Bachelorarbeit

Hochschule
Augsburg University of
Applied Sciences

Studienrichtung Informatik

Michael Wager

Entwicklung eines Systems zur automatischen Notentranskription von monophonischem Audiomaterial.

Erstprüfer
Prof. Dr. Thomas Rist,
Hochschule Augsburg

Zweitprüfer
Prof. Dr.-Ing. Christian Märtin,
Hochschule Augsburg

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Hochschule Augsburg
University of Applied Sciences
Baumgartnerstraße 16
D 86161 Augsburg

Telefon +49 821 5586-0
Fax +49 821 5586-3222
http://www.hs-augsburg.de
poststelle@hs-augsburg.de

Erstellungserklärung

Hiermit erkläre ich, dass ich die vorgelegte Arbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel verwendet, keine Urheberrechtsschutz-Verletzungen begangen sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Michael Wager

Inhaltsverzeichnis

Zusammenfassung

Diese Arbeit befasst sich mit dem Problem der automatischen Notentranskription von monophonischem Audiomaterial. Die automatisierte Transkription einer einstimmigen Melodie bringt viele Vorteile für alle Arten von Musikern mit sich. Einerseits nimmt Sie eine Menge Arbeit ab. Gerade viele Jazzmusiker müssen aufgrund der Notenknappheit immer wieder Transkriptionen von anderen Liedern anfertigen, was ein gutes Gehör und viel Geduld erfordert. Andererseits wäre es ein großer Vorteil für Anfänger um ihr Spiel auf Genaugigkeit zu analysieren.

Die vorgestellte Pipeline basiert hauptsächlich auf zwei Algorithmen zur Erkennung der fundamentalen Frequenz in einem zeit-diskreten Signal, YIN und MPM, sowie einem Notenbasierenden Onset-Kollektor.

Es wurde ein Prototyp entwickelt, das System wurde evaluiert und die Ergebnisse sind gut. Eine durchschnittliche Notenerkennungsrate von 79.58% wurde erreicht, was den praktischen Einsatz einer solchen Anwendung ermöglicht.

1 Einleitung

1.1 Motivation

Die Idee dieser Arbeit entwickelte sich vor allem dadurch, dass Ideen unter Musikern auszutauschen sehr aufwendig sein kann. Ein gängiges Vorgehen ist eine Melodie anderen Musikern vorzuspielen. Jedoch ist dies immer schwierig und mühevoll, besonders für Musiker welche nicht allzu viel mit Notenschrift zu tun haben, wie zB viele Gitarristen. Am Besten ist es natürlich zum Beispiel eine Melodie für Klarinette dem Klarinettisten direkt in Form von Noten zu geben, da Notenschrift alle relevanten Informationen repräsentiert. Dafür ist es jedoch notwendig, zuerst eine Transkription der Melodie anzufertigen.
Zusätzlich äußerst zeitaufwendig ist das sogenannte Transponieren. Bestimmte Musikinstrumente wie beispielsweise die Bb-Klarinette sind transponierende Instrumente, d.h. sie klingen anders als notiert. Daraus folgt, wenn eine Bb-Klarinette ein C spielt, entspricht die Tonhöhe, also die fundamentale Frequenz aber einem Bb bzw. b. Somit braucht man für ein Gitarren-Klarinetten Duett zwei verschiedene Versionen der Transkription, einmal für Gitarre und einmal für die Klarinette.

Wäre es nun möglich aus aufgenommenem, monophonischem[1] Audiomaterial automatisiert Noten zu generieren, wäre dies eine große Erleichterung für Musiker um Ideen auszutauschen, transponierte Versionen davon an andere Instrumentenspieler verteilen zu können ohne sie selbst erst "übersetzen" zu müssen, aber auch für Lehrer um Schülern kleine Übungen einzuspielen und Anfänger, um ihr Spiel und ihre Tonhöhengenauigkeit zu kontrollieren. Man spielt "einfach" die Melodie ein und erhält die dazugehörigen Noten.

Ein weiterer großer Vorteil wäre die Erleichterung des Einstiegs in die Notenschrift für Anfänger. Dadurch, dass das Gespielte direkt in Notenschrift umgesetzt wird, hat der Anfänger die Möglichkeit Noten schneller zu erlernen. In dieser Arbeit soll versucht werden die automatische Transkription in Echtzeit zu realisieren, da es gerade für Anfänger äußerst nützlich wäre, die Noten zu sehn *während* man spielt.

1.2 Automatische Transkription

Transkription von Musik bedeutet, eine Melodie oder ein musikalisches Stück in eine symbolische, vom Menschen lesbare Repräsentation zu bringen, die Notenschrift.
Automatische Transkription bezieht sich auf das Verfahren, computergestützt automatisiert Notenschrift zu generieren. Dafür ist es notwendig ein akustisches Musiksignal blockweise zu verarbeiten und die relevanten Informationen, hier Tonhöhe und Tonlänge, herauszufiltern. Dies ist ein äußerst komplexer Prozess, welcher näheres Wissen bezüglich physikalischen Grundlagen wie Schwingung oder Frequenz und der Eigenschaften von Tönen erfordert. Dazu mehr in Kapitel 2.
Die Abbildungen 1 und 2 zeigen den Plot der ersten paar Takte einer Melodie gespielt von einer Klarinette und ihr dazugehöriges Notenbild. Dies verdeutlicht die grundlegende Anforderung an ein automatisches Notentranskriptionssystem. Abbildung 1 zeigt den Plot einer Aufnahme einer Klarinette ca. 25 Sekunden lang, Abbildung 2 die gewünschte Ausgabe in Form moderner Notenschrift. Man hat also eine Aufnahme eines einstimmigen Audiosignals und möchte mit Hilfe eines computergestützten Verfahrens daraus die Noten generieren.

[1] Einstimmigkeit, also das Vorhandensein nur einer einzigen Stimme

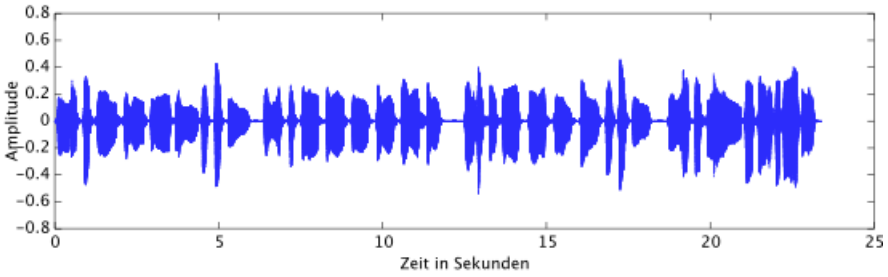

Abbildung 1: Plot der ersten Takte einer Melodie, gespielt mit einer Klarinette

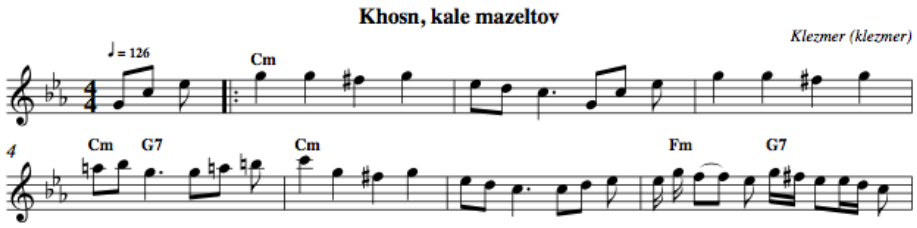

Abbildung 2: Das dazugehörige Notenbild

1.3 Ziel dieser Arbeit

Das Ziel dieser Arbeit ist die automatische Transkription einer monophonischen, akustischen Aufnahme. Ich werde eine Pipeline vorstellen, welche alle zu erledigenden Aufgaben hinreichend erfüllt um den praktischen Einsatz solch einer Anwendung zu ermöglichen und das Ganze in Form einer Implementierung eines Prototyps abschließen.
Es gibt eine Menge weiterer Anwendungsgebiete der automatischen Transkription einer Audioaufnahme, weshalb großer Wert auf Modularität bzw. Erweiterbarkeit der zu entwickelnden Software gelegt wird, um einzelne Komponenten später auszutauschen oder zu portieren. Dazu mehr in Kapitel 5.
Als Namen für die Anwendung entschied ich mich für *"jAM - (java) automatic music transcription"*, zum Einen weil ich es in der Programmiersprache Java implementieren werde. Abgesehen davon steht der Begriff "jam" (jamsession, jammen) im Musikerjargon für spontanes Musizieren, und da mein System vor Allem für die Erstellung von Notenschrift aus spontanen Ideen entwickelt wird, viel es mir nicht allzu schwer auf einen geeigneten Namen zu kommen.

2 Grundlegende Begriffe und Definitionen

2.1 Physikalische Begriffe

2.1.1 Schwingung

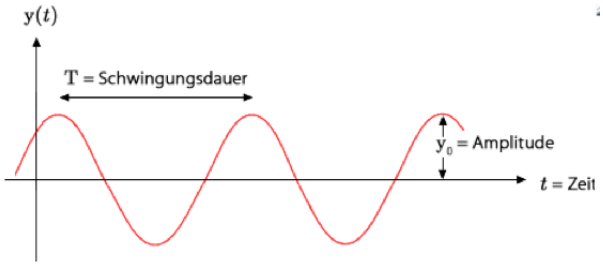

Abbildung 3: Darstellung des zeitlichen Verlaufs der Größe y(t) bei einer harmonischen Schwingung. [Bild1]

Schwingung (auch Oszillation) ist der Begriff für einen Vorgang, genauer den Verlauf einer Zustandsänderung, welcher sich in einem bestimmten Zeitinterval wiederholt. Diese Wiederholungen können periodisch sein, wie im Falle einer harmonischen Schwingung, oder nicht. Abbildung 3 zeigt eine harmonische Schwingung über der Zeit. Die Werte der Funktion bestimmen die Auslenkung. Die Schwingungsdauer ist mit T gegeben. Grob gesagt gleicht eine Schwingung einem doppeltem Nulldurchgang, also einmal Positiv, einmal Negativ. Der Kehrwert davon ist die Frequenz:

$$f = \frac{1}{T}$$

2.1.2 Frequenz

Um periodische Vorgänge wie zum Beispiel die Schwingung zu Beschreiben ist es wichtig eine physikalische Größe zu besitzen, welche die Anzahl sich wiederholenden Vorgängen pro Zeiteinheit angibt. Die am Häufigsten angewandte Größe ist wohl Hertz:

$$1Hz = \frac{1}{s}$$

Diese gibt an wie oft sich ein Vorgang pro Sekunde wiederholt. Wenn wir musikalische Töne hören, so nehmen wir diese als höher war wenn deren Frequenz steigt, tiefe Töne haben eine niedrigere Frequenz. Das menschliche Ohr nimmt Schallwellen mit Frequenzen zwischen 20 Hz und 20.000 Hz wahr, wobei die Obergrenze mit zunehmendem Alter abnimmt.
Kennt man also die Frequenz eines harmonischen Signals, so kann man sagen welcher musikalische Ton, also welche Note, dieser Frequenz entspricht. Dazu ist es nur noch notwendig, sich auf eine Referenzfrequenz zu beziehen. Dies ist der sogenannte Kammerton, also der gemeinsame Ton, auf den die Instrumente einer Musikgruppe eingestimmt werden. Die Kammerton-Frequenz beträgt heutzutage in der Regel 440 Hz und entspricht dem Ton A4.
Daraus folgt, dass ein automatisches Transkriptionssystem zuerst die Frequenz des in der Aufnahme enthaltenen Signals zu einem bestimmten Zeitpunkt erkennen muss um somit Entscheidungen über die gespielten Noten treffen zu können.

2.2 Musikalische Begriffe

2.2.1 Das Notensystem

Um musikalische Abläufe zwischen Musikern erklären zu können ist es notwendig sich auf Symbole und Zeichen zu einigen [Haunschild98]. Grundlage der westlichen Notenschrift sind 5 Linien, auf denen Notenzeichen ihrer Tonhöhe entsprechend entweder auf oder zwischen den Linien gezeichnet werden. Mit der Platzierung einer Note kann man ihre *Tonhöhe* angeben. Zusätzlich gibt ein Notenschlüssel an wo ein bestimmter Ton liegt. Der g-Schlüssel beispielsweise, auch Violinschlüssel genannt gibt an, dass die Note g auf der zweiten Linie von unten liegen muss. Dadurch ist klargestellt, wo sich welche Note befindet. Abbildung 4 zeigt ein Notensystem für Violin- und Bassschlüssel. Es gibt noch mehr solcher Notenschlüssel, die gebräuchlichsten sind wohl der Violin- und Bassschlüssel. In dieser Arbeit soll nur Notenschrift für diese beiden Schlüssel generiert werden.

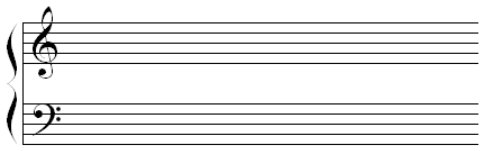

Abbildung 4: Zwei fünflinige Notensysteme im Violin- und Bassschlüssel [Bild3]

2.2.2 Noten

Das Alphabet von a bis g wird verwendet um Noten ihren Namen zu geben:

a b c d e f g a

Eine Oktave besteht im westlichen System aus 12 Halbtönen. Grundlegend lassen sich Noten durch 3 Parameter charakterisieren:

- Tonhöhe (Fundamentale Frequenz)
- Tondauer (Zeitintervall)
- Tondynamik (Lautstärke)

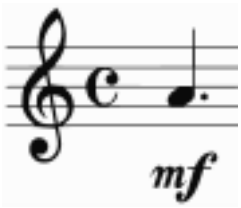

Abbildung 5: Punktierte Viertelnote, A4, mezzoforte(=natürliche Lautstärke des Instruments) [Bild4]

Wie man die Tonhöhe angibt wissen wir nun, jedoch nicht den Notenwert bzw. die *Tonlänge*. Der Notenwert gibt an wie lange ein Ton gehalten werden soll, also schwingt. Ein Ganzton beispielsweise besagt dass der Ton einen Takt, d.h. 4 Schläge bei einem 4/4 - Takt erklingen soll.

Abbildung 5 zeigt die punktierte Viertelnote A4 im Violinschlüssel mit der Dynamik “mezzoforze”. Der Punkt gibt an, dass die Länge der Note um ihre Hälfte verlängert wird, also eine Viertel plus eine Achtel. In dieser Arbeit wird auf die Dynamik einer Note kein Wert gelegt. Relevant sind nur Höhe und Länge der Note.
Die zwei wichtigsten Anwendungsfälle der Notenschrift sind einmal das Festhalten einer Komposition zur Überlieferung. Gäbe es keine Notation, würden Werke von berühmten Komponisten wie beispielsweise Johann Sebastian Bach heute nicht existieren.
Ein weiterer Nutzen der Notenschrift ist es, Ideen oder musikalische Einfälle schriftlich festzuhalten, um somit anderen Musikern zugänglich machen zu können, ohne dies selbst vorführen zu müssen.
Jedoch ist das Transkribieren der Musik immer ein mühevoller, gedanklich anspruchsvoller Prozess, wodurch die Idee naheliegend ist, das Ganze computerunterstützt zu automatisieren.

2.2.3 Die Obertonreihe

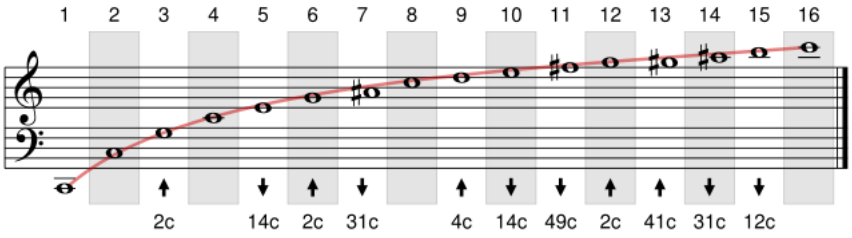

Abbildung 6: Die Obertonreihe vom Grundton C [Sikora03]

Ein wichtiges natürliches Klangphänomen ist die Obertonreihe [Sikora03]. Was wir als einen einzigen Ton wahrnehmen, ist in Wahrheit ein Zusammenklang verschiedener, sich überlagernder Schwingungen, einem Grund- bzw. Fundamentalton (“der am stärksten empfundene Sound”) und einer über diesem liegenden Reihe von Partialtönen, welche in Zusammenhang mit dem Grundton stehen, die sogenannten harmonischen Obertöne oder Naturtöne. Es entsteht eine theoretisch unendliche mathematische Reihe von Frequenzverhältnissen:

1:2:3:4:5:6:7 usw...

Abbildung 6 zeigt diese Verhältnisse vom Grundton C aus. Die Obertöne klingen umso leiser mit, je weiter sie vom Grundton entfernt sind. Das ist eine der Schwierigkeiten beim Erkennen der fundamentalen Frequenz. Manchmal sind die Obertöne an gewissen Stellen dominanter als der Grundton, was zu Fehlern in der generierten Transkription führen würde.

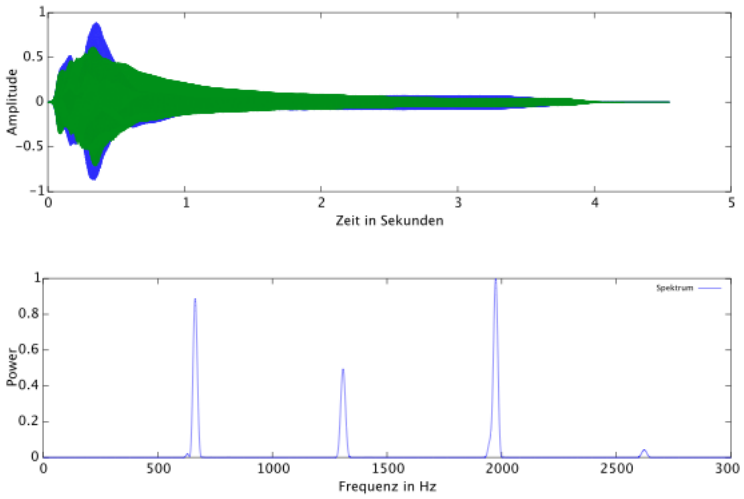

Abbildung 7: Plot des Tons A4, gespielt mit einer Klarinette, und Spektrum

Abbildung 7 verdeutlicht dieses Phänomen anhand zweier Graphen. Graph 1 zeigt einen einzelnen Ton auf der Klarinette: Amplitude über der Zeit in Sekunden. Graph 2 zeigt das normalisierte Spektrum über der Frequenz. Man kann sehen, dass die dominanteste Frequenz bei ca. 2000 liegt, diese jedoch nicht auf den Grundton schließen lässt, sondern auf einen der Obertöne, nämlich 1328.3Hz. Dies entspricht ungefähr dem 3. Oberton, also der Note E 6 mit einer Abweichung von ca. 12.80 cents [2]. Der richtige Ton, also der vom menschlichen Gehör am Stärksten wahrgenommene, beträgt jedoch 440Hz und entspricht der Note A4.

2.2.4 Unterstützter Frequenzbereich

Abbildung 8 zeigt den Frequenzbereich eines Pianos. Der unterstützte Frequenzbereich in dieser Arbeit wurde folgendermaßen eingeschränkt: Von C2 (65.41Hz) bis C6 (1046.50Hz)

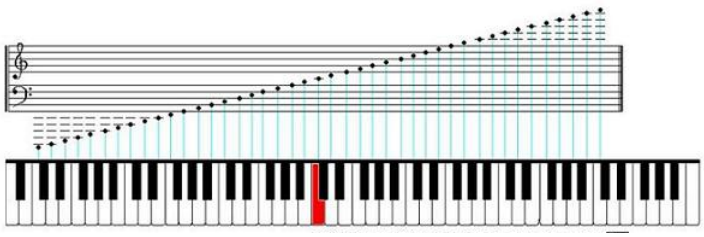

Abbildung 8: Frequenzbereich eines Pianos [Bild2]

[2] Cent - ein Hundertstel eines Halbtones oder Zwölfhundertstel einer Oktave

2.2.5 Unterschiede in der Klangerzeugung verschiedener Instrumentenfamilien

Nachfolgend soll ein grober Überblick über Instrumentenfamilien sowie deren Klangeigenschaften gegeben werden. Grundlegend lassen sich nicht-perkussive [3] Instrumente in 3 Gruppen einteilen: [Martin99]

1. Blechblasinstrumente
2. Streicher
3. Holzblasinstrumente

Zu den Blechblasinstrumenten gehören u.a. das Horn, die Trompete, das Flügelhorn, die Posaune und die Tuba. Der Ton wird durch Blasen in ein Mundstück erzeugt. Das interne Spektrum eines Blasinstruments variiert mit dem Luftdruck, erzeugt durch die Lippen des Spielers. Bei schwachen Amplituden ist das Signal annähernd sinusartig, bei starker Amplitudenauslenkung mehr pulsartig. In der Attackphase variiert die Frequenz, was besonders gute Spieler durch präzise Kontrolle über ihre Lippenspannung ausgleichen. Der Schallbecher reflektiert tiefe Frequenzenergie stärker als hohe woraus folgt, dass Onsets[4] besser erkannt werden können, bzw. Oktavfehler weniger häufig auftreten. Instrumente der Blechblasfamilie haben viel gemeinsam. Die Unterschiede liegen hauptsächlich im Frequenzbereich.

Zu den Streichern zählen in der Regel die Violine, die Viola, das Cello und der Kontrabass. Jedes Streichinstrument ist aus Holz gefertigt. Werden die Saiten nun mit Hilfe eines Bogens zum Vibrieren gebracht, so schwingt der gesamte Körper und es erklingt ein Ton. Attack- und Releasephase eines durch Streichen der Saite erzeugten Tones sind komplex und das Spektrum einer gezupften Saite ist nie harmonisch. Dies trifft auch auf Gitarren zu. Gerade in der Attackphase lassen sich starke Phasenüberlagerungen beobachten.

Holzblasinstrumente lassen sich grob weiter in Untergruppen aufteilen: Einzelblatt, Doppelblatt, Flöten und Saxophone. Wichtige Instrumente sind u.a. die Flöten, Klarinetten, Oboen, Fagotte und das Saxophon. Der Spieler erzeugt den Ton, indem er in ein Mundstück bläst. Es entstehen Wellen im Rohr, die Tonhöhe ist durch die Länge der Luftsäule im Instrument gegeben. Durch Öffnen und Schließen von Tonlöchern und Tonklappen kann die Tonhöhe beeinflusst werden.

Abbildung 9 zeigt den Plot des Tones C4, aufgenommen von 3 Instrumenten: Einer Gitarre mit Nylonsaiten, einer Bb-Klarinette und einer Violine. Es sind deutliche Unterschiede in den Wellen zu sehen. Offensichtlich ist, dass die Releasephase, also das Abklingen des Tons bei der Gitarre deutlich schneller abnimmt als bei den anderen Beiden Instrumenten. Eine Gitarrensaite wird einmal angeschlagen und klingt dann ab, wohingegen bei Geige und Klarinette der Ton theoretisch unendlich lang gespielt werden kann.
Wie oben schon erwähnt ist das Spektrum von Streichinstrumenten nie harmonisch. Deutlich zu sehen ist, dass bei Geige und Gitarre das Signal in der Sustainphase zwar gleichmäßig zu sein scheint, jedoch sind deutlich Überlagerungen zu beobachten, wohingegen das Signal der Klarinette am sinusartigsten ist. Auch bei dem kurzen Ausschnitt aus dem Spektrum der Violine in Abbildung 10 ist ein Oberton mehr zu sehen.

[3] perkussiv: rhytmisch, den Rhymus angebend
[4] Onset: Der Beginn eines Tons

In dieser Arbeit wird bzgl. der Instrumentenfamilie nicht unterschieden. Die hier zum Einsatz kommenden PitchDetektoren zeigen gute Ergebnisse bei allen Familien. Relevante Unterschiede in der erzeugten Transkription können allerdings in den Parametereinstellungen pro Familie liegen. So sind zum Beispiel kleinere Buffergrößen besser für Holzblasinstrumente geeignet. Die Evaluation dieses Systems soll zusätzlich instrumentenbasiert durchgeführt werden, wodurch es möglich ist, Entscheidungen bzgl. vorteilhaften instrumentenfamilienspezifischen Parametereinstellungen zu treffen.

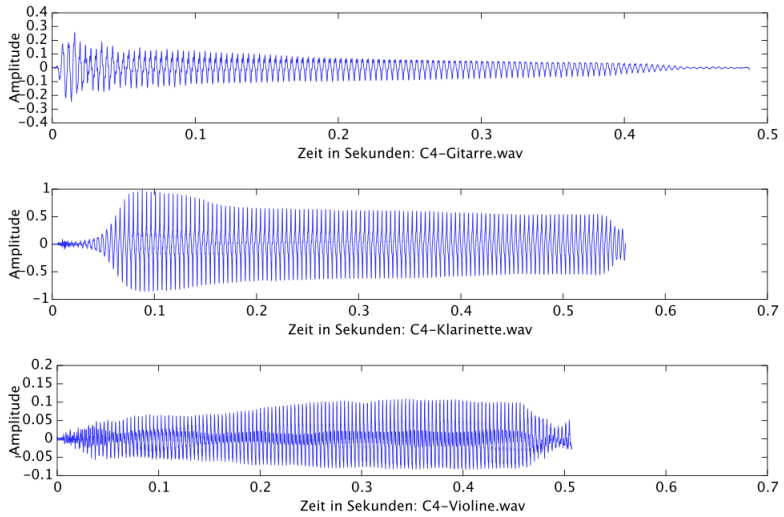

Abbildung 9: Plot des Tons C4, gespielt von einer Akustikgitarre, einer Klarinette und einer Geige

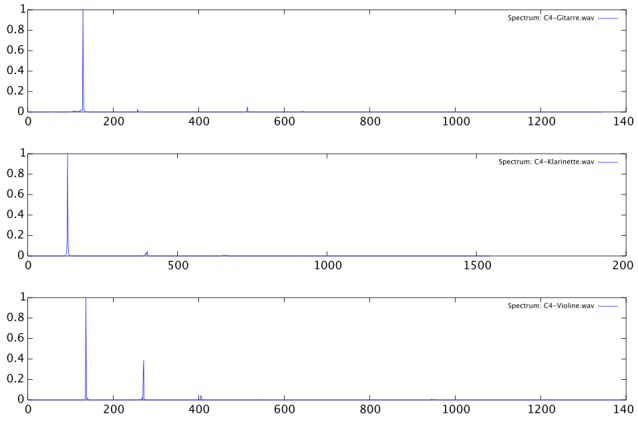

Abbildung 10: Spektren der oben gezeigten drei Töne, nach Maximum normalisiert

2.3 Digitale Audiosignalverarbeitung

2.3.1 Sampled Audio

Samples sind fortlaufende Ausschnitte eines Signals. Im Falle von Audio ist das Signal eine Schwingung. Ein Mikrofon konvertiert das Signal in ein analoges, elektrisches Signal, und ein Analog zu Digital Converter transformiert das analoge Signal in eine abgetastete Digitale Form. Abbildung 11 zeigt einen kurzen Ausschnit einer musikalischen Aufnahme. Der Graph plottet auf der vertikalen Achse die Amplitude, auf der Horizontalen die Zeit. Die blaue Linie zeigt das abzutastende Signal, die roten Punkte repräsentieren die Abtastungen.

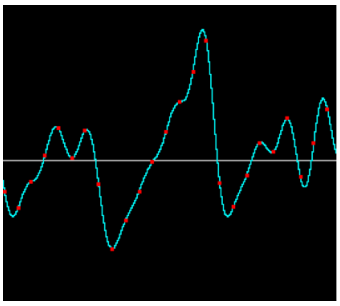

Abbildung 11: Ausschnitt eines abgetasteten, akustischen Signals [JavaSoundAPI]

2.3.2 Die Abtastrate

Ein Audiosignal (zum Beispiel eine Aufnahme eines Musikinstruments) wird anhand einer gegebenen Abtastrate, die sogenannte Samplerate oder Abtastfrequenz abgetastet. Diese Abtastfrequenz gibt an wieviele Abtastungen pro Sekunde stattfinden. Eine gängige Samplerate ist bei CD Qualität 44.1kHz, d.h. 44100 Abtastungen finden pro Sekunde statt. Aufgrund des sogenannten Abtasttheorems muss die Abtastrate mindestens doppelt so hoch sein wie die höchste zu erkennende Frequenz im Signal [Meffert04]. Möchte man ein Signal mit einer Grundfrequenz von 4000Hz reproduzieren, muss die Abtastfrequenz mindestens 8000Hz betragen.
Die Abtastrate in dieser Arbeit beträgt immer 44.1 kHz. Es findet kein downsampling statt. Parameter wie Buffersize, oder Buffer-Überlappung (overlap) sind über die graphische Benutzeroberfläche konfigurierbar.

3 Recherche - Automatische Notentranskription

3.1 Aktueller Stand

Automatische Transkription von monophonischem Audiomaterial gilt bereits als gelöst. Kommerzielle Anwendungen wurden bereits veröffentlicht. Im Folgenden werden einige Anwendungen vorgestellt, dessen Genauigkeit untersucht und erläutert warum ein neuer Ansatz überhaupt in Angriff genommen werden sollte.

3.2 Logic Pro's "Audio to Score" Feauture

Logic Pro [LOGIC] ist eine von Apple entwickelte, professionelle Audioanwendung. Sie besitzt eine Funktion namens "Audio to Score", welche es ermöglicht, aus einer monophonischen Aufnahme Notenschrift generieren zu lassen.
Abbildung 12 zeigt einen Ausschnitt aus der von Logic Pro generierten Transkription des Spirituals "Joshua fought the battle of jericho", gespielt auf einer akustischen Gitarre, Marke Alhambra, Modell S7C mit Nylonsaiten. Die Aufnahme wurde mit einem Shure SM11 gemacht. Man kann sehen, dass die Erkennungsrate doch sehr zu wünschen übrig lässt, obwohl die Aufnahme frei von Störgeräuschen ist. Als Hinweis steht bei dem Feature "Please make sure to process clean monophonic material".
Ein Editieren der Noten im Nachhinein und sogar eine saubere Ausgabe im PDF Format ist jedoch möglich. Das Problem ist, dass diese Anwendung sehr teuer und nur für MAC OS X erhältlich ist.

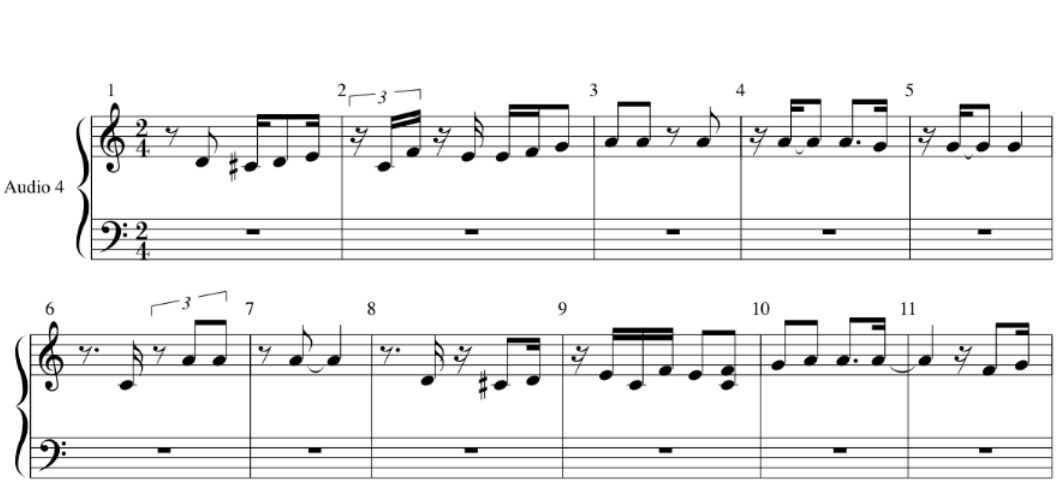

Abbildung 12: Beginn der von Logic Pro generierten Notenschrift für "Joshua fought the battle of jericho", gespielt auf einer akustischen Gitarre

3.3 Intelliscore

IntelliScore[INTELLISCORE] ist eine Anwendung, welche die Konvertierung von Audiodateien in MIDI[5] unterstützt. Hat man eine MIDI Datei, kann man daraus ganz einfach eine Transkription generieren lassen. Dafür ist dann allerdings zusätzliche Software notwendig. Die Ausgabe in MIDI ist relativ gut, jedoch unterstützt die Demoversion nur die ersten 30 Sekunden einer Aufnahme.
Auch diese Anwendung ist nur kostenpflichtig erhältlich und läuft ausschließlich unter dem Betriebssystem Windows.

3.4 Melodyne

Melodyne Editor [MELODYNE] ist eine Audiobearbeitungssoftware von Celemony Software, welche es ermöglicht durch ihre sogenannte "Direct Note Access-Technologie" Zugriff auf einzelne Noten in einstimmigem und mehrstimmigem Audiomaterial zu haben und diese zu manipulieren. Vor der Veröffentlichung galt das Editieren von Noten in mehrstimmigem Audiomaterial als unmöglich, und Melodyne sorgte für großes Aufsehen. Nach einigen Experimenten mit der TrialVersion stellte sich heraus, dass diese Anwendung äußerst akkurat ist, auch bei Aufnahmen mit viel Rauschen und schlechten Mikrofonen.
Jedoch ist auch Melodyne äußerst teuer und auf die Notenlängen wird in der erstellten Transkription scheinbar kein Wert gelegt.

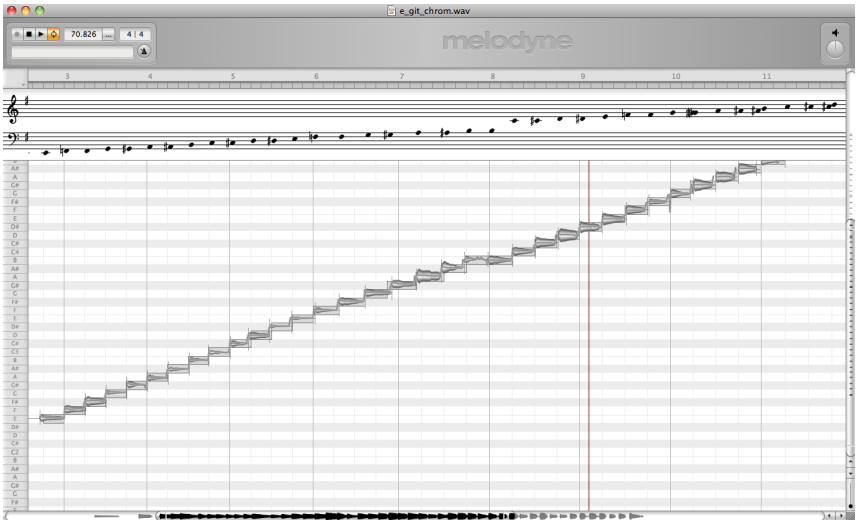

Abbildung 13: Screenshot des Melodyne Hauptfensters, Aufnahme einer E-Gitarre, chromatische Tonleiter

[5] MIDI - Musical Instrument Digital Interface

3.5 Tartini

Tartini [McLeod-2]ist ein freies und quelloffenes Musik Analyse Tool für Sänger und Instrumentalisten. Folgende Features sind implementiert:

- Echtzeit Pitch Kontur um Intonation, Vibrato oder die Note, welche gespielt wird, zu visualisieren
- Lautstärkevisualisation
- Harmonische Struktur einer Note

Tartini ist quelloffen, frei erhältlich und im Rahmen dieser Recherche wohl die Anwendung, welche die gegebenen Anforderungen am Besten erfüllt. Man kann aufnehmen und erhält Echtzeitfeedback über Intonation, Vibrato etc. Zusätzlich kann man sich dann eine Transkription der Aufnahme in Notenschrift anzeigen lassen. Schade ist, dass man keine Möglichkeit hat diese Transkription zu exportieren, sei es als MIDI, Bild oder Ähnliches. Zusätzlich ist das Äußere dieser Transkription nicht besonders ansprechend. Der Fokus dieser Anwendung liegt wohl eher in der Analyse von Intonation oder Vibrato als in der Transkription. Teile des darunterliegenden PitchDetection Algorithmus kommen jedoch u.a. in dieser Arbeit zum Einsatz. Siehe dazu Kapitel 4.

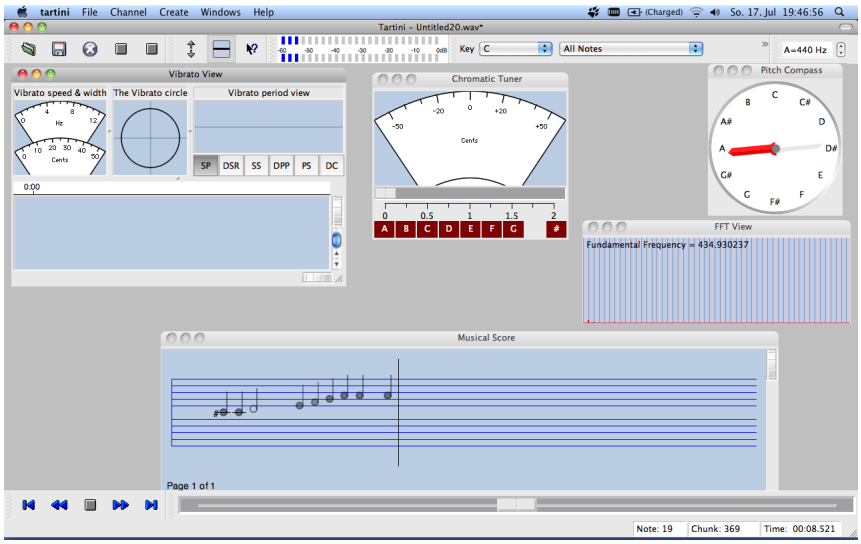

Abbildung 14: Screenshot des Tartini Hauptfensters

3.6 Zusammenfassung - Warum ein neuer Ansatz?

Mein Versuch zielt nicht darauf ab eine professionelle, studiotaugliche Anwendung zu entwickeln, sondern eher ein schlichtes, frei erhältliches Tool für alle Arten von Musiker um das schnelle Festhalten von Ideen, Übungen etc. ohne großen Aufwand zu ermöglichen. Musiker sollen die Möglichkeit haben einfach und schnell, ohne große Einarbeitungszeit das Programm zu nutzen und eine akzeptable Ausgabe in Form moderner Notenschrift zu erhalten. Fehler im Transkriptionsprozess und somit in der angezeigten Notenschrift können im Nachhinein unter Kenntnis der ABC-Notationssyntax editiert bzw. ausgebessert werden um somit eine akzeptable Transkription zu erhalten.

Die oben vorgestellten Anwendungen wurden von mir untersucht und getestet und erreichen entweder diese Anforderungen nicht akzeptabel genug oder sind zu teuer. Sehr viele Musiker haben nicht die finanziellen Möglichkeiten sich so etwas zu leisten. Auch konnte ich keine einzige Anwendung finden welche meine Anforderungen in Echtzeit erfüllt und frei zugänglich und erhältlich ist.

Durch den Dialog mit einigen Musikern vom Amateur bis zum Musiklehrer erkannte ich, dass reges Interesse an solch einer Anwendung besteht und zu guter Letzt durch den Bedarf meinerseits habe ich mich entschieden dieses Thema als meine Bachelorarbeit zu wählen.

4 jAM - Ein System zur automatischen Notentranskription von monophonischem Audiomaterial

Dieses Kapitel stellt die entwickelte Pipeline vor. Zuerst werden die grundlegenden Schritte eines automatischen Transkriptionssystems und daraufhin grob die gebräuchlichsten Methoden zur Erkennung der fundamentalen Frequenz in einem zeitdiskreten Signal vorgestellt. Danach wird erläutert, welche Methoden in dieser Arbeit zum Einsatz kommen und anschließend gezeigt wie Notenwerte in dieser Arbeit basierend auf sogenannten Noten On- bzw. Offsets erkannt werden.

4.1 Entwurf der Pipeline

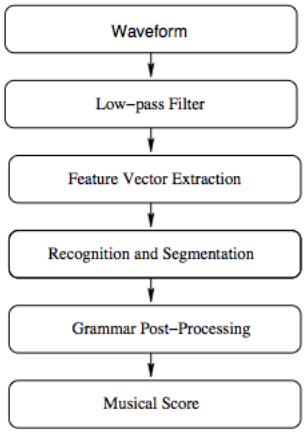

Abbildung 15: Grundlegende Pipeline eines automatischen Transkriptionssystems [Krige]

Abbildung 15 zeigt eine Skizze der grundlegenden Schritte eines automatischen Transkriptionssystems. Jeder dieser Schritte wird nun einleitend vorgestellt und erläutert, ob und warum er in dieser Arbeit eine Rolle spielt.

4.1.1 Waveform

Zuerst muss eine Audiodatei geladen werden, welche in mehreren Formaten vorliegen kann. Das WAVE-Format ist ein gebräuchliches Dateiformat zur digitalen Speicherung von Audiodaten, welches die Abtastwerte unkomprimiert speichert. Es ist also ein verlustfreies Format wohingegen beispielsweise das MP3-Format aufgrund der Komprimierung verlustbehaftet ist.
In dieser Arbeit werden ausschließlich WAV-Dateien mit den folgenden Eigenschaften unterstützt:

1. Abtastrate 44.1 kHz
2. PCM-Signed
3. Samplesize 16 Bit
4. mono
5. little-endian

Die Aufnahme muss monophonisch sein und sollte möglichst frei von Störfaktoren wie Umgebungsgeräuschen, Metronomklicks o. Ä. sein.

4.1.2 Low-pass Filter

Ein Tiefpass ist ein digitaler Filter, welcher Signalanteile mit Frequenzen unterhalb einer Grenzfrequenz annähernd ungeschwächt passieren lässt, Anteile mit höheren Frequenzen dagegen abschwächt. Somit ist solch ein Filter nützlich um die zu hohen Frequenzen zu filtern um zum Beispiel durch dominante Obertöne verursachte Oktavfehler in der Frequenzerkennung zu reduzieren.
Die TarsosLib enthält einen einfachen Low-pass Filter, welcher zugeschaltet werden kann, jedoch ist dieser Schritt bei PitchDetektoren wie der McLeod PitchMethod nicht notwendig. Dazu mehr in Kapitel 4.3.

4.1.3 Feature Vektor Extraktion

Die relevanten Features sind in dieser Arbeit die Amplitudenlevel (RootMeanSquared) und die Pitches, also die fundamentalen Frequenzen. Diese Features werden pro Buffer extrahiert. Unterstützte Buffersizes sind 512, 1024, 2048 und 4096 Byte mit jeweils 0, 10, 25, 50 und 75% Bufferoverlap bei einer stets gleichbleibenden Abtastrate von 44.1 kHz. Nachdem ein Buffer von der Soundkarte gelesen wurde, wird dieser zuerst einer ByteToFloat-Konvertierung unterzogen. Danach wird der Effektivwert pro Buffer und dann je nach Einstellung entweder per YIN oder MPM die Frequenz berechnet.

4.1.4 Segmentierung

Die Segmentierung der Buffer bzw. der Pitchsequenzen findet im Collector-Prozess statt. Dazu sind zwei zusätzliche Parameter notwendig: Die minimale Zeit, welche als Bedingung für den Beginn eines Tones steht, auch Minimumduration, sowie das minimale Level. Der zuvor berechnete Effektivwert muss diesen Wert übersteigen um als Ton zugelassen zu werden. Dazu mehr in Kapitel 4.4 und 4.5.

4.1.5 Grammar Post-Processing

Nachdem ein Ton erkannt wurde, müssen noch einige Bedingungen musiktheoretischer Art beachtet werden. Diese sind hier die Tonart und die Taktart welche das System vorher kennen muss.
Weiß das System zum Beispiel, dass wir uns in der Tonart G-Dur befinden (Ein #-Vorzeichen, F -> Fis), so muss jedes Mal wenn ein Fis erkannt wird aber ein F gemalt werden, da das Vorzeichen am Anfang der Zeile ja angibt das aus jedem F automatisch ein Fis wird. Intern jedoch wird das Fis gehalten bzw. der MidiSchlüssel um dennoch richtig zu evaluieren.

4.1.6 Notenschrift

Zu guter Letzt muss natürlich die Ausgabe in Form einer der modernen Notenschrift entsprechenden Transkription erfolgen. Zu diesem Zweck kommt in dieser Arbeit der textbasierende Notenstandard *abcnotation* zum Einsatz. Dazu mehr in Kapitel 4.5.

4.2 Methoden zur Erkennung der fundamentalen Frequenz

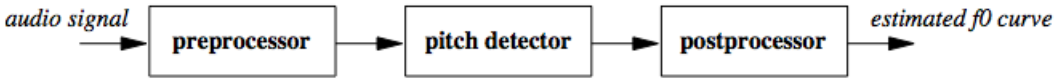

Abbildung 16: Grundlegende Schritte zur Erkennung der fundamentalen Frequenz [Ryynänen1]

Automatische Transkription einer Melodie ist wahrscheinlich der offensichtlichste Anwendungsfall für die Erkennung der fundamentalen Frequenz in einem Audiosignal [Savard] und PitchDetection, also die Erkennung der fundamentalen Frequenz in einem zeitdiskreten Signal ist wohl die wichtigste Aufgabe eines automatischen Transkriptionssystems. Es existieren zahlreiche Methoden hierzu. Im Folgenden wird ein grober Überblick gegeben und anschließend erläutert welche Methoden hier zum Einsatz kommen werden.
Grundlegend lassen sich diese Methoden in zwei Bereiche einteilen: Zeit- und Frequenzbereich.

4.2.1 Zeitbereich

Methoden des Zeitbereich sind unter Anderen ZeroCrossing und Autokorrelation. Beim ZeroCrossing geht es wie der Name schon sagt darum, Nulldurchgänge im Signal zu zählen und diese ins Verhältnis zur Abtastrate zu setzen um die Frequenz zu erhalten. Dieses Verfahren ist äußerst leicht zu implementieren, jedoch nicht akzeptabel genug in komplexen Signalen, welche mehrere Sinuswellen mit unterschiedlichen Perioden enthalten.
Die Autokorrelationsfunktion (AKF) ist eine der am Häufigsten eingesetzten Methoden der PitchDetection. Werden zwei identische Funktionen korreliert, so führt dies zur Autokorrelationsfunktion [Meffert04]:

$$r_m = \sum_{n=-\infty}^{\infty} f_n f_{n+m}$$

Werden periodische Funktionen korreliert, welche keine endliche Energie besitzen, so wird die Summation der oberen Gleichung über eine Periodenlänge (Windowsize) N ausgeführt:

$$r_m = \frac{1}{N} \sum_{n=0}^{N-1} f_n f_{n+m}$$

Wird eine periodische Funktion mit sich selbst korreliert, entsteht als Ergebnis wieder eine periodische Funktion. Die AKF hat dieselbe Frequenz wie das zu korrelierende Signal. Ist dieses harmonisch, so ergibt sich als AKF immer eine Kosinusfunktion. Die AKF gibt also die innere Verwandschaft einer Signalfolge wieder.
In digitaler Audiosignalverarbeitung ist es gebräuchlich eine leicht abgeänderte Version der AKF zu nutzen, die sogenannte *short time autocorrelation funcion*: [Ryynänen1]

$$r(n) = \frac{1}{N} \sum_{k=0}^{N-n-1} f_k f_{k+n}$$

n ist hier das sogenannte *lag* welches sich auf die Zeitverschiebung des Signals bezieht. Die Peaks in der AKF indizieren die lags welche sich auf die fundamentale Frequenz beziehen. Normalerweise existieren mehrere Peaks mit ungefähr den selben Maxima, was die Entscheidung der fundamentalen Frequenz erschwert.
Eine weitere häufig zum Einsatz kommende Methode ist die sogenannte *average magnitude-difference function* (AMDF):

$$D_n = \sum_{k=0}^{N-1} |f_k - f_{k+n}|$$

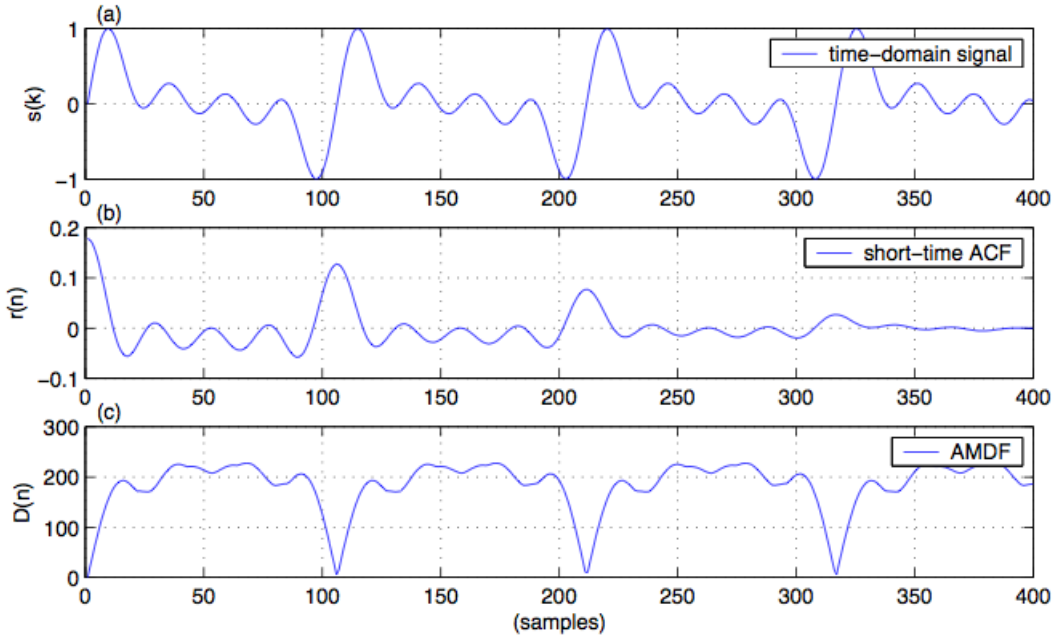

Abbildung 17: (a) Zu korrelierendes Signal (Abtastrate fs=21050, F0=200 Hz), (b) Ergebnis der short time AKF, und (c) AMDF. Extrema in (b) und (c) bei n=105 beziehen sich auf die fundamentale Frequenz von ca. 200Hz (21050/105=200.48Hz)[Ryynänen1]

4.2.2 Frequenzbereich

PitchDetection Algorithmen im Frequenzbereich bearbeiten das Signal im Frequenzbereich um so auf die fundamentale Frequenz zu schließen. Dies erfordert offensichtlich die Transformation vom Zeit- in den Frequenzbereich, zB mit der (eindimensionalen) Diskreten Fouriertransormation (DFT) [Meffert04]:

$$F(\omega) = \sum_{n=-\infty}^{\infty} f_n \cdot e^{-j\omega n T_A}$$

Abbildung 7 (Kapitel 2) zeigt den Plot eines Tons, gespielt mit einer Klarinette und das dazugehörige Frequenzspektrum. Da das Signal monophonisch ist, sind die im Spektrum enthaltenen

Frequenzen natürlich auf die Frequenz des gespielten Tons reduziert. Grundsätzlich zeigt das Spektrum alle im Signal enthaltenen Frequenzen in Form von Peaks an. Die X-Werte der Peaks entsprechen dann ähnlich der AKF den lags, welche sich im Verhältnis zu Abtastrate auf die fundamentale Frequenz beziehen. Da diese Methode sehr rechenintensiv ist, existiert eine optimierte Version der DFT unter dem Namen *Fast Fourier Transformation* (FFT), welche eine Implementierung in Echtzeit ermöglicht.
Die Diskrete Fouriertransformation kommt in dieser Arbeit nur für Analysezwecke zum Einsatz, jedoch nicht zur Berechnung der Frequenzen.

4.3 PitchDetection in dieser Arbeit

In dieser Arbeit wurden zwei Methoden zur Erkennung der fundamentalen Frequenz gewählt. Der sogenannte *YIN-Algorithmus* von de Cheveigne und Kawahara*[Cheveigné02]* und die sogenannte *McLeod Pitch Method* (MPM) von Phillip McLeod und Geoff Wyvill. [McLeod-1].
Obwohl beide in den Original-Papers ausreichend erläutert, evaluiert und diskutiert werden, sollen sie hier noch einmal grob vorgestellt werden.

4.3.1 Der YIN-Algorithmus

Der YIN-Algorithmus basiert auf der oben vorgestellten Average Magnitude Difference Funktion (AMDF) und enthält einige Optimierungen um die Akkuratheit der Pitcherkennung zu verbessern. Der Algorithmus lässt sich grob in die folgenden vier Schritte einteilen:

Gegeben ist ein zeitdiskretes Signal s(n), Buffergröße N und κ als ein konstanter Schwellwert.

1. Berechne die AMDF:

$$d(\tau) = \sum_{n=0}^{N-1} (s(n) - s(n+\tau))^2$$

2. Berechne die normalisierte Differenz Funktion (kumulative Häufigkeit) $d'(\tau)$:

$$d'(\tau) = \begin{cases} 1, & \tau = 0 \\ d(\tau) / \left[(1/\tau) \sum_{j=1}^{\tau} d(j) \right], & sonst \end{cases}$$

3. Suche den kleinsten Wert an τ für welchen ein lokales Minimum in $d'(\tau)$ kleiner ist als der gegebene Schwellwert κ. Wird kein solcher Wert gefunden, suche stattdessen das globale Minimum in $d'(\tau)$. $\hat{\tau}$ ist der gefundene Wert.

4. Interpoliere die Funktionswerte von $d'(\tau)$ an den Abszissen $\{\hat{\tau} - 1, \hat{\tau}, \hat{\tau} + 1\}$ mit einem Polynom zweiter Ordnung. Suche das Minimum des Polynoms im Bereich $\{\hat{\tau} - 1, \hat{\tau} + 1\}$ um einen Schätzwert der fundamentalen Periode zu erhalten.

Dieser Algorithmus wurde gewählt weil er äußerst robust ist, nicht rechenintensiv, es ist also eine Implementierung in Echtzeit möglich, weil er sich schon in anderen Systemen etabliert hat [Ryynänen1] [Krige]und weil eine Implementierung in Java existiert [TARSOS].

4.3.2 Die McLeod PitchMethod

Der Author Philip McLeod entwickelte dieses Verfahren und die dazugehöre Anwendung Tartini [McLeod-2] im Rahmen seiner Doktorarbeit. Zitat aus dem Artikel[McLeod-1]:

> A fast, accurate and robust method for finding the continuous pitch in monophonic musical sounds. [It uses] a special normalized version of the Squared Difference Function (SDF) coupled with a peak picking algorithm.
>
> MPM runs in real time with a standard 44.1 kHz sampling rate. It operates without using low-pass filtering so it can work on sound with high harmonic frequencies such as a violin and it can display pitch changes of one cent reliably. MPM works well without any post processing to correct the pitch.

Diese Methode zur Erkennung der fundamentalen Frequenz ist also auch perfekt geeignet für ein automatisches Transkriptionssystem für monophonisches Audiomaterial. Sie ist schnell genug um in Echtzeit implementiert werden zu können und arbeitet ohne pre- oder postprocessing wie Tiefpass Filter oder Tuning. Auch dieser Algorithmus existiert als Implementation in Java [TARSOS] und soll hier noch einmal rudimentär vorgestellt werden:

Gegeben ist ein zeitdiskretes Signal x(n), Buffergröße W.

1. Berechne die *normalized square difference function* (NSDF) :

$$d'_t(\tau) = \sum_{j=t-(W-\tau)/2}^{t+(W-\tau)/2-1} (x_j - x_{j+\tau})^2$$

2. *PeakPicking* Algorithmus: Suche zuerst alle nutzvollen lokalen Maxima, im Paper sogenannte "Key-Maxima". Die x-Werte dieser Maxima repräsentieren unter Umständen die Periode des Signals, aus welcher auf die Frequenz schließen lässt. Um dies zu optimieren wird nun nach dem höchsten Wert zwischen jedem Paar positiver Nulldurchgänge gesucht. Das erste Maximum, an Stelle 0, wird ignoriert.

4.3.3 MIDI-Repräsentation

Erkannte Frequenzen werden mit Hilfe der folgenden logarithmischen Funktion zu MIDI Nummern übersetzt. Dadurch ist es möglich auch "ungenaue" Töne (zB bei nicht richtig gestimmten Instrumenten) dennoch richtig anzuzeigen:

$$x' = 69 + 12\log_2(\frac{F_0}{440})$$

F_0 ist hier die erkannte Frequenz in Hertz, x' die daraus folgende MIDI Nummer. Die Zahlen 69 und 440 repräsentieren eine MIDI-Noten Referenz, 69 steht im MIDI System für die Note A4 mit der fundamentalen Frequenz 440 Hz. Außerdem steht die 12 für die 12 Halbtöne innerhalb einer Oktave. Dies bezieht sich auf das westliche Tonsystem.

4.4 Erkennung der Notenwerte

Bis jetzt wurde schon viel über die Erkennung der fundamentalen Frequenz und somit der Tonhöhe gesprochen. Der Zweite grundlegende Schritt eines automatischen Transkriptionssystems besteht darin, die Notenwerte, also die Länge einer Note zu erkennen. Auch dies ist ein nicht zu unterschätzender Prozess.

4.4.1 Onset/Offset Erkennung

Eine wichtige Aufgabe eines automatischen Transkriptionssystems ist die sogenannte Onset-Erkennung. Es existieren zahlreiche Verfahren zu diesem Thema.
Ziel dabei ist es, zu erkennen, wann ein Ton startet bzw. endet um somit Entscheidungen bzgl. der Notenwerte treffen zu können. Um ein musikalisches Signal in Notenschrift umzuwandeln ist es notwendig, den physikalischen Beginn einer Note genau zu erkennen[Klich04]. Mit Hilfe des sogenannten *ADSR-Modells* lässt sich dieser Sachverhalt verdeutlichen. Ein Klang wird in vier Phasen geteilt:

1. Attack - Anschlag
2. Decay - Dämpfung
3. Sustain - Haltephase
4. Release - Ausschwingen

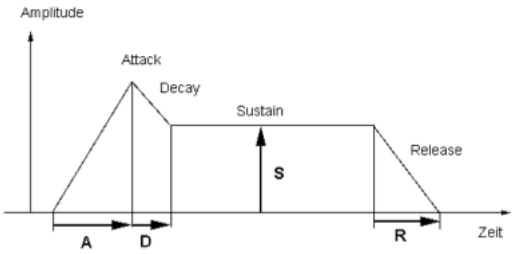

Abbildung 18: ADSR - Modell [Klich04]

Manche Instrumente treffen nicht hundertprozentig auf dieses Modell zu. So besitzt zum Beispiel der Ton einer Orgel fast keinen Attack oder Release, da der Ton nach Drücken der Taste sofort steht und nach Loslassen der Taste auch sofort wieder weg ist. In dieser Arbeit werden Onsets basierend auf Noten bzw. den errechneten Frequenzen erkannt. Diese Idee stammt aus [Monti2000]. Pro Buffer wird eine Frequenz bzw. eine Note berechnet. Das System sammelt diese Werte und wertet diese bezogen auf die folgenden Bedingungen aus.

4.4.2 Sechzehntelnoten

Wie schon erwähnt wird der kleinste unterstützte Notenwert in dieser Arbeit die Sechzehntel Note sein, da ich der Meinung bin, dass dies ausreichend ist für meine in Kapitel 1 vorgestellten Zwecke. Die kleinste Notenlänge die das System erkennen kann ist also eine Sechzehntel. 32tel können also nicht mehr erkannt werden und demnach auch keine punktierte Sechzehntel. Dies bringt bei genauerer Betrachtung einige Bedingungen mit sich. Aufgrund der Tatsache dass mindestens eine Sechzehntel erkannt werden kann folgt, dass immer nur Sechzehntelnoten, bzw. Vielfache davon erkannt werden können.

Angenommen es wird eine einzige Sechzehntel Note erkannt, so muss auch diese angezeigt werden. Bei der Länge von zwei Sechzehntel Noten kann eine Achtel angezeigt werden, bei drei eine punktierte Achtel, bei vier eine Viertel Note usw.
Werden jedoch zum Beispiel fünf Sechzehntel Noten zusammenhängend erkannt, also dazwischen war nie eine Pause, Abklingen des Instruments oder ein anderer Ton, so wäre eine Möglichkeit eine Viertel Note und eine Sechzehntel Note mit Bindebogen anzuzeigen, man könnte aber auch eine Sechzehntel und eine Viertel Note mit Bindebogen anzeigen:

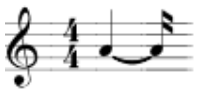

Abbildung 19: Anzeigevariante 1 bei erkanntem Ton mit Länge = $\frac{5}{16}$

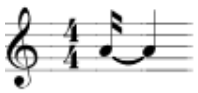

Abbildung 20: Anzeigevariante 2 bei erkanntem Ton mit Länge = $\frac{5}{16}$

Beide Töne werde exakt gleich gespielt, dies ist nur eine Frage der Darstellung, welche wiederum abhängig von der Melodie, Konventionen oder Vorlieben einzelner Musiker ist. In dieser Arbeit werden die Noten in den beschriebenen Sonderfällen immer gemäß der Anzeigevariante 1 gemalt. Ein Editieren der Noten ist ja ohnehin nach der automatischen Transkription noch möglich.

4.4.3 Übersetzung der Zeit

Desweiteren ist relevant unter welchen Bedingungen welcher Notenwert gewählt werden sollte. Es geht also um die Übersetztung der Zeit (in Millisekunden) in Notenwerte unter Bezugnahme der Kenntnis der Taktschläge pro Minute. Folgende Überlegungen geben einen Überblick über diese Bedingungen:

Angenommen wir haben 60 Schläge pro Minute welche den Vierteln entsprechen, dann dauert eine Viertel Note im *perfekten Fall* genau eine Sekunde bzw. 1000 Millisekunden (60000ms/60bpm). Eine Achtel dauert dann 500ms und eine Sechzehntel 250ms. Um bei der Erkennung der Notenwerte tolerant zu sein, habe ich mich entschieden einen Toleranzbereich von der Länge einer 32tel zu nutzen (hier 125ms). Daraus folgt dass ich jede Länge unter 250ms + 125 ms = 375ms schon bzw. noch als Sechzehntel Note anzeige, dann ist der nächste Bereich von 375ms bis 500+125 also im Bereich 375-625 wird eine 8tel gemalt, usw. Es wird also immer genau die Mitte als Grenzwert genommen. Zahlreiche Diskussionen mit Musikern ließen mich zu dem Entschluß kommen dass dies wohl die toleranteste Methode ist. Folgende Skizze verdeutlicht dieses Beipiel anhand eines Zahlenstrahls:

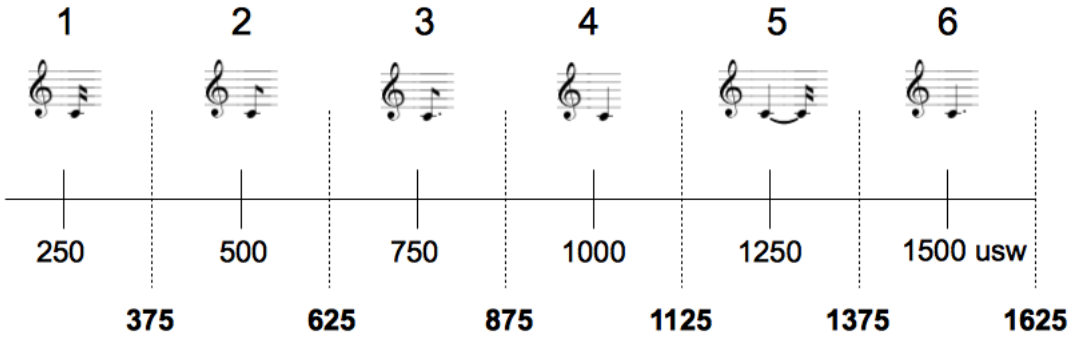

Abbildung 21: Zahlenstrahl visualisiert die Notenwertbedingungen bis zur punktierten Viertel. Die normalen Linien entsprechen den perfekten Notenlängen und die gestrichelten Linien sind die Grenzen. (60 bpm, Toleranz: 125ms)

Die Zahlen oben sind die Vielfachen der Sechzehntel. Die 5 ist der erste Fall bei dem 2 Noten mit Bindebogen gemalt werden müssen. Die weiteren Fälle sind dann 7, 9, 10, 11, 13, 14 und 15. Ziel ist es also eine beliebige erkannte Notenlänge in Millisekunden in einen Wert zwischen 1 und 16 zu übersetzen. Dabei entspricht 1 einer Sechzehntel, 2 einer Achtel 3 einer punktierten Achtel usw.

Abgesehen davon liegt es noch an der Qualität der Aufnahme, bzw. der Fähigkeit des Musikers, exakt zu spielen, wie gut das System sein wird. Das System kann, wie auch ein MIDI Transkriptionssystem, nur das wiedergeben, was eingespielt wurde. Wie schon erwähnt ist es möglich die Transkription im Nachhinein zu editieren. Man kann sie aber natürlich auch nochmal einspielen.

4.5 Der Notenkollektor

4.5.1 Erste Versuche

Mein erster Versuch war, die Aufnahme synchron zu einem MIDI-basierten Metronom zu realisieren. Der Metronomprozess wüsste immer genau Bescheid wieviel Zeit vergangen ist und könnte zu fest definierten Zeiten, zB in 16tel Notenintervallen, Events an den Aufnahmeprozess senden um diesem mitzuteilen, dass er eine Entscheidung treffen soll. Jedoch ist solch eine Synchronisation nicht ohne Weiteres in Java zu implementieren. Der Grund dafür ist u.a. der Garbagecollector der JVM, welcher für die automatische Speicherbereinigung zuständig ist. Um diesen zu umgehen müsste man eine spezielle virtuelle Maschine nutzen, welche sich um Garbagecollection in Echzeit kümmert.
Abgesehen davon sind die Synchronisationsmethoden in der Java Sound API nicht implementiert, und auch andere Versuche, zB zu einer externen Clock(Soundkarte) zu synchronisieren, wären alle äußerst aufwändig und würden Java's Vorteil der Plattformunabhängigkeit zu Nichte machen. Abgesehen davon könnte man keine Audiodateien in das System schicken, da zu dem Metronom eingespielt werden "muss".
Aus diesen Gründen habe ich mich entschieden, die Notenlängen basierend auf sogenannten notenbasierenden Onsets und Offsets zu erkennen. Mein System muss die Geschwindigkeit der monophonischen Aufnahme in beats per minute kennen, und kann somit Entscheidungen bzgl. der Notenwerte treffen. Daraus folgt dass das hier zu entwickelnde System nicht in der Lage sein wird, das Tempo oder die Taktart der Aufnahme zu erkennen. Diese Parameter müssen vorher eingestellt werden.
Von Anfang an war es mir wichtig, dass man die Noten sehen kann während man spielt. Um keine Zeit beim Extrahieren der notwendigen Features aus den Samplebuffern zu verlieren, gibt es grob gesagt zwei Prozesse. Der Eine liest ständig die Buffer vom Mikrofon und macht nichts anderes als diese, nach einer Byte-to-Float Konvertierung, an eine blockierende FIFO-Queue[6] zu geben, welche von dem Folgenden, zweiten Prozess abgearbeitet wird.

4.5.2 Funktionsweise

Der Notenkollektor ist das Herzstück des Systems. Dieser kennt die PitchDetectoren (YIN, MPM), kann somit also die fundamentale Frequenz eines Buffers berechnen, und nun beginnen zu "sammeln". Als Erstes wird aus dem Buffer Pitch und Level extrahiert und der Pitch zu einem MidiKey gemäß der oben genannten MIDI-Mapping-Funktion übersetzt. *MidiKey*, *Level* und *Länge* des Buffers in Millisekunden werden nun an den CollectorProzess übergeben.

Zwei Parameter sind besonders wichtig für den Collector: *Minimumduration* und *Minimumlevel*. Diese Parameter sind die Bedingungen für einen Onset bzw. Offset. Wenn also eine Note solange die Selbe bleibt bis die Länge dieser Sequenz größer als *Minimumduration* ist, so ist die Bedingung für ein Onset erfüllt und es wird begonnen diese Noten zu sammeln. Nun gibt es zwei Möglichkeiten: Offset basierend auf einer neuen Note oder Offset basierend auf einer Pause. Auch hier muss die Länge mindestens *Minimumduration* betragen. Der Parameter *Minimumlevel* ist notwendig um zu überprüfen ob die Amplitude auch hoch genug ist um dies als Ton zuzulassen. Berechnet wird das Level eines diskreten Signals x(n) mit Hilfe der "Root Mean Square" - Formel:

$$rms = \sqrt{\frac{1}{2}\sum_{n=0}^{N} {x_n}^2}$$

[6] FIFO - First In First Out, Queue: Warteschlange

Treffen die Bedingungen für ein Offset zu, so werden die bis dahin gesammelten Noten ausgewertet. Der am Häufigsten vorkommende Pitch in dem Pitchvektor wird gewählt und dessen Länge mit Hilfe der in Kapitel 4.4.3 erläuterten Prozedur errechnet. Treffen die Bedingungen für ein Onset ein, so werden die bis dahin gesammelten Pausen ausgewertet. Jeder Pitch bzw. jede Pause in dem bis dahin gesammelten Pitchvektor hat die selbe Länge, welche sich aus der BufferSize und der Abtastrate berechnen lässt. Bei einer BufferSize von 1024 Byte und einer Abtastrate von 44100 Samples pro Sekunde ergibt sich also eine Bufferlänge von ca. 23.21ms (1024/44100*1000). Man muss also die Länge des Pitchvektors mit diesem Wert multiplizieren um die Länge des erkannten Tons zu erhalten.

4.5.3 ABC Notation

Um Notenschrift textbasierend darzustellen gibt es mehrere Standards, einer davon ist *abc-notation* [ABC]. *abc-notation* ist ein textbasierendes Musiktranskriptionssystem und weltweit stark verbreitet. Es ermöglicht also eine rein textbasierte Repäsentation von Notenschrift. Dadurch ist es möglich von mehreren Programmen verarbeitet zu werden (zB [ABCMIDI]). Der Grund für diese Wahl war einerseits die weite Verbreitung und andererseits existiert für die Sprache Java eine freie Bibliothek, welche die Verarbeitung ermöglicht wie zum Beispiel die Konvertierung eines ABC-Strings in MIDI und auch in Notenschrift. Der Collector muss also aus den erkannten Noten und deren Längen einen der abc-notations Syntax entsprechenden String erzeugen.
Da MidiKeys von 0 bis 127 existieren, gibt es eine "MidiKey nach ABC"-MappingFunktion welche in einem Stringarray 128 AbcNoten hält. Somit ist es einfach vom erkannten MidiKey auf die ABC-Note zu schließen. Wird zum Beispiel MidiKey=60 erkannt, so ist im Array an 60. Stelle die ABC-Note C. Der Collector kann diese Note nun an die graphische Oberfläche übergeben wo sie dann gemalt wird.

Im Folgenden soll ein kurzer Überblick über die wichtigsten Abc-Notationsparameter anhand der Referenzmelodie "Korobeiniki" gegeben werden:

X:0
T:Korobeiniki (Tetris Melody)
Q:1/4 = 60
M:4/4
L:1/16
K:C
E2B,1C1D2C1B,1A,2A,1C1E2D1C1|B,3C1D2E2C2A,2A,4|z1D2F1A2G1F1E2z1C1E2D1C1
B,2B,1C1D2E2C2A,2A,4|E4C4D4B,4|C4A,4^G,4z4|E4C4D4B,4|A,2C2A4^G4z4

Das X in der ersten Zeile entspricht dem Index eines Abc-Tunes. Mit dem Buchstaben T wird der Titel angegeben, Q ist das Tempofeld. Damit wird angegeben welches Tempo das Lied hat (hier 60 Schläge pro Minute, den Vierteln entsprechend). Mit M gibt man die Taktart an, und das L-Feld gibt an welche Notenlänge die Standardlänge ist. Mit L:1/16 gibt man also an, dass die Standardlänge 1 einer Sechzehntel Note entspricht. Der Buchstabe K gibt die Tonart und den Notenschlüssel an und danach folgen die Noten.
Diese abc-Notation ergibt die Transkription in Kapitel 6, Abbildung 28.

5 Implementierung

5.1 Programmiersprache und externe Bibliotheken

jAM wurde in der Programmiersprache Java implementiert. Als Entwicklungsumgebung wurde Eclipse gewählt. Externe Bibliotheken sind Tarsos [TARSOS]und abc4j [ABC4J].
Tarsos ist eine kleine, freie Audioprocessing API, welche hier vor allem wegen den Beiden oben vorgestellten PitchDetection Implementationen genutzt wird.
Um die Notenschrift anzuzeigen habe ich mich für abc4j entschieden, ein Toolkit welches es ermöglicht Noten anzuzeigen und diese als MIDI auch abzuspielen, zu speichern etc. Diese API bezieht sich auf den nichtkommerziellen, offenen Standard zur Notation von Musik in einem Textformat (ASCII) "ABC-Notation" [ABC].

5.2 Beschreibung der einzelnen Komponenten

5.2.1 Paketstruktur

jAM wurde nach dem Model - View - Controller Prinzip (MVC) implementiert. Dies dient der Trennung von Präsentation und Anwendungslogik was u.a. den Vorteil hat, dass man bei einer späteren Portierung gegebenenfalls nur Präsentation und Steuerung ändern muss. Auch auf Modularität und Wiederverwendbarkeit wurde großen Wert gelegt, weshalb die Software in einzelne Pakete aufgeteilt wurde, dessen Zwecke und Funktionalitäten hier nun erläutert werden.

Im Verzeichnis *src* gibt es drei Pakete, welche nicht zu jAM gehören. Diese sind *abc*, *scanner* (gehören zu abc4j) und *be.horgent.tarsos.* Der Grund dafür ist, dass evtl. Änderungen an diesen externen Paketen vorgenommen werden müssen und deshalb deren Quellcode direkt eingebunden wurde.
Das Paket mit dem Code von jAM heißt *de.hsa.jam.* Darunter gibt es weitere Pakete, welche hier nun der Reihe nach vorgestellt werden.

5.2.1.1 Paket de.hsa.jam Hier liegen nur zwei Klassen: *jAM* und *ControllerEngine*. Die Klasse *jAM* beinhaltet die main-Methode um die Anwendung zu starten. Sie initialisiert Model, Views und eine ControllerEngine und startet somit das Programm. Es gibt drei Views zu initialisieren: MainWindow, ChromaticTunerFrame und MetronomeFrame. Diese drei Views werden dem Controller zusammen mit dem Model übergeben. Außerdem wird die Logging-Funktionalität hier konfiguriert. Wird an die main jedoch als Programmparameter "eval" übergeben, so wird keine View initialisiert, sondern nur die Evaluation gestartet.

Die Klasse *ControllerEngine* ist gemäß dem ModelViewController Prinzip die zentrale Steuerungsklasse. Sie leitet von der Klasse AbstractController ab welche ein Interface PropertyChangeListener implementiert, wodurch es möglich wird per "propertyChange" sogenannte "Propertyupdates" an alle registrierten Views zu senden. Ändert sich etwas im Model (Propertychange), so wird diese Änderung an alle Views weitergeleitet.
Abgesehen davon gibt es noch die Funktion setModelProperty, welche es ermöglicht allen registrierten Models Änderungen einer View mitzuteilen. Die Klasse ControllerEngine macht nun nichts anderes als den Vermittler zwischen graphischer Benutzeroberfläche und dem Backend zu spielen.

5.2.1.2 Paket de.hsa.jam.ui Dieses Paket ist für die graphische Benutzeroberfläche verantwortlich. Es repräsentiert das Frontend, also die View im MVC-Prinzip, enthält keine Pro-

grammlogik sondern nur Informationen für die Darstellung. Alle Benutzer Events werden über die Klasse ControllerEngine abgefangen und von dort verarbeitet. Die wichtigsten Klassen hier sind *MainWindow*, welche für das Hauptfenster verantwortlich ist und unter Verwendung der Klasse *ScorePanel* die Noten anzeigt, *ChromaticTunerFrame*, welche ein Fenster mit chromatischem Stimmgerät implementiert und *MetronomeFrame*, ein einfaches Fenster mit Metronomfunktion.

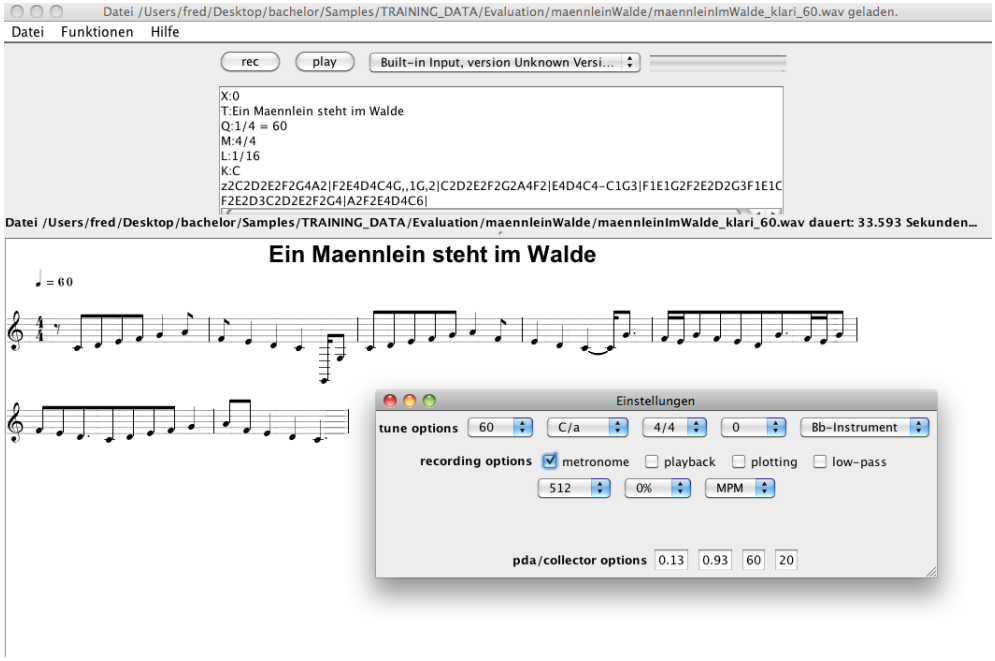

Abbildung 22: Hauptfenster von jAM mit Einstellungen: Transkription von "Ein Männlein steht im Walde", gespielt mit einer Klarinette bei 60 Schlägen pro Minute.

Die Klasse MainWindow kümmert sich also um die Anzeige des Hauptfensters (siehe Abbildung 22) und zusätzlich um das Fenster mit den Einstellungen und Parametern für das System. Man kann sehen dass die Einstellungen in 3 Bereiche aufgeteilt wurden: Tuneoptions, recording options und pda/collector options. Diese Einstellungen sollen nun hier erklärt werden:

Tune Optionen:
Das sind Einstellungen bzgl. des Lieds, also musiktheoretischer Art.

1. BPM - Die Schläge pro Minute (hier nur Viertel). Die Aufnahme muss dieser Geschwindigkeit entsprechen und das System muss diesen Parameter kennen um Entscheidungen bzgl. der Notenwerte zu treffen.

2. Tonart - Die Tonart der einzuspielenden Melodie

3. Taktart - Die Taktart der einzuspielenden Melodie

4. Transponieren - Nach der Aufnahme ist es möglich die Transkription zu transponieren

5. Transponiert aufnehmen - Es ist jedoch auch möglich transponiert aufzunehmen. Das macht zum Beispiel bei Bb-Instrumenten Sinn. Spielt ein Spieler eines Bb-Instruments ein C, so

erklingt in Wirklichkeit jedoch ein Bb. Die Transkription wäre eigentlich falsch. Deshalb kann man hier sagen dass man ein Bb-Instrument spielt, was das System dazu veranlasst, jedes mal nach der Erkennung eines Pitches, diesen um zwei Halbtöne nach oben zu transponieren. Somit wird zwar ein Bb erkannt, angezeigt jedoch ein C.

Recording Optionen:
Einstellungen bzgl. der Aufnahme

1. metronom - Man kann einstellen ob man das Systeminterne Metronom nutzen möchte oder nicht. Dennoch muss man zu den hier eingestellten BPM spielen um akzeptable Ergebnisse zu erhalten.
2. playback - Man kann sich die Aufnahme anhören während man sie geladen hat, und durchläuft. Dann dauert der Prozess natürlich länger.
3. plotting - ermöglicht eine einfache zwei-dimensionale Visualisation der Buffer
4. low-pass - Tiefpass Filter hinzuschaltbar
5. Buffersize - Die Größe eines Buffers in Byte
6. Bufferoverlap - Die Bufferüberlappung in Prozent
7. PitchDetektor - Welcher PitchDetektor zum Einsatz kommen soll (YIN, MPM)

PDA/Collector Optionen:
Einstellungen bzgl. der PitchDetektoren sowie des Collectors.

1. YIN-Threshold - Der Yin-Schwellwert
2. MPM-Threshold - Der Mpm-Schwellwert
3. Minimumduration - Die minimale Länge in Millisekunden, welche als Bedingung für den Beginn eines Tons steht
4. Minimumlevel - Die Amplitudenauslenkung, welche als Bedingung für einen "laut genug" gespielten Ton steht, alles darunter wird als "Stille" interpretiert

Die Klasse *MetronomeFrame* ermöglicht ein einfaches, Midi-basierendes Metronom zum Üben und *ChromaticTunerFrame* implementiert ein einfaches Stimmgerät. Dies ermöglich es sein Instrument zu stimmen, bevor man aufnimmt.

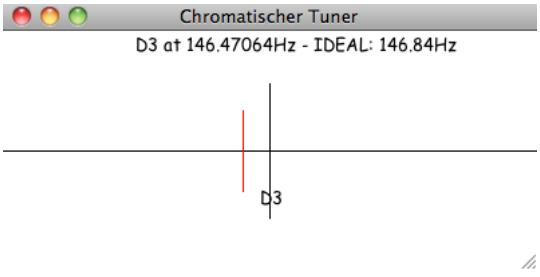

Abbildung 23: *ChromaticTunerFrame* - Ein einfaches chromatisches Stimmgerät

Abbildung 24: Das Dateimenü

Abbildung 24 zeigt das Dateimenü. Folgende Funktionen wurden implementiert:

1. Neues Projekt - Solange keine Datei geladen wurde wird immer aufgenommen. Ist eine Datei geladen und man will jedoch neu aufnehmen muss man “Neues Projekt” wählen
2. Öffne WAV Datei - Lade eine WaveDatei.
3. Öffne jAM Projekt - Lade eine Datei mit abcnotation.
4. Speichern als WAV Datei - Speichere die aufgenommene Wav Datei.
5. Speichern als jAM Projekt - Speichere die generierte Transkription als Textdatei gemäß abcnotation.
6. Speichern der Noten als Bild - Die generierte Transkription als Bild speichern, zum Beispiel um diese auszudrucken.
7. Speichern als MIDI Datei - Speichere die generierte Transkription als MIDI-Datei. Dadurch ist es möglich diese mit anderen Programmen zu bearbeiten.
8. Einstellungen - Zeige das Einstellungsfenster.

Zuletzt sei hier noch die Klasse *SimplePlotterFrame* erwähnt, welche eine einfache zwei-dimensionale Darstellung eines Float-Arrays in einem Koordinatensystem ermöglicht. Somit ist es möglich in einem zusätzlichem Thread zum Beispiel das reine Audiosignal sowie das Signal nach der Verarbeitung des laufenden PitchDetctors in Echtzeit zu visualisieren.

5.2.1.3 Paket de.hsa.jam.audio Dieses Paket repräsentiert das Backend. Hier liegt die Klasse Model, welche alle darzustellenden Daten kennt und gemäß MVC-Prinzip von Präsentation und Programmsteuerung unabhängig ist.
Abbildung 25 zeigt ein sehr einfaches Klassendiagramm um einen Überblick über die wichtigsten Klassen zu geben. Gut zu sehen ist, dass die Klasse *ControllerEngine* sozusagen in der Mitte steht und somit der Vermittler zwischen Präsentation und Steuerung ist. Außerdem kann man sehen, dass die Klasse *AudioDispatcher* als “Produzent” und die Klasse *NoteCollectorWorker* als “Konsument” gekennzeichnet wurde. Dies hat folgenden Grund: *AudioDispatcher* ist eine Klasse der TarsosLib, welche in einem Thread (Produzent) von einem gegebenen *AudioInputStream* liest um dann die gelesenen Daten an alle Implementatoren des Interfaces *AudioProcessor* zu übergeben. Die Klasse *MyAudioProcessor* macht nun nichts anderes als diese Daten an die *AudioBufferQueue* zu geben, welche dann vom *NoteCollectorWorker* in einem anderen Thread (Konsument) abgearbeitet wird.

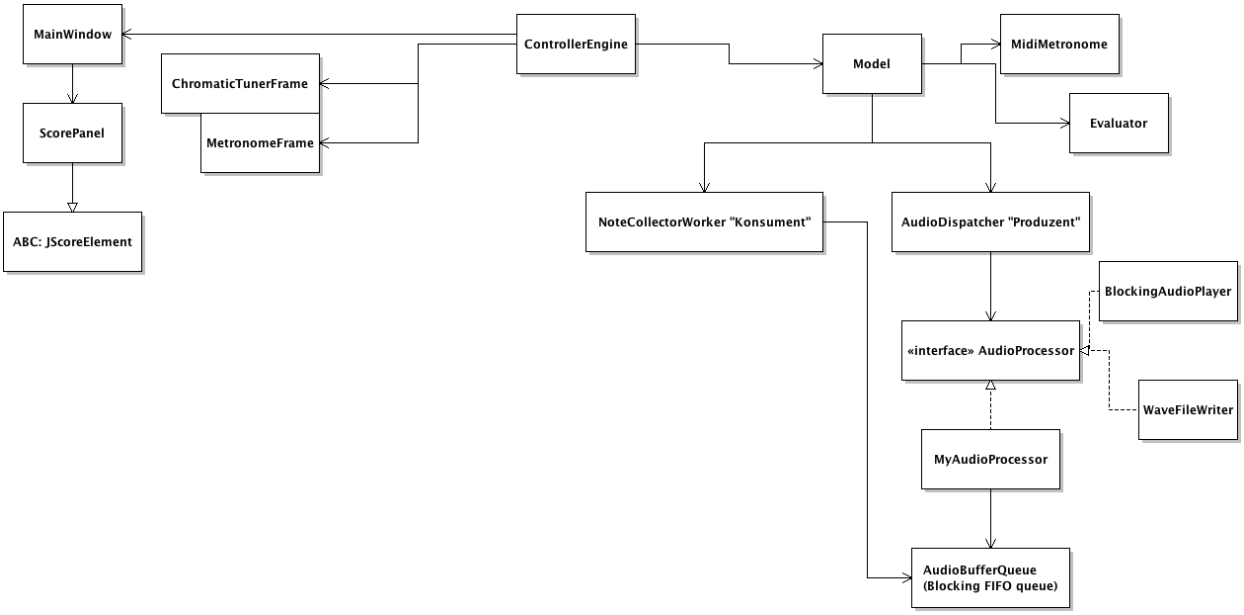

Abbildung 25: Rudimentäres Klassendiagramm, die wichtigsten Klassen

Die wichtigste Klasse in diesem Paket ist wohl *Model.* Diese kennt die wichtigsten Programmparameter wie Buffersize, Tonart etc. und übernimmt die wichtigsten Aufgaben wie das Initialisieren der Aufnahme oder der Evaluation, globales Errorhandling, Abspielen der Transkription als Midi, speichern dieser als Bild oder MIDI Datei usw. Sie leitet von AbstractModel ab, welche u.a. die Methode firePropertyChange enthält. In der Klasse jAM wurde dem Controller das Model übergeben. Der AbstractController implementiert wie oben erwähnt das Interface PropertyChangeListener. In der Funktion addModel wird nun zusätzlich dem Model per model.addPropertyChangeListener(this) ein PropertyChangeListener übergeben. Somit ist es für das Model möglich, Änderungen im Backend, welche Änderungen in den Views nach sich ziehen, allen registrierten Views mitzuteilen. Dies geschieht dann über die Methode firePropertyChange der Klasse java.beans.PropertyChangeSupport.
Desweiteren ist noch die Klasse WaveFileWriter zu erwähnen. Diese schreibt nach jeder Aufnahme eine Wave-Datei namens “jAM_session.wav” ins Heimverzeichnis.

5.2.1.4 Paket de.hsa.jam.audio.collector Dieses Paket enthält 2 Klassen: *AudioBufferQueue* und *NoteCollectorWorker*. Die Klasse AudioBufferQueue implementiert eine blockierende Warteschlange, welche float Arrays hält. Sie wird vom aufnehmenden Prozess befüllt und von der Klasse NoteCollectorWorker abgearbeitet. Sie enthält zwei synchronisierte Methoden: add() und get(). add() fügt der Warteschlange einen neuen Buffer hinzu und get() gibt gemäß dem FirstInFirstOut Prinzip den zuerst hinzugefügten Buffer zurück. Ist die Schlange gerade leer wird blockierend gewartet. Es kommen die Javainternen Methoden wait() und notify() zum Einsatz.

Die Klasse *NoteCollectorWorker* ist für die wichtigsten Schritte in der Pipeline verantwortlich: Feature Vektor Extraktion, Segmentierung und Grammar PostProcessing. Sie kümmert sich also um den Fluss der Pipeline von der Pitcherkennung bis zur angezeigten Note. Zusätzlich sammelt diese Klasse alle (auch falsche) erkannten Noten für Statistiken, und alle tatsächlich erkannten

Noten für die Evaluation.

5.2.1.5 Paket de.hsa.jam.audio.midi Hier befindet sich die Klasse MidiMetronome, welche ein einfaches, MIDI-basierndes Metronom implementiert. MIDI steht für Musical Instrument Digital Interface, also eine Digitale Schnittstelle für Musikinstrumente. Es ist ein Datenübertragungs-Protokoll zur Übermittlung musikalischer Steuerinformationen zwischen elektronischen Instrumenten wie Keyboards, Synthesizern etc. Drückt man eine Taste auf einem MIDI-Keyboard, so werden die Informationen Tonhöhe, Tonlänge und Anschlagdynamik, also Lautstärke übertragen, wodurch dann ein Klangerzeuger Töne erzeugt. Die Dauer eines Tones ergibt sich aus den "Note-On" und "Note-Off" Befehlen. Die übertragenen MIDI-Daten können dann an beliebige Klangerzeuger gesendet werden, womit es möglich ist, jedes beliebige Instrument zu simulieren. Ein weiteres Anwendungsgebiet der automatischen Transkription von digitalen Audiosignalen ist die Echtzeit-Übersetzung des Signals in MIDI. So könnte man live eine E-Gitarre spielen, erzeugt wird aber zum Beispiel ein Saxophon Sound.

5.2.1.6 Paket de.hsa.jam.util Die einzige Klasse hier ist jAMUtils. jAMUtils ist eine Helferklasse mit statischen Methoden für Aufgaben der digitalen Signalverarbeitung wie zum Beispiel die Berechnung der RootMeanSquare-Formel. Zusätzlich befinden sich hier mathematische Funktionen wie das Finden von Extrema in einem Vektor.

5.2.1.7 Paket de.hsa.jam.evaluation In diesem Paket liegt die Klasse Evaluator. Diese ist verantwortlich für die Verbindung zur Datenbank und der anschließenden Kommunikation mit dieser. Außerdem evaluiert diese Klasse das System basierend auf den in Kapitel 6 vorgestellten Referenzmelodien und Bewertungsmethoden und gibt eine graphische Ausgabe in Textform aus.

5.2.2 Logging

Um den Programmfluss nachvollziehen zu können wird eine Logdatei mit dem Namen jAM.log im Homeverzeichnis erstellt. Dies dient u.a. zur Fehlerbehebung. Grundsätzlich werden alle Aktionen mitgeschrieben, die für eine spätere Untersuchung notwendig sind oder sein könnten. Somit lassen sich zum Beispiel Fehler bezogen auf die Soundkarte oder im Collector-Prozess besser nachvollziehen.

5.3 Dokumentation

Um den Quellcode von jAM zu dokumentieren wurde Javadoc gewählt. Javadoc ist ein Software-Dokumentationswerkzeug, das aus Java-Quelltexten automatisch HTML-Dateien erstellt. Es ist im Java Development Kit (JDK) enthalten. Diese HTML-Dateien machen es dann möglich die Dokumentation der Pakete, Klassen oder Methoden im Browser anzusehen. Durch bestimmte Kommentare im Quelltext (sogenannte tags, zB @author) lässt sich dann das Tool automatisiert ausführen um die HTML-Seiten zu erzeugen. Die generierten HTML-Seiten befinden sich in “jAM_Quellen/doc”.
Es wurden im Paket de.hsa.jam und all dessen Unterpaketen die wichtigsten Methoden in allen Klassen dokumentiert. In der Dokumentation der Klasse jAM wurde dokumentiert, in welchen Klassen der externen Bibliotheken abc4j und tarsos Änderungen vorgenommen wurden.

6 Evaluation

Um das System bewerten zu können ist ein Bewertungsmaßstab notwendig, welcher die Erkannten mit den eigentlichen Melodien vergleicht. Es werden mehrere Durchläufe mit unterschiedlichen Systemkonfigurationen durchgeführt und die Ergebnisse hier präsentiert. Anschließend werden Fehler und Verbesserungsmöglichkeiten diskutiert.

Generell kann man ein automatisches Notentranskriptionssystem qualitativ sowie quantitativ evaluieren[Ryynänen1]. Dieses System wird quantitativ bewertet, indem die Differenz zwischen einer Reihe vorher bekannter "Referenzmelodien" und den dazugehörigen, tatsächlich erkannten Melodien berechnet wird. Diese Referenzmelodien wurden von mehreren Instrumenten eingespielt und dann anhand folgender Verfahren evaluiert.
Zusätzlich wurden all diese Melodien in reinem MIDI Format erstellt und dann aufgenommen, um zu zeigen wie sich das System im "perfekten" Fall verhält.

6.1 Aufnahme der Referenzmelodien

Die Melodien wurden alle bei 60 Schlägen pro Minute mit folgenden Mikrofonen aufgenommen:

1. Shure SM58
2. Shure SM11
3. Integriertes Mikrofon eines MacBook Pro

6.1.1 Ein Männlein steht im Walde

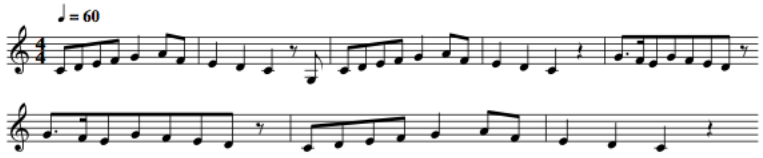

Abbildung 26: Referenznoten zu "Ein Männlein steht im Walde"

"Ein Männlein steht im Walde" ist ein volkstümliches Kinderlied von August Heinrich Hoffmann von Fallersleben aus dem Jahr 1843. Es enthält insgesamt 50 Noten bzw. Pausen, wurde per MIDI-Saxophon aufgenommen und zusätzlich von einer Akustikgitarre (Nylonsaiten), einer Violine, einem Akkordeon und einer Klarinette.
Dieses Lied wurde gewählt weil es eine äußerst einfache Melodie ist. Tonart ist C-Dur, d.h. keine Vorzeichen, die Taktart beträgt 4/4.

6.1.2 Joshua fought the battle of Jericho

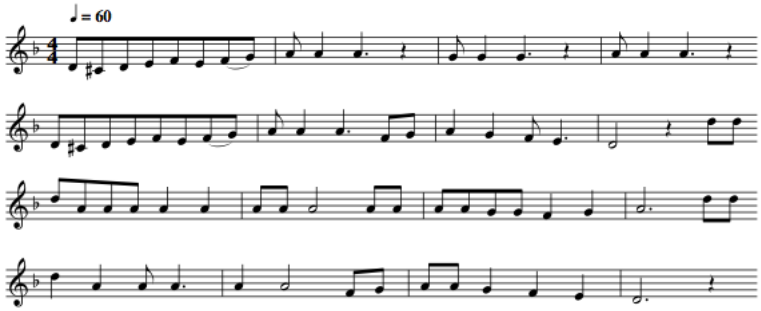

Abbildung 27: Referenznoten zu “Joshua fought the battle of Jericho”

“Joshua fought the battle of Jericho” ist ein Spiritual. Es handelt von der im Buch Josua erzählten Eroberung der Stadt Jericho durch die aus Ägypten ausziehenden Israeliten unter Führung Josuas. Es enthält insgesamt 76 Noten bzw. Pausen, wurde auch per MIDI-Saxophon aufgenommen und zusätzlich von einer Akustikgitarre (Nylonsaiten), einer Violine, einem Akkordeon und einer Klarinette.
Dieses Lied wurde gewählt wegen den vielen Onsets/Offsets mit dem selben Ton. Der Collectorprozess hält solange eine Note, bis die Bedingungen für einen Offset erfüllt sind. Spielt man nun zum Beispiel ganz oft die selbe Achtelnote kann es sein, dass das System diese zusammen führt, bspw. zwei Achtel zu einer Viertelnote. Deshalb spielt auch die Lautstärke bzw. Amplitude des Signals eine Rolle: Erhöht diese sich zwei Mal in der selben Notensequenz, kann man davon ausgehen dass ein neuer Ton kommt. Um dies zu testen eignet sich dieses Lied recht gut.

6.1.3 Korobeiniki (Tetris Melodie)

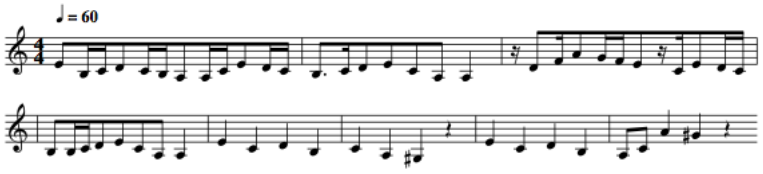

Abbildung 28: Referenznoten zu “Korobeiniki”

“Korobeiniki” ist ein russisches Lied, welches auf dem 1861 geschriebenen, gleichnamigen Gedicht von Nikolai Alexejewitsch Nekrassow (1821–1878) basiert. Korobeiniki waren Hausierer, die im vorrevolutionären Russland unter anderem Stoffe, Kurzwaren und Bücher verkauften. Es enthält insgesamt 56 Noten bzw. Pausen, wurde auch per MIDI-Saxophon aufgenommen und zusätzlich wieder von einer Akustikgitarre (Nylonsaiten), einer Violine, einem Akkordeon und einer Klarinette.
Der Grund für dieses Lied sind die vielen Sechzehntelnoten.

6.2 Datenbank

Um den Evaluationsprozess automatisieren zu können wurde eine Datenbank gewählt. Dadurch ist es möglich alle relevanten Informationen einer Referenzmelodie in einer Zeile unter zu bringen. Die MIDI-Melodien wurden mit Hilfe Apple's Garageband erstellt und dann über ein Mikrofon ganz normal aufgenommen.

Als Datenbank für die Referenzmelodien wurde MySQL gewählt. Um den Evaluationsprozess simpel und leicht erweiterbar zu halten und weil die Datenmenge relativ klein ist gibt es nur eine Tabelle mit dem Namen *Referenzmelodie.* Diese Tabelle enthält die folgenden Spalten:

1. id - INT - PRIMARY KEY
2. name - VARCHAR - Name bzw. Beschreibung der Melodie
3. WaveDatei - BLOB - Die Aufnahme im Wave-Format
4. referenznotenstring - VARCHAR - Die MIDI Keys und Notenwerte der Referenzmelodie, je als Paar: 60,2, 62,2, 64,2, 65,2, usw
5. BPM - Beats per minute
6. Tonart - VARCHAR - Tonart der Melodie, zB C
7. Taktart - VARCHAR - Taktart der Melodie, zB 4/4
8. transponierend - ob das Instrument transponierend ist (zB Bb Klarinette), 0 oder 1
9. Mic - VARCHAR - Mikrofon, mit welchem diese Aufnahme gemacht wurde
10. instrument - VARCHAR - Instrument, mit welchem diese Aufnahme eingespielt wurde

Somit ist es einfach neue Melodien hinzuzufügen um den Evaluationsvorgang zu erweitern. Jede Zeile hält alle für die Evaluation notwendigen Daten. Die Wavedatei (PCM signed, 16bit, mono, little-endian), der Referenznotenstring und die beats per minute sind die wichtigsten Parameter. Nach der Verarbeitung der Buffer in der Wavedatei können somit die erkannten Noten mit den Referenznoten verglichen und ausgewertet werden.
Jede der drei Melodien wurde mit fünf Instrumenten aufgenommen (MIDI, Gitarre, Klarinette, Violine, Akkordeon), woraus folgt, dass sich 15 Zeilen in der Tabelle befinden, welche je 15 mal durchlaufen werden. Das Ganze für jeden PitchDetektor ergibt 450 Durchläufe.

6.3 Bewertung der Noten

Bei der notenbasierenden Bewertung wird die Korrektheit von Notensequenzen evaluiert. Dazu muss das System die Noten der Referenzmelodie kennen, um diese mit den erkannten Noten vergleichen zu können. Es ist notwendig beide Richtungen zu betrachten, also welche Referenznoten in den Erkannten vorkommen und umgekehrt. Jede Note ist paarweise kodiert mit Notenhöhe und Notenwert. Die Tonhöhe ist MIDI Nummern entsprechend kodiert, die Tonlänge von 1(16tel) bis 16(ganze Note). Ein Treffer ist gegeben wenn beide Parameter übereinstimmen.

6.3.1 Recall und Precision

Zwei Bewertungsmaße bilden wichtige Werte für die Bewertung eines (Musik) Information Retrieval - Systems. Das Recall-Maß sowie das Prezäsions-Maß. Das Recall-Maß liefert eine Aussage über die *Vollständigkeit* der angezeigten Treffermenge, wohingegen das Prezäsions-Maß die *Genauigkeit* der Treffermenge angibt:

Recall*: (Vollständigkeit)* ist die Wahrscheinlichkeit dafür, wieviele Referenznoten, also *relevante* Noten, erkannt wurden

Precision: (*Genauigkeit*) ist die Wahrscheinlichkeit dafür, wieviele erkannte Noten relevant sind

Folgende Parameter spielen eine Rolle:

1. a - Die gefundenen, relevanten Treffer, also die Anzahl erkannter Noten, welche in der Referenzmelodie vorkommen
2. b - Die gefundenen nicht-relevanten Treffer, also die Anzahl erkannter Noten, welche nicht in der Referenzmelodie vorkommen
3. c - Die nicht gefundenen relevanten Treffer, also die Anzahl Referenznoten, welche nicht in der erkannten Melodie vorkommen

Recall- und Prezäsions-Maß sind dann definiert als:

$$recall = \frac{a}{a+c} * 100\%$$

$$precision = \frac{a}{a+b} * 100\%$$

Ein Maß, welches beide Werte kombiniert ist der harmonische Mittelwert von recall und precision, die sogenannte F-Score [PR]:

$$F = 2 * \frac{recall*precision}{recall+precision} * 100\%$$

6.3.2 Notenfehler E_n

Zusätzlich zu recall und precision wird noch nach einem weiteren Kriterium gemäß der folgenden Formel aus einer Arbeit von Matti Ryynänen zum Thema automatische Transkription [Ryynänen1] bewertet, dem Notenfehler E_n:

$$E_n = \frac{1}{2}\left[\frac{c_R - v_R}{c_R} + \frac{c_T - v_T}{c_T}\right] * 100\%,$$

mit folgenden Parametern:

1. c_R Anzahl Noten in der Referenzmelodie

2. c_T Anzahl erkannter Noten
3. v_R Anzahl Referenznoten, welche in der erkannten Melodie vorkommen (Wieviele Relevante wurden erkannt?)
4. v_T Anzahl erkannter Noten, welche in der Referenzmelodie vorkommen (Wieviele Erkannte sind relevant?, entspricht a)

Der symmetrische Ansatz, welcher beide Richtungen betrachtet ist notwendig weil die Verwendung von v_R oder v_T alleine unter Umständen zu Unvollständigkeiten führen würde. Es ist wichtig zu wissen wieviele Referenznoten erkannt wurden und wieviele erkannte Noten relevant sind.

Die daraus folgende Notenerkennungsrate N_n in Prozent ergibt sich als:

$$N_n = 100 - E_n$$

6.3.3 Beispiel

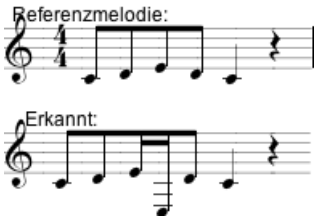

Abbildung 29: oben: Referenzmelodie, unten: Erkannte Melodie

Kodierung:

1. Referenzmelodie: [(60, 2) , (62, 2), (64, 2), (62, 2), (60, 4), (0, 4)] $c_R = 6$
2. Erkannte Melodie: [(60, 2) , (62, 2), (64, 1), (52, 1), (62, 2), (60, 4), (0, 4)] $c_T = 7$

v_R ergibt sich zu 5 aus 6. v_T ist hier auch gleich 5 aus 7. Daraus ergibt sich nach Anwendung der oben vorgestellten Formel ein Notenfehler $E_n = 22.62\%$, und somit eine Notenerkennungsrate $N_n = 77.38\%$.

F beträgt bezogen auf dieses Beispiel dann 76.92% bei einem recall von 83.33% und precision von 71.42% (a=5, b=2, c=1).

6.3.4 Mögliche Bewertungskriterien

Zusätzlich wäre es noch möglich zu bewerten, wie weit ein fälschlich erkannter Ton vom Richtigen entfernt liegt. Somit könnte eine Art Konfusionsmatrix erstellt werden, ähnlich wie bei der Bewertung eines Klassifikators und man würde eine genauere Einsicht in die Fehlerverteilung erhalten.
Jedoch bin ich der Ansicht dass eine Bewertung bzgl. der Vollständigkeit und Genauigkeit des Systems ausreichend ist.

6.4 Evaluationsparameter

Folgende Parameter spielen in der Evaluation zusätzlich eine Rolle:

1. PitchDetektoren: Je ein ganzer Evaluationsdurchlauf für YIN und McLeod um beide Algorithmen zu vergleichen
2. Buffersize: 512, 1024, 2048 mit jeweils 0, 10, 25, 50 und 75% Bufferoverlap, d.h. 15 Runden pro Lied
3. Die Collector-Constraints "MinimumDuration" und "MinimumLevel"
4. Instrumentenbasiert: Es werden für jedes Instrument gemäß der Spalte "instrument" die Erkennungsraten summiert, um dessen durchschnittliche Erkennungsrate zu berechnen. So lässt sich die Erkennungsrate des Systems bezogen auf einzelne Instrumente ausdrücken.
5. Die beiden Schwellwerte der PitchDetektoren betragen immer 0.13(YIN) und 0.93(MPM)

Für jeden der beiden PitchDetektoren YIN und MPM wird jede Zeile der Datenbanktabelle "Referenzmelodie" geholt und dann an die Transkriptionsengine geschickt. Jede dieser Melodien wird dann 15 mal mit allen möglichen Buffersize/Bufferoverlap Kombinationen transkribiert. Der Durchschnitt von N_n sowie F kann somit am Schluß ausgegeben werden.

Zusammenfassend werden also 2 Evaluationsmaße eingesetzt: Die Notenerkennungsrate N_n und der harmonische Mittelwert von recall und precision F. All diese Werte werden für jede Melodie berechnet und deren Durchschnitt ausgegeben. Somit lässt sich eine durchschnittliche Erkennungsrate des Systems bezogen auf alle Referenzmelodien und auf den jeweiligen PitchDetetktor ausdrücken.

6.5 Ergebnisse

Da jede Zeile, d.h. jede Melodie aufgrund der Buffersize- und Overlapeinstellungen 15 mal durchlaufen wird, ist es nützlich dort je die *beste* Erkennungsrate zu sammeln um somit eine durchschnittliche Erkennungsrate bezogen auf die Besten pro Zeile anzugeben**.**

6.5.1 Durchlauf #1

Parameter:
Minimumduration: 50ms
Minimumlevel: 20.0

	YIN	MPM
N_n	67.19%	70.93%
F	66.67%	70.14%
N_n (bezogen auf die Besten pro Melodie)	72.05%	77.75%

Instrumentenbasierende Evaluation: (Durchschnitt von N_n bezogen auf jedes instrument)

	N_n (YIN)	N_n (MPM)
MIDI	79.08%	77.51%
Akustikgitarre	42.44%	59.66%
Bb-Klarinette	71.43%	70.60%
Violine	70.65%	71.88%
Akkordeon	72.32%	75.00%

Auf den ersten Blick fällt auf dass der Durchlauf mit der McLeod Pitch-Method besser abgeschnitten hat. Bei der instrumentenbasierenden Evaluation ist erstens MIDI bei YIN und MPM relativ gleich, woraus man schließen kann dass beide sich im "perfekten" Fall ziemlich gleich verhalten, mit echten Instrumenten jedoch anders umgehen und zweitens ist YIN nur besser für die MIDI Melodien. Bei allen anderen zeigt MPM bessere Ergebnisse. Die Akustikgitarre hat deutlich am schlechtesten abgeschnitten.

6.5.2 Durchlauf #2

In Duchlauf #1 wurde das RMS-Level gemäß Kapitel 4.5 berechnet, was Werte zwischen ca. 0 und 70 brachte. Der Minimumlevel-Parameter betrug 20. In diesem Durchlauf wurde ein anderer Ansatz verfolgt. In der TarsosLib gibt es eine Funktion namens isSilence(). Diese errechnet auch den RMS-Wert und diesen danach in Dezibel um, gemäß folgender Formel. Ist dieser Wert nun kleiner als ein Geräuschlosigkeitsschwellwert (neuer Minimumlevel-Parameter), so gibt die Funktion isSilence() true zurück.

$$dB = 20 * log10(rms)$$

Parameter:
Minimumduration: 50ms
Minimumlevel: -70dB

	YIN	MPM
N_n	67.43%	71.68%
F	66.92%	70.69%
N_n (bezogen auf die Besten pro Melodie)	72.23%	78.82%

Instrumentenbasierende Evaluation: (Durchschnitt von N_n bezogen auf jedes instrument)

	N_n (YIN)	N_n (MPM)
MIDI	79.10%	77.48%
Akustikgitarre	43.55%	61.65%
Bb-Klarinette	71.80%	70.81%
Violine	70.71%	72.56%
Akkordeon	71.97%	75.92%

Es ergab eine minimale Verbesserung durch den neuen Ansatz.

6.6 Diskussion

6.6.1 Amplituden

Die oben vorgestellten Ergebnisse geben die durchschnittlichen Erkennungsraten des Systems für beide PitchDetektoren an. Man kann sehen dass die McLeod-PitchMethod besser abgeschnitten hat und somit eine durchschnittliche Erkennungsrate(N_n) von 71.68% erreicht wurde. Das ist akzeptabel, jedoch nicht umwerfend. Als erstes fiel mir auf dass die Amplitudeninformationen im Collectorprozess noch nicht berücksichtigt wurden. Folgende Abbildungen geben einen Überblick:

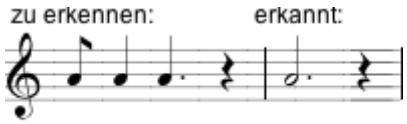

Abbildung 30: Zu erkennende und tatsächlich erkannte Noten

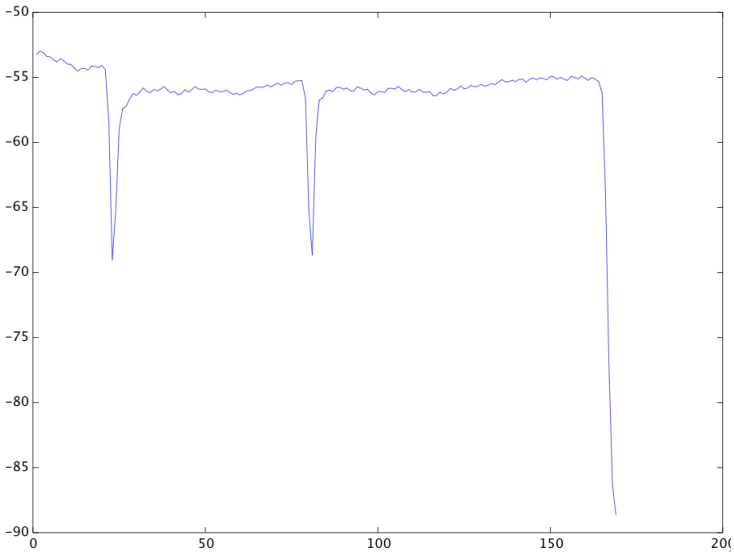

Abbildung 31: Plot der RMS-Amplitudenlevel in einer Notensequenz (A4 in "Joshua fought the battle of Jericho", 2. Takt, MIDI)

Die zu erkennende Notenlänge in diesem Takt entspricht insgesamt einer Viertel plus eine Achtel plus eine punktierte Viertel. Dies ergibt die Länge einer punktierten Halben Note. Das System hat anstatt 3 Noten hier nur eine Note erkannt, nämlich eine punktierte Halbe Note A4. Es sollte jedoch eine Achtel, eine Viertel und eine punktierte Viertelnote sein. In der Abbildung sind deutlich 3 lokale Minima zu sehen, woraus man schließen kann dass innerhalb dieser Notensequenz 3 Noten gespielt wurden, das System jedoch nur eine auswerten würde, da diese Minima in den Offset Bedingungen nicht berücksichtigt wurden. Deshalb wäre eine mögliche Verbesserung zusätzlich jede auszuwertende Notensequenz bezogen auf die RMS-Level auf lokale Minima zu untersuchen und wenn welche vorkommen, die Sequenz aufzuteilen um die selbe Note mehrmals zu malen.

6.6.2 Durchlauf #3

Dieser Ansatz brachte beim ersten Versuch folgende Ergebnisse:

	YIN	MPM
N_n	68.24%	72.04%
F	67.97%	71.76%
N_n (bezogen auf die Besten pro Melodie)	73.95%	79.58%

Instrumentenbasierende Evaluation:

	N_n (YIN)	N_n (MPM)
MIDI	85.50%	85.04%
Akustikgitarre	45.00%	59.52%
Bb-Klarinette	73.46%	73.82%
Violine	71.68%	72.73%
Akkordeon	65.55%	69.10%

Man kann sehen dass sich eine minimale Verbesserung ergeben hat. Diese Verbesserung bezieht sich hauptsächlich auf das Lied "Joshua fought the battle of jericho", woraus man schließen kann, dass das System nun immer wenn der selbe Ton mehrmals hintereinander gespielt wird, dieser auch entsprechend segmentiert wird. Für die MIDI-Melodien ergab sich eine deutliche Verbesserung, nur das Akkordeon hat sich um ca. 6% verschlechtert.

Die folgende Tabelle gibt einen Überblick über die jeweils besten Erkennungsraten (Durchschnitt von N_n) pro Melodie, zusammen mit den Parametern Buffersize und Bufferoverlap in Prozent bezogen auf beide PitchDetektoren:

YIN	Männlein	jericho	tetris
MIDI	91.88% (512/25%)	89.97% (512/10%)	93.70% (1024/25%)
Akustikgitarre	54.42% (1024/75%)	47.81% (1024/75%)	49.82% (2048/75%)
Bb-Klarinette	90.0% (1024/15%)	68.41% (512/0%)	77.22% (512/10%)
Violine	84.62% (512/10%)	65.01% (1024/10%)	80.723% (1024/10%)
Akkordeon	70.76% (2048/10%)	62.31% (512/50%)	82.64% (1024/75%)

MPM			
MIDI	89.83% (1024/0%)	88.63% (1024/75%)	94.58% (1024/0%)
Akustikgitarre	78.84% (512/25%)	62.86% (512/10%)	68.62% (512/25%)
Bb-Klarinette	87.14% (1024/75%)	74.87%(1024/25%)	79.65% (512/10%)
Violine	84.62% (1024/25%)	69.08% (512/50%)	82.63% (2048/10%)
Akkordeon	72.55% (2048/10%)	73.58% (1024/75%)	86.31% (2048/75%)

Durch diese Tabelle ist es möglich die Parameter Einstellungen für die Besten Ergebnisse pro Melodie bezogen auf jedes Instrument zu vergleichen. Nochmal zu erwähnen ist, dass beide PitchDetektoren äußerst gute Ergebnisse für die MIDI-Melodien erreichen. Klarinette, Akkordeon und Violine sind alle akzeptabel, nur die Gitarre schneidet deutlich am Schlechtesten ab. Das beste Ergebnis ergab sich für die Tetris-Melodie mit 94.58%.

6.6.3 Weiterer Ansatz

Ein weiterer Ansatz wäre, die Pipelineschritte Feature Vektor Extraktion und Segmentierung zu vertauschen. Es wird also zuerst segmentiert, zum Beispiel basierend auf Maxima im Signal, und danach die fundamentalen Frequenzen berechnet um somit innerhalb der Segmente eine Entscheidung treffen zu können. Wird kein Wert darauf gelegt, dass man die Noten sehen kann während man spielt, so könnte man die gesamte Aufnahme zuerst segmentieren und dann eine Ausgabe machen. Will man die Noten jedoch in Echtzeit sehen, so muss schon während der Aufnahme segmentiert werden. Dieser Ansatz wurde jedoch in dieser Arbeit nicht weiter verfolgt.

6.6.4 Interpretation

Einige Fehler wurden ziemlich wahrscheinlich auch dadurch verursacht, dass manche Melodien nicht hundertprozentig perfekt, also fehlerhaft eingespielt wurden und aufgrund den nicht allzu guten Aufnahmebedingungen (Handelsübliche Mikrofone, normales Zimmer). Das ist der Grund warum die Melodien auch in reinem MIDI aufgenommen wurden. Für die MIDI Melodien ergab der PitchDetektor YIN mit einer Erkennungsrate N_n von 85.50% das beste Ergebnis. Dies zeigt, dass das System im perfekten Fall relativ akkurat ist.

Abbildung 35 in Kapitel 8 zeigt die Transkription für “Joshua fought the battle of Jericho”, eingespielt mit Akustikgitarre. Obwohl das System für dieses Lied nur eine Erkennungsrate von $N_n = 47.82\%$ erreicht, so stellt man jedoch bei genauerer Betrachtung der Transkription oder durch Anhören dieser als MIDI Datei fest, dass das Ergebnis eigentlich doch sehr gut ist. Es sind überwiegend Fehler in der Erkennung der Notenwerte, die Noten selbst sind nahezu immer richtig erkannt worden. Meist ist der erkannte Notenwert ein bisschen zu lang oder ein bisschen zu kurz. Mit ein wenig Nachbearbeitung sind die Fehler schnell ausgebessert.

6.6.5 Zusammenfassung

Das Ziel dieser Arbeit war die automatische Transkription einer monophonischen Aufnahme. Alles in allem bin ich der Ansicht dass ich dieses Ziel erreicht habe. Eine durchschnittliche Notenerkennungsrate von 72.04% ist zwar nicht außergewöhnlich, jedoch akzeptabel. Wie schon gesagt liegt es auch an Fehlern und Störfaktoren in den Aufnahmen und abgesehen davon gibt es noch genug Raum für Verbesserungen.
Da die Noten selbst bis auf selten auftretende Oktavfehler überwiegend immer richtig erkannt werden ist es meiner Ansicht nach besser bei einem Optimierungsversuch zuerst den Segmentierungsprozess ins Auge zu fassen. Vor allem die Bedingungen für die Onsets und Offsets lassen einiges an Optimierung zu.

7 Ausblick

Zusammenfassend kann gesagt werden, dass ich das Ziel meiner Arbeit definitiv erreicht habe. Ein automatisches Notentranskriptionssystem in Form einer plattformunabhängigen, einfachen und leicht erweiterbaren Anwendung, welches akzeptablen Output in Form einer attraktiven Notentranskription ermöglicht und einfach zu nutzen ist.

Der in dieser Arbeit entwickelte Prototyp dient als gute Basis für spätere Erweiterungen wie zum Beispiel der Schritt von der Erkennung von monophonischem zu polyphonem Material. Es müssten nur einzelne Schritte in der Pipeline ausgetauscht werden und die PitchDetektoren sind leicht auswechselbar bzw. erweiterbar. Der Prototyp ist akzeptabel genug um produktiv eingesetzt werden zu können und ich erhielt positive Kritiken von einigen Musikern. Mit ein wenig Einarbeitungszeit in die abc-notations-Syntax kann nun ein Musiker viele Vorteile aus der Anwendung ziehen.

Streicher zum Beispiel ziehen Vorteile daraus, da es bei Streichinstrumenten keine Bünde gibt. Somit ist es gerade für Anfänger sehr schwierig, den Ton exakt zu treffen da es keine Anhaltspunkte gibt. Man kann überprüfen ob die gespielten Töne stimmen. Anfängern hilft es beim Noten lernen, fortgeschrittenen Musikern beim Üben von Tonleitern da das System genau das wiedergibt was eingespielt wurde, für Sänger oder Bläser kann es sehr nützlich sein, um ihre Intonation zu analysieren.
Auch für Lehrer ist es von großem Nutzen Arrangements oder kleine Übungen einzuspielen, da diese nicht mühevoll und zeitintensiv handschriftlich niedergeschrieben werden müssen. Desweiteren kann solch eine Anwendung auch als Kompositionstool eingesetzt werden, was gerade in kleineren Bands oder Musikgruppen brauchbar ist.

Abgesehen davon wurde ein Fundament geschaffen welche andere Arten von Anwendungen leicht möglich macht wie zum Beispiel ein webbasiertes Transkriptionssystem oder eine Anwendung für Handys. Der Backendcode des Prototyps ist modular und simpel gehalten worden, wodurch Änderungen oder Portierungen in andere Programmiersprachen leicht realisierbar sein sollten.

8 Appendix

8.1 Transkriptionen

Im Folgenden werden einige der generierten Transkriptionen der drei Melodien präsentiert:

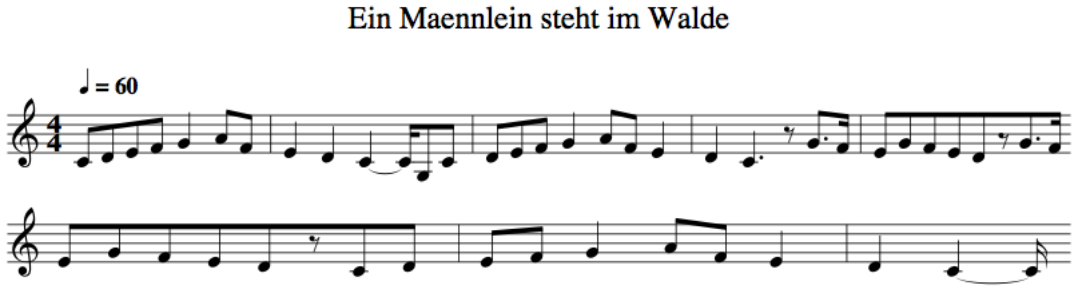

Abbildung 32: generierte Transkription für “Ein Männlein steht im Walde”, (Instrument: MIDI, PDA: YIN, $N_n = 91.88\%$)

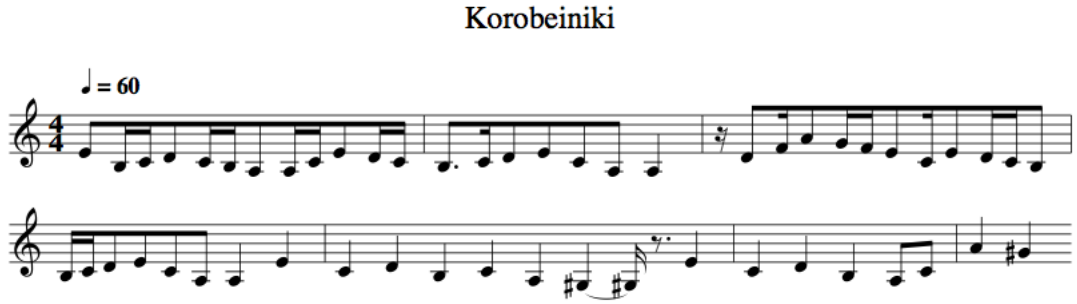

Abbildung 33: generierte Transkription für Korobeiniki”, (Instrument: MIDI, PDA: MPM, $N_n = 94.58\%$)

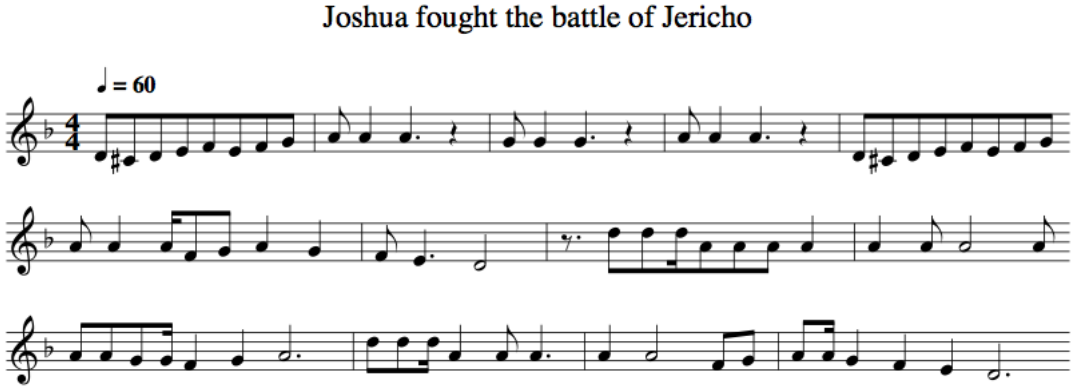

Abbildung 34: generierte Transkription für “Joshua fought the battle of Jericho”, (Instrument: MIDI, PDA: YIN, $N_n = 89.97\%$)

Zum Vergleich eine schlechtere Transkription:

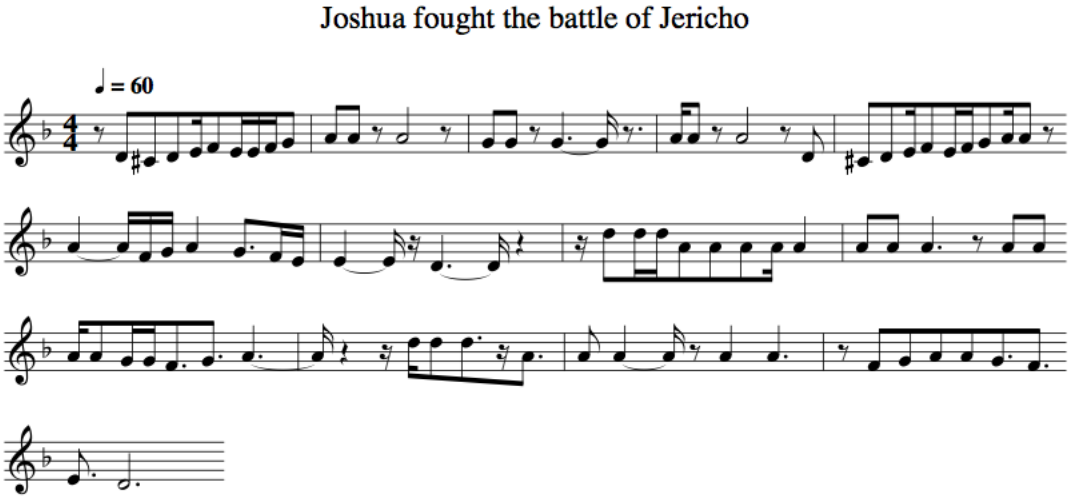

Abbildung 35: generierte Transkription für “Joshua fought the battle of Jericho”, (Instrument: Akustikgitarre, PDA: YIN, $N_n = 47.82\%$)

8.2 Die Referenzmelodien

Im Verzeichnis *Evaluation/* befinden sich die Aufnahmen der Referenzmelodien, ein SQL-Dump der Tabelle mit vollständigem Inhalt sowie die Ergebnisse der Evaluation.

9 Quellangaben

Literatur

[Cheveigné02] Cheveigné, A. "YIN, a fundamental frequency estimator for speech and music", Journal of the Acoustical Society of America, Vol 111(4), April 2002.
http://www.ircam.fr/pcm/cheveign/pss/2002_JASA_YIN.pdf
zuletzt besucht: August 2011

[McLeod-1] Philip McLeod, Geoff Wyvill "A smarter way to find pitch", University of Otago, Department of Computer Science
http://miracle.otago.ac.nz/tartini/papers/A_Smarter_Way_to_Find_Pitch.pdf
zuletzt besucht: August 2011

[McLeod-2] Philip McLeod "Tartini" Freies Musikanalyse Tool
http://miracle.otago.ac.nz/tartini/
zuletzt besucht: August 2011

[Savard] Alexandre Savard "Overview of Homophonic Pitch Detection algorithms", Schulich School of Music - McGill University, Canada

[Meffert04] Beate Meffert, Olaf Hochmuth "Werkzeuge der Signalverarbeitung", Pearson Studium 2004, ISBN: 3-8273-7065-5

[Ryynänen1] Matti Ryynänen "Probabilistic Modelling of Note Events in the Transcription of Monophonic Melodies", Tampere University of technology, Department of Information Technology, Institute of Signal Processing
www.cs.tut.fi/sgn/arg/matti/mryynane_thesis.pdf
zuletzt besucht: August 2011

[Ryynänen2] Matti Ryynänen, Anssi Klapuri "POLYPHONIC MUSIC TRANSCRIPTION USING NOTE EVENT MODELING", Institute of Signal Processing, Tampere University of Technology, Tampere, Finland
http://www.cs.tut.fi/sgn/arg/matti/ryynklapuri_polytrans_final.pdf
zuletzt besucht: August 2011

[Krige] W.A. Krige and T.R. Niesler "An HMM Based Singing Transcription System", Department of Electric and Electronic Engineering Stellenbosch University
www.dsp.sun.ac.za/∼trn/reports/krige+niesler_prasa06.pdf
zuletzt besucht: August 2011

[Martin99] Keith Dana Martin "Sound-Source Recognition: A Theory and Computational Model", MASSACHUSETTS INSTITUTE OF TECHNOLOGY, Juni 1999
http://xenia.media.mit.edu/∼kdm/research/papers/kdm-phdthesis.pdf
zuletzt besucht: August 2011

[Klich04] Ingmar-Leander Klich "Automatische Erkennung von Onsets in Musiksignalen zur Steuerung von Beattracking-Systemen", Fachgebiet Kommunikationswissenschaft, Institut für Sprache und Kommunikation, Fakultät I der Technischen Universität Berlin

[Monti2000] Giuliano Monti, Mark Sandler "MONOPHONIC TRANSCRIPTION WITH AUTOCORRELATION", Department of Electronic Engineering, King's College London, Strand, London WC2R 2LS, UK
profs.sci.univr.it/∼dafx/Final-Papers/pdf/Monti_DAFX00poster.pdf
zuletzt besucht: August 2011

[Haunschild98] Frank Haunschild "Die neue Harmonielehre", AMA Verlag GmbH 1998, ISBN: 3-927190-00-4

[Sikora03] Frank Sikora "Neue Jazz Harmonielehre", Schott Musik International 2003, ISBN: 3-7957-5124-1

[TARSOS] TarsosDSP: a small JAVA audio processing library http://tarsos.0110.be/

[ABC] Textbasiertes Musiknotationssystem http://abcnotation.com/

[ABC4J] java API to handle abc musical notation http://code.google.com/p/abc4j/

[ABCMIDI] http://abc.sourceforge.net/abcMIDI/

[LOGIC] http://www.apple.com/de/logicstudio/logicpro/

[INTELLISCORE] multi-instrument MP3 to MIDI converter http://www.intelliscore.net/

[MELODYNE] http://www.celemony.com/

[JavaSoundAPI] http://java.sun.com/j2se/1.3/pdf/javasound.pdf
zuletzt besucht: August 2011

[PR] http://en.wikipedia.org/wiki/Precision_and_recall

[Bild1] http://de.wikipedia.org/wiki/Schwingung

[Bild2] http://www.casualstrolltomordor.com/2011/02/the-123s-of-abcs-the-basics/
zuletzt besucht: August 2011

[Bild3] http://de.wikipedia.org/wiki/Notensystem_%28Musik%29

[Bild4] http://de.wikipedia.org/wiki/Dynamik_(Musik)