Medientechnik

Entwicklung einer Software zur automatisierten Erkennung von Sprache und Musik in den Radiobeiträgen des Cultural Broadcasting Archive

2. Bachelorarbeit

eingereicht von

Ewald Wieser
mt091110

im Rahmen des
Studiengangs Medientechnik an der Fachhochschule St. Pölten

Betreuer: FH-Prof. Dipl.-Ing. Markus Seidl

Aschbach, 3.6.2012

(Unterschrift Autor/Autorin) (Unterschrift Betreuer/Betreuerin)

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Bachelorarbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Bachelorarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

- ich hiermit der Fachhochschule St. Pölten das ausschließliche und räumlich unbeschränkte Werknutzungsrecht für alle Nutzungsarten an dieser Bachelorarbeit einräume, und behalte das mir Recht vor, als Urheber dieses Werkes genannt zu werden.

Aschbach, 3.6.2012

(Unterschrift Autor/Autorin)

Zusammenfassung

Ziel dieser Arbeit ist die Beschreibung des Entwicklungsprozesses einer Software zur Audioklassifizierung für das Cultural Broadcasting Archive. Der Arbeitsauftrag ist, die gesamte Musik in den Radiobeiträgen des CBA automatisiert zu annotieren, um diese bei Bedarf wegen möglicher Urheberrechtsprobleme löschen zu können.
Ausgangspunkt ist eine umfangreiche Literaturrecherche zu Mustererkennung und Klassifizierung im Allgemeinen, sowie zu bereits existierenden Features und Klassifikatoren zur Musikerkennung in Audiodateien. Um aussagekräftige Vergleiche anstellen zu können, wird eine repräsentative Ground Truth für das CBA erstellt. Anschließend wird das Continuous Frequency Activation-Feature in Matlab implementiert und erste Tests zur Verifizierung der Angaben der Entwickler dieses Features durchgeführt. Es folgt die Umsetzung des CFA-Features in einer Hochsprache mit Hilfe eines Audio Feature Extraction Frameworks und der Vergleich der Ergebnisse der beiden Implementierungen.
Diese Umsetzung und die ersten Erkenntnisse dienen als Basis für weitere Untersuchungen des CFA-Features anhand der Ground Truth und zum Vergleich mit anderen Audiofeatures und Klassifikatoren. Durch Verbesserungen am CFA-Feature und/oder Kombination mit anderen Features soll eine entsprechend hohe Erkennungsgenauigkeit für Musik in den Dateien des CBA erreicht werden.

Abstract

This thesis is aimed to describe the development process of a software tool for audio classification for the Cultural Broadcasting Archive. The goal is to automatically annotate all the music in the audio files of the CBA in order to delete it, if necessary, due to possible copyright issues.
The starting point is an extensive literature recherche on pattern recognition and classification in general, as well as on existing features and classifiers for music detection in audio files. For meaningful comparisons, a representative ground truth for the CBA is created. Then the Continuous Frequency Activation feature is implemented in Matlab and initial tests are done to verify the data of its developers. Consequently the CFA feature is implemented in a high-level programming language with the help of an audio feature extraction framework and the results of the two implementations are compared.
This implementation and the initial findings serve as a basis for further investigations on the CFA feature with the ground truth and for comparison with other audio features and classifiers. Further improvements on the CFA feature and/or a combination with other features should lead to a suitably high accuracy for music detection in the audio files of the CBA.

Inhalt

1 EINLEITUNG 6

2 WIE FUNKTIONIERT AUDIO-KLASSIFIZIERUNG 8

2.1 PREPROCESSING 9
2.2 FEATURE-EXTRAKTION 10
2.3 KLASSIFIZIERUNG 10
2.4 POSTPROCESSING 11

3 PROJEKT CBA 12

3.1 ERSTELLEN EINER GROUND TRUTH 12
3.2 DAS CONTINUOUS FREQUENCY ACTIVATION-FEATURE 15
3.3 IMPLEMENTIERUNG IN MATLAB 17
3.4 TOOLS ZUR FEATURE-EXTRAKTION 18
3.5 IMPLEMENTIERUNG DES CFA-FEATURES IN YAAFE 20
3.5.1 Anforderungen und Installation von Yaafe 20
3.5.2 Unterteilung in Features und Components 21
3.5.3 Aufteilung des CFA-Features in Komponenten 22
3.5.4 Umsetzung der neuen Komponenten 23
3.6 VERGLEICH MIT MATLAB-ERGEBNISSEN 27

4 SCHLUSSFOLGERUNGEN 29

5 WEITERFÜHRENDE SCHRITTE 29

5.1 VERGLEICH DES CFA-FEATURES MIT ANDEREN FEATURES 29
5.2 FERTIGSTELLEN DES SOFTWAREPAKETS 30

6 LITERATURVERZEICHNIS 31

7 ABBILDUNGSVERZEICHNIS 32

8 TABELLENVERZEICHNIS 33

9 ABKÜRZUNGSVERZEICHNIS 33

1 Einleitung

Das Cultural Broadcasting Archive (CBA) wurde 1999 gegründet und ist eine Audiodatenbank, die mittlerweile über 20.000 verschiedene Radiosendungen umfasst, die von derzeit 23 hauptsächlich österreichischen, freien Radiostationen produziert wurden. Als eine offen zugängliche Austausch- und Kommunikationsplattform unterstützt es „die wechselseitige Übernahme von Radiosendungen sowie die Publikation von und Bezugnahme auf Inhalte außerhalb der jeweiligen Reichweite der lokalen Sender“ (CBA 2012).

> Durch die langfristige, kollaborative Archivierung der medialen Produktionen [...] hat sich zudem ein kontinuierlich wachsendes Pool alternativer Medienberichterstattung herausgebildet, das neben dem privaten Gebrauch insbesondere für Recherchezwecke offen steht. (CBA 2012)

Diese archivierten Audiodateien enthalten, wie bei Radiosendungen so üblich, zwischendurch auch immer Musik, für die die Radiostationen Abgaben an die Verwertungsgesellschaft für Autoren, Künstler und Musikverleger (AKM) leisten müssen. Für die Ausstrahlung über Radiofrequenzen und einen Livestream im Internet sind die Rechte an den Musikstücken meist durch einen Pauschalvertrag abgedeckt, nicht jedoch für die Speicherung und Veröffentlichung im Internet. Daraus könnten urheberrechtliche Probleme für das CBA resultieren, deshalb soll die gesamte vorkommende Musik im Archiv markiert werden, um bei Bedarf auch gelöscht werden zu können.
Eine händische Annotierung des bestehenden CB-Archivs durch einen Menschen würde zumindest Echtzeit benötigen, da jede Datei komplett durchgehört und jedes Musikstück mit Beginn und Ende markiert werden müsste. Ebenso wird das Archiv laufend durch neue Sendungen erweitert, die auch immer wieder annotiert werden müssten. Hierzu könnte man zwar die Uploader in die Pflicht bringen, zusätzlich zum Audiomaterial die Annotierungsdaten zu liefern, das würde sich jedoch wahrscheinlich in fälschlicher oder unterschiedlicher Annotierung auswirken. Im schlimmsten Fall könnte es zur Reduktion der Anzahl an Uploads führen, wegen des zusätzlichen Mehraufwandes für die Uploader.
Eine automatisierte Musikerkennung ermöglicht eine rasche, zuverlässige und gleichzeitig vergleichsweise kostengünstige Annotierung der bestehenden Radiosendungen des CB-Archivs. Außerdem besteht die Möglichkeit der automatischen Annotierung von neuen Files direkt beim Upload. Deshalb soll in dem hier beschriebenen Projekt, welches an der Fachhochschule St. Pölten als Innovationsscheck Plus durchgeführt wird, eine Software zur automatisierten Erkennung von Sprache und Musik in den Radiobeiträgen des Cultural Broadcasting Archives entwickelt und umgesetzt werden. Es ergeben sich folgende, für das Projekt relevante Fragen:

- Wie funktioniert Audio-Klassifizierung?
- Wie kann ein einfacher aber effektiver Audio-Klassifikator für das CBA umgesetzt werden?
- Welche Schritte sind für dessen Umsetzung notwendig?

In Kapitel 2 dieser Arbeit wird basierend auf einer umfangreichen Literaturrecherche beschrieben, wie Audio-Klassifizierung funktioniert und es wird in reduziertem Umfang auf verschiedenen Audiofeatures und Klassifikatoren eingegangen. In Kapitel 3 wird die konkrete Vorgehensweise im Projekt „Cultural Broadcasting Archive“ und die Umsetzung des Continuous Frequency Activation (CFA)–Features in einem Feature-Extraction Framework näher erklärt. Ebenso werden erste Testergebnisse in Matlab analysiert und auf Plausibilität untersucht. Kapitel 4 bietet eine kurze Zusammenfassung der Ergebnisse und das Kapitel 5 beinhaltet eine Kurzbeschreibung der aus jetziger Sicht noch ausständigen Arbeitsschritte für die Fertigstellung des Projekts.

2 Wie funktioniert Audio-Klassifizierung

Die Klassifizierung von Audiodaten basiert auf dem System der Mustererkennung (*Pattern Recognition*). Das zu klassifizierende Material wird vorverarbeitet (*Preprocessing*), anschließend werden verschiedene Eigenschaften (*Features*) daraus extrahiert und aufgrund dieser Eigenschaften erfolgt anschließend die Klassifizierung (*Classification*) (vgl. Duda / Hart / Stork 2001, S.3–5).

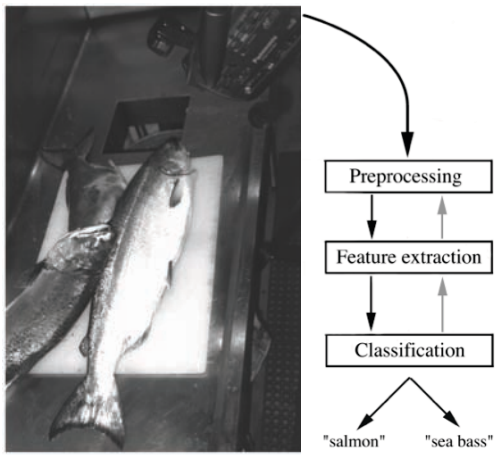


Abbildung 1: Unterscheidung von Lachs und Barschen anhand photographischer Aufnahmen.
(Duda / Hart / Stork 2001, S.5)

Im oben gezeigten Anwendungsfall, werden in einer Fischfabrik Lachse und Barsche durch photographische Aufnahmen anhand verschiedener Merkmale (Länge, Form des Körpers, Flossenform, ...) unterschieden. Die Art und Anzahl der extrahierten Features variiert je nach Art des Datenmaterials.

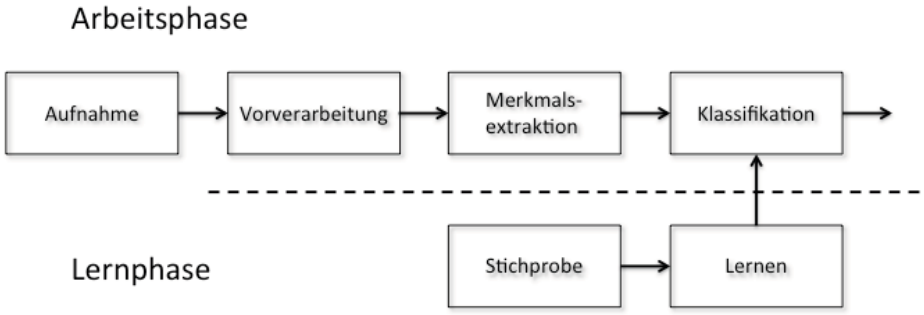


Abbildung 2: Typischer Ablauf von Mustererkennung.
(nachgezeichnet nach Niemann 2003, S.26)

Typischerweise werden solche Klassifizierungsaufgaben heutzutage von einem Machine Learning Algorithmus erledigt. Dazu wird ein Klassifikator mit einem zuvor erstellten Trainingsset angelernt, dessen Einteilung in die jeweiligen Klassen bekannt ist (siehe Abbildung 2, Seite 8). Aus diesen Trainingsdaten werden vom Klassifikator Regeln aufgestellt, um die einzelnen Klassen anhand der extrahierten Eigenschaften unterscheiden zu können. Die daraus erstellten Regeln werden dann verwendet, um in der Arbeitsphase neues, unbekanntes Material entsprechend zu klassifizieren (vgl. Bishop 2006, S.1f.).

Audio Klassifizierung, oder auch genannt Audio Segmentierung, ist die Vorverarbeitung von Audiodaten für weiterführende Audioerkennung, wie Spracherkennung (z.B.: für Diktiersoftware) und Sprechererkennung (z.B.: für die Authentifizierung bei Sicherheitssystemen), Musikerkennung (Erkennung von Instrumenten, des Musikgenres oder einzelnen Sängern) oder Geräuscherkennung (z.B.: Tiergeräusche in der Forschung oder für Überwachungsanlagen) (vgl. Mitrovic / Zeppelzauer / Breiteneder 2010, S.130).
Durch Analyse der Audiodaten wird versucht, diese in die verschiedenen Schallarten einzuteilen:

- Sprache
- Musik
- Umgebungsgeräusche
- Stille
- eine Kombination aus diesen

Die einzelnen Segmente können dann, je nach Typ, unterschiedlich weiterbehandelt und mit speziell dafür zugeschnitten Algorithmen genauer unterteilt und untersucht werden (vgl. Mitrovic / Zeppelzauer / Breiteneder 2010, S.130f.).

Bei diesem konkreten Projekt jedoch, soll nur ein Audio-Klassifikator realisiert werden, der erkennt, ob und wo im vorliegenden Audiomaterial Musik vorhanden ist. Deshalb beschränkt sich die Auswahl der zu extrahierenden Features auf solche, die speziell für die Erkennung von Musik ausgelegt sind. Auch wird fürs Erste keine weitere Verarbeitung der klassifizierten Daten angestrebt.
Für die Klassifizierung des Audiomaterial des Cultural Broadcasting Archives in die Klassen *Musik* und *Nicht-Musik* ergeben sich somit folgende Schritte:

2.1 Preprocessing

Um die Audiodaten für die Weiterverarbeitung vorzubereiten und gleichzeitig die Menge an zu klassifizierendem Material zu reduzieren, ohne wesentliche Informationen zu verlieren, werden die Dateien in das WAV-Format umgewandelt, in mono konvertiert und auf 11025 Hz Samplingrate heruntergerechnet. Außerdem werden die großen Dateien in zahlreiche kleine, oft nur wenige Millisekunden lange, gleich große, sich teilweise überlappende Frames aufgeteilt, um für die

Feature-Berechnung gleiche Bedingungen zu erlangen. Für jeden dieser Frames werden anschließend die gewünschten Features berechnet.

2.2 Feature-Extraktion

Es gibt eine Vielzahl an Features, die aus Audiomaterial extrahiert werden können und für dessen Klassifizierung verwendet werden. Mitrovic, Zeppelzauer und Breiteneder vergleichen mehr als 70 aktuelle und traditionelle Audiofeatures aus der Literatur und erstellen eine Taxonomie zur Einteilung der Features in bestimmte Klassen (vgl. Mitrovic / Zeppelzauer / Breiteneder 2010, S.99–130).
In der Literatur wurden bereits eine Vielzahl an Features für ähnliche Anwendungsfälle von Sprach-/ Musikerkennung in Radio- und Fernsehsendungen entwickelt und untersucht. Die erreichten Genauigkeiten können aber nicht direkt miteinander verglichen werden, da sie sehr stark vom verwendeten Audiomaterial abhängen.
Scheirer und Slaney untersuchen die Features *4Hz Modulationsenergie*, die *Prozentrate von „Low Energy"-Frames*, den *Spectral Rolloff Point*, das *Spectral Centroid*, das *Spectral „Flux"*, die *Zero-Crossing Rate*, die *Cepstrum Resynthesis Residual Magnitude* und die *Pulse Metric* und erreichen eine Erkennungsgenauigkeit von ca. 92% (vgl. Scheirer / Slaney 1997).
Khan und Al-Khatib verwenden ebenso die Features *Prozentrate der „Low Energy"-Frames*, *Spectral Flux* und *Linear Predictive Coefficients (LPC)*, erweitern diese Palette jedoch um einige neue Features: die *Range of Zero-Crossings (R-ZC)*, *Mittelwert und Varianz der Discrete Wavelet Transform (DWT)*, den *Quadratischen Mittelwert eines Lowpass-Signals* und die *Varianz der Mel Frequency Cepstrum Coefficients (MFCC)* und erreichen damit eine Genauigkeit von zeitweise 100% (vgl. Khan / Al-Khatib 2006).
Seyerlehner u.a. berechnen zum Vergleich die Features *Spectral Entropy (SE)*, *Chromatic Spectral Entropy (CSE)*, *Mel Frequency Cepstrum Coefficients (MFCC)*, *Linear Predictive Coefficients (LPC)* und die Ableitungen dieser Features. Zusätzlich entwickeln sie jedoch ein eigenes Feature *Continuous Frequency Activation (CFA)*, bei dem durch einen einzelnen Schwellwertvergleich die Notwendigkeit eines Klassifikators entfällt und trotzdem eine Genauigkeit von 89% verglichen mit 81% beim herkömmlichen Machine Learning Ansatz erreicht wird (vgl. Seyerlehner u. a. 2007).
Dieses neue Feature soll als Anhaltspunkt für weiterführende Forschung in diesem Projekt dienen und wird zu diesem Zwecke unter Punkt 3.5 in dem Softwaretool Yaafe umgesetzt.

2.3 Klassifizierung

Ebenso wie die große Menge an Features, gibt es eine Vielzahl an Klassifikatoren, die auf verschiedenen Grundsätzen basieren und von Duda u.a. (vgl. Duda / Hart / Stork 2001, Kap. 2-10) bzw. Bishop (vgl. Bishop 2006, S.67–676) ausführlich beschrieben werden.
In der, im vorigen Kapitel bereits erwähnten Literatur zur Unterscheidung von Sprache und Musik in Radioprogrammen, werden verschiedene Klassifikatoren mit den extrahierten Features trainiert und auf ihre Erkennungsgenauigkeit miteinander verglichen.

Scheirer und Slaney vergleichen einen *Multidimensionalen Gaussian maximum a posteriori (MAP) estimator*, ein *Gaussian Mixture Model (GMM)*, ein *Spational Partitioning Scheme based on k-d trees* und einen *Nearest-Neighbor Classifier* (vgl. Scheirer / Slaney 1997).
Khan und Al-Khatib verwenden für ihre Tests ein *Multilayer Perceptron (MLP)* und *Radial Basis Functions (RBF)*, beides Klassifikatoren vom Typ eines Künstlichen Neuronalen Netzes, sowie ein *Hidden Markov Model (HMM)* (vgl. Khan / Al-Khatib 2006).
Seyerlehner u.a. vergleichen auch den einfachen *Nearest Neighbor Classifier IBk* als Vertreter einer instanz-basierten Lernmethode, eine *Support Vector Machine (SVM)* als kernel-basierte Machine Learning Methode, ein *Multilayer Perceptron* als bekanntesten Vertreter der Neuronalen Netze Klassifizierer, sowie den *REPTree* und *RandomForest* Klassifikatoren als Entscheidungsbaum-Lerner (vgl. Seyerlehner u. a. 2007).
Grundsätzlich liefern alle diese Klassifikatoren relativ brauchbare Ergebnisse, weshalb nicht ein einzelner empfohlen werden kann. Letztendlich hängt die Klassifizierungsgenauigkeit größtenteils vom verwendeten Featureset und vom Datenmaterial ab.

2.4 Postprocessing

In der vorliegenden Literatur manchmal nur beiläufig erwähnt, aber für das Endergebnis mindestens ebenso wichtig wie die Extraktion der richtigen Features, ist eine Nachverarbeitung der klassifizierten Daten. Da die Feature-Berechnung und die Klassifikation jeweils für einen kleinen Ausschnitt des Datenmaterials (Frame) erfolgen, können diverse Störungen und Fehler in einem oder beiden Schritten eine häufige und schnelle Änderung des Ausgangssignals bedeuten.
Für die Klassifizierung von Musik in Radiobeiträgen würde sich eine Mittelwertbildung oder Tiefpassfilterung über mehrere dieser Frames anbieten, da Musikstücke im Radio in der Regel eine längere zeitliche Dauer haben. Ausnahmen können natürlich auftreten, wenn zum Beispiel Musik im Hintergrund einer Interviewsituation auftritt.

Im folgenden Kapitel wird, ausgehend von dieser Literaturrecherche, die Vorgehensweise zur Erstellung eines Audioklassifikators für das Cultural Broadcasting Archive erläutert.

3 Projekt CBA

Zur Entwicklung und Umsetzung eines Audioklassifikators für das Cultural Broadcasting Archive wurde folgende Herangehensweise gewählt (siehe Abbildung 3): Zuerst wird eine ausreichend große, repräsentative Menge an Audiodateien des Cultural Broadcasting Archives händisch annotiert und bildet die Ground Truth für die Tests. Anschließend erfolgt die Auswahl und das Austesten von Audiofeatures zur Musikerkennung. Als Anhaltspunkt wird hierfür das Continuous Frequency Activation-Feature gewählt, welches von Seyerlehner u.a. zur Detektion von Musik in Fernsehsendungen entwickelt wurde und auf der Erkennung von längeren, stationären Frequenzbändern in Musik basiert. Es wird für jedes Frame ein einzelner CFA-Wert berechnet und mit einem Schwellwert verglichen. So kann auf einen aufwändigen Klassifikator verzichtet werden (vgl. Seyerlehner u. a. 2007).
Das Continuous Frequency Activation-Feature wird zuerst in Matlab implementiert und anhand von Testdateien auf Plausibilität geprüft. Die Scriptingsprache Matlab wurde deshalb gewählt, weil es mit ihrer Hilfe möglich ist, ohne großen Aufwand komplizierte mathematische Berechnungen durchzuführen. Diese Umsetzung dient dazu, ein gewisses Gefühl für die Daten und die Berechnung des CFA-Features zu bekommen. Für den Echtbetrieb ist diese Implementierung aufgrund mangelnder Performance allerdings nicht brauchbar.
Deshalb wird das Feature anschließend mit Hilfe eines zuvor ausgewählten Feature Extraction Frameworks in einer Hochsprache umgesetzt, um in annehmbarer Geschwindigkeit auf den Servern des CBA ausgeführt werden zu können. Abschließend werden die Ergebnisse der beiden Implementierungen miteinander verglichen.

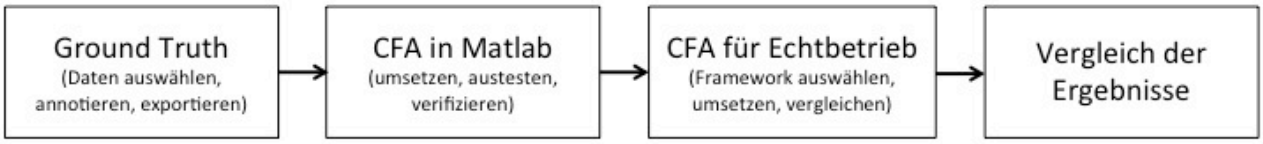


Abbildung 3: Vorgangsweise für die Entwicklung eines Feature Extractors für das CBA-Projekt.

3.1 Erstellen einer Ground Truth

Aus den 25.126 zur Zeit im Cultural Broadcasting Archive befindlichen Sendungen wurde mit Hilfe der Software *Mix2Stix (*http://software.azett.com/index.php?cat=Mix2Stix*)* eine zufällige Auswahl von 50 Dateien getroffen und anschließend durchgehört und grob kategorisiert (siehe Abbildung 4, Seite 13). Die Einteilung erfolgte in drei Kategorien:

- Musiksendung (grün hinterlegt): viel Musik, nur wenig bis gar keine Sprache dazwischen
- Sprachsendung (rot hinterlegt): nur Sprache, keine Musik bzw. Musik nur im Hintergrund
- Gemischt (nicht markiert): Sprache und Musik anteilsmäßig annähernd gleich enthalten

Außerdem wurden weitere Merkmale wie Dauer der Sendung, Vorkommen und Dauer eines Jingles (Intros) und diverse Anmerkungen zur Qualität des Materials (Störungen, schlechte Aufnahmequalität, Hall, Hintergrundgeräusche, ...) notiert.

	Dateiname	Dauer	Sprache	Musik	Intro	Art
	002unireden01110508.mp3.wav	3min 40	x			
x	88acta.mp3.wav	58min 51	x	x		diskussionssendung mit musik dazwischen
x	2010.11.19_2100.00-2159.25__Top_Records.mp3.wav	59min 25	x	x		musiksendung
x	2011.03.01liebe.mp3.wav	1h 04	x	x		diskussionssendung mit musik dazwischen
	2011.04.081900.102000.00pentagrama.mp3.wav	58min 36	x	x		spanische musik und sprache
	2012.03.191800.101850.10FROzine.mp3.wav	49min 59	x	x	12s	diskussionssendung, wenig musik, telefoninterview
x	30211kremsegg.mp3.wav	1h	x	x	28s	klassische musik, zwischendurch kurz jingle
	20100810_ausgestrahlt_GabiPohlova_45_31.mp3.wav	45min 31	x	x	32s	diskussionssendung, wenig musik
	20110411Respektiere.mp3.wav	30min 06	x	x	25s	dokumentation, wenig musik, teilweise übersteuert
	20110522IVfrischluft.mp3.wav	28min 07	x	x		interviewsendung, nur kurz musik
	20110907JugendworkshopSt.Paul.mp3.wav	51min 39	x	x	25s	musiksendung, wenig sprachbeiträge
	20120227frontex.mp3.wav	3min 59	x			
	8120112606CULT.mp3.wav	1h 49min 34	x	x		musiksendung, viel country, sehr leise gepegelt
	artarium-09-02-2009_23-30-16.mp3.wav	50s				nur werbung für sendung
x	context_xxi-24-05-2005_07-33-52.mp3.wav	57min	x	x	27s	doku, musik zum schluss
x	frozine-26-08-2008_14-07-17.mp3.wav	29min 34	x	x	10s	interviewsendung, wenig musik
	frozine-29-05-2007_15-09-16.mp3.wav	14min 26	x	x		doku, wenig musik, live mit viel hintergrundgeräushe
	gegenargumente-13-05-2007_13-13-34.mp3.wav	59min 34	x	x	1min 07	doku, telefonstimme, wenig musik
x	gfvamosmujer201201171300.mp3.wav	56min 41	x	x	7s	interviewsendung, liveaufnahmen, tw. viel hall
	h__r_saal-17-11-2010_09-10-14.mp3.wav	29min 06	x	x		doku
x	h__rmahnmal___jingles-12-05-2009_12-42-55.mp3.wav	30s				nur jingle
	interviewtimstuettgen.mp3.wav	24min 33	x			nur interview
x	klangspuren20110920.mp3.wav	52min 56	x	x		doku, viel geräuschmusik
	mitternachtsreigen-17-12-2009_07-44-13.mp3.wav	59min 24	x	x	31s	musiksendung, gothic
x	musikredaktion-20-04-2007_10-53-21.mp3.wav	13min 38	x	x	18s	telefoninterview, nur 1 langes musikstück
x	nachtfahrt-12-07-2009_00-21-22.mp3.wav	2h 59min 32	x	x		musiksendung
	Netwatcher20110530.mp3.wav	1h	x	x	6s	interviewsendung, wenig musik
x	noso___-16-08-2005_17-46-48.mp3.wav	29min 55	x	x	53s	dokusendung, viele versch. sprecher, schlechte sprachqualität, viel hall, hintergrundgeräusche, tw. spanisch
	playlist02.11.11.mp3.wav	43min 48		x		nur musik
	radio_enterbach-31-01-2010_20-38-25.mp3.wav	55min 59	x	x		musiksendung, sehr wenige ansagen
x	radio_fro__frozine-05-12-2002_15-31-57.mp3.wav	16min 07	x			telefoninterview
	radio_fro__frozine-24-06-2004_15-57-34.mp3.wav	5min 03	x			interview
	radio_fro__frozine-31-03-2003_11-38-26.mp3.wav	14min	x			interview
x	radio50plus-08-10-2010_17-07-32.mp3.wav	18min 41	x			interviews mit leiser hintergrundmusik
	radio50plus-25-03-2009_20-59-59.mp3.wav	10min 51	x			interviewsendung
x	radioattac_specials-03-02-2010_19-37-34.mp3.wav	53min 16	x	x		doku
	radioattac_specials-18-06-2006_22-43-49.mp3.wav	6min 02	x			interview
x	radioattac-19-05-2004_01-04-44.mp3.wav	28min 17	x	x	20s	interviews mit hintergrundgeräuschen
	radioattac-28-04-2009_00-04-25.mp3.wav	29min 32	x		13s	interviews mit hintergrundgeräuschen, keine musik
	radiofabrik_um_6-16-08-2002_19-14-31.mp3.wav	4min 44	x			interview
	schlossmuseum-20-06-2006_11-19-49.mp3.wav	13min 16	x	x		interviewsendung, viele kodierungsfehler
x	sg27kommunikation.mp3.wav	59min 32	x	x	1min	gesprächssendung
x	spacefemfm-26-04-2007_23-39-55.mp3.wav	1h	x		54s	musik nur im hintergrund
	trotz_allem-10-06-2009_16-39-26.mp3.wav	13min 32	x			interview
x	vice_versa__beitr_ge-13-07-2006_20-12-19.mp3.wav	15min 19	x	x		interviewsendung, free jazz
x	volksmusik_und_tradition___im_freien_radio_freistadt-16-04-2010_12-16-42.mp3.wav	58min 08	x	x		musiksendung, volksmusik
	wegstrecken-27-02-2005_11-50-48.mp3.wav	59min 41	x	x		doku, livebeiträge mit viel hintergrundgeräuschen
	willkommen_in_salzburg_2010-22-03-2010_12-25-40.mp3.wav	52min 48	x	x	47s	doku, versch. sprachen, nur kurze stücke klassischer musik

Abbildung 4: Liste der ausgewählten und kategorisierten Audiodateien für die Ground Truth (grün = Musiksendung, rot = Sprachsendung).

Anschließend erfolgte die Konvertierung der mp3-Dateien in das WAV-Format, die Reduktion der Spuren auf mono und das Resampling auf 22050 Hz mittels der Software *Sound eXchange (SoX* http://sox.sourceforge.net/*)*.

Aus dieser zufälligen Auswahl wurden 20 repräsentativen Dateien in annähernd gleichem Verhältnis von Musik-, Interview- und gemischten Sendungen ausgesucht (siehe x in Spalte 1 von Abbildung 4) und in der Folge mit Hilfe der Software *Sonic Visualizer (*http://www.sonicvisualiser.org/*)* jene Bereiche markiert, in denen Musik vorhanden ist (siehe Abbildung 5, Seite 14). Hierbei wurden auch jene Bereiche als Musik markiert, in denen diese nur im Hintergrund zu hören ist, wie etwa als Untermalung einer Gesprächssendung oder eine Band, die im Hintergrund eines Live-Interviews zu hören ist.

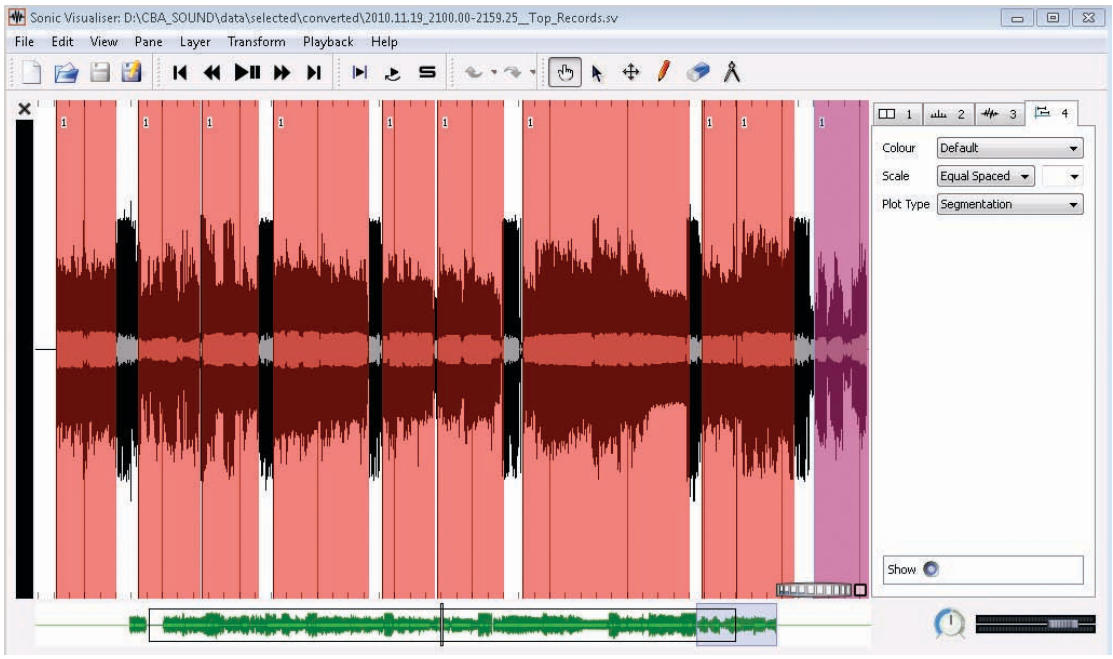


Abbildung 5: Markieren der Bereiche in denen Musik vorhanden ist.

Die Annotierungsdaten wurden dann in einem Sonic Visualizer-eigenen XML-Format exportiert und können so universell weiterverwendet werden (siehe Abbildung 6). Die markierten Bereiche werden dabei jeweils als *point* mit einem Start (*frame*) und einer Dauer (*duration*) mit der Zahl der jeweiligen Samples (Abtastwerte) angegeben. Diese Daten werden als Ground Truth für die Überprüfung der Erkennungsgenauigkeit des Continuous Frequency Activation-Features verwendet.

2010.11.19_2100.00-2159.25__Top_Records.svl — Bearbeitet

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sonic-visualiser>
<sv>
  <data>
    <model id="8" name="" sampleRate="22050" start="0" end="68845680" type="sparse" dimensions="3"
resolution="1" notifyOnAdd="true" dataset="7"  subtype="region" valueQuantization="0" minimum="1"
maximum="1" units=""/>
    <dataset id="7" dimensions="3">
      <point frame="0" value="1" duration="2108160" label="" />
      <point frame="4169728" value="1" duration="5180160" label="" />
      <point frame="11172960" value="1" duration="5406208" label="" />
      <point frame="16600140" value="1" duration="4950808" label="" />
      <point frame="22705920" value="1" duration="8256240" label="" />
      <point frame="32008320" value="1" duration="4515712" label="" />
      <point frame="36740096" value="1" duration="5698048" label="" />
      <point frame="44041216" value="1" duration="14238464" label="" />
      <point frame="59310180" value="1" duration="2937768" label="" />
      <point frame="62281440" value="1" duration="4921888" label="" />
      <point frame="68845680" value="1" duration="9673696" label="" />
    </dataset>
  </data>
  <display>
    <layer id="9" type="regions" name="Regions" model="8"  verticalScale="1" plotStyle="1"
colourName="Purple" colour="#c832ff" darkBackground="false" />
  </display>
</sv>
```

Abbildung 6: XML-Struktur der Datei von Sonic Visualizer.

3.2 Das Continuous Frequency Activation-Feature

Das Continuous Frequency Activation-Feature ist darauf ausgelegt, kontinuierliche Aktivierungen von Frequenzen in einem Signal zu finden, die im Spektrogramm des Signals durch horizontale Bänder zu erkennen sind. Diese werden üblicherweise von anhaltenden musikalischen Tönen verursacht und treten bei Musik tendenziell häufiger auf als bei anderen Schalltypen (siehe Abbildung 7). Das CFA-Feature ist eine Verbesserung bereits bekannter Methodiken, um Musik auch im Hintergrund gut erkennen zu können (vgl. Seyerlehner u. a. 2007).

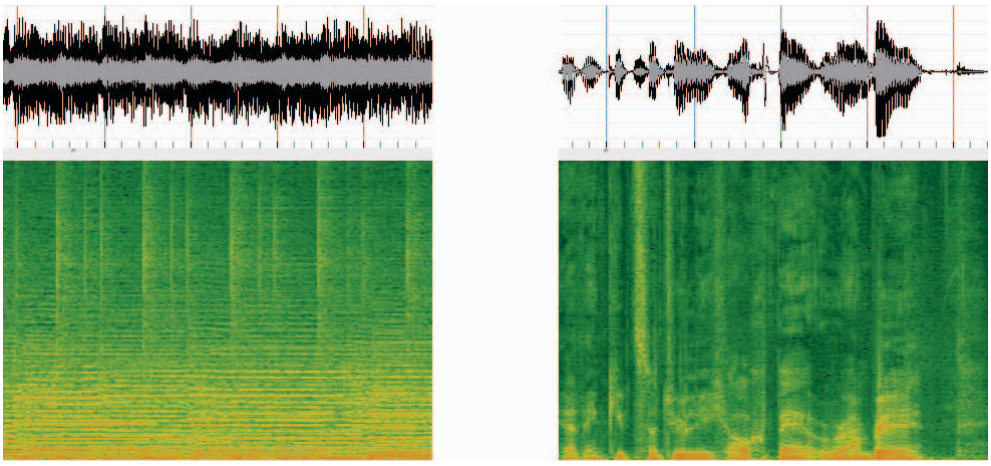


Abbildung 7: Unterschiede im Spektrogramm von Musik (links) und Sprache (rechts).
(eigene Darstellung)

Die Berechnung des Features wird dabei in folgende Schritte unterteilt (vgl. Seyerlehner u. a. 2007):

- **Umwandlung des Eingangssignals** in ein Mono-Signal mit 11kHz Samplingfrequenz.
- **Berechnung des Power Spectrum** mit einer Fenstergröße von 1024 Samples und einer Überlappung von 75% (siehe Abbildung 8 a und d, Seite 16).
- **Hervorhebung von Lokalen Spitzen** durch Abziehen des gleitenden Mittelwerts über 21 Werte, um leise Töne hervorzuheben (siehe Abbildung 8 b und e, Seite 16).
- **Binärisierung** mit einem fixen Schwellwert, um die absoluten Signalstärken herauszufiltern und so auch Hintergrundmusik erkennen zu können (siehe Abbildung 8 c und f, Seite 16).
- **Berechnung der Frequenzaktivierung** über längere Zeit durch Aufsummieren des binärisierten Signals über 100 Frames (siehe Abbildung 9, Seite 16).
- **Herausfilterung von starken Spitzen** durch Berechnen der Steigung der Flanken aller lokalen Maxima.
- **Quantifizierung der Frequenzaktivierung** durch Aufsummieren der fünf steilsten Flanken.

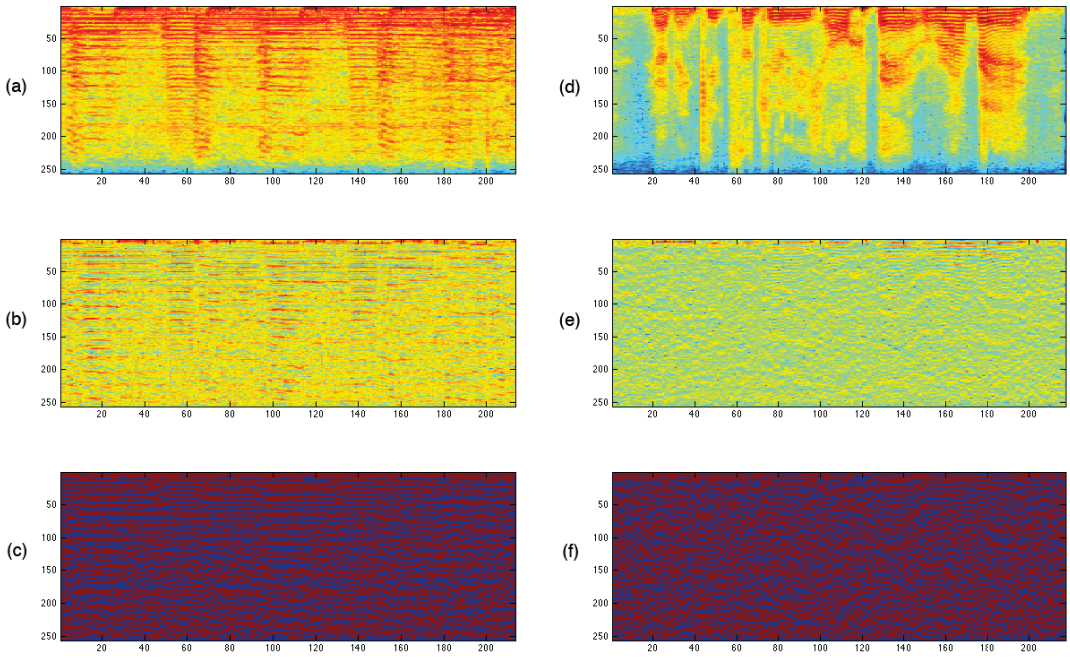


Abbildung 8: Berechnungsschritte des CFA-Features für Musik (links) und Sprache (rechts). Power Spectrum (a, d), Local Peaks (b, e), Binarization (c, f) (eigene Darstellung)

In der oberen Grafik sind sehr gut die kontinuierlichen Frequenzaktivierungen bei einem Musiksignal zu erkennen (siehe Abbildung 8 a, b und c), während diese bei einem Sprachsignal nicht so stark vorkommen (siehe Abbildung 8 d, e und f). Diese kontinuierlichen Bänder resultieren beim Aufsummieren in stärkeren und steileren Spitzen (siehe Abbildung 9). Die Steilheit dieser Spitzen wird anschließend für die Berechnung des CFA-Wertes herangezogen.

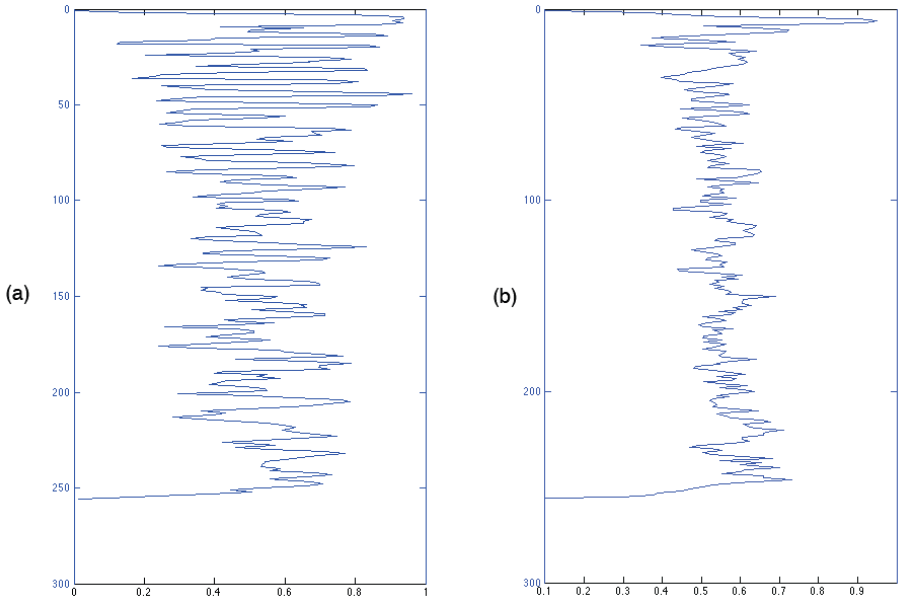


Abbildung 9: Spitzen der aufsummierten Frequenzaktivierungen für Musik (links) und Sprache (rechts). (eigene Darstellung)

3.3 Implementierung in Matlab

Das Continuous Frequency Activation-Feature wurde fürs Erste in Matlab implementiert, um die Funktionalität zu testen und die Plausibilität der im Paper von Seyerlehner u.a. angegebenen Daten zu überprüfen (vgl. Seyerlehner u. a. 2007).
Der Matlab-Code bildet genau die einzelnen Schritte nach, die für die Berechnung des CFA-Features notwendig sind. Einige Parameter sind dabei fix im Code eingestellt (*fft_size* und *fft_overlap*), einige andere können beim Funktionsaufruf mitübergeben werden (*treshold* für Binarization und *noOfPeaks* die aufsummiert werden sollen). Zusätzlich ist es möglich, sich mit dem Parameter *doVisu* die Ergebnisse der einzelnen Zwischenschritte für Testzwecke grafisch anzeigen zu lassen.
Bis auf den Schritt PeakDetection sind alle anderen Schritte mit wenigen Zeilen Matlab-Code relativ schnell und einfach umzusetzen, was der wesentliche Vorteil dieser Scriptingsprache ist.

Erste Tests der Implementierung erfolgten anschließend mit mehreren, ca. 2,5s langen Sprach- und Musikausschnitten (siehe Tabelle 1, Seite 18). Hierbei wurde der im Paper von Seyerlehner u.a. angegebene Schwellwert 0.1 für die Binärisierung angenommen. Für die ersten drei Musikdateien, die sehr tonale Musik enthalten, lässt sich erkennen, dass der CFA-Wert deutlich über der von Seyerlehner u.a. angenommenen Schwelle von 1.24 für die Unterscheidung von Musik und Nicht-Musik liegt (vgl. Seyerlehner u. a. 2007). Der CFA-Wert für die drei darauffolgenden Musikstücken, die Ausschnitte von Rockmusik sind, liegt bei zweien knapp und bei einem Ausschnitt deutlich unter diesem Schwellwert (siehe Tabelle 1 linke Hälfte, Seite 18). Dies ist darauf zurückzuführen, dass die Testausschnitte hauptsächlich rhythmische Musikanteile enthalten und wenig bis gar keine tonalen. Hier stößt das CFA-Feature schnell an seine Grenzen und muss eventuell noch verbessert werden, um eine ausreichende Erkennungsgenauigkeit für die Gesamtheit der Audiodateien im CB-Archiv zu erreichen.
Für die Sprachausschnitte wurden verschiedene Sprecherinnen und Sprecher ausgewählt, um nicht von einer Sprechweise oder Tonlage abhängig zu sein. Bei allen liegt der berechnete CFA-Wert deutlich unter dem Schwellwert (siehe Tabelle 1 rechte Hälfte, Seite 18).
Ob es allerdings reicht, zugunsten der Erkennung von Musik, den Schwellwert einfach entsprechend tiefer anzusetzen, kann noch nicht beantwortet werden. Das muss erst anhand einer entsprechend großen, repräsentativen Menge von annotierten Radiosendungen aus der Ground Truth ermittelt werden. Diese ersten Tests dienen nur dazu, die Implementierung des CFA-Features auf Richtigkeit zu überprüfen. Es können sich jederzeit, durch etwaige Berechnungsunterschiede der FFT oder der PeakDetection, größere oder kleinere Unterschiede zur Implementierung von Seyerlehner u.a. ergeben. Diese schlagen sich natürlich in unterschiedlichen Ergebnissen nieder. Auch sind die Werte des Features immer vom jeweiligen Audiomaterial abhängig.

MUSIC			SPEECH		
Filename	Peaks	CFA-Value	Filename	Peaks	CFA-Value
mus_git_01_11k_mono.wav	0.3988	1.5869	sprache_fem_01_11k_mono.wav	0.1156	0.4971
	0.3526			0.1040	
	0.2832			0.1040	
	0.2832			0.0867	
	0.2717			0.0867	
mus_git_02_11k_mono.wav	0.3954	1.8007	sprache_fem_02_11k_mono.wav	0.1636	0.5596
	0.3856			0.1212	
	0.3627			0.0990	
	0.3399			0.0909	
	0.3170			0.0848	
mus_intro_01_11k_mono.wav	0.6012	2.5318	sprache_fem_03_11k_mono.wav	0.1895	0.6863
	0.5260			0.1569	
	0.4855			0.1242	
	0.4740			0.1111	
	0.4451			0.1046	
mus_rock_01_11k_mono.wav	0.2543	1.0751	sprache_mask_01_11k_mono.wav	0.2701	0.7591
	0.2312			0.1533	
	0.2254			0.1241	
	0.1850			0.1095	
	0.1792			0.1022	
mus_rock_02_11k_mono.wav	0.1156	0.5048	sprache_mask_02_11k_mono.wav	0.2810	0.8139
	0.1118			0.1460	
	0.1040			0.1314	
	0.0867			0.1314	
	0.0867			0.1241	
mus_rock_03_11k_mono.wav	0.3584	1.1869	sprache_mask_03_11k_mono.wav	0.1440	0.5227
	0.2197			0.1200	
	0.2197			0.0960	
	0.2081			0.0827	
	0.1811			0.0800	

Tabelle 1: Berechnungsergebnisse des CFA-Features in Matlab anhand einiger Testdateien.

3.4 Tools zur Feature-Extraktion

Um eine geeignete Basis für die Umsetzung des Audio Klassifikators für das CBA zu finden, wurden verschiedene Tools zur Extraktion von Audiofeatures untersucht und miteinander verglichen:

- **Matlab MIR-Toolbox**:
  (http://www.mathworks.com/matlabcentral/fileexchange/24583-mirtoolbox)
  ist eine sehr umfangreiche Signalverarbeitungs-Toolbox zur Musikinformationsgewinnung (Music Information Retrieval – MIR) für die bekannte Programmierumgebung Matlab (vgl. Lartillot / Toiviainen 2007). Wegen der schnellen und einfachen Anwendbarkeit ist es für Test- und Vergleichsberechnungen zwar durchaus das Produkt der Wahl, jedoch für den Einsatz als Echtzeitsystem aufgrund des hohen Lizenzpreises und aus Performancegründen nicht brauchbar.

- **Marsyas:** (http://marsyas.info/)
  ist ein Open Source Framework zur Audioverarbeitung in C++ und Java, das sehr darauf ausgelegt ist, mit großen Datenmengen umzugehen (vgl. Tzanetakis / Cook 1999). Die Feature Extraktion wird von dem Programmteil *bextract* durchgeführt und ist nur ein kleiner Teil des gesamten Frameworks. Dieser ist allerdings nur mäßig dokumentiert und scheint aus diesem Grunde ungeeignet für eigene Erweiterungen.

- **Yaafe:** (http://yaafe.sourceforge.net/)
  ist ein Audio Feature Extraction Tool, das speziell darauf ausgelegt ist, eine große Anzahl an Features aus großen Datenmengen zu extrahieren, und dabei ressourcenschonend zu arbeiten. Trotzdem ist es mittels eines *Feature Extraction Plans* noch relativ einfach aber umfangreich zu parametrieren (vgl. Mathieu u. a. 2010). Zudem ist die Software Open Source, der Aufbau gut strukturiert und somit ohne großen Aufwand erweiterbar.

- **libxtract**: (http://libxtract.sourceforge.net/)
  ist eine kleine und schnelle C++-Bibliothek, die ein Vielzahl an Features bietet (vgl. Bullock 2007). Die Implementation allerdings ist nur dürftig dokumentiert und so nur mit viel Einarbeitungsaufwand zu erweitern. Ebenso sind auf der Projekthomepage schon seit mehr als einem Jahr keine nennenswerten Aktivitäten mehr verzeichnet.

- **jAudio:** (http://jaudio.sourceforge.net/)
  ist ein Java-basierter Audio Extractor, der zwar speziell darauf ausgelegt ist, dass er einfach erweitert werden kann, ohne neu kompiliert werden zu müssen (vgl. McEnnis / McKay / Fujinaga 2006) und auch sehr gut dokumentiert ist, jedoch nicht so ressourcenschonend arbeitet, wie andere Frameworks und laut Projekthomepage auch nicht mehr weiterentwickelt wird.

Die Auswahl des Feature Extraction Frameworks für das CBA erfolgte nach folgenden Kriterien:

- Open Source Software: Der Sourcecode soll frei zugänglich sein und es dürfen keine Lizenzkosten für das CBA entstehen.
- Anzahl und Erweiterbarkeit der Features: Es sollen möglichst viele Standardfeatures bereits implementiert sein und der Aufbau des Frameworks soll eine einfache Erweiterung um neue Features zulassen bzw. durch ausreichende Dokumentation unterstützen.
- Geschwindigkeit / Stabilität: Das Framework soll möglichst schnell und ressourcenschonend auf dem Server des CBA arbeiten und die gewünschten Features extrahieren.

Im Fall des Softwaretools *Yaafe* scheinen alle diese genannten Anforderungen erfüllt zu sein und so wurde es als Audio Feature Extractor für das CBA-Projekt ausgewählt.

3.5 Implementierung des CFA-Features in Yaafe

Wie bereits zu Beginn dieses Kapitels erwähnt (siehe Seite 12), soll das Continuous Frequency Activation-Feature von Seyerlehner u.a. in dieses Framework implementiert werden, da es sehr gute Ergebnisse für die Erkennung von Musik liefert und dennoch einfach und effizient zu berechnen ist (vgl. Seyerlehner u. a. 2007).

3.5.1 Anforderungen und Installation von Yaafe

Für die Installation von Yaafe, welches momentan nur für die Betriebssysteme Linux und MacOSX lauffähig ist, bestehen folgende Voraussetzungen (vgl. Yaafe 2011a):

- Ein **C-Compiler**, um die Source Codes kompilieren zu können (ist meist im Betriebssystem schon integriert).
- Das Paket **CMake** in der Version 2.6 ist ebenfalls für die Kompilierung notwendig.
- Die Bibliothek **Argtable2** zum Parsen der Kommandozeilenparameter.
- Die Bibliothek **libsndfile** zum Lesen von WAV-Files.
- Ein **Python**-Interpreter in einer Version >= 2.5.
- Das Paket **NumPy**, um große Datenstrukturen verarbeiten und C/C++-Code einfach integrieren zu können.

Einige weitere Pakete (libmpg123, liblapack, libhdf5, libfftw3, ...) sind je nach Anwendungsfall optional und nicht zwingend für die Lauffähigkeit von Yaafe notwendig.

Im Fall des Betriebssystems **Linux** können all diese Pakete mit Hilfe des Paketmanagers *APT* (Advanced Packaging Tool) mit folgendem Kommando auf einmal installiert werden:

```
sudo apt-get install cmake cmake-curses-gui libargtable2-0 libargtable2-dev
     libsndfile1 libsndfile1-dev libmpg123-0 libmpg123-dev libfftw3-3 libfftw3-dev
     liblapack-dev libhdf5-serial-dev libhdf5-serial-1.8.4
```

Beim Betriebssystem **MacOSX** kann auf den Paketmanager *HomeBrew* (http://mxcl.github.com/homebrew/) zurückgegriffen werden, für den Devon Bryant eine Installationsformel für Yaafe und dessen Paketabhängigkeiten geschrieben hat (https://github.com/devonbryant/homebrew/blob/master/Library/Formula/yaafe.rb).

Es können aber auch die Source Codes aller Pakete einzeln heruntergeladen werden. Die Links dazu befinden sich auf der Webseite von Yaafe (http://yaafe.sourceforge.net/manual/install.html). Zur Installation folgt man den Anweisungen in den jeweiligen Readme-Files. Die meisten Pakete lassen sich mit den drei Unix-Standardbefehlen kompilieren und installieren:

```
sudo ./configure
sudo make
sudo make install
```

Wenn alle Abhängigkeiten des Yaafe-Pakets erfüllt sind, kann mit folgenden Befehlen die Installation vorbereitet werden:

```
mkdir build
cd build
ccmake ..
```

Der letzte Befehl starten den Konfigurationsclient, in dem noch einige Parameter für die Installation gesetzt werden können, die dann mit den Tastenkürzeln **c** für *configure* und **g** für *generate* noch bestätigt werden müssen. Die Installation selbst wird dann mit den folgenden Befehlen gestartet:

```
sudo make
sudo make install
```

Zum Abschluss der Installation müssen noch einige Umgebungsvariablen gesetzt werden, damit die Erweiterungen beim Ausführen auch gefunden werden (vgl. Yaafe 2011a).

3.5.2 Unterteilung in Features und Components

In Yaafe erfolgt die Unterteilung der Berechnung eines Features in einzelne Teilschritte, sogenannte Komponenten. Diese Aufteilung ist einerseits sinnvoll, da so der Sourcecode einzelner Komponenten in anderen Features wiederverwendet werden kann. Zum anderen wird in Yaafe erst zur Laufzeit entschieden, welche Berechnungsschritte für gerade zu extrahierende Features gleich sind und welche Berechnungsergebnisse so wiederverwendet werden können (vgl. Mathieu u. a. 2010).

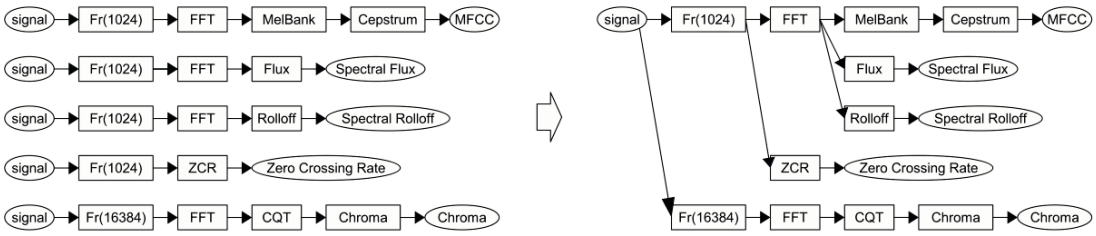


Abbildung 10: Aufteilung der Features in Komponenten und Wiederverwendung bereits berechneter Komponenten zur Laufzeit (Mathieu u. a. 2010)

Bei Extraktion einer großen Anzahl von Features macht sich die erwähnte Unterteilung bemerkbar, da jeweils gleiche Teilkomponenten für unterschiedliche Features nicht mehrmals berechnet werden müssen und so Zeit bzw. Rechenleistung gespart werden können. Die Feature-Bibliothek ist hierbei aus Gründen der Einfachheit und besseren Lesbarkeit in Python umgesetzt, während hingegen die einzelnen Komponenten-Bibliotheken aus Performancegründen in C++ geschrieben sind (vgl. Mathieu u. a. 2010).
Zahlreiche Audiofeatures, die für die Erkennung von Musik eingesetzt werden (MFCC, ZCR, Spectral Rolloff, LPC, ..., siehe. Kapitel 2.2), sind in Yaafe bereits implementiert (vgl. Yaafe 2011b) und können zu Vergleichszwecken mit dem CFA-Feature einfach mitextrahiert werden.

3.5.3 Aufteilung des CFA-Features in Komponenten

Für die entsprechende Aufteilung des CFA-Features in Komponenten wurden die einzelnen Teilschritte genauer untersucht und auf Gemeinsamkeiten mit bestehenden Features/Komponenten in Yaafe analysiert. Die Deklaration, welches Feature aus welchen Komponenten bestehen und in welcher Reihenfolge die Berechnungsschritte abgearbeitet werden, erfolgt in der Datei *„/src_python/yaafefeatures.py“* mit nur wenigen Zeilen Sourcecode.

Für den Schritt *Berechnung des Power Spectrum* kann auf das bestehende Feature *MagnitudeSpectrum* zurückgegriffen werden, das wiederum aus den Komponenten *Frames*, *FFT* und *Abs* besteht. Zusätzlich wurde der Berechnungsschritt **WindowNormalize** eingeführt, welcher als Komponente erst umgesetzt werden muss. Anschließend ist die Quadrierung des Spektrums notwendig, wo wiederum die bestehende Komponente *SQR* verwendet werden kann, und die **DB-Konvertierung**, die noch selbst umgesetzt werden muss.
Für die *Hervorhebung der Lokalen Spitzen* wird die Komponente **SubRunningAverage** eingeführt, zur *Binärisierung* die Komponente **Binarization**, zur *Berechnung der Frequenzaktivierung* die Komponente **FrameSum** und zum *Herausfiltern der Starken Spitzen* die Komponente **PeakDetection**. Abschließend ist noch die *Quantifizierung* notwendig, bei der die Steigungen der fünf steilsten Flanken mittels der bereits existierenden Komponente *Sum* aufsummiert werden und so den Wert des CFA-Features ergeben.

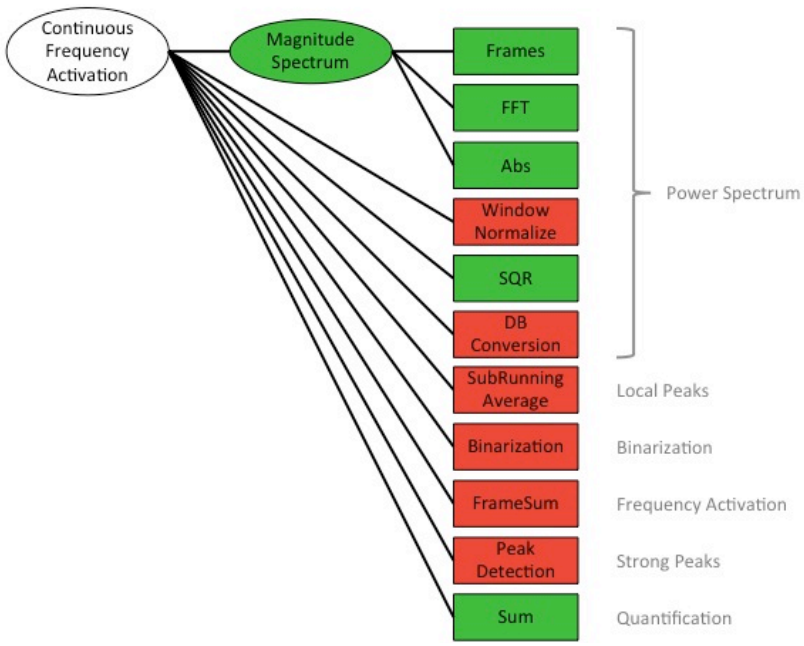


Abbildung 11: Aufteilung des CFA-Features in Komponenten.
(eigene Darstellung)

Bei den grün hinterlegten Schritten kann auf bereits existierende Komponenten des Yaafe-Frameworks zurückgegriffen werden (vgl. Yaafe 2011b), während die rot hinterlegten erst selbst implementiert werden müssen.

3.5.4 Umsetzung der neuen Komponenten

Jede Komponente ist in Yaafe durch eine eigene Sourcedatei (*.cpp*-Datei) mit einer zugehörigen Headerdatei (*.h*-Datei) im Ordner *„/src_cpp/yaafe-components/audio/“* umgesetzt. Darin wird jeweils eine neue Klasse definiert, die entweder von der Klasse *ComponentBase*, für frameübergreifende Berechnungen, oder der Klasse *StateLessOneInOneOutComponent*, bei der für jeden Eingangswert ein Ausgangswert berechnet wird, abgeleitet wird. Die beiden Klassen sind in den Dateien *Component.h / .cpp* bzw. *ComponentHelpers.h* im Ordner *„src_cpp/yaafe-core“* definiert.
Dort wird außerdem noch die Klasse *TemporalFilter* definiert, die das Grundgerüst für Filter bereitstellt, welche in unserem Fall allerdings nicht verwendet werden.

Diese drei Klassen sind ihrerseits von der Klasse *Component* abgeleitet, die in *Component.h / .cpp* definiert ist, und die Grundstruktur einer Komponente bereitstellt. Im Wesentlichen besteht diese aus den 4 Funktionen *init*, *reset*, *process* und *flush*, die nach folgendem Schema aufgerufen werden:

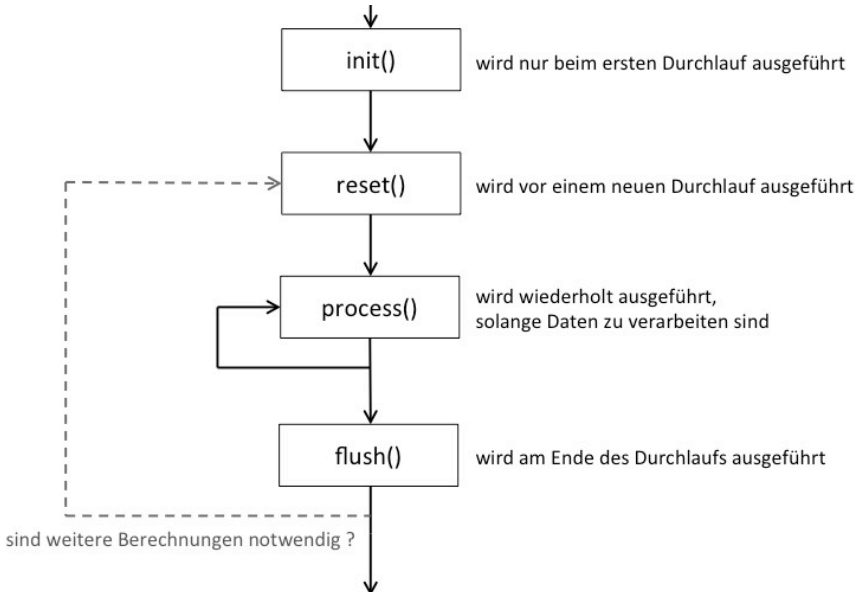


Abbildung 12: Ausführungsablauf einer Komponente.
(eigene Darstellung)

Für die Umsetzung der neuen Komponenten wurden jeweils bestehende Komponenten als Vorlage genommen, analysiert und anschließend deren Code entsprechend abgeändert bzw. erweitert. Die Headerdatei jeder neuen Komponente muss in der Datei *„src_cpp/yaafe-components/registration.cpp“* inkludiert und der Prototyp der Klasse registriert werden, damit sie zur Verarbeitung zur Verfügung steht.

Nach dem Hinzufügen von neuen Dateien zum Sourcecode-Paket müssen außerdem die Befehle **`ccmake ..`** , **`sudo make`** und **`sudo make install`** vom Installationsvorgang ausgeführt werden (siehe Kapitel 3.5.1), damit der Sourcecode neu kompiliert wird. Bei Änderungen in einer bereits bestehenden Datei sind die beiden letzten Befehle ausreichend.

Für die Komponente **WindowNormalize** wurde die bestehende Komponente *Normalize* als Vorlage genommen. Diese wurde dahingehend abgeändert, dass nicht durch die Summe der Eingangsdaten dividiert wird, sondern durch die Summe der Werte des Normalisierungsfensters. Dessen Einbindung wurde von der Komponente *FFT* abgeschaut und kann als Parameter festgelegt werden.

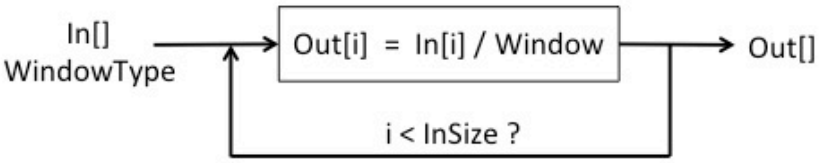


Abbildung 13: Ablauf der Komponente WindowNormalize.

Der Sourcecode der Komponente **DBConversion** wurde von der Komponente *Abs* abgeschaut. Für jeden Eingangswert wird ein neuer Ausgangswert nach der Formel dB=10*log10(P) berechnet, ohne Parameter oder sonstigen Abhängigkeiten.

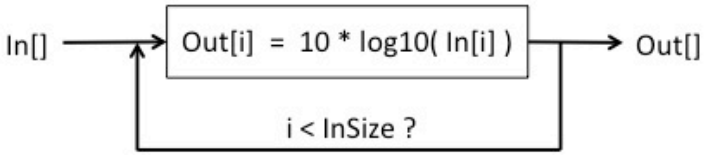


Abbildung 14: Ablauf der Komponente DBConversion.

Für die Komponente **SubRunningAverage** wurden die beiden Komponenten *SlopeIntegrator* und *Decrease* als Vorlage hergenommen. Sie hat als Parameter die Anzahl der Werte, die aufsummiert werden sollen. Es wird jeweils von einem Eingangswert der Mittelwert der n/2 vorangegangenen und n/2 nachkommenden Werte abgezogen. Am oberen und unteren Ende der Eingangswerte wird der erste bzw. letzte Wert wiederholt.

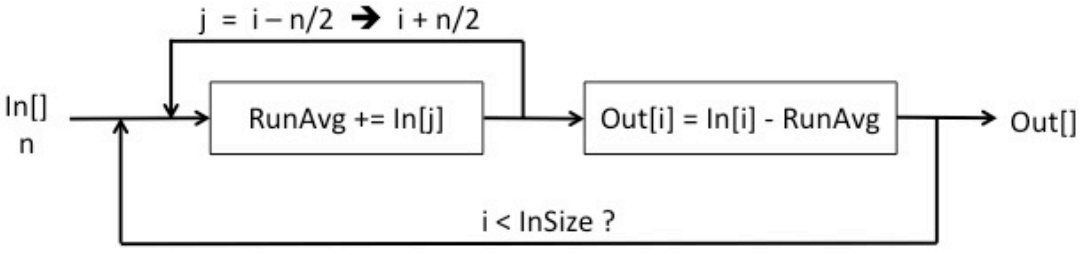


Abbildung 15: Ablauf der Komponente SubRunningAverage.

Gleich wie beim Schritt *DBConversion,* wurde für die Komponente **Binarization** die bestehende Komponente *Abs* als Vorlage genommen und um einen Parameter erweitert, der den Schwellwert für die Binärisierung angibt. Jeder Eingangswert wird mit dem Schwellwert verglichen und wenn er darüber liegt, wird der Ausgangswert auf 1 gesetzt, andernfalls wird er auf 0 gesetzt. Die Implementierung des Schwellwertes wurde von der Komponente *AutoCorrelation* abgeschaut.

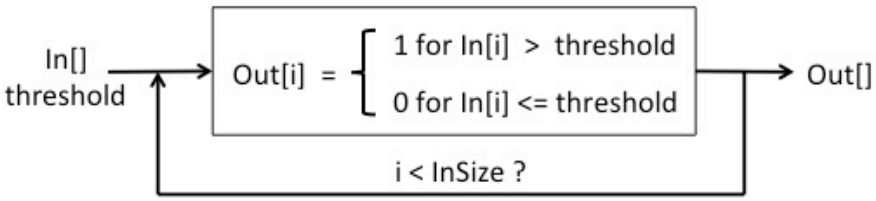
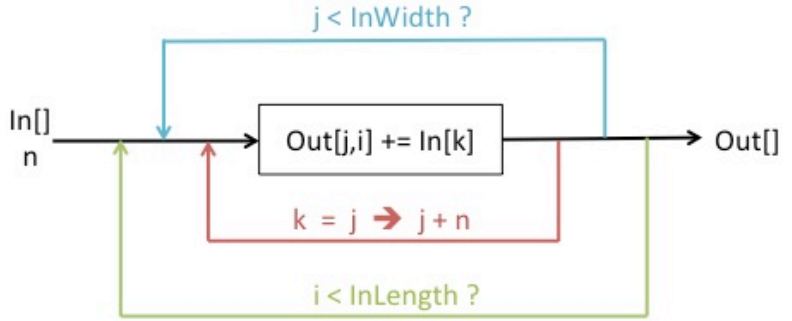


Abbildung 16: Ablauf der Komponente Binarization.

Die Komponente **FrameSum** fällt schon um einiges komplizierter aus als die vorangegangenen. Sie hat zwei Parameter: die Anzahl der Frames, die aufsummiert werden sollen, und die Schrittweite. Über drei, ineinander verschachtelte Schleifen werden die Werte der angegebenen Anzahl an Frames zu sogenannten *Blocks* aufsummiert. Dieser Mechanismus wurde von der Komponente *StatisticalIntegrator* abgeschaut.



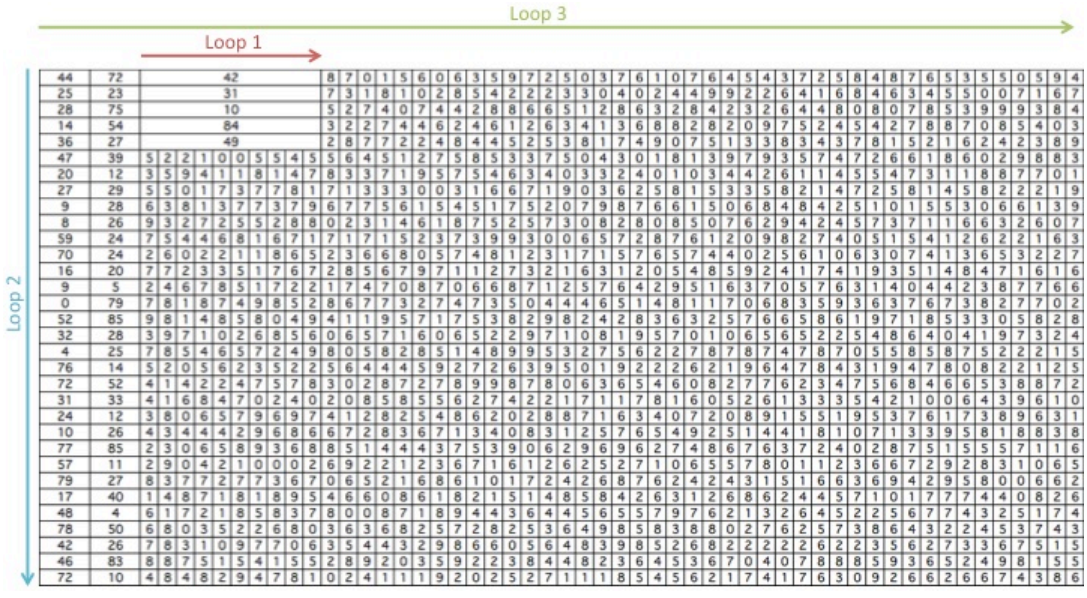


Abbildung 17: Ablauf der Komponente FrameSum.

Die Komponente **PeakDetection** ist zwar die aufwändigste von allen, deswegen jedoch nicht die komplizierteste. Sie hat nur einen Parameter, die Anzahl der Spitzen, die ausgegeben werden sollen. Das Grundgerüst wurde von der Komponente *ShapeStatistics* abgeschaut, der Ablauf folgt eigentlich der Implementation in Matlab.
Es wird jeder Eingangswert mit dem vorherigen verglichen und detektiert, ob eine Richtungsänderung des Signalverlaufs erfolgt ist. Diese Punkte werden dann als Minima bzw. Maxima markiert. Jedes Maximum wird mit den links und rechts von ihm liegenden Minima verglichen und die Steigung der Flanke des Minimums mit der geringeren Höhendifferenz als Peakwert gespeichert. Alle diese Schritte können innerhalb eines Durchlaufs der Daten durchgeführt werden. Anschließend werden diese Peakwerte noch in absteigender Reihenfolge sortiert und die als Parameter festgelegte Anzahl ausgegeben.

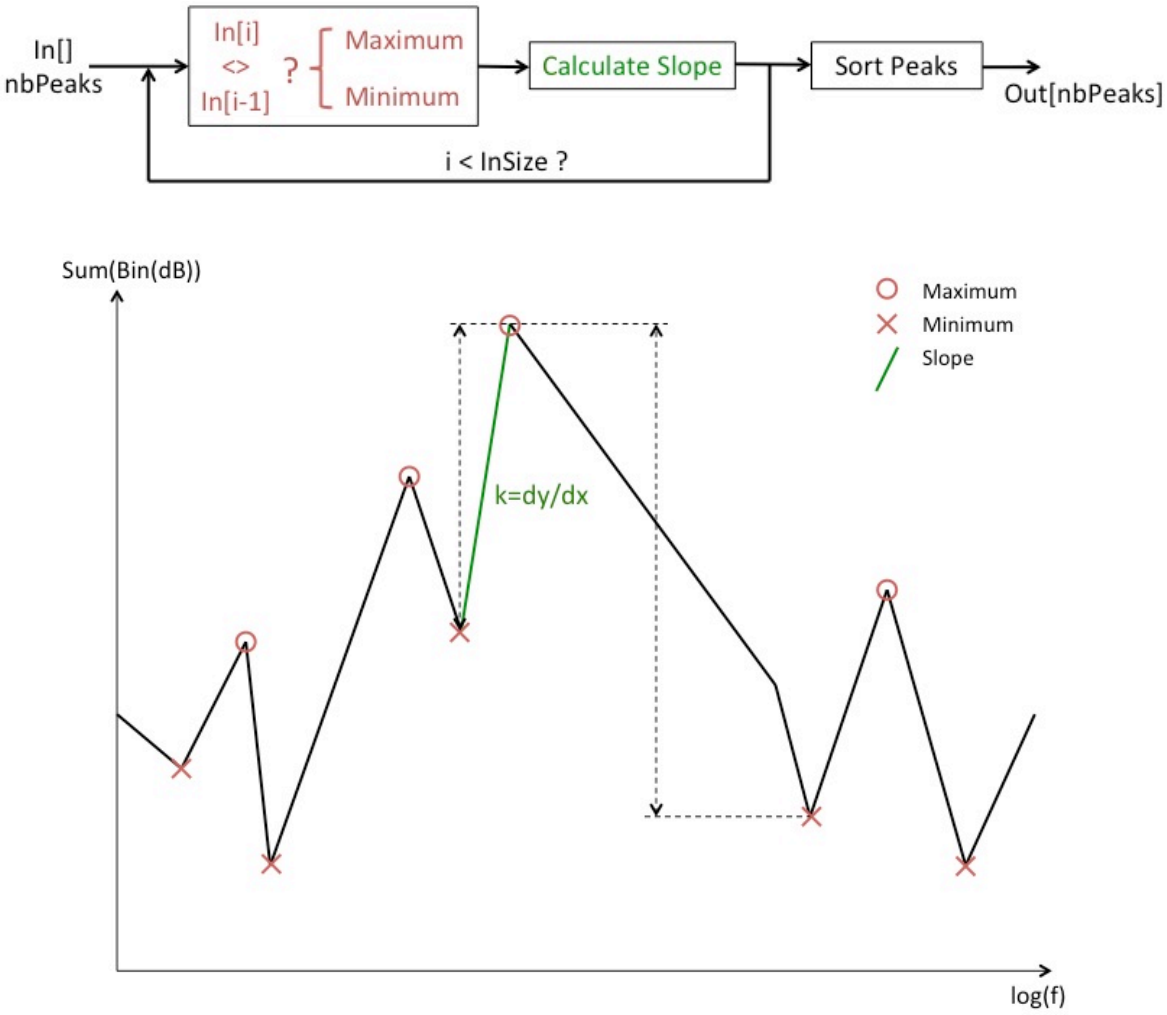


Abbildung 18: Ablauf der Komponente PeakDetection.
(eigene Darstellung)

In der letzten Komponente *Sum* werden diese Peakwerte addiert und ergeben so den CFA-Wert für jeden *Block*.

3.6 Vergleich mit Matlab-Ergebnissen

Während der Implementation der einzelnen Komponenten in Yaafe, wurden die Ergebnisse der Berechnungen mit der Hilfe von Testdateien schrittweise immer mit den Ergebnissen der Matlab-Implementation verglichen. Hierzu wurden die von Yaafe ausgegebenen *.csv-Dateien in Matlab importiert und die Daten dort grafisch dargestellt.

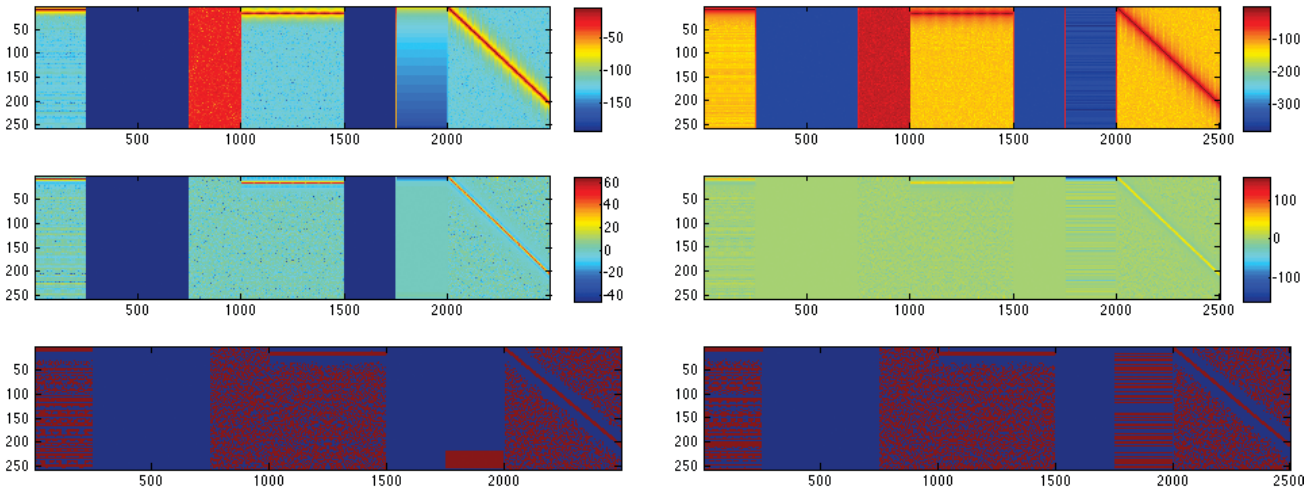


Abbildung 19: Grafischer Vergleich der Berechnungsergebnisse aus Matlab (links) und Yaafe (rechts).
(eigene Darstellung)

Auffällig sind sofort die unterschiedlichen Farben im ersten (Fouriertransformation, Normierung und DB-Konvertierung) und zweiten Berechnungsschritt (Betonung der lokalen Spitzen), die das Ergebnis eines unterschiedlichen Wertebereichs sind (siehe Abbildung 19). In Matlab treten Werte bis ca. -180dB auf, während in Yaafe Werte bis ca. -340dB berechnet werden. Zurückzuführen ist dies auf die Verwendung unterschiedlicher Zahlenformate. Durch den dritten Berechnungsschritt (Binärisierung) werden diese Unterschiede aber wieder relativiert, da alle Werte unter einem gewissen Schwellwert auf 0 gesetzt werden.

Im Bereich zwischen 1.750 und 2.000 lässt sich noch ein Unterschied der beiden Grafiken erkennen, der seinen Ursprung in der unterschiedlichen Reaktion der Fouriertransformation (FFT) auf das Audiosignal in diesem Bereich hat. Dieser Ausschnitt besteht nur aus einem positiven Gleichanteil (Offset) und kann somit in einem einfach Preprocessingschritt herausgefiltert werden.

Ein weiterer, kleiner Unterschied in der Berechnung der FFT ist am Beginn und am Ende eines relativ kurzen Audiosignals erkennbar (siehe Abbildung 20, Seite 28). Yaafe bildet hierfür jeweils ein Fenster, das die halbe Fensterbreite weit zurück und die halbe Fensterbreite weit nach vor reicht. Nicht vorhandene Werte werden dabei mit 0 aufgefüllt. Deshalb ist im rechten Bild an den Rändern eine farbliche Veränderung bemerkbar, die bei der Berechnung der FFT in Matlab nicht vorkommt.

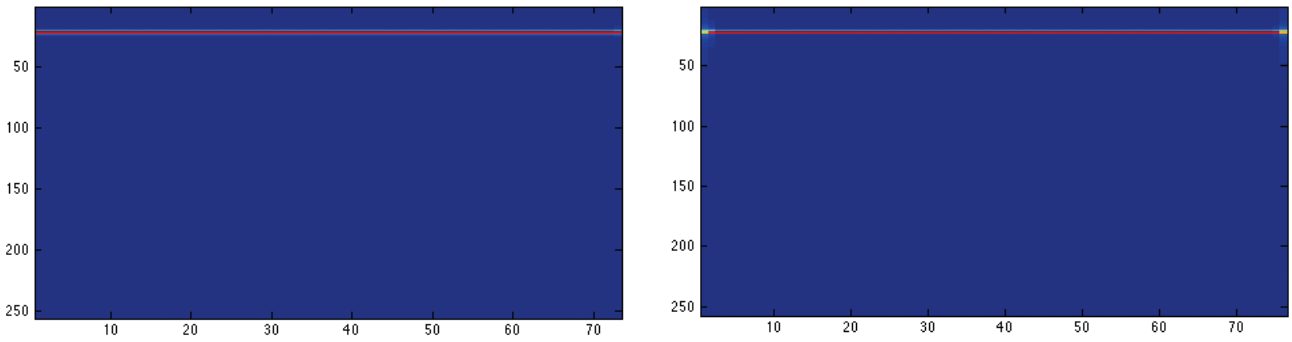


Abbildung 20: Unterschiede in der Berechnung der FFT von kurzen Audiosignalen in Matlab (links) und Yaafe (rechts). (eigene Darstellung)

Minimale Abweichungen gibt es noch in der Anzahl der Samplewerte bei gleicher Fensterbreite (Matlab: 73 und Yaafe: 76, siehe Abbildung 20), die ihren Ursprung ebenfalls in der unterschiedlichen Bildung des Fensters für die FFT hat.

In absoluten Zahlen sehen die Abweichungen der Berechnungen für zwei Testfiles so aus:

	MUSIC		SPEECH	
	Matlab	Yaafe	Matlab	Yaafe
Peaks	0.3662	0.3657	0.1475	0.1512
	0.2793	0.2824	0.1336	0.1342
	0.2441	0.2453	0.1083	0.1041
	0.2183	0.2152	0.1060	0.0833
	0.1995	0.1990	0.0876	0.0833
CFA-Value	1.3075	1.3078	0.5829	0.5563

Tabelle 2: Vergleich der CFA-Werte von Matlab und Yaafe.

Die Zahlenwerte stimmen größtenteils zumindest auf die erste Nachkommastelle überein, in vielen Fällen sogar auf die zweite (siehe Tabelle 2). Diese Abweichungen ergeben sich aus den oben genannten Unterschieden in der Fensterbildung für die FFT und treten bei diesen kurzen Testdateien natürlich verstärkt auf. Im Echtbetrieb mit entsprechend langen Audiodateien und nachfolgendem Postprocessing sollten die Unterschiede nicht mehr so deutlich auftreten, diese Theorie muss jedoch erst noch ausreichend verifiziert werden.

4 Schlussfolgerungen

Die umfangreichen Literaturrecherchen zum Thema Mustererkennung, Klassifizierung im Allgemeinen, und im Speziellen von Audiomaterial bildeten eine gute Wissensbasis für den Beginn des CBA-Projekts. Es gibt bereits eine große Anzahl an Audiofeatures, die zur Musikerkennung verwendet und in zahlreichen wissenschaftlichen Artikeln beschrieben werden. Die Klassifizierungsergebnisse sind allerdings sehr unterschiedlich und hängen stark vom verwendeten Audiomaterial ab. Als guter Ausgangspunkt schien das Continuous Frequency Feature geeignet, welches von Seyerlehner u.a. 2007 zur Musikerkennung in Fernsehsendungen entwickelt wurde (vgl. Seyerlehner u. a. 2007).
Zu Beginn wurde das vorhandene Datenmaterial des Cultural Broadcasting Archive grob analysiert und eine Ground Truth gebildet, die als Referenz für die Umsetzung des CFA-Features und zum Training für Klassifikatoren mit anderen Features dienen soll. In weiterer Folge wurde das CFA-Feature zuerst in Matlab umgesetzt, um einfache Tests zur Verifizierung der von Seyerlehner u.a. angegebenen Daten durchführen zu können. Diese zeigten, dass eine eindeutige Erkennung von Musik nicht immer gegeben ist, besonders wenn diese wenig, bis gar keine tonalen Anteile besitzt.
Dennoch wurde das Feature mit Hilfe des Feature Extraction Frameworks *Yaafe* in C++ umgesetzt, um auch größere Datenmengen in einer halbwegs brauchbaren Zeit analysieren zu können.
Die Berechnungsergebnisse der beiden Implementierungen wurden anschließend miteinander verglichen und systembedingte Unterschiede aufgezeigt. Hierbei war zu erkennen, dass die Testergebnisse des CFA-Features noch wenig Aussagekraft besitzen und es noch weiterer Tests und Vergleiche bedarf, um zu einer relevanten Erkennungsgenauigkeit von Musik für das CBA zu kommen.

5 Weiterführende Schritte

Umfang dieser Bachelorarbeit war es, die Aufgabe und Funktion von Audio-Klassifikation zu erklären und die erste Umsetzung des CFA-Features in einem Feature Extraction Framework zu beschreiben. Für einen erfolgreichen Abschluss des CBA-Projekts sind jedoch noch folgende Schritte notwendig:

5.1 Vergleich des CFA-Features mit anderen Features

Um eine bestmögliche Klassifizierungsgenauigkeit von Musik im CBA-Archiv erreichen zu können, muss das CFA-Features noch mit einer ausreichend großen Ground Truth getestet werden. Ebenso müssen die Schwellwerte für die Binärisierung und die Erkennung von Musik und Nicht-Musik für das vorliegende Datenmaterial noch genau ermittelt werden.

Außerdem ist es notwendig, andere Features, die sich für die Erkennung von Musik in Audiodaten eignen (siehe Kapitel 2.2), mit geeigneten Klassifikatoren (siehe Kapitel 2.3) anhand der Ground Truth zu testen. Diese Ergebnisse können dann für Vergleiche der Erkennungsgenauigkeit des CFA-Features herangezogen werden. Sollte diese nicht ausreichend groß sein, muss das CFA-Feature evtl. noch verbessert oder mit anderen Features kombiniert werden.

5.2 Fertigstellen des Softwarepakets

Für die einfache und sichere Anwendbarkeit ist es für das CBA wichtig, nur ein Softwaretool ansprechen zu müssen, das alle notwendigen Schritte der Klassifizierung übernimmt. Deshalb müssen die nachfolgend erklärten Teile noch mit dem Yaafe-Feature Extraction Tool zu einem gesamten Paket zusammengefügt werden. Hierzu wird voraussichtlich die Scripting-Sprache Python zum Einsatz kommen, welche bereits in Yaafe die Aufgabe des Feature Plan Parsers übernimmt (vgl. Mathieu u. a. 2010) und aufgrund der einfachen Anwendbarkeit und schnellen Ausführung dafür bestens geeignet scheint.
Ein Punkt, der mit dem Kunden noch abgeklärt werden muss, ist das Format der Ausgabedatei. Aus Flexibilitäts- und Portabilitätsgründen wird dies wahrscheinlich das XML-Format sein.

Preprocessing

Für die Feature-Extraction müssen die Audiodateien entsprechend vorverarbeitet werden (siehe Kapitel 2.1). Die Reduktion der Samplingrate auf 11025Hz kann durch den, in Yaafe bereits integrierten, SMARC Audio Rate Converter erfolgen. Eine Umkonvertierung der Dateien in das WAV-Format mit einer Reduktion der Kanäle auf mono muss vorher von einem anderen Tool erledigt werden.
Ebenso überlegens- und überprüfenswert wäre noch eine weitere Vorverarbeitungen des Signals, wie z.B. eine Tiefpassfilterung oder Nullstellen herauszufiltern, um die zu analysierende Datenmenge noch weiter zu reduzieren und eventuelle Berechnungsfehler schon vorher abzufangen.

Klassifizierung

Für den Fall, dass das CFA-Feature alleine keine ausreichende Erkennungsgenauigkeit der Musik bietet und mehrere Features gemeinsam herangezogen werden müssen, muss erst noch ein geeignetes Klassifizierungstool ausgewählt und integriert werden (z.B. WEKA, LibSVM, ...). Der Klassifikator muss dann mit dem ausgewählten Featureset und der Ground Truth trainiert werden.

Postprocessing

Als letzter Schritt im Klassifizierungsablauf ist noch die Nachverarbeitung der Daten notwendig (siehe Kapitel 2.4). Hierfür müssen noch geeignete Methodiken heuristisch ermittelt (z.B. Glättung durch Mittelwertbildung, ...) und entsprechend umgesetzt werden.

6 Literaturverzeichnis

Bishop, C.M. (2006). Pattern Recognition And Machine Learning. New York, NY, USA: Springer Science+Business Media.

Bullock, J. (2007). LibXtract: A Lightweight Library for Audio Feature Extraction. In: Proceedings of the International Computer Music Conference. Copenhagen, Denmark: ICMA, S. 25–28.

CBA (2012). Was ist das CBA? Cultural Broadcasting Archive. [http://cba.fro.at/info (ausgehoben 17.05.2012)].

Duda, R.O. / Hart, P.E. / Stork, D.G. (2001). Pattern Classification. 2. Aufl. New York, NY, USA: Wiley-Interscience.

Khan, M. / Al-Khatib, W. (2006). „Machine-learning based classification of speech and music." In: Multimedia Systems, (2006), 12(1), S. 55–67.

Lartillot, O. / Toiviainen, P. (2007). A Matlab Toolbox for Musical Feature Extraction from Audio. In: Proceedings of the 10th Int. Conference on Digital Audio Effects (DAFx-07). Bordeaux, France, S. 237–244.

Mathieu, B. / Essid, S. / Fillon, T. / Prado, J. / Richard, G. (2010). YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software. In: Downie, J.S. / Veltkamp, R.C. (Hrsg.). Proceedings of the 11th International Society for Music Information Retrieval Conference. Utrecht, Netherlands: International Society for Music Information Retrieval, S. 441–446.

McEnnis, D. / McKay, C. / Fujinaga, I. (2006). jAudio: Additions and Improvements. In: Proceedings of the International Conference on Music Information Re- trieval. Victoria, Canada, S. 385–386.

Mitrovic, D. / Zeppelzauer, M. / Breiteneder, C. (2010). Features for Content-Based Audio Retrieval. In: Zelkowitz, M.V. (Hrsg.). Advances in computers. Volume 78, Improving the Web. Amsterdam; Boston: Academic Press, S. 71–150.

Niemann, H. (2003). Klassifikation von Mustern. 2. Aufl. Berlin: Springer.

Scheirer, E. / Slaney, M. (1997). Construction and Evaluation of a Robust Multifeature Speech/Music Discriminator. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP '97. Munich, Germany, S. 1331–1334.

Seyerlehner, K. / Pohle, T. / Schedl, M. / Widmer, G. (2007). Automatic Music Detection in Television Productions. In: Proceedings of the 10th Int. Conference on Digital Audio Effects (DAFx-07). Bordeaux, France, S. 221–228.

Tzanetakis, G. / Cook, P.R. (1999). A Framework for Audio Analysis based on Classification and Temporal Segmentation. In: Proceedings of the 25th EUROMICRO '99 Conference, Informatics: Theory and Practice for the New Millenium. Milan, Italy, S. 2061–2069.

Yaafe (2011a). Installing Yaafe. YAAFE - Yet Another Audio Feature Extractor. [http://yaafe.sourceforge.net/manual/install.html (ausgehoben 17.05.2012)].

Yaafe (2011b). Yaafe core features. YAAFE - Yet Another Audio Feature Extractor. [http://yaafe.sourceforge.net/features.html (ausgehoben 17.05.2012)].

7 Abbildungsverzeichnis

Abbildung 1: Unterscheidung von Lachs und Barschen anhand photographischer Aufnahmen. (Duda / Hart / Stork 2001, S.5) 8

Abbildung 2: Typischer Ablauf von Mustererkennung. (nachgezeichnet nach Niemann 2003, S.26) 8

Abbildung 3: Vorgangsweise für die Entwicklung eines Feature Extractors für das CBA-Projekt. 12

Abbildung 4: Liste der ausgewählten und kategorisierten Audiodateien für die Ground Truth (grün = Musiksendung, rot = Sprachsendung). 13

Abbildung 5: Markieren der Bereiche in denen Musik vorhanden ist. 14

Abbildung 6: XML-Struktur der Datei von Sonic Visualizer. 14

Abbildung 7: Unterschiede im Spektrogramm von Musik (links) und Sprache (rechts). (eigene Darstellung) 15

Abbildung 8: Berechnungsschritte des CFA-Features für Musik (links) und Sprache (rechts). Power Spectrum (a, d), Local Peaks (b, e), Binarization (c, f) (eigene Darstellung) 16

Abbildung 9: Spitzen der aufsummierten Frequenzaktivierungen für Musik (links) und Sprache (rechts). (eigene Darstellung) 16

Abbildung 10: Aufteilung der Features in Komponenten und Wiederverwendung bereits berechneter Komponenten zur Laufzeit (Mathieu u. a. 2010) 21

Abbildung 11: Aufteilung des CFA-Features in Komponenten. (eigene Darstellung) 22

Abbildung 12: Ausführungsablauf einer Komponente. (eigene Darstellung) 23

Abbildung 13: Ablauf der Komponente WindowNormalize. 24

Abbildung 14: Ablauf der Komponente DBConversion. 24

Abbildung 15: Ablauf der Komponente SubRunningAverage. 24

Abbildung 16: Ablauf der Komponente Binarization. 25

Abbildung 17: Ablauf der Komponente FrameSum. 25

Abbildung 18: Ablauf der Komponente PeakDetection. (eigene Darstellung) 26

Abbildung 19: Grafischer Vergleich der Berechnungsergebnisse aus Matlab (links) und Yaafe (rechts). (eigene Darstellung) 27

Abbildung 20: Unterschiede in der Berechnung der FFT von kurzen Audiosignalen in Matlab (links) und Yaafe (rechts). (eigene Darstellung) 28

8 Tabellenverzeichnis

Tabelle 1: Berechnungsergebnisse des CFA-Features in Matlab anhand einiger Testdateien. 18
Tabelle 2: Vergleich der CFA-Werte von Matlab und Yaafe. 28

9 Abkürzungsverzeichnis

(in der Reihenfolge ihres Vorkommens im Text)

CBA:	Cultural Broadcasting Archive
AKM:	Autoren, Künstler und Musikverleger
CFA:	Continuous Frequency Activation
LPC:	Linear Predictive Coefficients
ZCR:	Zero Crossing Rate
R-ZC:	Range of Zero Crossings
DWT:	Discrete Wavelet Transform
MFCC:	Mel Frequency Cepstrum Coefficients
SE:	Spectral Entropy
CSE:	Chromatic Spectral Entropy
MAP:	Maximum a Posteriori
GMM:	Gaussian Mixture Models
MLP:	Multilayer Perceptron
RBF:	Radial Basis Functions
HMM:	Hidden Markov Model
SVM:	Support Vector Machine
YAAFE:	Yet Another Audio Feature Extractor
XML:	Extended Markup Language
MIR:	Music Information Retrieval
APT:	Advanced Packaging Tool
FFT:	Fast Fourier Transformation