Universität für Musik und Darstellende Kunst Graz
Institut für Elektronische Musik und Akustik (IEM)

Projektarbeit

Resynthese von Audiosignalen mittels Feature Extraction

Johannes Luig
johannes.luig@student.kug.ac.at

April 2007

Betreut durch
o.Univ.-Prof. Mag.art. Dipl.-Ing. Dr. Robert Höldrich
Univ.-Ass. Dipl.-Ing. Dr. Alois Sontacchi

Zusammenfassung

Die vorliegende Projektarbeit stellt einen Ansatz zur Resynthese von Audiosignalen aus fremdem Audiomaterial vor – ein Musikstück soll aus den Einzelteilen anderer Stücke einer Musikbibliothek neu zusammengesetzt werden. Die große Herausforderung besteht darin, die „Ähnlichkeit“ zweier Klangeindrücke quantitativ zu erfassen; d.h. in Zahlen auszudrücken, die von einer Maschine interpretiert werden können.

Der Ansatz ist folgender: die Musikstücke werden jeweils in einzelne *Frames* im Millisekundenbereich unterteilt. Für jeden dieser Frames wird ein musikalischer „Fingerabdruck“ erstellt und gespeichert. Im Anschluss wird nun jedem Frame des *Target Songs* derjenige aus der Bibliothek zugewiesen, der den ähnlichsten Fingerabdruck aufweist.

Zum Erstellen eines solchen charakteristischen Profils werden ausschließlich aus dem Audiosignal selbst gewonnene Daten verwendet. So lassen sich im Zeit- wie im Frequenzbereich verschiedene *Features* extrahieren, mit deren Hilfe man die Frames in „musikalische Klassen“ einteilen und miteinander vergleichen kann.

Abstract

The present project thesis introduces an approach to re-synthesize audio signals from other audio material—a piece of music is to be reconstructed from fragments of other songs in a music library. The challenge is to quantify the “similarity” of sound impressions; i.e., to express it in numbers that can be interpreted by a machine.

The approach is as follows: the songs are each divided into single *frames* of several milliseconds. For every frame, a musical “fingerprint” is created and stored. Following, each frame of the *Target Song* is assigned to that frame from the library with the most similar fingerprint.

To create such a characteristic profile, solely data extracted from the audio signal itself are used. Both in the time and in the frequency domain, we can extract miscellaneous *features* which help us to “classify” and compare our frames.

Inhaltsverzeichnis

1 Überblick

In diesem Paper werden zwei alternative Implementationen vorgestellt, auf deren spezielle Eigenschaften noch näher eingegangen wird. Die beiden Methoden werden *Fixed Frame Length* (FFL) und *Transient/Stable Separation* (TSS) genannt.

Der prinzipielle Ablauf ist in beiden Fällen folgender: der Benutzer wählt eine beliebige Anzahl an Musikstücken aus, die anschließend sequentiell eingelesen, in kurze Abschnitte unterteilt und analysiert werden. Dabei wird neben der eigentlichen Feature-Extraktion auch auf die zeitliche Struktur des Signals geachtet, sprich: die Länge der Abschnitte sollte eine perzeptive Relevanz aufweisen.

Ist die Bibliothek vollständig, werden die chronologisch aneinandergereihten Frames sortiert, d.h. anhand bestimmter Merkmale in *Cluster* und *Sub-Cluster* eingordnet. Diese Maßnahme ist angesichts der Datenmenge höchst sinnvoll und verringert die Zahl der Vergleiche (und damit die Rechenzeit) dramatisch!

Das zu resynthetisierende Stück darf aus naheliegenden Gründen natürlich nicht in die Bibliothek aufgenommen werden. Es wird analog zu oben analysiert, bevor jedem Frame dieses *Target Songs* ein Sub-Cluster zugeordnet wird, in dem sich die „ähnlichsten“ Frames aus der Bibliothek befinden.

Im nächsten Schritt wird nun der Fingerabdruck eines jeden Frames mit den im zugehörigen Sub-Cluster verfügbaren verglichen und jeweils ein *Best Match* ermittelt. Auf diese Art und Weise entsteht eine Zuordnungstabelle, die dem Resynthese-Algorithmus mitteilt, an welcher Stelle er welchen Frame aus welchem Song mit welcher Länge einzufügen hat.

1.1 Analyse und Feature-Extraktion

1.1.1 FFL-Methode

Ein Song wird zur Bibliothek hinzugefügt, indem er in sich überlappende Frames unterteilt wird. Die obere Grenze für die Länge eines Frames ergibt sich durch die Forderung, dass man den Inhalt eines einzelnen Frames immer als „stationär“ betrachten können muss. In der Sprachsignalverarbeitung wählt man häufig eine Dauer von $\Delta t = 23,2ms$.

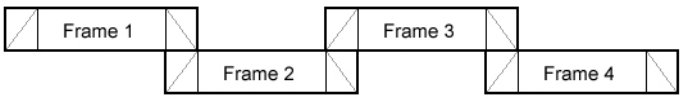

Abbildung 1.1: Feste Framelänge mit Overlap

Für Musik ist dieser Spielraum jedoch etwas größer: wählt man beispielsweise Sechzehntelnoten bei 200 bpm als „kleinste musikalische Einheit“, ergibt sich eine maximal erlaubte Framelänge von $\Delta t = 75,2ms$, was bedeutet, dass man hier bezüglich des oben genannten Kriteriums mit einem Dreifachen der Framelänge arbeiten kann. Zur Analyse werden die Wave-Files in Mono konvertiert und mit 11,025 kHz unterabgetastet – die Resynthese erfolgt selbstverständlich im Originalformat (44,1 kHz Stereo)!

Bei Schallereignissen, die sich über mehrere Frames stationär bzw. stabil verhalten, ist es wünschenswert, diesen Abschnitt komplett zu erhalten und eine gewisse Anzahl an Frames als zusammenhängend zu behandeln. Dazu wird zuerst eine *Peak Detection* durchgeführt, bevor beim anschließenden *Grouping* durch Korrelationsberechnungen die Länge des folgenden stabilen Abschnitts ermittelt wird.

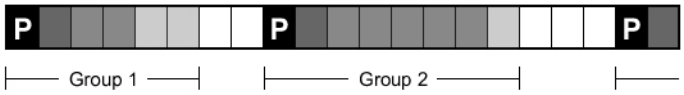

Abbildung 1.2: Gruppenbildung

Als Features werden nun die *Mel-Frequency Cepstrum Coefficients* (MFCCs, siehe Kap. 2.4) extrahiert. Sie repräsentieren das Spektrum in einer sehr kompakten Form, wobei die Auflösung variabel ist und mit der Zahl der verwendeten Koeffizienten steigt[1].

1.1.2 TSS-Methode

Ein Song wird zur Bibliothek hinzugefügt, indem er zuerst einer Analyse im Zeitbereich unterzogen wird. Der Algorithmus teilt das Signal in „transiente“ und „stabile“ Regionen ein, wobei letztere anschließend einem weiteren Analyseschritt unterzogen werden. Die Dauer eines transienten Ereignisses ist dabei festgelegt, die anschließende stabile Phase dauert jeweils bis zum Auftreten der nächsten Transienten – die Framelänge ist also (insgesamt gesehen) variabel.

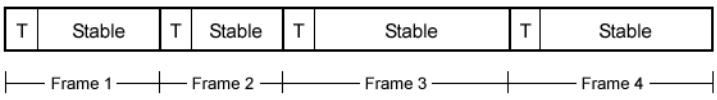

Abbildung 1.3: Variable Framelänge

Für die stabilen Regionen werden nun der *RMS*-Wert (im Zeitbereich) sowie die Grundtonhöhe (*Pitch*), der *Spectral Rolloff* und die MFCCs (im Frequenzbereich) ermittelt, wobei letztere schlussendlich zum Vergleich herangezogen werden. Detaillierte Beschreibungen dieser Features sind in Kap. 2 zu finden.

[1] In diesem Fall werden 20 MFCCs bei einer Analyse-Samplerate von 11,025 kHz verwendet.

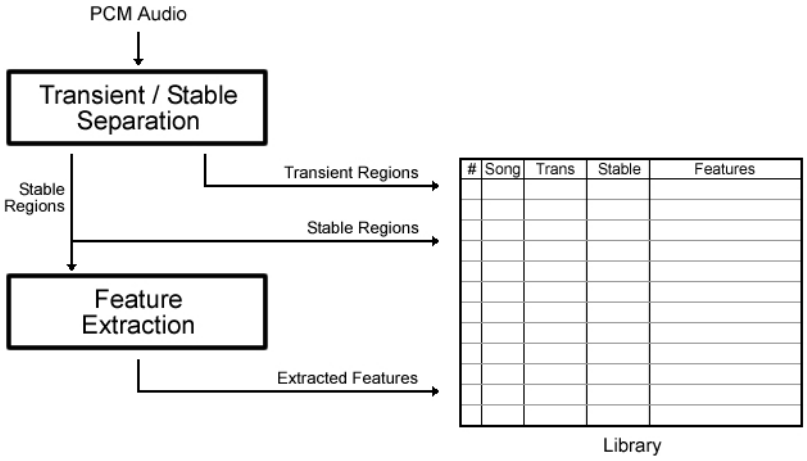

Abbildung 1.4: Feature Extraction (TSS-Methode)

1.2 Clustering

Ist die Bibliothek vollständig, werden die bislang chronologisch aneinandergereihten Frames sortiert: zuerst entscheidet die Grundtonhöhe (TSS) beziehungsweise der 2. Cepstralkoeffizient[2] (FFL) über die Zugehörigkeit zu einem Cluster, anschließend wird innerhalb der Gruppen anhand des spektralen Rolloff-Faktors (TSS) bzw. des 3. MFCC (FFL) noch einmal in Sub-Cluster unterteilt. Die Größe der Cluster sowie die mittlere Anzahl der Elemente pro Sub-Cluster können durch Initialisierung der entsprechenden Variablen verändert werden.

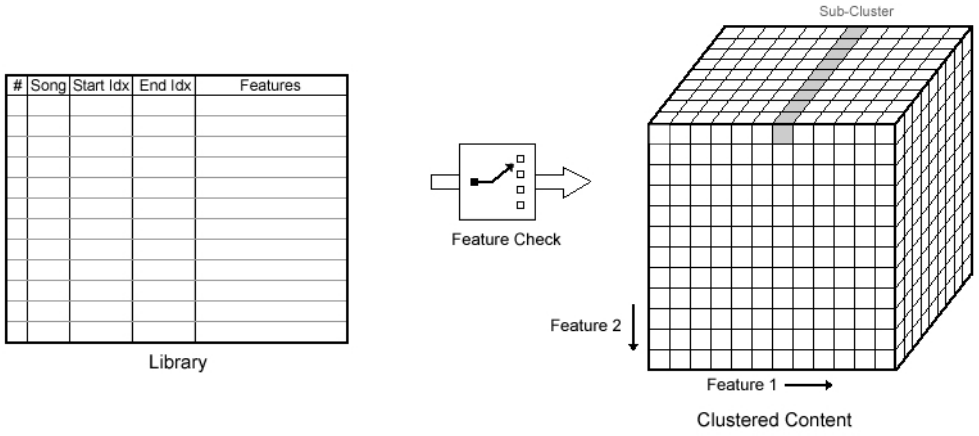

Abbildung 1.5: Clustering

Diese Maßnahme verringert die Rechenzeit enorm: Ein Standard-Popsong (3:30 min.) wird beispielsweise mit der TSS-Methode in ca. 500 Frames unterteilt, wobei für jeden Frame die

[2]Die MFCC-Indizierung wird nicht immer einheitlich gehandhabt: in der Literatur findet man sowohl die Zählweise „von Null weg“, als auch „von Eins weg“ – hier wird durchgängig letztere verwendet.

MFCCs zu vergleichen sind. Das würde für lediglich 10 Songs in der Bibliothek bereits einen Rechenaufwand von $5000^2 = 25$ Millionen Vergleichen zweier 20-dimensionaler Vektoren bedeuten! In einem Sub-Cluster befinden sich dagegen durchschnittlich etwa 25 Frames, was die Anzahl der Vektoren-Vergleiche für dieses Beispiel auf lediglich 625 drückt. Ergibt eine Reduktion des Rechenaufwands um den Faktor $4 \cdot 10^4$!

1.3 Cluster-Zuordnung

Der Target Song durchläuft denselben Analyseprozess wie die Stücke aus der Bibliothek – mit dem Unterschied, dass für ihn eine eigene Liste mit Informationen zu Song-Nr., Framelänge und extrahierten Features erstellt wird.

Im nächsten Schritt wird ermittelt, an welcher Position der Cluster-Algorithmus die einzelnen Frames einsortieren würde, um herauszufinden, wo die „ähnlichsten“ Pendants zu finden sind.

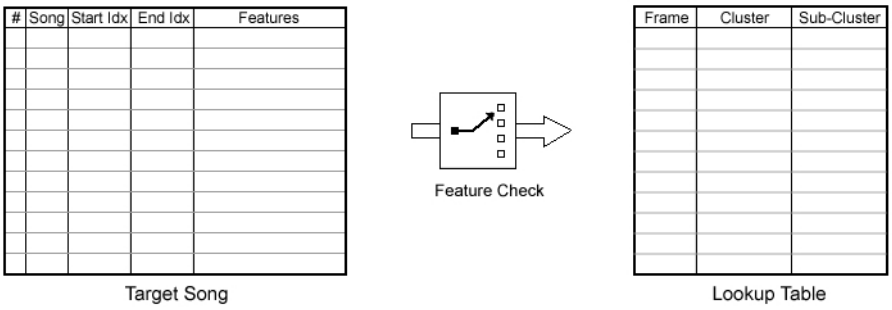

Abbildung 1.6: Zuweisen des passenden Sub-Clusters

1.4 Frame-Auswahl

Nun werden für jeden Frame des Target Songs die Elemente im zugewiesenen Sub-Cluster mit dem Original verglichen, wobei als Distanzmaß im Endeffekt ein einziger, skalarer Wert herauskommt. Die Informationen des Frames, der die geringste Differenz aufweist, werden tabellarisch erfasst; diese Tabelle wird im nächsten Schritt Punkt für Punkt abgearbeitet.

Sollten die beiden „Partner“ eine unterschiedliche zeitliche Dauer (TSS: Framelänge, FFL: Gruppengröße) aufweisen, gibt es zwei Möglichkeiten der Anpassung: ist das Pendant zum Target-Song-Frame zu lang, wird es einfach gekürzt; ist es zu kurz, wird es entweder geloopt (FFL) oder mittels *Time Scaling* (TSS, Näheres dazu in Kap. 4.5.1) auf die nötige Länge gebracht.

1.5 Resynthese

Anhand der ermittelten Daten kann der Target Song nun Stück für Stück „modelliert“ werden: die Zuordnungstabelle gibt vor, aus welchem Song welcher Bereich ausgelesen, eventuell mit

einem Gain Factor beaufschlagt, gekürzt oder verlängert und schließlich angefügt werden muss, um dem Original bestmöglich zu entsprechen.

Array Editor - synth_stats

Stack: Base

	1	2	3	4	5	6	7	8	9	10
1	6	3.4117e+006	3.4127e+006	3.4127e+006	3.415e+006	1.1594	0.78155	0	10.564	23483
2	1	9.3863e+006	9.3873e+006	9.3873e+006	9.3961e+006	1.1028	0.71652	0	2.6669	23295
3	3	6.3405e+006	6.3416e+006	6.3416e+006	6.3505e+006	1.0349	1.377	0	0	0
4	23	8.1282e+006	8.1293e+006	8.1293e+006	8.1303e+006	1.4339	0.40259	0	0	0
5	21	7.2542e+006	7.2552e+006	7.2552e+006	7.2658e+006	1.2435	1.5359	0	1.0941	11583
6	3	2.5945e+006	2.5955e+006	2.5955e+006	2.5998e+006	0.99377	0.55942	0	2.6928	11695
7	11	3.8791e+006	3.8801e+006	3.8801e+006	3.8911e+006	1.0177	1.3048	0	0	0
8	14	7.8134e+006	7.8145e+006	7.8145e+006	7.8259e+006	1.1623	1.4075	0	1.0529	11455
9	4	8.0615e+006	8.0625e+006	8.0625e+006	8.0644e+006	1.2132	2.9243	0	1.369	2523
10	21	2.1313e+006	2.1323e+006	2.1323e+006	2.1397e+006	1.084	2.2092	0	0	0
11	11	3.8791e+006	3.8801e+006	3.8801e+006	3.904e+006	0.85252	1.4559	0	1.0655	23895
12	8	6.8675e+006	6.8685e+006	6.8685e+006	6.8917e+006	0.99877	0.7791	0	1.0514	23175
13	3	5.6213e+006	5.6224e+006	5.6224e+006	5.6348e+006	0.62838	1.4465	0	0	0
14	8	9.2113e+006	9.2123e+006	9.2123e+006	9.2225e+006	1.062	0.88962	0	3.4967	10151
15	21	3.1526e+006	3.1537e+006	3.1537e+006	3.1562e+006	0.80046	2.0377	0	0	0
16	3	5.7551e+006	5.7561e+006	5.7561e+006	5.7646e+006	1.3535	1.1541	0	8.4492	8407
17	12	5.3787e+006	5.3797e+006	5.3797e+006	5.3906e+006	1.3262	0.90715	0	0	0
18	23	1.0635e+007	1.0636e+007	1.0636e+007	1.0646e+007	0.76794	0.80892	0	1.1761	12527
19	17	7.9206e+006	7.9216e+006	7.9216e+006	7.9314e+006	1.6887	0.79502	0	0	0
20	14	7.8624e+006	7.8634e+006	7.8634e+006	7.874e+006	1.1484	0.68456	0	2.2225	23467
21	14	7.8624e+006	7.8634e+006	7.8634e+006	7.8654e+006	1.1287	0.32471	0	1.7195	1931
22	23	8.1345e+006	8.1355e+006	8.1355e+006	8.1437e+006	1.2418	0.59665	0	1.2587	10295
23	14	1.2051e+007	1.2052e+007	1.2052e+007	1.2057e+007	1.4708	1.292	0	0	0
24	11	8.9663e+006	8.9674e+006	8.9674e+006	8.9714e+006	1.3822	1.9573	0	0	0
25	23	9.869e+006	9.8701e+006	9.8701e+006	9.8763e+006	1.4392	0.98629	0	1.797	11147

Abbildung 1.7: Zuordnungstabelle (TSS-Methode) mit Song-Nr., Start- und End-Indizes für transiente und stabile Regionen sowie Distanzmaß, Verstärkungsfaktor, Zero-Pad, Time-Stretch-Faktor und „Desired Length“

2 Features

2.1 Pitch

Eine sehr aussagekräftige Eigenschaft eines Klangs ist seine Grundtonhöhe. Diese wird nach einem Ansatz ermittelt, der in [1] vorgestellt wird: zunächst werden die lokalen Maxima des Betragsspektrums ermittelt, anschließend durch Berechnung der Phasendifferenz zwischen dem aktuellen und einem verschobenen Fenster die Frequenzauflösung erhöht. Als korrigierter Wert für die Frequenzen der Maxima ergibt sich schließlich

$$f_n(k) = \frac{\varphi_{n+1}(k) - \varphi_n(k)}{2\pi \cdot hopsize} \cdot f_s$$

(statt vorher $f = k \cdot f_s/N$), wobei *hopsize* die Verschiebung der beiden Analysefenster zueinander in Samples angibt. Im Anschluss wird aus dem Abstandmuster der „Kandidaten“ die Grundtonhöhe abgeleitet.

2.2 Spectral Rolloff

Ein Indikator für die Energieverteilung im Spektrum ist diejenige Frequenz, bis zu der sich ein gewisser Prozentsatz der Energie des Signalausschnitts aufsummiert. Der Grenzwert wird z.B. in [9] mit 85% angegeben. Dieser sogenannte Rolloff-Faktor wird wie folgt berechnet:

$$SR = min\left\{ X(R) \;\middle|\; \sum_{k=0}^{R} |X(k)|^2 \geq 0,85 \cdot \sum_{k=0}^{N/2-1} |X(k)|^2 \right\} \cdot \frac{f_s}{N} \; .$$

2.3 RMS

Der quadratische Mittelwert (engl.: Root Mean Square) ist eine Methode der Mittelung, bei der größere Werte einer Zahlenreihe einen stärkeren Einfluss als die kleineren haben. Er steht für die mittlere Signalleistung und hängt damit weitgehend mit der empfundenen Lautstärke zusammen:

$$RMS = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (x[n])^2} \; .$$

Der RMS-Wert wird nicht zum Vergleich herangezogen, sondern als Maß für einen *Gain Factor* verwendet, der bei der Resynthese einen „passenden, aber zu leisen/lauten“ Frame entsprechend verstärkt bzw. dämpft.

2.4 Mel-Frequency Cepstrum Coefficients

In der Sprachsignalverarbeitung haben sich die MFCCs als sehr effiziente Methode zur Spracherkennung erwiesen. Sie repräsentieren das Spektrum in einer sehr kompakten Form, wobei die Auflösung variabel ist und mit der Zahl der verwendeten Koeffizienten steigt[1].

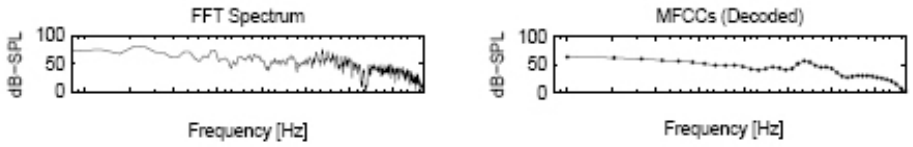

Abbildung 2.1: „Originales“ (FFT-) und aus den MFCCs abgeleitetes Spektrum

Die Berechnung der Koeffizienten erfolgt nach diesem Schema: Das diskrete Betragsspektrum eines jeden Frames wird durch eine Filterbank geschickt, die es an den wichtigsten Stützstellen in Frequenzgruppen zusammenfasst. Diese sind nicht gleichmäßig verteilt, sondern werden „wahrnehmungsangepasst“ anhand der psychoakustischen Tonheits-Skala positioniert.

Anschließend werden die Werte über die Frequenzgruppen aufsummiert und logarithmiert, was dem Verhalten der Cochlea im Innenohr des Menschen entspricht: sie integriert die Energie über die komplette Frequenzgruppe, die Nervenimpulse werden gemeinsam ausgewertet. Schließlich werden die erhaltenen Werte mit der Diskreten Cosinus-Transformation (DCT) in den Cepstralbereich transformiert.

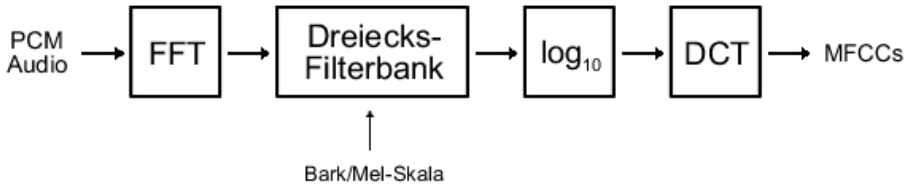

Abbildung 2.2: MFCC-Berechnung

Um auch die Eigenheiten des menschlichen Ohres bezüglich der Lautstärkeempfindung miteinzubeziehen, wird bei der MFCC-Analyse eine dB(A)-Bewertung durchgeführt. Diese hat vor allem auf die tiefen Frequenzen einen starken Einfluss; so wird in unserem Falle der Bereich um 80 Hz (Mittenfrequenz des ersten Filters) um gut 25 dB abgeschwächt!

2.5 Weitere Möglichkeiten

Im Laufe dieser Projektarbeit wurde mit weiteren Features „experimentiert“, wobei sich herausstellte, dass manche Kombinationen aufgrund einer hohen Korrelation der Ergebnisse wenig Sinn ergeben.

[1] In diesem Fall werden 20 MFCCs bei einer Analyse-Samplerate von 11,025 kHz verwendet.

Darüber hinaus hätte sich bei der Verwendung mehrerer Features für das Distanzmaß die Frage nach der Gewichtung der einzelnen Komponenten gestellt; eine detaillierte Analyse hätte jedoch den Rahmen dieser Arbeit gesprengt.

Der Vollständigkeit halber werden diese Features an dieser Stelle noch einmal aufgelistet:

2.5.1 Zero-Crossing Rate

Die Anzahl der Nulldurchgänge im Signal kann Aufschluss darüber geben, ob es sich eher um einen „Ton“ oder um ein „Geräusch“ handelt. Bei monophonen Signalen kann man auf diese Weise sogar die Tonhöhe grob bestimmen (daher die Korrelation mit dem Pitch). Die Zero-Crossings werden folgendermaßen berechnet:

$$ZC = \frac{1}{2} \sum |sgn(x[n]) - sgn(x[n-1])| \qquad \text{mit } sgn(x) = \begin{cases} -1 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases}.$$

Nachdem die Länge des analysierten Abschnitts variiert, muss der errechnete Wert anschließend normiert werden – man erhält ein Verhältnis der Nulldurchgänge pro Zeiteinheit.

2.5.2 Spectral Centroid

Durch Berechnung des spektralen Schwerpunkts (Zentrum der spektralen Energieverteilung) erhält man ein Maß für die „Fülle“ eines Klangs. Je höher dieser Schwerpunkt liegt, desto präsenter sind die oberen Partialtöne, d.h. der Klang wird brillianter. Die zugehörige Formel

$$SC = \frac{\sum_{k=0}^{N/2} k \cdot |X(k)|}{\sum_{k=0}^{N/2} |X(k)|} \cdot \frac{f_s}{N}$$

liefert einen Wert, der allerdings nur dann aussagekräftig ist, wenn man ihn auf die zugehörige Grundfrequenz bezieht. Der Zentroid korreliert in vielen Fällen mit dem Rolloff-Faktor, als natürlich auch mit dem Pitch.

2.5.3 Harmonicity

Als Maß für die „Harmonizität“ eines Klangs wählt man die Amplitude des zweithöchsten Peaks der normierten Autokorrelationsfunktion des Audiosignals.

$$H = max \left\{ \frac{ACF(k)}{ACF(0)} \right\}$$

3 Implementation: FFL-Methode

3.1 Analyse und MFCC-Berechnung

Der eingelesene Song muss zuerst ein wenig Preprocessing über sich ergehen lassen: das Stereo-File wird (ohne Rücksicht auf eventuelle Probleme mit der Mono-Kompatibilität) in Mono umgewandelt, anschließend mit einem FIR-Tiefpass gefiltert und unterabgetastet[1].

Es folgen ein paar grundlegende Berechnungen, beginnend mit der Anzahl an Frames, in die das aktuelle Musikstück „gechopped“ wird. Zu jedem Frame muss man am Anfang und am Ende den Overlap addieren, wobei dieser überlappende Bereich natürlich nicht doppelt gezählt werden darf. Nachdem die Länge des Songs in den seltensten Fällen ein ganzzahliges Vielfaches der (Frame+Overlap)-Länge sein wird, wird der letzte Frame einfach ignoriert – wenn er nicht ohnehin nur aus Nullen besteht, ist er zumindest nicht komplett und somit wertlos.

Fade-In und Fade-Out werden mit $\sin^2$-/$\cos^2$-Funktionen realisiert, da so beim Überlappen der Frames kein Energieverlust auftritt oder gar übersteuert wird. Ein Zero-Padding am Anfang des Songs verhindert, dass der erste Fade-In Schaden anrichtet.

Nun geht es an die Erstellung einer linearen Frequenzskala, mit deren Hilfe anschließend die Bark/Mel-Skala berechnet wird, auf der die Dreiecksfilter positioniert werden. Für die Umrechnung von Hertz in Bark und Mel wird in [6] folgende Näherungsformel angegeben:

$$\hat{f}_{mel} = 2595 \cdot \log_{10}\left(1 + \frac{f_{lin}}{700}\right) \ .$$

Anhand dieser Skala, die bis ca. 1 kHz linear und darüber logarithmisch verläuft, werden nun gleichverteilt (in diesem Fall 20) Dreiecksfilter positioniert. In der *Auditory Toolbox* von Michael Slaney, dessen Ansatz hier verfolgt wurde, werden die Dreiecke nach folgendem Muster designed (siehe [6]):

$$H_i(k) = \begin{cases} 0 & \text{für} \quad k < f_{b_{i-1}} \\ \frac{2\left(k - f_{b_{i-1}}\right)}{\left(f_{b_i} - f_{b_{i-1}}\right)\left(f_{b_{i+1}} - f_{b_{i-1}}\right)} & \text{für} \quad f_{b_{i-1}} \le k \le f_{b_i} \\ \frac{2\left(f_{b_{i+1}} - k\right)}{\left(f_{b_{i+1}} - f_{b_i}\right)\left(f_{b_{i+1}} - f_{b_{i-1}}\right)} & \text{für} \quad f_{b_i} \le k \le f_{b_{i+1}} \\ 0 & \text{für} \quad k > f_{b_{i+1}} \end{cases} ,$$

wobei $i = 1, 2, \ldots, M$ die Ordnungszahl des jeweiligen Filters kennzeichnet, f_{b_i} die Mittenfrequenz des i-ten Filters und $k = 1, 2, \ldots, N$ den k-ten DFT-Koeffizienten darstellt.

Die von den Dreiecken eingeschlossene Fläche ist jeweils gleich groß, ein Dreieck reicht jeweils von der Mittenfrequenz seines linken bis zu jener seines rechten Nachbarn. Die Spektren der Filter werden anschließend jeweils aufsummiert und logarithmiert.

[1] die Funktion `decimate()` aus der *Signal Processing Toolbox* stellt eine bequeme Möglichkeit dar, Filtering und Downsampling in einem Schritt zu erledigen

Im nächsten Schritt wird die Filterbank zusammengesetzt. Der Term $2 \cdot (f_{b_{i+1}} - f_{b_{i-1}})^{-1}$ bestimmt die „Höhe“ eines jeden Filters und sorgt dafür, dass die Fläche unter den Dreiecken stets gleich bleibt; aus Gründen der Einfachheit wird diese Berechnung aus der Schleife herausgezogen. Das Ergebnis ist eine Matrix, die für jedes Filter eine Zeile mit den zugehörigen FFT-Koeffizienten beinhaltet (die Zeilensummen sind jeweils identisch).

Bei der Berechnung der Filterbank wird zusätzlich eine dB(A)-Bewertung durchgeführt. Für jeden FFT-Koeffizienten wird die zugehörige Dämpfung mit der Formel

$$A_{dB(A)}(f) = \frac{12200^2 \cdot f^4}{(f^2 + 20,6^2) \cdot (f^2 + 12200^2) \cdot \sqrt{f^2 + 107,7^2} \cdot \sqrt{f^2 + 737,9^2}}$$

ermittelt und an passender Stelle in einen Vektor geschrieben, der im darauffolgenden Schritt die Filtergewichte „manipuliert“. Das Ergebnis wird anschließend noch normalisiert.

Die eigentliche Berechnung (FFT, Filterbank, Logarithmus, DCT) ist nun nicht mehr besonders problematisch und in drei, vier Zeilen erledigt. Die Structure `pool` bildet die Musikbibliothek, in der jede Zeile den „Feature-Vektor“ eines Frames repräsentiert. Er enthält neben den MFCCs Informationen zu Song- und Frame-Nr. sowie die Anzahl der „rechten Nachbarn“, wenn der Frame Teil einer Gruppe ist.

→ `xtract_ffl.m`

Anmerkungen zur Vorgangsweise

Da die Berechnung der MFCCs in der *MA Toolbox* [4] bereits als m-File verfügbar war, wurde diese Implementation genauer unter die Lupe genommen und diente zuerst als Basis für den vorliegenden Code. Die Erstellung der Tonheits-Skala sowie die Berechnung der Dreiecksfilter wurden im Großen und Ganzen übernommen, wogegen beispielsweise die DCT-Berechnung „per Hand“ durch die Matlab-eigene Funktion `dct()` ersetzt wurde, nachdem kein Vorteil erkennbar war.

Ebenfalls unverständlich war, weshalb in der MA Toolbox die Filter von 20 bis 16000 Hertz „fix“ positioniert werden, um bei $f_s < 32$ kHz anschließend festzustellen, dass man sich ab der halben Samplingfrequenz eine gewisse Anzahl an Filtern eigentlich sparen kann und sie ignoriert, so dass effektiv nur mit z.B. 13 MFCCs gerechnet wird. Der präsentierte Algorithmus verwendet alle 20 Dreiecke im Frequenzband bis 5500 Hertz.

3.2 Peak Detection und Grouping

Mittels diskreter Kurzzeit-Fouriertransformation wird ein Spektogramm erstellt, welches das Spektrum eines jeden Frames enthält. Dazu wird das Eingangssignal blockweise in eine Matrix geschrieben, die dann komplett fouriertransformiert wird. Nach Extraktion der positiven FFT-Koeffizienten werden Energie und *Spektrale Differenz* ermittelt, mit Hilfe eines Medianwerts als zeitvariantem Threshold kann man schließlich die Peaks detektieren.

→ `xtract_ffl.m`

Diese Vorgangsweise wies bei einem umfassenden Vergleich von Methoden zur Onset Detection [7] eine sehr gute Performance bei niedrigster Detektion sogenannter *False Positives* auf.

Nun sind zwar diejenigen Frames, die einen Peak enthalten, dementsprechend markiert (*neighbour* = 1) – was jedoch über die Zusammengehörigkeit der Frames zwischen zwei solchen Markern noch nichts aussagt. Daher wird im folgenden Schritt die Größe der „Lücke“ zwischen zwei Peaks ermittelt und für jeden Frame in dieser Region die Abweichung seiner MFCCs von denen des ersten Frames mithilfe von Mittelwert und Varianz berechnet (siehe „Anmerkungen“).

Die normierten Abstandswerte müssen nun unterhalb einer gewissen Hürde bleiben; der erste Wert oberhalb des Thresholds markiert denjenigen Frame, der „gerade nicht mehr“ dazugehört.

→ `group_lib.m`, `group_tar.m`

Anmerkungen zur Vorgangsweise

Nachdem eine erste Version mit fester Framelänge einmal implementiert war und prinzipiell funktionierte, sollte die Bildung von zusammenhängenden Gruppen, die „gemeinsam ersetzt“ werden, die Resynthesequalität weiter steigern. Für die Peak Detection wurde zuvor der Ansatz verfolgt, den Verlauf des 1. MFCC über die Zeit/Frames zu protokollieren und anschließend lokale Maxima zu detektieren. Dies erwies sich jedoch leider als wenig zuverlässig. (Für die TSS-Methode wurde – nach neueren Erkenntnissen – schließlich noch ein weiterer Ansatz implementiert, siehe Kap. 4.1.)

Das folgende Grouping lässt auch in der vorliegenden Version noch sehr zu wünschen übrig; es hing und hängt immer an der Korrelationsberechnung mit anschließendem Thresholding. Weder der Abstand der zugehörigen MFCC-Vektoren (Euklidsche Norm), noch Kovarianz und Korrelationskoeffizienten führten zu einem zufriedenstellenden Ergebnis. Die vorliegende Methode war Stand der Dinge zum Zeitpunkt der Entscheidung, für „beatorientierte“ Signale einen komplett anderen Weg zu gehen – was schließlich in der TSS-Methode mündete.

Aus diesem Grund lässt sich die Peak Detection im „Kontrollalgorithmus“ `run_ffl.m` durch Setzen einer einzigen Variable an- und ausschalten, standardmäßig ist sie nicht aktiviert.

3.3 Clustering

Zu Beginn wird Anzahl der Cluster und Sub-Cluster sowie die der darin enthaltenen Elemente festgelegt. Für die Sub-Cluster ist der erhaltene Wert ein Mittelwert, der im rechentechnisch gesehen idealen Fall der Gleichverteilung gilt und somit ein Mindestmaß für die dritte Dimension des Cell-Arrays darstellt.

Anschließend werden die zweiten MFCCs komplett in einen Vektor geschrieben, dessen Elemente dann in aufsteigender Reihenfolge sortiert werden. Der obere Grenzwert eines jeden Clusters wird nun ermittelt, indem abhänging von der Zahl der Elemente der jeweils x-te Wert in einem weiteren Vektor gespeichert wird. Dieser Vektor enthält ein Element weniger als Cluster vorhanden sind; der letzte Cluster beinhaltet die Feature-Vektoren, deren 2. Koeffizient über dem letzten Wert dieses neues Vektors liegt.

Nun wird zuerst „grob“ sortiert: für jeden Feature-Vektor werden diejenigen Cluster gesucht, deren oberer Grenzwert über dem aktuellen 2. MFCC liegt. Der erste dieser Cluster ist der passende; sollte kein Cluster gefunden worden sein, kann es sich nur um den letzten Cluster handeln (für den keine obere Grenze definiert worden ist). Ein Indexvektor hilft bei der Positionierung der neuen Elemente im Cell-Array.

Danach beginnt das komplette Procedere noch einmal von vorne: diesmal werden die Grenzen für die Sub-Cluster (anhand des 3. MFCC) ermittelt, bevor schließlich „endgültig“ einsortiert wird.

→ `cluster_ffl.m`

Anmerkungen zur Vorgangsweise

Da einige wenige Cepstrum-Koeffizienten das Spektrum eines Audiosignals bereits grob approximieren, erschien es direkt auf den ersten Blick sinnvoll zu sein, eine Heuristik anhand der ersten paar MFCCs zu erstellen. Da Unterschiede in der mittleren Energie zweier Spektren nicht in das Vergleichsmaß einfließen sollen, wurde auf den 1. MFCC verzichtet.

Auch das Clustering lässt sich an- und ausschalten, um einen A/B-Vergleich zu ermöglichen.

3.4 Zuweisung des „passenden“ Sub-Clusters

Nachdem das zu resynthetisierende Stück ebenfalls analysiert worden ist, wird jedem einzelnen Frame ein bestimmter Sub-Cluster zugeordnet, in dem sich der „ähnlichste“ Frame aus der Bibliothek befindet. Ist dieser Sub-Cluster leer, wird der nächste nicht-inhaltslose ausgewählt.

Das Handling leerer Sub-Cluster läuft wie folgt: zunächst werden die ersten Elemente der Sub-Cluster des aktuellen Clusters extrahiert und in einem Vektor gespeichert (mit Matrizen lässt sich wesentlich bequemer rechnen als mit *Structures*). Dieser Vektor wird nun an der Stelle des leeren Sub-Clusters so geteilt, dass ein Vektor mit den Elementen unterhalb sowie einer mit den Elementen oberhalb dieses Sub-Clusters entsteht – ersterer wird horizontal gespiegelt, um eine einheitliche Zählrichtung zu erhalten.

Anschließend wird in jedem Vektor das erste Element ungleich Null (bzw. `[]`) gesucht und dessen Position festgehalten; dieser Wert gibt an, wie weit der „nächste volle“ Sub-Cluster vom aktuellen, leeren entfernt ist. Ist einer der beiden Vektoren ohne Inhalt, erübrigt sich die ansonsten anstehende Entscheidungsfindung mittels Vergleich.

→ `assign_clusters_ffl.m`

3.5 Frame-Auswahl

Die jeweils ausgewählten Sub-Cluster werden nun auf den „ähnlichsten“ Frame hin durchsucht. Zusätzlich zur Position (d.h. Song- und Frame-Nr.) werden noch die ermittelte Distanz sowie der Gain Factor gespeichert, um den der Frame bei der Resynthese angehoben bzw. abgesenkt werden kann, um eine noch größere hörbare Übereinstimmung zu erzielen. Um das *Group Re-Synthesizing* zu ermöglichen, wird zusätzlich die Anzahl der verbleibenden Elemente pro Gruppe gespeichert.

Nach den üblichen Laufvariablen-Ermittlungen und Initialisierungen wird in einer Schleife für jeden Frame des Target Songs zunächst einmal das Distanzmaß auf einen schwerlich erreichbaren Wert gesetzt, bevor anschließend alle MFCC-Vektoren im entsprechenden Sub-Cluster auf ihre Ähnlichkeit mit dem aktuellen im Target Song überprüft werden. Dies geschieht ganz einfach mittels des Euklidschen Abstands $dist(x, y) = \|x - y\|^2$. Ist diese Distanz nun geringer als die kleinste bisher gefundene, werden die zugehörigen Werte übernommen. Das Ganze läuft so

lange, bis der Algorithmus auf ein leeres Element stößt (soll heißen: das Ende des Sub-Clusters erreicht ist).

→ `compare_mfcc1_ffl.m`, `compare_mfcc2_ffl.m`

3.6 Resynthese

Zu guter Letzt geht es darum, die jeweiligen Frames aus den Musikstücken in der Bibliothek zu kopieren, zu fenstern und zu einem Song zusammenzusetzen. Dazu müssen natürlich Anzahl und Größe der Frames bekannt sein, die übergeben bzw. aus der Länge der Tabelle `synth_stats` ermittelt werden.

Aus der Nummer des Stückes und des Frames ergeben sich mit den oben ermittelten Werten die Grenzen (d.h. die Stellen, an denen „geschnitten“ wird). Die Frames werden jeweils mit dem bereits bekannten $\cos^2$-Fenster sowie dem Gain Factor gewichtet und anschließend überlappend zusammengefügt.

Nachdem es Gruppen von Frames gibt, die zusammenhängend extrahiert werden sollen, muss bei der Resynthese eine Fallunterscheidung stattfinden. Zunächst wird anhand der Information aus der Zuordnungstabelle geprüft, ob der aktuelle Frame im Target Song Teil einer solchen Gruppe ist. Ist dies der Fall, ist zu unterscheiden, ob das zur Verfügung stehende Audiomaterial „lang genug“ ist oder weiteres Processing verlangt.

Der erste Fall ist jener, bei dem die Gruppengröße der Frames aus der Bibliothek nicht ausreicht. Hier werden zwei Songparts extrahiert: zum einen die komplette Gruppe (die eventuell mehrmals *gelooped* werden muss), zum anderen den Teil der Gruppe, der bei nicht-ganzzahligem Verhältnis der Gruppengrößen „übrig bleibt“. Diese Parts werden im Anschluss sinnvoll zusammengefügt.

Im zweiten Falle ist das Angebot (in der Bibliothek) größer als die Nachfrage bzw. stimmt exakt mit dieser überein – hier wird einfach ein Stück der Länge `needed_size` extrahiert, der Rest funktioniert analog. Anschließend wird wie oben beschrieben gewichtet, gefenstert und die Gruppe an das bestehende Output-File (anfangs eine leere Matrix) angehängt. Im „Standardfall“ wird lediglich ein einzelner Frame extrahiert.

Zu guter Letzt wird das Ergebnis normalisiert und im Wave-Format auf die Festplatte geschrieben.

→ `resynth_ffl.m`

4 Implementation: TSS-Methode

4.1 Unterteilung in transiente und stabile Regionen

Die Audiodateien werden wieder in das Monoformat konvertiert und auf 11,025 kHz heruntergerechnet, zusätzlich erfolgt ein kurzes Zero-Padding am Anfang, um auch jene Transienten korrekt identifizieren zu können, die direkt mit dem ersten Sample beginnen.

Die Transienten-Erkennung erfolgt durch Berechnung des *Normalized Spectral Flux* $\Phi[n]$ nach einer Methode, die in [2] vorgestellt wurde: das Signal wird in sich überlappende Frames zerlegt, für die jeweils das Betragsspektrum per FFT ermittelt wird. Die Differenz zweier benachbarter diskreter Spektren $\Delta[n,k]$ ergibt einen Vektor, dessen Absolutbetrag nichtlinear gewichtet und aufsummiert wird, um den *Spectral Flux* $\rho[n]$ zu erhalten:

$$\rho[n] = \sum_k |\Delta[n,k]|^{\frac{1}{2}} \quad .$$

Dieser spektrale Fluss wird anschließend mit einem Wert β_n normalisiert, der einen *Peak Follower* darstellt: solange sich $\rho[n]$ erhöht, wird der Wert einfach übernommen; sinkt er betragsmäßig, so wird stattdessen der vorherige Wert β_{n-1} – leicht abgeschwächt – übernommen. Das Ganze resultiert im normalisierten spektralen Fluss $\Phi[n]$:

$$\Phi[n] = \frac{\rho[n]}{\beta_n} \qquad \text{mit } \beta_n = \begin{cases} \rho[n] & \text{für } \rho[n] > \beta_{n-1} \\ \gamma\beta_{n-1} & \text{sonst} \end{cases} \quad ,$$

wobei $(0 \ll \gamma < 1)$.

Um nun die transienten Vorgänge von gelegentlich auftretenden lokalen Maxima unterscheiden zu können, wird in dem erwähnten Paper eine „parametrisierte, abgestufte Antwort“

$$\alpha[n] = G(\Phi) = \frac{\alpha_0 + 1}{2} + \frac{\alpha_0 - 1}{2} \cdot \tanh\Big(\pi\lambda(\Phi[n] - \Phi_0)\Big)$$

auf den normalisierten spektralen Fluss eingeführt, mit deren Hilfe man die Transienten „tunen“ (also verstärken oder dämpfen) kann, indem man die Parameter α_0 (*Range*), λ (*Slope*) und Φ_0 (*Threshold*) manipuliert – in unserem Fall wird natürlich verstärkt.

Nun geht es darum, aus dem modifizierten Eingangssignal die Peaks zu extrahieren. Nachdem diese *Peak-Picking Function* ja bereits bestmöglich aufbereitet worden ist, kann man hier mit einem festen Threshold arbeiten. Mehrfach-Peaks werden anschließend eliminiert, wobei vereinfachend davon ausgegangen wird, dass ein transientes Ereignis mindestens 256 Samples (also die typischen 23 ms) dauert. Sind die transienten Regionen gefunden, wird eine Tabelle mit den Start- und End-Indizes für „Transients“ und „Stables“ erstellt (bezogen auf die Analyse-Abtastrate von 11,025 kHz).

→ `xtract_tss.m`

4.2 Feature-Extraktion

Für jeden einzelnen Frame des aktuellen Songs wird die stabile Region ausgeschnitten und gefenstert. Ist das Signal kürzer als 1024 Punkte, wird es per Zero-Padding auf eben diese Länge gebracht, um *Dimension Mismatches* zu vermeiden (bei der anschließenden Berechnung des RMS-Wertes werden die angefügten Nullen natürlich ignoriert!).

Der spektrale Rolloff-Wert wird nach der in 2.2 angegebenen Formel ermittelt, die Berechnung der Grundfrequenz erfolgt mithilfe der ohne Änderung aus [1] übernommenen Funktion `find_pitch_fft()`. Diese liefert einen Vektor mit gefundenen „Kandidaten", aus deren Kreis die niedrigste Zahl, die mit (ungefähr) doppeltem Wert noch einmal vorkommt, als Grundfrequenz ausgewählt wird:

Die MFCC-Extraktion erfolgt komplett analog zu oben (siehe FFL-Methode, Kap. 3.1).

Zu guter Letzt werden alle Informationen zum aktuellen Frame in die Bibliothek geschrieben. Dabei handelt es sich um die Song-Nr., die Start- und End-Indizes (umgerechnet für 44,1 kHz) sowie die extrahierten Features.

→ `xtract_tss.m`, `select_pitch.m`, `get_mfccs.m`

Anmerkungen zur Vorgangsweise

Aus Gründen der Einfachheit werden die Features nur für die stabile Region eines Frames extrahiert – der Gedanke dahinter ist die Annahme, dass diese „Sustain"-Phase viel charakteristischer für einen Klang ist als der transiente Abschnitt. Selbstverständlich ist es möglich, auch die Transienten zu analysieren und zu katalogisieren; in diesem Fall wäre lediglich die Schleife zu duplizieren und eine eigene Bibliothek mit „Attack"-Phasen zu erstellen.

Es gibt auch Frames, für die keine eindeutige Bestimmung der Grundtonhöhe möglich ist. Hier wird der *Pitch*-Wert einfach Null gesetzt, was zur Folge hat, dass diese Frames beim Sortieren im ersten Cluster landen. Dies gilt natürlich auch für entsprechende Stellen im Target Song, der denselben Analyseprozess durchläuft.

4.3 Clustering und Zuweisung

Die Algorithmen stimmen bis auf die Auswahl der „Sortierkriterien" mit denen für die FFL-Methode überein, von daher sei auf Kap. 3.3 und 3.4 verwiesen.

Eine Ausnahme gibt es bei der Frame-Zuweisung: besitzt ein Frame keine stabile Region, so gibt es auch keinen zugehörigen Fingerprint. Der Frame besteht also nur aus einer Transienten, die in diesem Fall „stehen gelassen", d.h. nicht ersetzt wird! Dies widerspricht zwar prinzipiell der Idee der Resynthese, ist aber unumgänglich, wenn man die transienten Regionen nicht gesondert analysieren, clustern und vergleichen möchte. In diesem Fall wird statt der Sample-Indizes der Wert `'NaN'` in die Tabelle eingetragen.

→ `cluster_tss.m`, `assign_clusters_tss.m`

4.4 Frame-Auswahl

Hier gibt es ein paar Unterschiede zum entsprechenden Algorithmus bei der FFL-Methode: neben dem Abfangen der rein transienten Frames (s.o.) wird die Längenanpassung der Fra-

mes bei TSS bereits beim Aufstellen der Zuordnungstabelle `synth_stats` (statt erst bei der Resynthese) durchgeführt.

Ist der Originalframe nicht länger als der ihm zugewiesene, soll letzterer nach Zugabe von 1 ms (für einen kurzen Fade-Out) abgeschnitten werden – im Extremfall bestehen diese zusätzlichen 44 Samples aus angefügten Nullen. Andernfalls soll der Frame aus der Bibliothek durch Time Scaling auf die gewünschte Länge (wieder plus 1 ms) gebracht werden.

Gespeichert werden an dieser Stelle lediglich die Start- und End-Indizes des Frames sowie entweder die Anzahl der anzufügenden Nullen (falls lang genug) oder „Scale-Faktor“ und gewünschte Länge (falls zu kurz). Letzteres dient der Kompensation von eventuell auftretenden Rundungsfehlern beim Stretching.

→ `compare_features_tss.m`

4.5 Resynthese

Auch bei der Resynthese muss zuerst überprüft werden, ob der aktuelle Frame eventuell „stehen bleiben“ soll; in diesem Fall wird die entsprechende Stelle aus dem Target Song ausgelesen, die Start- und End-Indizes wurden bei der Frame-Auswahl dementsprechend angepasst.

Bei der Pegelanpassung muss man die Tatsache berücksichtigen, dass bei der TSS-Methode nur die stabile Region analysiert, anhand der ermittelten Parameter jedoch der komplette Frame (und damit auch die transiente Region) manipuliert wird. Um also die Transienten beim Absenken des Pegels nicht zu „verwischen“ oder – weit schlimmer – beim Anheben übermäßig zu verstärken (was beim späteren Normalisieren des gesamten Files ziemlichen Schaden anrichtet), wird an dieser Stelle mit einem *Smart Gain Factor* gearbeitet, der die transiente Region weitgehend unverändert lässt.

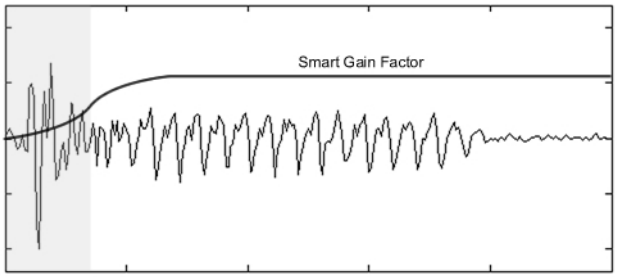

Abbildung 4.1: Smart Gain Factor

Dazu werden Koordinaten festgelegt, durch die eine „Gain-Kurve“ mit folgenden Eigenschaften gehen soll: beim ersten Sample soll die Verstärkung in jedem Fall Eins betragen, am Ende der transienten Region den halben und schließlich den vollen Verstärkungs-/Dämpfungsgrad erreichen. Da MATLAB zwischen diesen Punkten interpoliert, müssen deren vier definiert werden, um zu verhindern, dass die Kurve erst am Ende des Frames den Maximalwert erreicht.

Nun wird der Frame je nach Bedarf gekürzt oder verlängert sowie ein- und ausgeblendet (was per Multiplikation der ersten und letzten 44 Samples mit jeweils einem halben *Hanning-*

Fenster vonstatten geht), bevor er mit 1 ms Overlap an das Fragment des resynthetisierten Stückes angefügt wird.

→ `resynth_tss.m`

4.5.1 Time Scaling

Das zeitliche Verlängern eines Frames geschieht nach der in [1] vorgestellten *Synchronous Overlap-and-Add*-Methode von Roucos und Wilgus. Der Algorithmus basiert auf der im Buch vorgestellten MATLAB-Funktion `TimeScaleSOLA.m`, wobei diese nur für Mono-Signale funktioniert und dementspechend angepasst werden musste.

Bei der SOLA-Methode wird das Signal in überlappende Blöcke gleicher Größe unterteilt, die abhängig vom Scaling-Faktor verschoben werden; d.h. die Länge des überlappenden Bereiches ändert sich. Um das Ganze möglichst unhörbar zu gestalten, werden die beiden Blöcke im Overlap-Bereich miteinander kreuzkorreliert, um den Punkt der größten Übereinstimmung zu finden. Dementsprechend wird die Verschiebung zueinander noch einmal korrigiert – wodurch sich natürlich die Gesamtlänge des „gestretchten“ Signals noch einmal ändert, weshalb ein nachträglicher Vergleich mit der Länge des Target-Song-Frames erforderlich ist.

→ `time_scale.m`

Anmerkungen zur Vorgangsweise

Um ein Audiosignal zu verlängern, gibt es prinzipiell zwei Möglichkeiten: „loopen“ oder „stretchen“. Die Wahl fiel auf Zweitere, da das simple Wiederholen der kompletten Region insbesondere bei abklingenden Tönen die Erwartungen an das resultierende Klangereignis nicht erfüllt und komplexere Looping-Methoden gerade bei variablen Framelängen leider auch kompliziert zu implementieren sind.

5 Ergebnis

Ziel dieser Projektarbeit war es, einen theoretischen Ansatz zur Resynthese von Audiosignalen aus fremdem Audiomaterial auf seine Machbarkeit hin zu überprüfen. Es wurden zwei Methoden implementiert, die sich in erster Linie bezüglich der Analyse voneinander unterscheiden: während bei der FFL-Methode für (relativ kurze) Frames fester Länge die MFCCs berechnet werden und überlappend zusammengesetzt wird, basiert die TSS-Methode auf einer stabilen Peak-Erkennung, die variable Framelängen und nichtüberlappendes Zusammensetzen ermöglicht.

Beide Implementationen funktionieren nach folgendem Schema:

- Analyse der Musikstücke, Unterteilung in Frames und Extraktion der Features für jeden Frame
- Aufbau einer Listenstruktur, die alle Frames in der Reihenfolge des Einlesens enthält
- optionales Speichern dieser Struktur
- Sortieren der Frames in Cluster und Sub-Cluster anhand ihrer Features
- Analyse des zu resynthetisierenden Stückes („Target Song“)
- Zuweisen des entsprechenden Sub-Clusters für jedem Frame des Target Songs
- Suche des „ähnlichsten“ Frames im zugewiesenen Sub-Cluster
- evtl. Pegelanpassung und Längenänderung, Resynthese

5.1 Fazit: FFL-Methode

Die Vorgangsweise mit fester Framelänge hat den Vorteil, dass sie keine speziellen Anforderungen an das Audiosignal stellt. Sowohl Musikstücke mit großer Dynamik (die jeder Art von Thresholding Probleme bereiten können), als auch „wenig rhythmische“ Passagen, die keinen detektierbaren Onset[1] aufweisen, werden mehr oder weniger korrekt behandelt.

Die Performance von Peak Detection und Grouping lässt leider sehr zu wünschen übrig; zum einen arbeitet die Peak Detection mit einer effektiven Auflösung von 1 Frame (vermutlich zu grob), zum anderen wurde für die Korrelationsberechnungen zwischen den einzelnen Frames zur Bestimmung der Gruppengröße einfach keine wirklich funktionierende Methode gefunden.

[1] Bisher wurde immer nur von *Peak Detection* gesprochen. Ein „Onset“ beschreibt dagegen allgemein den Beginn eines neuen Ereignisses, wie beispielsweise eine reine Tonhöhenänderung – die ja nicht zwingend mit einer Transienten einhergehen muss.

Ein Manko der FFL-Methode ist die fehlende rhythmische Komponente. Zwar sind gewisse Klangfärbungen und Formantbereiche im resynthetisierten Stück durchaus wiederzuerkennen (das Modellieren eines Sprachsignals aus Instrumentalmusik funktioniert erstaunlicherweise auch recht gut), dem Original entspricht das Ergebnis jedoch insbesondere bei breitbandigen (Musik-)Signalen höchstens entfernt.

5.2 Fazit: TSS-Methode

Das Grundgerüst der zweiten Methode ist die Peak Detection durch Trennung von transienten und stabilen Regionen (wobei erstere dann als „Peak“ behandelt werden). Dieses Verfahren funktioniert einwandfrei, wie untenstehende Grafik und das Klangbeispiel auf der beiliegenden CD zeigen. Auch das Sortieren in Cluster und Sub-Cluster anhand von Grundtonhöhe und Energieverteilung im Spektrum (Spectral Rolloff) macht Sinn, in den Sub-Clustern finden sich durchwegs „ähnliche“ Frames.

Zur Bestimmung des Distanzmaßes werden auch bei der TSS-Methode die MFCCs herangezogen. Erstens haben sie sich bei der FFL-Methode im Prinzip bewährt, zweitens liefert kein anderes (einzelnes) Feature eine solch detaillierte und perzeptiv nachvollziehbare Repräsentation des „Klangs“. Auf die Verwendung mehrerer Features wurde verzichtet, da dazu eine umfassende Evaluierung der einzelnen Features Voraussetzung gewesen wäre – dies war im Rahmen dieser Arbeit leider nicht möglich.

Insgesamt erfüllt auch der TSS-Algorithmus in seiner derzeitigen Form die Erwartungen leider nur ansatzweise. Die rhythmische Struktur des Target Songs wird sehr gut wiedergegeben, die jeweiligen „Sustain-Phasen“ dagegen entsprechen selten in etwa dem Original.

5.3 Unter der Lupe

Performance der Transient/Stable Separation (TSS)

Zur Überprüfung der Qualität des Algorithmus liefert einem `CHECK_PDETECT` das Ergebnis auditiv wie visuell: mit der Differenzfunktion eines extrem kurzen Hanning-Fensters wird ein „Klick“ erzeugt, der mit dem Vektor mit gefundenen Peaks gefaltet und anschließend dem Originalsignal überlagert wird; zusätzlich öffnet sich ein Fenster, in welchem das PCM-Signal und die Peaks grafisch dargestellt werden:

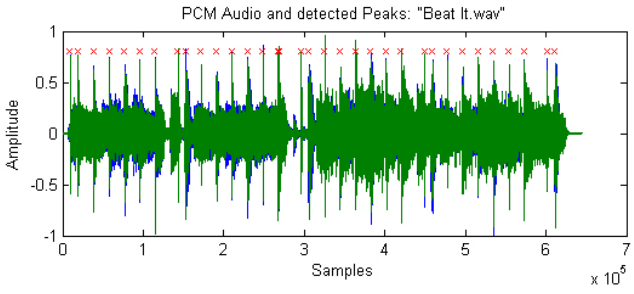

Abbildung 5.1: Audiosignal und detektierte Peaks

Performance des Clustering-Algorithmus

Die folgende Grafik (iterativ erstellt mit `CHECK_CLUST`) zeigt die gemittelten Betragsspektren der einzelnen Frames eines Sub-Clusters incl. Standardabweichungen in vier (Extrem-)Fällen für die TSS-Methode, wobei jeweils der erste und letzte Sub-Cluster im ersten und im letzten Cluster ausgewählt wurden.

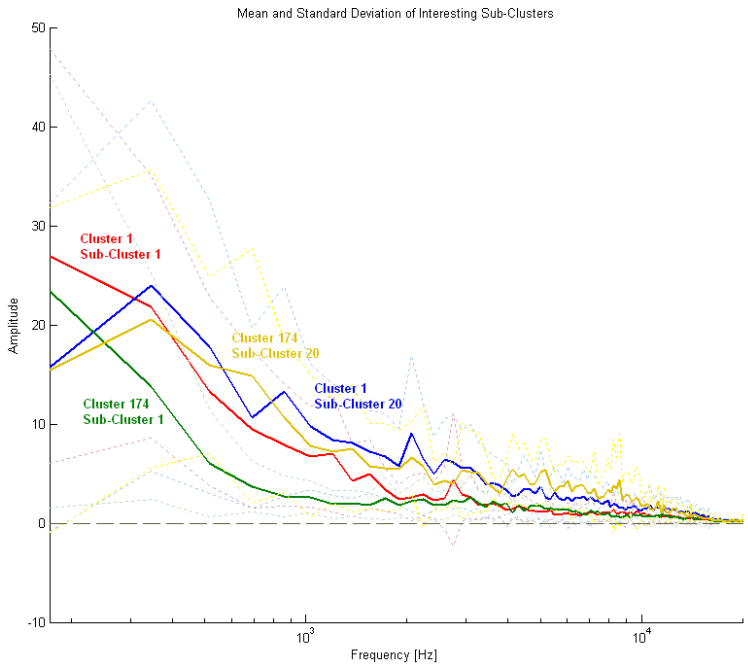

Abbildung 5.2: Spektren der Sub-Cluster (1,1), (1,max), (max,1) und (max,max)

Dabei lässt sich erkennen, dass die Frames innerhalb der Cluster anhand der spektralen Energieverteilung tendenziell richtig einsortiert worden sind (wenn man beispielsweise die rote mit der blauen und die grüne mit der gelben Kurve vergleicht), wogegen der Erfolg des Sortierens nach Pitch anhand dieser Grafik nicht bestätigt werden kann. Anzumerken ist, dass der Algorithmus durch die Analyse mit nur 11,025 kHz das Spektrum nur bis etwa 5,5 kHz „sehen“ kann!

Ein weiteres Qualitätsmerkmal der Clustering-Algorithmus ist die Verteilung der Frames über die einzelnen Sub-Cluster. Dazu erzeugt `VISU_CLUST` einen matrixähnlichen Plot, der die Größe der einzelnen Sub-Cluster farblich kennzeichnet. Aus Gründen der Übersichtlichkeit wurde der Farb-Wertebereich so gewählt, dass einige wenige Extremwerte nicht ganz maßstabsgetreu dargestellt werden, dennoch zeugt Abb. 5.3 von einer insgesamt guten gleichmäßigen Verteilung.

Um das Ganze quantitativ noch genauer erfassen zu können, mittelt `CLUST_MEAN` über die Zeilen bzw. Spalten der oben dargestellten Matrix. Der Sollwert von durchschnittlich 25 Frames pro Sub-Cluster wird jeweils ziemlich genau erreicht; dargestellt einmal über alle Cluster und einmal über alle Sub-Cluster.

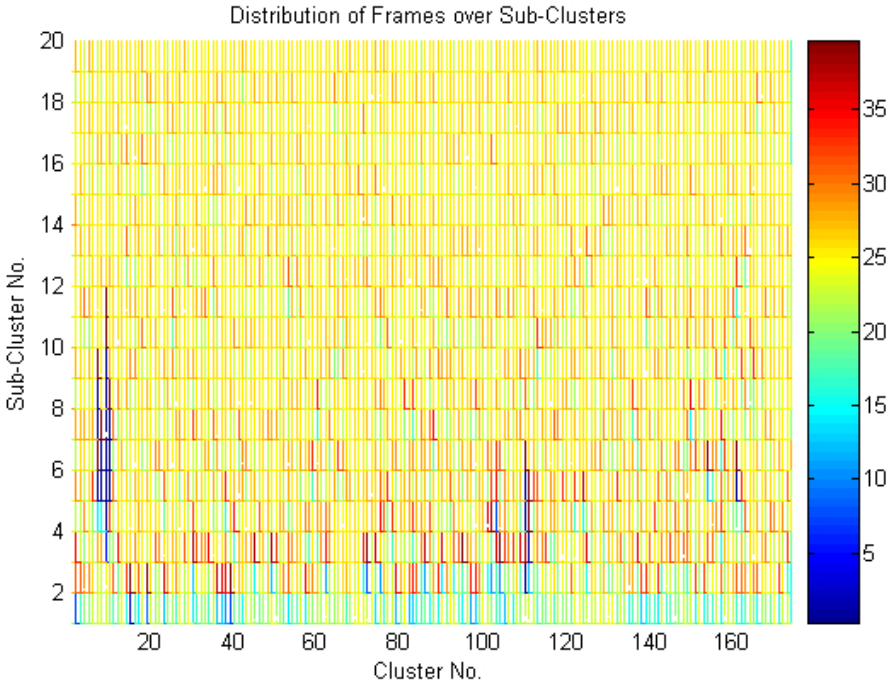

Abbildung 5.3: Verteilung der Frames über die Cluster und Sub-Cluster (TSS)

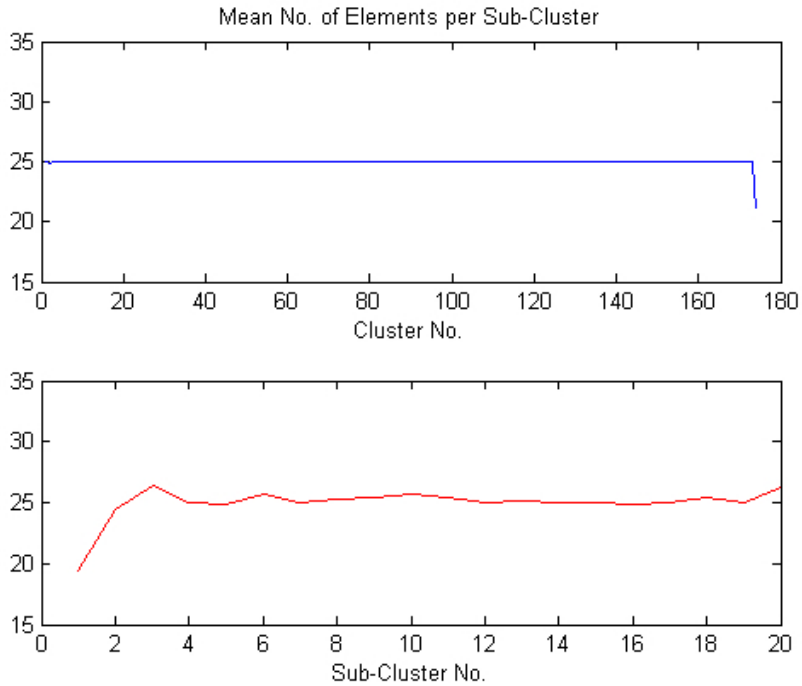

Abbildung 5.4: Mittlere Anzahl der Frames pro Sub-Cluster (TSS)

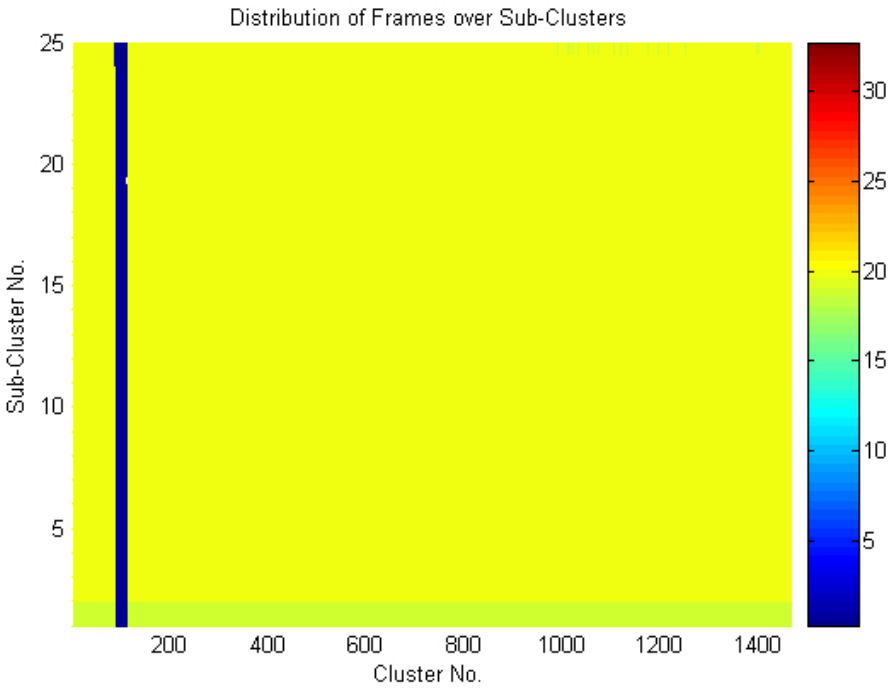

Abbildung 5.5: Verteilung der Frames über die Cluster und Sub-Cluster (FFL)

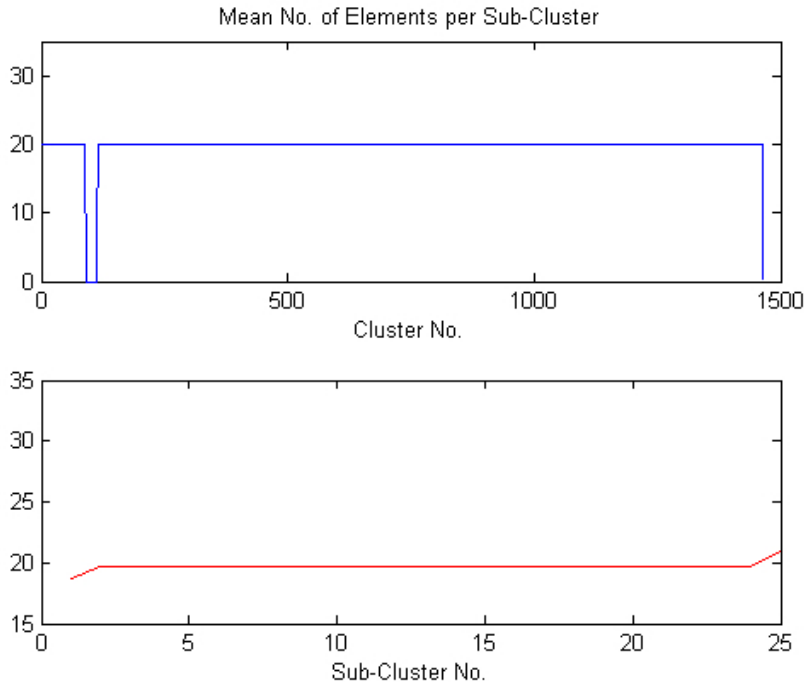

Abbildung 5.6: Mittlere Anzahl der Frames pro Sub-Cluster (FFL)

Bei den Plots für die FFL-Methode fällt auf, dass sich in den Clustern Nr. 90-123 kein einziges Element befindet, was natürlich nicht zufriedenstellend sein kann. Ansonsten sind die Frames zumindest quantitativ bestens verteilt – man könnte darüber diskutieren, ob sich angesichts der enormen Gesamtanzahl eine Erhöhung der Clustergröße auszahlen würde.

Performance des kompletten Programms

Um den Output des „Gesamtsystems“ auf einen Blick erfassen zu können, kann man mit `CHECK_ALL` einen sogenannten *Scatter Plot* erstellen lassen: in einem zweidimensionalen Koordinatensystem geben farbige Punkte mit ihrer Position Auskunft darüber, aus welchem Song der jeweilige Frame entnommen wurde, während die Farbe für die Position des entnommenen Frames (FFL: Frame-Nr., TSS: Sample-Index) im Stück steht. Zusätzlich wird in einem zweiten Plot die Einhüllende des Originalsignals der des resynthetisierten Stückes gegenübergestellt.

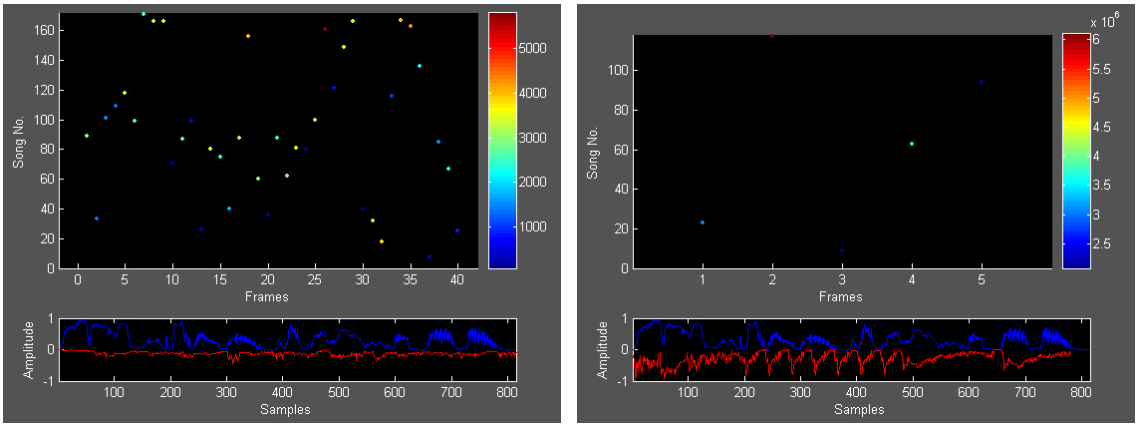

Abbildung 5.7: Performance von FFL (links) und TSS (rechts) bei der Resynthese der Anweisung eines Navigationssystems: „An der nächsten Ampel bitte links abbiegen.“

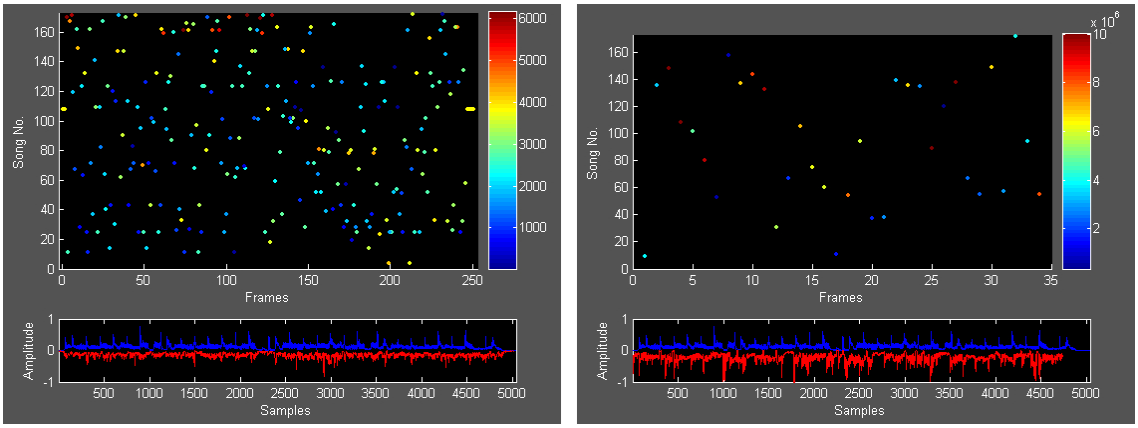

Abbildung 5.8: Performance von FFL (links) und TSS (rechts) bei der Resynthese der ersten acht Takte aus „Beat It“ von Michael Jackson

Im Normalfall lässt sich der Verteilung der Punkte wenig Information entnehmen, da sie scheinbar willkürlich im Raum verteilt sind. Ergeben sich jedoch erkennbare Muster, lässt dies Rückschlüsse auf Ähnlichkeiten zwischen den Stücken, mögliche Fehlerquellen oder zumindest Eigenarten des Programms zu.

Ein Beispiel wäre die „Positiv-Probe“, bei der ein Stück aus der Bibliothek zur Resynthese ausgewählt wird. Funktioniert alles einwandfrei, sollte der Plot eine gerade Linie zeigen, die farblich das gesamte Spektrum von blau nach rot durchläuft.

Ein weiteres Beispiel zeigt der linke Scatter Plot der zweiten Abbildung („Beat It“, FFL). Hier lässt sich feststellen, dass der Algorithmus für die „Stille“ zu Beginn und am Ende des Songs jeweils ein- und denselben Frame verwendet, dargestellt durch die kurze gelbe Linie bei (ca.) Song Nr. 110.

6 Diskussion und Ausblick

Einem Musikstück charakteristische klangliche Eigenschaften (Features) zu extrahieren, diese zu quantifizieren und einen Computer anhand dieser Informationen das Stück aus anderen Songs nachbilden zu lassen, funktioniert prinzipiell. Insbesondere ist die Tatsache hervorzuheben, dass man auch Sprachsignale mit reiner Instrumentalmusik erkennbar reproduzieren kann.

Die einzelnen Programmkomponenten (Peak Detection, Clustering, Auswahl der Features) bergen jeweils mit Sicherheit noch einiges an Optimierungspotential in sich; es war natürlich nicht möglich, im Rahmen dieser Projektarbeit alle in der Literatur vorhandenen Ansätze zu implementieren und zu testen.

Sehr zum Leidwesen des Autors lässt sich für die beiden vorgestellten Methoden jeweils nur ein Merkmal positiv hervorheben: die tatsächlich vorhandene Ähnlichkeit eines Sprachsignals mit seinem aus Musik neu zusammengesetzten Pendant bei der FFL- sowie die sehr gut erhaltene rhythmische Struktur bei der TSS-Methode. Ob eine noch größere Anzahl an Songs in der Bibliothek das Ergebnis entscheidend verbessern würde, sei dahingestellt (der nicht unbedingt schlecht ausgerüstete Test-PC[1] kam bei den verwendeten 172 Stücken jeweils an seine Grenzen).

Es läuft alles darauf hinaus, ein „komplexeres“ Distanzmaß zu finden, welches mehrere Features berücksichtigt. Voraussetzung dafür ist allerdings eine intensive Evaluierung der perzeptiven Relevanz sowie der Korrelation zwischen den einzelnen Features. Die MFCCs alleine sind mit breitbandigen Signalen wie Musik anscheinend überfordert.

Die größte Hürde auf dem Weg zur Serienreife dürfte – neben der klanglichen Qualität des Outputs – die Ausführungszeit des Analyse-Algorithmus darstellen, die wohl nicht allein durch weitere Reduktion von Samplerate und Auflösung in einen akzeptablen Bereich zu bringen sein wird. Vielmehr ist zu überlegen, ob nicht eine separate Analyse der Makrostruktur eines Songs zum Auffinden von „redundanten“ Abschnitten (wie wiederkehrenden Strophen und Refrains) vorangehen sollte, um die Größe der Bibliothek zu reduzieren. In diesem Zusammenhang könnte ein in [8] vorgestellter Ansatz zur *Timbre Model-Based Song Segmentation* interessant sein.

Ein weiterer Punkt ist natürlich die Beschränkung auf gewisse Genres bzw. „Signalklassen“, was zu Forschungs- und Demonstrationszwecken zwar Sinn macht, einen potentiellen User jedoch eher vom Kauf abhalten könnte. Hier könnte man ein neu einzulesendes Stück mit einem Standard-Set vergleichen, um es grob zu kategorisieren und daraufhin die passende Methode (FFL oder TSS) zu wählen und gewisse Parameter entsprechend zu initialisieren.

[1] Verwendet wurde ein Windows-PC mit 2,8 GHz Pentium-4-Prozessor mit 1 GB Arbeitsspeicher.

A Anhang: Verwendete Musiktitel

Die Bibliothek zum Testen des Programms wurde größtenteils aus Radio-Popmusik der späten 90er Jahre des letzten Jahrhunderts zusammengestellt. Diese Auswahl hat den Vorteil einer gewissen Nichtkomplexität bezüglich Melodik, Harmonik und Rhythmik sowie geringer Dynamik, was den Analyse-Algorithmen prinzipiell entgegenkommen sollte. (Um das Ganze etwas aufzulockern, wurden zwei weitere Alben hinzugefügt, die musikalisch anspruchsvoller, aber ebenfalls sehr „rhythmisch“ sind.)

Aus Gründen der Vergleichbarkeit wurde natürlich für beide Implementationen (FFL und TSS) dasselbe Audiomaterial verwendet.

Interpret	Album
Backstreet Boys	Backstreet Boys
Backstreet Boys	Backstreet's Back
Backstreet Boys	Millennium
Backstreet Boys	Black & Blue
Britney Spears	Baby One More Time
Britney Spears	...Oops! I Did It Again
Heavytones	No. 01
N*Sync	N*Sync
N*Sync	No Strings Attached
N*Sync	Celebrity
Spice Girls	Spice
Spice Girls	Spiceworld
Spice Girls	Forever
Tower Of Power	Soul Vaccination (Live)

Literaturverzeichnis

[1] Zölzer: **DAFX – Digital Audio Effects**
Wiley & Sons, 2002 (ISBN 978 0471 49078 4)

[2] Goodwin: **Frequency-Domain Algorithms for Audio Signal Enhancement Based on Transient Modification**
Journal of the Audio Engineering Society, Vol. 54, 9/2006

[3] Logan: **Mel Frequency Cepstral Coefficients for Music Modeling**
Proc. Int. Symposium on Music Information Retrieval (ISMIR), 2000

[4] Pampalk: **A MATLAB Toolbox to compute Music Similarity from Audio**
Proc. Int. Symposium on Music Information Retrieval (ISMIR), 2004

[5] West/Cox: **Features and Classifiers for the Automatic Classification of Musical Audio Signals**
Proc. Int. Symposium on Music Information Retrieval (ISMIR), 2004

[6] Ganchev et al: **Comparative Evaluation of Various MFCC Implementations on the Speaker Verification Task**
Proc. 10th International Conference on Speech and Computer (SPECOM), 2005

[7] Bello et al: **A Tutorial on Onset Detection in Music Signals**
IEEE Transactions on Speech and Audio Processing, Vol. 13, 9/2005

[8] Aucutourier et al: **"The Way It Sounds": Timbre Models for Analysis and Retrieval of Music Signals**
IEEE Transactions on Multimedia, Vol. 7, 12/2005

[9] Sturm: **Adaptive Concatenative Sound Synthesis and its Application to Micromontage Composition**
Computer Music Journal, Vol. 30, 04/2006

[10] Oppenheim/Schafer/Buck: **Zeitdiskrete Signalverarbeitung**
Pearson Education/Prentice Hall, 2004 (ISBN 978 3827 37077 8)