Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

ZSN Zentrum für Signalverarbeitung
und Nachrichtentechnik

Projektarbeit (Informatik)

Erweiterung einer Audio Signal Processing Toolbox für Android

Autoren	Matthias Kaderli Ravivarnen Sivasothilingam
Betreuung	Dr. Martin Loeser

Datum	20.12.2017

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

……

Unterschriften:

……

……

Inhaltsverzeichnis

Abstract 2
Theorie 2
Zielsetzung 2
Ausgangslage 3
Theorie und verwendete Technologien 4
Beat-Detection 4
Metronom Generierung 9
Implementierung 13
Vorgehen / Methoden 13
Algorithmen Test Set 13
Untersuchte Algorithmen 14
Implementierter Algorithmus in Android 25
Resultate 29
Beat-Detection 29
Metronom Generierung 29
Diskussion und Ausblick 30
Resultate und Zielsetzung 30
Beat-Detection 30
Metronom Generierung 30
Mögliche Erweiterungen der Resultate 31
Allgemein 31
Beat-Detection 31
Metronom Generierung 31
Schlusswort 32
Verzeichnisse 33
Literaturverzeichnis 33
Abbildungsverzeichnis 34
Tabellenverzeichnis 34
Anhang 35
Glossar 35
Offizielle Aufgabenstellung 36

Abstract

Das Praktizieren eines Musikstücks benötigt viel Übung. Diese wird häufig mithilfe eines Metronoms gemacht. Je nach Bekanntheit des Stücks kennt man jedoch die Geschwindigkeit (Beats per Minute, BPM) nicht immer. Erschwert wird das ganze durch Stücke, bei welchen die BPM variieren. Diese Arbeit soll den Musikern helfen diese Hürde zu überwinden.
Es wurden im Voraus sechs Musikstücke als Tests bestimmt, vier davon sind generierte Metronome und zwei entsprechen Ausschnitten aus Songs. Von diesen elf Algorithmen konnten zwei die Test-Musikstücke genug gut erkennen um damit weiterzuarbeiten. Ein Algorithmus davon wurde in Java für Android umgeschrieben.
Als Ergebnis entstand eine Erweiterung zu einer bestehenden Android-App, welche Songs auf dem Smartphone auf deren BPM-Wert untersuchen kann. Der BPM wird als statisch angenommen. Der Fehler dieser Analyse liegt bei $\pm 8.5\%$. Die Analyse mit dynamischem BPM wurde nicht umgesetzt. Nach der Analyse kann ein MP3-File mit dem Metronom zum Song generiert werden.

Metronome können mit einem bestimmten BPM generiert werden und danach mit dem Song zusammen oder ohne Song abgespielt werden. Metronome können nach der Generierung für die gleiche Dauer wie beim Song abgespeichert werden.

Theorie

Dieses Kapitel befasst sich mit der Ausgangslage des Projekts, der daraus folgenden Zielsetzung und eine Grobe Übersicht über die Theorie. Die Anwendung der Theorie ist im Kapitel *Implementierung* ersichtlich.

Zielsetzung

Eine bestehende App soll um die Funktionalität der Beat-Detection und Metronom-Generierung erweitert werden. Als Input wird ein Song aus der Musikbibliothek des Smartphones verwendet. Als Output des ersten Schrittes soll ein BPM-Wert rauskommen zusammen mit einem Offset. Der Offset definiert die Dauer vom Start des Songs bis zum ersten Beat.
In einem zweiten Schritt werden diese beiden Zahlen als Input für den Beat-Generator genommen. Dieser generiert als Output ein MP3-File. Der Vorgang wurde in der folgenden Abbildung 1 dargestellt. Der Song kann sowohl einen statischen BPM-Wert haben, als auch einen Dynamischen.

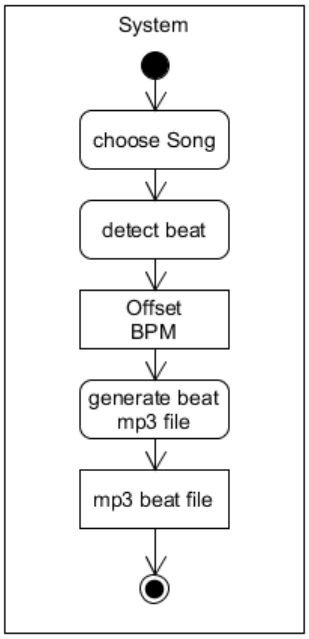

Abbildung 1 Generierung eines mp3 files als Metronom

Die zweite Aufgabenstellung war die Erweiterung der Streaming-Möglichkeiten der App, z.B. Auf den Spotify Premium Dienst. Aus zeitlichen Gründen wurde diese Funktionalität nicht umgesetzt.

Ausgangslage

Diese Arbeit baut auf einer bestehenden App auf. Diese wurde als Bachelorarbeit von den Studenten Rémi Georgiou und André Stocker implementiert. Die Funktionalitäten erlauben Songs mit Filtern zu manipulieren und Spektrogramme zu erstellen. Es wurde eine erste Spotify-Schnittstelle implementiert um mit der App den kostenlosen Spotify-Audio-Stream in gleicher Weise manipulieren zu können wie die Songs in der Bibliothek des Smartphones.

Für die neue Funktionalität wurden zuerst bestehende Lösungen angeschaut. Es gab bereits etliche Lösungsansätze für die Beat-Detection. Unter den untersuchten Algorithmen waren die meisten in Matlab geschrieben. Vertreten waren noch Javascript, Python und C++.Bei etlichen untersuchten Algorithmen stand bereits in der Dokumentation, dass die Genauigkeit entweder nicht sehr hoch ist oder es einen Trade-off zwischen Geschwindigkeit und Genauigkeit gibt. Es wurde kein Algorithmus gefunden, welcher explizit für wechselnde Geschwindigkeiten gemacht wurde. Für diese Arbeit wurden elf Algorithmen ausgewählt und mittels einem Testset von Songs auf ihre Tauglichkeit überprüft. Für die Details der jeweiligen Algorithmen siehe Kapitel *Implementierung*.

Theorie und verwendete Technologien

In diesem Kapitel werden die benötigten Kenntnisse für das Verstehen der Arbeit angesprochen und einige wenige Punkte näher erläutert.

Beat-Detection

Es wird angenommen, dass der Leser die einfachsten Grundlagen der Fourier-Transformation (FFT) kennt. Die einfachsten Grundlagen beinhalten, was die FFT grundsätzlich macht. Die restliche notwendige Theorie wird hier erklärt.

Was ist Beat-Detection

Beat-Detection bezeichnet den Vorgang ein Input-Signal so zu filtern, dass das Auftreten eines Beats möglichst genau bestimmt werden kann. Dieser Beat wird daraufhin für die Weiterverarbeitung extrahiert. Für diesen Vorgang gibt es verschiedene Algorithmen mit stark variierender Genauigkeit (siehe Resultate einzelner Algorithmen im Kapitel *Implementierung*).

JLayer

JLayer ist eine Java-Bibliothek, welche MP3-Dateien decodieren kann. Im Umfang dieser Arbeit werden die Methoden gebraucht um die einzelnen Samples der MP3-Datei auszulesen. Die Bibliothek ist Open Source und unter der LGPL Lizenz erhältlich. [1]. Die Bibliothek wurde bereits in der bestehenden App verwendet und wurde direkt übernommen. Aufgrund der Implementierung in Java wird angenommen, dass JLayer weniger performant ist als nicht-Java Implementierungen wie mpg123. Dies wird in Kauf genommen für im Gegenzug einfache Inbetriebnahme und simpleres Debugging.

JTransforms

JTransforms ist eine Java-Bibliothek, welche eine reine Java-Implementation von verschiedenen Fast-Fourier-Transformationen anbietet. Die Bibliothek ist Open Source und unter der BSD-2-clause Lizenz erhältlich [2]. Die Bibliothek wurde bereits im bestehenden Projekt verwendet und wurde wegen schneller Inbetriebnahme übernommen.

Komponenten

In diesem Kapitel werden einzelne Komponenten der Beat-Detection erläutert. Zudem wird die Grundstruktur der beiden häufigsten Techniken zur Beat-Detection aufgezeigt. Für die genaue Funktionalität eines spezifischen Algorithmus, siehe Kapitel *Implementierung*.

Low-Pass Filter

Ein solcher Filter kann auf ein Signal im Frequenzbereich angewandt werden um nur tiefe Frequenzen durchzulassen. Verwandte Filter sind Band-Pass- und High-Pass Filter, welche nur Frequenzen in einem Intervall, bzw. über einem gewissen Wert durchlassen.
In dieser Arbeit wird per Default ein Low-Pass Filter verwendet um die tieferen Frequenzen eines Audio-Signals zu isolieren und daraus den Beat zu extrahieren.

Energie basierte Beat-Detection

Diese Variante besteht in den Grundzügen aus vier Schritten. Als Input werden die Samples des Audiosignals verwendet. Für die genauen Berechnungen siehe die kommenden Unterkapitel.
In einem ersten Schritt wird die instantaneous energy (IE) berechnet, welche einer Menge von Samples einen Energiewert zuweisen. In einem zweiten Schritt wird die steady local energy (SLE) aus den IE Blöcken berechnet, was dem Durchschnitt der IE Blöcken entspricht. Ab hier hat man zwei Möglichkeiten um weiterzufahren. Entweder man nimmt eine Konstante um die IE Blöcke auf einen Beat hin zu untersuchen, oder man versucht die Konstante durch die Varianz zu ersetzen.
Sollte man sich für die Konstante entscheiden, so muss man für die Konstante einen Wert auswählen, welcher stark für die Qualität der Beat-Detection verantwortlich ist. Hat man einen Wert ausgewählt, so erscheint in einem IE Block ein Beat, falls dieser Block einen höheren Wert hat als der dazugehörige SLE multipliziert mit der Konstanten.
Sollte man sich für die dynamische Form entscheiden, so wird die Konstante b ersetzt durch die Variance over instantaneous Energy (VIE). Bei dieser Methode kommen zwei weitere Konstanten α und β hinzu, deren Wert allerdings einen tieferen Einfluss auf das Ergebnis hat im Verhältnis zu b als Konstante. Die Berechnung, ob in einem IE Block ein Beat auftritt, ist danach gleich wie bei der ersten Methode. Ein Beat tritt auf, falls der IE Block einen höheren Wert aufweist als der dazugehörige SLE Block multipliziert mit der berechneten Variablen b [3]. Der ganze Vorgang mit den beiden Möglichkeiten ist in Abbildung 2 ersichtlich.

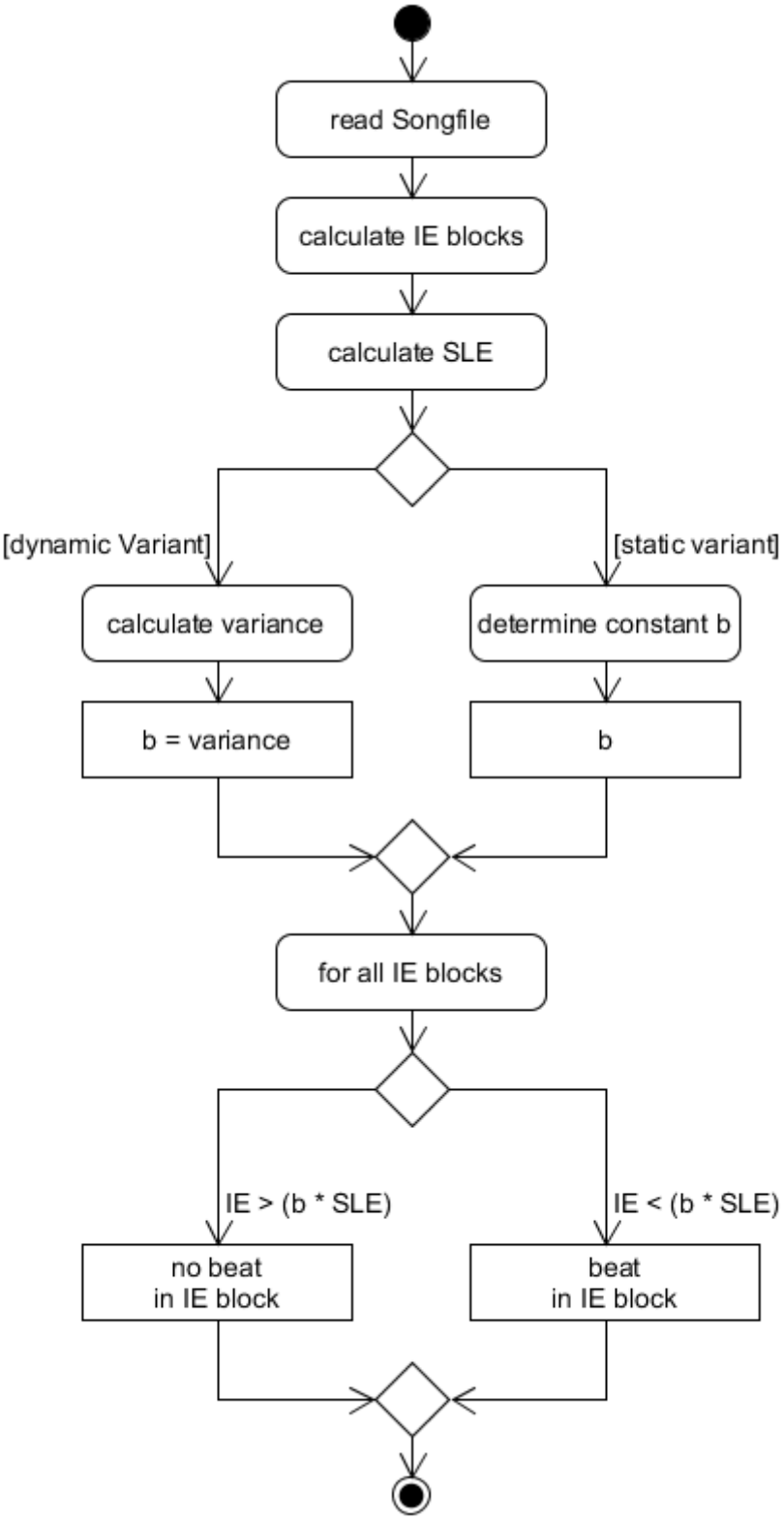

Abbildung 2 Ablauf energiebasierte Beat-Detection

Instantaneous energy oder momentane Energie (IE)

Zuerst wird ein Zeitfenster bestimmt, in welchem die Energy gemessen werden sollte. Anschliessend werden die Quadrate der Samples in diesem Zeitraum aufsummiert. [3]
z.B. $20\,ms$ Zeitfenster bei einer Sample-Frequenz von $8\,kHz$. Dies führt zu $n = 160$ Samples.

$$p[n] = \sum_{k=0}^{n-1} x[n-k]^2$$

Steady local energy (SLE)

Es werden 100 IE aufeinanderfolgende Blöcke von IE Werten benötigt. Diese Werte werden gemittelt. Ein Beat findet in einem Block statt, falls die SLE multipliziert mit einer Konstante kleiner ist als die IE des Blocks. Die Wahl des Werts für die Konstante ist essentiell für eine gute Beat-Detection. [3]

$$SLE = \frac{1}{100}\sum_{j=0}^{99} p[n-j]$$

$$beat\ in\ p[n]\ falls: p[n] > b * SLE$$

wobei
SLE = Steady local energy
p[n] = IE von Samples
b = Konstante

Variance over instantaneous energy (VIE)

Berechnen der der Konstanten b aus der SLE, um die Beat-Detection weniger fehleranfällig zu machen. Es werden 100 aufeinanderfolgende Blöcke von IE Werten benötigt, sowie den dazugehörigen SLE Wert. Die gemittelte quadratische Differenz zwischen jedem IE Block und dem SLE Wert entspricht der Varianz. [3]

$$VIE = \frac{1}{100}\sum_{j=0}^{99}(p[n-j] - SLE)^2$$

$$b = \beta - \alpha * v[n]$$

Wobei
VIE = Varianz über den IE Blöcken
p[n] = IE von Samples
SLE = Steady local energy über den p[n] Blöcken
α= 0.0025
β= 1.5

FFT basierte Beat-Detection

Diese Variante besteht in den Grundzügen aus fünf Schritten. Als Input werden die Samples des Audiosignals verwendet.
Zuerst wird die FFT auf dem absoluten Signal ausgeführt. Vom erhaltenen Resultat wird nur die erste Hälfte weiterverwendet. Auf dieser resultierenden Abbildung der Frequenzen wird ein Low-Pass Filter gelegt. Dieses gefilterte Signal wird durch die inverse FFT wieder in die Zeit Domäne überführt. Jedes Sample, welches einen Wert > 0 hat, wird nun als Beat behandelt.
Vor dem Zählen der Abstände kann je nach Input-Signal noch weiteres Filtern der Daten nötig sein. Der Vorgang ist in Abbildung 3 ersichtlich.

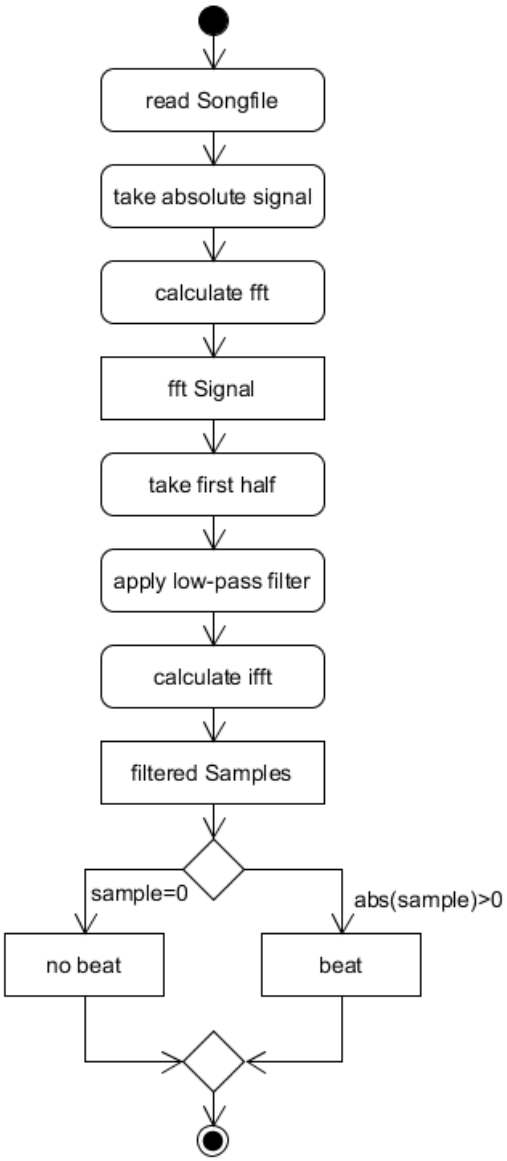

Abbildung 3 Ablauf FFT basierte Beat-Detection

Metronom-Generierung

LAME

LAME ist ein MP3-Encoder, welcher Open Source ist [4]. Der Encoder wurde verwendet, um das vorprogrammierte Metronom als MP3-File abzuspeichern und wieder abzuspielen. Der Encoder ist als NDK für Android vorhanden und wurde für die Metronom-Generierung in die App eingebaut.

SoundPool

SoundPool ist eine Android-Java-Klasse, welche mehrere kleine Audio-Files (unter 1 MB) gleichzeitig abspielen kann. SoundPool übernimmt die Ressourcen-Management auf der Smartphone und folglich muss man sich nicht um CPU-Last oder Memory-Management kümmern.

Mediaplayer

MediaPlayer ist eine Android-Java-Klasse, welche ein grosses Audio-Files abspielen kann. Für eine Instanz von MediaPlayer wird jeweils ein Thread gebraucht, welches von der Medi-Player-Klasse aus einfach erstellt werden kann. Das Ressourcen-Management wird vom Media-Player übernommen.

Wave-Reader/Decoder

Ein Wave-Decoder war in der App schon vorhanden, jedoch ist dieser mit der Implementation von LAME nicht ganz kompatibel. Deswegen wurde der im NDK mitgelieferte Wave-Decoder verwendet. Ein Wave-Decoder dekodiert WAV-Files und stellt Samples der WAV-Files zur Verfügung.

Generierung

Für die Generierung eines Metronoms wurden die oben genannten Technologien und ein WAV-File eingesetzt. Spielt man das WAV-File ab, ist ein Metronom mit vier Schlägen (ein Takt) auf Tempo 120 BPM zu hören.

Metronom direkt abspielen

Das WAV-File wird ins SoundPool geladen. Dabei wird beim Laden das File in Samples bereitgestellt. Dem SoundPool kann für das Abspielen eine Abspielgeschwindkeits-Rate (Playback-Rate) und eine Wiederholungsrate mitgegeben werden.
Der SoundPool kann zwischen 0.5 und 2.0 Abspielgeschwindigkeit-Rate entgegennehmen. Somit es ist es möglich eine BPM zwischen 60 und 240 einzustellen.
$(0.5 * 120 = 60, 2.0 * 120 = 240)$.
Mit der Wiederholungsrate kann mitgeteilt werden, wie oft das Metronom oder ein Takt wiederholt werden sollte. Ein Takt wird folglich beliebig oft wiederholt beziehungsweise abgespielt.
Um genau für die Dauer des Songs das Metronom abzuspielen, muss die Wiederholungsrate (loop) berechnet werden. Folgendermassen wurde die Wiederholungsrate berechnet:

$$Wiederholungsrate(loop) = \frac{bpm/60 * Dauer(in\ Sekunden)}{4}$$

Die 60 steht für die Anzahl Sekunden einer Minute und die 4 steht für die Anzahl Schläge pro Takt. Kommazahlen werden gestrichen, sprich die Wiederholungsrate wird auf die nächste untere Zahl abgerundet. Der gesamte Ablauf ist in Abbildung 4 ersichtlich.

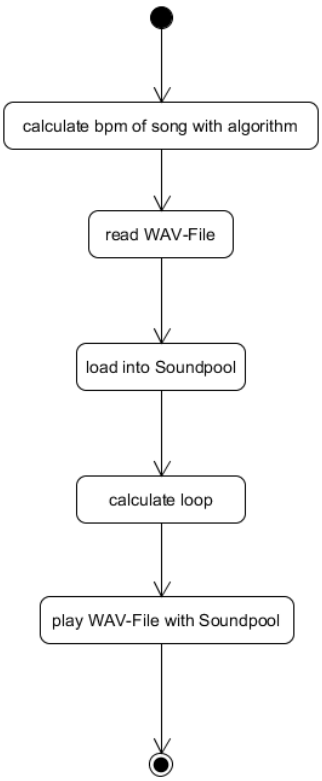

Abbildung 4 Metronom mit gewünschtem BPM im SoundPool abspielen

Metronom abspeichern

Ziel ist es das WAV-File in ein MP3-File umzuwandeln, welches das Metronom mit der gewünschten BPM enthält.
Das WAV-File wird mit dem Wave-Decoder dekodiert und auf diese Weise erhält man die Samples, welche anschliessend dem LAME überreicht werden.
Durch die Veränderung der Sample-Rate wird das Tempo und der Pitch verändert. LAME codiert die vom WAVE-Dekoder erhaltenen Samples mit der Sample-Rate, welche für das BPM des Songs passt. Die Sample-Rate muss für die BPM des Songs also berechnet werden.
Folgendermassen wurde die Sample-Rate berechnet:

$$Sample - Rate\ (neu)\ =\ Sample - Rate\ (alt)\ *\ bpm/120$$

"bpm/120" beschreibt das Verhältnis zwischen der BPM des Songs und der BPM des WAV-File Metronoms, welches ja die BPM 120 besitzt. Dieses Verhältnis wird als Faktor gebraucht, um die Sample-Rate hoch oder hinunter zu skalieren.

Metronom für die Dauer des Songs abspeichern

Nach der Generierung des MP3-Files wird das MP3-File, soviel Mal wie die Wiederholungsrate beträgt, aneinandergefügt und abgespeichert. Als Resultat erhält man ein MP3-File, welches die gleiche Dauer besitzt wie der Song. Für die Generierung des MP3-Files und das Aneinanderfügen der MP3-Files wurden je separate Threads verwendet. Der Gesamte Ablauf ist in Abbildung 5 ersichtlich.

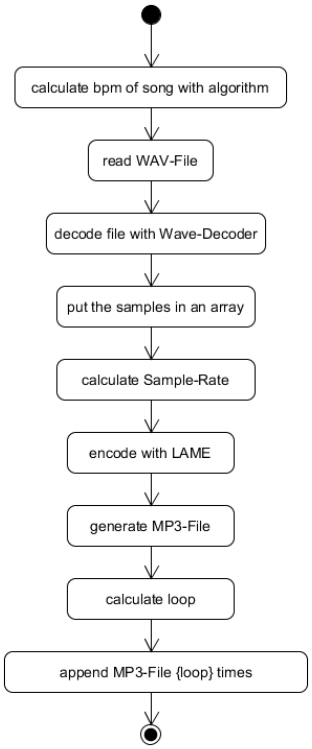

Abbildung 5 Metronom MP3-File Generierung

Metronom mit dem Song abspielen

Nachdem das Metronom für die Dauer des Songs abgespeichert wurde, kann nun das Metronom mit dem Song zusammen abgespielt werden. Hier dafür werden zwei Instanzen von MediaPlayer benötigt. Die Files müssen zuerst in die Instanzen geladen und anschliessend für das Abspielen vorbereitet werden. Nachdem die Files fertig abgespielt wurden, werden die beiden MediaPlayer-Instanzen freigegeben. Bei erneutem Abspielen werden die Instanzen wieder neu erzeugt.

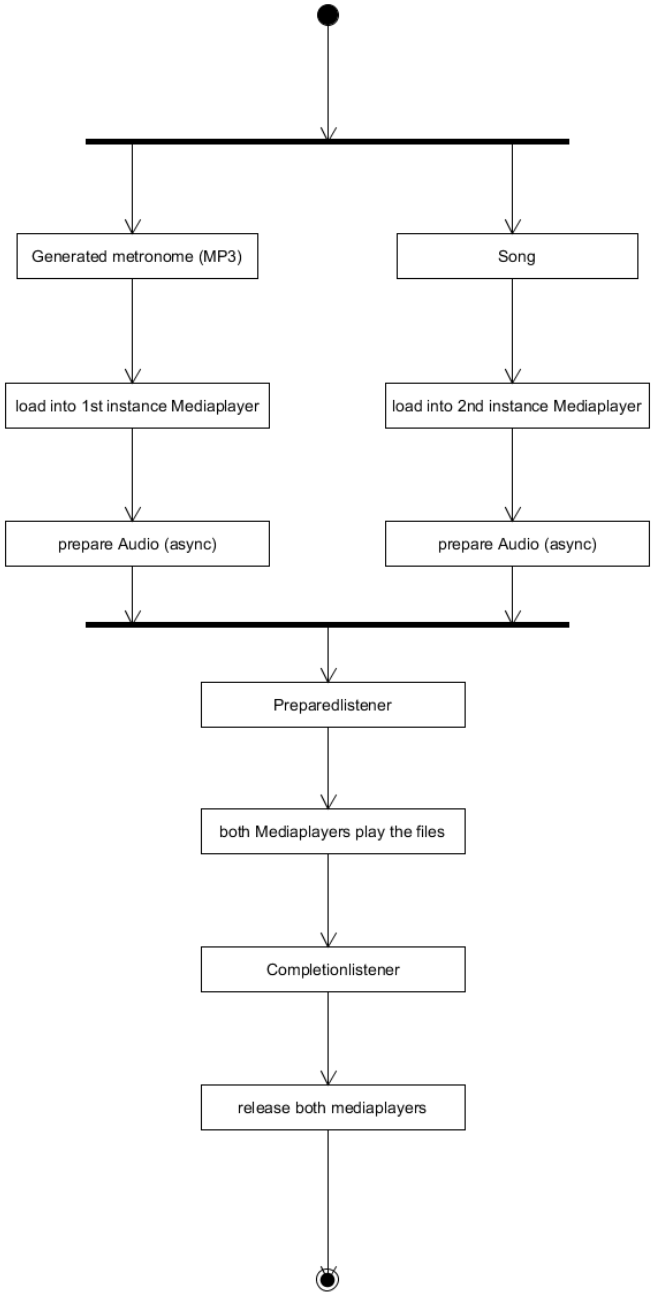

Abbildung 6 Song und Metronom gleichzeitig abspielen

Implementierung

In diesem Kapitel wird das Vorgehen während der Implementierung näher erläutert. Die Resultate der Algorithmen sind in dem jeweiligen Unterkapitel zu finden.

Vorgehen / Methoden

Die Arbeit besteht aus zwei Teilen. Im ersten Teil soll aus einem Song der Beat ausgelesen werden. Dieser Wert soll dann im zweiten Teil dazu verwendet werden ein Metronom zu diesem Song zu generieren. Das Metronom soll aus der App abspielbar und als MP3-Datei abgespeichert werden können. Zusätzlich soll man den Song mit dem erkannten Beat überlagern und als einzelne mp3 Datei abspielen können.
Für den ersten Teil wurden elf Algorithmen aus dem Internet gesucht und nach Möglichkeit in Matlab implementiert. Diese Implementierung wurde durch ein vordefiniertes Testset auf ihre Genauigkeit der Beat-Detection überprüft. Von den zwei Algorithmen, welche die Tests bestanden haben, wurde einer umgeschrieben für Android. Für das Bestehen der Tests in Matlab wurde eine maximale BPM Abweichung von 2 festgelegt.
Im zweiten Teil wurde nach effizienten und einfach zu implementierende Technologien im Internet gesucht. Diese Technologien wurden ausprobiert und schlussendlich wurden diejenigen selektiert, welche für die App am meisten tauglich waren.
Nach der Implementierung wurde das Metronom auf dem Smartphone mit einem echten Metronom unter verschiedenen Tempos verglichen und bezüglich Genauigkeit soweit wie möglich optimiert.

Algorithmen Testset

Das Testset beinhaltet Testdaten um eine möglichst genaue Beat-Detection zu gewährleisten. Es besteht aus insgesamt 8 Musikstücken. 6 Davon sind generierte Metronome mit einem BPM-Wert von 20, 60, 100, 120, 200 und 240. Die Metronome wurden von der Website "Metronomer" generiert. [5]. Bei den beiden anderen Musikstücken handelt es sich um Songausschnitte, welche von Youtube heruntergeladen wurden. Die Referenzangaben zur BPM stammen von der Website songbpm. [6]
Das erste Musikstück ist ein Ausschnitt aus einem Drum-Cover von “Sugar” von der Gruppe “Maroon5“. Der Ausschnitt läuft von 00:20 bis 00:50 [7]. Der als korrekt angenommene BPM-Wert ist 120. [8]
Das zweite Musikstück “Hey boy” von der Gruppe ”Teddybears sthlm“. Der Ausschnitt des Tests geht von 00:00 bis 01:00 [9]. Der als korrekt angenommene BPM-Wert ist 200 [10].

Untersuchte Algorithmen

In diesem Kapitel werden die untersuchten Algorithmen genauer beschrieben. Bei vorliegenden Resultate stammen diese aus der Matlab-Implementation.
Falls in der Detailbeschreibung der einzelnen Algorithmen Testergebnisse aufgeführt sind, werden sie folgende Form haben.

Metronom/Song	Soll BPM	Ist BPM	Faktor

Die erste Spalte klassifiziert das Audiosignal des Tests. Die möglichen Werte sind “Metronom” oder “Song”. Die Spalte “Soll BPM” definiert den Wert BPM des Audiosignals. Dieses gilt es so genau wie möglich zu erreichen. Die dritte Spalte beinhaltet den Wert, welchen der Algorithmus für das Audiosignal berechnet hat. Die vierte Spalte beinhaltet das Verhältnis von “Soll BPM” zu “Ist BPM”. EinWert von 1.00 ist hierbei das Optimum.
Manche der unten aufgeführten Algorithmen wurden nicht implementiert oder genauer angeschaut. Sie werden trotzdem aufgelistet um mögliche Ansatzpunkte für zukünftige Implementationen zu geben.

Übersicht

In der Tabelle 1 ist die Übersicht über die untersuchten Algorithmen zu sehen. Die Spalte “Basis“ bezeichnet dabei jeweils mit welcher theoretischen Basis die Beat-Detection funktioniert dieser Algorithmen funktioniert. Die dritte Spalte listet die Gründe auf, wieso der Algorithmus nicht gewählt wurde oder ggf., dass er akzeptiert wurde.

Algorithmus #	Basis	Akzeptiert / Grund Ablehnung
1	FFT + Energie	Metronome deutlich nicht erkannt
2	Energie	Metronome deutlich nicht erkannt
3	FFT	Akzeptiert, aus Zeitgründen nicht auf Android implementiert
4	Energie + FFT	Metronom nicht erkannt, evtl. falsche Implementierung
5	Energie	Metronom deutlich nicht erkannt
6	FFT	In C++ und Python geschrieben, nicht vollständig evaluiert
7	Energie + FFT	Aus Zeitgründen nicht vollständig evaluiert
8	Energie	Metronom nicht erkannt, gewählte Konstanten ungenügend
9	FFT	akzeptiert
10	Energie	Zu stark abhängig und eingeschränkt von Konstanten
11	Energie	Metronom nicht erkannt.

Tabelle 1 Übersicht analysierte Algorithmen

Algorithmus 1

FFT basierter Algorithmus in vier Schritten. In einem ersten Schritt wird die FFT aus dem Audiosignal genommen. Dieses erhaltene Signal wird in sieben Untersignale unterteilt, welche jeweils ein Intervall von Frequenzen beinhalten. Die Frequenzen sind 0-200 Hz, 200-400 Hz, 400-800 Hz, 800-1600 Hz, 1600-3200 Hz, 3200-6400 Hz und 6400-12800 Hz. Dieser Schritt soll die verschiedenen Klänge der Instrumente in einem Audiosignal trennen. Von diesen sieben Signalen wird jeweils die IFFT genommen und für jedes Signal werden die folgenden Schritte ausgeführt.
In Schritt zwei soll das Signal geglättet werden. Dazu wird auf jedes Signal nochmals die FFT angewendet. Auf diesem Signal wird nun die rechte Hälfte eines Hanning-Windows gelegt um die Stärke der höheren Frequenzen abzuschwächen. Auf dieses resultierende Signal wird wiederum die IFFT angewandt um die Zeitdomäne zu bekommen.
In Schritt drei wird das Signal nach der Zeit abgeleitet um Anstiege in der Amplitude besser zu erkennen. Das resultierende Signal wird noch gefiltert, so dass nur die steigenden Amplituden übrig bleiben.
Im vierten und letzten Schritt werden Kammfilter verwendet um sich dem BPM des Songs anzunähern. Ein Kammfilter ist einem spezifischen Tempo zugeordnet. Wird ein solcher Filter über ein Signal mit dem gleichen Tempo gelegt, so wird die Energie des Signals grösser, als wenn das Tempo nicht übereinstimmen würde. Um dies zu erreichen wird wiederum die FFT vom Audiosignal genommen. Dieses erhaltene Signal wird mit dem FFT-Signal des Kammfilters multipliziert. Die Energie dieses Frequenzsignals wird durch aufsummieren ihrer Teile berechnet. Für jedes Tempo auf das geprüft werden soll, muss die Energie berechnet werden. Der Kammfilter mit der höchsten Energie wird dann als BPM bestimmt. [11]
Diese vier Schritte sind in Abbildung 7 ersichtlich.

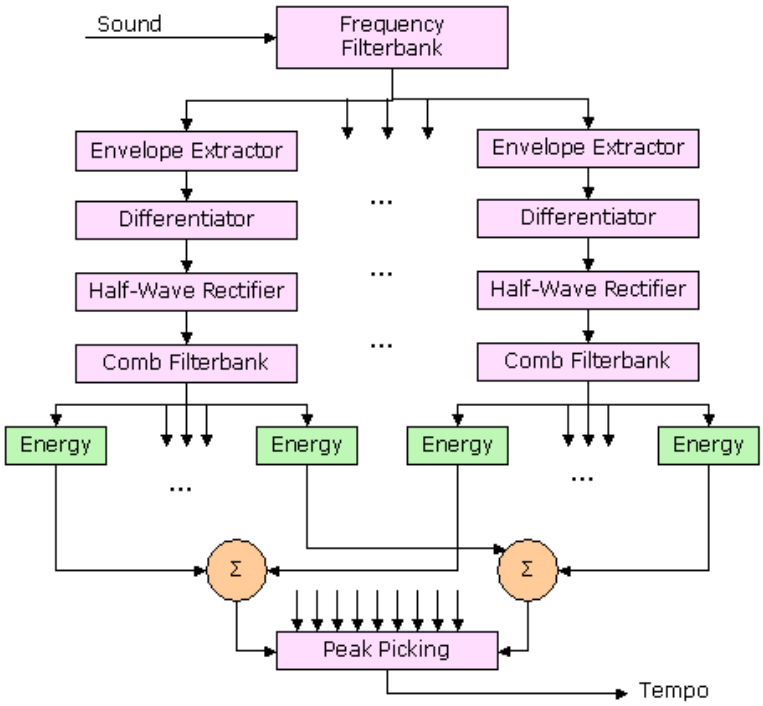

Abbildung 7 Ablauf Beat-Detection Algorithmus 1 [12]

Die Matlab-Dateien zu diesem Algorithmus wurden vom Autor auf seiner Webseite zur Verfügung gestellt. Während der Arbeit wurde am Ursprungscode nichts verändert, einzig die Parameter für die verschiedenen Frequenzintervalle im Schritt eins wurden verändert um eine bessere Erkennung zu erreichen.
Der Algorithmus konnte weder die Metronome noch die Songs korrekt, oder mit kleinem Fehler erkennen.
In Tabelle 2 sind die Resultate ersichtlich.
Aus der letzten Spalte ist ersichtlich, dass der Algorithmus am zuverlässigsten für Metronome mit BPM-Werten zwischen 100 und 240 funktioniert. Der Song 1 mit dem BPM-Wert von 120 wird jedoch wiederum schlechter erkannt als das gleich schnelle Metronom.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	239.96	0.08
Metronom	60	109.49	0.55
Metronom	100	83.33	1.20
Metronom	120	100.00	1.20
Metronom	200	166.69	1.20
Metronom	240	199.99	1.20
Song 1	120	73.09	1.65
Song 2	200	73.09	2.74

Tabelle 2 Ergebnisse Algorithmus 1

Algorithmus 2

Energie basierter Algorithmus. Dieser Algorithmus teilt die Datei in Sample-Blöcken und vergleicht die Energie des Blocks mit dem vorangehenden Block. Die Energie eines Blocks wird genutzt, um den Beat zu erkennen. Wenn die Energie eines Blocks eine bestimmte Schwelle übersteigt, wird der Block von Samples als Beat erkannt. [13]
Jedoch wurden bereits die Metronome deutlich nicht erkannt und somit wurde dieser Algorithmus nicht weiter untersucht und auch nicht in Android implementiert. Die Ergebnisse sind in Tabelle 3 ersichtlich

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	2629.00	0.007
Metronom	60	2478.00	0.02
Metronom	100	2374.00	0.04
Metronom	120	2158.00	0.05
Metronom	200	2231.00	0.08
Metronom	240	2062.00	0.11
Song 1	120	902.77	0.13
Song 2	200	1287.00	0.16

Tabelle 3 Ergebnisse Algorithmus 2

Algorithmus 3

Dieser Algorithmus stammt aus einem grösseren Projekt zum Thema „Cover-Song-Detection“. Der Matlab-Code wurde unverändert von der Webseite des Herstellers heruntergeladen. Der Algorithmus basiert auf der FFT. Die genaue Funktionsweise wurde aufgrund von Zeiteinschränkungen und Komplexität des Algorithmus nicht näher analysiert. [14]
Die Resultate aus den Tests sind in Tabelle 4 ersichtlich. Zwischen den BPM-Werten von 60 und 200 sind die Ergebnisse sehr gut. An den Rändern mit den BPM-Werten von 20 bzw. 240 zeigen sich Unterschiede zwischen „Soll“ und „Ist BPM“. Es wurde die Annahme getroffen, dass diese Ungenauigkeiten mit mehr Zeitaufwand minimiert werden könnten. Aufgrund davon wurde dieser Algorithmus als Kandidat für die Java-Implementation aufgenommen. Schlussendlich wurde er jedoch aus Zeitgründen nicht umgeschrieben.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	416.67	20.83
Metronom	60	60.00	1.00
Metronom	100	100.00	1.00
Metronom	120	120.00	1.00
Metronom	200	200.00	1.00
Metronom	240	120.00	2.00
Song 1	120	120.00	1.00
Song 2	200	200.00	1.00

Tabelle 4 Ergebnisse Algorithmus 3

Algorithmus 4

Energie und FFT basierter Algorithmus in drei Schritten. Im ersten Schritt berechnet man aus dem Input-Signal die IE für ein bestimmtes Zeitfenster. In einem zweiten Schritt wird die FFT aus den Ergebnissen aus dem ersten Schritt genommen. Im dritten Schritt wird die tiefste Frequenz aus dem Signal bestimmt. Diese wird als Annäherung für den BPM-Wert genommen. [3]
Die Resultate waren durchgehend auf einem BPM-Wert von 0 (sichtbar in Tabelle 5). Es wurde in Betracht gezogen, dass bei der Implementierung dieses Algorithmus in Matlab ein Fehler gemacht wurde.
Dadurch, dass bereits zwei andere Algorithmen als Kandidaten für Android vorhanden waren, wurde nur wenig Zeit in die Fehlersuche investiert.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	0.00	-
Metronom	60	0.00	-
Metronom	100	0.00	-
Metronom	120	0.00	-
Metronom	200	0.00	-
Metronom	240	0.00	-
Song 1	120	0.00	-
Song 2	200	0.00	-

Tabelle 5 Ergebnisse Algorithmus 4

Algorithmus 5

Gleicher Algorithmus wie Algorithmus 4. Aufgrund von parallelen Aktivitäten wurde dieser Algorithmus aus Versehen doppelt in die Liste der Algorithmen aufgenommen. Zugunsten der konsistenten Nummerierung von Algorithmen Im Code und in der Ordnerstruktur des Projekts wurde dieser Algorithmus beibehalten. Für die Details siehe Algorithmus 4.

Algorithmus 6

FFT basierter Algorithmus. Dieser Algorithmus basiert auf der Essentia Bibliothek, welche in C++ und Python geschrieben wurde. In diesen Sprachen war nur begrenztes Vorwissen vorhanden und somit wurde dieser Algorithmus erst gegen Ende der Evaluierungsphase angeschaut. Die Einarbeitung wurde als zu zeitaufwändig betrachtet und eine genauere Evaluation blieb aus. [15]

Algorithmus 7

FFT und Energie basierter Algorithmus. Dieser Algorithmus wird in einem Paper beschrieben. Aufgrund der hohen Komplexität wurde er nur oberflächlich angeschaut. Die genauere Analyse hätte stattgefunden, wenn kein anderer Kandidat für die Java-Implementierung gefunden oder genug Zeit vorhanden gewesen wäre. Es wurden jedoch Kandidaten gefunden und aus Zeitgründen wurde dieser Algorithmus nicht näher analysiert. [16]

Algorithmus 8

Energie basierter Algorithmus in vier Schritten. Im ersten Schritt wird über einen Zeitraum von einer Sekunde für alle 1024 Samples die IE bestimmt. Im zweiten Schritt wir die SLE von diesen erhaltenen Werten bestimmt. Im dritten Schritt wird die VIE bestimmt und die Konstante C folgendermaßen bestimmt: $C = -0.0024514 * VIE + 1.5142856$. Die beiden Konstanten stammen laut Autor aus der linearen Degression. In einem IE Block tritt nun ein Beat auf falls: $IE > C * SLE$. [17]
Bei den Resultaten in Tabelle 6 ist ersichtlich, dass sich dieser Algorithmus für die Erkennung der Testdaten nicht eignet. Die Stärke des Algorithmus scheinen klare Signale zu sein, mit einer maximalen Genauigkeit bei einem BPM-Wert von 60. Davon abweichende Signale werden nicht mehr richtig erkannt, wobei langsamere BPM-Werte noch besser erkannt werden als Schnellere.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	27.99	0.71
Metronom	60	59.98	1.00
Metronom	100	2.00	50.00
Metronom	120	0.00	-
Metronom	200	0.00	-
Metronom	240	0.00	-
Song 1	120	0.00	-
Song 2	200	0.00	-

Tabelle 6 Ergebnisse Algorithmus 8

Algorithmus 9

FFT basierter Algorithmus in sechs Schritten. Im ersten Schritt wird das Signal mit einem 0er Padding versehen, bis es die Grösse der nächsten Zweierpotenz aufweist. Im zweiten Schritt wird die FFT auf das Signal angewandt und nur die erste Hälfte des resultierenden Signals weiterverarbeitet. Im dritten Schritt wird ein Low-Pass Filter für bis zu 1500 Hz auf das Signal angewandt. Auf dieses gefilterte Signal wird im vierten Schritt die IFFT angewandt um zurück in die Zeitdomäne zu gelangen. Dieses Signal wird im Schritt fünf wiederum mit einem Threshold von 0.9 gefiltert. Jedes Sample unter diesem Wert wird auf den Wert 0 gesetzt, alle anderen behalten ihren Wert. In einem sechsten Schritt werden die Abstände zwischen den Samples ausgerechnet und mithilfe der Sample-Rate des Signals die BPM berechnet. [18]
Die Resultate für diesen Algorithmus waren gut und sind in Tabelle 7 ersichtlich. Die Metronome wurden durchgehend korrekt erkannt. Die Songs wurden mit einer Genauigkeit von $\pm 2\ BPM$ erkannt, was als genau genug angesehen wurde. Aufgrund der korrekten Resultate wurde dieser Algorithmus zum Kandidat für die Android-Implementation. Schlussendlich hat sich dieser Algorithmus gegen den Algorithmus 3 durchgesetzt und wurde für Android implementiert.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	20.00	1.00
Metronom	60	60.00	1.00
Metronom	100	100.00	1.00
Metronom	120	120.00	1.00
Metronom	200	200.00	1.00
Metronom	240	240.00	1.00
Song 1	120	122.00	0.98
Song 2	200	202.00	0.99

Tabelle 7 Ergebnisse Algorithmus 9

Algorithmus 10

Energie basierter Algorithmus in drei Schritten. Sehr ähnlich wie andere energiebasierte Algorithmen wie z.B. Algorithmus 8. Im ersten Schritt wird die IE für 2 Sekunden gemacht mit einer Fenstergröße von 20 Millisekunden. Im zweiten Schritt wird die SLE davon berechnet. In Schritt drei wird berechnet in welchem IE Block ein Beat vorkommt. Ein Beat kommt in einem IE Block vor falls $IE > b * SLE$, wobei b eine Konstante mit dem Wert 0.1 gewählt wurde. [3]
Diese Resultate sind in Tabelle 8 ersichtlich. Wie bei den Algorithmen 4 oder 8 sind einige „Ist BPM"-Werte 0.00. Dieser Algorithmus ist für unsere Zwecke aufgrund der Ungenauigkeiten ungeeignet.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	0.00	-
Metronom	60	0.00	-
Metronom	100	0.00	-
Metronom	120	0.00	-
Metronom	200	0.00	-
Metronom	240	0.00	-
Song 1	120	30.00	4.00
Song 2	200	30.00	6.67

Tabelle 8 Ergebnisse Algorithmus 10

Algorithmus 11

Energie basierter Algorithmus in vier Schritten. Sehr ähnlich wie Algorithmus 8 und direkte Erweiterung des Algorithmus 10. Im ersten Schritt wird die IE für 2 Sekunden gemacht mit einer Fenstergröße von 20 Millisekunden. Im zweiten Schritt wird die SLE davon berechnet. Im Unterschied zu Algorithmus 10 wird hier die Konstante b berechnet. Im dritten Schritt wird die VIE für die Daten aus Schritt eins und zwei berechnet. Die Variable b wird nun folgendermaßen berechnet: $b = \beta - \alpha * VIE[n]$. Wobei $\alpha = 0.0024$ und $\beta = 1.5$. Ein beat tritt auf falls für einen IE Block gilt: $IE > b * SLE$. [3]
Die Resultate sind in
Tabelle 9 ersichtlich. Der Algorithmus erkannte weder ein Metronom noch ein Song korrekt. Der errechnete BPM-Wert ist im Vergleich zu allen anderen Algorithmen grösser. Aufgrund der Ungenauigkeiten wurde der Algorithmus nicht zum Kandidat für die Android-Implementierung.

Song/Metronom	Soll BPM	Ist BPM	Faktor
Metronom	20	120.00	0.17
Metronom	60	330.00	0.18
Metronom	100	3000.00	0.03
Metronom	120	3000.00	0.04
Metronom	200	3000.00	0.07
Metronom	240	3000.00	0.08
Song 1	120	3000.00	0.04
Song 2	200	2010.00	0.10

Tabelle 9 Ergebnisse Algorithmus 11

Implementierter Algorithmus in Android

Die beiden Algorithmen 3 und 9 waren Kandidaten für die Implementierung für Android.
Algorithmus 9 hat sich aufgrund seiner tieferen Komplexität und seinen leicht besseren Testergebnissen als Sieger herausgestellt.
Die Implementation konnte nicht komplett übernommen werden aufgrund von Hardware-Einschränkungen von Smartphones und Unterschiede in der Implementierung der FFT und des MP3-Decoders für Java und Matlab.
Der Algorithmus wurde als AsyncTask implementiert um die Usability durch lange Analysen möglichst nicht zu verringern.

Memory Verbrauch

Für die Analyse des 60-BPM-Metronoms mit dem Algorithmus 9 benötigt Matlab laut Profiler knapp 700 MB Memory. Schwächere oder ältere Smartphones können dieses Memory nicht bereitstellen. Auf einem der Testgeräte (Sony Xperia Z5 Compact) steht laut der internen Memory Übersicht im Schnitt nur 100 MB Memory zur Verfügung. Mithilfe von programmatischen Anpassungen konnte der Memory Verbrauch auf ca. 160 MB für die Analyse reduziert werden. Der Verbrauch von durch den Profiler von Android Studio ermittelt. Für die Implementierung in Java wurde double point precision verwendet wo möglich.

Vergleich der verwendeten Funktionen in Matlab und Java

Der MP3-Decoder in Matlab ist durch den Befehl audioread() gewrappt. Dieser liefert normierte double point precision Samples. Im Gegensatz dazu liefert JLayer in Java die Samples in einem Short-Array. Genauere Untersuchungen der Samples zeigen, dass JLayer mindestens bei kleinen Werten ungenauer ist, da diese für Samples den Wert 0 liefert, wobei Matlab noch kleine Werte zurückgibt.
Sollte ein Mono-File analysiert werden, so muss in Matlab keine Fallunterscheidung gemacht werden, da das komplette File eingelesen wird. JLayer hingegen liefert Samples in Blöcken. Ein Monoblock besteht hierbei in der ersten Hälfte aus relevanten Samples und in der zweiten Hälfte aus 0.
Matlab hat eine out-of-the-box FFT-Implementation, im Gegensatz zu Java. Es wurde die Library JTransforms verwendet. JTransforms bietet eine Float und eine Double-Implementation an. Aufgrund der Präzision hat man sich für die Double-Implementation entschieden. Zusätzlich wurde die JTransforms-Möglichkeit benutzt, um nur die reellen Werte der FFT zu berechnen, da der komplexe Teil in dieser Aufgabenstellung nicht gebraucht wird.

Vergleich Algorithmus Java und Matlab in Funktion

In Matlab konnte der gesamte Song in einem Stück analysiert werden. Auf Android ist dies aufgrund des beschränkten Memorys nicht möglich. Es wurde die Entscheidung getroffen, dass bei Android jeweils 10 Sekunden analysiert werden. Sollte sich der Algorithmus im ersten 10-Sekunden-Block zu mindestens 60% sicher sein bezüglich eines BPM-Wertes, so wird dieser Wert als BPM des Songs genommen. Sollte diese prozentuale Sicherheit nicht erreicht werden, so wird für jeden 10-Sekunden-Block die möglichen BPM-Werte berechnet. Wurden alle Blöcke des Songs verarbeitet, so werden die einzelnen BPM-Werte zusammengefügt und gezählt, wie viele Male jeder BPM-Wert vorkommt. Diese BPM-Werte werden dann multipliziert bis zu einem maximalen BPM-Wert von 480. Auch in diesem Schritt wird wiederum gezählt, wie viele Male jeder BPM-Wert vorkommt. Derjenige BPM-Wert, welcher in diesem letzten Schritt am meisten vorkommt, wird als BPM-Wert des Songs behandelt. Sollten mehrere Werte den gleichen Wert haben, so wird der kleinste BPM-Wert genommen. Dieses Verfahren funktioniert nur für Songs mit einem statischen BPM-Wert. Für wechselnde BPM's siehe das Kapitel *Wechselnde Geschwindigkeiten*. Für den Ablauf des Algorithmus in Android siehe Abbildung 8. Die Matlab-Implementierung gibt nur einen BPM-Wert zurück und die Android-Implementierung gibt eine HashMap zurück, welche beliebige Werte aus der Analyse zurückgeben kann. Momentan wird der BPM-Wert, der Offset bis zum ersten Beat in Samples, die Sample-Rate und die Anzahl Kanäle des Songs zurückgegeben.

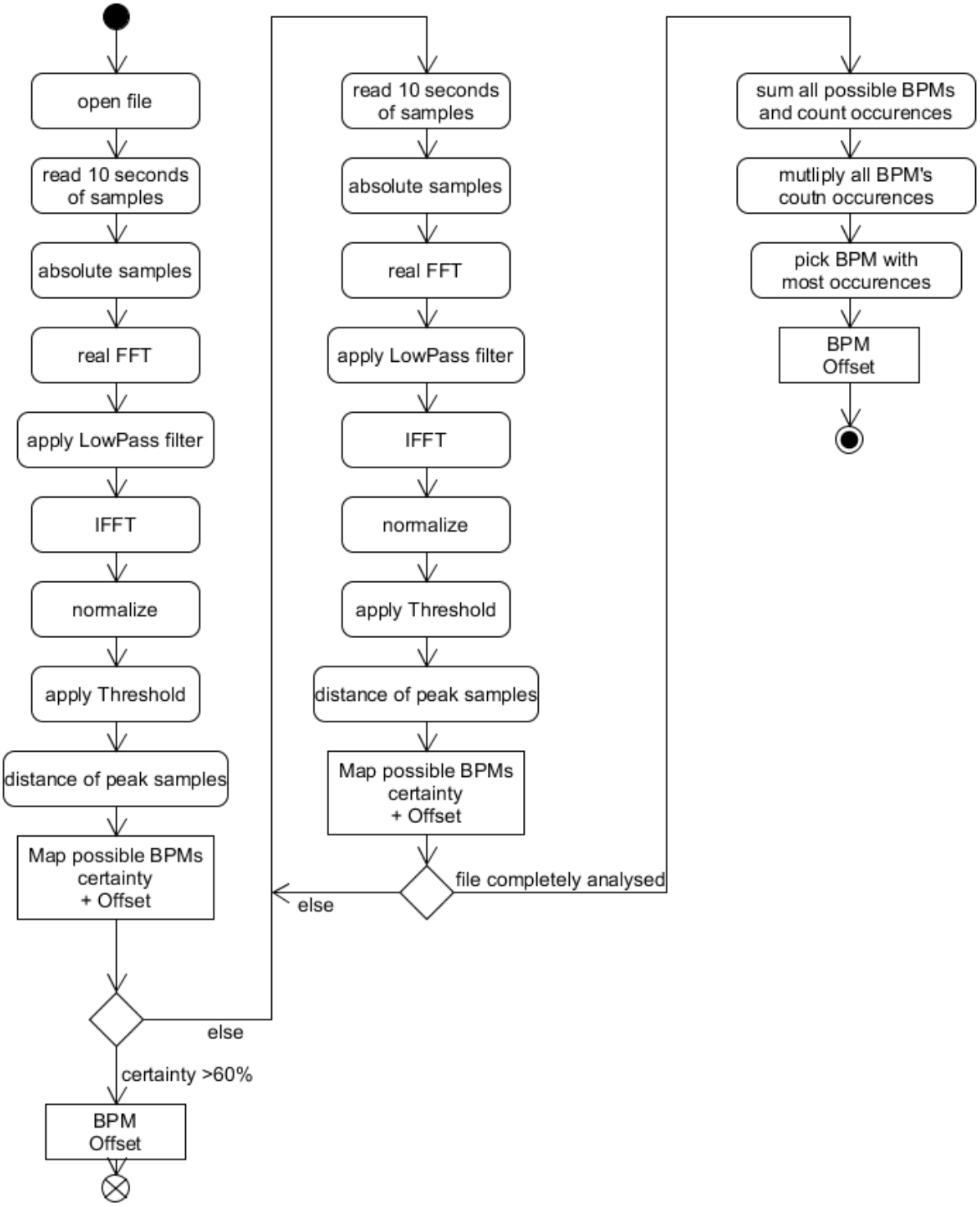

Abbildung 8 Ablauf angepasster Algorithmus 9 in Java für Android

Die Matlab-Implementation ist der Android-Implementation in Bezug auf Genauigkeit und der Geschwindigkeit überlegen. Die Geschwindigkeit ist durch die stärkeren Prozessoren eines Desktop-Computers im Vergleich zu einem Smartphone erklärbar. Es wird vermutet, dass die Ungenauigkeit der Ergebnisse bei der Android-Implementation zum einen von JLayer und zum anderen von JTransforms stammen. In der Tabelle 10 sind die Ergebnisse im Vergleich aufgelistet.
Die erste Spalte bestimmt die Klassifikation des Audiosignals in Song oder Metronom. Die zweite Spalte zeigt den gewünschten BPM-Wert auf. Die dritte Spalte zeigt das Ergebnis der Matlab-Implementation, Spalte vier dasjenige der Android-Implementation. Die fünfte Spalte zeigt ein Tupel von Werten. Der erste Wert ist die Abweichung in % des Matlab-Ergebnisses vom gewünschten Wert. Der zweite Wert des Tupels dasselbe mit der Android-Implementation. Es ist ersichtlich, dass bei den Metronomen beide Implementationen auf die gleichen und korrekten Ergebnisse kommen. Bei den Songs ist die Android-Implementation weniger genau, z.B. bei Song 1 mit gut 8% Fehler, wobei der Matlab-Algorithmus nur eine Abweichung von knapp 2% aufweist. Bei Song 2 schneiden beide Implementationen gleich schlecht ab mit jeweils 1% Fehler.

Song/Metronom	Soll BPM	Ist BPM Matlab	Ist BPM Android	Abweichung in %
Metronom	20	20.00	20.00	0.00 | 0.00
Metronom	60	60.00	60.00	0.00 | 0.00
Metronom	100	100.00	100.00	0.00 | 0.00
Metronom	120	120.00	120.00	0.00 | 0.00
Metronom	200	200.00	200.00	0.00 | 0.00
Metronom	240	240.00	240.00	0.00 | 0.00
Song 1	120	122.00	130.00	1.67 | 8.33
Song 2	200	202.00	198.00	1.00 | -1.00

Tabelle 10 Vergleich Resultate Matlab und Android

Das Feature, dass Songs mit wechselndem BPM-Wert erkannt werden, war in der Aufgabenstellung definiert. Es wurde nur der Theorieteil bearbeitet. Die Implementierung fand aufgrund des erhöhten Fehlers in der Android-Implementierung und Zeitmangel nicht statt.
Die Theorie besteht aus 2 Schritten. Im ersten Schritt wird der Song in 10-Sekunden-Blöcke unterteilt. Im zweiten Schritt wird für jeden von diesen Blöcken eine BPM-Analyse durchgeführt, wie bereits jetzt in der Android-Implementierung. Als Rückgabewert würde es nicht einen einzigen BPM-Wert geben, sondern eine Liste von BPM-Werten für jeden 10-Sekunden-Block. Die Wahl von 10 Sekunden wurde aufgrund der Implementierung des Algorithmus für statische BPM-Werte gewählt und noch nicht genauer auf ihre Tauglichkeit überprüft.

Resultate

In diesem Kapitel werden die Resultate der beiden Funktionalitäten Beat-Detection und Metronom-Generierung kurz zusammengefasst.

Beat-Detection

Es wurde ein Algorithmus in Matlab evaluiert und in Android implementiert. Der Algorithmus ist via einem UI-Fragment des bestehenden Apps erreichbar. Jeder Song aus der Song Bibliothek des Smartphones kann analysiert werden. Der ganze Analysevorgang geschieht asynchron und bei Beendigung des Vorgangs wird der Benutzer via Toast informiert.
Der Algorithmus selbst erkennt Metronome mit einem statischen BPM-Wert. Bei Songs mit einem statischen BPM-Wert erkennt der den Wert mit einem Fehler von $\pm 8.5\%$. Songs mit einem variablem BPM werden als Songs mit einem statischen BPM-Wert behandelt und werden dementsprechend ungenaue Resultate liefern. Die Funktionalität für variable BPM-Werte wurde nur in Theorie für den Algorithmus umgesetzt, es existiert kein Code dazu.

Metronom Generierung

Das Metronom ist durch den gleichen UI-Fragment erreichbar. Der Benutzer kann ein einfaches Metronom mit einem BPM, welches von 60 bis 240 beträgt, abspielen und kann dabei zudem auswählen, wievielmal er es abspielen möchte. Die BPM und die Wiederholungsrate können auf der UI per Slider ausgewählt werden. Nach der Auswahl kann das Metronom per Button abgespielt werden.
Die Metronome werden sauber, schnell und effizient generiert und werden auf dem internen Speicher in einem separaten Ordner namens „audio_signal_processing_metronome“ abgespeichert. Das Generieren und Abspeichern geschieht ebenfalls asynchron und der Benutzer wird per Text-Anzeige unter den Buttons bei Beendigung benachrichtigt. Um diese Funktion auszuführen steht ein Button zur Verfügung.
Das generierte und abgespeicherte Metronom kann mit dem Song zusammen abgespielt werden. Auch dafür gibt es einen separaten Button.

Diskussion und Ausblick

In diesem Kapitel werden die Resultate in Bezug auf die Aufgabenstellung analysiert und mögliche Erweiterungen/ Verbesserungen aufgelistet.

Resultate und Zielsetzung

Beat-Detection

Der erste Punkt der Aufgabenstellung war die Erweiterung der App um weitere Funktionen. Die App wurde erfolgreich erweitert. Die Fragment-Architektur der bestehenden App wurde um ein Metronom-Fragment erweitert. Das UI-Design des Fragments ist funktional und nicht bis ins letzte Detail poliert.
Eine Funktionalität sollte die Beat-Detection sein, welche einen ausgewählten Song auf seinen BPM analysiert. Die Funktion wurde implementiert, der User hat die Möglichkeit ein Song aus seiner Bibliothek auszuwählen und die Beat-Detection mit dessen Input zu starten. Der User bekommt als Output wie gefordert sowohl den BPM-Wert, als auch das Offset bis zum ersten Beat in Samples. Zusätzlich gibt der Algorithmus die Anzahl Kanäle und die Sample-Rate des Songs zurück. Der ganze Algorithmus läuft asynchron ab. Nach dem Ende der Analyse hat der User die Möglichkeit sich aus den Daten ein Metronom zu generieren.
Eine Zielsetzung war die Beat-Detection auch für Songs mit variablem BPM-Werten zu entwickeln. Diese Funktion wurde nicht implementiert. Im momentanen Zustand betrachtet der Algorithmus alle Songs als Songs mit statischem BPM.
Die Zuverlässigkeit der Beat-Detection liegt an der Grenze zu schlecht. Die Testmetronome werden korrekt erkannt, die Songs jedoch haben einen Fehler von bis zu $\pm 8.5\%$. In diesem Gebiet gibt es definitiv noch starkes Verbesserungspotential.

Metronom Generierung

Die andere Funktionalität sollte die Metronom-Generierung sein. Einerseits sollte das Metronom direkt abspielbar sein, andererseits sollte man das Metronom als MP3-File abspeichern können. Das direkte Abspielen wurde vollständig implementiert. Beim Abspeichern des MP3-Files konnte das vom Algorithmus mitgegebene Offset nicht verarbeitet werden. Mit dem Offset sollte ein MP3-File generiert werden, welches für die Dauer bis zum ersten Beat Funkstille beinhaltet. Dieses generierte MP3-File, welches an den Fang des Metronom-Files angeheftet werden sollte, konnte aus unerklärlichen Gründen nicht abgespielt werden. Im Endeffekt ertönen die Beats beim gleichzeitigen Abspielen des Metronoms und des Songs nicht gleichzeitig. Auch hier gibt es noch Verbesserungspotential.

Mögliche Erweiterungen der Resultate

Allgemein

Die Beat-Detection wurde als asynchroner Task implementiert, der eine URI zu einem File als Parameter nimmt. Somit ist die Wiederverwendung dieses Moduls in einem anderen Kontext sehr einfach möglich. Die Funktionalität wird über das AsyncTask Interface gesteuert.

Beat-Detection

Eine Erweiterung wäre die Implementierung von Beat-Detection für variable BPM-Werte in einem Song. Um dies zu bewerkstelligen muss wahrscheinlich die Beat-Detection mit statischem BPM-Werten zuerst genauer gemacht werden.
Der Memory Verbrauch konnte im Vergleich zur Matlab-Version stark reduziert werden, trotzdem ist noch Platz für Verbesserungen vorhanden. Ein möglicher Anfangspunkt wäre von Double auf Float zu wechseln. In diesem Fall muss die Qualität der Beat-Detection im Auge behalten werden. Ein weiterer Ansatzpunkt wäre die Reduktion der im Speicher gehaltenen Array mit Zwischenergebnissen des Algorithmus.
Der Algorithmus 3 hatte Potential wurde jedoch aus Zeitgründen nicht implementiert. Eine mögliche Erweiterung wäre es diesen Algorithmus noch zu implementieren und die Resultate mit dem jetzigen zu vergleichen.
Algorithmus 6 wurde nur kurz angeschaut, da er in der Sprache C++ und Python geschrieben wurde. Sollte jemand bereits Wissen dazu besitzen wäre es eine Möglichkeit diesen entweder direkt einzubinden oder auf Java umzuschreiben.
Das Testset ist im Moment noch relativ klein. Eine Erweiterung könnte beinhalten Songs von verschiedenen Genres mit verschiedenen Takt-Instrumenten zu erstellen und zu kategorisieren. Dies würde eine klare Aussage machen, wo der jetzige Algorithmus seine Stärken und Schwächen besitzt.
Im Moment werden die Samples mit der Java Bibliothek JLayer extrahiert, welche im Vergleich zu Matlab ungenauer ist. Eine andere Bibliothek könnte implementiert werden.

Metronom Generierung

Anstatt die Sample-Rate zu verändern, könnte man die Samples direkt verarbeiten und damit die Sample-Rate und den Pitch beibehalten. Hier dafür eignet sich ein Algorithmus namens WSOLA (Waveform Similarity and Overlap Add) [19]. Dieser Algorithmus basiert auf der segmentierten Faltung. Die segmentierte Faltung ist ein Verfahren, welches auf der Fast- Fourier-Transformation basiert [20] .

Anhand dieses Algorithmus ist es möglich den Song und das Metronom zusammen auf verschiedenen Tempos abzuspielen und abzuspeichern. Um dies jedoch zu bewerkstelligen, sollte das Problem mit dem Offset zuerst behoben werden.

Schlusswort

An dieser Stelle möchten wir uns bei Herr Dr. Martin Loeser für die stetige Unterstützung und Betreuung während dieser Projektarbeit bedanken. In den regelmässigen Meetings konnte er uns reichlich an Tipps und Ideen vermitteln und bei Problemen weiterhelfen.

Letztlich ist uns die Erweiterung des Audio Signal Processing Toolboxes zum Teil gelungen. Durch diese Projektarbeit konnten wir vieles Neues sowohl in der digitalen Signalverarbeitung als auch im Bereich Android lernen. Natürlich gibt es viel mehr in diesen beiden Bereichen zum Lernen und das haben wir auch festgestellt, als wir zur Kenntnis nahmen, wieviel Verbesserungspotential in der Beat-Detection und Metronom steckt.

Die Projektarbeit gab uns eine Vorstellung, wie die kommende Bachelorarbeit aussehen könnte und daher konnten wir uns anhand der Projektarbeit optimal auf die Bachelorarbeit vorbereiten. Somit wurde das eigentliche Ziel dieser Arbeit erreicht.

Verzeichnisse

Literaturverzeichnis

[1] M.McGowan. (16.11.2008). *et al*, *Javazoom* [Online] URL: http://www.javazoom.net/javalayer/sources.html [Stand: 20.09.2017]

[2] P. Wendykier. *GitHub* [Online]. URL: https://github.com/wendykierp/JTransforms [Stand: 26.11.2017]

[3] P. Cheung. *Imperial College London* [Online]. URL: http://www.ee.ic.ac.uk/pcheung/teaching/DE2_EE/Lecture%2014-%20Beat%20Detection%20Algorithm%20(slides).pdf [Stand: 10.8.2017]

[4] R.Amorim und S.Mares. (14.10.2017). *The LAME Project* [Online]. URL: http://lame.sourceforge.net/ [Stand: 16.12.2017]

[5] J. Oslud. *metronomer* [Online]. URL: http://metronomer.com/. [Stand: 11.09.2017]

[6] SONG BPM. a blendist cocktail. *songbpm* [Online]. URL: https://songbpm.com/ [Stand: 18.12.2017]

[7] crappyAssDrummer. *youtube* [Online]. URL: https://www.youtube.com/watch?v=vFe0G-38mYs [Stand: 18.9.2017]

[8] SONG BPM. a blendist cocktail. *songbpm* [Online]. URL: https://songbpm.com/maroon-5-sugar [Stand: 18.12.2017]

[9] teddybearssthlmVEVO. *youtube* [Online]. URL: https://www.youtube.com/watch?v=vfW6ONAtWQk [Stand: 5.12.2017]

[10] SONG BPM. a blendist cocktail. *songbpm* [Online]. URL: https://songbpm.com/teddybears-hey [Stand: 18.12.2017]

[11] Beat This. *Beat This, A Beat Synchronization Project* [Online]. URL: https://www.clear.rice.edu/elec301/Projects01/beat_sync/beatalgo.html [Stand: 20.9.2017]

[12] Beat This. *Beat This, A Beat Synchronization Project* [Online]. URL: https://www.clear.rice.edu/elec301/Projects01/beat_sync/images/FinalBlockDiag2.gif [Stand: 20.9.2017]

[13] M. Ziccardi. *mziccard.me* [Online]. URL: http://mziccard.me/2015/05/28/beats-detection-algorithms-1/ [Stand: 20.9.2017]

[14] D. Ellis. *labrosa.colambia.edu.edu* [Online]. URL: https://labrosa.ee.columbia.edu/projects/beattrack/ [Stand: 20.9.2017]

[15] MTG. Music Technology group. *Essentia, Open-source library and tools for audio and music analysis, description and synthesis* [Online]. URL: http://essentia.upf.edu/documentation/reference/std_RhythmExtractor2013.html [Stand: 20.09.2017]

[16] J. R. Zapata, M. E. P. Davies und E. Gomez, *„ Multi-Feature Beat Tracking," IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2014, pp. 816 - 825

[17] F. Patin. *gamedev.net* [Online]. URL:

] http://archive.gamedev.net/archive/reference/programming/features/beatdetection/index.html [Stand: 20.9.2017].

[18] C. Guttandin. *github.com* [Online]. Available: https://github.com/chrisguttandin/web-audio-beat-detector [Stand: 20.9.2017]

[19] M.Schmakeit. *Tempo Adjustment with Waveform Similarity based Overlap-Add (WSOLA)* [Online]. URL: http://hpac.rwth-aachen.de/teaching/sem-mus-16/presentations/Schmakeit.pdf [Stand: 16.12.2017]

[20] *Wikipedia* [Online]. URL: https://de.wikipedia.org/wiki/Segmentierte_Faltung. [Stand: 16.12.2017]

Abbildungsverzeichnis

Abbildung 1 Generierung eines mp3 files als Metronom 3
Abbildung 2 Ablauf energiebasierte Beat-Detection 6
Abbildung 3 Ablauf FFT basierte Beat-Detection..... 8
Abbildung 4 Metronom mit gewünschtem BPM im SoundPool abspielen..... 10
Abbildung 5 Metronom MP3-File Generierung 11
Abbildung 6 Song und Metronom gleichzeitig abspielen 12
Abbildung 7 Ablauf Beat-Detection Algorithmus 1 [12] 15
Abbildung 8 Ablauf angepasster Algorithmus 9 in Java für Android..... 27

Tabellenverzeichnis

Tabelle 1 Übersicht analysierte Algorithmen 14
Tabelle 2 Ergebnisse Algorithmus 1..... 16
Tabelle 3 Ergebnisse Algorithmus 2..... 17
Tabelle 4 Ergebnisse Algorithmus 3..... 18
Tabelle 5 Ergebnisse Algorithmus 4..... 19
Tabelle 6 Ergebnisse Algorithmus 8..... 21
Tabelle 7 Ergebnisse Algorithmus 9..... 22
Tabelle 8 Ergebnisse Algorithmus 10..... 23
Tabelle 9 Ergebnisse Algorithmus 11..... 24
Tabelle 10 Vergleich Resultate Matlab und Android..... 28

Anhang

Glossar

Ausdruck	Synonym(e)	Bedeutung
BPM	beats-per-minute	Tempo eines Musikstücks
FFT	Fast Fourier Transformation	Funktion für Transformation eines Signals aus der Zeit Domäne in die Frequenz Domäne
IFFT	Inverse FFT	Funktion für Transformation eines Signals aus der Frequenz Domäne in die Zeit Domäne
JLayer	-	Java Bibliothek für mp3 file-decoding
JTransform		Java Bibliothek für Fourier Transformationen
IE	Instantaneous energy	Momentane Energie einer Menge von Samples eines Input Signals
VIE	Variance over IE	Varianz über den untersuchten IE Blöcken
MP3	MPEG-1 Audio Layer III	Audio Datei
LAME	-	MP3-Encoder
NDK	Native Development Kit	Benötigt für einbinden von C oder C++ in Android
Pitch		Tonhöhe
WAV-Datei		Audio-Datei
Toast		Android Funktionalität für simples Feedback
Thread	Aktivitätsträger/leichtgewichtiger Prozess	Eine Instanz, die einen Teil des Programms abarbeitet.

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Offizielle Aufgabenstellung

Projektarbeit *PA17_loma_2*
Institut für Informatik

BA-Studenten:	Matthias Kaderli (kadermat) Ravivarnen Sivasothilingam (sivasrav)	Betreuer:	Dr. Martin Loeser
Industriepartner:	-		

Ausgabetermin:	Dienstag, 18. September 2017	Abgabeter-	Freitag 22. Dezember
Anzahl Credits:	6 bzw. ca. 180h	Präsentation:	Form und Zeitpunkt folgen

Thema: Erweiterung einer Audio Signal Processing Toolbox für Android

1. Hintergrund und Zielsetzung

Für viele technische Applikationen ist es von grosser praktischer Bedeutung, Audiosignale direkt auf mobilen Endgeräten wie Tablet-Computern oder Smartphones bearbeiten zu können. Bisher sind wenige open-source libraries verfügbar, die für Geräte, die mit dem Betriebssystem Android laufen, uneingeschränkt geeignet sind.
In einer vorhergehenden Arbeit wurde das Fundament einer solchen Toolbox entworfen. Zum jetzigen Zeitpunkt funktioniert die Toolbox mit .mp3-Dateien. Hier lassen sich in Echtzeit Spektren und Spektrogramme der Audiofiles anzeigen. Zudem sind bereits verschiedene Signalfilter wie Hochpass, Tiefpass, Bandpass, Kammfilter, Distortion Filter etc. implementiert. Auch die Wirkung eines Filters auf das entsprechende Signal lässt sich in Echtzeit visualisieren und natürlich akustisch verifizieren. In dieser Arbeit soll die bestehende App um einige Features erweitert werden – im Fokus steht dabei vor allem die automatische Detektion von Rhythmen.

2. Aufgabenstellung

Ziel dieser Arbeit ist die Erweiterung der vorgängig vorgestellten Audio Signal Processing Toolbox für Android. Dabei sollen unter anderem Algorithmen getestet und implementiert werden, die es erlauben, den Rhythmus von Musikstücken automatisch zu detektieren und auszugeben. Die zu untersuchenden Musikstücke können dabei sowohl lokal gespeicherte Audio-Files, als auch web-basierte Quellen, beispielsweise Spotify-Streams, sein. Insbesondere soll diese Arbeit die folgenden Punkte umfassen:

- Literaturrecherche über gängige Algorithmen zur Beat-Detektion
- Implementieren und Testen verschiedener Algorithmen mit Matlab
- Implementierung der am besten geeigneten Algorithmen in Java, Integration in eine Android-App
- Testen der Android-App mit verschiedenen Audiosignalen (schwerpunktmässig verschiedenen Musikstücken)
- Optional: Erweiterung der Android-App, so dass sowohl lokal gespeicherte Audio-Files als auch web-basierte Audiostreams (z.B. Spotify) mit den selektierten Algorithmen untersucht werden können
- Optional: Metronom zu einem ausgewählten Musikstück erstellen.
- Ziel: Lauffähige Demo-App

3. Bericht/Präsentation/Randbedingungen

- Sie übernehmen die Leitung des Projektes. Sie organisieren, leiten und treffen alle nötigen Massnahmen, um das Projekt zu einem erfolgreichen Abschluss zu führen.
- In der Regel findet alle zwei Wochen eine Besprechung statt. Rechtzeitig vor der Besprechung ist wöchentlich ein kurzer Report per Mail an die Betreuer zu senden, mit präzisen Angaben zu den Berichtspunkten „diese Woche gemacht“, „für nächste Woche geplant“, „Probleme“, etc.
- Über die Projektarbeit ist ein Bericht zu schreiben und in passender Form termingerecht abzugeben. Es gelten die üblichen Dokumente zum Ablauf und zur Form (Leitfaden_PABA.pdf, Zitierleitfaden, etc.).

4. Bewertungskriterien, Gewichtung:

Projektverlauf, Leistung, Arbeitsverhalten ca. ⅓; Qualität der Ergebnisse ca. ⅓; Form und Inhalt des Berichts ca. ⅓.