Automatic Identification of Musical Versions Using Harmonic Pitch Class Profiles

Christoph Engelmayer

DIPLOMARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2011

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 25. September 2011

Christoph Engelmayer

Contents

Kurzfassung

In dieser Arbeit behandle ich Möglichkeiten um automatisch unterschiedliche Versionen eines Musikstücks zu erkennen. Der Begriff „Version“ beschreibt in diesem Fall alle Aufführungen eines musikalischen Werkes das bereits vor dieser Aufführung existiert hat. Die meisten Menschen erkennen auf Anhieb, ob ein Lied eine Version eines anderen Liedes ist. Die selbe Aufgabe stellt einen Computer vor eine weit größere Herausforderung.

Zum Beginn gebe ich einen Überblick über unterschiedliche musikalische Charakteristiken, die sich mit hoher Wahrscheinlichkeit in unterschiedlichen Versionen eines Songs nicht ändern. Weiters werden Wege aufgezeigt um diese Charakteristiken zu extrahieren und miteinander zu vergleichen um so Aussagen über die Ähnlichkeit zweier Werke treffen zu können. Hierzu werden unterschiedliche aktuelle Ansätze für die automatische Erkennung von Versionen vorgestellt.

Basierend auf aktuellen Erkenntnissen habe ich einen eigenen Algorithmus zur Erkennung von musikalischen Versionen programmiert. Ich kombinierte hierfür zwei bereits erprobte Methoden. Zur Beschreibung eines Musikstücks extrahiere ich sogenannte Harmonic Pitch Class Profiles (HPCP) aus dem rohen Audio-Signals. Um die HPCP von zwei Stücken zu vergleichen, um Aussagen über deren Ähnlichkeit zu machen, kommt Kreuzkorrelation und Dynamic–Time–Warping zum Einsatz. Jeden Schritt der zur Erzeugung der Harmonie Pitch Class Profiles notwenig ist wird in der Arbeit ausführlich beschrieben.

Das vorgestellte System zur musikalischen Versionserkennung wird weiters mit einem anderen Algorithmus auf zwei unterschiedlichen Datensätzen getestet. Der zweite Algorithmus unterscheidet sich vom vorgestellten dadurch, dass er anstatt HPCP sogenannte Mel Frequency Cepstral Coefficients (MFCC) zur Beschreibung des Audio-Signals verwendet. Die Testergebnisse zeigen, dass der im Zuge dieser Arbeit entwickelte Ansatz zwar funktioniert, jedoch noch nicht an die Genauigkeit des zweiten Algorithmus herankommt.

Im weiteren Verlauf werden mögliche Verbesserungen des entwickelten Systems diskutiert und der Einfluss verschiedener Faktoren des Algorithmus behandelt. Weiters werden relevante Probleme, die während der Entwicklung des Systems auftraten, aufgezeigt. Mögliche zukünftige Arbeit auf dem Gebiet der musikalischen Versionserkennung wird im Bereich der näheren

Erforschung der Faktoren zur Erstellung von HPCP gesehen. Hier könnte beispielsweise versucht werden, Einstellungen für spezielle Anwendungsfälle wie z.B. die Erkennung eines bestimmten Genres zu finden.

Abstract

In this thesis I write about ways to compare audio in order to automatically find different versions of a song. A "version" is every new performance of a previously released piece of music. Most people can recognize if a given song is a version of another one. The same task performed by a computer although is a much more advanced problem involving many different aspects of music similarity.

First, an overview of different musical features that are most likely to stay the same across versions is presented. Also ways how to extract these features and compare them to estimate the similarity of two songs. Different state of the art approaches for music version detection are discussed.

Based on current algorithms, I implemented my own music version detection system. Therefore I extract beat–aligned harmonic pitch class profile features (HPCPs) out of a songs raw audio signal. To compare the features I used cross-correlation and dynamic time warping. All the processing steps that are necessary to generate HPCP features are described in detail.

The presented version detection system is compared to another approach using Mel Frequency Cepstral Coefficients (MFCC). The results show that the presented approach is basically working but its performance is not better than the one of the other algorithm.

Possible further improvements to the algorithm are discussed and the influence of different settings are presented. Also relevant problems that appeared while implementing the algorithm are mentioned. Further work directions are seen in exploring the settings involved in creating HPCP vectors to see if there are any ways to tune the features to work better for a specific genre.

Chapter 1

Introduction

The way how people consume music has changed dramatically in the last 15 years. After the invention of the MPEG-1 Audio Layer 3 (MP3) codec for audio compression by the Fraunhofer Institute, a music collection that years ago physically filled a whole shelf with CDs now fits into a pocket.

But not only our personal have libraries changed. Nowadays nearly every piece of music ever produced is available on the internet. Online music stores like the Apple iTunes store[1] or the Amazon music store[2] sell more songs online than their real world competitors. Also platforms like YouTube[3] offer millions of music videos from professional to amateur artists.

The most common way to organize these libraries is to assign metadata to each item. Text based queries are used to search for artist, title or album information. This gives the user the possibility to localize a desired piece of music but it requires him or her to know particular information abut the song he or she is looking for.

To allow the user to explore new music within the database, genre classification is a common used approach. Every song within a library is assigned to a musical-style like e.g. rock, pop or classic. Users preferring music from artists of the rock genre can look for songs tagged as rock and will find music they may like. This of course strongly depends on the previous (manual) organization of the library. Not every song can be assigned to one specific genre.

With growing personal and commercial music libraries, the urge to find better ways to handle this amount of data has raised. The scientific community for Music Information Retrieval (MIR) started research on how to "automatically understand, describe, retrieve and organize musical contents"[4].

In this thesis I am writing about methods for music similarity estimation,

[1] http://www.apple.com/itunes

[2] http://www.amazon.com

[3] http://www.youtube.com

[4] A detailed list of MIR topics can be found at http://ismir2011.ismir.net/callforpapers.html.

in specific I am looking at solutions for *music version identification*.

1.1 Music similarity estimation

Music similarity estimation in general explores ways to automatically recognize and describe similarity between musical pieces. This is interesting for applications that try to simplify the way users search for music, navigate through music libraries and explore new (unknown) music that might fit their taste.

Two songs can be identified as similar if they share one or more musical facets. But music similarity may also depend on different cultural (or contextual) and personal (or subjective) aspects.

According to [5], music similarity tasks could be specified by

- their query type
- extracted descriptors
- how these descriptors are compared
- the form of the output.

Possible query types are either content-based music descriptors (information extracted from the raw-audio data e.g. wav or mp3 files) or a symbolic representation of the audio-content (e.g. MIDI files). The used descriptors (e.g. Timbre, Tempo, Key, ...) and the method of their comparison depends on the use-case. They can be categorized by their degree of specificity. Either *exact*, retrieving perfect matches (e.g. audio-fingerprinting) or *approximate*, retrieving matches that are similar but not exactly the same.

Some common music-similarity tasks are:

Music recommendation Traditional ways to discover new music like radio-broadcasts or record-stores are replaced by more personal ways. Version detection algorithms can be used to recommend new music to users according to their musical taste. Online music stores already use recommendation services based on the songs a user bought.

Content-based music retrieval The user specifies a query by choosing one or more songs he or she likes (instead of e.g. a genre) and the system automatically returns a list of similar songs which the user may also enjoy.

Genre-classification The MIR community started to research on features for audio similarity with the effort to build an audio-based genre classification system. As described above, automatic-genre classification tries to automatically assign a genre to a song in a library.

Audio fingerprinting In [53, 54] the authors present a system for audio fingerprinting, a common topic of the MIR community. Their algorithm is used in the very popular *"Shazaam"* application for smartphones. The software gives its users the possibility, to record a sample of a song in

a real world environment (e.g. club or bar) using their phones. The query is sent to a server which tries to find a match in the database and return information about the artist and title of the recognized song back to the user. Their system shows very high recognition rates and robustness against ambient noise.

A more complex problem the MIR community is exploring (see e.g. [7]) is to compare songs in order to find music that sounds similar but is not exactly the same. This could be for example a live performance or cover-version of a previously recorded song. In contrast to other music-similarity tasks, version-detection is a more complex task. It tries to detect matches like audio-fingerprinting while allowing many musical aspects to change. It is more specific than genre-classification as it is based on the idea that different versions of one musical piece keep their identity even if many musical dimensions vary [13].

A song contains many different sources of information (e.g. instruments) and every source has its own structure (i.e. played notes). All this information is combined into the audio-signal. For humans it is very easy to extract and process this information (e.g. melody, rhythm, ...). Most listeners can recognize if two songs are different versions of the same song. However, the same task performed by a machine is a much more advanced problem.

Looking at fields of application for version detection algorithms, they could be used in a variety of ways:

Search: Nowadays, the search in large music databases is based on keywords and tags like artist, title or genre most of the time. This information has to be evaluated by humans and manually assigned to each musical piece in the database. Version detection algorithms offer the possibility to search using semantic descriptors extracted directly from the audio content.

Music history: Exploring the history/evolution of a particular song version. Find other songs based on the same musical piece.

Musical rights management: Identification of copyright protected material on platforms where users can upload content (e.g. youtube.com) to avoid licensing problems.

In this thesis, I provide an overview of current state of the art approaches for version detection. The scope is focused on content-based methods, so no work that uses any kind of input data other than the audio signal is referred. Algorithms that use metadata supplied by humans, assigned by other sources or any high-level representations (e.g. MIDI) are excluded.

In chapter 2 the general approach is described and an overview on current state of the art approaches is given. Different musical features predestinated for music version detection are presented and discussed. Common ways to extract them out of raw audio data and compare them according to their similarity are introduced.

In chapter 3 I will present my own implementation of a version detection algorithm, based on approaches presented in chapter 2. A detailed step by step guide on how to extract meaningful features and use them for similarity estimation is given. My approach was implemented and tested using MATLAB. The results are presented and discussed in chapter 4. In chapter 5 possible future directions are pointed out.

Chapter 2

Music version detection – state of the art

Before going into detail about musical version detection I first want to specify what exactly is meant when talking about a *"version"* of a song.

2.1 The definition of "version"

The term *"version"* describes a new performance or recording of a previously composed musical piece. Previous published work (like e.g. [13,16,39,46,49]) also often uses the term *"cover-song"* instead of version. In modern western music a tradition of one artist playing a song from another artist as a homage is called cover song. I see them as a specific version category so I prefer talking about versions and not cover-songs in this thesis.

[46] divides versions into the following categories. Every category depends on the way how a version is created.

Remaster Creating a new master for an album or song generally implies some sort of sound enhancement to a previously existing product (e.g. compression, equalization, different endings or fade-outs).

Instrumental Sometimes, versions without any sung lyrics are released. These might include karaoke versions to sing or play along with, alternative versions for different record-buying public segments (e.g. classical versions of pop songs, children versions, etc.) or rare instrumental takes of a song in CD-box editions specially made for collectors.

MashUp This is a song or composition created by blending two or more pre-recorded songs, usually by overlapping the vocal track of one song seamlessly on the instrumental track of another.

Live performance A recorded track from live performances. This can correspond to a live recording of the original artist who previously released the song in a studio album or to other performers.

Acoustic The piece is recorded with a different set of acoustical instruments in a more intimate situation. Sometimes "unplugged" is used as a synonym.

Demo It is a way for musicians to approximate their ideas on tape or disc, and to provide an example of those ideas to record labels, producers or other artists. Musicians often use demos as quick sketches to share with band mates or arrangers. In other cases, a music publisher may need a simplified recording for publishing or copyright purposes, or a songwriter might make a demo in order to be sent to artists in the hope of having the song professionally recorded.

Standard In jazz music, there are compositions that are widely known, per- formed and recorded. Musicians usually maintain the main melodic and/or harmonic structure but adapt other musical characteristics to their convenience. There is no definitive list of jazz standards though this might change over time.

Medley Mostly in live recordings, and in the hope of catching listeners attention, a band performs a set of songs without stopping between them and linking several themes. Usually just the more memorable parts of the music work are included.

Remix This word can be very ambiguous. From a "traditionalist" perspective, a remix implies an alternate master of a song, adding or subtracting elements or simply changing the equalization, dynamics, pitch, tempo, playing time or almost any other aspect of the various musical components. But some remixes involve substantial changes to the arrangement of a recorded work and barely resemble the original one. A remix may also refer to a re-interpretation of a given work such as a hybridizing process simultaneously combining fragments of two or more pieces of work.

Quotation The incorporation of a relatively brief segment of existing music in another work, in a manner akin to quotation in speech or literature. Quotation usually means melodic quotation, although the whole musical texture may be incorporated. The borrowed material is presented exactly or very similar so, but is not part of the main substance of the work. Incorporating samples of other songs into one own

song would fall into this category.

2.2 Musical aspects

When the human brain processes music it extracts certain musical information out of the audio and compares it to music we have heard before. It is unknown what exactly the essential information is, that has to be encoded by the listener to identify a song as a version from another one, but it seems that versions usually use the same melodies or tonal progression but differ in one or more other aspect like e.g. tempo, key or timbre. (Talking about western pop-music. Versions in classical music may only vary very slightly.) To build a music version identification system we use the Musical Cognition approach. This means that we try to understand how humans "decode" music to rebuild a system similar to that.

The authors of [46] differentiate the following musical aspects:

Timbre: Many variations changing the general color or texture of sounds might be included in this category. Two predominant groups are:

Production techniques: Different sound recording and processing techniques introduce texture variations in the final audio rendition (e.g. equalization, microphones or dynamic compression).

Instrumentation: the fact that the new performers could be using different instruments, configurations or recording procedures can confer different timbres to the version.

Tempo: As it is not as common to strictly control the tempo in a concert, this characteristic can change or fluctuate even in a live performance of a given song by its original artist. In fact, strictly following a predefined beat or tempo might become detrimental for expressiveness and con- textual feedback. Even in classical music, small tempo fluctuations are introduced for different renditions of the same piece. In general, tempo changes abound, sometimes on purpose, with different performers.

Timing: In addition to tempo, the rhythmical structure of the piece might change depending on the performer intention or feeling. Not only by means of changes in the drum section, but also including more subtle expressive deviations by means of swing, syncopation, accelerandos, ritardandos or pauses.

Structure: It is quite common to change the structure of the song. This modification can be as simple as skipping a short introduction, repeating the chorus where there was no such repetition, introducing an instrumental section or shortening one. On the other hand, such modifications can be very elaborated, usually implying a radical change in the musical section ordering.

Key: The piece can be transposed to a different key or main tonality. This is usually done to adapt the pitch range to a different singer or instrument, for aesthetic reasons or to induce some mood changes in the listener. Transposition is usually applied to the whole song, although it can be restricted just to a single musical section.

Harmonization: Independently of the main key, the chord progression might change (e.g. adding or deleting chords, substituting them by relatives, modifying the chord types or adding tensions). The main melody might also change some note durations or pitches. Such changes are very common in introduction and bridge passages. Moreover, in instrumental solo parts, the lead instrument voice is practically always different from the original one.

Lyrics and language: One purpose for recording a version is to translate it to other languages. This is commonly done by high-selling artists to become better known in large speaker communities.

Noise: In this category we consider other audio manifestations that might be present in a recording. Examples include audience manifestations such as claps, shouts or whistles, speech and audio compression and encoding artifacts.

To identify two songs as versions of the same song, at least one of these characteristics has to be similar in both of them.

2.3 Version detection – general approach

To build a system that automatically detects versions we have to extract one or more meaningful features from the audio signal and compare it to identify similarities. Not all musical aspects are equally important for version detection as not all of them stay the same throughout versions. Which aspects vary and which of them remain similar strongly depends on the way the version is made. Table 2.1 lists several common features that are most likely to change between two versions.

To build an algorithm to identify different versions, it is important for this

	Timbre	*Tempo*	*Timing*	*Structure*	*Key*	*Harmonics*	*Lyrics*	*Noise*
Remaster	×							
Instrumental	×						×	×
Mashup	×			×			×	×
Live	×	×	×					×
Acoustic	×	×	×		×	×		×
Demo	×	×	×	×	×	×	×	×
Standard	×	×	×	×	×	×	×	×
Medley	×	×	×	×	×			×
Remix	×	×	×	×	×	×	×	×
Quotation	×			×				×

Table 2.1: Possible changes of musical features according to different version categories. Possible change is marked with ×. (Table adapted from [45])

to happen without being influenceable by tempo, key or structure. Instead we try to extract features that are most likely to stay similar in different versions. The accuracy of a music version detection algorithm strongly depends on the chosen features. Versions of a song usually preserve the main melodic and/or harmonic progression. This is why *tonal* and *harmonic* content are the most employed characteristics used for music version detection.

2.4 Approaches for feature extraction

To develop a music version detection algorithm, it is important to extract features that are most likely to stay the same across versions.

As described in [20], a feature that is largely preserved across different versions and is robust against changes in other musical facets is the *tonal sequence*. Nearly all current music version detection algorithms use some kind of tonality feature to describe polyphonic music. Exceptions are older systems that use other descriptors like the energy of the audio signal or spectral-based timbre [17, 56].

Tonality describes a system of hierarchical relationships between a series of pitches [41]. The hierarchy is based on the most stable element i.e. pitch in the system which is called *central pitch class*, *tonic* or *key*. Generally speaking, a tonal sequence is a sequentially-played series of different pitch combinations. If the played combination is unique for each time slot it is referred as *melody*. If more than one note are played together for each time

slot we speak of *chords* or *harmonic progression*.

Tonality is omnipresent in western music and easy to extract for human listeners. Tests in [47] have shown that most people can identify the most stable pitch while listening to tonal music [9], no matter if they are musically trained or not.

All these characteristics make tonal sequences an important feature for audio similarity tasks, especially for music version detection [1,6,26]. Similar information is also a very common used feature for speech recognition tasks like the one presented in [36].

Basically there are three different features modern version detection algorithms try to extract to describe tonal sequences. They either try to extract

- a predominant melody,
- a chord/harmonic progression,
- or a chroma sequence.

2.4.1 Predominant melody

In music, the term melody describes a sequence of tones. This could be e.g. the chorus of a song where the main melody is present and repeated. The melody of a song is one of the most recognizable characteristics of a song for human listeners which makes it a great descriptor of a musical piece [44]. This is why it is commonly used in music version identification systems e.g. [33, 34, 42, 51, 52].

To explore ways for extracting predominant melody out of raw audio data a lot of work e.g. [23, 28, 38] has been done in the MIR community.

Melody extraction is based on methods for pitch perception and fundamental frequency estimation. As the MIR community already focused on these tasks for several years, some advanced methods like [11, 12] already exist. However, melody extraction is still a difficult task. Real-word audio signals are a complex mixture of different information which leads to many problems with the perception and tracking of useful features. One problem is to identify the frequency belonging to the melody in case that multiple fundamental frequencies appear at the same time.

In the following sections I briefly describe different methods to handle these problems in order to generate more accurate melody representations.

Voice detection

One method to achieve a more reliable representation of the melody is to combine the its extraction with a voice/non-voice detector. The system presented in [51, 52] removes non-vocal parts that are longer than a certain timespan. This timespan depends on the tempo i.e. the beats per minute (BPM) of the song (two seconds for 120 BPM). The removed parts are expected to belong to the intro, bridge or outro of the song. Remaining parts

are therefore assumed to be part of the verse (typically containing the main theme) or chorus (most recognizable melody). These remaining parts are converted into musical note symbols. Sequences of these notes can later be compared to estimate similarities between songs.

Another melody estimation system that uses voice detection and some other post-processing modules is presented in [42]. The authors combine voice detection with a multi resolution Fourier transform and use a distance matrix to compare extracted descriptors.

First they use the algorithm presented in [14] to extract a pitch line for the parts of the song where a voice is present. From this pitch line and further spectral information, so called *note candidates*, are extracted. Note candidates with a duration and/or loudness under a certain threshold are discarded. All the remaining candidates are assigned to discrete pitches.

Another method is used to store the parts of the musical sequence that contain relevant melody information. In [50], a pitch sequence is considered as important if it is between 3 and 8 seconds long and consists of neither too few nor too many notes. In a final step, the algorithm looks for repeating sequences and weights them according to how often they appear in the song.

Mid level descriptors

In MIR we differentiate between different levels of abstraction for descriptors. They can be categorized in *High*, *Mid* and *Low* according to their level of abstraction. Low level descriptors are very closely related to the audio signal. Therefore they are hard to understand and describe for humans. High level descriptors are modeled according to knowledge from the fields of music cognition and music psychology. Therefore, they are a lot more meaningful for humans.

Another method for extracting and describing the predominant melody in a song is to use Mid level descriptors. These melody features reduce the semantic gap and describe audio in a way that simplifies retrieval. The authors of [33,34] combine melodic, rhythmic and structural aspects to generate Mid level features. They generate a melodic representation of the audio signal and look for salient melodic lines in it. To handle tempo variations, the representation is beat synchronous (see chapter 4). A search algorithm based on locality-sensitive hashing is used to perform retrieval according to similarity of the extracted melodic fragments. This guarantees fast performance for similarity search in large music databases.

2.4.2 Harmonic or chord progression

Instead of using the melody as the main descriptor, other version detection systems use representations that emphasize harmonic or chord progression. A chord is a set of notes played simultaneously. In western music chords most

frequently appear as triads e.g. three notes played at the same time.

Two common ways to extract this information are chromagram (e.g. [16]) and pitch class profiles (PCP) (e.g. [18]). According to [49], these features might provide a more complete, reliable and straightforward representation compared to the melody descriptors mentioned before.

Chromagram features

A chromagram is a special variation of a spectrogram, using a projection of the spectrum onto 12 bins where each bin represents one of the 12 tones of the musical octave. Therefore, chromagram features represent melodic and harmonic information of the song.

A similar representation is provided by *Mel-frequency Cepstral Coefficients* (MFCCs). They transform the spectrum in order to describe the sound characteristics as they are perceived by a human listener. MFCCs are computed by applying a mel-spaced set of filters to the spectrum. A mel measures the perceived pitch of a tone for the human ear.

The authors of [29] use a Chromagram representation of the music for chord recognition. Chroma based features could be used for a wide variety of MIR tasks like pattern discovery [10], audio thumbnailing and chorus detection [2, 24], or audio alignment [26, 35].

In [16] the authors use chromagram features to identify different versions of a song. They use a beat tracking algorithm in order to create a 12-dimensional chroma vector for every beat.

Every dimension of the vector represents one semitone on the musical octave. To handle pieces that are slightly out of tune a margin of ± 0.5 semitones is used for the construction of the chroma features.

Pitch class profiles

Another representation for harmonic content of audio are *pitch class profiles* [18, 20, 25, 32, 45]. These features where first introduced in [18] to perform chord recognition tasks. PCPs are vectors containing chroma features describing the power of each semitone of the octave for a given time frame in the audio signal [20].

To create PCPs, the audio is windowed (usually into 100ms frames) and every slice is transformed to the frequency domain by a discrete Fourier transformation. Energy-peaks within a given frequency range (usually 50HZ to 5kHz) are located and assigned to a pitch class band according to their frequency. Peaks on lower or higher octaves are folded into one resulting in a 12-dimensional PCP vector.

The advantages of PCP descriptors are that they combine chroma information and harmonic structure in the spectrum while keeping the amount of dimensions manageable. This is one fact why they are very popular for

MIR tasks.

2.5 Common pre- and post-processing steps

In nearly all systems mentioned in the previous sections some pre- and post-processing is done in order to prepare the raw data for feature extraction. Another goal of pre-processing is also to reduce the amount of data without loosing too much meaningful information.

The following are common pre- or post-processing methods to handle problems that appear while extracting features for music version detection. See table 2.2 for a detailed overview of algorithms and methods.

2.5.1 Reference frequency estimation

Theoretically, music instruments in western music are tuned to 440 Hz. In reality no instrument is perfectly tuned. Because of that, algorithms that use some kind of pitch class distribution to estimate a reference frequency the musical piece is tuned in order to assure robustness to tuning issues.

2.5.2 Key invariance

A musical characteristic that frequently changes between different versions of a song is the *key*. This is why nearly all algorithms try to handle key transposition between songs. The two most common ways to deal with this problem are either to *estimate the main key* or *shift the key sequence* to find the transposition that fits best.

Estimation of main key

This method estimates the main key for every song and transposes it accordingly. This is done by finding the most dominant key in a songs feature representation and set it to zero. The dominant key can be estimated by looking at the total of every pitch class throughout a whole song. A big drawback of this method is that its success strongly depends on the accuracy of the algorithm used for key estimation. In the worst case, the estimated key of the query song is wrong so no version will fit.

Shift key

Another method to handle key invariance between songs is to shift the key to all possible transpositions in order to find the one that fits best. The benefit of this method is that it will find the best fitting key transposition of the given feature representation. On the other hand the computation is intense because similarity measures for all possible transitions have to be done.

2.5.3 Tempo invariance

Another aspect that is most likely to change between different versions of a song and therefore interesting to handle is the tempo. There are different ways to deal with this problem.

Mean values (per beat)

As described in [33] the events in a musical piece do not appear in a direct relation to time, but in a relation within a metric hierarchy. The basic elements of this hierarchy are *beats*.

Systems like [16, 33, 37] try to estimate the tempo for every song and extract mean values accordingly. Most of the time, beat detection algorithms are used to estimate the beats per minute (BPM) and beat timestamps. All values between two beats are averaged to get a mean value per beat representation of the song. These representations are tempo invariant and can be compared with each other.

The results for this kind of tempo handling strongly depend on the accuracy of the beat estimation.

Temporal compression

Temporal compression resamples the extracted descriptor sequence of a song to several plausible expanded and compressed versions. These are compared in order to find the best matching sample rate.

In [39] the features of every song are resampled using three different tempo means (240, 120 and 60 BPM) leading to nine different scores for each song pair. All scores are normalized and placed in a nine-dimensional feature vector which is classified as cover or non-cover.

Similarity matrix

Another method to handle tempo invariance between song versions is to create a similarity matrix.

The authors of [49] create a binary similarity matrix where each value expresses if two extracted features are similar or not. They are considered as similar if their similarity is under a certain threshold value.

The similarity matrix representation is used as source for dynamic programming similarity estimation methods as described in chapter 2.6.3

2.5.4 Invariance in the structure

[22, 33] use song summarization or chorus extraction techniques to improve the results of their algorithms. Therefore they try to find meaningful parts of the song (e.g. repetitive parts like the chorus) and use them as input

for further similarity *estimation*. Methods to identify and extract meanigful sections of a song are presented in [8, 10].

2.6 Similarity estimation

The last step of each version detection algorithm is to compare the extracted feature sequences of two or more songs in order to estimate their similarity.

Basically there are two tasks for similarity estimation. A *query-by-example* task where users submit a song and the system returns a ranked list of similar songs out of a database. Another possibility is a *true/false* task with a query of two songs and a true or false response depending on if the query songs are versions of each other or not.

2.6.1 Euclidean distance

In [5] the authors use so called *shingles* to describe a musical piece. Shingles are sequences of audio features combined into a single high-dimensional vector.

To measure the similarity of two songs the authors place the shingles of both songs in a multidimensional space and pairwise compare their *Euclidean distance* in this space. The smaller the distance between shingle pairs, the more similar the two shingles are. The authors are looking for the amount of shingle pairs with a distance less than a defined threshold. The more shingles match, the higher the similarity of the songs is.

The success rate of the whole algorithm depends on this distance threshold value. The authors estimate the distribution of shingle distances between two tracks that are known to be unrelated.

2.6.2 Cross-correlation

In [16] the authors compare two songs by cross-correlating the beat-chroma matrices (MFCC features) of both songs as if they where images. The row in which the magnitudes correlate best are high pass filtered. Furthermore, a value for every beat is computed by calculating the reciprocal of the output maximum value. These values represent the degree of similarity for every beat of the two songs. Parts of the songs where beats with the same tonal structure occur create peaks in the 2D correlation.

2.6.3 Dynamic programming

Dynamic programming is a technique for aligning two data-sequences which may vary in time or speed and automatically discover their local correspondences. It is a popular technique (used in [3, 15, 17, 21, 22, 27, 31, 33, 37, 48, 49, 51, 52, 56] for handling tempo invariance and calculate similarity in version

detection algorithms. According to results presented in [3, 49], most systems that use dynamic programming perform better than those using mean-beat features.

To use dynamic programming for similarity estimation, a similarity matrix (as described in chapter 2.5.3) is needed as a source. The most common used dynamic programming techniques are *Dynamic time warping* and *Edit-distance variants*. These are currently two of the most accurate ways to handle tempo invariances but also computationally expensive (quadratic in the songs representation).

Dynamic time warping

Dynamic time warping (DTW) uses a path finding algorithm on the distance matrix to find an optimal alignment path.

To use DTW some pre-processing steps are necessary. Each HPCP vector is normalized by its maximum value and a key-detection algorithm is applied to estimate the key of the compared tracks. In a second step both songs are transposed to the same key. Now a cumulative distance matrix describing the similarity between the HPCPs of both songs is generated. Every cell represents the similarity between the two HPCPs (e.g. local alignment). Using this matrix it is possible to calculate an alignment path. The length of this path is representing the similarity between the two HPCP sequences (for a detailed description see [49]). The smaller the alignment costs of two songs, the more similar they are. A song compared to itself would be a perfect match with normalized alignment costs of 1 (i.e. a diagonal line in the distance matrix). DTW is a very common used technique in speech recognition and processing.

Edit-distance variants

The *edit-distance* of two character sequences is the number of operations required to transform one sequence of characters into the other. Originally edit-distance algorithms where developed to perform DNA alignment in molecular biology.

One very popular edit-distance algorithm is [55], the so called *Smith–Waterman* algorithm. It compares feature segments of all possible lengths and optimizes the similarity measure. The algorithm is run on the similarity matrix from chapter 2.5.3. The highest value of the returned dynamic program is used as a feature to describe the similarity between songs (for details see [49]).

Reference(s)	*Extracted features*	*Key invariance*	*Tempo invariance*	*Structure invariance*	*Similarity computation*
Foote (2000)	Energy + Spectral		DP		DTW
Yang (2001)	Spectral		DP	Linearity filtering	Match length
Nagano et a. (2002)	PBFV	All transp.	Beat + DP	Seq. windowing + DP	Match length
Izmirli (2005)	Key templates		DP		DTW
Müller et al. (2005)	PCP		Temporal comp./exp.	Sequence windowing	Dot product
Tsai et al. (2005, 2008)	Melodic	K trans.	DP		DTW
Gomez & Herrera (2006)	PCP	Key estim.	DP		DTW
Gomez et al. (2006)	PCP	Key estim.	DP	Repeated patterns	DTW
Lee (2006)	Chords	Key estim.	DP		DTW
Marlot (2006)	Melodic	Key estim.	DP	Repeated patterns	Cross-correlation
Sailer & Dressler (2006)	Melodic		Relative		Edit-distance
Bello (2007)	Chords	K transp.	DP		Edit-distance
Ellis & Cotton (2007); Ellis & Poliner (2007)	PCP	All transp.	FBeat		Cross-correlation
Kim & Perelstein (2007)	PCP	Relative	HMM		MLSS
Ahonen & Perelstein (2007)	PCP	Relative			NCD
Egorov & Linetsky (2008)	PCP	OTI	DP	DP	Match length
Jensen et al. (2008)	PCP	All transp.	Fourier transform		Frobenius norm
Jensen et al. (2008)	PCP	2D autocorrelation	2D autocorrelation		Euclidean distance
Kim & Narayanan (2008); Kim et al. (2008)	PCP + Delta PCP	All transp			Dot product
Kurt & Muller (2008)	PCP	All transp.	Temporal comp./exp.	Sequence windowing	Dot product
Marlot (2008)	Melodic	2D spectrum	Beat + 2D spectrum	Sequence windowing	Euclidean distance
Serra et al. (2008, 2009)	PCP	OTIs	DP	DP	Match length
Ahonen (2010)	Chords + Other	OTI			NCD
Serra et al. (2010, 2011)	PCP	OTIs			Prediciton error
Di Buccio et al. (2010)	PCP	K transp.		Sequence windowing	Set intersection

Table 2.2: *Version identification algorithms developed in the last 10 years* Version identification algorithms and their ways of overcoming changing musical characteristics. PBFV... polyphonic binary feature vector, PCP... pitch class profile, OTI... optimal transposition index, DP... dynamic programming, HMM... Hidden Markov Models, DTW... dynamic time warping, MLSS... most likely sequence of states, NCD... normalized compression distance. (Table adapted from [45])

Chapter 3

The Harmonic Pitch Class Profile

After working through all the state of art techniques I decided to build my own implementation of a music version detection algorithm by combining different parts of the systems presented in chapter 2. I built an algorithm based on HPCP features because they have the highest success rate according to MIREX cover song detection task[1] of the recent years. During research numerous questions appeared about how to generate meaningful HPCP features. In [20, 45, 49] some details are unspecified or poorly described so I spent a lot of time gathering the pieces that are missing. I created my own implementation using MATLAB.

In this section all the necessary steps to extract HPCP features are described in detail. Moreover, two methods for comparing them where implemented and described.

The basic steps for the extraction of HPCP features are:

1. Pre-processing
   (a) Beat detection
   (b) Transient detection and handling
   (c) Spectral analysis
      i. Windowing
      ii. Zero-Padding
      iii. Discrete Fourier Transformation
   (d) Spectral whitening
   (e) Peak detection
   (f) Frequency filtering

[1]The Music Information Retrieval Evaluation eXchange (MIREX) is an annual evaluation campaign for music information retrieval (MIR) algorithms. http://www.music-ir.org/mirex/

   (g) Reference frequency estimation
2. Harmonic Pitch Class Profile computation
   (a) Weighting
   (b) Considering harmonic frequencies
3. Post-processing
   (a) Normalization
   (b) Transposition

3.1 Pre-processing

The following pre-processing steps prepare the audio-signal for the extraction of the harmonic pitch class profile.

3.1.1 Beat detection

The beat detection is necessary if a per-beat HPCP representation is needed for the comparison method.

In the first step I calculate the time location of every beat in the query audio signal. Therefore, I use a pre-built beat-detection function[2] from [16] which delivers a vector containing the detected beat locations of the signal in milliseconds. This information is later used in chapter 3.4.1 for the computation of mean HPCPs for every beat. It is important to extract the beat locations out of the raw audio signal before it may be influenced by other pre-processing steps.

3.1.2 Transient detection and handling

In [20] the author suggest to handle transients in the audio signal to make the algorithm more robust against noise.

To detect transient positions in the input signal I use an approach similar to the one described in [4]. The basic idea is to compare the change of *Mel-frequency cepstral coefficients* (MFCCs) over time. Therefore, a MEL cepstrum (see figure 3.1a) with 42 bands between 40 and 20353.6 Hz using a Hamming window of 23.2 ms length is created (values are recommended in [4]).

Every MFCC is now compared to its sequential MFCC by calculating the absolute difference between each bin as shown in figure 3.1b. This information is stored in a matrix which is used to locate the transient locations in the signal. Hence, I calculate a graph by using only the maximum values of each column in the difference matrix. Peaks in this graph mark high differences between two sequential MFCCs i.e. transient positions (figure 3.1c).

[2] MATLAB code is available at http://labrosa.ee.columbia.edu/projects/coversongs

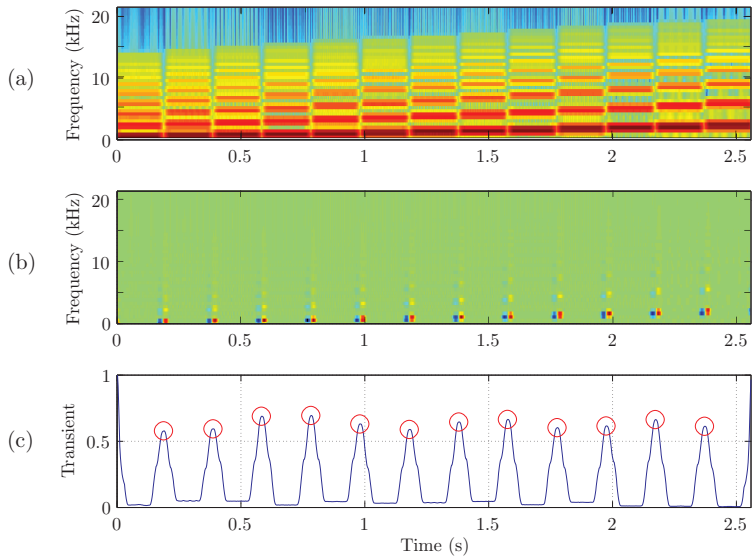


Figure 3.1: Transient detection steps for a generated sample signal playing a continous tone that changes pitch every 0.2 s. (a) MEL cepstrum, (b) difference matrix, (c) smoothed signal generated from maximum difference matrix with marked peaks.

To detect these peaks I first smoothed the data using a moving average filter followed by a peak-detection function from the MIRtoolbox[3]. The time values of the peak points are the transient locations in the query signal. Using the gathered transient time information I am now able to smooth the input signal with an inverse tukey window of 100 ms size at each detected peak time. Therefore 50 ms before and after each transient location are not analyzed. According to [49] this step is necessary to lower the influence of transients in the signal.

Other work like [45] does not use transient handling at all. I implemented the feature and tested it with the result that in most of the cases, transient detection does more harm than good on the accuracy of the HPCP features (See detailed test results in chapter 4). Consequently I do not recommend using a transient detection and handling for HPCP creation.

3.1.3 Spectral analysis

For the spectral analysis, the signal has to be converted into the frequency domain. This is done by windowing followed by a discrete Fourier transform.

[3]MIRtoolbox is a collection of functions for musical feature extraction in MATLAB [30]

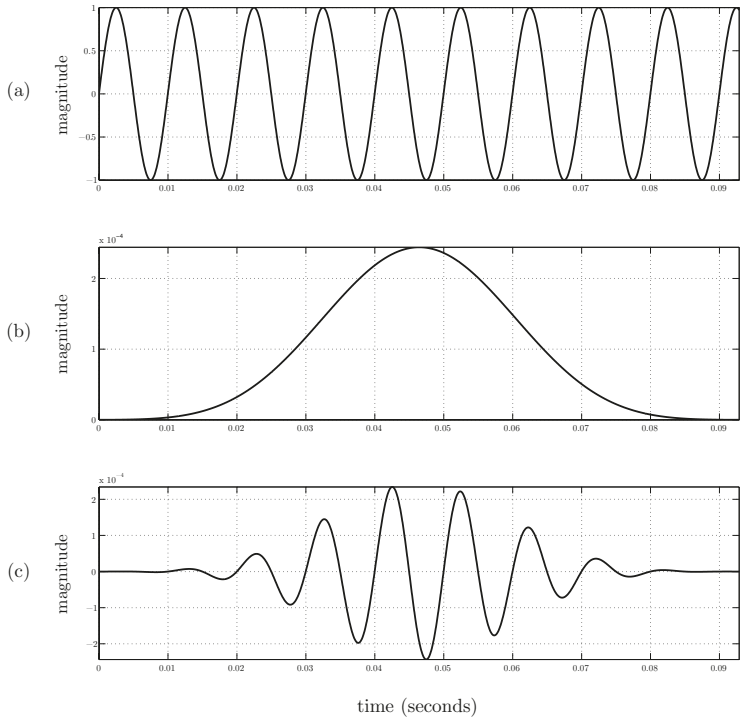


Figure 3.2: *Windowing of the signal* (a) signal (b) windowing function (c) windowed signal.

Windowing

The sampled audio signal $x(n)$ is split into frames of length N_{frame}. Every frame is defined as $x(n+l \cdot N_{hop})$. N_{hop} is the hop-size defining the overlap of the frames. In addition, a windowing function $w(n)$ is applied to each frame (see figure 3.2). The final windowed signal $x_w(n)$ is

$$x_w(n) = x(n + l \cdot N_{hop}) \cdot w(n). \tag{3.1}$$

In my implementation I use a frame length of $N_{frame} = 4096$ samples which equals 93 ms assuming an audio signal with a sample rate of 44.1 kHz. The overlap of the frames is set to 75% so $N_{hop} = 1025$ samples. (Notice that all values are specified in frames and therefore dependent on the sample rate of the audio signal). All these values are recommended in [20, 45] and be proven to generate reliable results. [20] uses a slightly smaller hop size of 512 samples. [18] defines a larger frame size of 400 ms. All values used in my implementation are listed in table 4.1.

The windowing function $w(n)$ is needed to create a periodic signal which can be used as input for the discrete Fourier transform. Window functions are defined by the width of their main lobe and their highest side lobe level.

According to [20] common windows are the rectangular window (main-lobe width of 2 bins and side-lobe level equal to 13 dB), the Hanning window (main-lobe width equal to 4 bins and side-lobe level equal to 23 dB), the Hamming window (main-lobe width of 4 bins and side-lobe level equal to 43 dB), the Blackman-Harris window or the Kaiser window. For my implementation I followed the recommendations in [45] and used a Blackman Harris window with a side lobe level of 92 dB.

A L-term Blackman-Harris window is generated using

$$w(n) = \frac{1}{N_{frame}} \sum_{l=0}^{L-1} \alpha_l \cdot cos(\frac{2nl\pi}{N_{frame}}), n = 0, 1, \dots N_{frame} - 1. \tag{3.2}$$

The side lobe level depends on the α parameters. To create a side lobe level of $92dB$ $\alpha_0 = 0.35875$, $\alpha_1 = 0.48829$, $\alpha_2 = 0.14128$ and $\alpha_3 = 0.01168$.

As a last step of the windowing process, the windowed data gets centered in the time origin (i.e. Zero-phase window) using

$$w_{wc}(n) = x_w(n + \frac{N_{frame}}{2}), n = -\frac{N_{frame}}{2}, \dots, \frac{N_{frame}}{2} - 1. \tag{3.3}$$

Discrete Fourier transform

In the next step the power spectrum $X(k)$ (see figure 3.3) of the windowed and zero-phased signal is created using the *discrete Fourier transform* (DFT). It is defined by

$$X(k) = DFT[x(n)] = \sum_{n=-\frac{Nframe}{2}}^{\frac{N_{frame}}{2}-1} x_{wc}(n) \cdot e^{\frac{-j2\pi nk}{N_{frame}}}, \tag{3.4}$$

with $k = 0, 1, \dots N_{frame} - 1$. The result of a DFT consists of a real X_r and an imaginary X_i part. The magnitude $|X(k)|$ and the phase $\phi(k)$ can be calculated with

$$|X(k)| = \sqrt{X_r(k)^2 + X_i(k)^2}, \tag{3.5}$$

and

$$\phi(k) = arctan\frac{X_i(k)}{X_r(k)}, \tag{3.6}$$

with again $k = 0, 1, \dots N_{frame} - 1$. According to the *Nyquist-Shannon sampling theorem*, the amount of data points in $X(k)$ is equal to N_{frames} and positive frequencies range from 0 Hz up to half of the sampling rate $f_s/2$ Hz. The resolution of the frequency is therefore $\frac{f_s}{Nframe}$.

The size of the analysis frame N_{frames} is a critical parameter in the calculation of the DFT. A small amount of frames results in a good temporal resolution of the whole signal. On the other hand, an increase of N_{frames} also increases the frequency resolution.

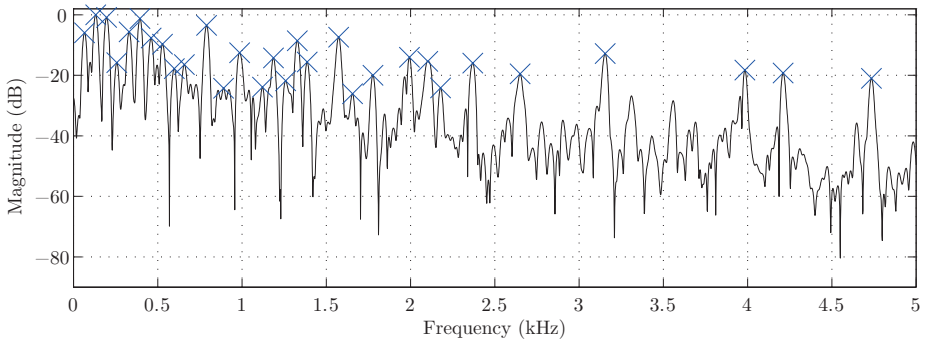


Figure 3.3: *Spectrum* with marked peak values.

Zero-padding

To increase the frequency resolution without using more samples of the input signal, [20, 45] recommend a technique called *zero-padding*.

Zero-padding takes the N_{frame} samples of the signal $x(n + l \cdot N_{hop})$ and extends it by adding samples containing zeros until a desired size of N_{FFT} samples is reached.

This zero-padded signal

$$X_{zp}(n) = \begin{cases} 0 & \text{for } n = -\frac{N_{FFT}}{2} \ldots -\frac{N_{frame}}{2} - 1, \\ x_{wc}(n) & \text{for } n = -\frac{N_{frame}}{2} \ldots \frac{N_{frame}}{2} - 1, \\ 0 & \text{for } n = \frac{N_{frame}}{2} \ldots \frac{N_{FFT}}{2} - 1, \end{cases} \tag{3.7}$$

now has the required frequency resolution $\frac{f_s}{N_{FFT}}$ with $N_{FFT} - N_{frame}$ zeros added to the original signal. In my implementation each window is zero-padded to $N_{FFT} = 4 \cdot N_{frame}$ so it uses 4 times the amount of samples as in the original frame.

It is important to note that zero-padding does not add more information to the signal. It just improves the information that already exists by increasing its resolution.

3.1.4 Peak detection

In the last pre-processing step I extract the information needed to create the HPCP vector. Therefore, the peak points (i.e. local maxima) of the spectrum generated in 3.1.3 are extracted.

A peak detection algorithm[4] is applied to the spectrum, looking for points that are surrounded by points with a lower magnitude than itself. I defined

[4] I used an algorithm by Edi Billauer which is available for free. http://billauer.co.il/peakdet.html

a threshold value of 1, meaning that it only declares a point as a peak if the difference to surrounding points is > 1.

The amplitude values of the spectrum used for peak-detection are given on a decibel (dB) scale, so every amplitude value a_i is in relation to the maximum possible magnitude a_{max} so that

$$a_i(dB) = 20 \cdot \log_{10} \frac{a_i}{a_{max}}. \tag{3.8}$$

This results in $a_{max} = 0$ and all other values are on a logarithmic scale below it. The dB scale is adapted to the human perception of sound and therefore peaks are more distinct in the dB spectrum.

The found peak points are stored in a list of frequency/amplitude pairs $\{a_i, f_i\}$ with $i = 1 \ldots n_{Peaks}$. Notice that the amplitude a_i is stored as linear instead of the dB value. The frequency f_i is stored in Hz according to its bin position and the frequency resolution $\frac{f_s}{N_{FFT}}$.

Results are filtered and only peaks within a range of 40 – 5000 Hz are considered. This is important because of percussion and instrumental noise. The audio is more noisy in high frequency regions. Also a threshold of -100 dB with respect to the local maximum possible magnitude is applied. Peaks with a maxima below this threshold are discarded. These filter ranges are recommended and proved to generate reliable results by [20, 45].

3.1.5 Reference frequency estimation

In the last pre-processing step, the reference frequency has to be estimated. This frequency is the one which all the instruments of a song are tuned to. In western music it is usually 440 Hz but we cannot assume that all instruments are perfectly in tune. To make the algorithm robust against variations of the reference frequency, we have to estimate it for every analyzed song. It is important for scaling the center frequencies of the HPCP bins and to fold each overtone series into a pitch class.

Basically there are two different ways for computing a songs reference frequency. One way is to estimate it before frequency values are mapped to pitch classes. Another way is to generate pitch classes using a standard reference frequency and shift the features afterwards to tune them to the right reference frequency.

In my implementation I am using the first approach. Therefore the reference frequency for each analysis frame is estimated by inspecting the deviation of the spectral peaks. A global frequency is then generated by combining the frame estimates.

Looking at the detected peak points of one frame the lowest peak frequency should be the fundamental frequency f_0 of all the others i.e. all the overtones are integer multiples of it. f_0 is defined as

$$f_0 = \max_N \{g_j \in \mathcal{G} : ng_j \in \mathcal{G}, n = 1, \ldots, N\}. \tag{3.9}$$

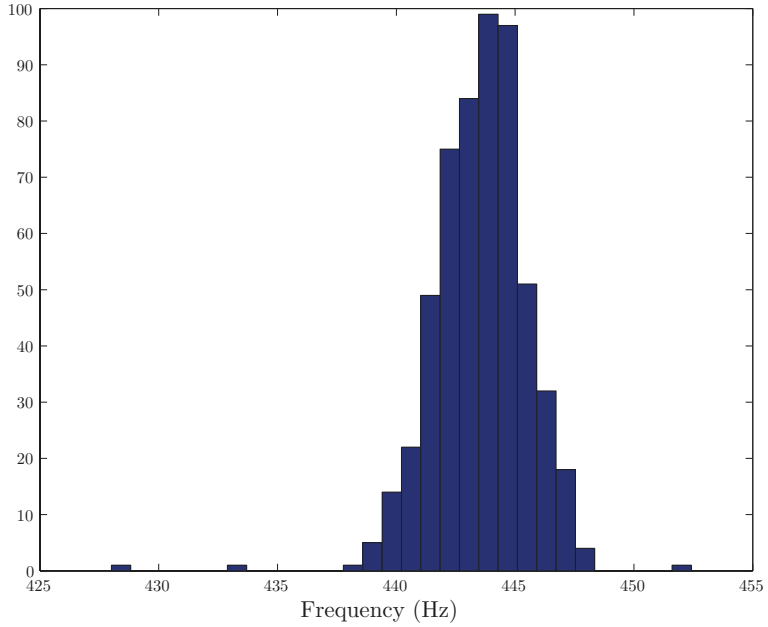


Figure 3.4: Histogram of estimated reference frequencies.

Assuming a 12-tone equal tempered tuning system with the reference frequency f_{ref}, the fundamental frequency f_0 of a given pitch can be calculated using

$$f_0 = f_t \cdot 2^{\beta(f0;f_r)/12}, \tag{3.10}$$

with $\beta(f_0; f_{ref})$ describing where the note is on the musical octave.

$$\beta(f_0; f_{ref}) := 12 \log_2(f_0/f_{ref}). \tag{3.11}$$

Knowing that $\beta(f_0; f_{ref})$ must be an integer, it is possible to adjust f_{ref} to make $\beta(f_0; f_{ref})$ an integer value in a plausible range of $f_{ref} \in [410, 460]$ Hz.

Unfortunately, overtones appearing in a perfect integer relationship are very rare or even not existing in reality. Therefore I am looking for a fundamental that fits for most of the overtone frequencies. This is done by only taking frequencies into account that are situated in the area around the perfect harmonic frequencies. Therefore, I define

$$f_0 = \max_N \{g_j \in \mathcal{G} : g \in [(1-\delta)ng_j, (1+\delta)ng_j] \in \mathcal{G}, n = 1, \ldots, N\}, \tag{3.12}$$

using a small $\delta \geq 0$. To get a estimated reference frequency for a whole musical piece, I create a histogram of the f_0 values from all windows and choose the frequency with the highest occurrence (see figure 3.4).

It is not necessary to estimate the reference frequency using the whole audio signal. Good results can be achieved by using just a 20 s sample from the

center of the signal (without removed transients). This reduces calculation time and speeds up the algorithm.

This method is an enhancement[5] of the one used in [20, 45].

3.2 HPCP computation

After the pre-processing procedure, we now have a reference frequency f_{ref} and a list of peak points found in the power spectrum containing frequency/amplitude pairs $\{a_i, f_i\}, i = 1 \dots n_{Peaks}$. This information is the input for the computation of the Harmonic Pitch Class Profile vector.

HPCPs describe the energy of each semitone at a given time frame of the audio file. It provides a sliced representation of a musical piece, where each slice is a vector $HPCP(n)$ describing the power of each semitone n. The HPCP vector for one time frame is defined as

$$HPCP(n) = \sum_{i=1}^{n_{Peaks}} \omega(n, f_i) \cdot a_i^2, \tag{3.13}$$

with $n = 1 \dots size$. $size$ equals multiples of 12 as they are representing the 12 different notes on the western musical octave. The HPCP bins n are dependent on the reference frequency f_{ref} (estimated in 3.1.5) and the amount of bins $size$.

3.2.1 Weighting

The energy of a peak a_i is not mapped to a single HPCP bin n but contributes to bins in its surrounding area depending on the length l of the weighting window. This is done to minimize estimation errors that occur if there are tuning differences or inharmonicity in the spectrum.

The contribution of each peak is specified by the weighting function $\omega(n, f_i)$ in eq. 3.13 dependent on frequency f_i considering HPCP bin n. Basically the weighting uses a cos^2 function centered between bin frequencies. In my implementation I used a $size$ of 36 bins so every bin represents $\frac{1}{3}$ of one note of the musical octave. For the weighting-window length $l = \frac{4}{3}$ semitone are used. The result is that each peak will contribute to 4 different HPCP bins (as illustrated in figure 3.5). These values are proposed in [20,45].

The center frequency f_n of bin $n = 1, \dots, N$ is

$$f_n = f_{ref} \cdot 2^{\frac{n}{size}}. \tag{3.14}$$

The distance between a peak frequency f_i and the bins center frequency f_n is defined as

$$d = 12 \cdot \log_2(\frac{f_i}{f_n}) + 12 \cdot m. \tag{3.15}$$

[5] It was introduced by Bob L. Sturm in http://media.aau.dk/CRISSP/2010/11/tuning-frequency-determination.html.

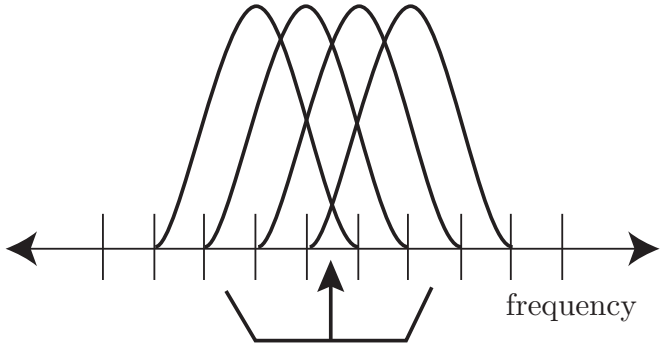


Figure 3.5: Weighting window of length $l = \frac{4}{3}$ semitone.

m is an integer that we choose to minimize $|d|$, so all frequencies are mapped to a single octave. Finally, the weight of peak f_i, a_i to bin n can be calculated using

$$\omega(n, f_i) = \begin{cases} \log_2(\frac{f_i}{f_n}) + 12 \cdot m & \text{if } |d| \leq 0.5 \cdot l, \\ 0 & \text{if } |d| > 0.5 \cdot l. \end{cases} \tag{3.16}$$

3.2.2 Harmonics

Most of the time signals that humans describe as pitched tones are not the result of a single– but combinations of multiple frequencies. If we observe the spectrum of an instrument playing a single note, we can observe that spectral components appear at the notes *fundamental frequency* f_0 and also at harmonic frequencies i_n. These harmonic frequencies (or harmonics) are (theoretically) integer multiples of the fundamental frequency ($f_0, 2 \cdot f_0, 3 \cdot f_0$ etc.).

Harmonic pitch class profiles take this fact into account by considering each peak frequency f_i as a possible fundamental- and harmonic frequency. Therefore they contribute not only to the pitch class related to f_i, but also to the pitch classes for frequencies that have f_i as harmonic frequency (i.e. $f_i, \frac{f_i}{2}, \frac{f_i}{3}, \frac{f_i}{4}, \ldots, \frac{f_i}{n_{Harmonics}}$).

To simulate the decrease of the spectrum amplitude for the calculated harmonics, a weighting function

$$\omega_{harm}(n) = s^{n-1}, \tag{3.17}$$

is introduced (see figure 3.6). $s < 1$ defines the curve and should ideally depend on the played instruments timbre. I used $s = 0.6$ and $n_{Harmonics} = 7$ as assumed by the authors of [20, 45].

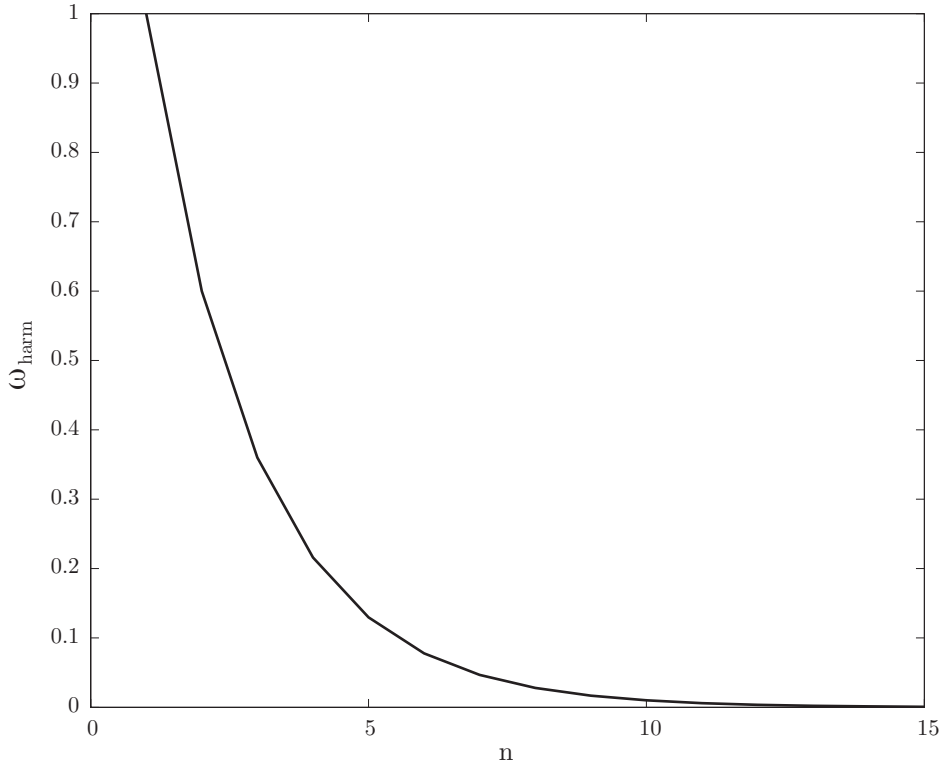


Figure 3.6: Weight of the harmonics $s = 0.6$.

3.2.3 Spectral whitening

In [20,45] the authors apply a timbre normalization to each peak point before adding its magnitude to a HPCP bin. As all detected notes are folded to a single octave, this step ensures that notes from all octaves contribute equally to the final HPCP vector. This is important in order to make the system robust against different instrument configurations or equalization settings.

Therefore the magnitude values of each peak are normalized according to the corresponding value of the *spectral envelope* of the spectrum. This envelope is a special smoothed representation of the spectral curve describing the general trend of it. The progress of normalizing a spectrum with its spectral curve is called *spectral whitening*. An overview about different methods to derive the spectral envelope of a spectrum are presented in [40, 43].

In my implementation I am using *Linear Predictive Coding* as described in [43] to estimate the spectral envelope. This function builds the spectral envelope as the transfer function of an all-pole filter with order p poles. In my implementation I used a filter order $p = 110$.

In a pre-processing step I inverted the estimated spectral envelope and used it as a filter (as demonstrated in figure 3.7) to smooth the spectrum.

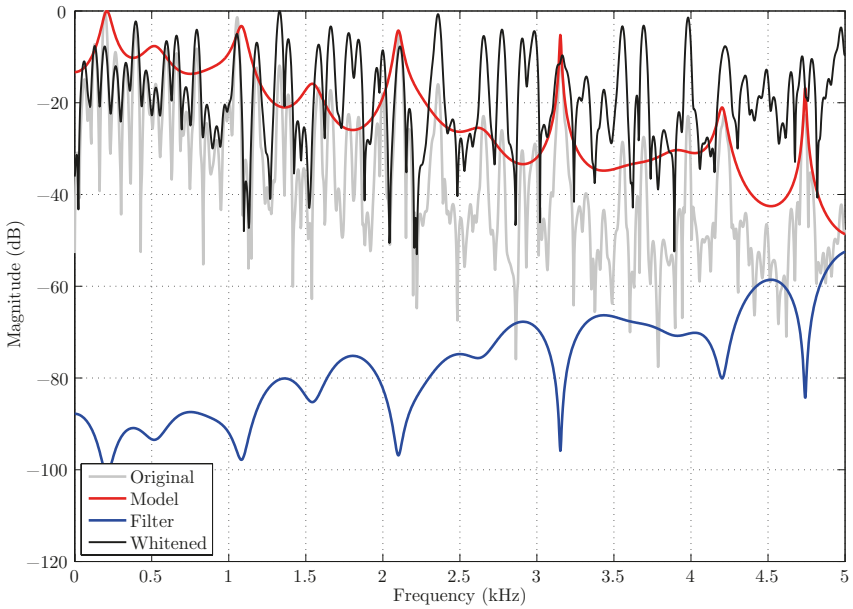


Figure 3.7: A spectrum; its spectral envelope; the flat spectrum. The spectral envelope was computed using a order of $p = 110$ poles.

The frequencies of the peaks used for the HPCP calculation are extracted using the unwhitened signal but for further calculation I used the magnitude values from the whitened spectrum.

According to [20, 45] the spectral whitening process should improve the systems accuracy but unfortunately it is insufficiently described so a lot of time was spent collecting the pieces that are missing. I tested several different implementations and settings but my algorithm always performed worse no matter how I whitened the spectral peaks. Therefore I decided to not apply the spectral whitening procedure in my system. For detailed results and explanations see chapter 4.

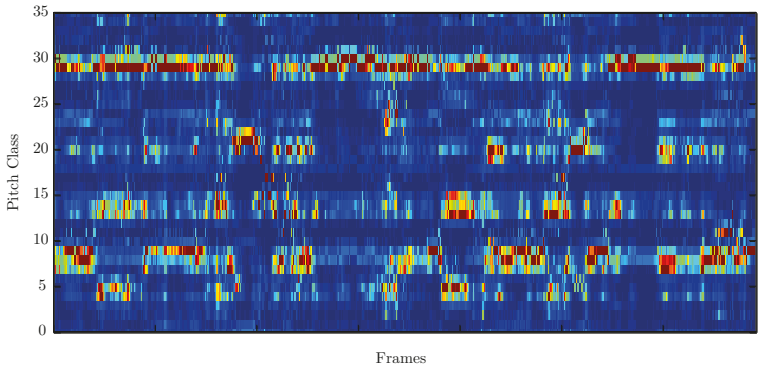
3.3 Post-processing

3.3.1 Normalization

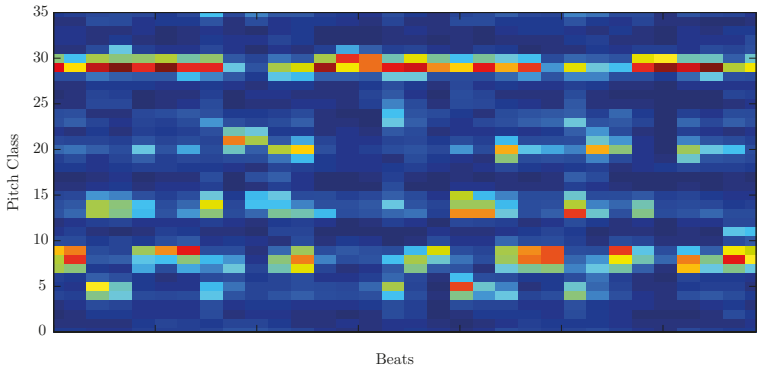
As we are dealing with linear magnitude values we need to normalize the HPCP vector in order to make them comparable. Therefore each $n = 1 \ldots size$ column of the HPCP vector $HPCP_n$ is normalized with respect to its maximum value. This process is defined as

$$HPCP_{normalized}(n) = \frac{HPCP(n)}{Max_n(HPCP(n))}. \tag{3.18}$$

Figure 3.8 a shows a visualization of a normalized HPCP vector.



(a)



(b)

Figure 3.8: *HPCP vector* (a) Normalized, (b) Average beat vector.

3.3.2 Transposition

As discussed in chapter 2.5.2 there are several ways to handle key differences. In my implementation I shift the normalized HPCP vector $HPCP_{normalized}$ according to a certain index $shift$. To define this index, I sum up the rows of the whole vector and set $index$ to the pitch class containing the maximum value. This method is based on the consideration that the most distinctive pitch class stays the same in different versions of a song, regardless of differences in the main key.

The transposed HPCP vector $HPCP_{transposed}$ is defined as

$$HPCP_{transposed}(n) = HPCP_{normalized}(\mathrm{mod}(n - shift, size)), \qquad (3.19)$$

for $n = 1 \ldots size$.

It is important that the HPCPs are aligned to the same key for similarity estimation. Otherwise even the same song transposed to another key would

not be recognized by the cross-correlation function.

3.4 Similarity estimation

This section presents two methods for comparing different HPCP vectors in order to estimate their similarity. As mentioned in chapter 2.6 there are many different ways to calculate the similarities of features. In my implementation I compared the generated HPCP vectors using a *Cross correlation* and a *dynamic time warping* approach.

3.4.1 Cross-correlation

The cross correlation approach first has to deal with possible tempo variations (as discussed in chapter 2.5.3). It does this by generating a beat aligned representation of the HPCP matrix. Therefore, I use the estimated beat timestamps created by the beat detection algorithm in chapter 3.1.1 and calculate an average HPCP from all values located between two beats. The result is a beat aligned HPCP vector (see figure 3.8b) in which each column represents the information of one beat instead of a certain time frame.

The beat aligned HPCP representations of two songs can now be compared using cross-correlation. One could say that the HPCP matrices of two songs are put on top of each other and compared as if they where images. They are shifted one beat at a time and the similarity is calculated for every step. The generated correlation graph (as shown in figure 3.9) describes the correlation of the two songs at each beat position. We use the maximum value of the graph as the similarity descriptor for the two pieces (a perfect match e.g. a song compared to itself would generate a maximum value of 1).

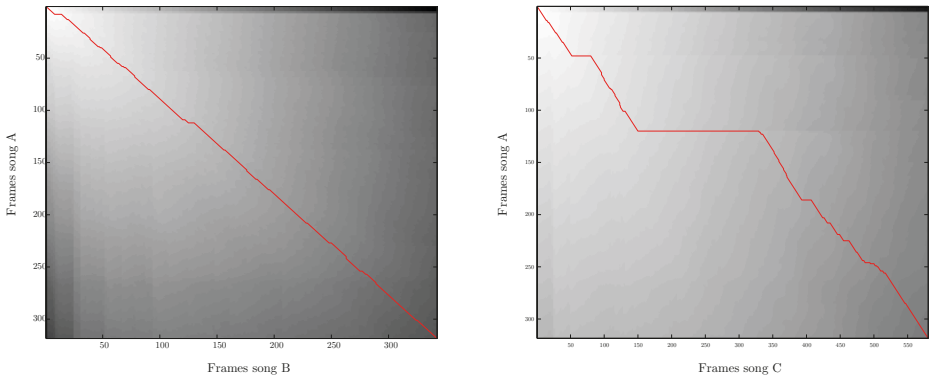
In my implementation I use the cross-correlation presented in [16]. The source code is available online[6].

3.4.2 Dynamic time warping

I also tested a second approach for similarity estimation of HPCPs using *dynamic time warping* (DTW) with global constraints. DTW is a method to measure similarities between two sequences which may vary in time or speed.

To use DTW the first step is to create a cumulative distance matrix representing the similarity between the HPCPs which should be compared. This matrix describes the similarity of each HPCP vector at each given timeframe (see figure 3.9). In the next step a path finding algorithm is used to find the shortest alignment path in the distance matrix. The length of the path is normalized by the length of the songs. Notice that in contrast to the

[6] http://labrosa.ee.columbia.edu/projects/coversongs/



(a) (b)

Figure 3.9: Cumulative distance matrix and alignment path. (a) version-pair e.g. short alignment path, (b) non-version pair e.g. long alignment path.

cross-correlation approach, a smaller value indicates a higher similarity (a song compared to itself would score 1).

A improved method using a binary similarity matrix and local instead of global constraints is presented in [49].

Chapter 4

Evaluation and discussion

Basically we can differentiate between two fields of application for version detection algorithms.

Finding versions in a collection of songs: The input is one query song which is compared to a set of songs in a database. The goal is to find the version(s) of the query from the database and rank them according to their similarity. This task assumes that there is at least one version in the test set.

Comparing two songs: The query consists of two songs and the output is either true if the pair is identified as versions of each other or false if not. The similarity of the songs is estimated as usual and compared to a certain threshold.

My system is designed to use the first approach as it gives me a good overview of my systems progress during the development. Moreover, many other version detection algorithms use the same approach which makes it easier to compare the results. Before I could start to test my system I had to find a collection of songs to use as data and query. While searching for an appropriate test set I became aware of the MIREX *audio cover song identification task*.

4.1 The MIREX cover song identification task

MIREX i.e. *Music Information Retrieval Evaluation eXchange*[1] is an annual evaluation campaign for music information retrieval algorithms. It hosts several competitions where researchers can hand in their work to let it compete on different MIR tasks using common evaluation standards. Since 2006 there is also an "audio cover song identification task" which they define as follows:

> This task requires that algorithms identify, for a query audio track, other recordings of the same composition, or "cover songs".

[1] http://www.music-ir.org/mirex/wiki/MIREX_HOME

> Within the collection of pieces in the cover song datasets, a number of different "original songs" or compositions are embedded, each represented by a number of different "versions". The "cover songs" or "versions" represent a variety of genres (e.g., classical, jazz, gospel, rock, folk-rock, etc.) and the variation span a variety of styles and orchestrations. Using each of these version files in turn as the "seed/query" file, we examine the returned ranked lists of items from each algorithm for the presence of the other versions of the "seed/query" file.

The dataset used for the cover song identification task contains 1000 pieces. Within these 1000 pieces there are 30 pieces that are represented by 11 different versions. Accordingly, a total of 330 audio files are versions. All the versions are used as query on the whole dataset. The target is to return a list of the 10 other versions of the query. The songs of the set are not published to avoid that algorithms get trained to the dataset. Therefore, I had to look for another test set to test my implementation.

4.2 The `covers80` dataset

During further research I found out that the authors of [16] created `covers80`[2], a dataset to evaluate version detection algorithms. The `covers80` collection consists of 80 version sets with a cardinality of 2. They are split into two lists with each list containing the same songs performed by different artists (i.e. large changes in style and/or harmonic). The complete list can be found in appendix A.

They describe the way they picked the songs for the set as follows:

> The covers were assembled somewhat haphazardly. First we went through the 8764 pop music tracks in `uspop2002`[3], listening to any tracks with the same name to see if they were covers. That didn't yield enough, so we decided to look for as many pairs as we could for two albums of cover songs we happened to have, one by Annie Lennox ("Medusa") and one by Tori Amos ("Strange Little Girls"). The rest were collected even more randomly.

They also published the code they used in [16] to do version detection. This solves two problems: I have a collection of songs to test my algorithm with and I can compare my results to their algorithm.

[2] `covers80` is available at http://labrosa.ee.columbia.edu/projects/coversongs/covers80/

[3] The `uspop2002` Pop Music data set is available at http://labrosa.ee.columbia.edu/projects/musicsim/uspop2002.html

4.3 The my30 dataset

In addition to the covers80 collection I also created a second dataset (my30) using 10 versions with a cardinality of 3. Each version set contains three different versions of one song:

1. a studio recording,
2. a live version performed by the same artist as the studio recording,
3. and a cover versions performed by a different artist.

Just as for covers80 the collection is divided into two lists. The first list contains the studio versions (i.e. a total of 10 songs) while the second list contains the live and the cover versions (i.e. a total of 20 songs). All songs were extracted from (and therefore can be found on) the internet video platform YouTube[4]. For a list of the songs used in my30 see appendix B.

The difference to the covers80 set is the cardinality of 3 and the quality of the audio files. The songs in my30 have a higher sample rate of 44.1 kHz compared to 16 kHz in covers80.

4.4 Evaluation measure

To evaluate the accuracy of the developed system in identifying song versions I run a test on each dataset. Therefore I choose a similar method to the one used in the MIREX competition.

I start with two lists, one containing the query songs, another one containing the versions. The goal is to detect all versions (depending on the cardinality of the dataset) from the second list using the songs from the first one as query. For this test we assume that there is at least one right match.

In the first step, a similarity matrix is created by calculating the similarity of all possible pairs. Each song from the first list is compared to all songs from the second list. The result is a similarity matrix of the size 80×80 for covers80 and 10×20 for my30.

Every row of the matrix represents the similarity of one song of the first list with all the songs in the second list. Each cell contains the estimated similarity of the corresponding song pair. As the order of the songs is the same in both lists (only the artists vary), matching versions are arranged on a diagonal line.See the similarity matrix produced by my algorithm using the covers80 test in figure 4.1.

To find the corresponding version in the other list, the entries in every row are sorted in descending order. The result is a ranking of all the songs in the second list compared to the query song from the first list. The first entry is the most similar and therefore assumed to be a version of the query.

To evaluate the success of an algorithm using this method, the amount

[4] http://youtube.com

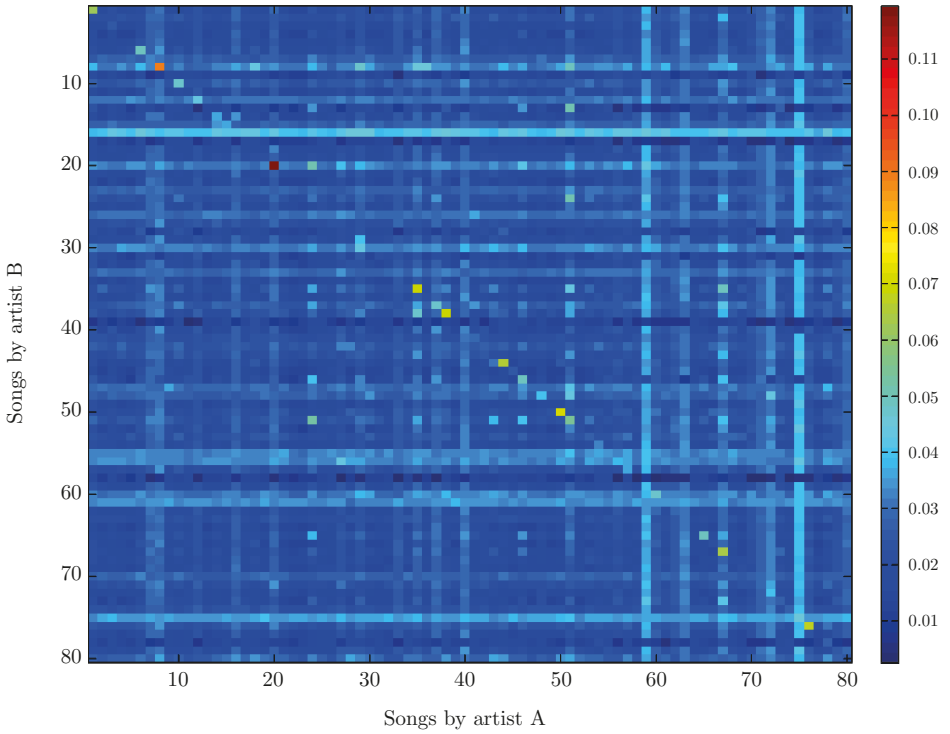


Figure 4.1: Results using cross-correlation of HPCPs on the `covers80` dataset.

of possible matches in the dataset M is compared to the number of correct matches found by the algorithm m_{cor} so the success rate μ is defined as

$$\mu = \frac{m_{cor}}{M}. \tag{4.1}$$

In addition, we introduce the variable x which defines the highest rank that is still considered as a match. Therefore, x can be used to define the algorithms accuracy. With the highest accuracy (e.g. $x = 1$) only the song with the highest correlation to the query is assumed to be a version.

4.5 Results

In the following chapter the results of my test runs are presented and explained. All the relevant settings I used for the tests are summarized in table 4.1 and described in chapter 3.

Description	*Value*
Window size (N_{frame})	4096 Samples
Peak min. frequency	40 Hz
Peak max. frequency	5000 Hz
Peak energy threshold	−100 dB
Weighting window length (l)	$\frac{4}{3}$ semitone
Amount of bins ($size$)	36
Parameter for weighting function of harmonics (s)	0.6
No. of harmonics ($n_{Harmonics}$)	7
Number of peaks used for HPCP creation (n_{peaks})	20
Detuning factor for considering frequency a harmonic (δ)	0.01
Assumed reference frequency	440 Hz
Spectral whitening filter order (p)	110
Frame size for Dynamic Time Warping ($size$)	24 Frames

Table 4.1: All relevant settings used for the evaluation.

4.5.1 Test results for the `covers80` dataset

For the first test I used the `covers80` dataset. As described in chapter 4.4, each of the 80 songs from the first list is compared to the songs from the second one using cross-correlation. From $M = 80$ possibles the algorithm detected $M_{1,CC} = 21$ versions as right. This equals a success rate of 26%.

Using dynamic time warping for similarity estimation the algorithm was able to find $M_{1,DTW} = 6$ out of 80 versions. This equals a 7.5% success rate.

The detailed results for this test are listed in table 4.2. A graphical representation of the cross-correlated results is shown in figure 4.1.

In comparison, table 4.2 also lists the results of the algorithm presented in [16]. It detected 30 right versions in the `covers80` dataset using cross-correlation. In other words, the algorithm found the right version for 37.5% of the songs in the collection. The system was not designed to work with dynamic time warping so no values are listed.

4.5.2 Test results for the `my30` dataset

The second test was performed using the songs of the `my30` dataset. In this set, the maximum possible amount of correct matches $M = 30$ because there are 10 songs in the query list and 2 versions for each in the collection (i.e. the test set has a cardinality of 3). Therefore we have to look at the two highest ranked songs in the similarity vector. In case of a match the first two ranks of a song are its live and cover version, or at least one of them.

My approach using HPCP features and cross-correlation found $M_{2,CC} =$

Features	HPCP								MFCC
Comparison method	CC				DTW				CC
x	1	3	5	10	1	3	5	10	1
M_x	21	26	30	34	6	10	17	22	30
Success rate	0.26	0.33	0.38	0.43	0.08	0.13	0.21	0.28	0.38

Table 4.2: Performance of my algorithm using HPCP features and the algorithm presented in [16] using MFCC features on the `covers80` test set. Maximum possible matches $M_{max} = 80$. $CC \ldots$ Cross-Correlation, $DTW \ldots$ Dynamic time warping, $M_x \ldots$ Correct matches found in the first x positions of the ordered similarity vector.

Features	HPCP				MFCC
Comparison method	CC		DTW		CC
x	2	5	2	5	2
M_x	11	13	7	8	16
Success rate	0.55	0.65	0.35	0.4	0.8

Table 4.3: Performance of my algorithm using HPCP features and the algorithm presented in [16] using MFCC features on the `my30` test set. Maximum possible matches $M_{max} = 20$. $CC \ldots$ Cross-Correlation, $DTW \ldots$ Dynamic time warping, $M_x \ldots$ Correct matches found in the first x positions of the ordered similarity vector.

11 matching versions which is a success rate of 55%. Using dynamic time warping, the algorithm was able to detect $M_{2,DTW} = 7$ versions (35%).

I also compared the results to the algorithm presented in [16]. It was able to detect 16 out of 20 versions as accordant. This makes up a success rate of 80%. Detailed results are listed in table 4.3.

4.6 Discussion

The test results in chapter 4.5 show that my algorithm performed worse compared to the one presented in [16]. This could have many reasons that I will explain in the following section.

First of all, the results of the MIREX cover song identification task[5] and chapter 4.1) clearly show that HPCPs could score better results. The

[5]Results of the MIREX audio cover song identification task are available online http://www.music-ir.org/mirex/wiki

three most successful algorithms ever submitted to the contest were based on HPCP features[6].

These evaluations proved that HPCP vectors are robust features that can be used for version detection tasks. HPCP features are relatively new in the MIR field and therefore there are not much publications or implementations of it. The authors of [20,45] describe the basic creation of HPCP, but certain parts of these papers are either assumed to be known or not discussed in proper detail. Hence, it was hard to find answers to questions that appeared during the implementation. For some I found answers in referenced papers, others were solved by trying different methods and settings to stick with the one that worked best.

Anyhow, I am sure that there are still some bad chosen settings or wrong assumptions included in my current algorithm. Consequently, these are responsible for the test results that did not match the expectations. The presented system found the right versions for over $\frac{1}{4}$ of the `covers80` dataset and even over $\frac{1}{2}$ of the `my20` dataset. Considering this, I still proved that my approach using cross-correlation of beat aligned Harmonic Pitch Class Profiles worked for music version detection tasks.

In the next sections I present an overview of different considerations that I found out during the work on this thesis.

Sample rate of the audio signal

One difference between the two datasets, except from the amount of songs and the cardinality, is the quality of the audio files. The authors of `covers80` used audio files with a sample rate of 16 kHz, whereas I used 44.1 kHz files for the `my30` set.

This is important as I specified the window size N_{frame} in samples and not milliseconds. The result is that the windows of HPCP vectors created from files of the `covers80` collection have a length of over 0.256 seconds compared to 93 ms using files from `my30`.

This is one possible reason why both algorithms performed better on the `my30` than on the `covers80` dataset.

Comparison methods

Another detail that is visible in the results is that the cross-correlation performed better in all tasks compared to the dynamic time warping. This is because I used the length of the normalized global alignment path which does not vary as much as expected. On the other hand, the success rate of the cross-correlation strongly depends on the reliability of the beat detection method, as we are correlating beat aligned HPCP vectors.

[6]The best algorithm in 2008 [48] had a success rate of 75%. For a detailed list of results see [45].

A more advanced dynamic time warping approach is introduced by the authors of [49]. They suggest to use a binary distance matrix and local instead of global constraints to enhance results.

4.6.1 The used datasets

In general, the success of course is influenced by the songs used in the dataset. Versions that change dramatically in style and genre (like e.g. a Hard Rock cover version of a Pop song) are harder to detect than versions that only change in a few aspects (like e.g. live performances by the same artist as in `my30`).

Another interesting observation I made during testing is, that regardless which test set I used, nearly all the time one or two songs correlate significantly higher to all the other songs. See e.g. song 16 by artist B or 59 by artist A in figure 4.1. I found out that the existence of these so called *Hubs* (i.e. Songs which appear similar to many other songs without showing any meaningful perceptual similarity) is a well known effect. In [19] the authors describe this effect and explain that it depends on the homogeneity of the samples used for feature creation. As these Hubs are most likely to distort the results, future work is seen in trying to detect and handle them.

4.6.2 Spectral whitening

One of the main problems that occurred while implementing the system was the spectral whitening of the peak points. [20, 45] mention the importance of this processing step but only describe it very briefly. Referenced work like [40] only deal with the way how to generate the spectral envelope e.g. the curve used to normalize the peaks but not when applying the whitening process. Therefore, I had to try different methods and test each one to see if the accuracy of the algorithm rises.

I tried to extract the peaks from the whitened signal and use their magnitude values for further calculations. Another approach was to use the un-whitened signal to find the peaks to then normalize their magnitudes according to the corresponding value of the estimated spectral envelope. I tried all possible combinations of whitened and un-whitened values applied at different processing steps of the algorithm, but in the end, the estimated similarity between two versions always got worse. In conclusion, I decided to exclude the spectral whitening and use un-whitened peaks as in [18].

Chapter 5

Conclusion

Music version detection is an interesting topic in MIR and so far many different approaches to solve this problem have been developed. In this thesis I provided an overview of different state of the art methods for music version detection and presented an implementation of my own approach. The developed algorithm combines two existing methods by using beat-aligned HPCP features of [45, 49] with the cross-correlation of [16]. I implemented the algorithm using MATLAB and tested it on two different datasets consisting of 2×80 and 3×10 songs. The system presented in [16] was used for comparison to measure the performance. The results of the tests proved that my approach basically works, but is not yet fully developed. Other types of approach using the same musical features score much higher accuracies at common MIR tasks.

A music version detection algorithm as the one presented in this thesis basically consists of two main parts. The first part is the feature extraction where musical aspects that are most likely to stay the same over different versions of a song are extracted from raw audio signals in order to be represented as features. Typical aspects are tonal sequences or harmonic progression. For my algorithm I used so called *Harmonic Pitch Class Profiles*. These feature vectors are a representation of the energy of each tone of the musical octave at a certain time frame of the audio signal.

I compared these features using a cross-correlation approach. Therefore, the extracted HPCP vectors of two songs are compared like images to find their maximum correlation. This value can be used to describe the similarity between the musical pieces. Furthermore, a second way to compare the similarity by creating a cumulative distance matrix of two songs and use the alignment costs as indicator for similarity was tested.

Some problems appeared while implementing the algorithm. [20, 45] present a step–by–step guide on how to create HPCP features out of an audio signal. Unfortunately, some parts of the papers provide only insufficient details on important parts like e.g. the spectral whitening of the peaks magni-

tude values. It is mentioned in a few sentences in [20,45] but further instructions on how exactly it is done are missing. I tried several ways to whiten the peaks for HPCP creation but the results only got worse.

Another problem was the debugging. There is no effective way to debug a complex system like a music version detection algorithm. For example if I changed some settings of the algorithm that influenced the way how the features are created, I had to analyze at least two songs and compare their features to find out if the new setting changed the algorithms behavior for the better or the worse.

Future directions are seen in exploring the settings involved in creating HPCP vectors. An interesting question could be if there are ways to tune the features to work better for e.g. a specific genre or task. Moreover, further experiments using different settings could provide new knowledge on their influence towards the end result.

Appendix A

The `covers80` dataset

Nr.	*Title*	*Artist A*	*Artist B*
1	A Whiter Shade Of Pale	Annie Lennox	Procol Harum
2	Abracadabra	Steve Miller Band	Sugar Ray
3	Addicted To Love	Robert Palmer	Tina Turner
4	All Along The Watchtower	Bob Dylan	Jimi Hendrix Experience
5	All Tomorrow S Parties	Japan	Velvet Underground
6	America	Paul Simon	Simon And Garfunkel
7	Before You Accuse Me	Creedence Clearwater Revival	Eric Clapton
8	Between The Bars	Elliott Smith	Glen Phillips
9	Blue Collar Man	Reo Speedwagon	Styx
10	Caroline No	Beach Boys	Brian Wilson
11	Cecilia	Paul Simon	Simon And Garfunkel
12	Claudette	Everly Brothers	Roy Orbison
13	Cocaine	Eric Clapton	Nazareth
14	Come Together	Aerosmith	Beatles
15	Day Tripper	Beatles	Cheap Trick
16	Don T Let It Bring You Down	Annie Lennox	Neil Young
17	Downtown Lights	Annie Lennox	Blue Nile
18	Enjoy The Silence	Depeche Mode	Tori Amos
19	Faith	George Michael	Limp Bizkit
20	God Only Knows	Beach Boys	Brian Wilson

Nr.	*Title*	*Artist A*	*Artist B*
21	Gold Dust Woman	Fleetwood Mac	Sheryl Crow
22	Grand Illusion	Reo Speedwagon	Styx
23	Happiness Is A Warm Gun	Beatles	Tori Amos
24	Heart Of Gold	Neil Young	Tori Amos
25	Hush	Deep Purple	Milli Vanilli
26	I Can T Get Next To You	Annie Lennox	Temptations
27	I Can T Get No Satisfaction	Britney Spears	Rolling Stones
28	I Don T Like Mondays	Boomtown Rats	Tori Amos
29	I Don T Want To Miss A Thing	Aerosmith	New Found Glory
30	I Love You	Celine Dion	Faith Hill
31	I M Losing You	Cheap Trick	John Lennon
32	I M Not In Love	10Cc	Tori Amos
33	It S Tricky	Bloodhound Gang	Run DMC
34	Lady	Reo Speedwagon	Styx
35	Let It Be	Beatles	Nick Cave
36	Little Wing	Corrs	Eric Clapton
37	Lodi	Creedence Clearwater Revival	Tesla
38	Love Hurts	Heart	Nazareth
39	Maggie S Farm	Bob Dylan	Rage Against The Machine
40	More Than Words	Extreme	Westlife
41	My Generation	Green Day	Who
42	My Heart Will Go On	Celine Dion	New Found Glory
43	Never Let Me Down Again	Depeche Mode	Smashing Pumpkins
44	New Age	Tori Amos	Velvet Underground
45	Night Time Is The Right Time	Aretha Franklin	Creedence Clearwater Revival
46	No Woman No Cry	Bob Marley	Fugees
47	Oh Pretty Woman	Al Green	Roy Orbison
48	Ooby Dooby	Creedence Clearwater Revival	Roy Orbison
49	Proud Mary	Creedence Clearwater Revival	Tina Turner
50	Purple Rain	Leann Rimes	Prince

Nr.	*Title*	*Artist A*	*Artist B*
51	Rattlesnakes	Lloyd Cole And The Commotions	Tori Amos
52	Real Men	Joe Jackson	Tori Amos
53	Red Red Wine	Neil Diamond	UB40
54	River Deep Mountain High	Celine Dion	Tina Turner
55	September Gurls	Bangles	Big Star
56	She Came In Through The Bathroom Window	Beatles	Joe Cocker
57	Something So Right	Annie Lennox	Paul Simon
58	Stone Cold Crazy	Metallica	Queen
59	Straight From The Heart	Bonnie Tyler	Bryan Adams
60	Strange Little Girl	Stranglers	Tori Amos
61	Street Fighting Man	Rage Against The Machine	Rolling Stones
62	Summer Of 69	Bryan Adams	Mxpx
63	Summertime Blues	Alan Jackson	Beach Boys
64	Take Me To The River	Al Green	Talking Heads
65	Take On Me	A Ha	Mxpx
66	The Boxer	Paul Simon	Simon And Garfunkel
67	Thin Line Between Love And Hate	Annie Lennox	Persuaders
68	Time	Tom Waits	Tori Amos
69	Tomorrow Never Knows	Beatles	Phil Collins
70	Toys In The Attic	Aerosmith	R.E.M.
71	Train In Vain	Annie Lennox	Clash
72	Tush	Nazareth	Zz Top
73	Waiting In Vain	Annie Lennox	Bob Marley & The Wailers
74	Walk This Way	Aerosmith	Run DMC
75	Walking After Midnight	Bryan Adams	Garth Brooks
76	We Can Work It Out	Beatles	Tesla
77	What S Going On	Cyndi Lauper	Marvin Gaye
78	White Room	Cream	Sheryl Crow
79	Wish You Were Here	Pink Floyd	Wyclef Jean
80	Yesterday	Beatles	En Vogue

Appendix B

The `my30` dataset

Artist A is also performing the live version.

Nr.	*Title*	*Artist A*	*Artist B*
1	Papa Don't Preach	Madonna	Kelly Osbourne
2	Personal Jesus	Depeche Mode	Marilyn Manson
3	Smells Like Teen Spirit	Nirvana	Tori Amos
4	Wonderwall	Oasis	Alex Goot
5	Umbrella	Rihanna	The Baseballs
6	Kids	MGMT	The Kooks
7	I Kissed A Girl	Katy Perry	William Fitzsimmons
8	Hey Ya	Outcast	Obadiah Parker
9	Nothing Else Matters	Metallica	Apocalyptica
10	Animal	Miike Snow	Juliana Richer Daily

Appendix C

Content of the CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

C.1 PDF-File

Path: /

Engelmayer_Christoph_2011.pdf Thesis (e.g. this document)

C.2 MATLAB files

Path: /Implementation/

calcListHPCP.m	Function to calculate HPCP features for a list of files
calculateHPCP.m	Function to generate HPCP features of an audio signal
calculateReferenceFrequency.m	Reference frequency estimation function
crosscorr.m	Cross-correlation function
dtw.m	Dynamic time warping function
extractPeakPoints.m	Function to extract the peak points
main.m	Main function to use the algorithm
peakdet.m	Peak detection function
plotSonogram.m	Function to plot the sonogram of the calculated features
removeTransient.m	Function to remove the transients of the audio signal
transposeHPCP.m	Function to transpose the calculated HPCP vector
windowfft.m	Windowing and FFT function

Path: /Implementation/include

coversongs/	The source code of the algorithm from [16]
dtw/	Functions for dynamic time warping
MIRtoolbox1.3.2/ . . .	The MIRtoolbox framework
mp3readwrite/	Functions to read and write MP3 files

Bibliography

[1] Adams, N.H., N.A. Bartsch, J.B. Shifrin, and G.H. Wakefield: *Time series alignment for music information retrieval.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 303–310, Barcelona, Oct. 2004. IEEE.

[2] Bartsch, N.A. and G.H. Wakefield: *To catch a chorus: Using chroma–based representations for audio thumbnailing.* In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 15–18, Gregory, Oct. 2001. IEEE.

[3] Bello, J.: *Audio–based cover song retrieval using approximate chord sequences: Testing shifts, gaps, swaps and beats.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 239–244, Vienna, Sept. 2007. IEEE.

[4] Bonada, J.: *Automatic technique in frequency domain for near–lossless time–scale modification of audio.* In *Proceedings of the International Computer Music Conference*, pp. 396–399, Berlin, Aug. 2000. International Computer Music Association.

[5] Casey, M., C. Rhodes, and M. Slaney: *Analysis of minimum distances in high–dimensional musical spaces.* IEEE Transactions on Audio, Speech and Language Processing, 16:1015–1028, July 2008.

[6] Casey, M. and M. Slaney: *The importance of sequences in musical similarity.* In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. V–5 – V–8, Toulouse, May 2006. IEEE.

[7] Casey, M., R. Veltkamp, M. Goto, M. Leman, C. Rhodes, and M. Slaney: *Content–based music information retrieval: Current directions and future challenges.* Proceedings of the IEEE, 96:668–696, Apr. 2008.

[8] Cooper, M. and J. Foote: *Automatic music summarization via similarity analysis.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 81–85, Barcelona, Oct. 2002. IEEE.

[9] Dalla Bella, S., I. Peretz, and N. Aronoff: *Time course of melody recognition: a gating paradigm study.* Perception & Psychophysics, 65(7):1019–1028, Oct. 2003.

[10] Dannenberg, R.B. and N. Hu: *Pattern discovery techniques for music audio.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 63–70, Paris, Sept. 2002. IEEE.

[11] De Cheveigne, A.: *Pitch perception models.* In Plack, C., A. Oxenham, and R. Fay (eds.): *Pitch: Neural coding and perception*, ch. 6. Springer, Heidelberg, 2005.

[12] De Cheveigne, A. and H. Kawahara: *Comparative evaluation of f0 estimation algorithms.* In *Proc. of the Eurospeech Conference*, pp. 2451–2454, Aalborg, Sept. 2001. International Speech Communication Association.

[13] Downie, J.S., M. Bay, A.F. Ehmann, and M.C. Jones: *Audio cover song identification: Mirex 2006–2007 results and analysis.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 468–473, Philadelphia, Sept. 2008. IEEE.

[14] Dressler, K.: *Sinusoidal extraction using an efficient implementation of a multi-resolution FFT.* In *Proc. of the International Conference on Digital Audio Effects*, pp. 247–252, Montreal, Sept. 2006. European research project for co-operation and scientific transfer.

[15] Egorov, A. and G. Linetsky: *Cover song identification with if-f0 pitch class profiles.* In *Music Information Retrieval Evaluation Exchange task on Audio Cover Song Identification*, Philadelphia, Sept. 2008. International Music Information Retrieval Systems Evaluation Laboratory.

[16] Ellis, D.P.W. and G.E. Poliner: *Identifying cover songs with chroma features and dynamic programming beat tracking.* In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp. IV–1429 – IV–1432, Honolulu, Apr. 2007. IEEE.

[17] Foote, J.: *Arthur: Retrieving orchestral music by long–term structure.* In *Proc. of the International Society for Music Information Retrieval Conference*, pp. 130–135, Plymouth, Oct. 2000. International Society for Music Information Retrieval.

[18] Fujishima, T.: *Realtime chord recognition of musical sound: a system using common lisp music.* In *Proc. of the International Computer Music Conference*, pp. 464–467, Beijing, Oct. 1999. International Computer Music Accociation.

[19] Gasser, M., A. Flexer, and D. Schnitzer: *Hubs and orphans – an explorative approach.* In *Proc. of the Sound and Music Computing Conference*, Barcelona, July 2010. Sound and Music Computing research community.

[20] Gómez, E.: *Tonal Description of Music Audio Signals.* PhD thesis, University Pompeu Fabra, Barcelona, July 2006.

[21] Gómez, E. and P. Herrera: *The song remains the same: identifying versions of the same song using tonal descriptors.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 180–185, Victoria, Oct. 2006. IEEE.

[22] Gómez, E., B.S. Ong, and P. Herrera: *Automatic tonal analysis from music summaries for version identification.* In *Proc. of the Audio Engineering Society Convention*, San Francisco, Oct. 2006. Audio Engineering Society.

[23] Goto, M.: *A real–time music–scene–description system: Predominant-f0 estimation for detecting melody and bass lines in real–world audio signals.* Speech Comun, 43:311–329, Sept. 2004.

[24] Goto, M.: *A chorus–section detection method for musical audio signals and its application to a music listening station.* IEEE Transactions on Audio, Speech and Language Processing, 14:1783–1794, Sept. 2006.

[25] H., P.: *Profiles of pitch classes – circularity of relative pitch and key: experiments, models, computational music analysis and perspectives.* PhD thesis, Technische Universität Berlin, Berlin, Aug. 2005.

[26] Hu, N., R.B. Dannenberg, and G. Tzanetakis: *Polyphonic audio matching and alignment for music retrieval.* In *Proc. of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 185–188, New Paltz, Oct. 2003. IEEE.

[27] Izmirli, Ö.: *Tonal similarity from audio using a template based attractor model.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 540–545, London, Sept. 2005. IEEE.

[28] Klapuri, A.: *Signal processing methods for the automatic transcription of music.* PhD thesis, Tampere University of Technology, Tampere, Mar. 2004.

[29] Kyogu: *Identifying cover songs from audio using harmonic representation.* In *Music Information Retrieval Evaluation Exchange task on Audio Cover Song Identification*, Illinois, Oct. 2006. International Music Information Retrieval Systems Evaluation Laboratory.

[30] Lartillot, O., P. Toiviainen, and T. Eerola: *A matlab toolbox for music information retrieval.* In Preisach, C., H. Burkhardt, L. Schmidt-Thieme, and R. Decker (eds.): *Data Analysis, Machine Learning and Applications*, ch. 4.5. Springer, Heidelberg, 2008.

[31] Lee, K.: *Identifying cover songs from audio using harmonic representation.* In *Music Information Retrieval Evaluation Exchange task on Audio Cover Song Identification*, Illinois, Oct. 2006. International Music Information Retrieval Systems Evaluation Laboratory.

[32] Leman, M.: *Music and schema theory: cognitive foundations of systematic musicology.* Springer, Heidelberg, 1st ed., 1995.

[33] Marolt, M.: *A mid–level melody–based representation for calculating audio similarity.* In *Proc. of the International Conference on Music Information Retrieval*, pp. 280–285, Victoria, Oct. 2006. IEEE.

[34] Marolt, M.: *A mid–level representation for melody–based retrieval in audio collections.* IEEE Transactions on Multimedia, 10(8):1617–1625, Dec. 2008.

[35] Mueller, M.: *Information Retrieval for Music and Motion.* Springer, Heidelberg, 1st ed., 2007.

[36] Nadeu, C.and Macho, D. and J. Hernando: *Time and frequency filtering of filter–bank energies for robust hmm speech recognition.* Speech Communication, 34:93–114, Apr. 2001.

[37] Nagano, H., K. Kashino, and H. Murase: *Fast music retrieval using polyphonic binary feature vectors.* In *Proc. of the International Conference Multimedia and Expo*, vol. 1, pp. 101–104, Lausanne, Aug. 2002. IEEE.

[38] Poliner, G.E., D.P.W. Ellis, A. Ehmann, E. Gómez, S. Streich, and B.S. Ong: *Melody transcription from music audio: Approaches and evaluation.* IEEE Trans. Audio, Speech, Lang. Process., 15:1247–1256, May 2007.

[39] Ravuri, S. and D.P.W. Ellis: *Cover song detection: from high scores to general classification.* In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 65–68, Dallas, Mar. 2010. IEEE.

[40] Röbel, A. and X. Rodet: *Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation.* In *Proc. of the International Conference on Digital Audio Effects*, pp. 30–35, Madrid, Sept. 2005. European research project for co-operation and scientific transfer.

[41] Sadie, S., J. Tyrrell, and B. Kernfeld: *The New Grove Dictionary of Music and Musicians*. Oxford University Press, Oxford, 2nd ed., 2005.

[42] Sailer, C. and K. Dressler: *Finding cover songs by melodic similarity*. In *Music Information Retrieval Evaluation Exchange*, pp. 38–40, Victoria, Sept. 2006. International Music Information Retrieval Systems Evaluation Laboratory.

[43] Schwarz, D. and X. Rodet: *Spectral envelope estimation and representation for sound analysis–synthesis*. In *Proc. of the International Computer Music Conference*, pp. 351–354, Beijing, Oct. 1999. International Computer Music Accociation.

[44] Selfridge-Field, E.: *Conceptual and representational issues in melodic comparison*. In Hewlett, W. and E. Selfridge-Field (eds.): *Melodic similarity: concepts, procedures and applications, Computing in Musicology*, ch. 1.1, pp. 3–64. The MIT Press, Cambridge, 1998.

[45] Serrà, J.: *Identification of versions of the same musical composition by processing audio descriptions*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Mar. 2011.

[46] Serrà, J., E. Gómez, and P. Herrera: *Audio cover song identification and similarity: Background, approaches, evaluation, and beyond*. In Ras, Z. and A. Wieczorkowska (eds.): *Advances in Music Information Retrieval*, ch. 14. Springer, Heidelberg, 2010.

[47] Serrà, J., E. Gómez, P. Herrera, and X. Serra: *Statistical analysis of chroma features in western music predicts human judgments of tonality*. Journal of New Music Research, 37:299–309, Dec. 2008.

[48] Serrà, J. and R.G. Serra, X.and Andrzejak: *Cross recurrence quantification for cover song identification*. New Journal of Physics, 11:093017, Sept. 2009.

[49] Serrà , J., E. Gómez, P. Herrera, and X. Serra: *Chroma binary similarity and local alignment applied to cover song identification*. IEEE Transactions on Audio, Speech and Language Processing, 16:1138–1151, Aug. 2008.

[50] Temperley, D.: *The Cognition of Basic Musical Structures*. The MIT Press, Cambridge, 1st ed., 2001.

[51] Tsai, W.H., H.M. Yu, and H.M. Wang: *A query–by–example technique for retrieving cover versions of popular songs with similar melodies*. In *Proc. of the International Conference on Music Information Retrieval*, pp. 183–190, London, Sept. 2005. IEEE.

[52] Tsai, W.H., H.M. Yu, and H.M. Wang: *Using the similarity of main melodies to identify cover versions of popular songs for music document retrieval*. Journal of Information Science and Engineering, 24(6):1669–1687, Mar. 2008.

[53] Wang, A.: *An industrial strength audio search algorithm*. In *Proc. of the International Conference on Music Information Retrieval*, pp. 7–13, (Baltimore, Maryland, USA), Oct. 2003. IEEE.

[54] Wang, A.: *The shazam music recognition service*. Communications of the ACM, 49:44–48, Aug. 2006.

[55] Waterman, M.S. and T.F. Smith: *Identification of common molecular subsequences*. Journal of Molecular Biology, 147:195–197, Mar. 1981.

[56] Yang, C.: *Music database retrieval based on spectral similarity*. Tech. Rep. 2001-14, Stanford InfoLab, Stanford, 2001. http://ilpubs.stanford.edu:8090/489/.