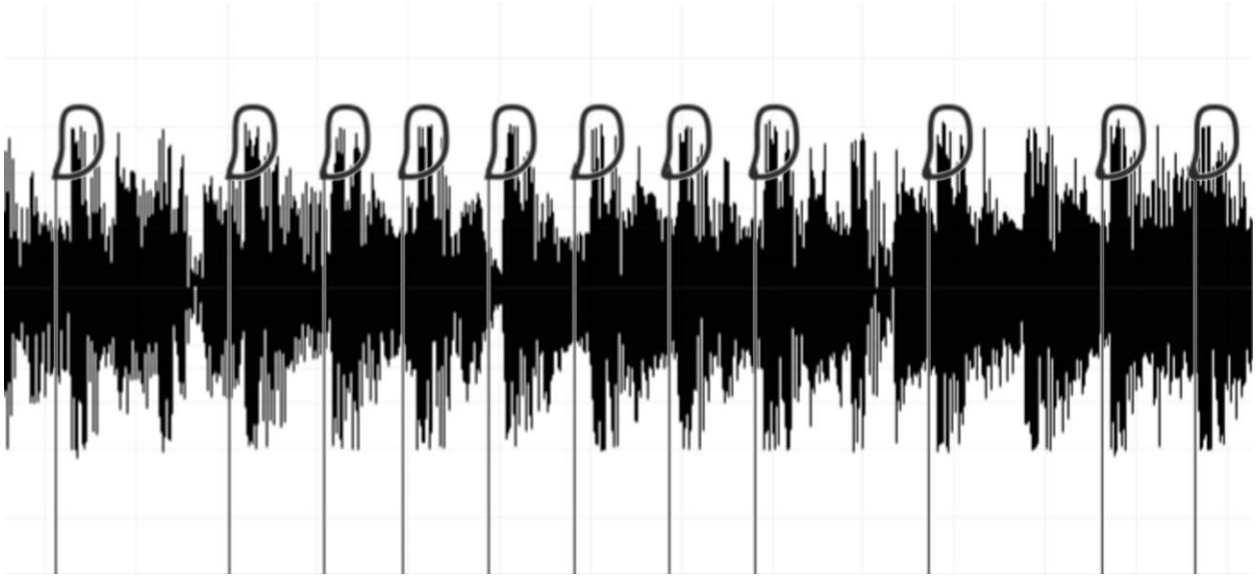

BPM Finder

08.01.2016

Daniel Menke (2094704)

Stefan Schäfers (2175460)

HAW Hamburg

Audiotechnik und -produktion 2

Prof. Dr.-Ing. Eva Wilk

Inhaltsverzeichnis

Einleitung 3
1. Rhythmus und Tempo 4
1.1 Rhythmus 4
1.2 Das Tempo und der BPM-Wert 4
1.3 Problemstellungen für Algorithmen 5
1.3.1 Vielfache des BPM-Werts 5
1.3.2 Eindeutige Erkennung der Notenanschläge 7
2. Anforderungen an die Audiodatei 9
2.1 Technische Vorraussetzungen 9
2.2 WAVE-Dateien lesen mit der “WavFile”-Klasse 9
3. Allgemeiner Ansatz für alle Algorithmen 11
3.1 “beat tracking” 11
3.2 BPM ermitteln 11
4. Algorithmus 1: Signalenergie 12
4.1 Beat tracking: Beats über “Energie-Schübe” identifizieren 12
4.1.1 Energie 12
4.1.2 Einteilung in Segmente 13
4.1.2 Durchschnittliches Energieniveau um ein Segment 15
4.1.3 Identifizierung der Beats durch den “Onset” 17
4.2 Von den markierten Beats zu einem BPM-Wert 18
4.3 Nachteile und Probleme des Algorithmus 20
4.3.1 Zeitliche Unschärfe 20
4.3.2 Starke Signalabhängigkeit 21
5. Algorithmus 2: Peaks im tiefen Spektralbereich 22
5.1 “beat tracking”: BPM-Ermittlung durch Intervall-Analyse 22
5.1.1 Tiefpassfilterung 22
5.1.2 Peak-Positions-Erkennung 24
5.1.3 Intervall-Analyse 24
5.2 Ermittlung des BPM-Werts 25
5.2 Nachteile und Probleme des Algorithmus 27
6. Algorithmus 3: Wavelet-Dekomposition und Autokorrelation 28

6.1 “beat tracking”: Erkennen der Beats durch “onset detection” 28
6.1.1 Wavelet-Transformation 29
6.1.2 Umsetzung eines “onset-detection”-Algorithmus mittels DWT 32
6.2 Aus der “Detection function” die BPM ermitteln 40
7. Auswertung und Vergleich 44
7.1 Vorgehen 44
7.1.1 Relative Abweichung und Vielfache der BPM-Werte 44
7.1.2 Das Testsignal 45
7.2 Ergebnis 45
7.2.1 Genauigkeit der BPM-Werte 45
7.2.2 Dauer des Analysevorgangs 48
7.3 Nachtrag zum Vergleich 49
8. Das BPM-Finder Programm 50
8.1 Aufbau 50
8.2 Mediaplayer 51
8.3 Klassen und Funktionen 52
9. Zusammenfassung und Ausblick 53
9.1 Zusammenfassung 53
9.2 Verlauf des BPM-Werts über die Zeit 54
9.3 Ausblick 54
10. Literatur und Quellenangaben 56
11. Bildverzeichnis 58
11. Anhang 59
11.1 Ergebnisse der Auswertung 59
11.2 CD-ROM 64

Einleitung

Im Rahmen der “Audiotechnik und -produktion 2” Vorlesung haben wir uns in einer projektorientierten Arbeit mit der Entwicklung einer Software zur Ermittlung des Musiktempos beschäftigt. In unserem Konzept haben wir uns das Ziel gesetzt, den Verlauf des Musiktempos in Form von BPM-Werten aus einer ladbaren Audiodatei mit Musikinhalt zu analysieren.

Die Software soll über eine Oberfläche verfügen, die es dem Benutzer ermöglicht, eine Audiodatei über einen Dateiauswahl-Dialog in das Programm zu laden. Dort wird per Knopfdruck ein Analysevorgang gestartet. Anschließend soll der zeitliche Verlauf des Musiktempos als Ergebnis dargestellt werden. Zusätzlich verfügt das Programm über simple Mediaplayer-Funktionen, damit der Benutzer die Audiodatei abspielen kann.

Die Entscheidung für JAVA als Programmiersprache wurde getroffen, da JAVA ein breites Band an impliziten Funktionen bietet, objektorientiert ist und eine, im Verhältnis zu vergleichbaren Programmiersprachen, relativ flache Lernkurve hat. JAVA gilt zwar nicht als außerordentlich performant, da unser Fokus allerdings auf der grundlegenden Entwicklung der Algorithmen und deren Analyse liegt und nicht unbedingt auf einer möglichst zeiteffizienten Berechnung der Ergebnisse, bietet JAVA eine vergleichsweise einfache Programmierarbeit, da es Aufgaben wie Speicherverwaltung und Garbage-Collection für den Entwickler übernimmt. Ein weiterer Vorteil der Sprache ist ihre plattformunabhängigkeit, so kann beispielsweise auf Windows geschriebener Quellcode problemlos auf Mac OS oder Linux ausgeführt werden.

Für die Genauigkeit der BPM-Werte, das Format der Audiodatei und dessen Inhalt in Bezug auf die Musik haben wir spezielle Anforderungen bestimmt, die wir auch in dieser Ausarbeitung erneut erläutern.

In unserem Vorgehen haben wir uns zunächst mit verschiedenen Ansätzen für Algorithmen auseinandergesetzt, von denen insgesamt drei vorgestellt und untersucht werden. Anschließend werden die entwickelten und implementierten Algorithmen in einem gemeinsamen Testverfahren miteinander verglichen und unsere anfangs festgelegten Ziele überprüft.

1. Rhythmus und Tempo

1.1 Rhythmus

Der Wechsel von Tag und Nacht, der Wechsel der vier Jahreszeiten oder auch der eigene Herzschlag oder Atemvorgang sind nur wenige Beispiele für rhythmische Bewegungen, die man in der Natur findet (HEWITT, 2008, S. 40). Bei diesen Rhythmen lässt sich immer ein bestimmtes Zeitintervall (Periode) definieren, welches in sehr ähnlicher Form wiederholt wird.

In der Musik ist dieses Konzept auch wiederzufinden, dort sind diese Perioden in Takten organisiert. Innerhalb eines Taktes können Einsätze von verschiedenen Schallereignissen oder Stille mit verschiedenen zeitlichen Dauern angeordnet sein, wobei jeder Takt immer eine definierte Dauer gruppiert. Im Gegensatz zu den Rhythmen in der Natur, werden die Perioden bzw. musikalischen Motive nicht streng wiederholt, sondern oft in abgewandelter Form.

1.2 Das Tempo und der BPM-Wert

Ein Rhythmus kann verhältnismäßig schnell oder langsam stattfinden. Erst mit einer Angabe über die zeitliche Dauer einer Periode ist die Geschwindigkeit bzw. das Tempo eines Rhythmus bestimmt. (vgl. HEWITT, 2008, S. 41)

In der Musik sind die Längen bzw. Dauern der einsetzenden Schallereignisse und Pausen alle relativ zueinander organisiert. Zwei Viertelnoten dauern zum Beispiel so lange an wie eine halbe Note, ein Takt fasst eine Anzahl an Notenwerten zusammen. Bei einem Viervierteltakt (4/4) sind pro Takt Notenwerte angeordnet, die zusammengefasst immer den Notenwert von einer ganzen Note (entspricht vier Viertelnoten) ergeben.

In der Notation steht meist zu Beginn, über dem ersten Notensystem, eine kleine Note mit einem Gleichheitszeichen und einem Wert dahinter (siehe Abbildung 1). Dieser Wert wird als BPM-Wert bezeichnet, BPM steht dabei für “beats per minute”, also “Schläge pro Minute”. Ein beat bzw. Schlag entspricht dabei dem Notenwert, der vor dem Gleichheitszeichen angegeben ist. Über diesen Wert ist also definiert, wie lange der entsprechende Notenwert zeitlich klingt. Da jeder andere Notenwert sich relativ aus dem angegebenen Notenwert ableiten lässt, ist somit das Tempo für das gesamte Musikstück bestimmt.

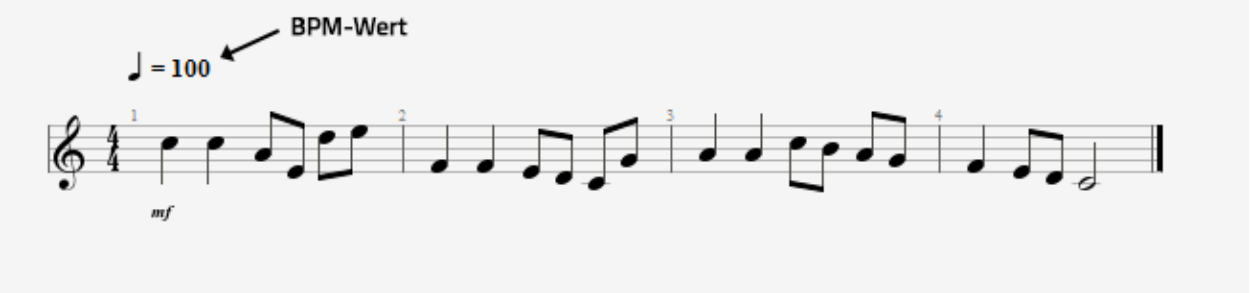

Abbildung 1: Notenbeispiel mit Viervierteltakt und BPM-Wert Angabe (Tempoangabe)

1.3 Problemstellungen für Algorithmen

1.3.1 Vielfache des BPM-Werts

Kennt man die Notation eines Musikstücks nicht, sondern hört es nur, dann erscheint in den meisten Fällen eine gleichmäßig verteilte rhythmische Folge als am stärksten hervorgehoben. Diese rhythmische Folge erscheint nach außen zum Beispiel durch ein gleichmäßiges Mitwippen mit dem Fuß (vgl. SCHEIRER, 1998, S. 588). Zählt man für eine Minute die Anzahl der Schläge dieser Folge, erhält man einen BPM-Wert. Die Dauer dieses Schlags bzw. dieser Zählzeit könnte theoretisch jedem beliebigen Notenwert zugeordnet werden. Trotzdem können alle auftretenden rhythmischen Strukturen dann um das gewählte rhythmische Raster aufnotiert werden. Die folgende Abbildung 2 soll dieses Problem verdeutlichen. Es sind drei Notenbeispiele zu sehen, die jeweils unterschiedliche BPM-Werte tragen. Trotzdem klingen alle drei identisch, wenn man sie gespielt hört.

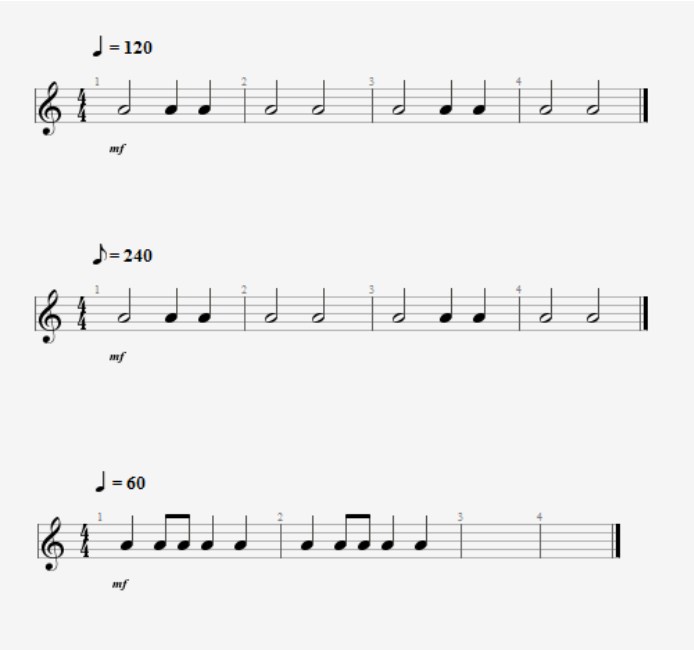

Abbildung 2: Alle drei Notenbeispiele klingen gespielt identisch, haben allerdings verschiedene BPM-Wert Angaben.

Bei dem Beispiel aus Abbildung 2 erscheint beim Hören eine metrische Ebene als am stärksten hervorgehoben. Der zugehörige BPM-Wert beträgt 60. Das liegt daran, dass sich diese hervorgehobene Ebene oft an den am häufigsten verwendeten Notenwerten orientiert. Bei den ersten zwei Notenbeispielen ist es die halbe Note, die 60 mal pro Minute angeschlagen wird. Bei dem dritten Notenbeispiel ist es die Viertelnote, die 60 mal pro Minute angeschlagen wird. Auch wenn diese Gemeinsamkeit nun eigentlich zu einem vermeintlich eindeutigen Ergebnis für den BPM-Wert führt, kann es sein, dass es sich nur um einen Ausschnitt aus einem Musikstück handelt, bei dem zeitweise die halbe Zählzeit betont wird. Ein Beispiel ist dafür ist der Wechsel zum “Halftime” im Break des Songs “Choose To Be Me” von Sunrise Avenue. Aber auch ganz unabhängig von eindeutigen Wechseln in bestimmten Songstrukturen können abschnittsweise Notenwerte in ihrer Anzahl gegenüber anderen überwiegen und so die doppelte oder halbe metrische Ebene betonen.

Da Notenwerte und BPM-Werte alle relativ zueinander organisiert sind, kann ein BPM-Wert immer wieder verdoppelt werden, ohne dabei seine Aussage über das Musiktempo zu verlieren, solange der verknüpfte Notenwert auch entsprechend neu gewählt wird. Aus diesem Grund werden beispielsweise 60 BPM und 120 BPM als gleichwertig gültige Ergebnisse für die Ermittlung des BPM-Werts durch einen Algorithmus gewertet.

Als Grenzen für den BPM-Wert wurden 60 BPM und 200 BPM gewählt. Außerdem soll der BPM-Wert, den das Programm ausgibt, eine ganze Zahl ohne Nachkommastellen sein, Ziel ist ein BPM-Wert mit einer relativen Genauigkeit von 10%.

1.3.2 Eindeutige Erkennung der Notenanschläge

Der Beginn einer neuen Note kann über die Änderung in zwei wesentlichen Prinzipien erkannt werden:

1. Über den **Einschwingvorgang**, also zum Beispiel das neue Anzupfen einer Gitarrensaite.
2. Über die **Tonhöhenänderung**, für den Fall, dass sich die Tonhöhe der Note ändert. (z.B.: *Masataka, Goto; Yoichi, Muraoka: Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions*)

Desweiteren können sich noch zeitliche Änderungen in der Klangfarbe (“tone color evolve”) ergeben, allerdings kann ohne irgendeine Änderung keine musikalische Bedeutung erkannt werden. (vgl. BELLO et al., 2005, S. 1035)

Der Beginn des Einsatzes einer neuen Note kann durch den Einschwingvorgang immer dann gut erkannt werden, wenn dieser sehr impulshaftig ist. Bei Instrumenten, die beim Spielen der Noten ein sehr langes Einschwingverhalten aufweisen und somit vergleichsweise weich in in die nächste Note gleiten, ist dieses Verfahren eher ungeeignet. Ein besonders scharfes Einschwingverhalten haben vor allen Dingen Percussion Instrumente.

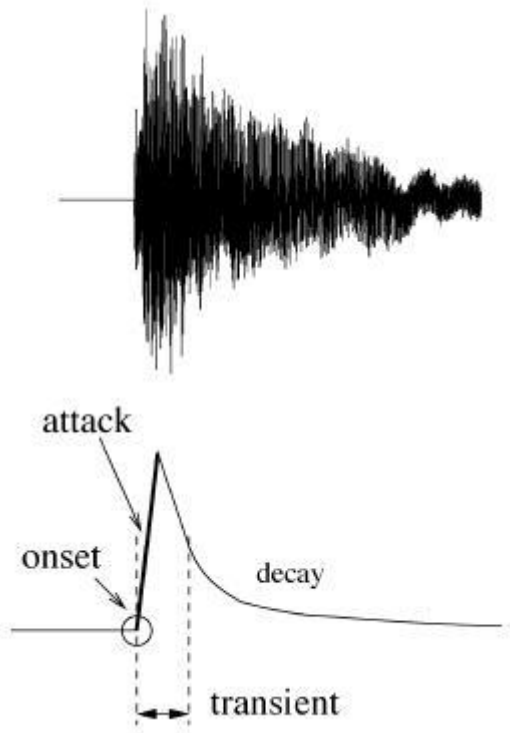

Abbildung 3: Ideale Darstellung der Hüllkurvenabschnitte “onset”, “attack”, “decay” und “transient”

Quelle: Bello, Juan Pablo et al., 2005, S. 1035, Fig. 1.

In Abbildung 3 sind einige idealisierte Abschnitte der Hüllkurve einer einzelnen Note dargestellt. Während “Attack” ein Zeitintervall beschreibt, in dem die Amplitude ansteigt und “Decay” ein Zeitintervall ab der maximalen Amplitude, in dem die Amplitude wieder auf ein gewisses Level (“Sustain”) zurückfällt, sind die Begriffe “onset” und “transient” nicht so einfach zu definieren (BELLO et al., 2005, S. 1035f.). Der zeitliche Abschnitt “transient” umfasst ein kurzes Intervall, in dem die Amplitude “[…] nicht-trivial, relativ unvorhersehbar […]” (BELLO et al., 2005, S. 1036, Z. 3f.) ansteigt oder abfällt. Der “onset” soll einen Zeitpunkt definieren, ab dem der Transient zuverlässig erkannt werden kann. (BELLO et al., 2005, S. 1036.) Für impulshafte Signalverläufe sind die Transienten sehr scharf, das bedeutet, die Attack-Zeit ist verhältnismäßig kurz.

Da wir uns nur mit Algorithmen beschäftigt haben, die über das Einschwingverhalten rhythmische Strukturen erkennen, ergeben sich somit einige Anforderungen für den musikalischen Inhalt der zu untersuchenden Audiodateien. Wie bereits im Konzept erwähnt, wird von den Algorithmen ein Musikstück erwartet, in dem ein Instrument aus der Rhythmussektion (Percussion) durchgängig spielt. Zusätzlich sollten diese Instrumente sehr direkt gemischt sein, das heißt im Verhältnis wenig Raumanteil besitzen. Andernfalls könnten die Einsätze nicht mehr sicher identifiziert werden.

2. Anforderungen an die Audiodatei

2.1 Technische Vorraussetzungen

Im letzten Abschnitt wurden einige inhatliche Anforderungen der Audiodateien vorgestellt. Auch für die technischen Daten gibt es Voraussetzungen:

- ➔ Dateiformat: WAV (PCM), unkomprimiert
- ➔ Auflösung: 16 Bit
- ➔ Abtastfrequenz: 44100 Hz

Da die Algorithmen Informationen über das Songtempo aus der Wellenform ermitteln, ist es erforderlich, dass die zeitdiskreten Werte für alle Samples zur Verfügung stehen. Komprimierte Formate müssten dekodiert werden. Um diesen Schritt zu sparen, erwartet das Programm direkt eine Datei im Format WAVE, in der die PCM-Rohdaten vorliegen und einfach ausgelesen werden können.

2.2 WAVE-Dateien lesen mit der “WavFile”-Klasse

Zum Einlesen der WAVE-Datei wird die frei verfügbare “Java Wav File IO” Klasse, geschrieben von Dr. Andrew Greensted (http://www.labbookpages.co.uk/audio/javaWavFiles.html), verwendet. Mit Hilfe der Funktionalität der Klasse wurde die Methode “getWavData()” entwickelt. (siehe Code 1)

Zunächst wird mit der Methode “openWavFile(currentFile)” ein WavFile-Objekt erzeugt; “currentFile” ist dabei ein File-Objekt, das mit der vom Benutzer ausgewählten Datei erzeugt wurde. Anschließend werden die Anzahl der Samples und die Anzahl der Kanäle der WAVE-Datei ausgelesen. Mit diesen Informationen wird eine zweidimensionale Array vorbereitet, in welche dann mit der Methode “readFrames” die Samples aus der WAVE-Datei in Form von double-Werten eingesetzt werden. Diese Array wird dann von der Methode zurückgegeben.

```
public double[][] getWavData() {
	try {
		// WAV-Datei öffnen.
		wavFile = WavFile.openWavFile(currentFile);

		// Die Gesamtanzahl der Samples in der zu untersuchenden WAV-Datei
		long totalFrames = wavFile.getNumFrames();
		int totWAVsamples = (int) totalFrames;

		// Anzahl der Kanäle (1=Mono, 2=Stereo)
		int numChannels = wavFile.getNumChannels();

		// Buffer der entsprechenden Größe anlegen:
		// -> erste Dimension der Array: Kanalauswahl
		// -> zeite Dimension der Array: einzelne Sample-Werte
		double[][] buffer = new double[numChannels][totWAVsamples];

		// Segment in den Buffer laden.
		int samplesRead = wavFile.readFrames(buffer, totWAVsamples);

		// WAV-Datei schließen.
		wavFile.close();

		return buffer;

	} catch (Exception e) {
		System.out.println(e);
	}

	return new double[0][0];
}
```

Code 1: Die Methode “getWavData()”, um die Samples einer WAVE-Datei in eine Java-Array zu laden.

3. Allgemeiner Ansatz für alle Algorithmen

Grundsätzlich kann das Arbeiten der Algorithmen in zwei wesentliche Schritte eingeteilt werden.

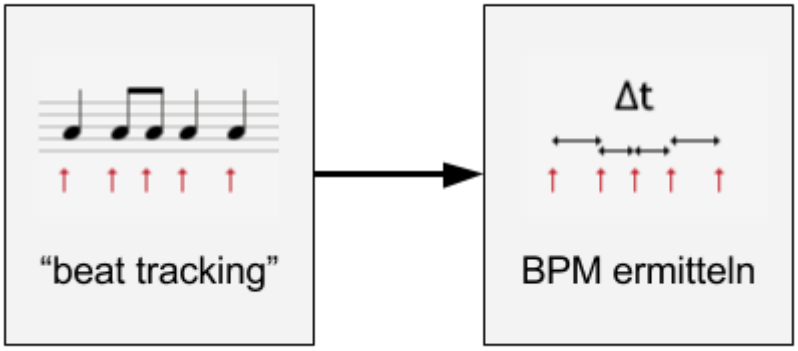

Abbildung 4: Schema der zwei Schritte bei der Ermittlung des BPM-Werts

3.1 “beat tracking”

Beim “beat tracking” geht es darum, die einzelnen rhythmischen Bestandteile aus einem Musikstück zu extrahieren. Wie in Abschnitt 1.3.2 beschrieben, versuchen die von uns untersuchten Algorithmen aus möglichst impulshaften Einschwingvorgängen der einsetzenden Schallereignisse rhythmische Strukturen zu erkennen. Die Schwierigkeit beim “beat tracking” ist es, diese Beats aus den zeit- und wertediskreten Samples der Audiodatei zu ermitteln. Dazu existieren verschiedene Ansätze und Möglichkeiten, einige davon werden in den Kapiteln zu den einzelnen Algorithmen genauer beschrieben und vorgestellt.

3.2 BPM ermitteln

Alle gefundenen Beats aus dem “beat tracking” werden anschließend in sinnvoller Weise sortiert und ausgewertet. Es geht in diesem Schritt darum, aus den gefundenen Werten einen finalen BPM-Wert als Ergebnis für das Musiktempo zu bilden. Im einfachsten Fall lässt das zeitliche Intervall zwischen den Beats einen Rückschluss auf das Tempo zu.

4. Algorithmus 1: Signalenergie

Dieser Algorithmus ist in der JAVA-Klasse “SoundEnergyAlgorithm” (SoundEnergyAlgorithm.java) umgesetzt.

4.1 Beat tracking: Beats über “Energie-Schübe” identifizieren

Die Wahl eines geeigneten Ansatzes, um Beats bzw. Onsets (vgl. Abschnitt 1.3.2) zu erkennen, orientiert sich sehr stark an den Instrumenten, welche die entsprechenden Noten spielen. Da für den Inhalt der Audiodateien ein spielendes Instrument aus der Rhythmussektion (Percussion) gefordert wird, kann sich zu Nutzen gemacht werden, dass die Transienten am Beginn von Soundeinsätzen der Percussion-Instrumente immer mit kurzen “[...] bursts of energy [...]” einhergehen. (BELLO et al., 2005, S. 1045, Sp. 2, Z. 21-25) Um den Einsatz einer neuen Note zu erkennen, müssen also diese kurzen Zeitintervalle erkannt werden, in denen “Energie-Schübe” stattfinden.

4.1.1 Energie

Die Energie eines zeitkontinuierlichen Signals berechnet sich zu

$$E_x = \int_{-\infty}^{\infty} |x(t)|^2 dt \,. \tag{1}$$

(MERTINS, 2012, S. 7)

Ist $x$ein unendlich langes, aber zeitdiskretes Signal, dann gilt

$$E_x = \sum_{n=-\infty}^{\infty} |x(n)|^2. \tag{2}$$

(MERTINS, 2012, S. 24)

4.1.2 Einteilung in Segmente

Über die zeitliche Länge eines Transienten kann nicht viel ausgesagt werden (vgl. Abschnitt 1.3.2), das menschliche Ohr kann aber nicht zwischen zwei Transienten differenzieren, die weniger als 10 ms auseinanderliegen. (vgl. BELLO et al., 2005, S. 1036, Z. 10ff.) Bei einer Abtastfrequenz von 44100 Hz entsprechen 10ms einer Sampleanzahl von

$$N = t \cdot f_s = 0{,}01\ s\ \cdot 44100\ Hz = 441.$$

Diese Anzahl an Samples wird gewählt, um damit die Energie-Schübe der Transienten zu identifizieren. Dazu wird das gesamte Audiosignal zunächst in Segmente eingeteilt, die jeweils 441 Samples zusammenfassen.
Für jedes Segment wird die Energie nach Formel (2) berechnet. In der folgenden Abbildung 5 sind für einen Ausschnitt aus dem Lied “Memory Lane” von Netsky die einzelnen Energiewerte der Segmente grafisch dargestellt.

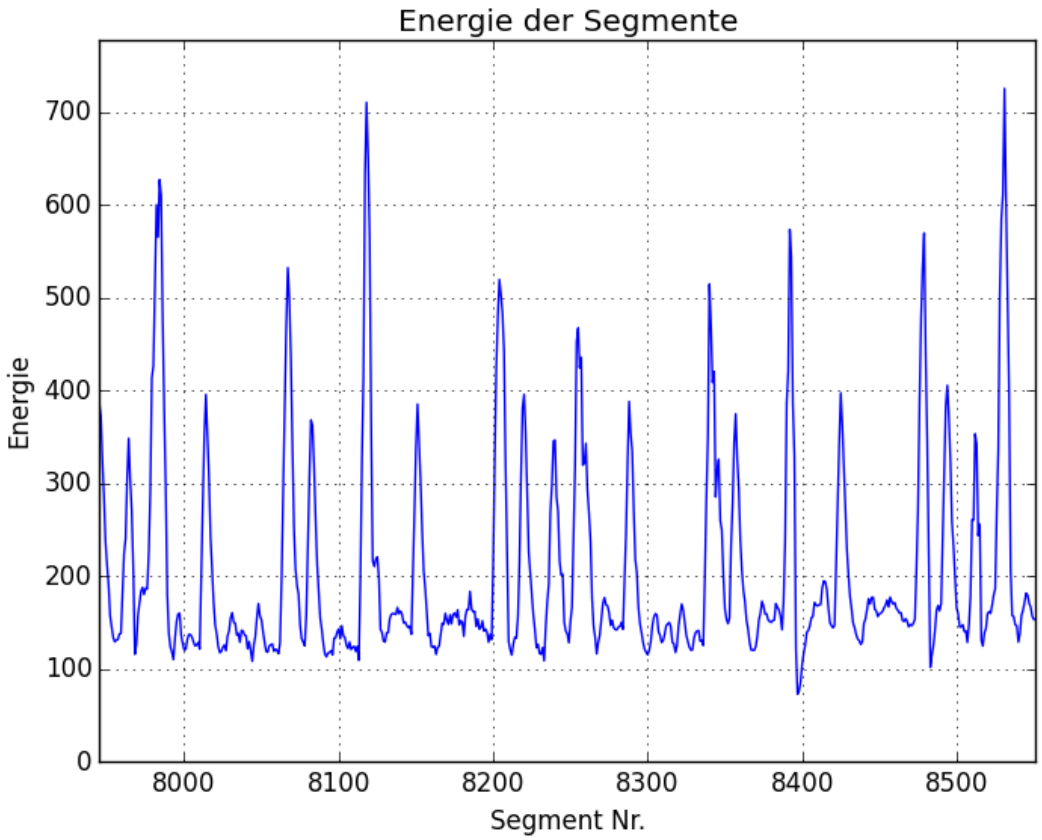

Abbildung 5: Ausschnitt der Energiewerte der einzelnen Segmente aus “Memory Lane” von Netsky

In der Abbildung sind die “Energie-Schübe”, die in kurzen Zeitintervallen auftreten, gut zu erkennen. Bei einem Gesamtblick auf die Energie aller Segmente des Musikstücks ist zu erkennen, dass sich das durchschnittliche Niveau der Energie in den Segmenten im Songverlauf entwickelt. (Abbildung 6) Bis zum Segment 5000 ist beispielsweise ein sehr geringes durchschnittliches Energieniveau zu erkennen, da es sich in diesem Abschnitt um das Intro des Songs handelt. Die Identifizierung, ob die Energie in einem bestimmten Segment nun gerade einem “Energie-Schub” zuzuordnen ist, sollte also mit einem zeitlichen Bezug zum jeweils aktuellen Energieniveau erfolgen.

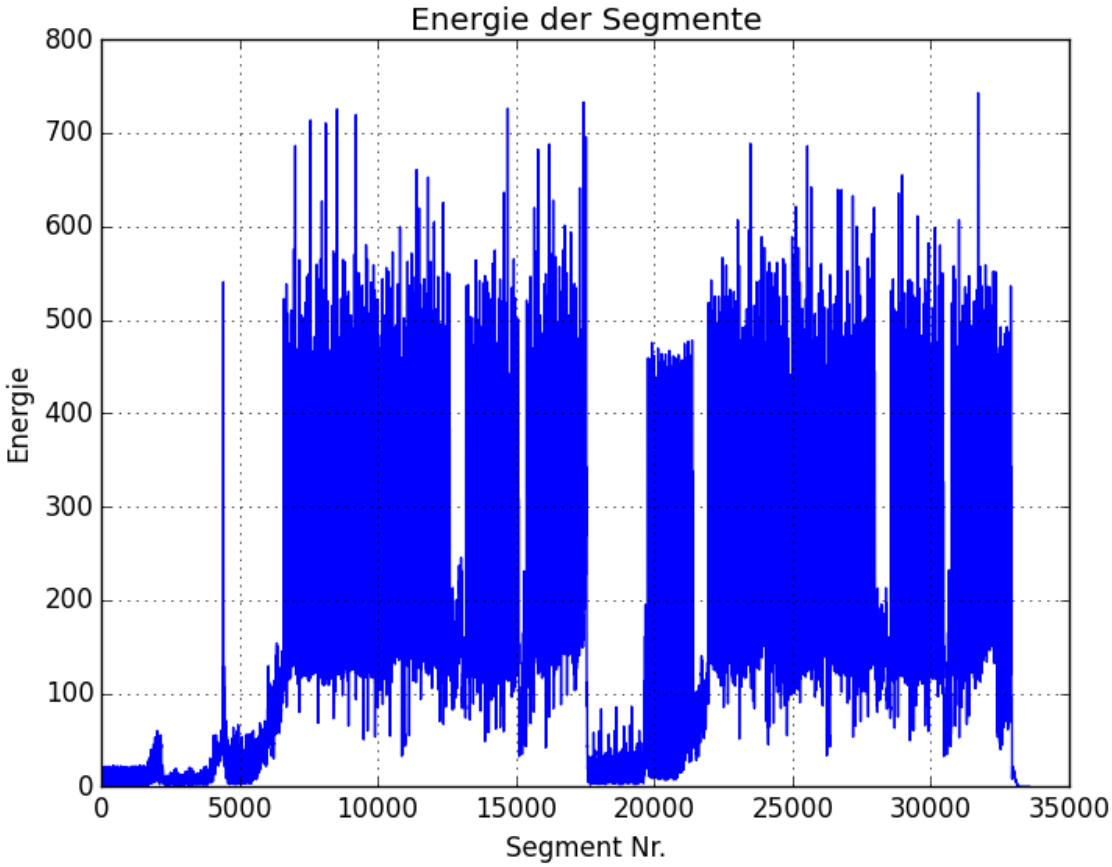

Abbildung 6: Alle Energiewerte der einzelnen Segmente aus “Memory Lane” von Netsky

4.1.2 Durchschnittliches Energieniveau um ein Segment

Wählt man beispielsweise das Segment Nr. 10000. Um das durchschnittliche Energieniveau um dieses Segment herum zu bestimmen, werden 50 Segmente (25 nach links und 24 nach rechts) um dieses Segment ausgewählt. In diesen 50 Segmenten wird der Mittelwert nach folgender Formel berechnet und ab nun als durchschnittliches Energieniveau bezeichnet.

$$\underline{E_{50Segmente}} = \frac{1}{50} \sum_{n=k}^{k+50} x(n) \tag{3}$$

$$k = 0,1,2,\ldots,z$$

z: ("Anzahl der Segmente" - 50)

In Abbildung 7 ist der Verlauf dieses durchschnittlichen Energieniveaus grafisch gezeigt. Werden mehr als 50 Segmente zusammengefasst, folgt der Energieniveau-Verlauf immer träger dem durchschnittlichen Energiewert und die zeitliche Auflösung wird geringer.

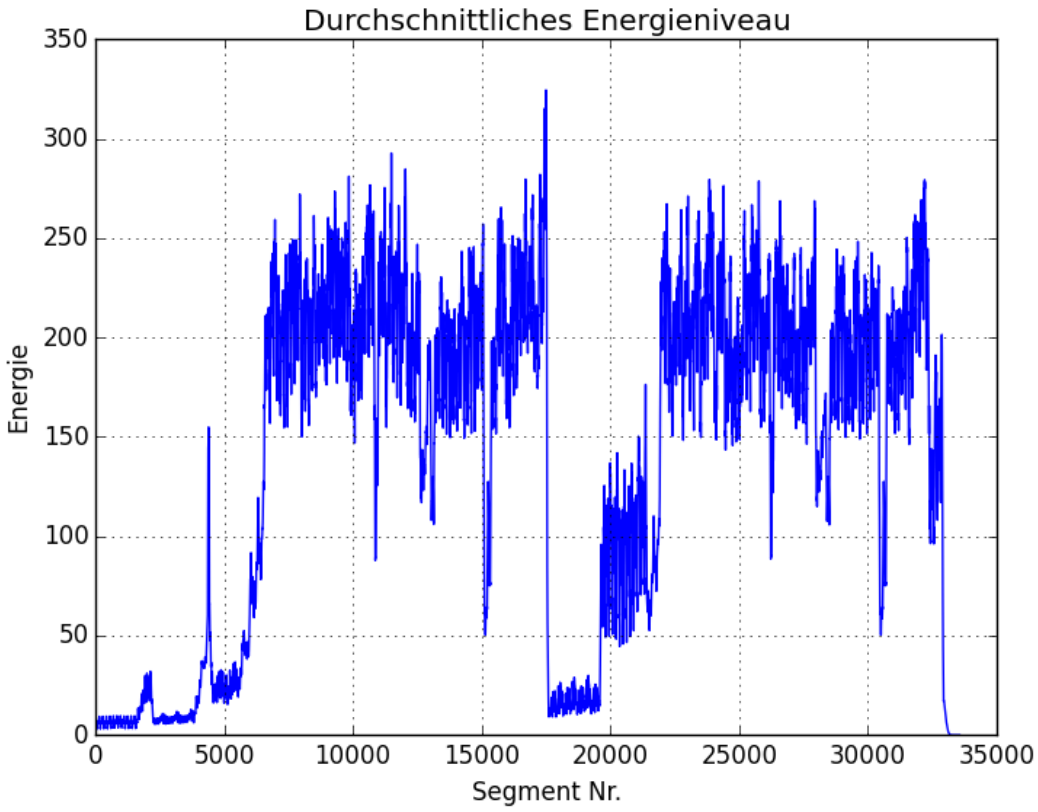

Abbildung 7: Durchschnittliches Energieniveau um ein Segment (aus "Memory Lane" von Netsky)

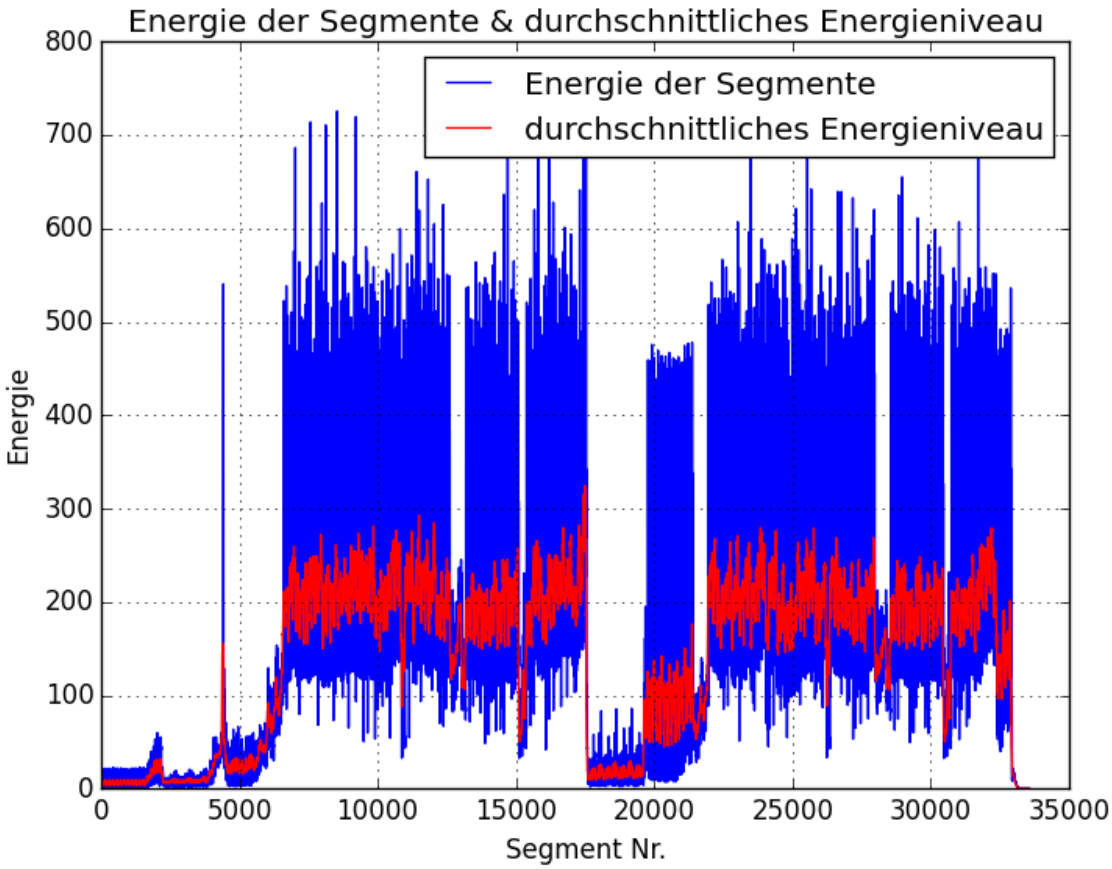

Abbildung 8: Energiewerte der Segmente und durchschnittliches Energieniveau um jedes Segment (aus “Memory Lane” von Netsky)

Noch deutlicher wird das durchschnittliche Energieniveau sichtbar, wenn man es zusammen mit den Energiewerten der einzelnen Segmente darstellt (Abbildung 8). In Abbildung 9 ist wieder ein Ausschnitt gewählt. Hier ist sehr gut ersichtlich, dass das durchschnittliche Energieniveau jetzt als Schwellwert genutzt werden kann. Immer wenn der Energiewert eines Segments größer ist als das durchschnittliche Energieniveau um dieses Segment herum (rot dargestellt), dann handelt es sich um einen “Energie-Schub”, der ab nun als Beat identifiziert werden soll.

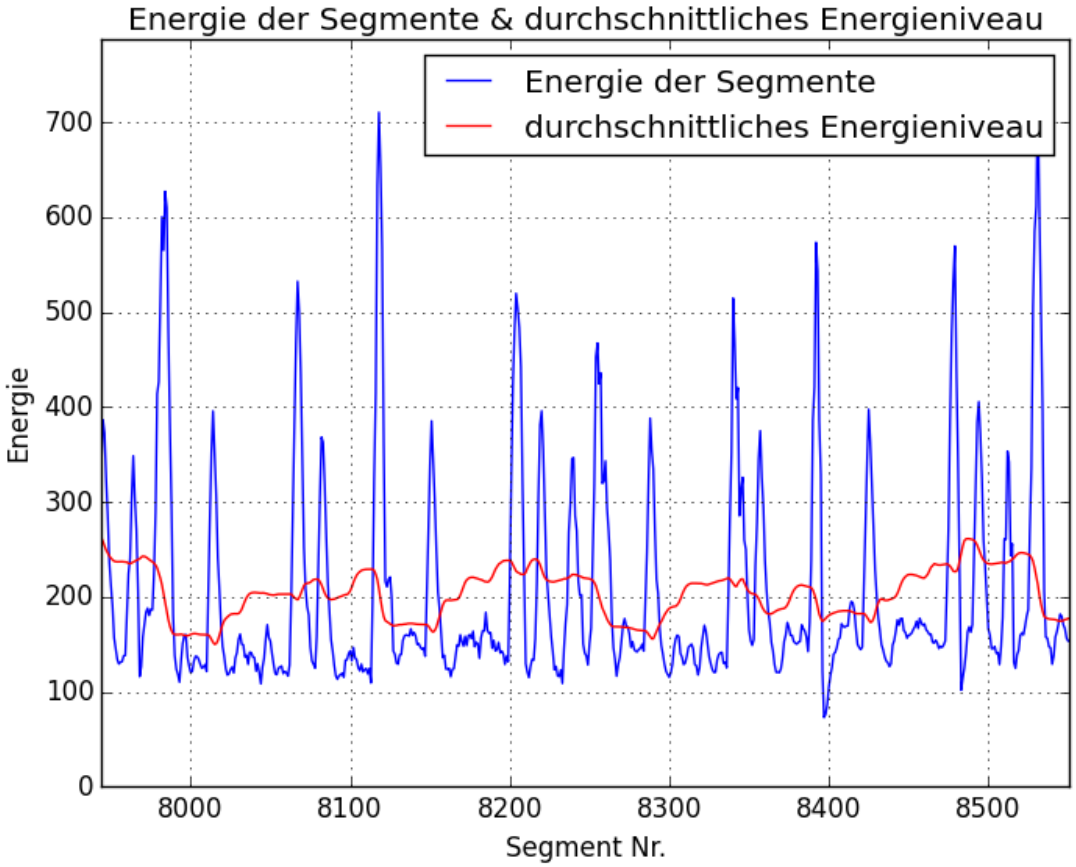

Abbildung 9: Ausschnitt aus den Energiewerten der Segmente und durchschnittliches Energieniveau um jedes Segment (aus “Memory Lane” von Netsky)

4.1.3 Identifizierung der Beats durch den “Onset”

In Abbildung 10 ist nun gezeigt, welche Segmente durch den Algorithmus als Beats markiert werden. Da immer ein ganzes Intervall an Segmenten über dem Energieniveau markiert wird, aber nur ein bestimmtes Segment als Zeitpunkt für ein Beat-Ereignis (Beginn einer neuen Note) betrachtet werden soll, wird der erste, am linken Rand befindliche Wert zu diesem Zeitpunkt bestimmt. Dies soll dem Zeitpunkt des “onset”-Werts, beschrieben in Abschnitt 1.3.2, möglichst Nahe kommen.

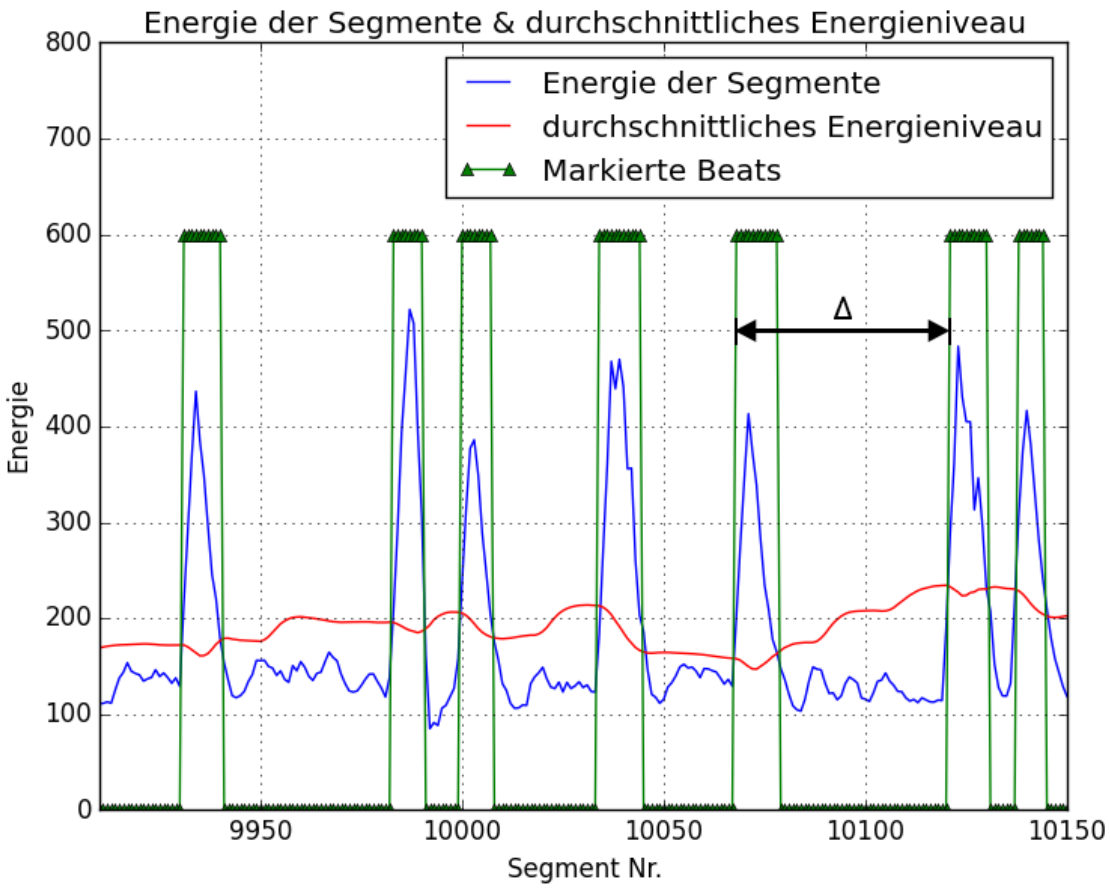

Abbildung 10: Markierte Segmente, die als Beats identifiziert sind (aus “Memory Lane” von Netsky)

4.2 Von den markierten Beats zu einem BPM-Wert

Von allen gefundenen Beats wird die Differenz (in Segmenten) zum nächsten Beat ermittelt, wie in Abbildung 10 durch das “Δ” gezeigt. Anschließend wird gezählt, wie oft jedes Intervall im gesamten Audiosignal gefunden wurde. Beats, die weniger als 5 Segmente auseinanderliegen, werden ignoriert. Es ergibt sich damit eine Häufigkeitsverteilung, die in Abbildung 11 gezeigt ist. Dort ist zu sehen, dass das Intervall “17 Segmente” und das Intervall “34 Segmente” in dem Beispielsong besonders häufig gefunden worden sind. Der Algorithmus sucht nun das Maximum in der Häufigkeitsverteilung, welches in diesem Fall bei “17 Segmenten” liegt.

Diese 17 Segmente können nun wieder in eine Anzahl an Samples überführt werden, da ein Segment auf eine Größe von 441 Samples definiert wurde.

$$N = Segmente \cdot 441 = 17 \cdot 441 = 7497 \tag{4}$$

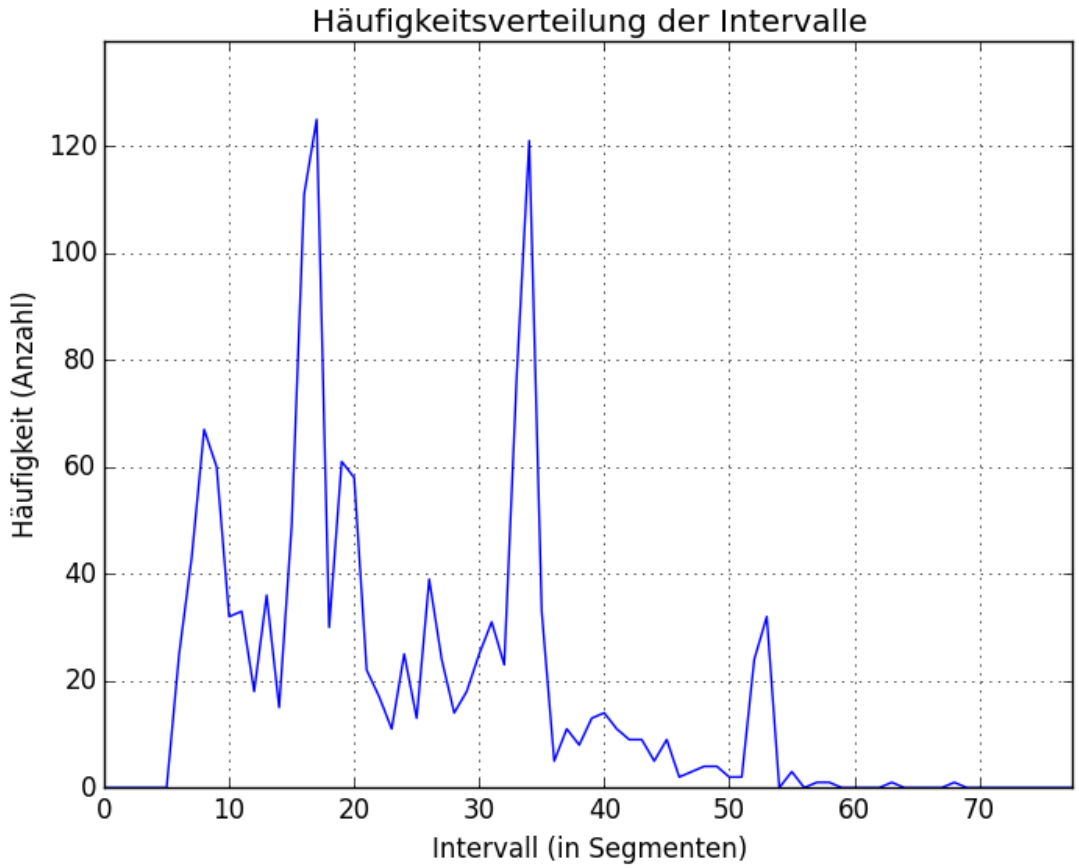

Abbildung 11: Häufigkeitsverteilung der Intervalle zwischen den identifizierten Beats (aus "Memory Lane" von Netsky)

Aus den 7497 Samples und der Abtastfrequenz von 44100 Hz kann nun ein zeitliches Intervall in Sekunden berechnet werden:

$$\Delta t = \frac{N}{44100\ Hz} = \frac{7497}{44100\ Hz} = 0{,}17\ s \tag{5}$$

Aus der zeitlichen Differenz kann schließlich der BPM-Wert berechnet werden:

$$BPM = \frac{60\ s}{\Delta t} = \frac{60\ s}{0{,}17\ s} \approx 352{,}94 \tag{6}$$

Um den BPM-Wert jetzt in das richtige Intervall zu skalieren, wird es solange durch zwei geteilt, bis gilt $60 \leq BPM \leq 200$. Außerdem wird der Wert auf eine ganze Zahl gerundet, da ganze BPM-Werte ermittelt werden sollen. (Abschnitt 1.3.1)

$$BPM = \frac{352{,}94}{2} \approx 176$$

Für den untersuchten Song “Memory Lane” von Netsky hat der Algorithmus ein Ergebnis von 176 BPM bestimmt. Zum Vergleich: Laut Beatport soll der Song ein Tempo von 175 BPM haben. (vgl. https://pro.beatport.com/track/memory-lane-original-mix/1623633, 23.12.2015)

4.3 Nachteile und Probleme des Algorithmus

4.3.1 Zeitliche Unschärfe

Ein offensichtliches Problem an diesem Algorithmus ist sichtbar, wenn man die Formeln (4), (5) und (6) zusammenfasst.

$$BPM = \frac{60\ s\ \cdot\ 44100\ Hz}{Segmente\ \cdot\ 441} = \frac{26{,}46 \cdot 10^3}{Segmente\ \cdot\ 441} \tag{7}$$

Für 40 Segmente ergibt sich damit:

$$BPM = \frac{60\ s\ \cdot\ 44100\ Hz}{Segmente\ \cdot\ 441} = \frac{26{,}46 \cdot 10^3}{40\ \cdot\ 441} = 150$$

Aber wenn das Intervall um nur ein Segment größer wird, ergibt sich folgender BPM-Wert:

$$BPM = \frac{60\ s\ \cdot\ 44100\ Hz}{Segmente\ \cdot\ 441} = \frac{26{,}46 \cdot 10^3}{41\ \cdot\ 441} \approx 146$$

Dieser Wert weicht schon relativ stark von den 150 BPM ab, obwohl sich das Intervall nur um ein Segment vergrößert hat. Es gibt also eine zeitliche Unschärfe, da zusammengefasste Sample-Pakete als Segmente betrachtet werden.

4.3.2 Starke Signalabhängigkeit

Ein weiteres Problem dieses Algorithmus ist die starke Abhängigkeit von den inhaltlichen Eigenschaften des Audiosignals. Für sehr klare, präzise Schläge von einem Schlagzeug, wie es in elektronischer Musik oder HipHop häufig der Fall ist, funktioniert das “beat tracking” relativ gut. Aber beispielsweise bei Rocksongs, in denen sehr geräuschhafte E-Gitarren vorkommen, kann der Energieansatz nicht immer die richtigen Beats erkennen.

5. Algorithmus 2: Peaks im tiefen Spektralbereich

Dieser Algorithmus ist in der JAVA-Klasse “LowpassPeakAlgorithm” (LowpassPeakAlgorithm.java) umgesetzt.

5.1 “beat tracking”: BPM-Ermittlung durch Intervall-Analyse

Grundannahme dieses Algorithmus ist das regelmäßige, wiederholte Auftreten gleichartiger Intervalle zwischen besonders hohen Ausschlägen (Peaks) im tiefen Spektralbereich des Audiosignal, also ein rhythmischer, bassbetonter Signalverlauf. Die Einschränkung auf Peaks in den tiefen Frequenzen begründet sich auf der Zusammensetzung zeitgenössischer Pop- und Tanzmusik und setzt voraus, dass die Rhythmik des zu analysierenden Musikstücks maßgeblich durch Frequenzen im tiefen Spektralbereich beinflusst wird. Die Erkennung der Peaks und deren Position im Signal, die Ermittlung der Abstände zwischen den Peaks und die daraus gezogenen Rückschlüsse auf das Tempo werden im Folgenden erläutert.

5.1.1 Tiefpassfilterung

Da hohe Signalausschläge ihren Ursprung auch im mittleren bis hohen Frequenzbereich haben können, wird das Signal im ersten Schritt des Algorithmus durch einen Bessel-Tiefpassfilter 4. Ordnung gefiltert. Die Grenzfrequenz des Filters ist auf 100 Hz eingestellt, um irreführende Ausschläge aus höheren Frequenzbereichen zu eliminieren. Implementiert wurde der Filter mithilfe der frei verfügbaren JAVA-Bibliothek “Java-DSP-collection”, (http://www.source-code.biz/dsp/java/). Durch die Filterung wird außerdem die Anzahl der zur Analyse in Frage kommenden Intervalle eingeschränkt. Die nachfolgenden Abbildungen 12 und 13 verdeutlichen dies.

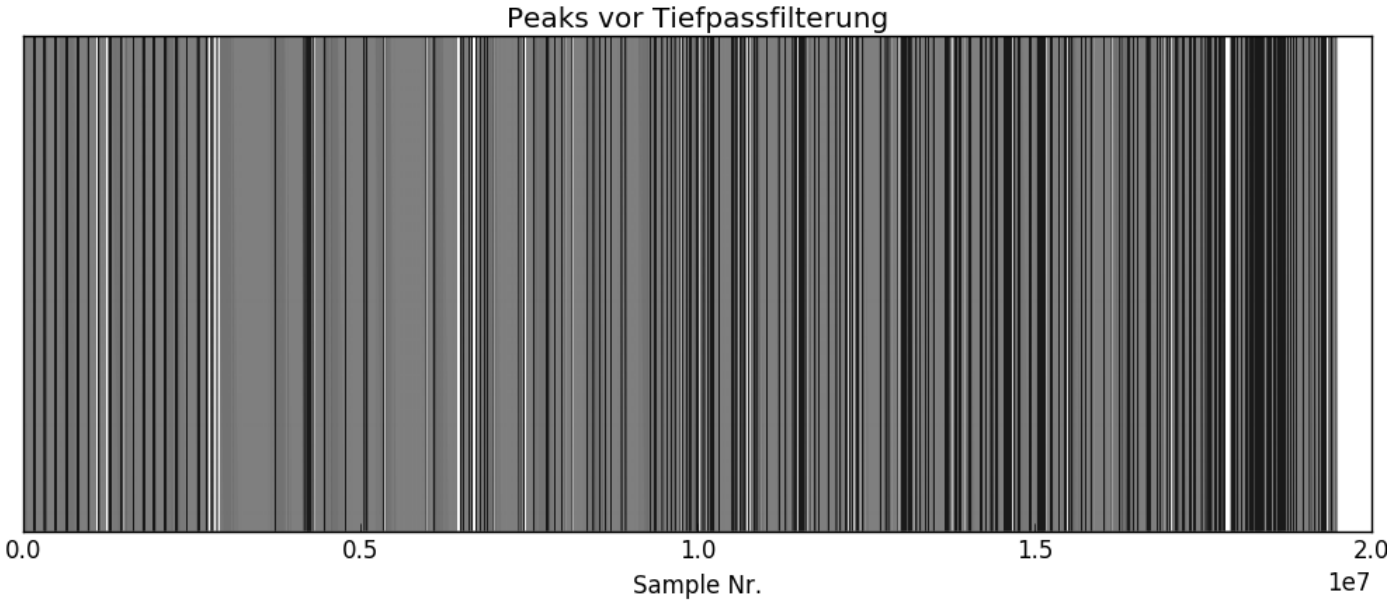

Abbildung 12: Position der erkannten Peaks vor der Tiefpassfilterung (Der komplette “Suga” von Technasia & Green Velvet). 1177 erkannte Peaks.

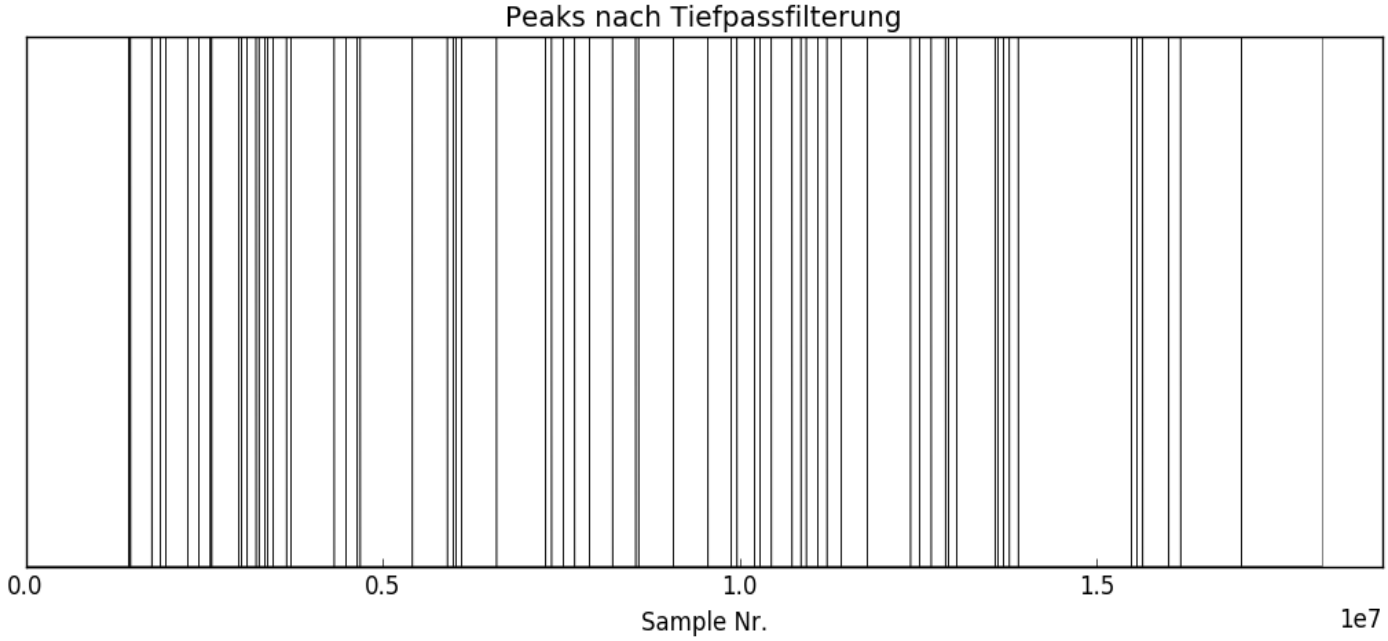

Abbildung 13: Position der erkannten Peaks nach der Tiefpassfilterung (Der komplette Track “Suga” von Technasia & Green Velvet). 72 erkannte Peaks.

5.1.2 Peak-Positions-Erkennung

Das zuvor gefilterte Signal wird nun auf Peaks untersucht. Um diese zu identifizieren, ist es nötig, einen Schwellwert festzulegen, ab dem ein Signalwert als Peak gilt. Da der Algorithmus auf der Analyse von Intervallen zwischen den Peaks beruht, wird eine Mindestanzahl an Ausschlägen benötigt. Aufgrund der Verschiedenheit von Musikstücken ist der Schwellwert dynamisch implementiert und passt sich, falls nötig, an die Gegebenheiten an. Die Funktion zur Erkennung der Peaks wird in einer Schleife solange ausgeführt, bis ein Minimum von 30 Peaks erkannt wurde.

Da die PCM-Samples der importierten WAV-Datei im Datentyp Double im Wertebereich von 0.0 bis 1.0 vorliegen, wird der erste Threshold-Wert mit 0.9 initialisiert und mit jedem Schleifendurchgang um 0.05 abgesenkt. Wird ein minimaler Schwellwert von 0.3 erreicht, bevor 30 Peaks erkannt wurden, versucht der Algorithmus das Tempo anhand der bisher erkannten Werte zu ermitteln. Die Positionen der ermittelten Werte im PCM-Array werden gespeichert und an die nächste Funktion weitergegeben.

5.1.3 Intervall-Analyse

Im Anschluss an die Positionserkennung werden die Abstände zwischen den einzelnen Peaks und deren zehn direkten Nachbarn gemessen. Jeder neue Abstand wird als Schlüssel in einer Hashmap gespeichert und bekommt einen Zähler als Wert zugewiesen. Tritt ein Intervall wiederholt auf, wird sein Zähler erhöht.

$$Intervall\ \Delta t = \frac{Peakosition\ [n]\ -\ Peakposition[n+1]}{44100\ Hz}$$

Abbildung 14 zeigt die sechs häufigsten Intervalle im tiefen Frequenzbereich des Songs “Blue Monday” von New Order. Sehr auffällig ist hier bereits, dass vier von sechs Zeitwerten Vielfache eines anderen Zeitwerts sind:

$$0{,}48 \cdot 2\ =\ 0{,}96 \cdot 2\ =\ 1{,}92 \cdot 2\ =\ 3{,}84$$

In Abschnitt 1.3.1 wurde erwähnt, dass ein BPM-Wert beliebig oft vervielfacht werden kann, ohne seine Aussagekraft über das Tempo des Musikstücks zu verlieren. Dies trifft auch auf die hier ermittelten Intervalle zu. Die Intervalle 0.48, 0.96, 1.92, 3.84 lassen sich also demselben Tempo zuordnen und können zusammengefasst werden.

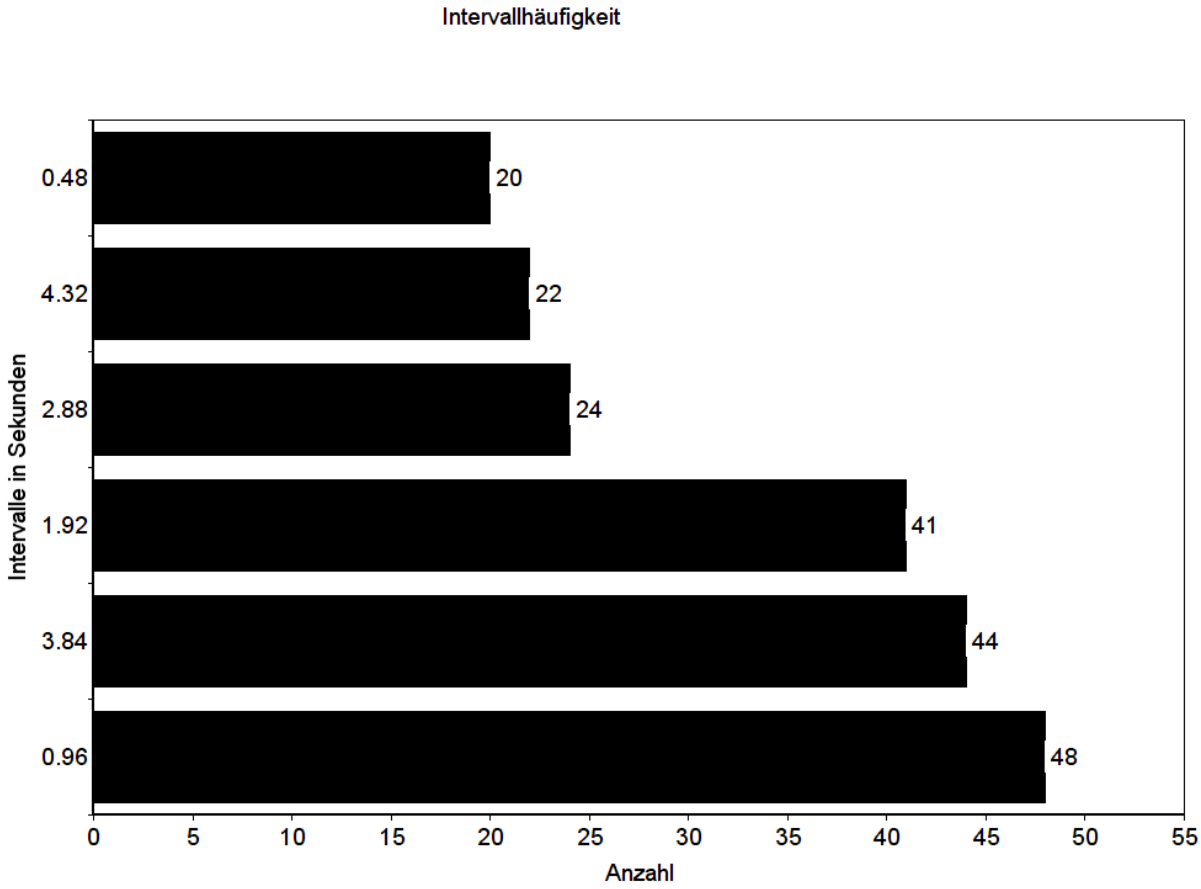

Abbildung 14: Die sechs am häufigsten auftretenden Intervalle in “Suga” von Technasia & Green Velvet

5.2 Ermittlung des BPM-Werts

Aus den berechneten Zeitwerten lassen sich schließlich die BPM-Werte berechnen:

$$BPM = \frac{60\ s}{\Delta t}$$

Wendet man diese Formel auf das am häufigsten auftretende Intervall aus Abbildung 13 an, erhält man:

$$BPM = \frac{60\ s}{\Delta t} = \frac{60\ s}{0{,}96\ s} = 62{,}5$$

Nun muss der BPM-Wert noch in das richtige BPM-Intervall skaliert werden. Der Algorithmus ist für das Intervall $100 \leq BPM \leq 200$ vorgesehen, deshalb wird der BPM-Wert solange mit 2 multipliziert bis diese Vorgabe zutrifft.

$$BPM = 62{,}5 \cdot 2 = 125$$

Im letzten Schritt werden alle Zeitintervalle mit dem obigen Verfahren in BPM-Werte umgerechnet und deren Vorkommen gezählt (siehe Abbildung 15). So kann gewährleistet werden, dass Vielfache, die für das gleiche Tempo stehen, auch diesem Tempo zugeordnet werden und das Ergebnis nicht nur vom Intervall mit dem höchsten Zählerwert bestimmt wird. Der am häufigsten auftretende BPM-Wert wird im Programm schließlich im mittleren Fenster angezeigt.

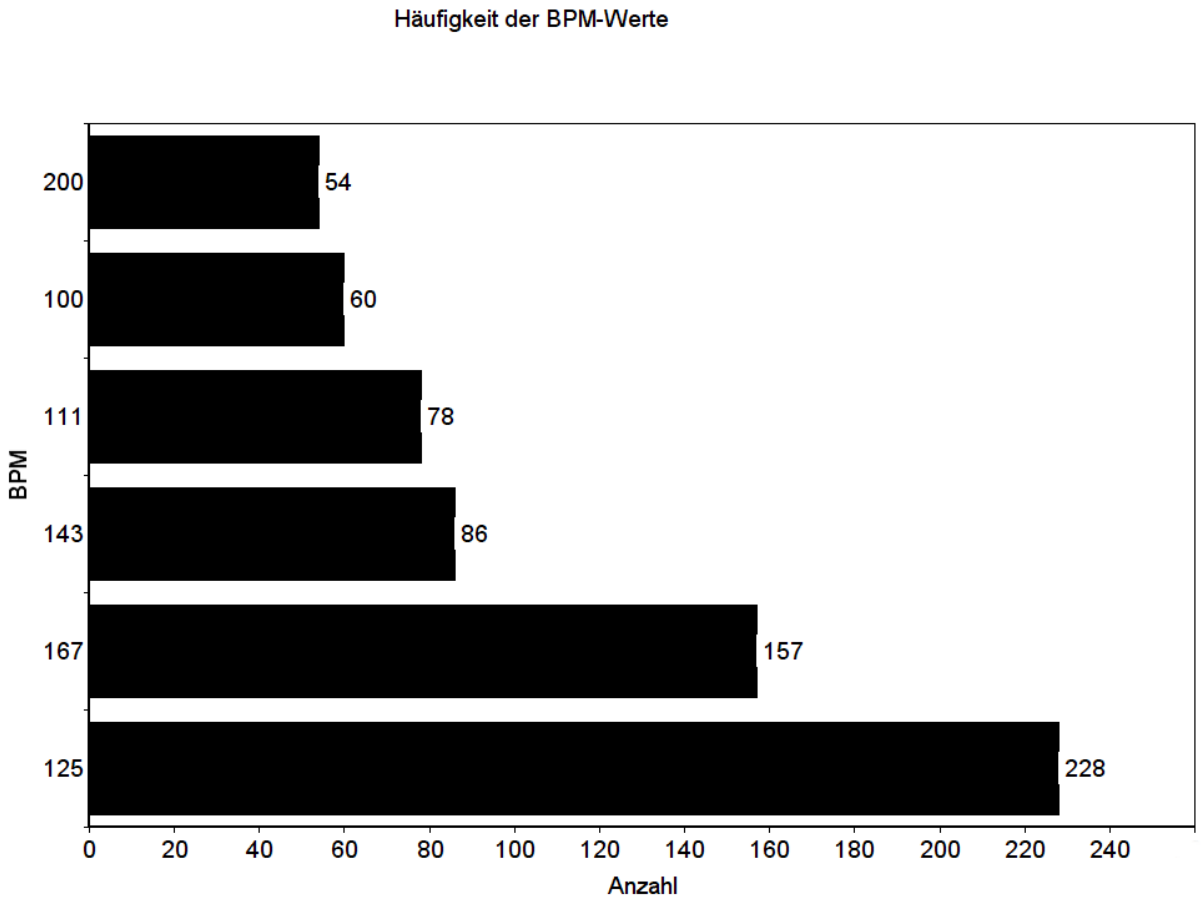

Abbildung 15: Die sechs am häufigsten ermittelten BPM-Werte für "Suga" von Technasia & Green Velvet

Der analysierte Song "Suga" von Technasia & Green Velvet hat laut der Musikplattform Beatport ein Tempo von 125 BPM, das Ergebnis des Algorithmus liegt also richtig (vgl. https://pro.beatport.com/track/suga-original-mix/6800712, 07.01.2016).

5.2 Nachteile und Probleme des Algorithmus

Der Algorithmus ist darauf angewiesen, dass das Audiosignal Informationen im Spektralbereich von 0 bis 100 Hz enthält, die im Bestfall das rhythmische Fundament des gesamten Musikstücks vorgeben. Sind diese Informationen nicht gegeben, befinden sich hinter dem Tiefpass keine verwertbaren Informationen für den Analysevorgang und seine Funktionen sind nutzlos. Er eignet sich also vor allem für moderne Tanzmusik.

6. Algorithmus 3: Wavelet-Dekomposition und Autokorrelation

Dieser Algorithmus ist in der JAVA-Klasse “WaveletAlgorithm” (WaveletAlgorithm.java) umgesetzt.

6.1 “beat tracking”: Erkennen der Beats durch “onset detection”

Bei diesem Algorithmus wird aus dem Audiosignal durch verschiedene Prozesse eine “Detection function” abgeleitet. Diese Funktion soll die “onsets” des Audiosignals (siehe Abschnitt 1.3.2) durch besonders hohe Werte herausstellen. Die wesentlichen Schritte eines solchen “onset detection”-Verfahrens in allgemeiner Form sind in Abbildung 16 dargestellt.

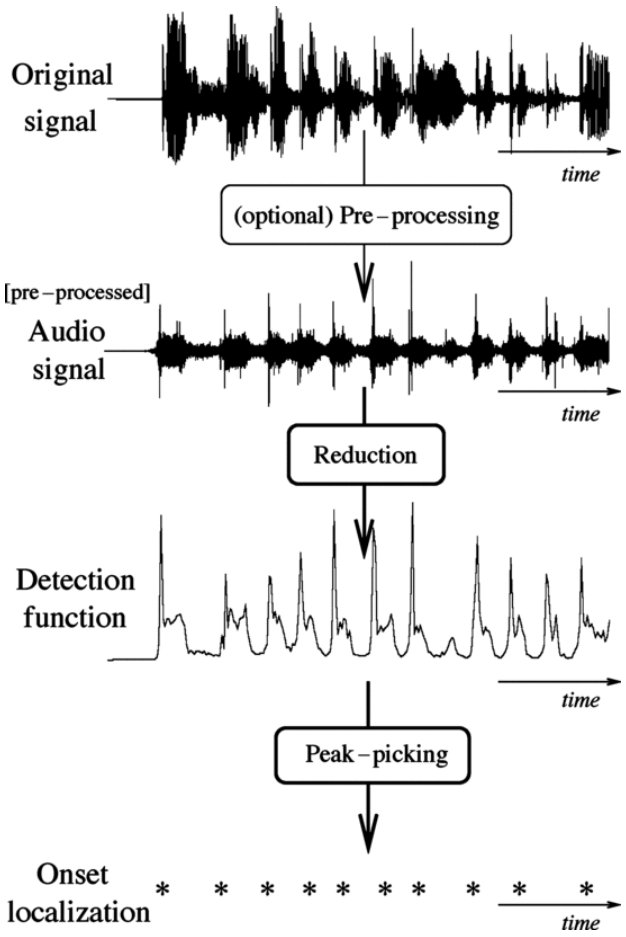

Abbildung 16: Wesentliche, schematisch dargestellte Schritte, um “onsets” aus einem Audiosignal zu gewinnen

Quelle: Bello, Juan Pablo et al., 2005, S. 1036, Fig. 2.

Für die dargestellten Schritte “Pre-Processing” und “Reduction” gibt es eine Vielzahl an Verfahren, die jeweils für unterschiedliche Signalbedingungen gut oder weniger gut geeignet sind. Im Schritt “Reduction” wird vor allem die Hüllkurve aus dem vorliegenden Signal entnommen.

Viele Verfahren haben allerdings gezeigt, dass es “[…] nützlich ist, die Informationen innerhalb verschiedener Frequenzbänder unabhängig voneinander zu untersuchen […]” (PABLO et al., 2005, S. 1036, Sp. 2, Z. 14ff.) und die Ergebnisse dann zusammenzuführen. SCHEIRER (1998, S. 591) hat dazu einen Algorithmus entwickelt, bei dem das Spektrum des Audiosignals in sechs Bänder eingeteilt wird, in denen jeweils die Hüllkurve extrahiert wird. Das Audiosignal wird somit in einzelne Frequenzbereiche dekomponiert. Die Filterbandbreiten sind so eingeteilt, dass sie jeweils eine Oktave im Frequenzbereich abdecken, man spricht dabei von einer Bandpassanalyse mit konstanter Güte (“constant-Q analysis”). Eine solche Auflösung erhält man auch aus der Zeit-Skalen-Analyse durch die Wavelet-Transformation, die sich damit gut anstelle der Filterbank nutzen lässt. Weitere Vorteile sind unter anderem, dass eine fehlerfreie Rücktransformation aus den dekomponierten Signalbestandteilen möglich ist und das Filterdesign der Filterbank gespart werden kann, da die diskrete Wavelet-Transforamtion dies bereits von sich aus mit sich bringt. (vgl. MERTINS, 2012, S. 302)

6.1.1 Wavelet-Transformation

6.1.1.1 Zeitkontinuierliche Wavelet-Transformation

Die zeitkontinuierliche Wavelet-Transformation eines Signals x(t) ist gegeben durch

$$W_x(b,a) = |a|^{-\frac{1}{2}} \int_{-\infty}^{\infty} x(t) \cdot \psi^*(\frac{t-b}{a})\, dt\,. \tag{8}$$

Die “[…] Wavelet-Transformierte wird als inneres Produkt des Signals x(t) mit verschobenen und skalierten Versionen einer einzigen Funktion $\psi(t)$ berechnet. […] Der Vorfaktor $|a|^{-1/2}$ sorgt dafür, dass alle Funktionen […] die gleiche Energie besitzen. Die Funktion $\psi(t)$ bezeichnet man als Wavelet.” (MERTINS, 2012, S. 301) Die skalierten und verschobenen Versionen des Wavelets sorgen dafür, dass sich bei einer hohen Frequenzen eine gute Zeitauflösung, aber eine schlechte Frequenzauflösung und bei tiefen Frequenzen eine schlechte Zeitauflösung, dafür eine gute Frequenzauflösung ergibt. Anders als die Fourier-Transformation ist die Wavelet-Transformation keine Zeit-Frequenz-Analyse, sondern eine Zeit-Skalen-Analyse. (vgl. MERTINS, 2012, S. 302)

6.1.1.2 Berechnung der DWT durch Multiraten-Filterung

Die Koeffizienten der Diskreten Wavelet-Transformation (DWT) lassen sich durch eine Wavelet-Reihenentwicklung berechnen.

In Abbildung 17 ist das Prinzip der Multiraten-Filterung dargestellt. Es werden Filtersysteme hintereinandergeschaltet, die zwei Koeffizientenfolgen ausgeben, einmal die Detail-Koeffizienten d_n und einmal die Approximations-Koeffizienten c_n. Den Schritt, bei dem eine Eingangsfolge (c_0) in zwei Ausgangskoeffizientenfolgen unterteilt wird, bezeichnet man als *einen* Dekompositionsschritt. Die Approximationskoeffizienten werden nun im nächsten Dekompositionsschritt durch das Filtersystem in zwei neue Koeffizientenfolgen c_2 und d_2 entwickelt. Dieser Vorgang führt sich hierarchisch fort (siehe Abbilung 18).

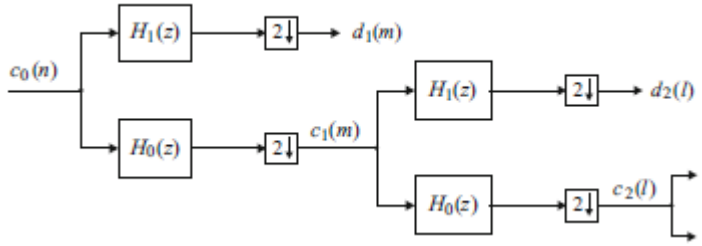

Abbildung 17: Prinzip der Filtersysteme zur Berechnung der DWT
Quelle: Mertins, Alfred: Signaltheorie, 2012, S. 320, Bild 9.11

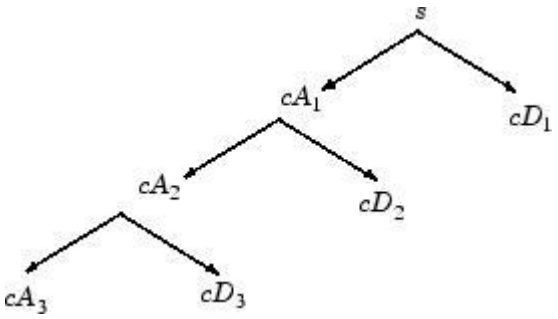

Abbildung 18: Hierarchische Wavelet-Reihenentwicklung (cA: Approximations-Koeffizienten, cD: Detail-Koeffizienten)
Quelle: http://de.mathworks.com/help/wavelet/ref/wavedec.html, 27.12.2015

Das Filtersystem besteht aus den Filtern H_1 und H_0, die zugehörigen Filterkoeffizienten leiten sich aus der Funktion des gewählten Wavelets ab. Der Filter H_0 kann als Tiefpass, während der Filter H_1 als Hochpass aufgefasst werden kann. Die Ausgangsfolgen sind also gefilterte Signale im Zeitbereich. Angedeutet durch "2 ↓" werden diese gefilterten Koeffizienten dezimiert, sodass jeweils nur noch die Hälfte der Anzahl an Werten vorhanden sind. Das ist ohne Verlust möglich, da durch die Filterung die Bandbreite des Signals halbiert wird, einmal für hohe Frequenzen und einmal für tiefe Frequenzen. Wie oft Approximations-Koeffizienten erneut zerlegt werden können, also wie viele Dekompositionsschritte insgesamt möglich sind, hängt von der Anzahl der Werte des Eingangssignals ab, da der Vorgang so oft wiederholt werden kann, bis die letzten Koeffizientenfolgen nur noch zwei Werte beinhalten. Auch muss die Anfangslänge des Eingangssignals einer Potenz von zwei entsprechen. Die Stufen der Reihenentwicklung werden dabei "level" genannt. Beispielsweise ist in Abbilung 18 eine Reihenentwicklung vom "level" 3 dargestellt.

Angenommen die Wavelet-Dekomposition ist in Abbildung 18 bis in das letzte "level" abgeschlossen, dann liegen nach der Berechnung die endgültigen Koeffizientenfolgen cD_1, cD_2, cD_3 und cA_3 vor. Addiert man die Anzahl der Werte aller dieser genannten Koeffizientenfolgen zusammen, erhält man die Anzahl der Werte des Eingangssignals. Entsprechend der Abbildung 19 werden diese Koeffizienten in der DWT zu einem Ergebnissignal C aneinandergereiht. Dieses Signal ist als Array in vielen Implementierungen der DWT als Ergebnis berechenbar.

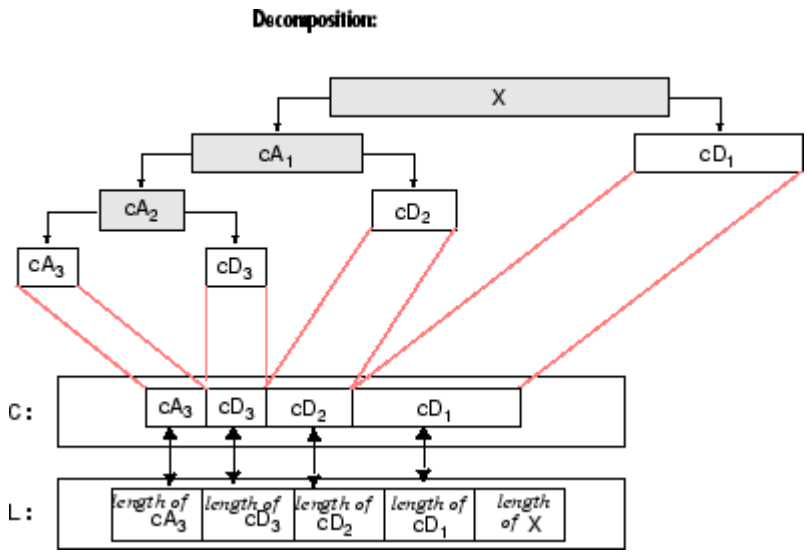

Abbildung 19: "X" ist das Eingangssignal, cD_n die Detail-Koeffizienten, cA_n die Approximations-Koeffizienten. "C" ist das Ausgangssignal (Ergebnis) des DWT-Algorithmus

Quelle: http://de.mathworks.com/help/wavelet/ref/wavedec.html, 27.12.2015

6.1.1.3 Verwendung der Bibliothek “JWave” zur Berechnung der DWT

Es wird die Implementierung eines Fast-Wavelet-Algorithmus zur Berechnung der DWT der der frei verfügbaren JAVA-Bibliothek “JWave”, programmiert von Christian Scheiblich, genutzt. (https://github.com/cscheiblich/JWave)

Die Bibliothek verfügt über eine Methode “decompose(…)”, die ein Eingangssignal in alle möglichen “level” der DWT dekomponiert. Als Ergebnis gibt die Methode ein zweidimensionales Array zurück, in der ersten Dimension kann die Tiefe, also das “level” gewählt werden, in der zweiten Dimension sind die Koeffizienten gemäß der Darstellung in Abbildung 19 angelegt.

6.1.2 Umsetzung eines “onset-detection”-Algorithmus mittels DWT

Anstelle einer Filterbank, wie SCHEIRER (1998, S. 591) sie nutzt, wird das Audiosignal also durch die Verwendung der Wavelet-Transformation dekomponiert.

Einen Algorithmus zur Tempo-Gewinnung mittels Wavelet-Transformation hat TZANETAKIS (et al.) entwickelt. Er wählte als Wavelet das “Daubechies 4”, das für den im Folgenden vorgestellten Algorithmus ebenfalls verwendet werden soll. Ebenfalls angelehnt an diesen Algorithmus werden die vier Prozess-Schritte “LPF (low pass filtering), FWR (full wave rectification), Downsampling und Normalization” in angepasster Reihenfolge verwendet.

Sie dienen zur Gewinnung der “onset”-Informationen in den einzelnen dekomponierten Signalanteilen des Audiosignals, wie sie in den Schritten “Pre Processing” und “Reduction” in Abbildung 16 dargestellt sind.

(vgl. TZANETAKIS et al., http://soundlab.cs.princeton.edu/publications/2001_amta_aadwt.pdf, S. 2)

Die Gewinnung der “Detection function” (Abbildung 16) geschieht nun in folgenden wesentlichen fünf Schritten (siehe Abbildung 20):

1. Zunächst wird das Signal in Fenster konstanter Länge aufgeteilt.
2. Für jedes Fenster werden die Samples mittels der DWT transformiert. Es entstehen für das “level” fünf somit sechs Koeffizientenfolgen.
3. Jede dieser Koeffizientenfolgen wird tiefpassgefiltert und anschließend unterabgetastet. Anschließend werden die Absolutwerte für alle Werte der Koeffizientenfolgen gebildet. (Hüllkurvenextraktion)
4. Es wird für jede jetzt reduzierte Koeffizientenfolge eine Normalisierung vorgenommen, in dem der Mittelwert der gesamten Folge von jedem Folgenwert subtrahiert wird.
5. Durch die Reduktion haben alle Koeffizientenfolgen die gleiche Länge und können jetzt durch Summenbildung zusammengefasst werden.

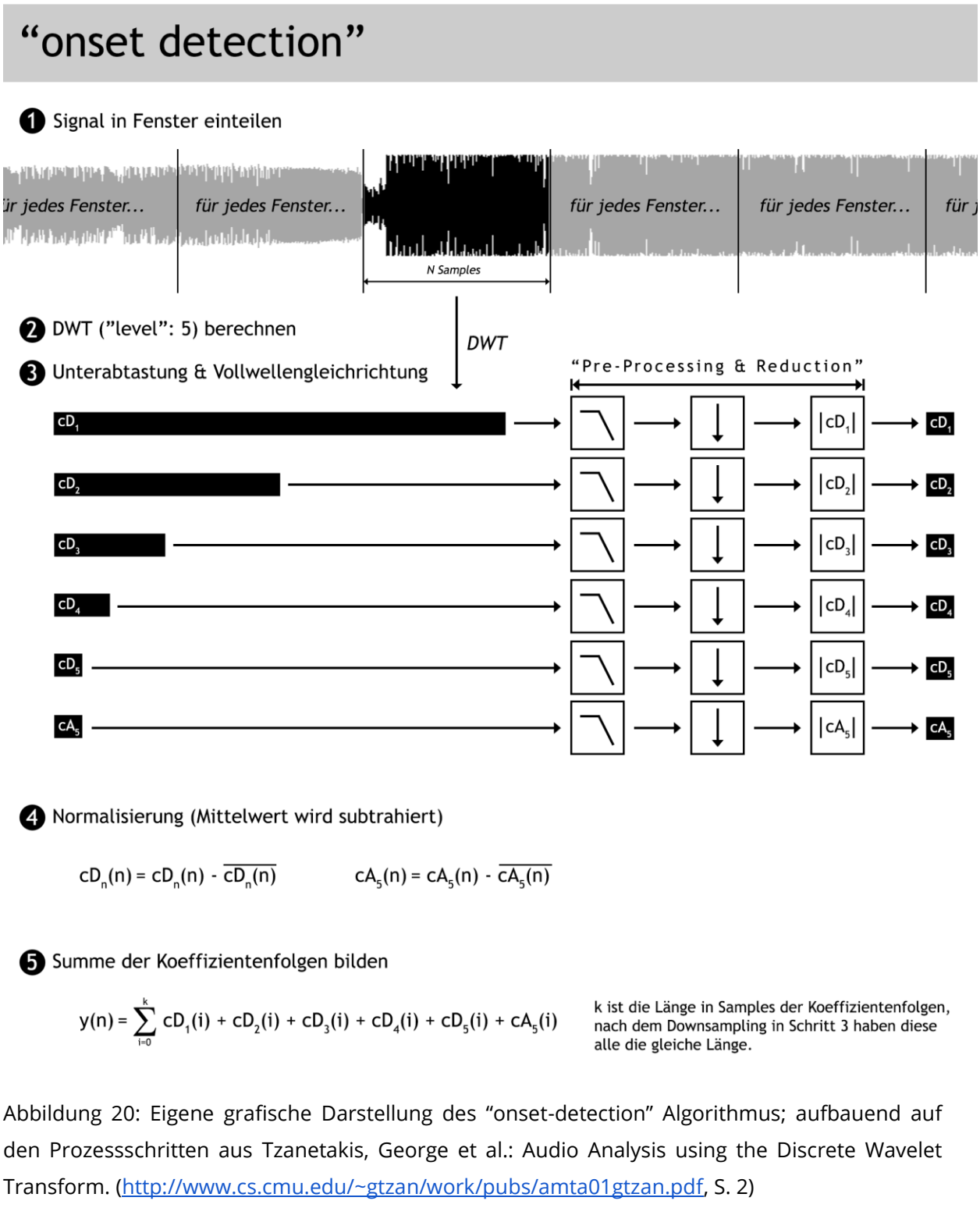

Abbildung 20: Eigene grafische Darstellung des “onset-detection” Algorithmus; aufbauend auf den Prozessschritten aus Tzanetakis, George et al.: Audio Analysis using the Discrete Wavelet Transform. (http://www.cs.cmu.edu/~gtzan/work/pubs/amta01gtzan.pdf, S. 2)

6.1.2.1 Einteilung des Audiosignals in Fenster

Zunächst wird das gesamte Audiosignal in einzelne Abschnitte (Fenster) eingeteilt, um abschnittsweise die DWT berechnen zu können. Wie in Abschnitt 6.1.1.2 beschrieben, muss die Anzahl der Samples in einem Fenster einer Potenz von zwei entsprechen. Es wird ein Fenster von etwa drei Sekunden angestrebt, passend dazu die Anzahl an Samples von

$$2^{17} = 131072.$$

Das langsamste Tempo, welches erkannt werden soll, beträgt 60 BPM. In einem Fenster von etwa drei Sekunden sind mindestens zwei Schläge dieses Tempos enthalten. Damit ist sichergestellt, dass für jedes Fenster ein BPM-Wert bestimmt werden kann.

6.1.2.2 Berechnung der DWT für jedes Fenster

Für jedes dieser Fenster wird nun zunächst die DWT mittels JWave für alle “level” berechnet. Von den 17 möglichen “level” wird das fünfte “level” ausgewählt. Durch Zerlegung der Ergebnisarray der Dekomposition nach dem Muster in Abbildung 19 können somit die Koeffizientenfolgen cD_1, cD_2, cD_3, cD_4, cD_5 und cA_5 entnommen werden.

6.1.2.3 Tiefpassfilterung, Unterabtastung und Vollwellengleichrichtung

Im ersten Schritt wird jede Koeffizientenfolge durch ein digitales Tiefpassfilter erster Ordnung geschickt. Der Aufbau dieses Filters ist in Abbildung 21 gezeigt.

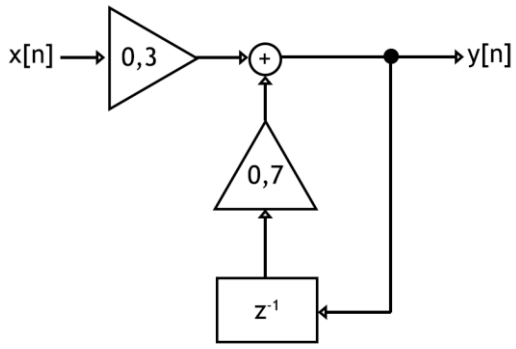

Abbildung 21: Einfaches Tiefpassfilter erster Ordnung

An den Tiefpassfilter schließt nun die Unterabtastung an. Diese Unterabtastung hat vor allem das Ziel, dass anschließend alle Koeffizientenfolgen dieselbe Anzahl an Werten tragen, und zwar die gleiche Anzahl an Werten wie die Approximations-Koeffizientenfolge. Da die Detail-Koeffizientenfolgen unterschiedliche Anzahlen an Werten haben, ist die Ordnung der Unterabtastung für jede Folge unterschiedlich. Beispielsweise muss die Folge cD_1 16-fach unterabgetastet werden, die Folge cD_2 nur noch 8-fach. In der folgenden Tabelle 1 sind alle Unterabtastungen aufgeführt.

Koeffizienten-folge	Länge [in Samples]	Unterabtastung	Länge nach Unterabtastung
cD_1	$N_1 = \frac{131072}{2^1} = 65536$	$d_f = 2^{5-1} = 2^4 = 16$ $cD_1(16n) \quad n \epsilon \mathbb{N}_0$	$N_{1,16\downarrow} = \frac{65536}{16} = 4096$
cD_2	$N_2 = \frac{131072}{2^2} = 32768$	$d_f = 2^{5-2} = 2^3 = 8$ $cD_2(8n) \quad n \epsilon \mathbb{N}_0$	$N_{2,8\downarrow} = \frac{32768}{8} = 4096$
cD_3	$N_3 = \frac{131072}{2^3} = 16384$	$d_f = 2^{5-3} = 2^2 = 4$ $cD_3(4n) \quad n \epsilon \mathbb{N}_0$	$N_{3,4\downarrow} = \frac{16384}{4} = 4096$
cD_4	$N_4 = \frac{131072}{2^4} = 8192$	$d_f = 2^{5-4} = 2^1 = 2$ $cD_4(2n) \quad n \epsilon \mathbb{N}_0$	$N_{4,2\downarrow} = \frac{65536}{16} = 4096$
cD_5	$N_5 = \frac{131072}{2^5} = 4096$	$d_f = 2^{5-5} = 2^0 = 1$ $cD_5(n) \quad n \epsilon \mathbb{N}_0$	$N_{5,1\downarrow} = N_5 = 4096$
cA_5	$N_5 = \frac{131072}{2^5} = 4096$	$d_f = 2^{5-5} = 2^0 = 1$ $cA_5(n) \quad n \epsilon \mathbb{N}_0$	$N_{5,1\downarrow} = N_5 = 4096$

Tabelle 1: Unterabtastung der Koeffizientenfolgen

Von jedem Wert wird im Schritt der Vollwellengleichrichtung nun der Absolutwert gebildet. Dieser Schritt ist vor allem dafür zuständig, dass die Hüllkurve der Koeffizientenfolgen abgebildet wird, in der die Informationen über “onsets” enthalten sind.

Die drei Schritte Tiefpassfilterung, Unterabtastung und Vollwellengleichrichtung sind in den nächsten drei Abbildungen für die Koeffizientenfolge cD_1 an einem Beispiel dargestellt. Das zugehörige Fenster entspricht in diesem Beispiel den ersten drei Sekunden aus dem Song “Dreams” von Jelle Slump.

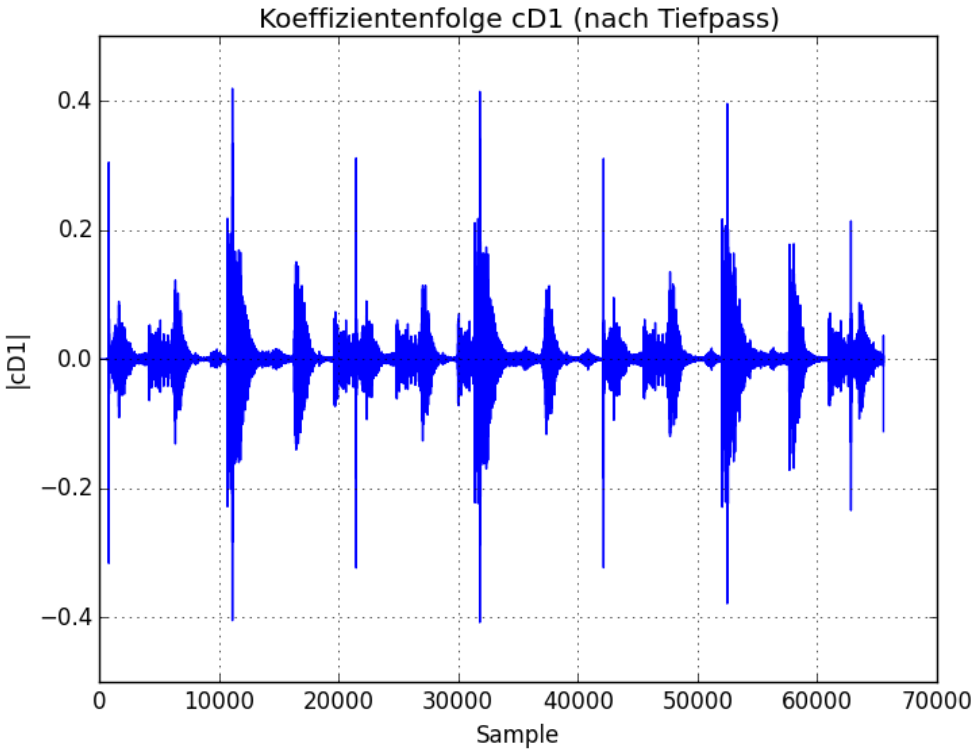

Abbildung 22: Tiefpassgefilterte Koeffizientenfolge cD_1 der ersten drei Sekunden aus “Dreams” von Jelle Slump.

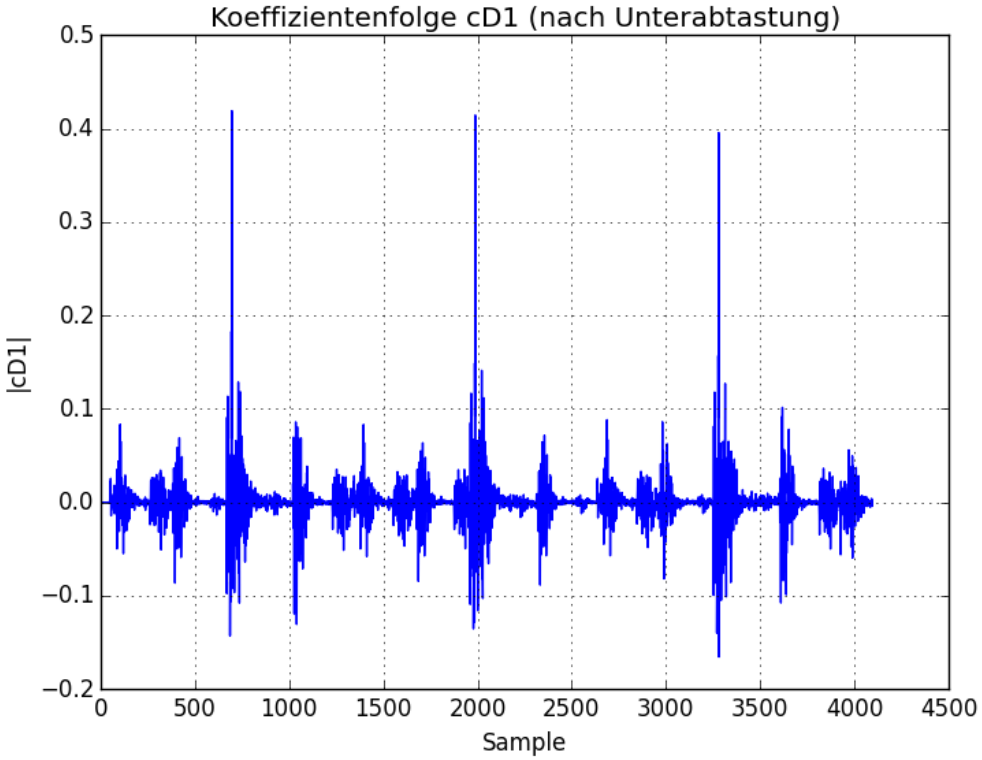

Abbildung 23: Koeffizientenfolge cD_1 nach Tiefpassfilterung und Unterabtastung der ersten drei Sekunden aus “Dreams” von Jelle Slump.

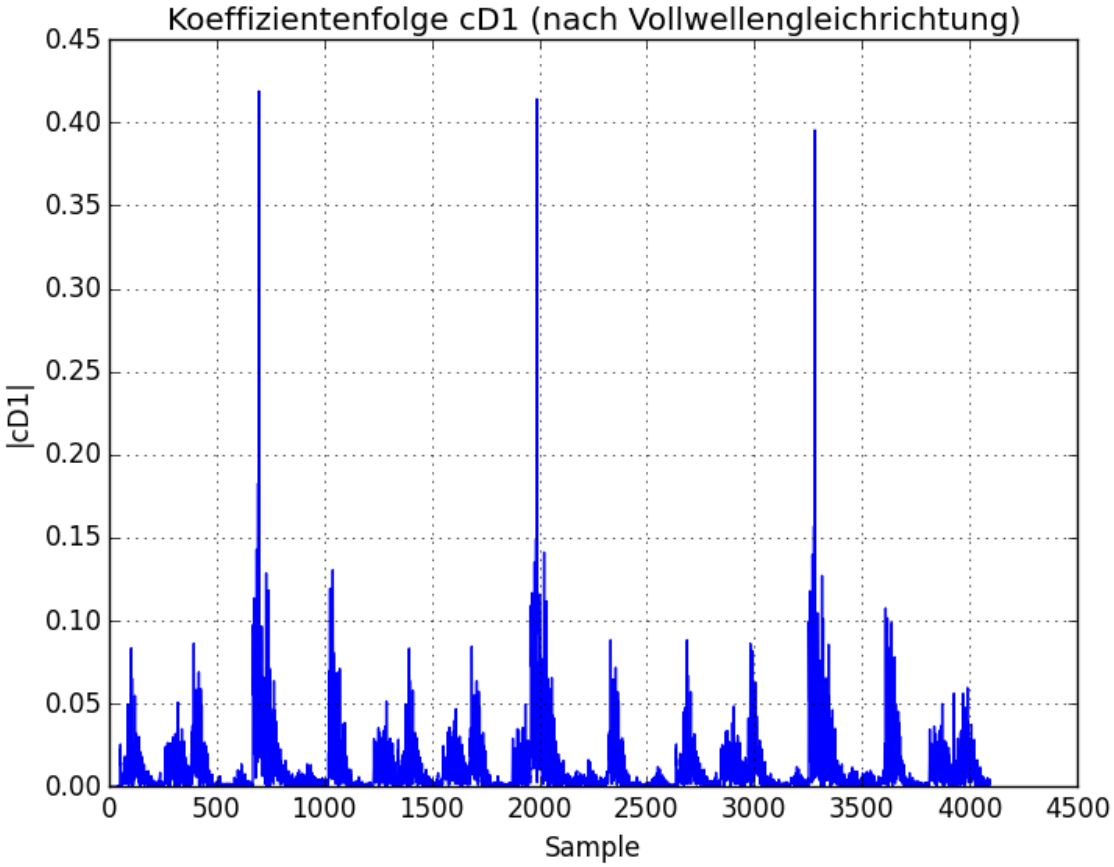

Abbildung 24: Koeffizientenfolge cD_1 nach Tiefpassfilterung, Unterabtastung und Vollwellengleichrichtung der ersten drei Sekunden aus "Dreams" von Jelle Slump.

6.1.2.4 Normalisierung

Als Vorbereitung für die Zusammenfassung der Koeffizientenfolgen dient eine Normalisierung, bei der von jedem Wert der Koeffizientenfolge der Mittelwert der Folge abgezogen wird.

$$cD_n(n) = cD_n(n) - \underline{cD_n(n)} \quad \text{bzw.} \quad cA_5(n) = cA_5(n) - \underline{cA_5(n)}$$

Grafisch bedeutet dies, dass die Koeffizientenfolge um den Mittelwert nach unten verschoben wird. Bei der anschließenden Zusammenfassung aller Folgen bilden sich somit bei den positiven Werten größere positive Werte, damit werden die "onsets" stärker ausgeprägt. Alle Werte unterhalb der x-Achse bilden größere negative Werte.

6.1.2.5 Zusammenfassung der Koeffizientenfolgen durch Summenbildung

Sind die letzten drei Schritte Tiefpassfilterung, Unterabtastung und Vollwellengleichrichtung für alle Koeffizientenfolgen eines Fensters aus dem Audiosignal durchgeführt worden, können diese nun nach folgender Formel zu einer "Detection function" y zusammengefasst werden:

$$y(n) = \sum_{i=0}^{k} cD_1(i) + cD_2(i) + cD_3(i) + cD_4(i) + cD_5(i) + cA_5(i).$$

k ist die Länge der Samples der Folgen.
(Alle Folgen haben durch die Unterabtastung die gleiche Länge)

Alle Werte < 0 dieser Funktion y werden nun zu 0 gesetzt. Für das erste Fenster aus dem Lied "Dreams" von Jelle Slump erhält man nun die in der folgenden Abbildung 25 dargestellte "Detection function". Dort sind die "onsets" des Audiosignals gut erkennbar, die Funktion ähnelt der Darstellung des Beispiels einer "Detection function" aus Abbildung 16.

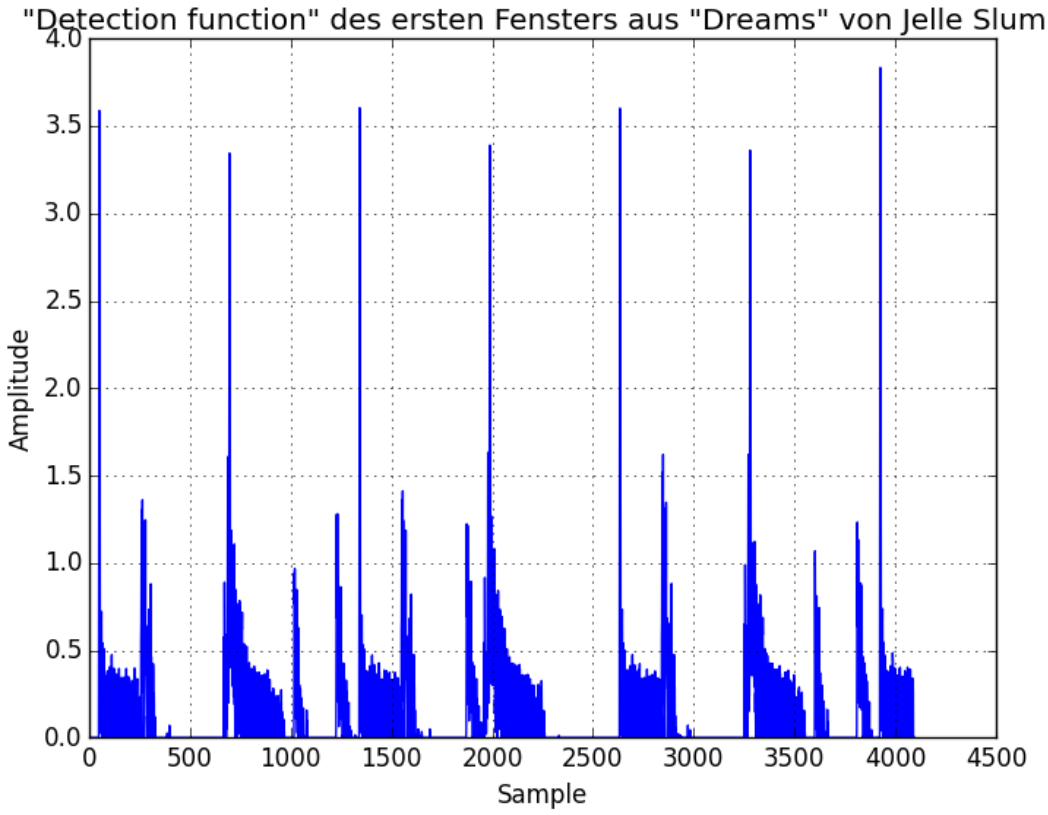

Abbildung 25: "Detection function" (onsets der ersten drei Sekunden aus "Dreams" von Jelle Slump).

6.2 Aus der “Detection function” die BPM ermitteln

Auch für die Ermittlung des Tempos als BPM-Wert aus der “Detection function” gibt es wieder einige unterschiedliche Möglichkeiten:

1) **Peaks finden** (vgl. PABLO et al., 2005, S. 1036)
   Wie in der Abbildung 25 gut zu erkennen ist, sind einige “onsets” durch besonders hohe Werte sehr ausgeprägt und gut durch einen einfachen Algorithmus ermittelbar, der nach diesen hohen Werten sucht. Anschließend kann über die zeitliche Differenz zwischen den “onsets” wieder auf ein Tempo geschlossen werden.
2) **Resonante Kammfilter** (vgl. SCHEIRER, 1998, S. 592)
3) **Autokorrelation**
   (vgl. TZANETAKIS et al., http://www.cs.cmu.edu/~gtzan/work/pubs/amta01gtzan.pdf, S. 3)

In diesem Algorithmus wird die Autokorrelation verwendet. Dabei wird die Autokorrelation als ein Maß für die Ähnlichkeit der “Detection function” mit sich selbst genutzt. Ein Peak in dieser Funktion zeigt, dass die “onsets” nach dem zugehörigen Sample-Wert als zeitliche Differenz wieder in sehr ähnlicher Form übereinstimmen. Da Rhythmen sich immer in sehr ähnlicher Form wiederholen (siehe Abschnitt 1.1), kann durch ein Maximum in der Autokorrelationsfunktion der Puls des Tempos bestimmt werden.

Die Autokorrelationsfunktion (AKF) ist definiert durch:

$$AKF\{x(n)\} = \sum_{n=0}^{N-1} x(n) \cdot x(n+k)$$

Aus Gründen der Performance wird die AKF allerdings mittels einer Multiplikation im Frequenzbereich berechnet.

Aus der Autokorrelation im diskreten Zeitbereich wird damit im Frequenzbereich:

$$AKF\{x(n)\} \circ\!\!-\!\!\bullet\; X \cdot \underline{X}$$

Wobei $X(n)$ die Fouriertransformierte von $x(n)$ ist und damit die einzelnen Werte komplexe Größen darstellen. Eine Multiplikation von einer komplexen Zahl mit dem konjugiert komplexen Paar entspricht:

$$z \cdot \underline{z} = (x + jy) \cdot (x - jy) = x^2 + y^2 + 0j$$

Alle Ergebnisse dieser Multiplikationen in $X \cdot \underline{X}$ werden anschließend invers fouriertransformiert. Man erhält damit wieder eine Funktion im Zeitbereich, die der AKF entspricht. Dies gilt allerdings nur, sofern $x(n)$ so viele Werte hat, wie das Ergebnis der linearen Faltung erwarten lässt. Die Eingangsfolge $x(n)$ müsste mit Nullen erweitert werden. (zero-padding) In diesem Fall könnte das zero-padding allerdings gespart werden, da nur ein bestimmter Bereich der AKF untersucht werden soll. Dieser Bereich ist auch aus dem Ergebnis der zyklischen Faltung entnehmbar.

Die Fouriertransformation und die inverse Fouriertransformation wird mit einer frei verfügbaren JAVA-Klasse, programmiert von Nayuki Minase, berechnet. (http://www.nayuki.io/page/free-small-fft-in-multiple-languages)

Die Autokorrelationsfunktion der “Detection function” aus den ersten drei Sekunden (erstes Fenster) aus “Dreams” von Jelle Slump ist in Abbildung 26 gezeigt.

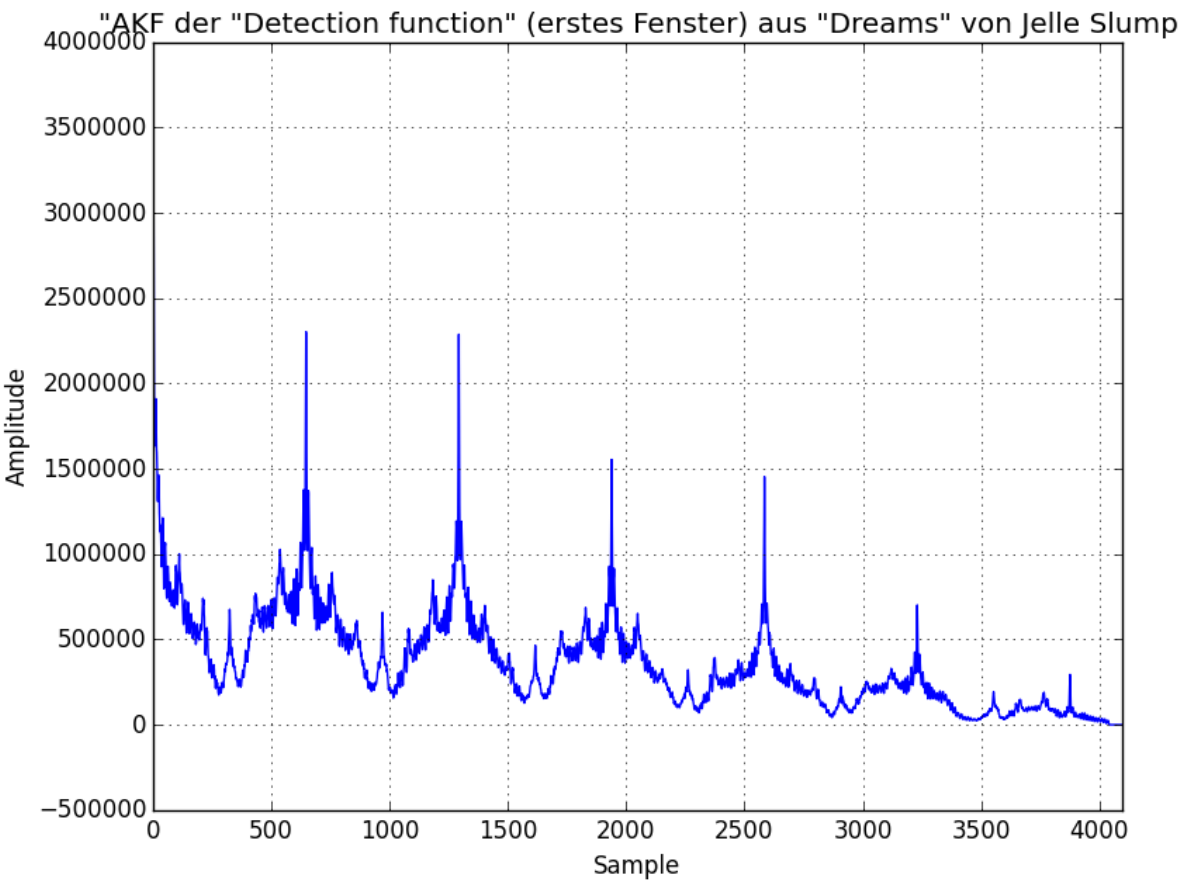

Abbildung 26: AKF der “Detection function” (der ersten drei Sekunden aus “Dreams” von Jelle Slump).

Der Algorithmus bestimmt aus der AKF nun den größten vorkommenden Wert in einem bestimmten Bereich. Da nur BPM-Werte zwischen 60 und 200 infrage kommen, können so Intervallgrenzen für die Suche nach dem Maximalwert berechnet werden:

$$min = \frac{60s}{200} \cdot \frac{44100\,Hz}{2^5} \approx 413$$

$$max = \frac{60s}{60} \cdot \frac{44100\,Hz}{2^5} \approx 1378$$

In dem Beispiel liegt der größte Wert in diesem Bereich bei 646. Die “Detection function” ist sich also in dem gewählten Bereich zu der um 646 Samples verschobenen “Detection function” am Ähnlichsten. Mit dieser Information kann nun ein BPM-Wert errechnet werden:

$$BPM = \frac{60s}{646} \cdot \frac{44100\ Hz}{2^5} \approx 128$$

Zum Vergleich: Laut Beatport hat der Song tatsächlich ein Tempo von 128 BPM. (vgl. https://pro.beatport.com/release/dreams/1673437, 23.12.2015)

Die BPM-Werte, die in dieser Form für jedes ca. drei Sekunden lange Fenster aus dem Audiosignal gewonnen werden, werden in ein Array gespeichert. Anschließend wird die Häufigkeit der BPM-Werte gezählt. Als Gesamtergebnis wird der BPM-Wert gewählt, dessen Vorkommen am Häufigsten ist.

7. Auswertung und Vergleich

7.1 Vorgehen

Im Konzept wurde ein Verfahren festgelegt, mit dem die Genauigkeit der BPM-Werte überprüft werden soll. Dabei soll durch die “Digital Audio Workstation” Ableton Live ein Drumloop in den verschiedenen Tempos von 60 BPM bis 200 BPM generiert werden. Der jeweils bei Ableton Live eingestellte BPM-Wert gilt dann als Ist-Wert und die Ergebnisse aus den Algorithmen des Programms haben das Ziel, diesen Ist-Wert mit einer Genauigkeit von 10% zu erreichen. Ein weiteres Kriterium ist die Rechenzeit, die für den Analysevorgang nicht größer sein darf als die doppelte Spieldauer der Audiodatei.

7.1.1 Relative Abweichung und Vielfache der BPM-Werte

In Abschnitt 1.3.1 wurde beschrieben, dass auch die doppelten oder halben Werte des BPM-Werts eines Musikstücks als gültige Ergebnisse für das Musiktempo gewertet werden sollen. Demnach sollen die BPM-Ergebnisse 71 BPM und 142 BPM den gleichen Fehler produzieren, wenn ein Musikstück mit dem Tempo 70 BPM untersucht wurde. Die relative Abweichung berechnet sich damit je nach Fall durch eine der folgenden Gleichungen:

$$f_{BPM} = \frac{Istwert \ - \ Sollwert}{Sollwert} \cdot 100\%$$

$$f_{BPM,2} = \frac{Istwert \ - \ 2 \cdot Sollwert}{2 \cdot Sollwert} \cdot 100\%$$

$$f_{BPM,3} = \frac{Istwert \ - \ \frac{1}{2} \cdot Sollwert}{\frac{1}{2} \cdot Sollwert} \cdot 100\%$$

Weitere Vielfache des BPM-Werts müssen nicht beachtet werden, da sich diese nicht im Intervall von 60 BPM bis 200 BPM darstellen lassen.

7.1.2 Das Testsignal

Für das Testsignal wurde ein simpler Drumloop in Ableton Live generiert, bei dem alle vier Zählzeiten mit der Bassdrum betont werden, auf den Zählzeiten zwei und vier kommt eine Snare dazu und auf jeder Achtelnote spielt eine geschlossene Hihat. In Abbildung 27 ist dieser Loop in notierter Form gezeigt.

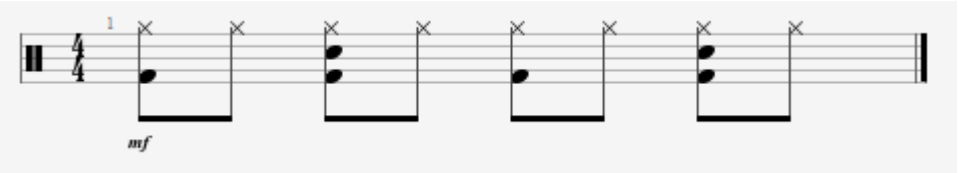

Abbildung 27: Notierter Drumloop für den Algorithmen-Vergleich

Dieser Loop wurde für jedes ganzzahlige Tempo zwischen 60 BPM und 200 BPM in eine Audiodatei im Format WAVE mit einer Minute Länge gerendert. Damit ergeben sich 141 Dateien, die nun mit jedem der drei Algorithmen untersucht werden sollen. Die Berechnungen wurden durchgeführt auf einem Computer mit einem i7 3770K Prozessor und 16 GB Arbeitsspeicher.

Diese Audiodateien sind auch in einem ZIP-Archiv auf der angehängten CD-ROM verfügbar.

7.2 Ergebnis

7.2.1 Genauigkeit der BPM-Werte

Der Algorithmus 1 (Soundenergie, Abschnitt 4) zeigt eine allgmeine Ungenauigkeit bei den BPM-Werten, bleibt aber bis auf eine Ausnahme bei 104 BPM immer unterhalb der geforderten relativen Genauigkeit < 10 %. Auffällig sind hier ungewöhnliche Abweichungen von 88-104 BPM. (vgl. Abbildung 28)

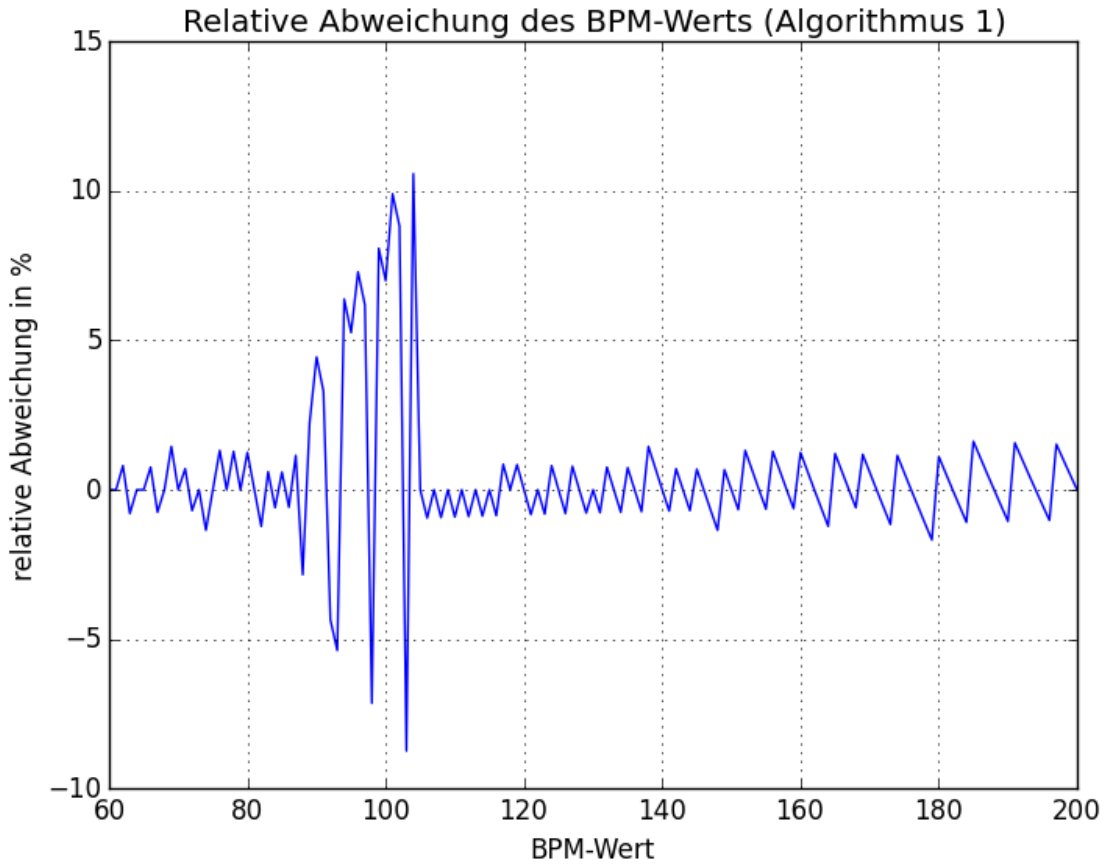

Abbildung 28: Relative Abweichung vom Ziel-BPM-Wert des Algorithmus 1 (Abschnitt 4)

Der zweite Algorithmus (Abschnitt 5) zeigt bis auf zwei Ausnahmen konstant exakte Ergebnisse. Die Ausnahmen befinden sich 100 BPM und 200 BPM. Ursachen dafür könnten Randartefakte sein, da 200 BPM die Obergrenze darstellt, bis zu welcher der Algorithmus arbeitet. (vgl. Abbildung 29)

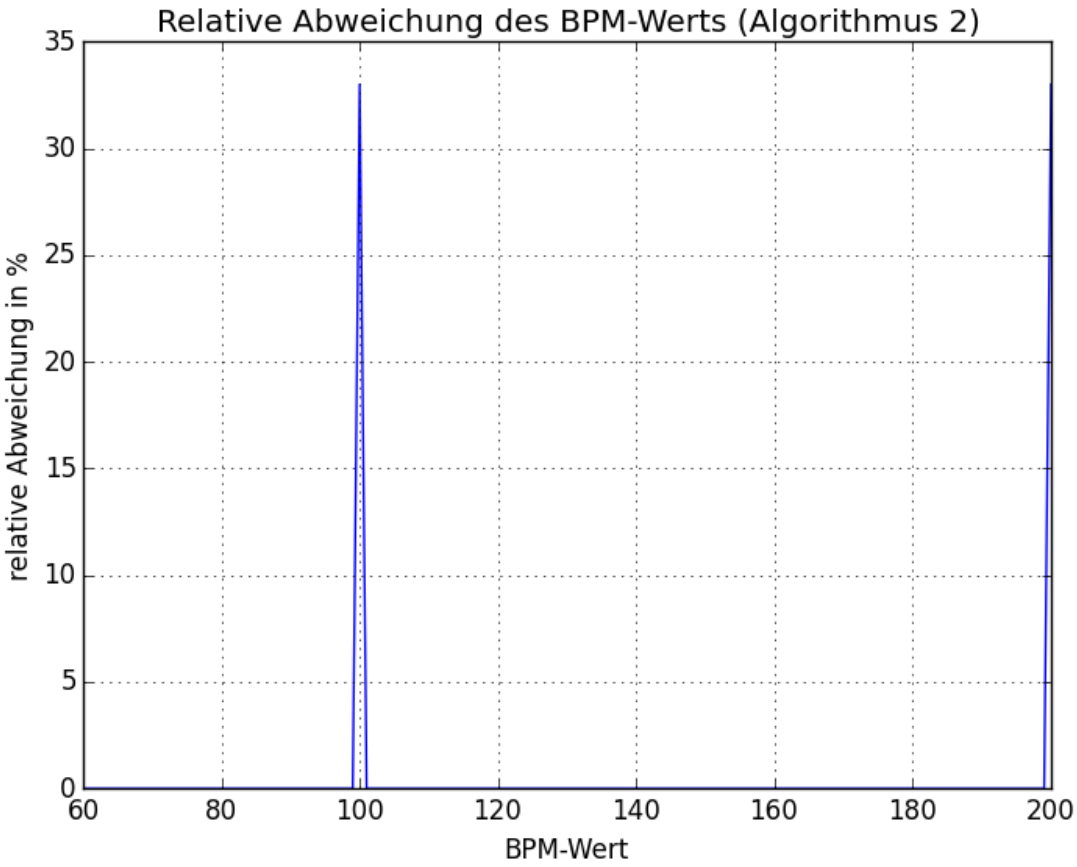

Abbildung 29: Relative Abweichung vom Ziel-BPM-Wert des Algorithmus 2 (Abschnitt 5)

Der dritte Algorithmus (Abschnitt 6) zeigt oberhalb von 100 BPM auch konstant exakte Ergebnisse. Unterhalb von 100 BPM gibt es aber vor allem einmal von 86-90 BPM und einmal von 72-75 BPM starke relative Abweichungen. (Abbildung 30)

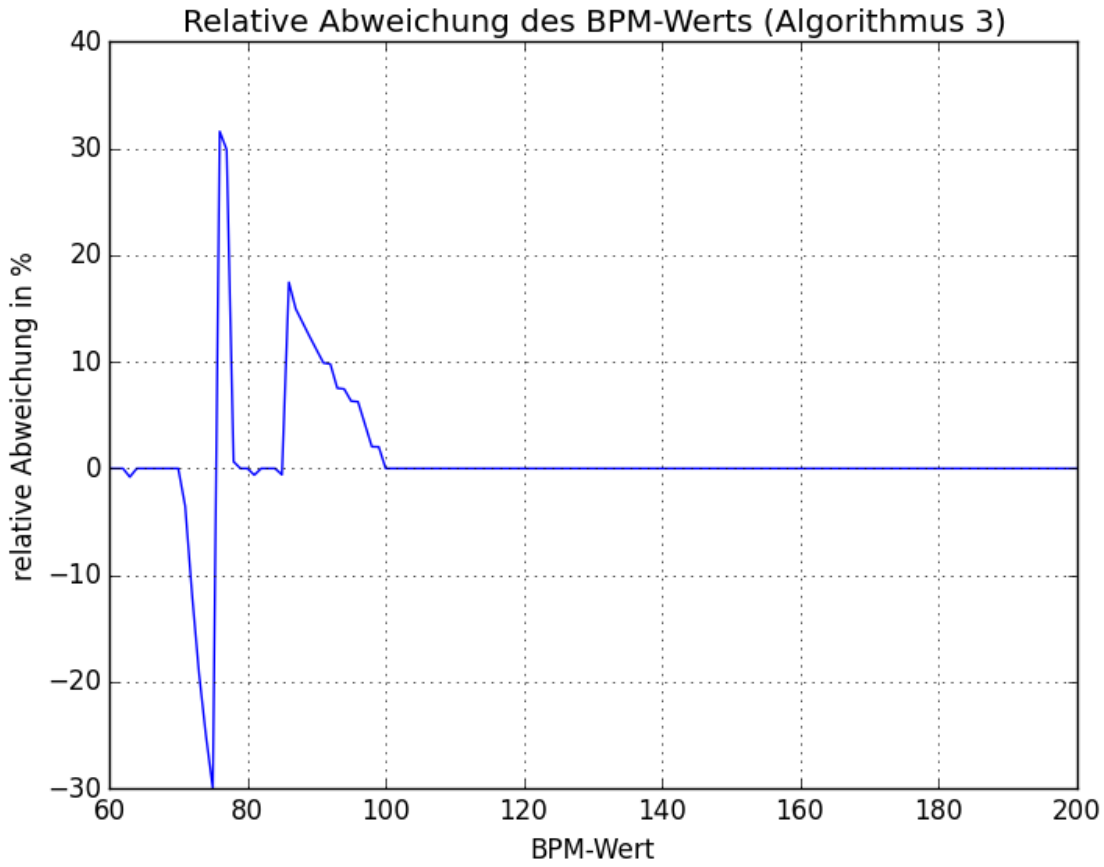

Abbildung 30: Relative Abweichung vom Ziel-BPM-Wert des Algorithmus 3 (Abschnitt 6)

7.2.2 Dauer des Analysevorgangs

Die Kriterien für die Beanspruchung der Rechendauer von maximal der doppelten Spieldauer der Audiodatei wurden von allen Algorithmen konsequent eingehalten.

Die Mittelwerte der Rechenzeiten lauten:

- Mittelwert der Rechenzeiten von Algorithmus 1: 52,12 ms
- Mittelwert der Rechenzeiten von Algorithmus 2: 170,41 ms
- Mittelwert der Rechenzeiten von Algorithmus 3: 1233,34 ms

Der einfache Algorithmus 1 ist sehr schnell, während Algorithmus 2 vermutlich vor allem durch die Filterung etwas mehr Zeit beansprucht. Die Wavelet-Transformation und die Autokorrelation in Algorithmus 3 für jedes Fenster sind wahrscheinlich die Ursache dafür, dass Algorithmus 3 im Verhältnis der Langsamste ist.

7.3 Nachtrag zum Vergleich

Genau genommen hat in diesem Testverfahren keiner der Algorithmen die Kriterien vollständig erfüllt. Trotzdem haben vor allem Algorithmus 2 und 3 in vielen Fällen erstaunlich exakte Ergebnisse ausgegeben. Aus diesem Grund sind wir mit dem Ergebnis doch sehr zufrieden.

Im Nachhinein betrachtet ist das Testverfahren mit den generierten Drumloops doch kein so passendes Verfahren wie im Konzept angenommen. Während der Algorithmus 1 ziemlich dankbar ist für ein Signal, in dem der Verlauf der Signalenergie vollständig genutzt werden kann, um das Tempo zu bestimmen, würden Algorithmus 2 und 3 ihren Trumpf eher bei komplexeren Rhythmen ausspielen, da sie das Audiosignal spektral aufteilen. Algorithmus 2 zum Beispiel würde durch den Tiefpass viele Informationen über die Rhythmen, die ein Bass im tieffrequenten Spektrum spielt, nutzen. Algorithmus 3 dekomponiert das Audiosignal in viele einzelne spektrale Abschnitte und sucht dort nach rhythmischen Strukturen. Damit ist anzunehmen, dass die Algorithmen 2 und 3 für “richtige” Audiosignale mit Musikinhalt besser gerüstet sind, während Algorithmus 1 durch viele gleichzeitige Rhythmen den Bezug zu einem einheitlichen Grundschlag eher verlieren würde. Die Wahl eines passenden Verfahrens, um die Algorithmen miteinander zu vergleichen ist also ein schwieriges Thema, da Musik so unterschiedlich gestaltet, gemischt und gemastert sein kann.

8. Das BPM-Finder Programm

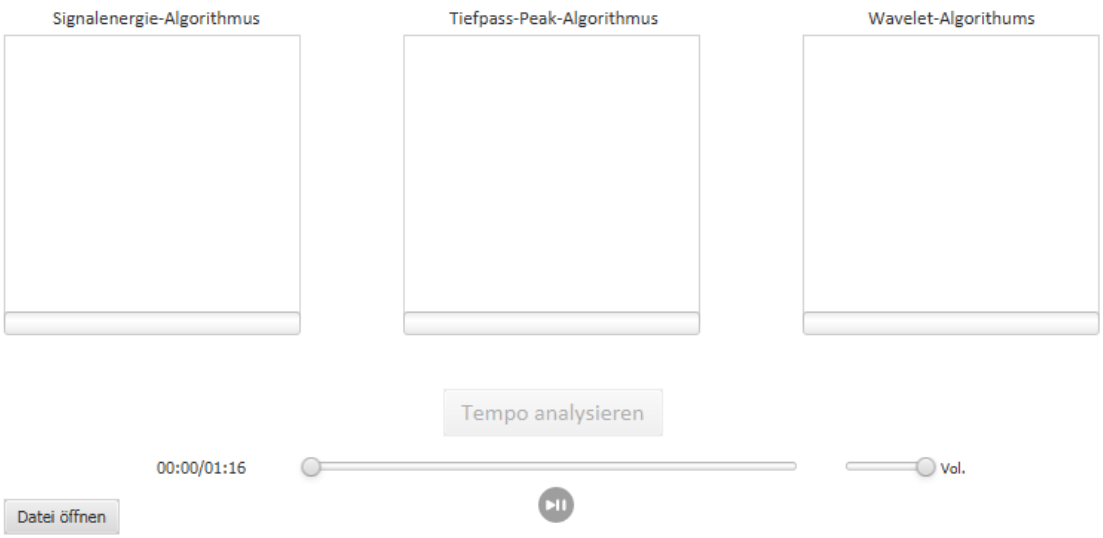

Abbildung 31: Die grafische Oberfläche des Programms “BPM-Finder” im Ausgangszustand

8.1 Aufbau

Der Aufbau des Programms soll eine intuitive Bedienung ermöglichen, weshalb sich die Bedienelemente auf das Nötigste beschränken und in einem schlichten Stil gehalten sind. Im Fokus des Programms stehen die drei Algorithmus-Ausgabefelder. Öffnet man über den Button “Datei öffnen” eine WAVE-Audiodatei, wird der Button “Tempo analysieren” aktiviert. Wird dieser betätigt, wird das Musikstück von den drei vorgestellten Algorithmen analysiert. Der Fortschritt der Berechnungen wird mit einer Fortschrittsanzeige in Form eines wachsenden grünen Balkens unterhalb der Ausgabefelder angezeigt. Schließt ein Algorithmus seine Berechnungen ab, übergibt er das ermittelte Tempo an sein Ausgabefeld und auf der Fortschrittsanzeige wird die Zeit in Millisekunden angezeigt, die der Algorithmus benötigte um das Musikstück zu analysieren (siehe Abbildung 32). An dieser Stelle sei darauf hingewiesen, dass die drei Algorithmen mittels Multi-Threading gestartet werden und so nebeneinander ablaufen. Das ermöglicht, dass die grafische Oberfläche während der Berechnung weiterhin bedienbar ist, geschieht jedoch auf Kosten der Performance der Algorithmen. Somit

sollen die Zeitwerte nur einen Vergleich der Algorithmen zum Zeitpunkt der Anwendung ermöglichen und stellen an dieser Stelle keine zuverlässigen Benchmark-Werte dar.

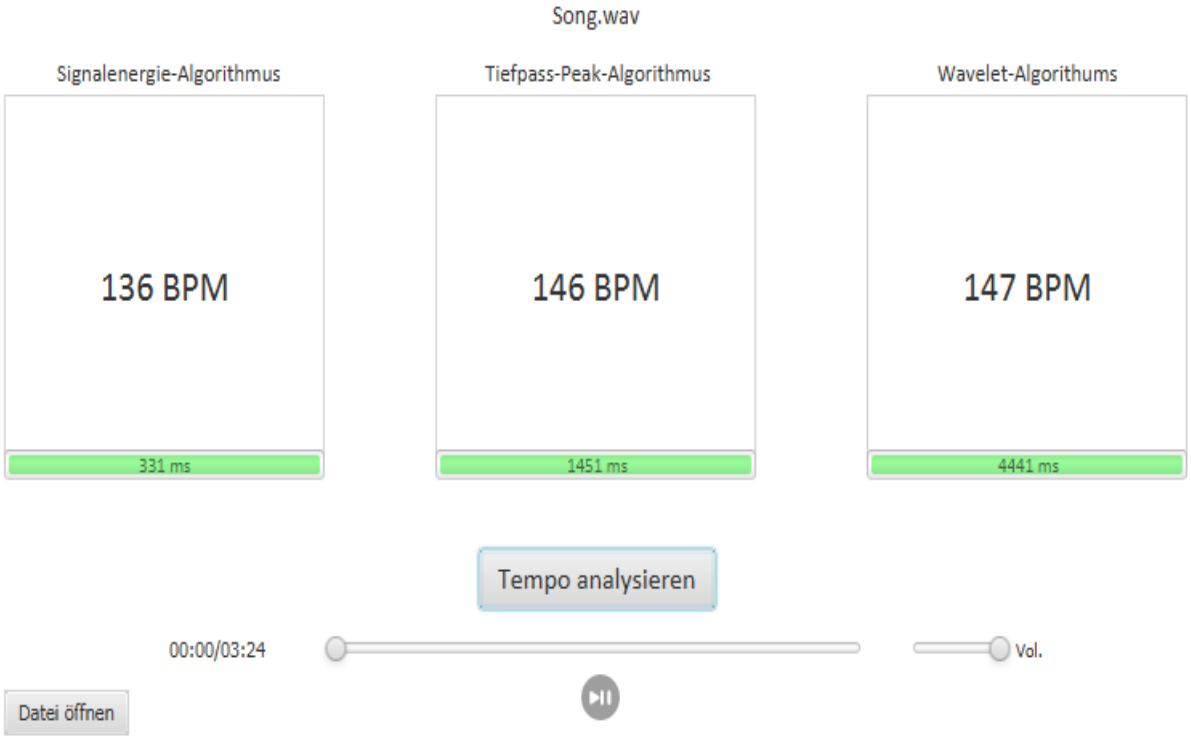

Abbildung 32: Die grafische Oberfläche des Programms “BPM-Finder” nach der Analyse eines Musikstücks

8.2 Mediaplayer

Unterhalb der Ausgabefelder und des “Tempo analysieren”-Buttons befindet sich das Bedienfeld für einen Media-Player, der die grundlegenden Funktionen zur Wiedergabe des geladenen Musikstücks bietet. In der Mitte befindet sich ein Schieberegler zum Navigieren durch den Song. Zentral darunter kann die Wiedergabe über einen Kreis-Button gestartet oder pausiert werden. Links neben dem Navigations-Schieberegler wird die Position im Song im Format “mm:ss” angezeigt, rechts lässt sich die Lautstärke über einen weiteren Schieberegler einstellen.

8.3 Klassen und Funktionen

Die folgende Tabelle 2 soll einen Überblick über alle Java-Klassen und deren Funktion des BPM-Finder-Programms geben.

AudioPlayer.java	Initialisiert das Programm und ruft die Controller-Klasse auf
fft.java	Umsetzung der schnellen Fouriertransformation (FFT) und der schnellen inversen Fouriertransformation (IFFT) Von Nayuki Minase (http://www.nayuki.io/page/free-small-fft-in-multiple-languages).
FileHandler.java	Ruft die zu analysierenden Dateien auf und verwaltet diese
LowPassPeakAlgorithm.java	Umsetzung des "Algorithmus 2: Peaks im tiefen Spektralbereich" (siehe Abschnitt 5)
MediaControl.java MediaControlBeanInfo.java *(mediacontrol.css, MediaControl.fxml)*	MediaControl.java: Controller-Klasse - Steuert die GUI und verarbeitet Eingaben des Benutzers MediaControlBeanInfo.java: Automatisch generierte Klasse zur Speicherung des GUI-Layouts
SoundEnergyAlgorithm.java	Umsetzung des "Algorithmus 1: Signalenergie" (siehe Abschnitt 4)
WaveletAlgorithm.java	Umsetzung des "Algorithmus 3: Wavelet Dekomposition und Autokorrelation" (siehe Abschnitt 6)
WavFile.java WavFileException.java	"Java Wav File IO" Klasse zum Einlesen von WAVE-Dateien, geschrieben von Dr. Andrew Greensted (http://www.labbookpages.co.uk/audio/javaWavFiles.html).
WavFilter.java	Klasse zum Filtern der WAVE-Dateien, beruht auf der Bibliothek "Java-DSP-Collection" (http://www.source-code.biz/dsp/java/)

Tabelle 2: Übersicht über die Java-Klassen und deren Funktion

9. Zusammenfassung und Ausblick

9.1 Zusammenfassung

Zunächst haben wir uns mit der musikalischen Bedeutung von Rhythmus und Tempo auseinandergesetzt. Daraus ergibt sich vor allem das Problem, dass die zeitlichen Dauern von Notenwerten oder Takten in der Musik alle relativ zueinander organisiert sind. Daher haben wir für unsere Arbeit festgelegt, dass der doppelte und der halbe Wert des zu untersuchenden BPM-Werts auch ein gültiges Ergebnis für das Musiktempo darstellt. Mit einer Genauigkeit von 10% sollen die BPM-Werte in einem Intervall von 60 BPM bis 200 BPM durch das BPM-Finder-Programm erkannt werden. Der ausgegebene BPM-Wert muss eine ganze Zahl ohne Nachkommastellen sein, das Programm wird aufgrund des guten Kompromiss zwischen Performance und Systemunabhängigkeit in der Programmiersprache JAVA geschrieben.

Wir entschieden uns für Algorithmen, die Notenanschläge durch deren Einschwingvorgang erkennen. Da kurze Einschwingvorgänge durch Algorithmen gut erkannt werden können, wird gefordert, dass in dem zu untersuchenden Musikstück ein Instrument aus der Rhythmussektion durchgehend spielt. Die ladbare Audiodatei soll das Format WAVE (16 Bit, 44,1 kHz) haben, da die Algorithmen mit den Abtastwerten rechnen.

Daraufhin haben wir uns nacheinander mit drei verschiedenen Algorithmen befasst, die in zwei Schritten zunächst die Beats im Audiosignal erkennen und anschließend aus den gewonnenen Informationen einen BPM-Wert bilden. Im Vergleich hat sich gezeigt, dass im gewählten Testverfahren kein Algorithmus das Kriterium der Genauigkeit von 10% vollständig erfüllt, aber alles in allem das Ergebnis zufriedenstellend ist, da in vielen Fällen besonders die Algorithmen 2 und 3 sehr exakte Ergebnisse für das Musiktempo liefern konnten.

9.2 Verlauf des BPM-Werts über die Zeit

Eigentlich war ein Ziel, die BPM-Werte über die Zeit aus einer Audiodatei auszuwerten. Dies sollte eigentlich im zweiten Schritt passieren, nachdem ein Algorithmus für die BPM-Erkennung entwickelt wurde. Es hat sich aber gezeigt, dass die Algorithmen erst nach einer relativ langen Zeit durch das statistisch häufige Vorkommen eines bestimmten BPM-Werts diesen als Ergebnis bestimmen konnten. Es ist somit für uns kein Ansatz eingefallen, wie ein Tempowechsel eindeutig hätte erkannt werden können. Aus diesem Grund ist dieses Ziel als gescheitert anzusehen.

9.3 Ausblick

Während der Entwicklung an den verschiendenen Algorithmen und dem anschließenden Vergleich hat sich gezeigt, wie stark die Abhängigkeit vom Audiosignal ist. Kein Algorithmus ist für alle Musiksignale gleichzeitig gut geeignet, die “Idee von einem universellen Modell für das Beat tracking ist zwar ein attraktives Ziel, es gibt aber starke Argumente, warum dies (derzeit) unrealistisch ist.” (ZAPATA, et al., 2014, S. 816, Sp. 2, Z. 28ff.) Stattdessen stellt ZAPATA (et al., 2014) ein System vor, das er “Multi-Feature Beat Tracking” nennt. Ein Überblick über das System ist in Abbildung 33 gegeben. F1…Fn stellen verschiedene Ansätze dar, die dafür zuständig sind, “onsets” bzw. Beats aus einem Audiosignal (Audio) zu erkennen. Übertragen auf unser Vorgehen würden an dieser Stelle unsere drei Algorithmen arbeiten. Algorithmus 1 erkennt Beats über “Energie-Schübe”, Algorithmus 2 über Peaks im tiefen Spektralbereich und Algorithmus 3 über eine “onset”-“Detection function”. Das übergeordnete System verfügt über Prinzipien, um die aus den verschiedenen Algorithmen ermittelten Werte auf Sinnhaftigkeit zu untersuchen. Dazu wird beispielsweise eine sogenannte “Mutual Agreement” berechnet, die ein Maß dafür bestimmt, welche Ergebnisse aus welchem Algorithmus wahr sind und welche nicht. So kann im Abschnitt Selection entschieden werden, welcher Ansatz für das aktuelle Audiosignal am besten funktioniert. So kann dynamisch auf verschiedene Audiosignale reagiert werden und das jeweils geeignetste Verfahren für die Beat-Erkennung ausgewählt werden.

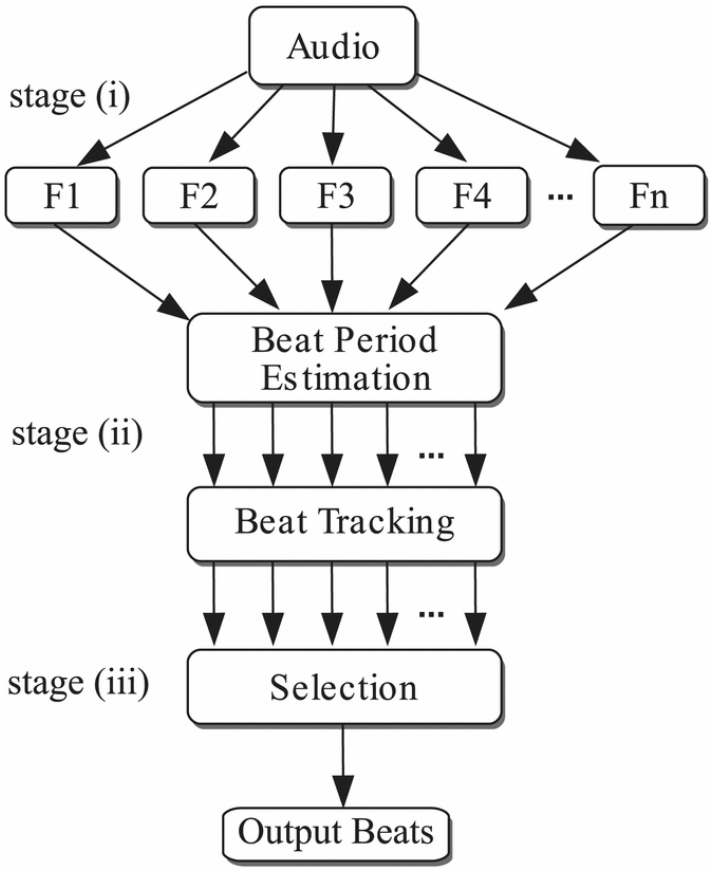

Abbildung 33: Überblick über das “Multi-Feature Beat Tracking”. (i): F1…Fn stellen verschiedene Algorithmen für die “onset”-Erkennnung dar, (ii): Beat tracking, (iii): dynamische Auswahl des richtigen Algorithmus

Quelle: Zapata, José R. et al., 2014, S. 817, Fig. 1.

10. Literatur und Quellenangaben

Hewitt, Michael: Music Theory for Computer Musicians, Boston (Cengage Learning), 2008

Scheirer, Eric D.: Tempo and beat analysis of acoustic musical signals, in: The Journal of the Acoustical Society of America 103 (1) (Januar 1998), S. 588-601

Goto, Masataka; Muraoka, Yoichi: Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions, in: Speech Communication Volume 27 (April 1999), S. 311-335

Bello, J.P.; Daudet, L.; Abdallah, S.; Duxbury, C.; Davies, M.; Sandler, Mark B.: A Tutorial on Onset Detection in Music Signals, in IEEE Transactions on Speech and Audio Processing, vol. 13, no. 5 (September 2005), S. 1035-1047

Greensted, Dr. Andrew: Java Wav File IO, 2010
http://www.labbookpages.co.uk/audio/javaWavFiles.html (08.11.2015)

Mertins, Alfred: "Signaltheorie: Grundlagen der Signalbeschreibung, Filterbänke, Wavelets, Zeit-Frequenz-Analyse, Parameter- und Signalschätzung", Wiesbaden (Springer Vieweg), 2012

Beatport: Netsky - Memory Lane (Original Mix), 2010
https://pro.beatport.com/track/memory-lane-original-mix/1623633 (23.12.2015)

Scheiblich, Christian: JWave, 2015
https://github.com/cscheiblich/JWave (10.12.2015)

Tzanetakis, George; Essl, Georg; Cook, Perry: Audio Analysis using the Discrete Wavelet Transform
http://www.cs.cmu.edu/~gtzan/work/pubs/amta01gtzan.pdf (18.12.2015)

Nayuki, Minase: Free small FFT in multiple languages, 2014
http://www.nayuki.io/page/free-small-fft-in-multiple-languages (08.11.2015)

Beatport: Jelle Slump - Dreams (Original Mix), 2015
https://pro.beatport.com/release/dreams/1673437 (23.12.2015)

The source-code.biz Java DSP collection
http://www.source-code.biz/dsp/java/ (08.12.2015)

Beatport: Technasia, Green Velvet - Suga (Original Mix), 2015
https://pro.beatport.com/track/suga-original-mix/6800712, 07.01.2016)

Zapata, J.R.; Davies, M.E.P.; Gomez, E.: Multi-Feature Beat Tracking, in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 4 (April 2014), S. 816-825

11. Bildverzeichnis

Abbildung 1, 2: eigene Abbildung

Abbildung 3: Bello, J.P.; Daudet, L.; Abdallah, S.; Duxbury, C.; Davies, M.; Sandler, Mark B.: A Tutorial on Onset Detection in Music Signals, in IEEE Transactions on Speech and Audio Processing, vol. 13, no. 5 (September 2005), S. 1035, Fig. 1

Abbildung 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15: eigene Abbildung

Abbildung 16: Bello, J.P.; Daudet, L.; Abdallah, S.; Duxbury, C.; Davies, M.; Sandler, Mark B.: A Tutorial on Onset Detection in Music Signals, in IEEE Transactions on Speech and Audio Processing, vol. 13, no. 5 (September 2005), S. 1036, Fig. 2

Abbildung 17: Mertins, Alfred: "Signaltheorie: Grundlagen der Signalbeschreibung, Filterbänke, Wavelets, Zeit-Frequenz-Analyse, Parameter- und Signalschätzung", Wiesbaden (Springer Vieweg), 2012, S. 320, Bild 9.11

Abbildung 18: MathWorks: wavedec (Multilevel 1-D Wavelet decomposition) http://de.mathworks.com/help/wavelet/ref/wavedec.html, Bild 5 (27.12.2015)

Abbildung 19: MathWorks: wavedec (Multilevel 1-D Wavelet decomposition) http://de.mathworks.com/help/wavelet/ref/wavedec.html, Bild 1 (27.12.2015)

Abbildung 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32: eigene Abbildung

Abbildung 33: Zapata, J.R.; Davies, M.E.P.; Gomez, E.: Multi-Feature Beat Tracking, in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 4 (April 2014), S. 817, Fig. 1

11. Anhang

11.1 Ergebnisse der Auswertung

In rot sind die Fälle markiert, in denen die relative Abweichung größer ist als die festlegten 10 %.

Tempo	Algorithmus 1: Soundenergie	Algorithmus 2: Tiefpass & Peaks	Algorithmus 3: DWT & AKF
60 BPM	120 BPM in 50 ms (0.00 %)	120 BPM in 162 ms (0.00 %)	120 BPM in 1184 ms (0.00 %)
61 BPM	122 BPM in 50 ms (0.00 %)	122 BPM in 165 ms (0.00 %)	122 BPM in 1193 ms (0.00 %)
62 BPM	125 BPM in 53 ms (0.81 %)	124 BPM in 170 ms (0.00 %)	124 BPM in 1306 ms (0.00 %)
63 BPM	125 BPM in 54 ms (-0.79 %)	126 BPM in 168 ms (0.00 %)	125 BPM in 1265 ms (-0.79 %)
64 BPM	128 BPM in 52 ms (0.00 %)	128 BPM in 168 ms (0.00 %)	128 BPM in 1264 ms (0.00 %)
65 BPM	130 BPM in 52 ms (0.00 %)	130 BPM in 169 ms (0.00 %)	130 BPM in 1239 ms (0.00 %)
66 BPM	133 BPM in 52 ms (0.76 %)	132 BPM in 168 ms (0.00 %)	132 BPM in 1234 ms (0.00 %)
67 BPM	133 BPM in 51 ms (-0.75 %)	134 BPM in 167 ms (0.00 %)	134 BPM in 1238 ms (0.00 %)
68 BPM	136 BPM in 52 ms (0.00 %)	136 BPM in 168 ms (0.00 %)	136 BPM in 1265 ms (0.00 %)
69 BPM	140 BPM in 53 ms (1.45 %)	138 BPM in 168 ms (0.00 %)	138 BPM in 1240 ms (0.00 %)
70 BPM	140 BPM in 57 ms (0.00 %)	140 BPM in 167 ms (0.00 %)	140 BPM in 1234 ms (0.00 %)
71 BPM	143 BPM in 52 ms (0.70 %)	142 BPM in 168 ms (0.00 %)	137 BPM in 1254 ms (-3.52 %)
72 BPM	143 BPM in 51 ms (-0.69 %)	144 BPM in 168 ms (0.00 %)	127 BPM in 1219 ms (-11.81 %)
73 BPM	146 BPM in 52 ms (0.00 %)	146 BPM in 164 ms (0.00 %)	118 BPM in 1195 ms (-19.18 %)
74 BPM	146 BPM in 50 ms (-1.35 %)	148 BPM in 166 ms (0.00 %)	111 BPM in 1190 ms (-25.00 %)
75 BPM	150 BPM in 52 ms (0.00 %)	150 BPM in 165 ms (0.00 %)	105 BPM in 1210 ms (-30.00 %)
76 BPM	154 BPM in 50 ms (1.32 %)	152 BPM in 165 ms (0.00 %)	100 BPM in 1178 ms (31.58 %)
77 BPM	154 BPM in 50 ms (0.00 %)	154 BPM in 166 ms (0.00 %)	100 BPM in 1224 ms (29.87 %)
78 BPM	158 BPM in 53 ms (1.28 %)	156 BPM in 170 ms (0.00 %)	157 BPM in 1260 ms (0.64 %)
79 BPM	158 BPM in 53 ms (0.00 %)	158 BPM in 171 ms (0.00 %)	158 BPM in 1218 ms (0.00 %)
80 BPM	162 BPM in 52 ms (1.25 %)	160 BPM in 167 ms (0.00 %)	160 BPM in 1195 ms (0.00 %)
81 BPM	162 BPM in 51 ms (0.00 %)	162 BPM in 176 ms (0.00 %)	161 BPM in 1190 ms (-0.62 %)
82 BPM	162 BPM in 51 ms (-1.22 %)	164 BPM in 168 ms (0.00 %)	164 BPM in 1232 ms (0.00 %)
83 BPM	167 BPM in 51 ms (0.60 %)	166 BPM in 176 ms (0.00 %)	166 BPM in 1237 ms (0.00 %)
84 BPM	167 BPM in 51 ms (-0.60 %)	168 BPM in 166 ms (0.00 %)	168 BPM in 1215 ms (0.00 %)
85 BPM	171 BPM in 52 ms (0.59 %)	170 BPM in 164 ms (0.00 %)	169 BPM in 1194 ms (-0.59 %)
86 BPM	171 BPM in 52 ms (-0.58 %)	172 BPM in 170 ms (0.00 %)	101 BPM in 1248 ms (17.44 %)
87 BPM	176 BPM in 52 ms (1.15 %)	174 BPM in 167 ms (0.00 %)	100 BPM in 1265 ms (14.94 %)

Tempo	Algorithmus 1: Soundenergie	Algorithmus 2: Tiefpass & Peaks	Algorithmus 3: DWT & AKF
88 BPM	171 BPM in 55 ms (-2.84 %)	176 BPM in 171 ms (0.00 %)	100 BPM in 1257 ms (13.64 %)
89 BPM	182 BPM in 51 ms (2.25 %)	178 BPM in 173 ms (0.00 %)	100 BPM in 1284 ms (12.36 %)
90 BPM	188 BPM in 51 ms (4.44 %)	180 BPM in 167 ms (0.00 %)	100 BPM in 1240 ms (11.11 %)
91 BPM	188 BPM in 56 ms (3.30 %)	182 BPM in 170 ms (0.00 %)	100 BPM in 1258 ms (9.89 %)
92 BPM	176 BPM in 53 ms (-4.35 %)	184 BPM in 184 ms (0.00 %)	101 BPM in 1240 ms (9.78 %)
93 BPM	176 BPM in 52 ms (-5.38 %)	186 BPM in 177 ms (0.00 %)	100 BPM in 1218 ms (7.53 %)
94 BPM	200 BPM in 51 ms (6.38 %)	188 BPM in 170 ms (0.00 %)	101 BPM in 1200 ms (7.45 %)
95 BPM	200 BPM in 50 ms (5.26 %)	190 BPM in 164 ms (0.00 %)	101 BPM in 1204 ms (6.32 %)
96 BPM	103 BPM in 53 ms (7.29 %)	192 BPM in 167 ms (0.00 %)	102 BPM in 1220 ms (6.25 %)
97 BPM	103 BPM in 51 ms (6.19 %)	194 BPM in 171 ms (0.00 %)	101 BPM in 1179 ms (4.12 %)
98 BPM	182 BPM in 51 ms (-7.14 %)	196 BPM in 165 ms (0.00 %)	100 BPM in 1191 ms (2.04 %)
99 BPM	107 BPM in 51 ms (8.08 %)	198 BPM in 166 ms (0.00 %)	101 BPM in 1254 ms (2.02 %)
100 BPM	107 BPM in 58 ms (7.00 %)	133 BPM in 201 ms (33.00 %)	100 BPM in 1500 ms (0.00 %)
101 BPM	111 BPM in 51 ms (9.90 %)	101 BPM in 165 ms (0.00 %)	101 BPM in 1246 ms (0.00 %)
102 BPM	111 BPM in 51 ms (8.82 %)	102 BPM in 174 ms (0.00 %)	102 BPM in 1283 ms (0.00 %)
103 BPM	188 BPM in 54 ms (-8.74 %)	103 BPM in 174 ms (0.00 %)	103 BPM in 1315 ms (0.00 %)
104 BPM	115 BPM in 54 ms (10.58 %)	104 BPM in 168 ms (0.00 %)	104 BPM in 1269 ms (0.00 %)
105 BPM	105 BPM in 52 ms (0.00 %)	105 BPM in 176 ms (0.00 %)	105 BPM in 1290 ms (0.00 %)
106 BPM	105 BPM in 52 ms (-0.94 %)	106 BPM in 176 ms (0.00 %)	106 BPM in 1227 ms (0.00 %)
107 BPM	107 BPM in 51 ms (0.00 %)	107 BPM in 172 ms (0.00 %)	107 BPM in 1275 ms (0.00 %)
108 BPM	107 BPM in 52 ms (-0.93 %)	108 BPM in 167 ms (0.00 %)	108 BPM in 1222 ms (0.00 %)
109 BPM	109 BPM in 52 ms (0.00 %)	109 BPM in 172 ms (0.00 %)	109 BPM in 1243 ms (0.00 %)
110 BPM	109 BPM in 53 ms (-0.91 %)	110 BPM in 183 ms (0.00 %)	110 BPM in 1222 ms (0.00 %)
111 BPM	111 BPM in 52 ms (0.00 %)	111 BPM in 166 ms (0.00 %)	111 BPM in 1207 ms (0.00 %)
112 BPM	111 BPM in 52 ms (-0.89 %)	112 BPM in 166 ms (0.00 %)	112 BPM in 1219 ms (0.00 %)
113 BPM	113 BPM in 50 ms (0.00 %)	113 BPM in 166 ms (0.00 %)	113 BPM in 1225 ms (0.00 %)
114 BPM	113 BPM in 52 ms (-0.88 %)	114 BPM in 174 ms (0.00 %)	114 BPM in 1228 ms (0.00 %)
115 BPM	115 BPM in 52 ms (0.00 %)	115 BPM in 174 ms (0.00 %)	115 BPM in 1271 ms (0.00 %)
116 BPM	115 BPM in 53 ms (-0.86 %)	116 BPM in 169 ms (0.00 %)	116 BPM in 1212 ms (0.00 %)
117 BPM	118 BPM in 54 ms (0.85 %)	117 BPM in 184 ms (0.00 %)	117 BPM in 1261 ms (0.00 %)
118 BPM	118 BPM in 53 ms (0.00 %)	118 BPM in 186 ms (0.00 %)	118 BPM in 1248 ms (0.00 %)
119 BPM	120 BPM in 53 ms (0.84 %)	119 BPM in 172 ms (0.00 %)	119 BPM in 1243 ms (0.00 %)
120 BPM	120 BPM in 51 ms (0.00 %)	120 BPM in 175 ms (0.00 %)	120 BPM in 1254 ms (0.00 %)
121 BPM	120 BPM in 51 ms (-0.83 %)	121 BPM in 166 ms (0.00 %)	121 BPM in 1199 ms (0.00 %)

Tempo	Algorithmus 1: Soundenergie	Algorithmus 2: Tiefpass & Peaks	Algorithmus 3: DWT & AKF
122 BPM	122 BPM in 51 ms (0.00 %)	122 BPM in 167 ms (0.00 %)	122 BPM in 1195 ms (0.00 %)
123 BPM	122 BPM in 53 ms (-0.81 %)	123 BPM in 165 ms (0.00 %)	123 BPM in 1198 ms (0.00 %)
124 BPM	125 BPM in 51 ms (0.81 %)	124 BPM in 165 ms (0.00 %)	124 BPM in 1211 ms (0.00 %)
125 BPM	125 BPM in 55 ms (0.00 %)	125 BPM in 171 ms (0.00 %)	125 BPM in 1227 ms (0.00 %)
126 BPM	125 BPM in 52 ms (-0.79 %)	126 BPM in 175 ms (0.00 %)	126 BPM in 1219 ms (0.00 %)
127 BPM	128 BPM in 52 ms (0.79 %)	127 BPM in 171 ms (0.00 %)	127 BPM in 1211 ms (0.00 %)
128 BPM	128 BPM in 52 ms (0.00 %)	128 BPM in 171 ms (0.00 %)	128 BPM in 1294 ms (0.00 %)
129 BPM	128 BPM in 52 ms (-0.78 %)	129 BPM in 166 ms (0.00 %)	129 BPM in 1242 ms (0.00 %)
130 BPM	130 BPM in 51 ms (0.00 %)	130 BPM in 170 ms (0.00 %)	130 BPM in 1226 ms (0.00 %)
131 BPM	130 BPM in 52 ms (-0.76 %)	131 BPM in 165 ms (0.00 %)	131 BPM in 1212 ms (0.00 %)
132 BPM	133 BPM in 54 ms (0.76 %)	132 BPM in 170 ms (0.00 %)	132 BPM in 1262 ms (0.00 %)
133 BPM	133 BPM in 52 ms (0.00 %)	133 BPM in 165 ms (0.00 %)	133 BPM in 1199 ms (0.00 %)
134 BPM	133 BPM in 52 ms (-0.75 %)	134 BPM in 169 ms (0.00 %)	134 BPM in 1189 ms (0.00 %)
135 BPM	136 BPM in 51 ms (0.74 %)	135 BPM in 164 ms (0.00 %)	135 BPM in 1212 ms (0.00 %)
136 BPM	136 BPM in 51 ms (0.00 %)	136 BPM in 171 ms (0.00 %)	136 BPM in 1193 ms (0.00 %)
137 BPM	136 BPM in 52 ms (-0.73 %)	137 BPM in 167 ms (0.00 %)	137 BPM in 1207 ms (0.00 %)
138 BPM	140 BPM in 52 ms (1.45 %)	138 BPM in 166 ms (0.00 %)	138 BPM in 1230 ms (0.00 %)
139 BPM	140 BPM in 53 ms (0.72 %)	139 BPM in 178 ms (0.00 %)	139 BPM in 1208 ms (0.00 %)
140 BPM	140 BPM in 51 ms (0.00 %)	140 BPM in 167 ms (0.00 %)	140 BPM in 1247 ms (0.00 %)
141 BPM	140 BPM in 52 ms (-0.71 %)	141 BPM in 168 ms (0.00 %)	141 BPM in 1243 ms (0.00 %)
142 BPM	143 BPM in 51 ms (0.70 %)	142 BPM in 165 ms (0.00 %)	142 BPM in 1251 ms (0.00 %)
143 BPM	143 BPM in 52 ms (0.00 %)	143 BPM in 170 ms (0.00 %)	143 BPM in 1220 ms (0.00 %)
144 BPM	143 BPM in 52 ms (-0.69 %)	144 BPM in 189 ms (0.00 %)	144 BPM in 1222 ms (0.00 %)
145 BPM	146 BPM in 54 ms (0.69 %)	145 BPM in 169 ms (0.00 %)	145 BPM in 1243 ms (0.00 %)
146 BPM	146 BPM in 50 ms (0.00 %)	146 BPM in 168 ms (0.00 %)	146 BPM in 1227 ms (0.00 %)
147 BPM	146 BPM in 53 ms (-0.68 %)	147 BPM in 171 ms (0.00 %)	147 BPM in 1250 ms (0.00 %)
148 BPM	146 BPM in 52 ms (-1.35 %)	148 BPM in 168 ms (0.00 %)	148 BPM in 1276 ms (0.00 %)
149 BPM	150 BPM in 53 ms (0.67 %)	149 BPM in 170 ms (0.00 %)	149 BPM in 1287 ms (0.00 %)
150 BPM	150 BPM in 55 ms (0.00 %)	150 BPM in 167 ms (0.00 %)	150 BPM in 1227 ms (0.00 %)
151 BPM	150 BPM in 53 ms (-0.66 %)	151 BPM in 177 ms (0.00 %)	151 BPM in 1240 ms (0.00 %)
152 BPM	154 BPM in 52 ms (1.32 %)	152 BPM in 165 ms (0.00 %)	152 BPM in 1256 ms (0.00 %)
153 BPM	154 BPM in 64 ms (0.65 %)	153 BPM in 180 ms (0.00 %)	153 BPM in 1219 ms (0.00 %)
154 BPM	154 BPM in 50 ms (0.00 %)	154 BPM in 166 ms (0.00 %)	154 BPM in 1213 ms (0.00 %)
155 BPM	154 BPM in 51 ms (-0.65 %)	155 BPM in 171 ms (0.00 %)	155 BPM in 1219 ms (0.00 %)

Tempo	Algorithmus 1: Soundenergie	Algorithmus 2: Tiefpass & Peaks	Algorithmus 3: DWT & AKF
156 BPM	158 BPM in 52 ms (1.28 %)	156 BPM in 170 ms (0.00 %)	156 BPM in 1275 ms (0.00 %)
157 BPM	158 BPM in 53 ms (0.64 %)	157 BPM in 172 ms (0.00 %)	157 BPM in 1265 ms (0.00 %)
158 BPM	158 BPM in 53 ms (0.00 %)	158 BPM in 170 ms (0.00 %)	158 BPM in 1209 ms (0.00 %)
159 BPM	158 BPM in 50 ms (-0.63 %)	159 BPM in 191 ms (0.00 %)	159 BPM in 1280 ms (0.00 %)
160 BPM	162 BPM in 52 ms (1.25 %)	160 BPM in 171 ms (0.00 %)	160 BPM in 1248 ms (0.00 %)
161 BPM	162 BPM in 54 ms (0.62 %)	161 BPM in 180 ms (0.00 %)	161 BPM in 1282 ms (0.00 %)
162 BPM	162 BPM in 52 ms (0.00 %)	162 BPM in 175 ms (0.00 %)	162 BPM in 1213 ms (0.00 %)
163 BPM	162 BPM in 53 ms (-0.61 %)	163 BPM in 168 ms (0.00 %)	163 BPM in 1213 ms (0.00 %)
164 BPM	162 BPM in 50 ms (-1.22 %)	164 BPM in 171 ms (0.00 %)	164 BPM in 1208 ms (0.00 %)
165 BPM	167 BPM in 50 ms (1.21 %)	165 BPM in 164 ms (0.00 %)	165 BPM in 1206 ms (0.00 %)
166 BPM	167 BPM in 52 ms (0.60 %)	166 BPM in 166 ms (0.00 %)	166 BPM in 1232 ms (0.00 %)
167 BPM	167 BPM in 53 ms (0.00 %)	167 BPM in 172 ms (0.00 %)	167 BPM in 1250 ms (0.00 %)
168 BPM	167 BPM in 53 ms (-0.60 %)	168 BPM in 165 ms (0.00 %)	168 BPM in 1238 ms (0.00 %)
169 BPM	171 BPM in 52 ms (1.18 %)	169 BPM in 167 ms (0.00 %)	169 BPM in 1213 ms (0.00 %)
170 BPM	171 BPM in 53 ms (0.59 %)	170 BPM in 186 ms (0.00 %)	170 BPM in 1244 ms (0.00 %)
171 BPM	171 BPM in 51 ms (0.00 %)	171 BPM in 170 ms (0.00 %)	171 BPM in 1242 ms (0.00 %)
172 BPM	171 BPM in 52 ms (-0.58 %)	172 BPM in 168 ms (0.00 %)	172 BPM in 1220 ms (0.00 %)
173 BPM	171 BPM in 52 ms (-1.16 %)	173 BPM in 170 ms (0.00 %)	173 BPM in 1218 ms (0.00 %)
174 BPM	176 BPM in 52 ms (1.15 %)	174 BPM in 167 ms (0.00 %)	174 BPM in 1227 ms (0.00 %)
175 BPM	176 BPM in 50 ms (0.57 %)	175 BPM in 165 ms (0.00 %)	175 BPM in 1269 ms (0.00 %)
176 BPM	176 BPM in 58 ms (0.00 %)	176 BPM in 169 ms (0.00 %)	176 BPM in 1227 ms (0.00 %)
177 BPM	176 BPM in 52 ms (-0.56 %)	177 BPM in 184 ms (0.00 %)	177 BPM in 1237 ms (0.00 %)
178 BPM	176 BPM in 52 ms (-1.12 %)	178 BPM in 183 ms (0.00 %)	178 BPM in 1377 ms (0.00 %)
179 BPM	176 BPM in 59 ms (-1.68 %)	179 BPM in 195 ms (0.00 %)	179 BPM in 1210 ms (0.00 %)
180 BPM	182 BPM in 51 ms (1.11 %)	180 BPM in 163 ms (0.00 %)	180 BPM in 1186 ms (0.00 %)
181 BPM	182 BPM in 53 ms (0.55 %)	181 BPM in 170 ms (0.00 %)	181 BPM in 1258 ms (0.00 %)
182 BPM	182 BPM in 57 ms (0.00 %)	182 BPM in 191 ms (0.00 %)	182 BPM in 1298 ms (0.00 %)
183 BPM	182 BPM in 55 ms (-0.55 %)	183 BPM in 182 ms (0.00 %)	183 BPM in 1208 ms (0.00 %)
184 BPM	182 BPM in 51 ms (-1.09 %)	184 BPM in 166 ms (0.00 %)	184 BPM in 1224 ms (0.00 %)
185 BPM	188 BPM in 53 ms (1.62 %)	185 BPM in 169 ms (0.00 %)	185 BPM in 1202 ms (0.00 %)
186 BPM	188 BPM in 55 ms (1.08 %)	186 BPM in 168 ms (0.00 %)	186 BPM in 1206 ms (0.00 %)
187 BPM	188 BPM in 51 ms (0.53 %)	187 BPM in 164 ms (0.00 %)	187 BPM in 1248 ms (0.00 %)
188 BPM	188 BPM in 51 ms (0.00 %)	188 BPM in 173 ms (0.00 %)	188 BPM in 1250 ms (0.00 %)
189 BPM	188 BPM in 51 ms (-0.53 %)	189 BPM in 166 ms (0.00 %)	189 BPM in 1237 ms (0.00 %)

Tempo	Algorithmus 1: Soundenergie	Algorithmus 2: Tiefpass & Peaks	Algorithmus 3: DWT & AKF
190 BPM	188 BPM in 52 ms (-1.05 %)	190 BPM in 174 ms (0.00 %)	190 BPM in 1239 ms (0.00 %)
191 BPM	194 BPM in 51 ms (1.57 %)	191 BPM in 167 ms (0.00 %)	191 BPM in 1236 ms (0.00 %)
192 BPM	194 BPM in 51 ms (1.04 %)	192 BPM in 165 ms (0.00 %)	192 BPM in 1200 ms (0.00 %)
193 BPM	194 BPM in 51 ms (0.52 %)	193 BPM in 163 ms (0.00 %)	193 BPM in 1199 ms (0.00 %)
194 BPM	194 BPM in 50 ms (0.00 %)	194 BPM in 165 ms (0.00 %)	194 BPM in 1202 ms (0.00 %)
195 BPM	194 BPM in 51 ms (-0.51 %)	195 BPM in 164 ms (0.00 %)	195 BPM in 1200 ms (0.00 %)
196 BPM	194 BPM in 51 ms (-1.02 %)	196 BPM in 164 ms (0.00 %)	196 BPM in 1162 ms (0.00 %)
197 BPM	200 BPM in 50 ms (1.52 %)	197 BPM in 164 ms (0.00 %)	197 BPM in 1197 ms (0.00 %)
198 BPM	200 BPM in 51 ms (1.01 %)	198 BPM in 165 ms (0.00 %)	198 BPM in 1171 ms (0.00 %)
199 BPM	200 BPM in 50 ms (0.50 %)	199 BPM in 165 ms (0.00 %)	199 BPM in 1224 ms (0.00 %)
200 BPM	200 BPM in 52 ms (0.00 %)	133 BPM in 166 ms (33.00 %)	100 BPM in 1186 ms (0.00 %)

11.2 CD-ROM