

Für jede Frequenzkomponente ist eine Normalisierung des Betrags wie folgt durchzuführen:

$$r = \frac{|X[k]|}{N} \tag{8}$$

Der normalisierte Betrag kann nun durch eine beliebige nicht lineare *Noise Gate* Funktion $f(r)$ verarbeitet werden. Durch die anschließende Multiplikation mit der jeweiligen Frequenzkomponente wird diese je nach Betragsgröße gemäß Gl. 9 skaliert:

$$X[k] = X[k] \cdot f(r) \tag{9}$$

Zur Skalierung der Frequenzbeträge verwendeten wir die folgende Funktion mit *const* = 0,01:

$$f(r) = \frac{r}{r + const} \tag{10}$$

Mit dem beschriebenen Verfahren konnte der Grundrauschpegel effektiv unterdrückt werden, wobei allerdings auch die Stimme des Sprechers leicht abgestumpft wurde.

4 Realisierung auf Android-Smartphones

4.1 Audio Ein- und Ausgabe

Die Android-Plattform stellt grundsätzlich zweierlei API zur Verfügung: die im Android Software Development Kit (SDK) enthaltene Java- und die im Android Native Development Kit (NDK) enthaltene C/C++-Programmierumgebung. Zur allgemeinen Implementierung von Applikationen ist die umfangreiche Java API die erste Wahl. Die native C/C++ API dient in der Regel dazu, zeitkritische Routinen effizienter zu gestalten. Doch im Gegensatz zur Java API sind hier die Möglichkeiten zum Debuggen der Applikation eingeschränkt und die Implementierung ist insgesamt aufwändiger.

Verantwortlich für die Audio Ein- und Ausgabe ist der betriebssysteminterne *AudioFlinger*-Dienst. Dieser interagiert über die *Hardware Abstraction Layer* (HAL) direkt mit der Hardware. Bei HAL handelt es sich um den vom jeweiligen Hardware-Hersteller abhängigen Audiotreiber. Der Zugriff auf den AudioFlinger-Dienst erfolgt über die Klassen *AudioRecord* und *AudioTrack*, die das blockweise Einlesen bzw. Ausgeben von PCM-codierten Audiosignal ermöglichen.

Zur Instanziierung dieser Klassen benötigt man einige Parameter:

- Mikrofon als Quelle bzw. Lautprecher als Senke,
- Anzahl der Tonkanäle (Mono oder Stereo),
- Abtastfrequenz,
- Quantisierung und
- Größe des internen I/O-Puffers.

Dabei kommt es darauf an, welche Einstellungen vom jeweiligen Smartphone unterstützt werden. Es gibt derzeit keine standardisierte API, um optimale Einstellungen im Einzelfall zu bestimmen. Die Festlegung der Einstellungen muss derzeit nach dem Trial-Error-Prinzip erfolgen. Bei Realisierung der Anwendung stellten wir fest, dass die gängigen Geräte eine Abtastrate von 44100 Hz, 16 bit Quantisierung sowie Monokanal für die Ein- und Ausgabe unterstützten. Die minimale Größe der internen Puffer wird durch betriebssysteminterne Funktionen aus den vorgegebenen Konfigurationen bestimmt und muss auf mindestens diesen Wert eingestellt werden.

4.2 Verzögerung

Im Wesentlichen entsteht die Verzögerung des auditorischen Feedbacks durch die eingesetzte Overlap-Add-Routine und das Betriebssystem.

Die Overlap-Add-Routine basiert auf der Verrechnung von zwei Signalblöcken. Das Analysefenster (hier *Hann*) muss dabei schrittweise um $\frac{1}{4}$ der Fenstergröße verschoben werden, um eine Amplitudenmodulation des Signals beim Overlap-Add zu vermeiden. Daher kann die DA-Wandlung frühestens nach der in Abbildung 4 dargestellten Zeitspanne von $1\frac{3}{4}$ Blöcken beginnen.

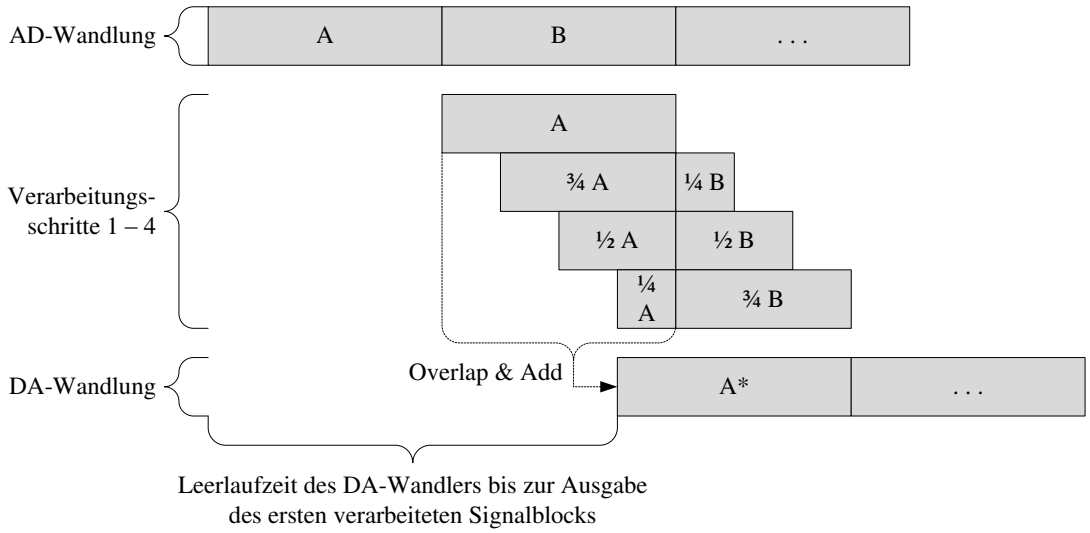


Abbildung 4 - Verzögerung durch die Overlap-Add-Routine.

Eine weitere Verzögerung des Signals kommt durch das Betriebssystem zustande. Zum einem handelt es sich um einen Multitasking-Betrieb, sodass der gesamte Verarbeitungsprozess potenziell verdrängt und erst verspätet ausgeführt werden kann. Die Verdrängung lässt sich nur begrenzt durch eine höhere Priorisierung aufhalten, jedoch nicht ganz ausschließen. Durch die eingeführte Laufzeitoptimierung konnten wir das Risiko der verspäteten Verarbeitung besonders auf älteren Geräten minimieren.

Zum anderen stellten wir eine Besonderheit der API fest. Der Ausgangspuffer muss mit ausreichend vielen Abtastwerten befüllt werden, bevor die DA-Wandlung beginnen kann. Das führt zwangsläufig zu einer recht großen Verzögerung bei der Signalausgabe. Die Anzahl der benötigten Abtastwerte hängt wiederum mit der minimalen Puffergröße zusammen – eine Größe, die vom Smartphone-Hersteller je nach eingebauter Hardware ermittelt und in Gerätetreibern festgelegt wird. Konkrete Informationen darüber werden von den Smartphone-Herstellern nicht preisgegeben.

Insgesamt ergaben die Overlap-Add-Routine und die betriebssystembedingte Pufferverzögerung bei unseren Tests eine Verzögerungszeit von 140 ms bis 200 ms je nach Gerät.

4.3 Laufzeitoptimierung

Zur Laufzeitoptimierung von zeitkritischen Programmabschnitten haben wir die *Java Native Interface* (JNI) Programmierschnittstelle eingesetzt. JNI gehört zum festen Bestandteil der Java-Laufzeitumgebung auf jedem Android-Smartphone. So lassen sich Funktionsaufrufe definieren, die abseits der *Java Virtual Machine* ausgeführt werden. Die Implementierung solcher Funktionen erfolgt in C/C++, wobei eine Möglichkeit Variablen zwischen Java und C/C++ Umgebungen auszutauschen gegeben ist. Das gilt sowohl für primitive als auch für referenzierte Datentypen. Schon allein durch die Auslagerung zeitkritischer Abschnitte kommt es zur Verkürzung der Ausführungszeit. Zudem können beim Kompilieren zusätzliche Optimierungsflags gesetzt werden. So änderten wir den Standardbefehlsatz *thumb* auf *arm*, wodurch die Ausführungszeit weiterhin verkürzt wurde.

Optimiert haben wir auf diese Weise insbesondere die FFT-Routine. Für eine Blockgröße von 2048 *float*-Werten konnte so eine vierfache Reduzierung der Rechenzeit erreicht werden.

5 Fazit

Das angestrebte auditive Feedback mit verschiedenen einstellbaren Stimmverfremdungseffekten konnte als geräteunabhängige Applikation für das Android-Betriebssystem realisiert werden. Im Hinblick auf die große und zunehmende Verbreitung Android-basierter Smartphones bietet sich somit für viele potentielle Nutzer eine kostengünstige Möglichkeit zum stotterreduzierenden Training.

Die benötigten Hardwarekomponenten sind auf jedem Gerät vorhanden und über eine einheitliche API von der Applikation nutzbar. Aufgrund herstellerbedingter Diversität kann die Qualität der Anwendung vom Gerät zu Gerät jedoch variieren. Durch eine Laufzeitoptimierung mit Hilfe von nativem Code konnten wir jedoch auch auf älteren Testgeräten Echtzeitfähigkeit erreichen. Nach unseren Testergebnissen sind Smartphones mit Prozessoren ab 800 MHz Taktfrequenz für unsere Anwendung geeignet. Da neue Smartphones bereits heute mit deutlich leistungsfähigeren Doppelkernprozessoren ausgestattet werden, gehen wir davon aus, dass die Performanz künftig kein entscheidendes Hindernis in der Entwicklung von Applikationen solcher Art sein wird.

Das größere Hindernis liegt noch in den herstellerspezifischen Hardware-Treibern zum Audio I/O. Die auf den getesteten Geräten festgestellte minimale Verzögerung von 140 ms bis 200 ms liegt deutlich höher als die Vorgaben der Logopäden. Eine Verbesserung könnte ggf. durch die neuartige Audio-API *OpenSL ES* erzielt werden. Damit versuchen die „Android-Macher“ strikte Vorgaben, insbesondere bzgl. Latenzzeiten, einzuführen. Allerdings ist diese API erst ab höheren Betriebssystemversionen verfügbar und nach Aussagen von einigen Android-Entwicklern erfüllen die Smartphone-Hersteller nicht alle gestellten Anforderungen.

Bezüglich weiterer Anwendungsszenarien sehen wir die Möglichkeit, DAF bzw. FAF während eines Telefongesprächs zu nutzen, was im Falle einer Smartphone-Applikation sehr naheliegend wäre. Es muss also untersucht werden, inwiefern die Stimmverfremdung mit Telefonie-APIs kombiniert werden kann. Eine anderweitige Nutzung von Stimmverfremdungseffekten, z.B. zur Unterhaltung, bleibt dabei dem Nutzer überlassen.

sich verarbeiten.

Literatur

[1] DE GÖTZEN, A., N. BERNARDINI und D. ARFIB: *Traditional (?) Implementations of a Phase-Vocoder : the Tricks of the Trade*. Proceedings of the COST G6 Conference on Digital Audio Effects DAFX00, S. 1–7, 2000.

[2] NATKE, U.: *Stotterreduktion bei frequenzverschobener und verzögerter auditiver Rückmeldung*. Folia Phoniatrica et Logopaedica, 52(4):151–159, 2000.

[3] PÄDAGOGISCHE HOCHSCHULE HEIDELBERG: *Technisch unterstützte Reduktion des Stotterns (TURS)*, 2011.

[4] ZÖLZER, U.: *DAFX – Digital Audio Effects (Second Edition)*. John Wiley & Sons, 2011.