Synchronisation von Multimediadaten auf Basis von Audiospuren

MASTERARBEIT

zur Erlangung des akademischen Grades
Diplom-Ingenieur

Masterstudium Informatik

Alpen-Adria-Universität Klagenfurt
Fakultät für Technische Wissenschaften

Begutachter: O.Univ.-Prof. Dr. Laszlo Böszörményi
Vorbegutachter: Ass.-Prof. Dipl.-Ing. Dr. Mathias Lux
Institut: Institut für Informationstechnologie

April/2012
Kurzfassung

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende wissenschaftliche Arbeit selbstständig angefertigt und die mit ihr unmittelbar verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle aus gedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für wissenschaftliche Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.

Die während des Arbeitsvorganges gewährte Unterstützung einschließlich signifikanter Betreuungshinweise ist vollständig angegeben.


Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

____________________ ____________________
(Unterschrift) (Ort, Datum)
Inhalt

1 Einleitung .................................................................................................................. 6

1.1 Ziele und Anwendungsgebiete .......................................................................... 6

1.1.1 Poor Man’s Multitrack Recording ............................................................... 8

1.1.2 Dubbing ...................................................................................................... 9

1.1.3 Video Compilations ................................................................................... 10

1.1.4 Online-Videoplattformen ......................................................................... 11

1.1.5 Metadatengenerierung ............................................................................ 12

1.2 Ist-Stand ........................................................................................................... 12

1.2.1 Ähnliche Systeme ..................................................................................... 14

1.2.2 Anforderungen ......................................................................................... 20

2 Einführung in audio-basiertes DSP ......................................................................... 22

2.1 Grundlagen ...................................................................................................... 22

2.1.1 Probleme von Consumer-Geräten ........................................................... 25

2.1.2 Speicherung von digitalen Audiosignalen ................................................ 28

2.2 Verwendete Algorithmen ................................................................................ 30

2.2.1 Dezibel-Berechnungen ............................................................................. 30

2.2.2 Fast-Fourier-Transformation (FFT) ........................................................... 32

2.2.3 Fensterfunktionen .................................................................................... 34

2.2.4 Short-Time-Fourier-Transformation (STFT) .............................................. 36

2.2.5 Cross-Correlation ...................................................................................... 37

2.2.6 Resampling ............................................................................................... 38

3 Umsetzung des Systems ......................................................................................... 40

3.1 Fingerprinting / Audio-Matching ..................................................................... 40

3.1.1 Was ist Fingerprinting? ............................................................................. 41

3.1.2 Schematische Funktionsweise .................................................................. 43

3.1.3 Konkrete Implementierung ...................................................................... 47

3.2 Dynamic-Time-Warping ................................................................................... 56

3.2.1 On-Line-Time-Warping ............................................................................. 58

3.2.2 Audio-optimierte Dynamic-Time-Warping ............................................. 62

3.3 Cross-Correlation ............................................................................................. 63

3.4 Match-Filterung & Angleichung ....................................................................... 65

3.5 Architektur ....................................................................................................... 68
1 Einleitung


1.1 Ziele und Anwendungsbereiche

Ziel dieser Arbeit ist es, ein Verfahren und eine entsprechende Softwareimplementierung zu entwickeln, welche durch die Analyse von Aufnahmen gemeinsame Inhalte bzw. Synchronisationspunkte findet, und zusätzlich in der Lage ist,
die Dateien anhand der gefundenen Punkte so zu synchronisieren, dass sie ohne oder mit höchstens geringen manuellen Eingriffen konsumiert oder weiterverarbeitet werden können. Dabei wird das Anwendungsfeld auf den Amateur- und semiprofessionellen Bereich beschränkt, da im professionellen Umfeld Standards existieren und Verwendung finden, die alle hier beschriebenen und zu lösenden Probleme hinfällig machen.

Um das Ziel zu erreichen bietet sich die Analyse von Audiodaten an, da damit drei entscheidende Vorteile gegenüber der Analyse von Videodaten gegeben sind: Erstens sind Audiodaten bei einer Aufnahme so gut wie immer vorhanden (Consumer-Geräte bieten meistens Funktionen zur Aufnahme purer Audiodaten oder Videos inklusive Audiospur), zweitens sind zweidimensionale Videodaten wesentlich datenintensiver und aufwendiger zu verarbeiten als eindimensionale Audiodaten, und drittens besitzen Mikrofone auf Consumer-Geräten im Regelfall keine oder nur eine schwache Richtcharakteristik, während Videoaufnahmen streng auf ihren Bildausschnitt begrenzt sind.

Die zwei zum Großteil verwendeten Mikrofontypen sind jene mit omnidirektionaler (engl. omnidirectional) Charakteristik, welche den Ton kugelförmig aus jeder Richtung gleich stark aufnehmen, und jene mit Nierencharakteristik (engl. cardioid), welche zwar eine Richtwirkung in ihre Hauptrichtung aufweisen, den Bildausschnitt aber trotzdem weit überragen. Grundsätzlich nehmen Mikrofone Audiowellen immer aus allen Richtungen auf, die Richtcharakteristik sorgt allein dafür, dass Wellen aus unerwünschten Richtungen mehr oder weniger stark gedämpft werden. Wenn also z.B. von ein und demselben Punkt aus zwei Videoaufnahmen erstellt werden, wobei eine Kamera Richtung Norden und eine Richtung Westen ausgerichtet ist, so werden ihre Bildausschnitte vollständig unterschiedlich sein (unter der Annahme dass keine Spezialobjektive verwendet werden). Die Klangbilder werden sich jedoch stark überschneiden, woraus im Nachhinein geschlossen werden kann, dass beide Videos
Einleitung


Abbildung 3: Gängige Richtcharakteristiken von Mikrofonen [1]


1.1.1 Poor Man's Multitrack Recording

Einleitung

definiert sein, der sich entweder daraus ergibt, dass die Aufnahme aller Spuren zur exakt selben Zeit gestartet wird, oder durch die Speicherung eines Timecode, der die nachträgliche Startzeitsynchronisierung ermöglicht. Zweitens müssen die Analog-Digital-Wandler (ADC, engl. Analog-to-Digital-Converter) aller Spuren von einem gemeinsamen Taktgeber versorgt werden. Werden die ADCs der einzelnen Spuren von verschiedenen Takten versorgt, so tritt ein sogenannter Time-Drift (auch Clock-Drift genannt) auf, was bedeutet, dass die Synchronität der Spuren verloren geht, was sich im besten Fall auf die Tonqualität auswirkt, im schlechtesten Fall aber das Resultat ungenießbar macht. Dieser Drift kann linear oder nichtlinear sein und diverse Gründe haben – so kann er zum Beispiel selbst bei baugleichen Taktgebern durch Fertigungstoleranzen, durch Schwankungen in der Versorgungsspannung oder der Umgebungstemperatur auftreten. Es können unter bestimmten Umständen, z.B. durch physikalische Einwirkungen auf mechanische Aufnahmegeräte oder Buffering-Probleme, auch Aussetzer auftreten, wodurch meist kurze Abschnitte verloren gehen. Dadurch wird das kontinuierliche Zeitgefüge in der Aufnahme verletzt, was beim Playback als „Sprünge“ (engl. Skips) auffällt und sich bei der Synchronisierung als nichtlinearer Drift bemerkbar macht. Der nichtlineare Fall tritt erfahrungsmäßig nur sehr selten ein, ist aber dafür umso problematischer.


Da Amateure oft nicht gewillt sind, in teures professionelles Equipment zu investieren, wird meist zu verhältnismäßig günstigen Stereoaufnahmegeräten gegriffen, die den eben beschriebenen Nachteilen unterliegen. Hier wäre es also wünschenswert, wenn die erforderliche Nachbearbeitung auf ein Minimum reduziert oder gänzlich vermieden werden könnte.

1.1.2 Dubbing

Dubbing bzw. Overdubbing bezeichnet im Kontext dieser Arbeit einen Vorgang, bei dem ein Audiosignal durch ein anderes Audiosignal, das zwar grundsätzlich den selben Inhalt enthält, aber von besserer Qualität ist oder aus anderen Gründen bevorzugt wird, ersetzt wird.
Diese Technik wird oft bei Dokumentationen und Interviews angewendet, bei denen eine Szene mit einer Kamera gefilmt wird, und der Ton zusätzlich mit einem externen Mikrofon auf einem Audiorekorder aufgezeichnet wird. Die Gründe dafür können vielfältig sein, etwa weil die in der Kamera eingebauten Mikrofone nicht den gewünschten Qualitätsansprüchen genügen, die Kamerafirmware nicht die benötigten Konfigurationsmöglichkeiten bietet, eine andere Richtcharakteristik zur Tonaufnahme benötigt wird, oder weil vor allem bei Interviews der Ton so nahe wie möglich am Mund des Sprechers abgenommen werden sollte, um Störgeräusche wie z.B. Hall zu vermeiden. In der Nachbearbeitung wird dann die Tonspur der Kamera als Referenz angenommen, an welche die separate Audioaufnahme angeglichen wird. Die Audiospur deshalb, weil sie im Vergleich zu einer Videospur einfacher und qualitativ mit weniger Einbußen in ihrer Geschwindigkeit verändert werden kann.


In beiden Fällen muss man sich also wieder um das Finden eines gemeinsamen Startpunktes, und das Angleichen der Abspielgeschwindigkeit kümmern.

1.1.3 Video Compilations


Veranstaltung anfertigt, zu welchem die einzelnen Videos dann zugeordnet werden. Fertigt man mehr als einen Mitschnitt an verschiedenen Positionen an, so wird zusätzlich die Wahrscheinlichkeit erhöht, dass übereinstimmende Punkte mit den Videos gefunden werden. Außerdem wären mehrere Mitschnitte zwingend notwendig, falls die Veranstaltung auf mehrere Bereiche aufgeteilt ist, welche audiotechnisch unabhängig beschallt werden. Zwar läuft man hier wieder in das Problem, dass man entweder professionelles Equipment benötigt, um diese Mastertracks synchron aufzunehmen, dass man die Synchronisierung manuell vornehmen muss, oder dass man den zu entwickelnden Algorithmus benötigt, um diese Spuren wieder anhand übereinstimmender Punkte zu synchronisieren. Falls sich die Veranstaltung durch die gesammelten Videos nicht durchgehend rekonstruieren lässt, so sollte das Material in den meisten Fällen zumindest für eine Zusammenfassung reichen, was jedoch in jedem Fall mit manueller Nachbearbeitung verbunden wäre.

Ein großes Problem bei dieser Idee wird jedoch bleiben, an die Videoaufnahmen der Besucher zu gelangen. Hierzu müsste man einen Mechanismus finden, der die Benutzer dazu motiviert, ihre Clips z.B. über eine Internetplattform an das System zu übermitteln. Diese Motivation werden die Besucher jedoch erfahrungsgemäß nur dann finden, wenn sie darin einen Vorteil erkennen, den sie ansonsten vermissen würden.

1.1.4 Online-Videoplattformen


1.1.5 Metadatengenerierung

Besitzt man eine Referenzspur, an welche andere Clips angeglichen werden sollen (wie im Fall von Video Compilations), so ist es in bestimmten Fällen möglich, fehlende Metadaten zu ermitteln und fehlerhafte Daten zu korrigieren. Angenommen die Referenzspur wurde automatisch oder manuell mit stattfindenden Ereignissen annotiert (z.B. Songtitel bei Konzertmitschnitten), so kann man daraus verschiedene Metadaten für die einzelnen angeglichenen Clips ermitteln. Denkbar wäre z.B. das Datum bzw. die Uhrzeit, an welchem die Aufnahme eines Clips begonnen wurde, oder der Inhalt des Clips (z.B. der Titel des aufgenommenen Songs bei einem Konzert). Der umgekehrte Weg, nämlich den Inhalt eines Referenztracks aus einer Menge an annotierten Clips zu gewinnen, wäre natürlich ebenfalls möglich, ebenso wie die Übertragung von Metadaten eines Clips auf einen anderen parallel aufgenommenen Clip. Sollten die Metadaten bei einem Paar paralleler Clips im einen vorhanden sein, und im andere fehlen, so kann man die Metadaten einfach übertragen. Sollten beide Clips Metadaten besitzen stellt sich jedoch die Frage, welche Daten die korrekten bzw. korrekteren sind, also welche man für die Übertragung auf den jeweils anderen Clip auswählt. Hierbei ist es grundsätzlich fraglich, ob dieser Vorgang automatisiert werden kann, aber zumindest könnten dadurch automatisiert Vorschläge zur manuellen Metadatenannotation gewonnen werden.

1.2 Ist-Stand


Abbildung 4: Hüllkurve (rot) einer Waveform (Quelle: Wikimedia / Creative Commons CC0 1.0)

Um zwei Audiospuren miteinander zu synchronisieren, lädt man sie in einen Mehrspur(audio)editor, sieht sich im einfachsten Fall die Hüllkurven der Tracks, bzw.

\[2\] engl. NLE (non-linear editing system), z.B. Adobe Premiere Pro, Sony Vegas Pro; bzw. DAW (digital audio workstation), z.B. Audacity, Adobe Audition, Avid Pro Tools, Apple Logic, Steinberg Nuendo und Cubase, REAPER
deren Peaks (Spitzen) an und erkennt anhand der Struktur (validiert durch eine Hörprobe) einen gemeinsamen Anfangspunkt, um die zweite Spur synchron zur ersten zu platzieren.

Danach prüft man durch eine weitere Hörprobe, ob die Spuren auch an ihrem Ende noch akustisch synchron sind. Sollten die Spuren mehrere Minuten lang sein, wird man in den meisten Fällen feststellen, dass sie bedingt durch den Time-Drift nicht mehr synchron sind, was das Suchen eines gemeinsamen Endpunktes nach sich zieht. Dabei wird die Länge der zweiten Spur durch eine Geschwindigkeitsänderung an die erste Spur angeglichen, so dass die Synchronität auch am Ende gegeben ist. Als nächstes müssen über die gesamte gemeinsame Laufzeit visuelle und/oder akustische Proben durchgeführt werden, um zu kontrollieren, ob es sich um einen linearen oder nichtlinearen Time-Drift handelt. Im Falle des linearen Drifts ist die Arbeit abgeschlossen, bei einem nichtlinearen Drift muss die zweite Spur in mehrere Teilabschnitte aufgeteilt werden, welche wiederum individuell an die erste Spur angeglichen werden müssen. Dabei ist darauf zu achten, dass die Übergänge zwischen den Abschnitten akustisch makellos abgespielt werden können, d.h. Teile der ursprünglich zusammenhängenden Spur dürfen weder wiederholt, noch ausgelassen werden.

Abbildung 5: Zwei anhand ihrer Hüllkurven synchronisierte Audiospuren

Die Methode der Hüllkurvensynchronisierung kann jedoch nur angewandt werden, solange der Inhalt der Spuren über einen ausreichend großen Dynamikumfang verfügt, wodurch optisch eindeutig zuordnungsfähige Muster entstehen. Je höher die aufgenommene Lautstärke, desto mehr nimmt vor allem bei Consumer-Geräten der Dynamikumfang ab, was durch Techniken wie automatic gain control (dt. automatische Verstärkungsregelung) drastisch verschlimmert wird. Dadurch kann es passieren, dass es optisch unmöglich wird, gemeinsame Abschnitte zu finden, da die Hüllkurven sich immer mehr einer geraden Linie annähern. In diesem Fall muss man sich auf eine akustische Suche beschränken, die den Zeitaufwand erheblich in die Höhe treibt.

Abbildung 6: Zwei optisch unsynchronisierbare Hüllkurven

Fließen Videospuren in die Synchronisierung mit ein, so sollte man um jeden Preis verhindern, dass ihre Geschwindigkeit verändert wird. Im Endeffekt bedeutet dies,
dass man Audiospuren ohne große Qualitätseinbußen an die Videogeschwindigkeit anpassen kann, umgekehrt kann dies jedoch selbst von Laien erkennbare Qualitätseinbußen nach sich ziehen. Um eine oder mehrere Audiotracks an ein einziges Video anzulegen, verfährt man im Prinzip gleich wie im Fall ohne Video, nimmt jedoch die Audiospur des Videos als Referenz und gleicht alle andere Tracks daran an. Komplizierter wird es, falls mehrere durchgängige Videos verfügbar sind. Da in diesem Fall auch die einzelnen Videos zeitlich auseinanderdriften können, man ihre Geschwindigkeit aber nicht ändern sollte, umgeht man das Problem durch geeignete Schnitttechniken, wodurch der Drift unbemerkt wird. Audiospuren werden entweder zuerst an ein als Referenz angenommenes Video, oder nach dem Schnitt an das resultierende Video angeglichen.

1.2.1 Ähnliche Systeme


1.2.1.1 Adobe Premiere Pro

Die Videoschnittsoftware Premiere Pro von Adobe bietet neben einer für das professionelle Umfeld hilfreichen Funktion zur Clipsynchronisierung per Timecode auch die Möglichkeit, Clips durch das manuelle Setzen von Marken zu synchronisieren. Hierfür sucht man sich in den Audiospuren übereinstimmende Punkte, setzt jeweils einen nummerierten Marker mit gleicher Nummer und wählt dann aus dem Kontextmenü den Eintrag „Synchronisieren“. Im folgenden Fenster steht die Option zur Synchronisierung anhand der Marker zur Auswahl.

Im Endeffekt wird der Synchronisierungsvorgang dadurch aber keineswegs automatisiert, sondern es resultiert daraus lediglich eine geringe Zeiteinsparung, die man andernfalls für das manuelle Positionieren der Clips in der Timeline aufbringen müsste.

– entspricht der vollständig manuellen Synchronisation

1.2.1.2 Music Alignment Tool Chest (MATCH)


---

3 Steuerelement, in welchem die einzelnen Spuren angezeigt werden, in denen man Audio- und Videoclips zeitlich positionieren kann

Abbildung 7: MATCH Hauptfenster


+ automatisiertes Finden von übereinstimmenden Abschnitten
− lange Berechnungsdauer
− kein Support von Videos
− kein gleichzeitiges Abspielen mehrerer Dateien
− keine Ausgabe von synchronisierten Dateien

1.2.1.3 Sonic Visualizer MATCH Vamp Plugin


---


+ automatisiertes Finden von übereinstimmenden Abschnitten
− lange Berechnungsdauer
− kein Support von Videos
− kein gleichzeitiges Abspielen mehrerer Dateien
− keine Ausgabe von synchronisierten Dateien

1.2.1.4 VocALign


⁵ [http://www.synchroarts.com]

+ gute Synchronisierung von Sprache  
+ nicht-lineare Time-Drift-Korrektur  
+ Pitch (Tonhöhe) von synchronisierter Fassung unverändert  
− schlechte Synchronisierung von Musik  
− Clips müssen am Anfang und am Ende jeweils den gleichen Inhalt haben (d.h., keine Angleichung von einer kurzen Aufnahme an einen Abschnitt einer längeren Aufnahme möglich; sonst Vorverarbeitung notwendig)  
− 120 Sekunden Längenbeschränkung (sonst Vorverarbeitung notwendig)  
− Angleichung nur paarweise möglich
1.2.1.5 PluralEyes / DualEyes


+ unterstützt beliebige Anzahl von Clips
+ gute Synchronisierung
± akzeptable Erkennungsrate (kein Vergleichswert vorhanden)

6 http://www.singularsoftware.com
Einleitung

- langsamer Vorverarbeitung (Generierung temporärer Dateien)
- vielfache Verarbeitungsdauer im „Try Really Hard“-Modus (mit nur leicht verbesserter Erkennungsrate)


Abbildung 11: DualEyes Hauptfenster nach erfolgter Synchronisierung

+ unterstützt beliebige Anzahl von Clips
+ gute Synchronisierung
+ Time-Drift-Korrektur
± akzeptable Erkennungsrate (kein Vergleichswert vorhanden)
- langsamer Vorverarbeitung (Generierung temporärer Dateien)

\(^7\) http://www.singularsoftware.com

\(^8\) digitale Spiegelreflexkameras
vielfache Verarbeitungsdauer im „Try Really Hard“-Modus (mit nur leicht verbesserter Erkennungsrate)
− keine nicht-lineare Time-Drift-Korrektur
− keine Synchronisierung von Audiodateien

1.2.2 Anforderungen

Ausgehend von der zuvor ausgeführten Funktionsanalyse bestehender Software werden an dieser Stelle Anforderungen an eine Software gestellt, welche die in diesem Kapitel beschriebene Problematik zumindest großteils lösen sollte:

Unterstützung von beliebigen Dateiformaten

Als Eingabe sollen alle anerkannten Audio- und Videodateiformate unterstützt werden und direkt gelesen werden können, ohne dass eine Vorverarbeitung durchgeführt werden muss.

Finden von übereinstimmenden Abschnitten

Der Inhalt aller eingegebenen Audiodateien muss analysiert und untereinander verglichen werden, so dass übereinstimmenden Abschnitte in jeder beliebigen Konstellation von Dateien gefunden werden. Die Längen der Dateien und ihre Anordnung müssen dabei unberücksichtigt bleiben.

Hohe Erkennungsrate

Die Erkennungsrate von übereinstimmenden Punkten sollte mindestens der von VocALign und DualEyes entsprechen, d.h. es müssen Testfälle erstellt werden, in welchen die Software gleich oder besser als diese zwei Produkte abschließen sollte.

Manuelles Anlegen von Synchronisierungspunkten

Für den Fall dass übereinstimmende Punkte schon bekannt sind, oder vorliegende Punkte nicht gefunden werden, muss der Benutzer die Möglichkeit haben, Übereinstimmungen manuell zu definieren.

Keine Beschränkungen

Keine Beschränkungen von Dateigrößen und –längen und Verarbeitung von beliebig vielen Tracks. Limitierender Faktor sollte nur die eingesetzte Hardware sein.

Akzeptable Verarbeitungsdauer

Die Dauer des Synchronisierungsvorgangs muss kurz genug sein, damit ein fühlbarer Mehrwert über die manuelle Synchronisation entsteht. Die Dauer sollte relativ zur gesamten Länge der Eingabedateien höchstens linear ansteigen.

Erkennung und Korrektur von linearem und nicht-linearem Time-Drift

Time-Drift jeglicher Form sollte erkannt und korrigiert werden können.
Einleitung

Pitch-Beibehaltung
Die Tonhöhe darf nach einer Time-Drift-Korrektur nicht merkbar verschoben sein. Auch bei nichtlinearerem Driftaußgleich muss die Tonhöhe durchgehend konstant bleiben.

Visualisierung der Synchronisation
Die zu synchronisierenden bzw. synchronisierten Clips sollten in einer DAW-ähnlichen Timeline angezeigt werden, sodass man ihre zeitliche Anordnung am ersten Blick erkennen kann. Zusätzlich sollten alle Punkte markiert werden, an denen eine Synchronisierung stattgefunden hat.

DAW-ähnliches User-Interface zur Analyse des Ergebnisses
Die synchronisierten Clips sollten nicht nur zeitlich richtig angezeigt werden, sondern es sollte wie in einer DAW üblich, die Wellenform visualisiert werden. So ist es in den meisten Fällen möglich zu erkennen, wie genau die Synchronisierung ausgeführt wurde. Es muss eine Möglichkeit geben, die Clips parallel abzuspielen, um einen akustischen Qualitätseindruck der Synchronisierung zu bekommen. Es sollte außerdem die Möglichkeit geben, die Clips selektiv abzuspielen.

Export zu NLEs und DAWs
Das Ergebnis der Synchronisation sollte exportiert werden können, sodass es in anderen Editoren geladen und verwendet bzw. weiterbearbeitet werden kann.
2 Einführung in audio-basiertes DSP


2.1 Grundlagen


In unserer analogen Welt kann sich ein Geräusch aus Tönen mit beliebigen Amplituden und Frequenzen zusammensetzen, wobei die Grenzen nur durch bestehende physikalische Grundgesetze gezogen werden. Konkret bedeutet das, dass z.B. die Lautstärke eines Tons beliebig groß oder klein sein kann, solange sie sich im Rahmen unserer Physik bewegt. Der Dynamikumfang, also der Unterschied zwischen der minimalen und maximalen Amplitude, kann im Extremfall nahe Unendlich liegen. Auf dieses Problem stößt man spätestens, wenn man Audiosignale mit technischen Apparaten verarbeiten (z.B. aufzeichnen) will, da hier sowohl in der Analog- als auch Digitaltechnik durch elektrische Bauteile, Schaltungen und Kodierungen unumgängliche Limitierungen auftreten. Trivial formuliert kann jeder elektrische
Schaltkreis nur einen bestimmten endlichen Dynamikumfang und endliches Frequenzspektrum verarbeiten, wobei die jeweiligen Untergrenzen je nach Aufbau variieren und somit Schaltkreise für verschiedenste Anwendungszwecke erstellt werden können.

Abbildung 12: Audioverarbeitungskette (diese Arbeit betrifft den DSP-Bereich)


Abbildung 13: Umwandlung eines analogen in ein digitales Signal

Um das analoge in ein digitales Audiosignal umzuwandeln, wird die Amplitude des Signals entlang der Zeitachse mit gleichbleibendem Zeitabstand abgetastet (gesampled, von engl.: sampling) und entlang der Wertachse quantisiert [8]. Die durch Sampling und Quantisierung ermittelten diskreten Werte werden Samples genannt, wobei ein Sample einem einzelnen binären Zahlenwert entspricht. Dabei gibt die Samplingrate an, wie viele Samples pro Sekunde vom ADC ermittelt werden. Sie steht in direktem Zusammenhang mit der maximal abbildbaren Frequenz, welche definiert durch das Nyquist-Shannon-Abtasttheorem [9] kleiner der halbierten Samplingrate sein muss \((f_{abtast} > 2 f_{max})\). Die halbe Samplingrate wird als Nyquist-Frequenz bezeichnet, der Obergrenze des abbildbaren Frequenzspektrums. Sollte das aufzuzeichnende Signal Frequenzen beinhalten, die größer oder gleich der Nyquist-Frequenz sind, müssen diese mittels eines Tiefpassfilters entfernt oder die Samplingrate angehoben werden. Konkret bedeutet das, dass man für eine digitale Abbildung des für den Menschen hörbaren Frequenzbereichs, welcher bis zu einer Obergrenze von ca. 20000 Hz reicht, eine Samplingrate mit mehr als 40000 Hz benötigt. Dies ist auch der Grund weshalb sie für Audio-CDs auf 44,1 kHz festgelegt wurde [10]. Die zweite neben der Samplingrate wichtige Eigenschaft eines digitalen Audiosignals ist seine Auflösung, definiert durch die Bitrate, die in Relation zum Dynamikumfang steht\(^{10}\). Das analoge Signal, das zwischen den minimalen und maximalen vom ADC unterstützten Amplitudenwerten unendlich viele verschiedene Werte annehmen kann, wird durch die Quantisierung an eine endliche Folge von diskreten Zahlenwerten (Quantisierungsstufen) angeglichen. Die Anzahl der möglichen Zahlenwerte wird dabei durch die vom ADC unterstützte Bitrate beschränkt.

<table>
<thead>
<tr>
<th>Gerät</th>
<th>Samplingrate (kHz)</th>
<th>Bitrate (Bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAT-Rekorder</td>
<td>32; 44,1; 48; 96</td>
<td>12; 16; 24</td>
</tr>
<tr>
<td>Diktiergerät</td>
<td>8 – 44,1</td>
<td>8 - 16</td>
</tr>
<tr>
<td>MiniDisc-Rekorder</td>
<td>44,1</td>
<td>16 - 24</td>
</tr>
<tr>
<td>CD-Rekorder</td>
<td>44,1</td>
<td>16</td>
</tr>
<tr>
<td>DV-Kamera (MiniDV, Digital8)</td>
<td>32 / 48</td>
<td>12 / 16</td>
</tr>
<tr>
<td>Mobiltelefon</td>
<td>8 - 48</td>
<td>8 - 16</td>
</tr>
<tr>
<td>Digicam (Fotokamera)</td>
<td>8 - 48</td>
<td>8 - 16</td>
</tr>
</tbody>
</table>

Tabelle 1: digitale Aufnahmegeräte und ihre unterstützten Aufnahmeformate

Eine Quantisierung mit 3 Bit kann somit acht verschiedene Ausgangswerte erreichen \((2^3 = 8)\), typische Werte sind jedoch weit höher, wie z.B. 8 Bit in vielen Sprachanwendungen oder 16 Bit bei der Audio-CD. Das Mapping der analogen auf die

\(^{10}\) Gilt nur für den hier beschriebenen herkömmlichen „Nyquist-ADC“. Der Sigma-Delta-Modulation-ADC (SDM-ADC) gibt nur ein 1-Bit Signal aus, schlägt aber unter Umständen herkömmliche ADCs [12,56].

Abbildung 14: analoges Signal, digitalisiert mit 12 Samples / 9 Quantisierungsstufen, digitalisiert mit 24 Samples / 18 Quantisierungsstufen

2.1.1 Probleme von Consumer-Geräten

Nachdem der Aufnahmevorgang von Audiosignalen nun skizziert wurde, wird es Zeit, auf die unterschiedlichsten Probleme von Consumer-Geräten einzugehen, die bei Aufnahmen auftreten können. Alle Probleme haben dabei gemein, dass sie zu einer verminderten Aufnahmequalität führen, und sie somit nicht nur den menschlichen Hörgenuss trüben, sondern auch die automatische Analyse erschweren.


Ein weiteres großes Problem sind eingebaute Mikrofone. In den meisten Geräten werden Mikrofone eingebaut, die mit professionellen Mikrofonen nicht im Ansatz vergleichbar sind. Sie sind sehr günstig zu beziehen, können dafür hohe Schallpegel und/oder hohe Frequenzen nicht aufnehmen und sind oft an sehr ungünstigen Stellen positioniert. Hohe Schallpegel sind hierbei der Faktor, der die Aufnahmequalität am


Ein Nachteil, der erst unter Extrembedingungen zum Problem wird, ist das Grundrauschen, welchem jede elektrische Schaltung unterliegt. Sowohl das Mikrofon, als auch der Verstärker sowie der ADC, und auch die gesamte restliche elektrische Schaltung des Geräts unterliegen jeweils diesem Rauschen, und das schwächste Glied

Die Kombination von schlechten und/oder schlecht platzierten Mikrofonen, hohem Grundrauschen und automatischer Verstärkungsregelung kann also in vielen Situationen dazu führen, dass die Aufnahmequalität leidet. In einigen Situationen führt es leider zum Worst Case, bei dem die resultierende Aufnahme gänzlich unbrauchbar wird.

trotzdem mit der Samplingrate R verarbeitet, was dazu führt dass die Aufnahme im Vergleich zur Quelle ein wenig langsamer oder schneller abläuft. Diese Abweichung bleibt bezogen auf die einzelne Aufnahme unbemerkt, erstellt man jedoch gleichzeitig mehrere Aufnahmen mit unterschiedlichen Geräten, die jeweils von einer eigenen Sampling-Clock getaktet werden, ist es unmöglich die resultierenden Signale zu synchronisieren, es sei denn, alle Geräte unterliegen zufällig dem gleichen Drift. Würde man die verschiedenen Signale an einem Anfangspunkt exakt synchronisieren, so würden sie über die Zeit hinweg stetig asynchroner werden (siehe Abbildung 15). Erschwerend kommt hinzu, dass auch Spannungsschwankungen oder Veränderungen in der Umgebungstemperatur Einfluss auf die Sampling Clock haben, und der Drift durch diese Schwankungen nicht-linear verändert wird.

### 2.1.2 Speicherung von digitalen Audiosignalen


<table>
<thead>
<tr>
<th>Sampleformat</th>
<th>Minimum</th>
<th>Mittelwert</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Bit Unsigned Integer</td>
<td>0</td>
<td>128</td>
<td>255</td>
</tr>
<tr>
<td>8-Bit Signed Integer</td>
<td>-128</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>16-Bit Signed Integer</td>
<td>-32768</td>
<td>0</td>
<td>32767</td>
</tr>
<tr>
<td>24-Bit Signed Integer</td>
<td>-8388608</td>
<td>0</td>
<td>8388607</td>
</tr>
<tr>
<td>32-Bit IEEE Float</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Tabelle 2: Verschiedene Sampleformate und ihre Wertebereiche

Der erste Schritt, um die PCM-Rohdaten im Computer zu speichern, ist der Transfer in den Hauptspeicher und die damit einhergehende Codierung in Form der kleinsten adressierbaren Speichereinheiten, den Bytes. Dabei gibt es mehrere Möglichkeiten. Einerseits bei der Codierung der Samplewerte, die in Form von vorzeichenlosen Ganzzahlen (Unsigned Integer), vorzeichenbehafteten Ganzzahlen (Signed Integer), oder vorzeichenbehafteten Fließkomma-Zahlen (IEEE Float [13]) vorliegen können. Andererseits gibt es mehrere Möglichkeiten bei der Art und Weise, wie die Samplewerte in eine Abfolge von Bytes konvertiert werden können. Der einfachste Fall ist ein 8-Bit PCM Sample, das direkt in einem Byte abgelegt werden kann. Etwas schwieriger wird es bei einem 16-Bit Sample, das auf zwei Bytes aufgeteilt werden muss. Hierbei gibt es zwei Möglichkeiten der Byte-Reihenfolge: Bei Big-Endian werden das Byte mit den höchstwertigen Bits zuerst und das Byte mit den niederwertigsten Bits zuletzt gespeichert, Little-Endian beschreibt den umgekehrten Fall. Auf Grund der vorherrschenden Marktstellung der Intel x86-Architektur wird die damit kompatible
Little-Endian-Richtung bevorzugt. Bei Samples, deren Bitanzahl kein vielfaches von acht ist, wie es z.B. bei 12- oder 20-Bit ADCs der Fall ist, werden die Werte entweder links- oder rechtsbündig in eine nächstgrößere passende Bytefolge geschrieben. Die letzte mögliche Variante dient der Einfachheit und Performanceoptimierung, wobei die Samples in einem Vielfachen der Architektur-Wortbreite ausgerichtet werden. So können z.B. für 20- oder 24-Bit Samples, die nur 3 Bytes benötigen würden, 4 Bytes reserviert werden, womit eine Verarbeitung auf 32- und 64-Bit Prozessoren für den Preis eines erhöhten Speicherbedarfs effektiver durchgeführt werden kann.

Abbildung 16: Beispiele für bytecodierte Rohdatensamples

Audio Coding (AAC), welche eine Reduktion auf ca. 20% bis 10% der Ursprungsgröße erreichen, bei subjektiv unverminderter Tonqualität.


Besteht die Aufnahme aus mehreren Kanälen, z.B. Stereo mit einem linken und einem rechten Kanal, oder 5.1 Surround mit 6 Kanälen, so werden die Samples der einzelnen Kanäle im Interleaving-Verfahren gespeichert. Dazu werden zuerst alle ersten Samples der jeweiligen Kanäle abgelegt, danach alle zweiten Samples, usw. Die Samples der verschiedenen Kanäle, die den gleichen Zeitpunkt im Signal beschreiben, werden dabei zu Sample-Blöcken zusammengefasst. Im Stereo-Modus folgt immer das rechte dem linken Sample, bei mehreren Kanälen ist die Lautsprecherbelegung der Kanäle über Zusatzinformationen in den Metadaten persistiert [15].

### 2.2 Verwendete Algorithmen

Nachdem die möglichen Rohdatenformate der Audiosignale nun bekannt sind, folgt hier eine Einführung in diverse Algorithmen und Methoden, die in den nachfolgenden Kapiteln wiederholt angewandt werden. Sie werden hier nur konzeptuell erklärt, da die finale Implementierung von der eingesetzten Audiopipeline bzw. dem zu Grunde liegenden Rohdatenformat abhängt. Außerdem haben sie wie die Mehrheit der DSP-Algorithmen gemein, dass sie nur auf unkomprimierten Daten eingesetzt werden können.

#### 2.2.1 Dezibel-Berechnungen

Um die Lautstärke eines Signals zu messen, gibt es verschiedene Skalen. Die bekanntesten davon sind Dezibel (dB), Phon und Sone [7]. Alle drei messen verschiedene Arten der Lautstärke: dB misst den Unterschied zwischen Amplituden, Phon vergleicht die Lautstärke verschiedener Frequenzen mit gleicher Amplitude, und Sone vergleicht die Lautstärke verschiedener Geräusche. Die vom Menschen wahrgenommene Lautstärke ist eine psychoakustische Größe, die von mehr als nur der

Wenn im weiteren Verlauf dieser Arbeit von Lautstärke gesprochen wird, ist damit nicht diese subjektive psychoakustische Größe gemeint, sondern rein der objektive Amplitudenunterschied zwischen Signalen bzw. Signallevel. Dafür wird die Einheit dB verwendet, welche als Logarithmus zur Basis 10 des Verhältnisses zweier Leistungen definiert ist [16]. Die aus dieser Definition resultierende Größe ist eigentlich dimensionslos, zu Ehren des Erfinders des modernen Telefons, Alexander Graham Bell, wird sie jedoch mit Bel bezeichnet. Um mit angenehmeren Zahlen arbeiten zu können, erfolgen die Angaben nicht direkt in Bel (B), sondern Dezibel (dB), also in Zehntelschritten. Die Angabe in dB bietet den Vorteil, dass die unter Umständen viele Größenordnungen umfassenden Werte der Samples mit handlichen Zahlenwerten dargestellt werden können. Unter der Annahme von 24-Bit Signed-Integer-Samples beträgt der Unterschied zwischen den weit auseinanderliegenden Werten 8000000 und 10 genau 7999990, oder 118 dB, der Unterschied zwischen 10 und 5 genau 5, oder 6 dB.

\[ a = 20 \times \log_{10} \left( \frac{U_1}{U_2} \right) \text{ dB} \]

Formel 1: Berechnung des dB-Abstands zweier Spannungen U₁ und U₂


<table>
<thead>
<tr>
<th>Größe</th>
<th>Verdoppelung</th>
<th>Faktor für dB-Berechnung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lautstärke (subjektiv)</td>
<td>+10 dB(^{11})</td>
<td>33,22</td>
</tr>
<tr>
<td>Schalldruck (objektiv)</td>
<td>+6 dB</td>
<td>20</td>
</tr>
<tr>
<td>Schallintensität/Leistung</td>
<td>+3 dB</td>
<td>10</td>
</tr>
</tbody>
</table>

Tabelle 3: Pegelgrößen

Wenn das lineare analoge Signal mit linearer Quantisierung digitalisiert wird, kann die Spannungsformel direkt in die digitale Domäne übernommen werden, statt Spannung

\(^{11}\) Ungefähre Erfahrungswert aus der Psychoakustik, gilt ebenfalls für den Faktor.
in Volt werden dann die Samplewerte zur Berechnung herangezogen. Um den digitalen Pegel zu berechnen, wird als Referenzwert der minimal bzw. maximal mit der entsprechenden Bitanzahl darstellbare Wert verwendet (siehe Tabelle 2). In diesem System ist der Pegelwert der maximalen Aussteuerung 0 dB, alle geringeren Werte werden mit negativem Pegel angeschrieben. Die Einheit dieser Skala wird als dBFS (Dezibel Full Scale) bezeichnet, und findet standardmäßig in vielen digitalen Audiosystemen Verwendung. Weißt das Signal keine Amplitude auf (entspricht üblicherweise dem Mittelwert 0), kann kein Pegel berechnet werden, da der Logarithmus von 0 nicht definiert ist. In solch einem Fall wird der Pegel von vielen Anzeigen als minus unendlich (−∞) angegeben.

2.2.2 Fast-Fourier-Transformation (FFT)


Abbildung 18: Spektralanalyse eines Audiosignals

Generell wird bei der Fourier-Transformation eine Sequenz von $n$ Werten der Zeitdomäne zu $n$ Werten (Bins) der Frequenzdomäne umgewandelt, wobei die Werte durch komplexe Zahlen repräsentiert werden. Im Unterschied zur DFT, welche eine Eingabe beliebiger Länge akzeptiert, muss der FFT eine Eingabe der Länge $n = 2^m$ vorliegen. Eine Eingabe, die nicht dieser Längeneinschränkung genügt, kann durch Auffüllen auf eine der nächstgrößeren Zweierpotenz entsprechenden Länge transformiert werden.

\textsuperscript{12} Mit den inversen Transformationen iDFT bzw. iFFT kann das Spektrum zurück in das Ursprungssignal gewandelt werden. Durch die Kombination beider Transformationsrichtungen können die Frequenzanteile eines Signal manipuliert werden, wie z.B. bei einem Equalizer.

Abbildung 19: Spektralberechnung eines mit 1000 Hz gesampelten Signals

Das Ergebnis der Fourier-Transformation enthält selbst keine absoluten Informationen über die im Signal vorkommenden Frequenzen, aus dem Ergebnis kann also nicht direkt geschlossen werden, ob z.B. der Kammerton a\(^1\) mit 440 Hz Teil des Signals ist. Dies ergibt sich erst im Zusammenhang mit der Samplingrate. Ein mit einer Rate von 2000 Hz gesampeltes Signal kann wie in Kapitel 2.1 beschrieben nur Frequenzen von 0 Hz bis 1000 Hz abbilden. Transformiert man 2000\(^1\) Samples aus diesem Signal in das entsprechende Spektrum, so erhält man 1000 Werte (Bins), die den Anteil jeder einzelnen Frequenz am Signal enthalten. Genaugenommen enthalten die Werte nicht nur die Anteile der ganzzahligen Frequenzen, sondern das Intervall aller Frequenzen zwischen zwei ganzzahligen Frequenzen. Der zweite Wert im Ergebnis repräsentiert also alle Frequenzen \( F \) mit 0 Hz < \( F \leq 1 \) Hz, der dritte Wert alle mit 1 Hz < \( F \leq 2 \) Hz, usw. Würde man nur 500 Samples transformieren, so würde man trotzdem die Frequenzverteilung über das gesamte Band erhalten, wobei jeder Wert nun 4 Hz umspannen würde. Der zweite Wert gäbe also den Anteil aller Frequenzen 0 Hz < \( F \leq 4 \) Hz an. Umgekehrt führt die Annahme, dass die 2000 Samples nicht mit 2000 Hz, sondern mit 4000 Hz gesampelt wurden, dazu, dass jeder Wert nun ein Intervall von 2 Hz repräsentieren würde und das Spektrum von 0 Hz bis 2000 Hz verläuft. Die Frequenzauflöschung kann erhöht werden, indem man die zu transformierenden Samples mit Nullwerten verlängert, wobei es unerheblich ist, ob die Nullen vorangestellt oder nachgestellt werden. Nimmt man 2000 mit einer Rate von 2000 Hz

\(^1\) Die Werte in diesem Beispiel sind der Einfachheit halber in Dekaden angegeben. Bei der Verwendung von FFT müssen in Wirklichkeit 2048 Samples zur Berechnung herangezogen werden.
gesampelte Werte und verlängert sie durch Nullen auf 4000 Werte, so beträgt die Auflösung des resultierenden Spektrums 0,5 Hz und wurde damit verdoppelt. Der erste Wert im Ergebnis, der Anteil der Frequenz mit 0 Hz, ist ein Spezialfall und wird als DC-Offset bezeichnet. Er gibt die Abweichung des arithmetischen Mittels vom Mittel- bzw. Neutralwert des Samplewertbereichs an.

Abbildung 20: DC-Offset


2.2.3 Fensterfunktionen

Wie im vorhergehenden Abschnitt angesprochen, dient die FFT dazu, periodische Signale zu analysieren und ihr Frequenzspektrum zu berechnen. Das Resultat der Berechnung ist daher nur dann exakt, wenn der transformierte Abschnitt genau eine einfache oder mehrfache Periodenlänge enthält. Dies ist erfüllt, wenn man zwei Kopien dieser Abschnitte hintereinanderreihen kann, so dass im entstehenden Signalabschnitt und seiner ersten Ableitung keine Unstetigkeit entsteht.

Sollte diese periodische Fortsetzbarkeit nicht erfüllt sein, tritt der sogenannte Leck-Effekt (engl. leakage) auf. Dabei wird die Energie der Frequenzen mit unvollständigen Perioden auf benachbarte Frequenzen verteilt [18], Spektrallinien werden ausgeweitet und unter Umständen Seitenbänder verstärkt. Damit enthält das Ergebnis Frequenzkomponenten, die im Eingangssignal nicht vorhanden sind, das Resultat wird verfälscht und das Grundrauschen steigt. Schön zu erkennen ist dieser Effekt in Abbildung 22, wo das Spektrum über das gesamte Band hinweg Frequenzenergie anzeigt, obwohl das Ursprungssignal nur eine einzige Frequenz enthält. Im Gegensatz dazu ist in Abbildung 21 kein Leck-Effekt erkennbar, da die Messdaten vollständige Perioden enthalten.

14 Der Begriff stammt aus der elektrischen Gleichstromtechnik (DC, engl. direct current) und beschreibt eine unerwünschte Grundspannung im Übertragungskanal.

Beispiele für Fensterfunktionen sind die in vielen DAWs integrierten Rechteck-, Dreieck, Hann- und Hammingfenster, wobei die zwei letztgenannten vielfach als Standardfenster eingesetzt werden. Es existiert jedoch noch eine ganze Reihe an Alternativen [20]. Je nach eingesetzter Funktion unterscheidet sich das Spektrum hinsichtlich der Auflösung, Stärke und Breite von Spektrallinien der Hauptbänder und ihrer unerwünschten Seitenbänder. Einzig das Rechteckfenster hat keinen Einfluss auf das Spektrum, solange es auf die gesamten Eingabedaten angewandt wird. Abbildung
24 zeigt das in Abbildung 22 analysierte Signal, nachdem es mit der Hann-Funktion multipliziert wurde. Das Ergebnisspektrum ist im Vergleich zur fensterlosen Analyse um Einiges repräsentativer, jedoch nicht optimal, da der Leck-Effekt nicht vollständig ausgemerzt werden kann.

2.2.4 Short-Time-Fourier-Transformation (STFT)

In einigen Fällen interessiert einen nicht nur die globale Frequenzverteilung in einem Signal oder Signalabschnitt, sondern auch die Änderung des Frequenzverlaufs über die Zeit, z.B. um bestimmte Muster im Signal zu finden. Diese Analyse wird mit Hilfe der Short-Time-Fourier-Transformation durchgeführt. Dazu teilt man das Signal in Abschnitte der Länge $N$ auf, multipliziert sie mit einem Fenster (Windowing) und wendet darauf die FFT an [8].

Die Zeit- und Frequenzauflosung der STFT sind invers proportional, das heißt mit einer Verfeinerung der Frequenzauflosung verringert sich die zeitliche Auflösung. Erhöht man die zeitliche Auflösung, verringert sich die Frequenzauflosung. Dies liegt daran, dass für eine höhere Frequenzauflosung mehr Samples durch die FFT analysiert werden müssen, und diese Samples stehen im direkten Zusammenhang mit der Zeit.

Um bei gleichbleibender Frequenzauflosung dennoch eine Steigerung der zeitlichen Auflösung zu erreichen, können überlappende Abschnitte (Overlap) analysiert werden [8]. Dabei gilt, dass die Anfänge der Abschnitte mit Länge $N$ nicht $N$ Werte auseinanderliegen, sondern $M < N$ Werte (Hop-Size). Als Ergebnis erhält man dadurch

Abbildung 27: Spektrogramm

2.2.5 Cross-Correlation

Die Cross-Correlation-Funktion (CC) [21] findet in den verschiedensten Gebieten, wie z.B. der Mustererkennung, Kryptoanalyse und natürlich der Signalverarbeitung Anwendung und dient der Bestimmung der Ähnlichkeit zweier Signalabschnitte bei unterschiedlichen Zeitverschiebungen (Lags).

$$ (f \ast g)[l] \equiv \sum_{m=-\infty}^{\infty} f[m] * g[m + l] $$

Formel 2: Berechnung des Korrelationskoeffizienten mit Lag $l$

Üblicherweise wird die CC für zwei Signalabschnitte repräsentierende Vektoren $f$ und $g$ mit Länge $n$ berechnet, wobei das Ergebnis einem Vektor der Länge $2n - 1$ entspricht. Dieser Ergebnisvektor enthält die Korrelationskoeffizienten für alle Sample-Lags $-n \leq l \leq n$. Ein Korrelationskoeffizient (Formel 2) gibt dabei an, wie stark die zwei Vektoren $f$ und $g$ bei Lag $l$ übereinstimmen. Der Index, der den höchsten absoluten Korrelationskoeffizienten enthält, gibt an, um wie viele Samples Funktion $g$ von Funktion $f$ verschoben ist. Abbildung 28 zeigt im rechten Diagramm die Korrelationskoeffizienten an den berechneten Lags für zwei Funktionen, es lässt sich herauslesen dass Funktion $g$ um 5 Samples nach vorne verschoben ist.

In der Literatur wird die Cross-Correlation vielerorts synonym mit der Cross-Covariance verwendet [22], obwohl zwischen beiden ein entscheidender Unterschied besteht [23]. Bei der Cross-Covariance wird vor der Berechnung der Koeffizienten der DC-Offset von beiden Signalabschnitten entfernt, damit nur noch relative Signaländerungen
berücksichtigt werden. Da es in der Signalverarbeitung hier scheinbar keine einheitlichen Definitionen gibt, hält sich auch diese Arbeit an andere Autoren und verwendet unter der Bezeichnung „Cross-Correlation“ die Formel der Cross-Covariance.

Abbildung 28: Cross-Correlation


Da die Komplexität der CC in der Zeitdomäne $O(n^2)$ beträgt, steigt die Berechnungszeit mit Vergrößerung der Eingabevektoren rapide an. Zur Beschleunigung der Berechnung existiert auch eine in der Frequenzdomäne ausführbare Variante, die Fast-CC. Sie wird mit Hilfe der FFT berechnet und führt schon bei sehr kleinen Eingabevektoren zu einer vielfachen Zeitersparnis [22].

2.2.6 Resampling

Beim Resampling wird die Samplingrate eines digitalen Audiosignals in eine andere Samplingrate umgerechnet, so dass $S_2 = S_1 \cdot \alpha$. Dadurch ändert sich die temporale Auflösung des Signals. Sinkt die Samplingrate, wird der Vorgang als Downsampling bezeichnet, steigt sie, handelt es sich um Upsampling. Wichtig dabei ist, dass wie beim Sampling das Nyquist-Shannon-Abtasttheorem gilt und ein Tiefpassfilter zwingend erforderlich ist.

Die Gründe fürs Resampling sind vielfältig. Downsampling wird oft eingesetzt, um die Datenrate zu reduzieren. So werden z.B. Studioaufnahmen meist mit sehr hohen Samplingraten erzeugt, um bei der darauf folgenden Bearbeitung so wenig Information

Zur Änderung der Samplingrate existieren zahlreiche Algorithmen, die sich in der damit erzielten Ergebnisqualität, dem Rauschabstand und der Berechnungskomplexität unterscheiden. Die einfachste Variante ist das lineare Resampling, bei dem neue Samples durch lineare Interpolation zwischen bestehenden Samples berechnet werden.

Resampling kann jedoch nicht nur zur Modifikation der temporalen Auflösung verwendet werden, sondern auch zur Modifikation der Abspieldauer bzw. Geschwindigkeit bei gleichbleibender temporaler Auflösung (Time-Stretching\(^{15}\)). Resampelt man einen Track einer Audio-CD auf 22,05 kHz und spielt ihn trotzdem mit einer Rate von 44,1 kHz ab, wird er mit 2-facher Geschwindigkeit wahrgenommen. Leider verschiebt sich dabei auch das Frequenzspektrum nach oben und die Tonhöhe steigt, es entsteht der allseits bekannte „Chipmunk“-Effekt. Im Gegensatz kann man mit einem Upsampling auf 88,4 kHz und einer Playbackrate von 44,1 kHz eine Verlangsamtats von 50% erreichen, das Frequenzspektrum wird jedoch nach unten gestaucht und die Tonhöhe wird vermindert. Die dabei resultierende Tonhöhenänderung wird Pitch-Shift genannt.

Da ein Pitch-Shift beim Time-Stretching normalerweise nicht erwünscht ist, existieren einige Alternativen zum einfachen Resampling, bei denen die Tonhöhe unverändert bleibt [24]. Für Fälle, in denen nur eine minimale Zeitmodifikation notwendig ist und der Fokus nicht stark auf der Ergebnisqualität liegt, ist das Resampling trotz seiner Nachteile eine einfach einsetzbare Option. Viele Menschen werden z.B. bei einer Kompression oder Dehnung der Zeit um 1% keinen Tonhöhenunterschied zum Original feststellen, solange kein direkter Vergleich besteht.

\(^{15}\) Time-Stretching (dt. Zeitdehnung) umfasst im Gegensatz zu seiner Bezeichnung nicht nur die Dehnung, sondern auch die Zeitkompression.
3 Umsetzung des Systems


3.1 Fingerprinting / Audio-Matching

Um Audiospuren zueinander synchronisieren zu können, müssen automatisiert sich überschneidende Abschnitte (Matches) in ihren Signalen gefunden werden. Es genügt also nicht nur zu erkennen, ob sich Signale überschneiden bzw. gemeinsame Inhalte aufweisen, es ist wichtig zu wissen wo genau diese Überschneidungen zu finden sind – denn Matching führt nicht zwingend zur Möglichkeit der Angleichung. Auf welche Weise dies geschieht ist für das Endergebnis nicht von Belang, wichtig ist nur, dass die Matches ausreichend feingranular sind, so dass ihre Positionsinformationen zu einem Synchronisierungsergebnis führen, dass von einem Menschen subjektiv als gut empfunden wird. Außerdem wäre es von großem Vorteil, wenn der zu Grunde liegende Algorithmus mit linearem Speicherverbrauch in linearer Zeit ausgeführt werden könnte. Ein naiver Ansatz wäre zu denken, dass bei einer Synchronisierung von mehreren Spuren jede mit jeder verglichen werden müsste, was in einer annähernd


3.1.1 Was ist Fingerprinting?


Analog dazu kann Multimedia-Fingerprinting als effizienter Mechanismus angesehen werden, um die wahrgenommene Gleichheit von Multimediaobjekten zu ermitteln [31]. So werden auch hier die zugehörigen sehr kleinen Fingerprints anstatt der oft

\textsuperscript{16} Als Alternative oder zusätzlich zum Abkürzen von Musikstücken spielen sie manche Radiosender schneller ab, um so Sendezeit einzusparen bzw. mehr verschiedene Inhalte in einer Sendung unterzubringen.
Umsetzung des Systems


- Robustheit (Invarianz bei Wahrnehmungsgleichheit): Fingerprints modifizierter bzw. verschlechterter Audioobjekte müssen den Fingerprints der originalen Audioobjekte gleichen oder ähneln.
- Paarweise Unabhängigkeit (Kollisionsfreiheit): Werden zwei Audioobjekte unterschiedlich wahrgenommen, müssen sich ihre Fingerprints deutlich unterscheiden.
- Sicheffizienz in Datenbanken: In großen Datenbanken muss die Suche schnell durchführbar sein, damit das System in der Praxis angewendet werden kann.


Da diese Technologie kein neuartiges Forschungsgebiet darstellt, kommt es heute schon in vielen praktischen Gebieten zur Anwendung. Broadcast- und Airplay-
Monitoring ermöglicht es Rechteinhabern nachzuvollziehen, wann, wie oft und wo ihr Content im Radio oder Fernsehen übertragen wird. Mobile Anwendungen wie Shazam\textsuperscript{17} [34] und SoundHound\textsuperscript{18} ermöglichen es durch kurze Aufnahmen mit dem Mobiltelefon innerhalb von Sekunden jeden erdenklichen Song unter den widrigsten Umständen zu erkennen. In Filesharing-Netzwerken wird sie eingesetzt, um unrechtmäßige Kopien aufzuspüren oder herauszufiltern. Musikbibliotheken können dadurch automatisch organisiert, Duplikate aufgespürt, fehlende Metadaten ergänzt und fehlerhafte Metadaten korrigiert werden. Onlinedienste wie Last.fm\textsuperscript{19} ermitteln, unterstützt durch Fingerprinting, die Hörgewohnheiten der Nutzer, um automatisch Vorschläge zu eventuell ebenfalls interessanten Songs und Interpreten zu unterbreiten.

3.1.2 Schematische Funktionsweise


\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{content_based_audio_identification_framework.png}
\caption{Abbildung 30: Content-Based Audio-Identification-Framework [33]}
\end{figure}

Bei der Extraktion werden relevante Wahrnehmungscharakteristiken aus den Audioaufnahmen gewonnen, die die zuvor beschriebenen Eigenschaften erfüllen [33]. Dabei muss ein Kompromiss zwischen Datenreduktion und Informationsverlust eingegangen werden – umso mehr Daten verworfen werden, umso kleiner wird ein resultierender Fingerprint, jedoch dürfen nicht zu viele Informationen verloren gehen.

\begin{thebibliography}{9}
\bibitem{shazam} \url{http://www.shazam.com/}
\bibitem{soundhound} \url{http://www.soundhound.com/}
\bibitem{lastfm} \url{http://www.last.fm/}
\end{thebibliography}
Umsetzung des Systems

Damit nun durch die Extraktion ein oder mehrere Fingerprints aus einem Signal gewonnen werden, werden mehrere Stufen durchlaufen (Abbildung 31).

Abbildung 31: Schema der Fingerprint-Extraktion [33]


20 Synonym zu Windowing
große Überschneidungen der Frames wird die Verschiebung der Zeitpunkte minimiert, es stehen mehrere Vergleichsframes zur Verfügung. Hier muss wiederum ein Kompromiss zwischen Robustheit und Datenvolumen getroffen werden.

Im nächsten Schritt, der linearen Transformierung, werden die Frames in eine alternative Darstellungsform transformiert, die bei passender Auswahl zu einer Redundanzverminderung beiträgt [33]. Meistens findet hier eine Umwandlung von der Zeit- in die Frequenzdomäne statt, um weitere Verarbeitungsschritte zu ermöglichen oder zu erleichtern. Das Standardverfahren hierbei ist die FFT, welche in Kombination mit dem Framing & Overlap der STFT entspricht. Alternativen sind unter Anderem die Diskrete Kosinustransformation (DCT) und diverse Wavelet-Transformationen.


Außerdem ist es möglich, die Quantisierung so zu gestalten, dass die Zufälligkeit erhöht wird, um die Kollisionswahrscheinlichkeit nicht-äquivalenter Fingerprints zu erhöhen.


Üblicherweise wird die Fingerprint-Extraktion für eine große Menge von Audioobjekten durchgeführt und aus den entstandenen Fingerprints, verbunden mit den Metadaten der Audioobjekte, eine Referenzdatenbank aufgebaut. Um ein unbekanntes Audioobjekt zu identifizieren, wird die Extraktion für dieses Objekt bzw. je nach Implementierung auch nur für einen kurzen Abschnitt dieses Objekts durchgeführt. Darauffolgend werden die entstandenen Fingerprints mit der Referenzdatenbank verglichen. Stimmen Fingerprints mit den in der Datenbank gespeicherten überein, werden die damit verknüpften Metadaten abgerufen und das Matching war erfolgreich. Stimmen keine Fingerprints überein, verlief das Matching erfolglos und das unbekannte Objekt konnte nicht identifiziert werden. Der konkrete Vergleich von zwei Fingerprints erfolgt dabei mittels einer Distanzmetrik, welche die Ähnlichkeit der beiden Abdrücke ermittelt. Wichtig zu beachten ist dabei, dass sich die Fingerprints wie in Abschnitt 3.1.1 beschrieben nicht zu 100% gleichen, sondern nur ähneln müssen. Umso größer die Ähnlichkeit, umso ähnlicher die Wahrnehmungscharakteristik und umso wahrscheinlicher gleichen sich die Audioobjekte. Um im Endeffekt zwischen Gleich und Ungleicht zu unterscheiden, wird die berechnete Distanz im einfachsten Fall mit einem statischen Threshold (Grenzwert) quantisiert.

Umsetzung des Systems

Der Vollständigkeit halber muss erwähnt werden, dass das Matching nicht in jedem Fall mit einer Distanzmertik, die Fingerprints desselben Formates miteinander vergleicht, durchgeführt werden. Es gibt Systeme, bei denen aus dem zu identifizierenden Audioobjekt keine Fingerprints generiert werden, sondern die Distanz direkt zwischen den Feature-Vektoren des unbekannten Audioobjekts und der in der Datenbank gespeicherten Referenz-Fingerprints berechnet wird.

Der letzte große Teil im Fingerprint-System ist die Suchmethode, mit der jene Fingerprints in der Datenbank identifiziert werden, die als Match-Kandidaten in Frage kommen [33]. In einem System mit Millionen von Fingerprints ist eine effiziente Methode von größter Bedeutung, damit das System die Vergleiche so effizient wie möglich durchführt. Im einfachsten Fall, dem sequenziellen Vergleich eines Fingerprints mit allen Referenz-Fingerprints, würde die Usability stark leiden und die Zeitspanne, die dieser Vergleich benötigt, schon bei einer geringer Datenbankgröße unakzeptabel groß sein. Ziel ist es jedenfalls, eine Datenstruktur bzw. einen Index zu erstellen, der es ermöglicht, die Menge der benötigten Distanzberechnungen zu minimieren. Dies kann unter Anderem durch Äquivalenzklassen erfolgen, wonach nur noch ein Teil der Datenbank durchsucht werden muss. Ein anderer Ansatz ist die Verwendung von Heuristiken, die in der Bioinformatik zum Vergleich von DNA-Sequenzen verwendet werden. Geachtet werden muss auch darauf, dass keine in Frage kommenden Fingerprint-Kandidaten verworfen werden. Das bedeutet, dass die grobe Distanz, mit welcher unpassende Kandidaten im Vorhinein aussortiert werden, die feine Distanz, mit der die eigentlichen Distanzwerte berechnet werden, von unten beschränken muss.

3.1.3 Konkrete Implementierung


21 Der Spectral-Centroid (dt. „Schwerpunkt des Spektrums“) charakterisiert ein Spektrum und entspricht perzeptuell der „Helligkeit“ eines Geräusches.


Die Implementierung dieser Arbeit entspricht vollständig der Beschreibung in [31], Angaben bestimmter Werte erfolgen hier im Gegensatz zum Paper jedoch so genau wie möglich. Um die Sub-Fingerprints zu berechnen, wird das Eingangssignal als erstes zu einem Monosignal konvertiert, und auf 5512 Hz downgesampelt. Danach werden dem Signal einzelne Frames mit einer Länge von 2048 Samples (=0,37 Sekunden) und einem Overlap von 1984 Samples entnommen. Das bedeutet, dass das jeweils erste Sample eines Frames nur 64 Samples (=11,6 Millisekunden) von seinem
Vorgängerframe entfernt ist. Daraus wiederum folgt, dass zwei Signale an der Stelle eines erfolgreichen Matches höchstens um $\approx 5,8$ Millisekunden unsynchron sind. Im nächsten Schritt wird auf jedem Frame eine Hann-Fensterfunktion angewendet und die FFT ausgeführt, weil die für die Wahrnehmung wichtigsten Audiofeatures in der Frequenzdomäne zu finden sind [31]. Da die FFT auf unterschiedliche Phasen stark reagiert, der menschliche Hörapparat aber, wie in den Grundlagen auf Seite 22 bereits beschrieben, nicht sensitiv dafür ist, werden die komplexen Ergebniswerte durch die Euklidische Distanz in absolute Werte umgerechnet, wodurch die Phaseninformation verschwindet. Außerdem muss die Phase auch eliminiert werden, weil sie dem Fingerprinting nicht zuträglich ist. Angenommen, es werden zwei Frames mit einem Abstand von 5 Samples entnommen, so würden sich diese Frames für den Menschen akustisch so gut wie gleich anhören und auch sehr ähnliche Frequenzspektra ergeben – die Phasenwerte würden sich jedoch unter Umständen massiv unterscheiden. Aus den resultierenden 1024 Frequenzmagnituden werden dann alle Werte, die in das Frequenzband von 300 Hz bis 2000 Hz fallen, auf 33 logarithmisch verteilte, nicht überlappende Frequenzbänder aufgeteilt und summiert. Logarithmisch deswegen, da der menschliche Hörapparat selbst auf ungefähr logarithmisch aufgeteilten Frequenzbändern operiert [31]. Diese 33 Werte aus jeweils zwei aufeinanderfolgenden Frames werden dann durch Formel 3 in jeweils einen 32-Bit-langen Sub-Fingerprint umgerechnet. Dabei entspricht $E(n,m)$ der Energie von Frame $n$ im Frequenzband $m$. Um einen vollständigen 32-Bit-Sub-Fingerprint von Frame $n$ zu erlangen, wird $F(n,m)$ 32 Mal für $m = 0$ bis $m = 31$ berechnet.

\[
F(n,m) = \begin{cases} 
1, & E(n,m) - E(n,m + 1) - (E(n - 1,m) - E(n - 1,m + 1)) > 0 \\
0, & \text{sonst}
\end{cases}
\]

Formel 3: Berechnung von Bit $m$ in Sub-Fingerprint $n$

Wird ein Audioobjekt also in $x$ Frames segmentiert, werden daraus $x - 1$ Sub-Fingerprints gewonnen. Ein Fingerprintf-Block wird aus jeweils 256 sequenziellen Sub-Fingerprints gebildet, was einer Länge von ca. 3 Sekunden entspricht – im Idealfall reicht also ein beliebiger Ausschnitt von 3 Sekunden Länge aus einem Audioobjekt aus, um das Objekt zu identifizieren. Die Blöcke können sich selbst beliebig überschneiden, aus einer Sequenz von 512 Sub-Fingerprints können also 257 Fingerprintf-Blöcke gebildet werden.

Abbildung 32: Fingerprintf-Blöcke aus 2 YouTube-Clips des selben Konzertes und ihre Differenz (Unterschiede als schwarze Pixel)

Abbildung 32 zeigt exemplarisch zwei Fingerprintf-Blöcke, in denen vertikal einzelne Sub-Fingerprints abgebildet sind und horizontal 256 sequenzielle Sub-Fingerprints
Umsetzung des Systems

aneinandergereiht sind. Schwarze und weiße Pixel entsprechen der 1 bzw. der 0 aus Formel 3. Der dritte Block visualisiert den Unterschied zwischen beiden Fingerprint-Blöcken, worin schwarze Bereiche einen Unterschied zwischen den beiden Blöcken symbolisieren. In diesem Fall unterscheiden sich 36% aller Bits.

Ob nun zwei 3 Sekunden lange Audiosignale vom selben Audioobjekt stammen und daher als gleich einzustufen sind, hängt vom Threshold $T$ ab. Er definiert die exklusive Obergrenze in der Anzahl von Bitfehlern bzw. die Bit-Error-Rate (BER), zwischen zwei Fingerprint-Blöcken. Die BER ist ein Wert zwischen 0 (100% Übereinstimmung) und 1 (keine Übereinstimmung), wobei bei 0,5 nicht zwischen einer 50%igen Übereinstimmung und Zufallswerten unterschieden werden kann – interessant sind daher nur darunterliegende Werte. Liegt die BER unter $T$, so werden beide Blöcke, und damit ihre Ursprungssignale, als gleich erkannt und ein Match ist gefunden. Umso kleiner der Wert gewählt wird, umso geringer ist die Falsch-Positiv-Rate, die Rate mit der unterschiedliche Signale fälschlicherweise als gleich eingestuft werden. Gleichzeitig steigt jedoch die Falsch-Negativ-Rate, also die Wahrscheinlichkeit, dass zwei gleiche Signale nicht als gleich identifiziert werden. In [31] wurde mit Hilfe einer Fingerprint-Datenbank von 10.000 Songs, einer zufälligen Auswahl von 100.000 Fingerprint-Block-Paaren und der Analyse ihrer BER ein optimaler Wert von $T = 0,35$ ermittelt, mit einer Falsch-Positiv-Rate von nur $3,6 \times 10^{-20}$. Das bedeutet, von den 8192 Bits, aus denen ein Block besteht, dürfen sich höchstens 2866 Bits von einem anderen Block unterscheiden, damit beide als gleich eingestuft werden. Im Zuge vieler für diese Arbeit durchgeführten Tests ergab sich jedoch, dass der Threshold problemlos auf 0,45 erhöht werden kann, ohne eine problematische Falsch-Positiv-Rate festzustellen. Die dadurch statistisch stark erhöhte Falsch-Positiv-Rate ist in diesem Fall kein Problem, da die Anzahl der Audioobjekte in der Datenbank in jedem Testfall unter 100 bleibt, verglichen mit den 10.000 in [31]. Auch für einen eventuellen Produktiveinsatz der entwickelten Software ist schwer vorstellbar, dass jemals eine dreistellige Anzahl von Aufnahmen gleichzeitig miteinander synchronisiert wird.

Nachdem die Fingerprints bzw. Sub-Fingerprints erstellt worden sind, müssen sie in einer geeigneten Datenbank bzw. Datenstruktur persistiert werden, um mit anderen Fingerprints verglichen werden zu können und dadurch ein Matching zu ermöglichen. Diese Aufgabe ist keineswegs trivial, da hier keine einfache Suche nach einem fixen Bitmuster durchgeführt werden kann, wofür z.B. eine Hashtable ausreichen würde. In diesem Fall muss das ähnliche Bitmuster in der Datenbank gefunden werden, wodurch impliziert wird, dass es erstens mehrere ähnliche Bitmuster in der Datenbank geben kann, und zweitens der ganze Bestand durchsucht werden muss, um sicher zu gehen, das ähnlichste Muster gefunden zu haben. Unter der Annahme, dass eine Datenbank mit 10.000 Audioobjekten von jeweils 5 Minuten Länge existiert, was ca. 250 Millionen Sub-Fingerprints entspricht, müssten bei einer linearen (Brute-Force) Suche ca. 250 Millionen Fingerprint-Blöcke verglichen werden [31]. Ein einziger Fingerprint-Block-Vergleich würde damit auf einem aktuellen Computer eine
Berechnungszeit von ca. 40 Sekunden benötigen, wobei für eine erfolgreiche Identifizierung eventuell mehrere Vergleiche notwendig sind – falls sich das zu identifizierende Audioobjekt nicht in der Datenbank befindet, könnte der Aufwand sogar auf eine undefiniert hohe Anzahl an Vergleichen steigen. Diese Zeit ist für praktische Anwendungen, bei denen eine große Anzahl von Vergleichen durchgeführt werden muss, nicht tolerierbar und damit steht fest, dass eine bessere Suchmethode gefunden werden muss.

Zur Optimierung schlagen die Autoren in [31] vor, dass eine Hashtable als Lookup-Tabelle (LUT) aufgebaut wird, die jeden einzelnen Sub-Fingerprint enthält, der in den indizierten Audioobjekten vorkommt (siehe Abbildung 33). Jeder Eintrag in der LUT zeigt zu den entsprechenden Positionen der Audioobjekte, an denen der Sub-Fingerprint vorkommt. Unter der Annahme dass in einem Fingerprint-Block mindestens ein Sub-Fingerprint dabei ist, der exakt mit einem Sub-Fingerprint in der Datenbank übereinstimmt, läuft die Suche nach einem Match folgendermaßen ab: Die Sub-Fingerprints im zu suchenden Block werden sequenziell in der Lookup-Tabelle aufgesucht. Enthält die LUT einen Sub-Fingerprint, wird der ganze Block an allen Stellen, auf jene die LUT zeigt, mit den entsprechenden Blöcken verglichen. Sobald die BER bei einem Vergleich unter dem Threshold liegt, ist das Objekt identifiziert und ein Match gefunden. Im Durchschnitt sind hierzu nur 300 Vergleiche notwendig, die in Bruchteilen einer Millisekunde durchgeführt werden können [38,31].

Leider funktioniert dieser Ansatz jedoch nur dann, wenn man leicht degradierte Audiosignale mit ihren Ursprungssignalen vergleicht. Werden wie in dieser Arbeit

mehrere eventuell stark veränderte Signale miteinander verglichen, ohne das Ursprungssignal als Referenz zu haben, so sinkt die Wahrscheinlichkeit sehr stark, dass sich Sub-Fingerprints verschiedener Aufnahmen vollständig gleichen. Zur Lösung dieses Problems schlagen die Autoren vor, für jeden Such-Fingerprint des zu identifizierenden Blocks zusätzlich mehrere abgewandelte Kandidaten zu generieren. Der einfachste Fall, die Generierung aller möglichen Sub-Fingerprints mit einer Hammingdistanz von 1 bis x<32, würde selbst bei einem kleinen x (z.B. x=3) zu einer stark erhöhten Suchzeit führen. Daher machen sich die Autoren Formel 3 zu Nutze, und speichern für jede berechnete Energiedifferenz, die der Bestimmung eines Bits dient, zusätzlich den Unterschied zu 0 als absoluten Wert ab. Diese Werte dienen der Wahrscheinlichkeitsbestimmung, mit welcher ein Bit richtig berechnet wurde. Umso näher der Wert bei 0 liegt, umso geringer ist die Wahrscheinlichkeit, dass das Bit gültig ist. Daraus kann ein Verlässlichkeitsranking der Bits erstellt werden, und durch kippen der unzuverlässigsten Bits zusätzliche Sub-Fingerprints erzeugt werden. Bei einer Toleranz von 3 fehlerhaften Bits pro Sub-Fingerprint steigt die Nummer der Fingerprint-Vergleiche mit der Hamming-Methode um den Faktor 5489, mit Zuverlässigkeitsranking nur um den Faktor 8. Da es passieren kann, dass als unzuverlässig eingestufte Bits stimmen und/oder als zuverlässig eingestufte Bits inkorrekt sind, führt diese Methode nicht in jedem Fall zum optimalen Ergebnis. Im Gegensatz zur Hamming-Methode ist sie jedoch praktisch anwendbar und führt unbestritten zu einer verbesserten Erkennungsrate, da sie nur bei einem der 256 Sub-Fingerprints eines Blocks dazu führen muss, dass ein entsprechender Eintrag in der LUT gefunden wird.

### 3.1.3.1 Unterschiede zum Referenzsystem


Die Modifikation der originalen Systemarchitektur liegt vor allem darin begründet, dass im Gegensatz zum Original kein Matching unbekannter Objekte in mit einer zuvor


<table>
<thead>
<tr>
<th>Einsatzzweck</th>
<th>Identifizierung von unbekannten Objekten</th>
<th>Synchronisierung von bekannten oder unbekannten Objekten</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fingerprint-Extraktion</td>
<td>Separate Extraktion aus Referenzobjekten und Matching-</td>
<td>Ein einziger Extraktionslauf, da alle Objekte gleichzeitig</td>
</tr>
</tbody>
</table>
Umsetzung des Systems

<table>
<thead>
<tr>
<th>Sub-Fingerprint-Varianten</th>
<th>Kandidaten</th>
<th>Referenz und Matching-Kandidat sind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generierung nur für Matching-Kandidaten (für Sub-Fingerprints des Query-Fingerprints-Blocks)</td>
<td>Generierung für alle Objekte (für alle Sub-Fingerprints aller Objekte)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Datenpersistenz</th>
<th>Sub-Fingerprints werden auf lange Zeit persistiert</th>
<th>Sub-Fingerprints werden nur temporär für ein einmaliges Matching benötigt und nicht persistiert</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Datenbankinhalt</th>
<th>LUT und Audioobjekte in Form von Sub-Fingerprints und Metadaten</th>
<th>LUT und Audioobjekte in Form von Sub-Fingerprints</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Lookup-Tabelle</th>
<th>Verweise der Sub-Fingerprints auf die entsprechenden Stellen der Audioobjekte</th>
<th>Verweise der Sub-Fingerprints und generierten Varianten auf die entsprechenden Stellen der Audioobjekte</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Matching-Vorgang</th>
<th>Suche eines Eingabe-Fingerprint-Blocks in der Datenbank</th>
<th>Keine explizite Eingabe; LUT gibt alle zu untersuchenden möglichen Überschneidungen vor</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Matching-Ergebnis</th>
<th>Erste Fingerprint-Übereinstimmung mit BER kleiner Threshold</th>
<th>Alle Fingerprint-Übereinstimmungen mit BER kleiner Threshold</th>
</tr>
</thead>
</table>

Tabelle 4: Zusammenfassung der Unterschiede beim Fingerprinting zwischen [31] und dieser Arbeit

3.1.3.2 Extraktionsprofile


Aus dieser zufällig entstandenen Erkenntnis folgte die Annahme, dass die Fingerprint-Extraktion parametrisierbar gestaltet werden sollte und je nach Anwendungsfall (z.B. Musikkonzert, Rede, usw...) angepasste Parameter zu einem besseren Ergebnis führen könnten, als das in [31] angegebene „Standardverfahren“. Somit wurde die Extraktion um einen Profilmechanismus erweitert. Ein Profil gibt an, auf wie viel Hz das Eingangssignal gesampelt wird (die Konvertierung zu Mono erfolgt in jedem Fall), wie
groß ein einzelnes Frame ist, wie groß der Overlap zwischen den Frames ist, und wie das FFT-Ergebnis auf die 33 Werte gemappt wird, mit denen ein Sub-Fingerprint schlussendlich berechnet wird. Testweise wurden 5 Profile implementiert, wobei diese abgesehen vom „Default“- und „Buggy“-Profil auf reinen Annahmen basieren und nicht durch irgendwelche statistischen Tests ermittelt oder durch bestehende Erfahrungen beeinflusst wurden.

<table>
<thead>
<tr>
<th>Default</th>
<th>Standardprofil, entspricht den in [31] angegebenen Parametern.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buggy</td>
<td>Entspricht dem Standardprofil, das Mapping der Frequenzbänder erfolgt jedoch nach der ursprünglich fehlerhaft implementierten Formel.</td>
</tr>
<tr>
<td>Bass</td>
<td>Entspricht dem Standardprofil, es werden jedoch nur Bassfrequenzen zwischen 80 Hz und 350 Hz berücksichtigt.</td>
</tr>
<tr>
<td>Voice</td>
<td>Entspricht dem Standardprofil, es werden jedoch nur die von menschlichen Stimmbändern erzeugbaren Grundfrequenzen berücksichtigt, welche zwischen 80 Hz und 1100 Hz liegen.</td>
</tr>
<tr>
<td>Human</td>
<td>Einfache Simulation des menschlichen Hörapparats. Sampling mit 44100 Hz, entsprechend große Frames mit 16384 Samples, Frequenzbereich 80 Hz bis 16000 Hz.</td>
</tr>
</tbody>
</table>

Tabelle 5: Profile zur Fingerprint-Extraktion

Um eine bessere Erkennungsrate zu erreichen, können mehrere Profile auch sequenziell auf Audioobjekten ausgeführt werden. Dadurch verlängert sich die Berechnungszeit auf die Summe der Berechnungszeit jedes einzelnen Profils, der Speicherverbrauch wird jedoch vom Profil mit dem meisten Verbrauch nach oben beschränkt. Die Kombination aus „Default“ und „Buggy“ hat sich als empfehlenswerte Methode erwiesen, die anderen Profile erreichen nur in sehr wenigen Ausnahmefällen bessere Resultate als das Standardprofil.

Abbildung 34: Durch Fingerprinting in AudioAlign ermittelte Synchronisationspunkte zwischen 3 Aufnahmen (farbkodiert nach ihrer BER: 0,5 rot / 0 grün)
3.2 Dynamic-Time-Warping


Abbildung 35: Links – zwei Aufnahmen die per DTW angeglichen werden können, da sie zu den gleichen Aufnahmezeitpunkten gestartet und gestoppt wurden. Rechts – eine lange und eine kurze Aufnahme, deren Zeitintervall in der langen Aufnahme enthalten ist. Hier ist eine Angleichung nur möglich, wenn die Zeitpunkte \( t_s \) und \( t_e \) in der langen Aufnahme bekannt sind.


23 Aus Erfahrung ist der Drift bei Längen bis zu ca. 10 Minuten in jedem Fall gering genug, um nicht negativ ins Gewicht zu fallen. Eine genaue Bestimmung dieser Threshold-Länge kann jedoch nicht durchgeführt werden, da sie von mehreren Faktoren, u. A. der Kombination der verwendeten Aufnahmegeräte, abhängt.
Abbildung 36: Euklidische Distanz vs. Dynamic-Time-Warping [40]

Bei der Angleichung wird aus zwei Wertsequenzen \( U = u_1, \ldots, u_m \) und \( V = v_1, \ldots, v_n \) der optimale Warping-Pfad \( W = W_1, \ldots, W_l \) berechnet. Jedes \( W_k \) entspricht einem geordneten Paar \((i_k, j_k)\), und für jedes \((i, j) \in W\) gilt, dass die entsprechenden Werte \( u_i \) und \( v_j \) zeitlich aneinander angeglichen sind [41]. Zur Berechnung des Warping-Pfads wird eine \( m \times n \)-große Warping-Matrix gebildet, in der die Kosten für jedes Paar \((i_k, j_k)\) durch die Kostenfunktion \( d_{U,V}(i, j) \) ermittelt werden. Der Kostenwert von 0 entspricht einer perfekten Übereinstimmung und steigt mit dem Grad des Unterschieds an. Die Summe aller Werte entlang des optimalen Warping-Pfads entspricht der Ähnlichkeit der beiden Eingabesequenzen. Ein Warping-Pfad muss immer in der Matrix-Zelle \((1,1) = W_1\) beginnen und in der diagonal gegenüberliegenden Zelle \((m, n) = W_l\) enden. Aus der Matrix können exponentiell viele Pfade entnommen werden, interessant ist jedoch nur jener mit den minimalen Kosten.


Zur Berechnung des optimalen Pfades wird folgende rekursive Formel angewandt:

\[
D(i, j) = d(i, j) + \min\{D(i, j-1), D(i-1, j), D(i-1, j-1)\}
\]

Formel 4: Berechnung des optimalen DTW-Warping-Pfades

Umsetzung des Systems

lokale Constraints angewendet werden, um z.B. um die Anzahl der Schritte, die der Pfad kontinuierlich in eine Richtung (horizontal, vertikal oder diagonal) gehen darf, zu beschränken. Auch dadurch können unerwünschte Pfade verhindert werden. DTW unterliegt reger Forschungsaktivität und so wurden auch einige ausgeklügelte Optimierungen entwickelt, wie der FastDTW-Algorithmus in [42].

Abbildung 38: Sakoe-Chiba-Band und Itakura-Parallelogramm [40]


3.2.1 On-Line-Time-Warping


\[
E'_x(f, t) = \max(E_x(f, t) - E_x(f, t-1), 0)
\]

Formel 5: Berechnung des Energieanstiegs von Signal \(x(t)\) für Frequenzband \(f\) zum Zeitpunkt \(t\)

Jeder Frame wird somit durch 84 Werte abgebildet, welche jeweils den Energieanstieg im entsprechenden Frequenzband abbilden. Zweck dieser Darstellung ist es, den Beginn von Tönen (Onset) abzubilden, da dies der wichtigste Faktor für die Angleichung ist [41]. Sie ist ähnlich zur Berechnung der Sub-Fingerprints in Kapitel 3.1.3, wo ebenfalls nur Energieanstiege berücksichtigt werden. Die Kostenfunktion für die Distanz zwischen zwei Spektralvektoren entspricht der Euklidianen Distanz.

\[
d(i, j) = \sqrt{\sum_{f=1}^{84} (E'_u(f, i) - E'_v(f, j))^2}
\]

Formel 6: OLTW Kostenfunktion

OLTW erreicht die lineare Komplexität durch eine Limitierung des Suchbereiches, ähnlich dem Sakoe-Chiba-Band. Der limitierte Suchbereich läuft jedoch nicht entlang der Matrix-Diagonale, was u. A. daran liegt, dass die Diagonale beim On-Line-Processing nicht bekannt ist, sondern dynamisch entlang eines berechneten Pfades. Daher kann selbst im Off-Line-Fall mit bekannten Diagonalen nicht vorausbestimmt werden, wie viel Speicher für die Warping-Matrix notwendig ist, der maximal in Anspruch genommene Speicherplatz wird jedoch durch eine Konstante nach oben beschränkt. Durch diese adaptive Diagonale kann die Breite des Suchbereichs im Vergleich zum Sakoe-Chiba-Band stärker eingeschränkt werden, ohne das Risiko zu erhöhen, die optimale Lösung zu verpassen. Um die Matrix zu generieren, muss als
Umsetzung des Systems


\begin{equation}
D(i,j) = \min\{D(i,j-1) + d(i,j), D(i-1,j) + d(i,j), D(i-1,j-1) + 2 \times d(i,j)\}
\end{equation}

Formel 7: Berechnung des optimalen OLTW-Warping-Pfades


3.2.1.1 Paper vs. Realimplementierung

Trotz mehrmaliger unterschiedlicher Versuche den OLTW-Algorithmus auf Basis der Papers [3,4,41] und [44] zu implementieren, gelang es nicht, nur annähernd die Matching-Performance der in Java realisierten Referenzimplementierung MATCH zu erreichen. Während in manchen Fällen zwar akzeptable Resultate erzielt wurden, versagte der Algorithmus in den meisten Fällen vollständig und lieferte Mapping-
Umsetzung des Systems

Pfade, die selbst von unerfahrenen Personen auf den ersten Blick als falsch klassifiziert werden würden. Erst durch Reverse Engineering von MATCH ergab sich die Erkenntnis, dass die veröffentlichten Beschreibungen des Algorithmus nicht nur teilweise falsch, sondern vor allem (zu) stark vereinfacht sind und gerade die für eine gute Performance ausschlaggebenden Optimierungen unerwähnt blieben. Die wichtigste aller Optimierungen ist die Normalisierung der Spektralvektoren anhand ihrer Gesamtenergie – nur so können Audioobjekte unterschiedlicher Lautstärken miteinander verglichen werden. Liegt der quadratische Mittelwert (RMS, Root Mean Square) eines Audioframes unter einem fixen Threshold, wird der gesamte entsprechende Spektralvektor auf Null gesetzt – vermutlich um die Angleichung in besonders leisen Passagen linear zu halten und falsche Abweichungen zu vermeiden. Wichtig ist außerdem die modifizierte Kostenfunktion, in der nicht wie angegeben die euklidische Distanz, sondern die simple Summe der absoluten Unterschiede zwischen zwei Spektralvektoren berechnet wird. Diese Summe wird weiter durch die Gesamtenergie beider Vektoren normalisiert und mit einem aus dieser Energie abgeleiteten logarithmischen Wert gewichtet, der Grund dafür bleibt leider unerschlossen. Das verwendete Frequenzmapping, um die 1024 FFT-Werte auf die 84 Spektralvektorkomponenten abzubilden, weicht in MATCH zwar vom beschriebenen Mapping ab, dies hat jedoch keinen subjektiv merkbaren Einfluss auf die Matchingperformance. Zu guter Letzt sind im Algorithmus angegebene If-Else-Abfragen fehlerhaft, die dazu führen, dass die Matrixerweiterung durch horizontale Reihen fälschlicherweise bevorzugt wird. Durch die beim Reverse Engineering erlangten Erkenntnisse konnte der Algorithmus so implementiert werden, dass er eine MATCH nicht nachstehende Matchingqualität erreicht. Geringe Unterschiede im berechneten Pfad sind dennoch vorhanden und ergeben sich dadurch, dass die Implementierungen nicht ident sind da nur einige wichtige Teile aus MATCH portiert wurden.

Abbildung 39: Ausschnitt aus der Warping-Matrix und des optimalen Warping-Pfades – MATCH (links) vs. AudioAlign (rechts)
Wie in Abbildung 39 erkannt werden kann, ist der resultierende Pfad der Implementierung geradliniger als der entsprechende Pfad in MATCH – er wirkt wie eine geglättete Version des MATCH-Pfades. Dies wird erreicht, indem die Kosten für diagonale Schritte entgegen dem Vorschlag in [3,41] nicht mit dem Faktor 2 gewichtet werden, wodurch die Neigung zu diagonalen Schritten nicht unterbunden wird. Dadurch werden zwar einige Zeitpunkte ein wenig unpräziser zueinander gemappt, was jedoch nur im direkten Vergleich negativ auffällt. Der positive Effekt ist auf jeden Fall größer, denn so werden unnatürlich starke Schwankungen im Zeitgefüge bzw. zu starke Abweichungen vom akustisch-subjektiven Optimalpfad (der nicht mit dem vom OLTW berechneten Optimalpfad übereinstimmen muss) automatisch verhindert. Außerdem kann man sich darauf verlassen, dass die Zeitabweichungen zwischen Liveaufnahmen in jedem Fall nur marginal ausfallen – im Gegensatz zur Angleichung unterschiedlicher Interpretationen eines Musikstücks. Die Implementierung wurde neben Liveaufnahmen, bei welchen exzellente Ergebnisse erzielt werden, auch mit verschiedenen Interpretationen von Musikstücken und Sprachaufnahmen getestet und liefert hier bis auf wenige Ausnahmen erstaunlich gute Ergebnisse.

Abbildung 40: Durch OLTW gefundene Synchronisationspunkte zwischen der Originalversion und einer Coverversion eines Songs

Abbildung 41: Durch OLTW gefundene Synchronisationspunkte zwischen zwei Liveaufnahmen eines Konzerts

3.2.2 Audio-optimiertes Dynamic-Time-Warping

Diese Implementierung des DTW hält sich so nah wie möglich an die Standardvariante [39], setzt jedoch auf die Distanzmetrik des OLTW. Sie wurde erstellt, um das allgemeine Verständnis des DTW zu fördern und die Ergebnisse des OLTW zu evaluieren, da OLTW im Vergleich zu DTW durch die Einschränkung des Suchbereichs nicht zwingend den optimalen Angleichungspfad zurückliefert [41]. Es stellte sich jedoch schnell heraus, dass die Berechnung der gesamten Warping-Matrix selbst für kurze Audioobjekte untolerierbar viel Zeit benötigt, daher wurde der Algorithmus mittels dem Sakoe-Chiba-Band auf linearen Platz- und Zeitbedarf beschränkt. Anfänglich lieferte diese Variante gegenüber dem OLTW weitaus bessere Ergebnisse, was jedoch an der in Abschnitt 3.2.1.1 beschriebenen unvollständig dokumentierten Distanzmetrik lag. Der OLTW-Algorithmus konnte durch seinen dynamischen
Umsetzung des Systems  63

Suchbereich beliebig stark vom optimalen Pfad abweichen, wogegen dieser Algorithmus auf den starren Suchbereich begrenzt blieb und die fehlerhaften Abweichungen im Warping-Pfad daher höchstens der halben Breite des Suchbandes entsprechen konnten. Bei einer Suchbreite von 10 Sekunden konnte der maximale Fehler somit höchstens 5 Sekunden betragen. Nachdem die Metrik verbessert wurde, lieferte dieser Algorithmus dieselben Ergebnisse wie OLTW, unter der Voraussetzung, dass der nichtlineare Drift schwach genug ist, um vom Suchbereich vollständig abgedeckt zu werden. OLTW bleibt jedoch die bevorzugte Variante, da sie auf eventuelle Unregelmäßigkeiten in den anzugleichenden Objekten dynamischer reagiert und das Suchfenster somit schmäler gehalten werden kann, was wiederum zu einer Verkürzung der Berechnungszeit beiträgt.

3.3 Cross-Correlation


Umsetzung des Systems

Abbildung 42: Durch CC gefundene Synchronisationspunkte (gleiche Aufnahmen wie in Abbildung 41)

Wird die CC für zwei Abschnitte in zwei Audioobjekten berechnet, so liefert sie zwei Rückgabewerte: den Offset, der den Versatz zur optimalen temporären Ausrichtung angibt, und den entsprechenden Korrelationskoeffizienten – einen Wert, der die Güte der Übereinstimmung zwischen 0 und 1 bewertet. Damit gibt es jedoch mehrere Probleme. Ein Wert von 1 wird nur erreicht, wenn zwei idente Aufnahmen miteinander verglichen werden. Beim Vergleich zweier unterschiedlicher Aufnahmen geht die Wahrscheinlichkeit gegen Null, dass zwei Abschnitte in zwei zu vergleichenden Objekten zu 100% übereinstimmen, selbst wenn bekannt ist, dass der optimale Synchronisierungspunkt innerhalb der Abschnitte liegt. In allen im Zuge der Arbeit durchgeführten Tests lag der Koeffizient jedenfalls unter 0,5, meist sogar unter 0,2. Die CC liefert außerdem in jedem Fall einen Koeffizienten, auch wenn sich die verglichenen Abschnitte an keiner Stelle überschneiden, und so kann es passieren, dass der Koeffizient in solchen Fällen sogar höher ist als im Fall der garantierten Überschneidung. Dies hat zur Folge, dass bei CC kein absoluter Threshold angewendet werden kann. Außerdem ergeben sich oftmals bessere Koeffizienten an Stellen, an denen zwar eine Überschneidung bekannt ist, der Offset jedoch nicht korrekt berechnet wurde, im Gegensatz zu Stellen, an denen ein korrekter Offset berechnet wurde.

Abbildung 43: Abschnitte aus zwei Aufnahmen mit gleichem Inhalt, jedoch keiner optisch erkennbaren Korrelation ihrer Hüllkurven

Es drängt sich die Vermutung auf, dass die CC auf Sample-Ebene nicht der optimalste Weg ist, um Audioaufnahmen zu synchronisieren – auf der Spektralebene würde jedoch wieder die Genauigkeit der Synchronisierung leiden. Ein Grund der Suboptimalität könnte sein, dass sich die Hüllkurven von Audioaufnahmen selbst bei akustisch gleich wahrgenommenem Inhalt nicht unbedingt ähneln müssen, z.B. im Fall eines zu geringen Dynamikumfangs (Abbildung 43). Aus diesen Gründen wurde der Ansatz mittels Cross-Correlation nicht weiter verfolgt, er sollte jedoch für zukünftige Untersuchungen im Hinterkopf behalten werden.
3.4 Match-Filterung & Angleichung

Matches bzw. Synchronisationspunkte, durch welche Audioobjekte ultimativ synchronisiert werden, können aus unterschiedlichen Quellen gewonnen werden. Fingerprinting, manuelle Definition, Time-Warping, Cross-Correlation oder einer beliebigen Kombination dieser Techniken. Woher die Matches stammen, ist im Endeffekt nicht wichtig.

Grundsätzlich ist ein Match ein Tupel aus fünf Werten, wobei zwei davon die zwei beteiligten Audioobjekte referenzieren, weitere zwei Werte geben die entsprechenden Zeitpunkte in den Audioobjekten an, an denen sie übereinstimmen, und der letzte Wert gibt die Ähnlichkeit der zwei Zeitpunkte als Zahl zwischen 0 (keine Übereinstimmung) und 1 (100% Übereinstimmung) an. Die Ähnlichkeit wird bei durch Fingerprinting gewonnenen Matches aus der Bit-Error-Rate abgeleitet (1 − BER), bei von Cross-Correlation abstammenden Matches entspricht sie dem Korrelationskoeffizienten. Manuelle Matches werden standardmäßig mit 1 bewertet, da davon auszugehen ist, dass absichtlich angelegte Synchronisationspunkte in jedem Fall korrekt sein sollten – der Wert kann jedoch manuell verändert werden. Von einem Time-Warping-Verfahren gewonnene Matches erhalten ebenfalls einen Ähnlichkeitswert von 1, da dieser Algorithmus es nicht zulässt, die Güte der Übereinstimmung absolut zu bewerten (ein dadurch gewonnener Angleichungspfad, aus dem die Matches bestimmt werden, ist per Definition optimal).

Abbildung 44: 3 Audioobjekte und 2 Matches (oben unsynchronisiert, unten synchronisiert)

Zur Synchronisation von \( n \) Audioobjekten sind im Optimalfall nur \( n-1 \) Matches notwendig, vorausgesetzt sie sind korrekt und verbinden alle Objekte miteinander. Praktisch tritt dieser Fall jedoch so gut wie nie ein, daher ist es umso besser, je mehr Matches bekannt sind. Dies steht im Gegensatz zur bloßen Identifizierung von Inhalten...
in Audioobjekten, bei der, z.B. beim Einsatz von Fingerprinting, ein einziger Match ausreicht. Umso mehr Matches verfügbar sind, umso größer ist erstens die Wahrscheinlichkeit, einen korrekten Match zur Verfügung zu haben, außerdem sind pro Objektpaar mindestens zwei Matches notwendig, um einen eventuell vorhandenen Time-Drift zu erkennen. Mit Time-Drift muss in jedem Fall gerechnet werden, nur bei sehr kurzen Objekten kann ein einziger Match ausreichend sein, um eine akzeptable Synchronisation über die gesamte Laufzeit zu erreichen.

Nachdem Matches gefunden wurden, müssen sie im nächsten Schritt gefiltert werden, um die Synchronisation vorzubereiten. Dabei ist es wichtig, Matches mit guter Übereinstimmungsqualität zu bevorzugen und jene von schlechter Qualität zu verwerfen. Außerdem müssen Zyklen zwischen durch Matches verbundenen Audioobjekten entfernt werden, damit jedes Audioobjekt höchstens ein Mal an ein anderes Objekt angeglichen wird. Dies ist notwendig, da nur so eine durchgehende Angleichung aller beteiligten Objekte realisiert werden kann. Angenommen es existieren drei Objekte \( A \), \( B \) und \( C \). Wird Objekt \( B \) an Objekt \( A \) angeglichen, so kann im nächsten Schritt nur noch Objekt \( C \) an \( A \) oder \( B \) angeglichen werden. Würde man jedoch \( A \) oder \( B \) an Objekt \( C \) angleichen, würde die Synchronisation zwischen \( A \) und \( B \) wieder verloren gehen. Erreicht wird dies mit Hilfe der Graphentheorie, indem folgende Schritte ausgeführt werden (vgl. Abbildung 45):


4. Konvertierung des minimalen Spannbaums in einen Wurzelbaum. Dazu wird ein beliebiger Knoten als Wurzel ausgewählt und der Baum mittels des Prinzips der Tiefensuche vollständig traversiert (Breitensuche wäre ebenfalls möglich). Die Reihenfolge der besuchten Kanten definiert die Reihenfolge, mit welcher die Audioobjekte zueinander synchronisiert werden.
Schlussendlich erfolgt die Angleichung, indem entsprechend der Synchronisierungsreihenfolge jeweils ein Audioobjektpaar miteinander synchronisiert wird. Dazu wird anhand des temporal ersten Matches (sofern mehrere übrig sind – ansonsten des einzigen Matches) das zweite Objekt zeitlich so zum ersten Objekt positioniert, dass beide von Match referenzierten Zeitpunkte in den Objekten am selben globalen Zeitpunkt liegen (siehe Abbildung 44). Für alle weiteren Matches werden die entsprechenden Abschnitte im zweiten Objekt durch Resampling so verlangsamt oder beschleunigt, dass auch alle weiteren Matches jeweils am selben globalen Zeitpunkt liegen (Driftkorrektur, siehe Abbildung 46).

3.5 Architektur


In den folgenden Abschnitten wird zuerst ein grober Überblick über die Hauptkomponenten gegeben, danach wird die Architektur der Audioverarbeitung und der Matchingfunktionalität genauer betrachtet, da sie den Kern der Software bereitstellen.

3.5.1 Komponenten

Abbildung 47: AudioAlign Hauptkomponenten
AudioAlign besteht aus 6 Hauptkomponenten, welche jeweils eine spezielle Aufgabe erfüllen. Sie bilden zusammen die im Zuge dieser Diplomarbeit erstellte Software. Ihre Aufgaben werden in der folgenden Tabelle erläutert:

<table>
<thead>
<tr>
<th>Komponente</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAudio</td>
<td>NAudio(^\text{25}) ist eine Open Source .NET Audiobibliothek unter der Microsoft Public License (Ms-PL), die grundlegende Funktionen zur Audioverarbeitung bereitstellt. Sie ermöglicht das Einlesen unterschiedlicher Audioformate (z.B. WAVE und MP3), die Verarbeitung auf Basis von Streams und die Ausgabe in verschiedene Audioformate oder an eine Playback-Schnittstelle. In AudioAlign wird sie verwendet, um Audiodateien einzulesen, und das Abspielen über die Windows Audio Session API (WASAPI) zu ermöglichen. Der wichtige Teil beim Einlesen ist das Dekodieren der Audioformate, um den AudioAlign-Streams (dokumentiert im folgenden Abschnitt 3.5.2) erstens benötigte Metainformationen, wie die Anzahl der Kanäle und die Samplingrate mitzuteilen, und die Audiodaten im unkomprimierten Rohformat (PCM) zur Verfügung zu stellen. Durch die WASAPI können in die Anwendung geladene Audiodateien über die Soundkarte ausgegeben werden. Dadurch wird das Mixen der Audiospuren erleichtert und das Ergebnis der Synchronisation kann akustisch validiert werden.</td>
</tr>
<tr>
<td>AudioAlign.FFTW</td>
<td>Diese Komponente bindet die native FFTW-Bibliothek(^\text{26}) („Fastest Fourier Transform in the West“) ein, eine hochoptimierte Bibliothek zur Berechnung der FFT, herausgegeben unter der GNU General Public License (GPL). Die Komponente kapselt sie auf objektorientierte Weise, so dass mehrere Instanzen auf einfache Weise verwendet werden können. Sie wird durchgehend zur Berechnung der FFT eingesetzt. Anfänglich wurde zur FFT-Berechnung die ebenfalls als Open Source verfügbare und in Managed C# geschriebene Exocortex.DSP-Bibliothek(^\text{27}) verwendet, die in einer 32-Bit-Umgebung zwar eine akzeptable Geschwindigkeit aufweist, sich in 64-Bit-Umgebungen jedoch als massiver Flaschenhals erwies.</td>
</tr>
<tr>
<td>AudioAlign.LibSampleRate</td>
<td>AudioAlign.LibSampleRate bindet den als Secret Rabbit Code(^\text{28}) bzw. libsamplerate bekannten Sampleratekonverter ein. Er ist ebenfalls in C geschrieben und als nativer Code ausführbar, steht unter der GPL und daher ebenfalls Open Source. Diese Bibliothek ist für ihre gute Konvertierungsqualität bekannt, und wird u. a. standardmäßig in vielen Linux-Distributionen verwendet. Auch sie wird durch diese Komponente als Managed</td>
</tr>
</tbody>
</table>

---


\(^{26}\) [http://www.fftw.org/](http://www.fftw.org/)

\(^{27}\) [http://www.exocortex.org/dsp/](http://www.exocortex.org/dsp/)

Umsetzung des Systems

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AudioAlign</td>
<td>AudioAlign ist die ausführbare Komponente, die unter Verwendung der AudioAlign.WaveControls-Komponente die GUI auf Basis der Windows Presentation Foundation (WPF) definiert. Sie verbindet die Benutzeroberfläche mit den Audioverarbeitungsfunktionen der AudioAlign.Audio-Komponente und sorgt so dafür, dass die Funktionen auf einfache Weise genutzt werden können, und der Benutzer jederzeit über die ausgeführten Vorgänge informiert ist.</td>
</tr>
</tbody>
</table>

Tabelle 6: Beschreibung der AudioAlign Hauptkomponenten
3.5.2 Stream-Modell


Abbildung 48: Exemplarischer Audiodatenverlauf


Grundsätzlich ist das Modell jedoch nicht auf einen bestimmten Datentyp beschränkt, durch die durchgehende Verwendung von Byte-Arrays als Transportdatenformat in der Pipeline kann jedes beliebige Rohdatenformat transportiert werden. Die meisten Streams sind jedoch intern auf die Verarbeitung von IEEE32-Daten beschränkt. Den Kern des Modells bietet das IAudioStream-Interface, das von jedem Stream implementiert werden muss. Es bietet in Anlehnung an die Datenstreams des Java...
Umsetzung des Systems


<table>
<thead>
<tr>
<th>Length</th>
<th>Die Länge des Streams in Bytes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Stellt Metadaten zum Audioformat des Streams dar. Ist untergliedert in:</td>
</tr>
<tr>
<td>SampleBlockSize</td>
<td>Die Größe eines Sampleblocks in Bytes. Ein Sampleblock ist ein Datenblock, der die Samples aller Kanäle zu einem Zeitpunkt enthält.</td>
</tr>
</tbody>
</table>

Jede konkrete Implementierung eines Streams erledigt eine spezielle Aufgabe, die üblicherweise in der Read-Funktion ausgeführt wird (z.B. Konvertierung zu Mono oder Lautstärkenmodifikation). Durch die Aneinanderkettung der Streams können komplexe Audioverarbeitungspipelines konstruiert werden. Die wichtigsten Streams werden in Tabelle 8 beschrieben.

| BufferedStream | Einfach oder doppelt gepufferter Audiostream. Kann nahe der Ein- oder Ausgabe verwendet werden, um eine größere Menge von Daten im Arbeitsspeicher vorzuhalten, als von der nachfolgenden Einheit pro Leseaufruf gelesen wird. Liegt die Quelldatei auf einem Datenträger mit hoher Latenz, kann diese durch die Doppelpufferung ausgeglichen werden. Bei der Doppelpufferung wird der Backbuffer asynchron befüllt. |
| CropStream | Beschneidet einen Audiostream, indem der Anfangspunkt zeitlich nach hinten und/oder der Endpunkt zeitlich nach vorne verlegt. |
Umsetzung des Systems


<table>
<thead>
<tr>
<th>Streamname</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataMonitorStream</td>
<td>Feuert nach jedem Lesevorgang ein Event, das die gerade gelesenen Daten an alle Event-Listener übergibt. Wird in AudioAlign verwendet, um den gemixten Datenstrom abzuhören, darauf die FFT anzuwenden, und dadurch das Spektrum, das Spektrogramm und das Korrelationsmometer für die GUI zu zeichnen.</td>
</tr>
<tr>
<td>IeeeStream</td>
<td>Wandelt einen 16-Bit-Integer-Stream in einen 32-Bit-Float-Stream um. Wird benötigt, falls eine Quelldatei in diesem Format vorliegt, was bei den meisten Dateien der Fall ist. In einer Stream-Hierarchie folgt dieser Stream meist direkt dem NAudioSourceStream.</td>
</tr>
<tr>
<td>NAudioSinkStream</td>
<td>Sollen Audiodaten über die NAudio-Bibliothek ausgegeben werden, egal ob per Playback oder als Datei, so kann er mit diesem Stream in das NAudio-Stream-Format umgewandelt werden. Dieser Stream stellt in diesem Fall also den letzten Stream in der Kette dar. NAudio übernimmt dabei die Konvertierung vom IEEE32-Format in das jeweilig gewünschte oder benötigte Zielformat.</td>
</tr>
</tbody>
</table>
| NAudioSourceStream    | Damit eine durch NAudio dekodierte Quelle im AudioAlign-Framework weiterverarbeitet werden kann, muss sie in einen AudioAlign-Stream umgewandelt werden. Diese Aufgabe wird vom NAudioSourceStream erledigt, er stellt somit den ersten Stream in der Kette dar. Liegen die Daten nicht im IEEE32-Format vor,
Umsetzung des Systems

müssen sie anschließend mit einem IeeeStream konvertiert werden.

**OffsetStream**

**PhaseInversionStream**
Kehrt die Phase eines Streams um, d.h. das Vorzeichen jedes Samples wird vertauscht. Der Effekt dieses Streams kann durch ein Flag ein- und ausgeschaltet werden. Werden mehrere Streams gemixt, kann dieser Stream in Vorschaltung dazu genutzt werden, ungewünschte Auslöschungseffekte zu verhindern.

**ResamplingStream**

**TimeWarpStream**
Dieser Stream dient dem Ausgleich des Time-Drifts relativ zu einem Referenzstream, indem einzelne Abschnitte des Streams durch Resampling zeitlich verkürzt oder verlängert („gewarped“) werden, und diese Abschnitte dadurch in ihrer Laufzeit an die Abschnitte des Referenzstreams angeglichen werden. Dadurch wird die Tonlage der Abschnitte zwar erhöht oder gesenkt, die Modifikation ist jedoch im Normalfall so minimal, dass sie unbemerkt bleibt.
Um den Stream zu warren, muss ihm eine Liste von Zeitmappings übergeben werden. Ein Zeitmapping gibt an, welcher Zeitpunkt im ungewarnten Stream einem Zeitpunkt im gewarnten Stream entspricht. Um im in Abbildung 49 ersichtlichen Beispiel den ersten Abschnitt von Stream #2 \( (t_{20} - t_{21}) \) an die Laufzeit des entsprechenden Abschnitts in Stream #1 \( (t_{10} - t_{11}) \) anzulegen, müssen Stream #2 die Mappings von Synchronisationspunkt #1 \( (t_{20} \rightarrow t_{10}) \) und Synchronisationspunkt #2 \( (t_{21} \rightarrow t_{11}) \) übergeben werden. In diesem Fall würde der Abschnitt upgesampelt werden, um seine Laufzeit passend zu Stream #1 zu verlängern. Alle anderen Abschnitte in Stream #2 würden downgesampelt werden, um ihre Laufzeit entsprechend zu verkürzen.

VisualizingStream

Mit Hilfe dieses Streams erfolgt die grafische Darstellung von Audiodaten in der GUI. Wird nur ein sehr kurzer Ausschnitt aus dem Stream gezeichnet, so können die dafür benötigten Samples direkt ausgelesen werden. Umso weiter man nach außen zoomt, d.h. umso mehr Samples gleichzeitig gezeichnet werden müssen, umso mehr Daten müssen bei jedem Zeichenvorgang geladen werden.

Ab einer gewissen Zoomstufe wird ein Zustand erreicht, bei dem Samples nicht mehr einzeln gezeichnet werden können, sondern mehrere Samples in eine vertikale Pixelzeile zusammenfallen. Ab diesem Zustand werden nicht mehr einzelne Samples, sondern eine Hüllkurve gezeichnet, bei der in jeder vertikalen Pixelzeile der Minimal- und Maximalwert (die Peaks) der zusammengefassten Samples gezeichnet wird.

Wenn man noch weiter nach außen zoomt, erreicht man
Umsetzung des Systems

irgendwann einen Punkt, an dem die Samples für einen Zeichenvorgang nicht mehr schnell genug gelesen und zu Peaks zusammengefasst werden können – hier wird entweder der Datenträger, der Arbeitsspeicher oder der Prozessor zum Flaschenhals und die GUI wird träge. Um dies zu verhindern und den Zeichenvorgang erheblich zu beschleunigen, werden die Peaks beim Laden des Streams vorberechnet. Dieser Stream stellt je nach Zoomstufe und Konfiguration die benötigten Samples oder Peaks zur Verfügung, um den Zeichenvorgang von Audiodaten performant durchführen zu können. Die berechneten Peaks können in Peakfiles gespeichert und aus diesen geladen werden, so dass die Generierung für eine Audiodatei nur ein einziges Mal erfolgen muss.

**VolumeClipStream**


**VolumeControlStream**


**VolumeMeteringStream**

Ermittelt den maximalen Samplewert für eine per Parameter definierbare Sequenz von Samples und feuert jeweils ein Event, wenn die Messung für einen entsprechend langen Abschnitt erfolgt ist. Den Event-Listenern wird ein Array mit dem maximalen Absolutwert für jeden Kanal im Stream übergeben. Dieser Stream
Umsetzung des Systems

wird verwendet, um in der GUI ein VU-Meter darzustellen.

Tabelle 8: Übersicht der wichtigsten Audiostreams

3.5.3 Matching


![Vereinfachtes Klassendiagramm](image)

Abbildung 52: Vereinfachtes Klassendiagramm

Wie bereits erwähnt gibt es vier Arten, Matches zwischen Audioobjekten zu generieren: Fingerprinting, Time-Warping, Cross-Correlation und manuelles Anlegen. Egal welche Variante, oder welche Kombination von Varianten ausgeführt wird, am Ende landen alle generierten Matches in einer Liste, welche zur Synchronisation anhand des in Abschnitt 3.4 erläuterten Algorithmus abgearbeitet wird. Manuell angelegte Matches werden im GUI erstellt und direkt zur Liste hinzugefügt, es findet also kein algorithmisches Matching statt. Ebenfalls nicht genauer beleuchtet wird die Cross-Correlation, da sie zum aktuellen Zeitpunkt keine verlässlichen und sinnvoll verwertbaren Ergebnisse liefert. Wissenswert ist lediglich, dass die statische Klasse...
CrossCorrelation eine Funktion bereitstellt, die als Eingabe zwei Abschnitte aus zwei Audiostreams erwartet (zugeschnitten mittels CropStream) und als Ausgabe einen Offset liefert, aus dessen ein Match erstellt und zur Liste hinzugefügt werden kann. Die übrigen zwei Methoden und die Match-Filterung werden folgend näher betrachtet.

Fingerprinting und Time-Warping setzen beide die Short-Time-Fourier-Transformation ein, um die entsprechenden Feature-Vektoren zu generieren, die in weiterer Folge zu Sub-Fingerprints bzw. zur Distanzberechnung verwendet werden. Die Generierung wird von der Klasse STFT übernommen, welche eine Fensterfunktion und einen Audiostream als Eingabe benötigt. STFT selbst erweitert die Basisklasse StreamWindower, welche einen Audiostream durch eine parametrisierte Window- und Hop-Size (Kapitel 2.2.4) in sequentielle überlappende Frames aufteilt auf denen danach in STFT die Fensterfunktion angewandt und die Umwandlung durch FFT in die Frequenzdomäne erfolgt.

Um das Fingerprinting auszuführen, wird als erstes eine Instanz der Klasse FingerprintStore erstellt, in welcher alle Sub-Fingerprints für jede analysierte Audiospur gespeichert werden. Zusätzlich wird on-the-fly eine Hashtabelle aufgebaut, die gleiche bzw. ähnliche Sub-Fingerprints zusammenfasst. Nachdem alle benötigten Audiospuren analysiert sind, können die zusammengefassten Sub-Fingerprints der Hashtabelle als Einstiegspunkt zum Vergleich der vollständigen Fingerprints verwendet werden. Daraus folgt als Ausgabe eine Liste mit Matches. Zur Generierung der Sub-Fingerprints dient die Klasse FingerprintGenerator, die einen Audiostream als Eingabe annimmt, die entsprechenden Sub-Fingerprints erstellt und sie an den FingerprintStore weiterreicht. Zusätzlich wird er mit einem das Interface IProfile implementierenden Profil konfiguriert, das die Charakteristik der generierten Sub-Fingerprints steuert (Abschnitt 3.1.3.2).


Nachdem das Matching durchgeführt wurde und eine Liste mit Matches vorhanden ist, sorgt der MatchProcessor unter Zuhilfenahme des UndirectedGraph für die Aufteilung der Matches in MatchPairs, von denen jedes ein durch Matches verbundenes Audiospurenpaar definiert. Diese Paare werden zusätzlich in Gruppen vom Typ MatchGroup gruppiert, entsprechend der Aufteilung des Ursprungsgraphen in seine Zusammenhangskomponenten (Kapitel 3.4). Am Ende sorgt der MatchProcessor je nach Konfiguration für die Filterung der Matches, was schliessendlich zur finalen Matchliste führt, mit der die Synchronisation durchgeführt wird.
3.6 Synchronisierungsworkflow


Abbildung 53: AudioAlign nach dem Start

Liegen die zu synchronisierenden Dateien im WAVE-Format vor, können sie entweder durch Drag & Drop oder über das Menü („File“ > „Add audio file“) in die Timeline eingefügt werden. Um eine bessere Übersicht zu erlangen, können die Befehle „Resize track heights“ und „Zoom to fit“ verwendet werden, welche die Höhe der einzelnen Spuren an die Fenstergröße anpassen und den Zoom der Timeline so justieren, dass

\(^{29}\) [http://www.mediacoderhq.com/](http://www.mediacoderhq.com/)
Umsetzung des Systems

der die gesamte Laufzeit der Spuren in den sichtbaren Bereich gefasst wird. Da die Höhe der Spuren der Sichtbarkeit wegen nicht beliebig verkleinert werden kann, kann ab einer gewissen Anzahl von Spuren nicht auf ein vertikales Scrollen verzichtet werden. Alle Einstellungen können natürlich per Maus manuell nach Belieben verändert werden.


Alle gefundenen Matches werden in der Timeline als Verbindungslinien visualisiert, die Farbe der Linien entspricht der Qualität der Übereinstimmung. Umso grüner die Linie, umso besser die Übereinstimmung. Wird ein Match in der Liste ausgewählt, wird es in der Timeline hervorgehoben, indem rote Pfeile auf die Enden der Line zeigen. Matches
können nach Belieben gelöscht oder verändert werden. Zusätzlich können Matches über den Button „Add from timeline...“ manuell hinzugefügt werden. Dazu muss man zwei Audiospuren manuell in der Timeline synchronisieren, den Button betätigen und die entsprechenden zwei Spuren auswählen.

Abbildung 56: Visualisierung der gefundenen Matches


Abbildung 57: Synchronisierte Audiospuren mit den dafür verwendeten Matches

Nun kann die Synchronisierung validiert werden, indem man Hörproben an den Synchronisierungspunkten durchführt. Dazu kann der Playback-Cursor in der Timeline per Mausklick an eine beliebige Stelle gesetzt werden, das Abspielen wird mittels der „Play“-/„Pause“-Buttons links unter der Timeline, oder der Leertaste gesteuert. Damit

Umsetzung des Systems

Abbildung 60: Endergebnis der Synchronisierung (die obersten 11 Spuren konnten nicht synchronisiert werden)

Im eben skizzierten Workflow wurden nur kurze Clips mit höchstens ein paar Minuten Länge an eine lange (implizite) Masteraufnahme angeglichen, daher musste keine Rücksicht auf Time-Drift genommen werden. Wie sieht das Vorgehen jedoch aus, wenn mehrere lange Spuren miteinander synchronisiert werden sollen?

Abbildung 61: Matches von 5 Spuren mit einer Länge von jeweils ca. 1 Stunde

Angenommen es müssen fünf Audiospuren mit einer Länge von jeweils ca. einer Stunde, wie in Abbildung 61 zu sehen, synchronisiert werden. Dann ist davon auszugehen, dass ein Time-Drift vorhanden ist – vor allem wenn die Aufnahmen mit
Umsetzung des Systems


Bei den Spuren in Abbildung 61 tritt jedoch der Fall ein, dass gewisse Abschnitte nicht korrekt synchronisiert sind, was sich durch Echos beim Abspielen bemerkbar macht. Um dieses Problem zu beheben wird der zuvor durchgeführte Time-Drift-Ausgleich rückgängig gemacht, und eine neuerliche Synchronisierung ohne Driftausgleich durchgeführt. Jetzt sind die Matches gefiltert und bei 5 Spuren noch 4 Matches übrig, die jeweils ein Spurpaar an einer einzelnen Stelle synchronisieren. Daraufhin wechselt man in den Reiter „Dynamic Time Warping“ und betätigt den „Start“-Button. Der jetzt folgende Vorgang nimmt einige Zeit in Anspruch, ermittelt jedoch Matches in kleinen regelmäßigen Abständen, womit die Angleichung wesentlich feingranular durchgeführt werden kann. Um den Time-Drift genauer zu analysieren, kann der „Alignment-Graph“, zu finden links unter der Timeline, herangezogen werden.

Abbildung 62: Alignment-Graph zur Driftanalyse

In diesem Graph ist deutlich zu erkennen, dass zwischen einem Spurpaar ein geringer (grüne Linie), zwischen den drei weiteren Paaren ein erheblicher Time-Drift (jeweils ca. 7 Sekunden) existiert. Wäre zwischen einem Paar kein Drift vorhanden, wäre die entsprechende Linie horizontal. Zusätzlich sind in der grünen und violetten Linie zwei
Umsetzung des Systems


3.6.1 Grenzen


Abbildung 63: Matching einer kontinuierlichen Aufnahme mit einem nicht-monotonen Zusammenschnitt

Ein weiteres Problem kann das in der Musik weitverbreitete Sampling sein. Gemeint sind in diesem Fall nicht die Samples des digitalen Audiosignals, sondern die Praxis, neben den Instrumenten auch Tonaufnahmen einzuspielen. Dies geschieht vor allem bei Studioaufnahmen, wo z.B. wiederkehrende Refrains oftmals die gleichen Samples
Umsetzung des Systems

In der Umsetzung des Systems werden Praktiken eingesetzt, die in Liveauftritten von Interpreten oder Bands, entweder weil gewisse Elemente grundsätzlich durch Playback hinzugefügt werden, oder weil in Songs Sounds verwendet werden, die von keinem Bandmitglied live gespielt werden können. Da somit immer wieder gleiche Tonfolgen wiedergegeben werden, kann es wie in Abbildung 64 passieren, dass diejenigen Stellen einer Audiospur auf mehrere Stellen einer anderen Audiospur matchen. Diese gefundenen Matches sind dann zwar nicht grundsätzlich falsch, da sie auf sich gleichende Frequenzmuster verweisen, eine Synchronisierung nach ihnen würde jedoch zu einem unerwünschten Ergebnis führen. In den unzähligen Tests, die im Zuge der Entwicklung von AudioAlign durchgeführt wurden, kam es zwar nie zu so einem Fall – das absichtlich provozierte Beispiel in Abbildung 64 zeigt aber, dass dieser Fall nicht grundsätzlich ausgeschlossen werden kann.

Abbildung 64: Matching von zwei Spuren mit gesampelten Abschnitten

3.7 Analyse

Die maximal erreichbare Synchronisierungsgenauigkeit hängt bei durch Fingerprinting und Time-Warping gefundenen Matches von der Hop-Size der zu Grunde liegenden Short-Time-Fourier-Transformation ab. Sie beträgt dabei 12 bzw. 20 Millisekunden, was einer Abweichung von ±6ms beim Fingerprinting und ±10ms beim Time-Warping entspricht. Solch geringe Abweichungen können bei den von dieser Arbeit anvisierten Anwendungsfallen vernachlässigt werden. Die Erfahrung zeigt, dass Fingerprint-Matches mit einer Bit-Error-Rate unter 0,35 in jedem Fall korrekt sind. Verschiebt man den Threshold auf 0,45, treten gelegentlich False-Positives auf – sie fallen jedoch nicht ins Gewicht, da normalerweise genug bessere Matches gefunden werden, sodass diese beim Filtern aussortiert werden. Auch das Time-Warping führt immer zu korrekten Resultaten, sofern es auf vorsynchronisierten Spuren ausgeführt wird.

Bei der Erläuterung des Synchronisierungsworkflows in Kapitel 3.6 wurde angemerkt, dass eine erfolgte Synchronisierung durch Hörproben validiert werden kann und sollte. Dabei drängt sich die Frage auf, ob es nicht möglich ist, die Synchronisierungsqualität algorithmisch zu bestimmen. Idealerweise sollte die Anwendung dem Anwender mitteilen, ob eine Synchronisierung korrekt oder fehlerhaft ist. Unglücklicherweise brachte eine Literaturrecherche keine anerkannten Methoden zu Tage, die für diese Aufgabe ausgelegt sind, daher wurden aufbauend auf den in Kapitel 2 vorgestellten Signalverarbeitungsmethoden vier selbst erdachte Varianten getestet, mit denen so eine Messung eventuell durchgeführt werden könnte.

Allen vier Methoden ist gemeinsam, dass sie nicht kontinuierlich auf alle Spuren in der Timeline angewendet werden, sondern nur auf in regelmäßigen Abständen entnommenen Ausschnitten. Eine Anwendung auf die vollständigen Signale würde je
Umsetzung des Systems

nach Methode schon bei Spuren mit Längen von unter einer Minute zu stundenlangen Berechnungszeiten führen. Die Breite der zu entnehmenden und analysierenden Ausschnitte, das Intervall in welchem die Ausschnitte entnommen werden und die Samplingrate der zu analysierenden Audiodaten können frei festgelegt werden, sollten aber an die Länge der zu analysierenden Timeline angepasst werden – dabei muss ein Kompromiss zwischen Granularität und Berechnungszeit gefunden werden. Eine geringe Samplingrate beschleunigt die Berechnung, vermindert jedoch die Genauigkeit des Ergebnisses. Ergebnis ist jeweils ein Prozentsatz zwischen 0% und 100%, der konzeptionell gesehen die Genauigkeit der Synchronisation angibt. Der Wert ist der Durchschnitt aller pro Ausschnitt berechneter Werte, welche wiederum Durchschnitte aller zwischen sich in den Ausschnitten befindlichen Spurpaaren ermittelten Absolutwerten sind.

Leider kann vorweggenommen werden, dass alle vier Methoden nicht dazu geeignet sind, die Synchronisierungsklärheit absolut zu beurteilen. In vielen Fällen kann eine Tendenz zwischen einer bekannten nicht synchronisierten und bekannten synchronisierten Timeline abgelesen werden, einige Testfälle haben jedoch für unsynchronisierte Timelines bessere Ergebnisse als für ihre synchronisierten Gegenstücke ergeben – obwohl eine Hörprobe das offensichtliche Gegenstück bewies. Ist der Synchronisierungsstatus einer Timeline also unbekannt, können die berechneten Werte nicht verlässlich zur Ermittlung der Synchronität verwendet werden. Ein Vorher/Nachher-Vergleich gibt zwar oftmals Hinweis darauf, ob die Synchronisierung besser geworden ist, jedoch nicht ob sie gut oder schlecht ist. Worin genau diese Problematik liegt, bzw. wie sie behoben werden könnte, wurde nicht untersucht – ein möglicher Grund könnte sein, dass nicht bekannt ist, ob sich die Inhalte der einzelnen analysierten Abschnitte überhaupt überschneiden und die Ergebnisse im negativen Fall mit Zufallswerten gleichgesetzt werden können. Außerordentlich werden sich Aufnahmen, die an unterschiedlichen Orten mit eventuell unterschiedlichen Geräten erstellt wurden, niemals exakt gleichen. Die einzelnen Methoden werden in der folgenden Tabelle beschrieben:

<table>
<thead>
<tr>
<th>Methodenbezeichnung</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Correlation-Offset</td>
<td>Hierbei wird der Offset zwischen Signalabschnitten durch Cross-Correlation berechnet und mit dem Offsetkoeffizienten normalisiert. Ist der Offset 0, werden beide Signalabschnitte als zu 100% synchron angesehen. Entspricht er dem maximal möglichen Offset, also der halben Länge der Signalabschnitte, werden die Abschnitte als vollständig unsynchron definiert und mit 0% bewertet.</td>
</tr>
<tr>
<td>Correlation</td>
<td>Berechnet den Korrelationskoeffizienten bei Offset 0. Sind beide Signalabschnitte ident bzw. nur durch einen DC-Offset verschoben oder invers, ist der Koeffizient 1 und der Wert somit 100%. Sind die Signale vollständig verschieden, ist der Koeffizient 0 (0%).</td>
</tr>
<tr>
<td>Interference</td>
<td>Ermittelt die Ähnlichkeit durch Anwendung der destruktiven Interferenz. Die Signale werden vom DC-Offset befreit und überlagert, indem ein Signalabschnitt vom anderen abgezogen wird. Die Samplewerte vom resultierenden Signal werden dann</td>
</tr>
</tbody>
</table>
Umsetzung des Systems

summiert und durch die Anzahl der Samples geteilt. Löschen sie sich vollständig aus, sind sie ident und resultieren daher in 100%iger Ähnlichkeit.

**Frequency Distribution**

Die Ähnlichkeit wird mittels der Summe der absoluten Differenzen der einzelnen durch die Gesamtenergie normalisierten Spektralwerte berechnet, welche aus den per Hann-Funktion gefensterten Signalabschnitten durch die FFT ermittelt werden. Eine vollständig gleiche Frequenzverteilung ergibt eine Ähnlichkeit von 100%.

**Tabelle 9: Methoden zur Analyse der Synchronisierungsqualität**

Das folgende Beispiel zeigt die Anwendung der Methoden auf zwei Audiospuren mit einer Länge von jeweils ca. einer Stunde. Die Messungen wurden auf Abschnitten von jeweils 5 Sekunden im Abstand von 5 Minuten, bei einer Samplingrate von 22050 Hz entnommen.

Abbildung 65 zeigt zwei unsynchronisierte Spuren mit einem Offset von ca. 2 Minuten. In den zur Messung entnommenen Abschnitten mit 5 Sekunden Länge findet daher keinesfalls eine Überschneidung ihrer Inhalte statt. Die Analyse ergibt für den Cross-Correlation-Offset eine Synchronisierungsqualität von 5%, für die Correlation 1%, Interference 69% und für die Frequency Distribution 26%. Einzig die Interference-Methode deutet auf eine eventuell vorliegende Synchronisierung hin, die anderen Methoden ergeben so schlechte Werte, dass von einer Unsynchronität ausgegangen werden kann.

Nachdem die Spuren durch Fingerprinting und OLTW angeglichen wurden (Abbildung 67), und Hörproben die tatsächliche Synchronität bestätigt haben, ergeben sich folgende Werte: Cross-Correlation-Offset 34%, Correlation 12%, Interference 68% und Frequency Distribution 39%. Die Interference-Methode erwies sich schon im unsynchronen Fall als unqualifizierter Maßstab, und liefert jetzt sogar einen, wenn
Umsetzung des Systems

auch gering, schlechteren Wert. Alle anderen Methoden weisen durch teils gravierende Anstiege auf eine bessere Synchronisierung hin, eine erfolgreiche und qualitativ gute Synchronisierung lässt sich daraus jedoch nicht ableiten.

Abbildung 67: Synchronisierte Spuren

Abbildung 68: Ermittelte Synchronisierungsqualität der Spuren aus Abbildung 67 (v.l.n.r.: Cross-Correlation-Offset, Correlation, Interference, Frequency Distribution)

Werden die beiden Spuren manuell durch Anlegen zweier Matches am Anfang und Ende der Überschneidung synchronisiert, ergeben sich im Vergleich zur automatischen Synchronisation folgende Werte: Cross-Correlation-Offset 34%, Correlation 18%, Interference 66% und Frequency Distribution 39%. Die erste und letzte Methode resultiert im exakt selben Ergebnis wie bei der automatischen Angleichung, die übrigen zwei Werte weisen geringe Abweichungen auf.

Abbildung 69: Ermittelte Synchronisierungsqualität der manuell synchronisierten Spuren (v.l.n.r.: Cross-Correlation-Offset, Correlation, Interference, Frequency Distribution)

Die algorithmische Beurteilung der Synchronisierungsqualität bleibt also ein offener Punkt, bei dem es weiterer Untersuchungen bedarf. Eventuell können durch statistische Untersuchungen Thresholds gefunden werden, welche eine Unterscheidung zwischen synchron und unsynchron ermöglichen bzw. ob und welche Methoden überhaupt für diese Beurteilung anwendbar sind. Außerdem ist es denkbar, dass Messungen globaler Eigenschaften von Audiospuren, wie z.B. Lautstärke, Dynamik oder Frequenzverteilung genutzt werden können, um die generellen Charakterunterschiede zwischen verschiedenen Aufnahmen durch Normalisierung aufzuheben. Da eine aufs Sample genaue Angleichung durch die in dieser Arbeit eingesetzten Synchronisierungsmethoden (mit Ausnahme der Cross-Correlation) nicht
erreicht werden kann und für ein akustisch gutes Ergebnis in den meisten Fällen auch nicht erreicht werden muss, machen Analysemethoden, die auf Sampleebene operieren (wie z.B. Interference), keinen Sinn – stattdessen sollten etwas gröbere und dafür gegen unterschiedliche Charakteristiken robuster Methoden gefunden werden, die auf der Ebene von aus dem Signal extrahierten Features operieren.
4 Evaluation


4.1 Speicher- und Laufzeitanalyse

Im Folgenden wird die Speicher- und Laufzeitcharakteristik der Matching-Algorithmen analysiert. Fingerprinting sowie Time-Warping haben linearen Zeit- und Speicherbedarf, Cross-Correlation ist die einzige Ausnahme mit quadratischem Zeitbedarf bei linearem Speicherbedarf. Die gesamte Analyse wurde auf einem durchschnittlichen x64-System mit Windows 7 SP1, AMD Phenom II 3 GHz Quad-Core-Prozessor und 8 GB DDR2-800 RAM durchgeführt. Die verwendeten Audiodateien waren im Audio-CD-Format von 44,1 kHz, 16 Bit, Stereo. Die ermittelten Werte sind ein Durchschnitt aus jeweils 10 Durchläufen.

Beim Fingerprinting dauert die Analyse der Audiospuren 1,5 Sekunden pro Minute Audiodaten. Hier ist zu beachten, dass bei der Fingerprinting-Analyse die Audiospuren auf 11 kHz Mono hinunter gerechnet werden, was ebenfalls eine bestimmte Bearbeitungszeit kostet. Würden die Dateien daher im Fingerprinting-Analyse-ZielfORMAT vorliegen, wäre die Analyse schneller – die verwendeten Dateien entsprechen jedoch dem derzeitigen Standard und sind daher ein realitätsnahes EingabefORMAT. Stehen mehrere Spuren zur Analyse, wird der Vorgang parallelisiert und auf die verfügbaren Prozessorkerne aufgeteilt. So werden auf dem Quad-Core-Prozessor bis zu vier Spuren gleichzeitig analysiert. Da ein Bruchteil der Zeit auch vom Betriebssystem in Anspruch genommen wird, steigt die Verarbeitungsgeschwindigkeit nicht ganz auf das Vierfache, die Steigerung ist jedoch trotzdem beträchtlich – eine Minute wird pro Kern in 2,1 Sekunden analysiert, das entspricht insgesamt vier Minuten in 4,2 Sekunden bzw. 32 Sekunden pro Audiominute Laufzeit der Analyse.

<table>
<thead>
<tr>
<th>Länge (Sekunden)</th>
<th>Zeit (Sekunden)</th>
<th>Speicher (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Tabelle 10: Fingerprinting Laufzeit- und Speicherverbrauch
Der Speicherverbrauch beim Fingerprinting setzt sich aus den generierten Sub-Fingerprints der Analyse und der Lookup-Tabelle zusammen. Pro 11,6 Millisekunden Audiodaten wird ein Sub-Fingerprint mit einer Größe von 32 Bit (4 Bytes) erstellt und als Integer in einer Array-Liste abgelegt, mit Ausnahme der letzten 3 Sekunden, die jedoch einen minimalen konstanten Faktor darstellen und daher nicht berücksichtigt werden. Dies ergibt ca. 5172 Sub-Fingerprints pro Minute, oder ca. 20 Kilobytes an Daten. Jeder Sub-Fingerprint wird außerdem der Lookup-Tabelle hinzugefügt, damit später gleiche Sub-Fingerprints in verschiedenen Spuren gefunden werden können. Ein Eintrag in dieser Tabelle besteht aus einer Referenz auf die entsprechende Audiospur (8 Bytes) plus dem Index des Sub-Fingerprints als Positionsmerke (Integer, 4 Bytes). Da durch die Methode des Zuverlässigkeitsrankings für jeden Sub-Fingerprint noch 7 alternative Sub-Fingerprints berechnet werden, kommen insgesamt 8 Einträge pro Sub-Fingerprint in die Lookup-Tabelle. Pro Sub-Fingerprint werden also 8*12 Bytes in der Lookup-Tabelle benötigt, was insgesamt ca. 485 Kilobytes pro Minute entspricht.

Time-Warping ist langsamer als Fingerprinting, da es immer paarweise durchgeführt werden muss und es so bei mehreren Spuren passiert, dass gewisse Spuren mehrmals vom Algorithmus durchlaufen werden. Hier könnten zwar die berechneten Frame-Werte zwischengespeichert und so eine Laufzeitoptimierung durchgeführt werden, zurzeit ist dies jedoch nicht implementiert. Es ist jedoch hauptsächlich deswegen langsamer, da beim Aufbau der Matrix sehr viele Berechnungen durchgeführt werden. Der Pfad durch die Matrix wird zwar auf eine Breite von 10 Sekunden beschränkt, trotzdem muss jeder Wert der einen Spur mit allen Werten der anderen Spur, die in die Fensterbreite passen, verglichen werden. Um eine Minute Audiodaten von einem Spur-Paar anzulegen, werden ca. 10 Sekunden benötigt. Die Angleichung einer Stunde Audio nimmt somit 10 Minuten in Anspruch, was aus Benutzersicht eine ziemlich lange Zeit ist. Glücklicherweise muss diese Berechnung jedoch nicht in allen Anwendungsfällen durchgeführt werden.

Time-Warping arbeitet mit Frame-Größen von 20 Millisekunden, welche jeweils einen Spektralvektor mit 84 Werten ergeben, der insgesamt 336 Bytes in Anspruch nimmt. Pro Sekunde ergibt das einen Speicherbedarf von 16 Kilobytes. Da die Frames aber nur für die Breite des Fensters vorgehalten werden müssen (160 Kilobytes pro Spur), entspricht dies ebenfalls einem konstanten Faktor der vernachlässigt werden kann. Die Matrix besteht aus Double-Werten (8 Bytes), ihr Speicherbedarf entspricht der Länge
der längeren Spur des Spurpaares multipliziert mit der Fensterbreite. Bei 10 Sekunden Breite (500 Frames) ergibt dies einen Bedarf von ca. 12 Megabytes pro Minute.

Die Cross-Correlation ist der einzige Algorithmus mit größerer als linearer Laufzeit. Seine Laufzeit steigt so schnell an, dass der benötigte lineare Speicherbedarf in jenen Fällen, in denen eine Berechnung in einer akzeptablen Zeitspanne durchgeführt werden kann, so gering ist, dass er nicht erwähnt werden muss. Fälle, bei denen der Speicherbedarf interessant wird, würden eine Wochen- oder jahrelange Berechnungszeit benötigen. Tabelle 12 zeigt, dass die Berechnung auf einem Audiodatenabschnitt von einer Sekunde Länge nur 1,2 Sekunden dauert, bei zwei Sekunden dauert die Berechnung jedoch schon 4,5 Sekunden.

![Diagramm: Berechnungszeitverbrauch](image)

Tabelle 12: Cross-Correlation Laufzeitverbrauch

### 4.2 Testdatensätze

Die nachfolgenden Testdatensätze sind eine subjektive Auswahl von Situationen, in denen eine automatische Synchronisation ein großes Hilfsmittel darstellt, ein bestimmtes Ziel zu erreichen. Manche Situationen sind aus dem „echten Leben“ gegriffen, andere zwar künstlich erstellt, aber gut geeignet um die Funktionalität von AudioAlign zu testen. Obwohl die Auswahl subjektiv erfolgte, wurde Acht gegeben, eine ausgeglichene Auswahl zwischen Datensätzen mit guten und schlechten Resultaten zu treffen. Sie stammen aus einer für die Entwicklung der Anwendung erstellten Bibliothek mit 20 Testdatensätzen, die zufällig entweder aus einem privaten Fundus von mehreren tausend Konzertaufnahmen oder aus dem Internet (YouTube) stammen, oder zum Testen erstellt wurden.

Die Testdatensätze werden jeweils kurz beschrieben und die darin vorkommenden Audio- bzw. Videoaufnahmen näher definiert. Danach folgt eine Auswertung der Fingerprints, indem alle auffindbaren Fingerprints ermittelt werden, um das Verhältnis zwischen der Anzahl aller Fingerprints und jener, die den Threshold und das True-Positive-Kriterium erfüllen, zu ermitteln. Alle Diagramme basieren auf dem Similarity-Maß der Matches, das beim Fingerprinting invers zur Bit-Error-Rate (1 − BER) ist. Ist die Similarity gleich 1, sind zwei Fingerprints 100% identisch. Dies bedeutet jedoch nicht zwingendermaßen, dass der aus dem Fingerprint entspringende Match auch logisch korrekt ist, da auch einer der in Abschnitt 3.6.1 beschriebenen Spezialfälle eintreten kann. Der Standard-Threshold liegt bei einer BER von 0,45 bzw. Similarity von

Abbildung 70: Timeline mit offensichtlich erkennbaren False-Positive-Matches


### 4.2.1 Testdatensatz 1 (Musik)


<table>
<thead>
<tr>
<th>Spur</th>
<th>Quelle</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DPA4061 Lavalier-Mikrofone &gt; iRiver H120 MP3-Player</td>
<td>3 sequenzielle Songs</td>
</tr>
<tr>
<td>2</td>
<td>Canon HF10 HD-Camcorder</td>
<td>1 Song</td>
</tr>
<tr>
<td>3</td>
<td>Canon HF10 HD-Camcorder</td>
<td>1 Song</td>
</tr>
</tbody>
</table>

Tabelle 13: Aufnahmen von Testdatensatz 1
In diesem Fall werden nicht besonders viele Matches gefunden. Da die Aufnahmen jedoch kurz sind und kein Driftausgleich notwendig ist, sind sie für eine erfolgreiche Synchronisierung vollkommen ausreichend. Die Anwendung des „Buggy“-Profils würde bei diesem Datensatz eine stark erhöhte Anzahl von True-Positives zurückliefern, aus dem eben genannten Grund sind diese Matches jedoch nicht notwendig.

Abbildung 72: Timeline mit Matches von Testdatensatz 1

Insgesamt resultiert der Testdatensatz in 657 Matches, von denen 51 über dem Similarity-Threshold liegen. Alle diese 51 Matches sind True-Positives und können zur Synchronisation verwendet werden. In diesem Fall wird aber pro Spurpaar nur das Match mit der höchsten Similarity zur Synchronisation verwendet.

Abbildung 73: Similarity-Verteilung von Testdatensatz 1 (links gesamt, rechts über Threshold)

Abbildung 74: Similarity-Histogramm von Testdatensatz 1
4.2.2 Testdatensatz 2 (Musik)


<table>
<thead>
<tr>
<th>Spur</th>
<th>Quelle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YouTube &quot;broke amplifier in flames metal camp 08&quot; (Q8JbVawxxY)</td>
</tr>
<tr>
<td>2</td>
<td>YouTube &quot;cloud connected in flames metal camp 2008&quot; (BL5kEpePLFY)</td>
</tr>
<tr>
<td>3</td>
<td>YouTube &quot;colony in flames metal camp 2008&quot; (SOQQnQ4f35A)</td>
</tr>
<tr>
<td>4</td>
<td>YouTube &quot;In Flames - Cloud Connected @ METALCAMP 2008&quot; (T4094tQmigk)</td>
</tr>
<tr>
<td>5</td>
<td>YouTube &quot;IN FLAMES - Colony (Metalcamp 2008)&quot; (sxVua5RD_0)</td>
</tr>
<tr>
<td>6</td>
<td>YouTube &quot;In Flames - Come clarity @ Metal camp08&quot; (eLEeVm4fvNE)</td>
</tr>
<tr>
<td>7</td>
<td>YouTube &quot;In flames - Delight and angers @ Metal camp08&quot; (jRxSMkc0nNA)</td>
</tr>
<tr>
<td>8</td>
<td>YouTube &quot;In Flames - Disconnected (Metalcamp 2008)&quot; (yVfAm-YAcMY)</td>
</tr>
<tr>
<td>9</td>
<td>YouTube &quot;In Flames - Graveland (live 2008-07-04 Metalcamp)&quot; (FmEQVlm3g3y)</td>
</tr>
<tr>
<td>10*</td>
<td>YouTube &quot;In Flames - Move Through Me (live 2008-07-04 Metalcamp)&quot; (12KcF3261Fs)</td>
</tr>
<tr>
<td>11*</td>
<td>YouTube &quot;In Flames - My Sweet Shadow (live 2008-07-04 Metalcamp)&quot; (b_pvQqQ9E5_8)</td>
</tr>
<tr>
<td>12*</td>
<td>YouTube &quot;In Flames - Only For The Weak (Metalcamp 2008)&quot; (WvLnJELu2nw)</td>
</tr>
<tr>
<td>13</td>
<td>YouTube &quot;In Flames - The Mirrors Truth (Metalcamp 2008)&quot; (g6_kzS8WE7A)</td>
</tr>
<tr>
<td>14</td>
<td>YouTube &quot;In FLAMES - The Mirror's Truth (Metalcamp 2008)&quot; (ZA294JvCqNE)</td>
</tr>
<tr>
<td>15</td>
<td>YouTube &quot;In Flames - Trigger @ MetalCamp 08&quot; (Ix_uqmeES1XQ)</td>
</tr>
<tr>
<td>16</td>
<td>YouTube &quot;In Flames: Bum !&quot; (osUrm migrants)</td>
</tr>
<tr>
<td>17</td>
<td>YouTube &quot;In Flames: Cloud Connected&quot; (ZzgQux9Pu0I)</td>
</tr>
<tr>
<td>18</td>
<td>YouTube &quot;In Flames: Delight and Angers&quot; (a-n9WfhG4tQ)</td>
</tr>
</tbody>
</table>

Bei diesem Testdatensatz werden über 88.000 Matches gefunden, von denen aber nur 224 über dem Threshold liegen – was einer geringen Erfolgsquote von nur 0,25% entspricht. Davon fällt der Großteil auf die drei mit * gekennzeichneten Spuren, da sie in gewisser Weise Teile der Referenz-Audiospur beinhalten. Alle dieser 224 Matches sind True-Positives.
Sieht man sich die synchronisierten Videos parallel an, stellt man fest, dass die Bilder trotz der korrekten Audiosynchronisation asynchron sind. Dies liegt daran, dass der Bereich vor der Bühne, aus dem die Besucher die Videos aufgenommen haben, ziemlich weitläufig ist. Durch die hohe Lichtgeschwindigkeit ergibt sich bei den Videospuren zwar keine Verzögerung, dies gilt aber leider nicht für die Audiospuren. Durch die Schallgeschwindigkeit von 343 m/s ergibt sich bei einer Entfernung von 30 Metern, was ein durchaus realistischer Wert ist, eine Verzögerung von 87 Millisekunden, die einem Offset von 3 Bildern bei einer Bildwiederholfrequenz von 25 Hz entspricht.

Abbildung 75: Timeline mit Matches von Testdatensatz 2
4.2.3 Testdatensatz 3 (Musik)


Abbildung 78: Timeline mit Matches von Testdatensatz 3
In diesem Fall sind die Spuren jeweils ca. 30 Minuten lang und ein leichter Time-Drift mit einem Maximaloffset von ca. 130 ms ist zu bemerken. Hier ist es zwingend erforderlich einen Driftausgleich durchzuführen, was jedoch hauptsächlich am Inhalt der Spuren liegt. Durch die bei dieser Musikrichtung übliche schnelle Spielweise ist vor allem beim Schlagzeug eine Asynchronität durch eine Verdoppelung bzw. Vervielfachung der Anschläge schnell zu bemerken. Der Einsatz von sogenannten Blastbeats, einer Reihe von Trommelschlägen mit hoher Geschwindigkeit (180 Beats-per-Minute und mehr) stellt den Synchronisierungsmechanismus hart auf die Probe. Zur Synchronisierung der beiden Audiospuren, die für den Audio-Mix benötigt werden, wäre daher der zusätzliche Einsatz von Time-Warping dringend zu empfehlen. Grundsätzlich sind aber alle durch das Fingerprinting gefundenen Matches genau genug, um auch in schnellen Abschnitten nicht extrem negativ aufzufallen.

Abbildung 79: Alignment-Graph von Testdatensatz 3

Abbildung 80: Similarity-Verteilung von Testdatensatz 3 (links gesamt, rechts über Threshold)
4.2.4 Testdatensatz 4 (Musik)

104  Evaluation

<table>
<thead>
<tr>
<th>Spur</th>
<th>Quelle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DPA4061 Lavalier-Mikrofone &gt; Sony MZ-NH900 Minidisc-Rekorder</td>
</tr>
<tr>
<td>2</td>
<td>Sony ECM-TS125 Stereo-Mikrofon &gt; Sony MZ-N10 Minidisc-Rekorder</td>
</tr>
<tr>
<td>3</td>
<td>Vivanco EM 216 Stereo-Ansteck-Mikrofon &gt; Aiwa AM-F65 Minidisc-Rekorder</td>
</tr>
<tr>
<td>4</td>
<td>Panasonic NV-GS11 MiniDV-Camcorder</td>
</tr>
<tr>
<td>5</td>
<td>unbekannte Digicam</td>
</tr>
</tbody>
</table>

Tabelle 16: Aufnahmen von Testdatensatz 4


Abbildung 82: Timeline mit Matches von Testdatensatz 4

Abbildung 83: Similarity-Verteilung von Testdatensatz 4 (links gesamt, rechts über Threshold)
4.2.5 Testdatensatz 5 (Sprache)

Dieser Testdatensatz ist künstlich erstellt und beinhaltet 8 Aufnahmen eines vorgelesenen Textes mit ca. zwei Minuten Länge. Zur Erstellung der Aufnahme wurden im Abstand von ein bis zwei Metern zum Mund sieben verschiedene Aufnahmegeräte platziert, das Headset für die übrige Aufnahme wurde direkt am Kopf getragen. Hier gibt es keinen konkreten Anwendungsfall, da die Aufnahmen in steriler und studioähnlicher Umgebung aufgenommen wurden und man in so einer Situation normalerweise keine Aufnahmen mit niederwertigem Equipment durchführt (Spuren 5 – 8), sofern höherwertiges Equipment zur Verfügung steht (Spuren 1 – 4). Der Datensatz dient jedoch als Test bzw. Beleg dafür, dass auch Sprachaufnahmen problemlos miteinander synchronisiert werden können.

<table>
<thead>
<tr>
<th>Spur</th>
<th>Quelle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Plantronics Headset &gt; Realtek ALC889A HD Audio Codec</td>
</tr>
<tr>
<td>2</td>
<td>Studio Projects C4 Kondensatormikrofon (Niere) &gt; Edirol UA-5 &gt; M-Audio MicroTrack 24/96</td>
</tr>
<tr>
<td>3</td>
<td>Studio Projects C4 Kondensatormikrofon (Hyperniere) &gt; Edirol UA-5 &gt; M-Audio MicroTrack 24/96</td>
</tr>
<tr>
<td>4</td>
<td>DPA4061 Lavalier-Mikrofon &gt; iRiver H120 MP3-Player</td>
</tr>
<tr>
<td>5</td>
<td>Samsung WB2000 Digicam</td>
</tr>
<tr>
<td>6</td>
<td>Canon HF10 HD-Camcorder</td>
</tr>
<tr>
<td>7</td>
<td>Samsung Galaxy Spica (Android 2.2) Smartphone</td>
</tr>
<tr>
<td>8</td>
<td>Samsung Nexus S (Android 4.0.4) Smartphone</td>
</tr>
</tbody>
</table>

Hier werden ca. 10.500 Matches gefunden, wovon 8279 Matches über dem Threshold liegen – dies ergibt eine sehr hohe Erfolgsquote von 79%. Unter diesen Matches befinden sich 101 False-Positives, die bei einer Erhöhung des Threshold auf 0,65 alle entfallen würden. Selbst dann wären noch mehr als genug Matches übrig, um alle Spuren miteinander zu synchronisieren. Die ungefähre Länge der Spuren ist zwei Minuten, daher kann ein Time-Drift ausgeschlossen werden und es genügt ein Match pro Spurpaar für die Synchronisation.
Interessant bei diesem Testfall ist vor allem die hohe Anzahl gefundenen Matches, obwohl die Gesamtlänge der beteiligten Spuren im Vergleich zu den anderen Testfällen sehr viel geringer ist. Vermutlich ruht dies auf dem Umstand, dass alle Aufnahmen unter den gleichen Bedingungen, ohne irgendwelche Störeinflüsse und in ruhiger Umgebung aufgenommen wurden. Nebengeräusche sind somit so gut wie ausgeschlossen. Grundsätzlich erbringt dieser Testdatensatz den Beweis, dass die Synchronisierungsmethode nicht ausschließlich nur für Musik geeignet ist.
4.2.6 Testdatensatz 6 (Sprache)


<table>
<thead>
<tr>
<th>Spur</th>
<th>Quelle</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YouTube &quot;Aufzeichnung der Rücktrittserklärung von Josef Pröll&quot; (Qhj3uTEnNGY)</td>
<td>APA-OTS Videoservice</td>
</tr>
<tr>
<td>2</td>
<td>YouTube &quot;Josef Pröll erklärt Rücktritt&quot; (PyZg_o11l38)</td>
<td>Handy oder Digicam</td>
</tr>
<tr>
<td>3</td>
<td>YouTube &quot;Josef Pröll tritt zurück - Pressekonferenz 13.4.2011 um 11 Uhr&quot; (Bo1Cd6LKa3Q)</td>
<td>ORF2-Liveübertragung</td>
</tr>
<tr>
<td>4</td>
<td>YouTube &quot;Rücktritt von Josef Pröll&quot; (GyXR2v1ND3U)</td>
<td>ÖVP Videos</td>
</tr>
</tbody>
</table>

Tabelle 18: Aufnahmen von Testdatensatz 6

Dieser Testdatensatz stellt das Gegenteil zum vorhergehenden Fall dar, nämlich Sprachaufnahmen unter Einfluss von Stör- und Umgebungsgeräuschen. Hier werden ca. 5400 Matches gefunden, von denen 2529 über dem Threshold liegen (Erfolgquote ca. 50%). Auch in diesem Fall ist kein Time-Drift bemerkbar, die Synchronisation kann durch ein Match pro Spurpaar erfolgen und endet wiederum in einem guten Ergebnis.
Ein interessantes Ergebnis der Similarity-Verteilungen ist, dass jene Matches über dem Threshold für Musik (Testdatensätze 1 – 4) sowie für Sprache (Testdatensätze 5 und 6) jeweils ähnliche Verteilungskarakteristiken aufweisen, wohingegen sich die Charakteristik zwischen Musik und Sprache stark unterscheidet. Auch das Verhältnis von insgesamt gefundenen Matches zu True-Positives unterscheidet sich stark – bei Musik nehmen die True-Positives nur einen sehr kleinen Teil unter allen Matches ein, bei Sprache überwiegen sie. Dies könnte jedoch daran liegen, dass die
Sprachaufnahmen der Testdatensätze von eher guter Qualität mit wenig Rauschen sind, während die Musikaufnahmen teilweise von sehr schlechter Qualität sind.


4.3 Vergleich mit Softwareprodukten

wurde, existierte keine öffentlich verfügbare Software, die einen automatischen Time-Drift-Ausgleich durchführt und zurzeit stellt sie die einzige „Konkurrenz“ dar.

<table>
<thead>
<tr>
<th></th>
<th>TDS 1</th>
<th>TDS 2</th>
<th>TDS 3</th>
<th>TDS 4</th>
<th>TDS 5</th>
<th>TDS 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AudioAlign</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Adobe Premiere Pro</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MATCH</td>
<td>Keine Synchronisation möglich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VocALign</td>
<td>Keine Synchronisation möglich</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PluralEyes</td>
<td>✓</td>
<td>✓</td>
<td>~(*)</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DualEyes</td>
<td>(**)</td>
<td>(**)</td>
<td>(*** )</td>
<td>(**)</td>
<td>(*** )</td>
<td>(*** )</td>
</tr>
</tbody>
</table>

Tabelle 19: Vergleich der Softwareprodukte anhand der Testdatensätze

* nur 2 von 5 Spuren werden miteinander synchronisiert und sind durch den hohen Time-Drift in den Aufnahmen auch nur kurzzeitig synchron

** keine globale Synchronisation möglich / DualEyes verwendet den gleichen Algorithmus wie PluralEyes und würde daher prinzipiell die Synchronisation durchführen können, der Testfall schlägt jedoch fehl, da das entsprechende Video(container)format (TS/AVCHD in Testdatensatz 1, MP4/AVC in Testdatensatz 2) nicht unterstützt wird

*** DualEyes kann für den Testdatensatz nicht angewendet werden, da er aus Audiodateien besteht

Ähnliche Arbeiten


5.1 Synchronization of Multi-Camera Video Recordings Based on Audio

In [52] beschäftigen sich die Autoren mit der Tatsache, dass heutzutage eine immer weiter steigende Anzahl von Personen in vielen Situationen Videoaufnahmen erstellt, wie z.B. bei Hochzeiten, Partys oder im Urlaub. Aus diesem Grund passiert es oft, dass bei einem einzelnen Event Videos aus unterschiedlichen Winkeln und zu unterschiedlichen Zeitpunkten erstellt werden, was zu einer größeren Abdeckung der Vorgänge führt. Es besteht daher die Möglichkeit, diese Aufnahmen miteinander zu Zusammenfassungen zu kombinieren, indem die qualitativ besten Ansichten ausgewählt werden.

zu berechnen, die zur temporalen Positionierung der Videos in einer Timeline verwendet werden.


Die Autoren schließen mit der Feststellung, dass dieser Fingerprinting-Algorithmus durch seine Robustheit gegenüber Rauschen, seine ausreichende Genauigkeit von 11,6 ms und seine Minimalanforderung von nur 3 Sekunden Überschneidung zwischen Aufnahmen gut für eine Synchronisierung geeignet ist.

5.2 Less Talk, More Rock: Automated Organization of Community-Contributed Collections of Concert Videos


Audioqualität kein Match gefunden wird, da diese in einer Zusammenfassung, in der es um die optimierte Repräsentation eines Events geht, ohnehin keinen Platz hätten.

Die durch das Fingerprinting gewonnenen Matches werden dann verwendet, um einen Graphen zwischen sich überlappenden Videos zu erstellen, ähnlich zu Kapitel 3.4. Auch hier repräsentieren die Clips einzelne Knoten im Graph, zwischen sich überlappenden Videos werden Kanten gezogen. Aus diesem Graph lesen die Autoren einige interessante Eigenschaften der Videos heraus: Erstens besteht der Graph üblicherweise aus mehreren Zusammenhangskomponenten – nur im Fall einer Aufnahme die das gesamte Konzert umfasst, oder, falls so viele sich überlappende Videos existieren, dass sie das gesamte Konzert abdecken, generiert das System einen auf Grund weiterer darauf aufbauenden Verarbeitungsschritte unerwünschten einzelnen zusammenhängenden Graphen. Zweitens kann beobachtet werden, dass die Komponenten üblicherweise nicht vollständig verbunden sind. Dies liegt daran, dass die Überlappung nicht transitiv ist, da sich bei Überlappung von Clip A mit Clip B und Clip B mit Clip C nicht unbedingt Clip A mit Clip C überlappen muss. Außerdem werden manche Überlappungen einfach nicht erkannt, z.B. weil zu viel Rauschen in den Audiospuren enthalten ist.

Aus dem generierten Graph wird anschließend der Level-of-Interest ermittelt, um die Frage zu beantworten, ob ein Clip einen besonders interessanten Moment eines Konzerts einfängt. Die Autoren stellen dazu die These auf, dass größere Komponenten im Graph einem größeren subjektiven Interesse entsprechen und beweisen sie durch ein Testset mit insgesamt ca. 600 von YouTube heruntergeladenen Clips von drei Konzerten, mit einer summierten Laufzeit von 23 Stunden. Die Größe der Komponenten kann entweder durch die Anzahl der Knoten oder den Verbindungsgrad gemessen werden. Die Untersuchung ergab, dass große Komponenten üblicherweise den Hit-Songs der Band oder interessanten Vorkommnissen auf der Bühne entsprechen.

Eine weitere aus dem Graph ablesbare Eigenschaft der zugrundeliegenden Clips ist die Audioqualität. Die Autoren sind sich bewusst, dass viele im Internet zu findende Clips von schlechter Qualität sind und bei Zusammenfassungen möglichst darauf Acht gegeben werden sollte, Clips von guter Qualität auszuwählen. Dies kann für die Audiospuren mittels Fingerprinting erreicht werden, das am besten funktioniert, wenn eine Datenbank mit qualitativ guten und sauberen Quellen vorhanden ist und eine verrauchte Kopie zur Quelle angeglichen werden soll. Das heißt, das System ist sehr robust gegenüber Rauschen in einer Kopie, solange die andere kein Rauschen enthält. Im Fall der Live-Clips ist man jedoch mit der Situation konfrontiert, dass viele Audiospuren von schlechter Qualität sind und daher zwischen ihnen keine Matches gefunden werden. Die Wahrscheinlichkeit für gefundene Matches steigt jedoch, wenn Aufnahmen schlechter Qualität mit Aufnahmen besserer Qualität verglichen werden. Daraus schließen die Autoren, dass jene Clips im Graph mit der größten Anzahl von Kanten wahrscheinlich die beste Audioqualität aufweisen. Bei einer empirischen Studie mit zwei Personen, denen Clips in zufälliger Reihenfolge zur Qualitätsbewertung vorgesprielt wurden, ergab sich eine starke Korrelation zur algorithmischen Bewertung. Daraus folgt, dass die von den Autoren aufgestellte These zur Bewertung der Audioqualität korrekt ist.

6 Fazit


Rückblickend auf die in Kapitel 1.2.2 aufgestellten Anforderungen an die zu erstellende Software kann festgestellt werden, dass die wichtigsten Punkte vollständig erfüllt wurden. Offen bleibt die Unterstützung beliebiger Eingabeformate, die mit einem gewissen Zeitaufwand problemlos integriert werden könnte. Die Pitch-Beibehaltung wurde nicht explizit behandelt, da ein auftretender Time-Drift normalerweise gering genug ist, um die Tonhöhenänderung durch Resampling nicht als störend zu empfinden – was jedoch nicht bedeutet, dass dies auf jeden Fall zutrifft bzw. in jedem Fall tolerierbar ist. Auch der Export zu Dritsoftware bleibt experimentell und zurzeit auf Sony Vegas Pro beschränkt. Ein problematischer Punkt, der die ganze Arbeit hindurch unerwähnt blieb, ist der Time-Drift-Ausgleich bei Videoaufnahmen. Sofern ein Drift vorhanden ist, muss entweder darauf Acht gegeben werden, dass die Videos kurz genug sind, oder dass nur Audiostreams an Videos angeglichen werden. Besteht ein Drift zwischen zwei Videoaufnahmen, musste neben der Audiospur auch die Videospur resampelt werden, was zwar von den meisten NLEs unterstützt wird, jedoch in vielen Fällen zu einer erheblichen Degradierung der Bildqualität führt. Als Alternative könnten in der Videospur einzelne Frames wiederholt oder ausgelassen, oder die Aufnahme in mehrere Abschnitte aufgeteilt und die fehlenden bzw. sich wiederholenden Abschnitte durch Schnitte zu anderen Spuren kaschiert werden. Dies wäre in einer weiterführenden Arbeit zu untersuchen.
7 Literaturverzeichnis


