TECHNICAL UNIVERSITY OF CRETE

DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

DIPLOMA DISSERTATION

‘COVER SONG’ IDENTIFICATION

by

XIROUCHAKIS NIKOLAOS

COMMITTEE:

POTAMIANOS ALEXANDROS, Associate Professor (Supervisor)

DIGALAKIS VASSILIS, Professor

PETRAKIS EURIPIDES, Professor

CHANIA

OCTOBER 2011

Acknowledgments

I would like to show my gratitude to my supervisor who guided me towards a very interesting topic.

I would like to thank my parents for their continuous support.

Finally, I would like to thank all those who made their work and research freely available on the internet.

Abstract

With the vast increase in size of music databases, automatic content-based search methods are required for facilitating fast retrieval of music or information about music. Such search methods may be based on an audio-query tactic making the extraction and comparison of musical features and thus the analysis of musical similarity an interesting field of research. In this context, the present work focuses on describing a system which tries to identify cover versions of a query-song. Such songs are usually different in many musical attributes but can still be defined as 'similar'. Hence, investigating such a system may provide an analysis of which features are considered relevant for music similarity. To this end song representations need to be invariant to instrumentation and tempo. This is achieved by making use of 12-dimensional chroma features which are then averaged by the song's beat times. Comparison of songs is then carried out by cross-correlating the above representations taking into account all possible key transpositions as well as two possible global tempi. By testing this system on a music collection consisting of 80 songs with 2 versions of each,querying each one of them resulted in a 52.5% accuracy, whilst for a collection containing 15 pairs of song's a correct cover was found 10 times out of all cases. Such work does not only contribute in searching retrieving and organizing music collections but may also influence commercial copyright management of music licenses. While the present work mainly focuses on the task of automatic cover song identification it also makes a preliminary analysis of automatic beat tracking, automatic fingerprint matching as well as automatic chorus detection systems.

Περίληψη

Με την ραγδαία αύξηση του μεγέθους των μουσικών βάσεων δεδομένων απαιτούνται αυτόματες μέθοδοι αναζήτησης βασισμένες στο περιεχόμενο των μουσικών σημάτων για την γρήγορη ανάκτηση μουσικών κομματιών ή πληροφοριών για αυτά. Τέτοιες μέθοδοι αναζήτησης μπορεί να είναι βασισμένες σε μια τακτική όπου η είσοδος στο σύστημα να είναι ένα ηχητικό σήμα, με αποτέλεσμα η εξαγωγή και η σύγκριση μουσικών χαρακτηριστικών και γενικότερα η ανάλυση της μουσικής ομοιότητας να είναι ένα ενδιαφέρον ερευνητικό θέμα. Στο πλαίσιο αυτό, η παρούσα εργασία επικεντρώνεται στην περιγραφή ενός συστήματος που προσπαθεί να αναγνωρίσει διασκευές ενός τραγουδιού που έχει δοθεί ως είσοδος. Τέτοιες διασκευές διαφέρουν σε πολλά μουσικά χαρακτηριστικά, όμως διατηρούν μια ορισμένη ‘ομοιότητα’. Έτσι με τη διερεύνηση ενός τέτοιου συστήματος παρέχεται μια ανάλυση για το ποια χαρακτηριστικά του μουσικού σήματος θεωρούνται σημαντικά για την μέτρηση της μουσικής ομοιότητας. Για το σκοπό αυτό οι αναπαραστάσεις των τραγουδιών θα πρέπει να είναι αμετάβλητες στην ενοργάνωση και στο ρυθμό. Αυτό επιτυγχάνεται με τη χρήση 12-διάστατων χαρακτηριστικών chroma των οποίων στη συνέχεια υπολογίζεται ο μέσος όρος με τη χρήση των χρονικών στιγμών κάθε παλμού (beat) μέσα στο κομμάτι. Σύγκριση τραγουδιών γίνεται με την ετεροσυσχέτηση (crosscorrelation) των παραπάνω αναπαραστάσεων λαμβάνοντας υπόψη όλες τις πιθανές μεταθέσεις κλειδιού όπως και δύο ρυθμών, πιθανώς με διαφορετικό μετρικό επίπεδο. Η δοκιμή αυτού του συστήματος σε μια μουσική συλλογή που περιέχει 80 διαφορετικά τραγούδια εκ των οποίων το καθένα έχει δύο διαφορετικές εκδόσεις και κάνοντας αναζήτηση για όλες τις περιπτώσεις είχε ως αποτέλεσμα 52.5% επιτυχία. Για μουσική συλλογή 15 ζευγών τραγουδιών, το σύστημα έβρισκε τη σωστή διασκευή σε 10 από τις 15 περιπτώσεις κατά μέσο όρο. Μια τέτοια έρευνα δεν συνεισφέρει μόνο στην αναζήτηση, ανάκτηση και οργάνωση μουσικών συλλογών, αλλά μπορεί και να επηρεάσει την εμπορική διαχείριση πνευματικών δικαιωμάτων στη μουσική. Ενώ η παρούσα εργασία εστιάζει κυρίως στο έργο της αυτόματης αναγνώρισης μουσικών διασκευών, αναλύει και συστήματα όπως: χρονικός εντοπισμός παλμών-beat μέσα στο μουσικό σήμα, αυτόματη μουσική ταύτιση με χρήση audiofingerprint και αυτόματη εύρεση ρεφραίν.

Contents

List of figures

List of abbreviations

In order of appearance in the text:

ACS	Automatic Cover Song
MIR	Music Information Retrieval
QBE	Query By Example
QBH	Query By Humming
PCP	Pitch Class Profile
MIREX	Music Information Retrieval Evaluation eXchange
DTW	Dynamic Time Warp
DSP	Digital Signal Processing
DTFT	Discrete - Time Fourier Transform
IDTFT	Inverse Discrete – Time Fourier Transform
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
STFT	Short – Time Fourier Transform (or Short – Term Fourier Transform)
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
SSM	Self – Similarity Matrix
DP	Dynamic programming

ASA	American Standard Association
MFCC	Mel Frequency Cepstral Coefficients
BPM	Beas Per Minute
AFP	Audio Fingerprint
BPF	Band-pass filter
ACF	Autocorrelation Function
ACD	Automatic Chorus Detection
GUI	Graphical User Interface
SNR	Signal-to-Noise Ratio
MRR	Mean Reciprocal Rank

List of tables

To my friends and loved ones…

CHAPTER 1

Introduction

The present study examines the issue of *automatic cover song* (*ACS*) identification, a prominent subject of research in recent years. The term *cover song* implies any new rendition of a previously recorded song, including alternate versions, performances, recordings or other. Cover songs may vary in many attributes but usually are ‘similar’ to their originals keeping the essence of the same melody and/or harmony. The task is to identify cover songs within a musical database on the basis of similar musical features. This chapter provides motivation as to why, within the context of *music information retrieval* (*MIR*), ACS identification is a noteworthy study, to be followed by an overview of music similarity and the definition of a system's specificity. The chapter concludes with the study's objectives and a thesis outline presenting each chapter’s content and purpose.

1.1. Motivation

Even nowadays, digital music collections are typically browsed by combinations of textual metadata, such as the song's title, composer or performer. However, the larger the musical collection in a database the more complex it is to navigate it easily, and therefore the less efficient the above approach becomes. With the drastic evolution in technology people are faced with larger music collections every day. Let us take for example a common iPod which can carry up to 20.000 songs, or a storage device which can carry up to 250.000 songs, or even an online music store which has millions of songs available to their customers. Such large music databases raise new

interesting computational problems which need solving, such as direct access in such large collections and thus the need of algorithms for automatic search methodsand automatic music similarity estimations. These methods and estimations are based on musical features such as melody, harmony, rhythm and instrumentation, which describe the audio at a new level and help respond to requests such as: what is the name of the song that goes like <whistle the melody> (query by example); could you raise the volume of a particular instrument recording? (source separation); skip to the next chorus while listening to a music piece (musical structure identification); what instrument is this, or what is the name of this artist? (instrument/singer identification); and find recordings to a specified genre (genre classification) [1]. These methods could be used in a number of applications such as: automatic playlist generators; organization and visualization of large music collections; systems for recommendation of unknown pieces and/or artists to one's preferences; or even a system which helps out a DJ in choosing his next song so that the current and the next have similar rhythm and/or melody.

Another practical use of music similarity is the *Query-By-Example* (*QBE*) system, whereby the user has the possibility to search a music collection using audio as an input query, and not text. This can be very handy. Consider the case of a user who is browsing a large music collection for a particular song but is unable to remember neither the song's title nor the artist's name to use as textual metadata. But what if he does remember the melody or possesses a recorded part of that song? We may use music similarity in such a way that the user can load, sing, whistle or hum (in this case *Query-By-Hum*ming or *QBH*) the melody to the system and obtain a list of closest matches ranked by their similarity measure.

Identifying two different versions of a song requires consideration in variations of several musical dimensions. The main attributes that may vary between two cover songs are: its timbre, representing difference in instrumentation and production techniques; noise, presented e.g. during audience manifestations; key, when a song is transposed to a different key to adapt the pitch range to a particular singer; language of lyrics, when a song is translated; timing, depending on the performer's feeling; harmonization, which is common in an introduction or a bridge part in a song; and also tempo, structure, duration, or even genre. The feature which is robust enough to endure several of the above mentioned changes and will be used throughout this work

is the *chroma* or *chromagram*, otherwise known as *Pitch Class Profile* (*PCP*) of the signal, which will described in more detail in the present study.

Identifying a cover given its original as a query, or finding the original given a cover using only musical features, is quite a challenge which is researched by a community named *Music Information Retrieval Evaluation eXchange* (or *MIREX*). MIREX is a community-based endeavor to scientifically evaluate *music information retrieval* (MIR) algorithms and techniques [2] [3]. Such algorithms describe tasks which range from low-level, like audio onset detection, to higher-level ones, like audio music similarity. Some of these tasks are: audio classification tasks (i.e. audio artist identification), audio beat tracking, audio tempo estimation, audio key detection, audio music similarity and retrieval and among others, our task, audio cover song identification. The MIREX community put its interest into the ACS identification task, in order to distinct the sense of 'music similarity' (melody) with the sense of 'timbral similarity' (instrumentation). Measuring and using the degree of *music* similarity is what we are trying to achieve, making it an important musical feature and a worthwhile study. Suffice it to say that ACS identification systems have a direct implication to musical rights' management and licenses. In addition, MIR-systems have the ability to manage extremely large digital music databases in an efficient and reliable manner paving the way for future music-related industry developments. ACS identification systems may contribute to this cause.

1.2. Defining a system's specificity

Stemming from the above motivation, each and every MIR system should be specified with the kind of information it should retrieve. Content-based music retrieval is thus structured in a way whereby a system firstly defines a type of query, secondly what is to be considered a match, and finally the form of its output. The 'sense of match' is the characteristic which makes a system unique in searching for information of a certain degree of *specificity* [4] [5]. The specificity degree could either be *exact* or *approximate* depending on whether the music retrieval is performed with specific content or in a broader sense of music similarity respectively. Clearly a system which

finds an exact duplicate of an audio snippet and a system which retrieves all songs belonging to a particular genre class do not have the same specificity degree. An audio fingerprinting-based system (Section 3.1) is an example of a high-match (exact) specificity, whereas a genre-classification one [6] is an example of a low-match (approximate) specificity. A cover song identification system has an intermediate specificity degree since it is less exacting than audio-fingerprint duplicating,but is more specific than an genre-classification system. Such a system tries to approximate duplicate detection whilst allowing many musical facets to change and incorporates the important idea that cover songs retain their identity notwithstanding variations in several musical dimensions. This is what makes the creation of automatic cover song identification systems a complex task. The scale of specificity together with an enumeration of some MIR tasks is displayed in Fig. 1.1 [5].

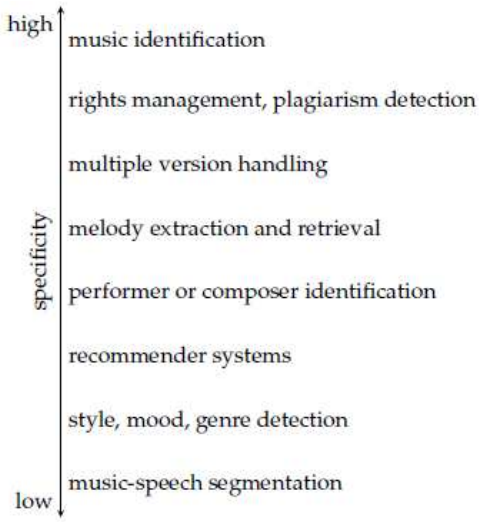


Figure 1.1: Examples of MIR Tasks and their 'specificities'

An overview of an ACS identification system can be 'broken' into five functional blocks as seen in Fig. 1.2 below. These blocks correspond to: a feature extraction block ensuring invariance of instrumentation and noise; a key invariance block which ensures invariance to transposition; a tempo invariance block; a structure invariance block and a similarity measure block which measures the final similarity between two songs. Tempo, key and structure changes are usually not handled by the features extracted from the signals and since these are common changes between cover songs they need to be processed individually.

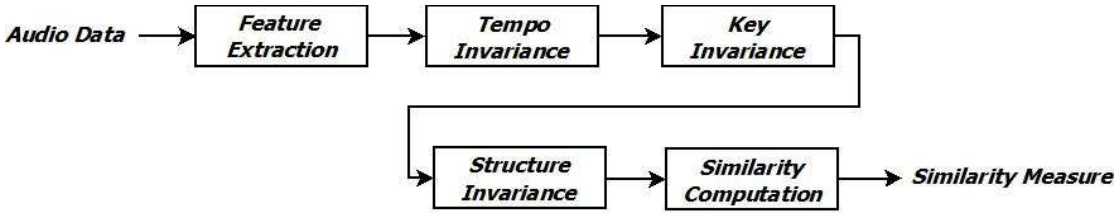


Figure 1.2: Overview of an ACS identification system

1.3. Related Work

Several approaches have been developed to attend to the cover song identification task. The different approaches of existing literature are referenced on the basis of the division displayed in Fig. 1.2 above, referring to each block separately [4]. Regarding the feature extraction block it should be pointed out that almost every ACS identification system makes use of *tonal sequence representation* (section 2.3), which either estimate the main melody or the chord or harmonic progression of raw audio signals. Examples of systems which make use of melody representations as main descriptors are [7] [8] [9] [10] and [11]. Such systems are usually based on a pitch tracking technique which tries to obtain the main melody of a song. Pitch tracking is a complex task if we are working with real-world polyphonic signals, since these present multiple pitches at each time point. Having an incorrect selection of the correct pitch to correspond to the main melody could result to an unreliable representation. Example systems which use harmonic representations with so called *pitch class profiles (PCP)*, or else named as *chroma*features are [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] and [26]. This representation is able to represent both monophonic and polyphonic signals and maintains an invariant stance in respect to several music characteristics. Hence the present study focuses on systems using the latter representation, namely those that use *chroma-representations*.

Four different methods stand out as being most effective for overcoming variability in tempo. First, finding the beats temporal locations in order to use them as an averaging factor for the descriptor information [14] [24]; second, applying temporal compression and expansion to the signal's representation [23] [24]; third, making use of the 2D power spectrum or the 2D autocorrelation function [8] [17] [18]; and last, by making use of dynamic programming aligning algorithms such as the *dynamic time warp* (*DTW*) and *edit distance variants* [10] [26] [25] [7] [16] [15] [12][27] [28]. In the present work we make use of the first method in order to examine Ellis' method of beat tracking [29].

Key invariance is handled is several ways: either by accounting all possible key transpositions in the final similarity measure; or by computing the most probable relative transpositions and accounting only them; or by estimating the *mainkey* and apply transposition accordingly; or by making use of 2D power spectrum or 2D autocorrelation function, an approach that achieves both invariance in tempo and key. For the purposes of this work we make use of the first method which consists of accounting all possible transpositions, on the grounds that it guarantees the best results at the cost of being slower.

Serra et al. [26] demonstrated the importance of structure invariance. A method used to accommodate this factor is summarizing a song through its most repeated or representative parts [7] [16]. Other methods consist of so-called *local alignment algorithms* [12] [25] [26] and *sequence windowing* [24] [23] [8] [22]. In this work it was found useful to include a presentation of the first method which considers finding the most repeated part of a song to be an attractive task in various applications, such as for music pre-listening stations whereby someone can automatically jump to the most representative part of a particular song. Therefore for the purposes of the present study research on musical structure detection limits to [30][31][32][33], which follow by and large the same method in order to extract an *audio thumbnail* (usually the chorus) of a song using stripe detection within a self similarity matrix (Section 2.3.2). Other methods try to segment a given song into its constituent parts, e.g. intro, verse, chorus, bridge and outro [34] [35]. A more thorough overview of the work done in the automatic music structure field is presented in [36].

Finally in each ACS identification system a similarity measure computation is required. The DTW algorithm used for tempo invariance already provides a similarity measure between two songs which is used by some systems as the conclusive similarity indicator [10] [11] [15] [16] and [28]. Alternative methods consist of using more conventional approaches such as the cross-correlation function [7] [13] [14], the Frobenius norm [17], the Euclidean distance [8] [18] or the dot product [19] [20] [22] [23]. In the present work we choose to use the method of cross-correlation, following Ellis' method [13] [14].

1.4. Objectives

The main objective of the present study is to investigate the existence of an automatic cover song identification system. For this purpose a chromagram-based approach has been followed. Background information on signal processing, some computational algorithms as well as on several musical attributes are also provided in order to introduce to the reader the basic concepts which are necessary for the understanding of the rest of the work. Consequently, four different MIR task systems are presented. The first two are automatic fingerprint (AFP) matching and ACS identification, which are both forms of QBE systems, each with a different degree of specificity. The other two systems are beat-tracking and automatic chorus detection (ACD), where each system is both an end in itself and a means to improve the performance of other systems such as the ACS identification one. Finally experimentation is performed to evaluate the systems' performance described throughout the present work, followed by a concluding discussion referring to topics such as appliance of music similarity systems in real-world and industry related applications as well as potential future work on the subject of the ACS identification task.

1.5. Thesis Outline

Chapter 2: Background information. This chapter introduces the main concepts which are necessary for understanding the work presented in the following sections. The information gathered and presented is based on the works of [37] [38] [39] [40] [43] [46] [47] [48] [49] and [50].

Chapter 3: Query-By-Example. This chapter describes the basic framework of query-by-example systems and analyzes more in detail systems using audio fingerprinting-based approaches. Consequently,two systems for audio matching are analyzed: one using a frame-based analysis and one using a landmark-based one.

Chapter 4:Automatic Cover Song (ACS) Identification Method. Here we present the core of this thesis, namely an automatic cover song identification system using a beat-synchronous chroma-based approach. The method used in this approach is divided into four processing steps: feature extraction, beat-times extraction, averaging of chroma and beat-times and matching. These steps are described and explained separately.

Chapter 5:Automatic Chorus Detection Method. In this chapter an automatic chorus detection method is described. We examine how repetitive parts within a song may be extracted using a self similarity matrix followed by a means to select the repetitive part which most likely corresponds to the chorus.

Chapter 6: Implementation. This chapter provides the experimentation made on automatic fingerprint matching, automatic cover song identification and automatic chorus detection systems. It presents the experimental datasets used for each system and makes an evaluation of the results.

Chapter 7: Conclusion and Future Work. Here, we discuss some conclusive remarks on this study, present possible applications in real-world situations and suggest some topics which could be thesubjects of future research.

CHAPTER 2

Background Information

2.1. Signal processing

In electrical engineering, *signal processing* is defined as the analysis, or handling of, signals, where signals are representations of time-varying or spatial-varying physical quantities. A signal can be used to carry a codified message and thus signals are used for storing, manipulating and transmitting information. Such information can be voice, music, images, video or other. In mathematical terms, we define a signal as the sequence of values of a quantity *y*, which changes according to a value of a quantity *x*. If *x* and *y* are defined on a continuous set of times, then the function *y(x)* is defined as a *continuous-time* or *analog* signal. If *x* is defined only on a discrete set of times (*n*), but *y* is defined on a continuous set of times, then the sequence *y[n]* is defined as a *discrete-time* signal. If *x*and *y* are both defined only on a discrete set of times, then the sequence *y[n]*, is defined as a *digital* signal. *Digital signal processing* (or *DSP*) and *analog signal processing* are subfields of signal processing. The aim of DSP, which includes audio signal processing as a subfield, is to use continuous real-world analog signals in a digitalized manner. To do so, an analog to digital conversion is necessary.

2.1.1. Analog to digital conversion

An analog signal can always be converted into a digital one so that it can be stored or processed by a digital computer. This is achieved by converting a continuous-time signal into a discrete-time one. This process is known as *sampling* and is realized by measuring the values of the continuous-time signal at discrete times, denoted as *samples*. Those times are periodically spaced every *T* seconds, and are given by the equation below, where *T* is the *sampling interval* and f_s is the *sampling frequency*, or *sampling rate*:

$$x(n) = x_a(nT) n = 0,1,2, \dots \qquad (2.1) \qquad f_s = \frac{1}{T} \qquad (2.2)$$

According to the *Nyquist-Shannon sampling theorem*, in order to be able to perfectly reconstruct the original signal *x(t)* from its samples *x(nT)*, two conditions must be met: on the one hand the signal must be *bandlimited*, meaning that the maximum value of the signal's frequency spectrum, f_{max} has a real value and is not infinite; and on the other that the sampling rate must be chosen to be at least twice that maximum frequency [37].

$$f_s \geq 2 * f_{max} \qquad (2.3)$$

$$T \leq \frac{1}{2 f_{max}} \qquad (2.4)$$

The minimum sampling rate given by the equation (2.3) above is called *Nyquist rate* and is equal to $f_s = 2 f_{max}$. The value $\frac{f_s}{2}$ is called *Nyquist frequency* or *folding frequency* and defines the endpoints of the $Nyquist\ interval = \left[-\frac{f_s}{2}, \frac{f_s}{2}\right]$. This theorem is useful because it provides a numerical value for the maximum frequency of a signal, in the present case a music song, given its sampling rate. If, however, the sampling frequency is selected to be smaller than *2fmax*, then there are not sufficient samples to capture all the variations within the signal, with the result that the representation of the samples is indistinguishable. The consequence is that without

previous knowledge of the original signal we could end up reconstructing a completely different signal than what would have been expected by using those samples. This effect is known as *aliasing.*An example is given in Fig. 2.1:

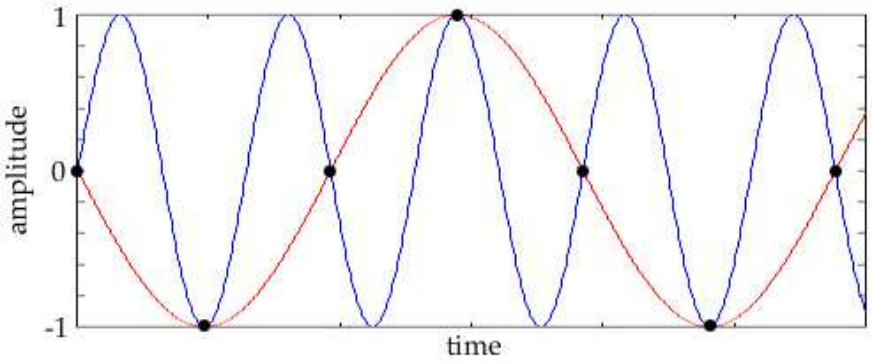


Figure 2.1: Effect of aliasing

Sometimes we have a discrete-time signal, but we require a version of that signal with fewer samples. The process of reducing the sampling rate of the signal is called *downsampling* or *subsampling* and can be achieved by keeping only every n^{th} sample, if we are downsampling by a factor of n.

2.1.2. The frequency domain

Frequency domain is a term used to describe the analysis' domain of signals with respect to frequency, rather than to time. The frequency domain's representation of a (time-domain's) signal is called *frequency spectrum* and carries information such as the amplitude and phase of the signal's frequency. A perfect reconstruction of the original signal is possible if the spectrum analyzed preserves both the amplitude and phase of each frequency component. In addition, a common technique in signal processing is to consider the squared amplitude (or power) of the frequency spectrum resulting in a *power spectrum*. It is also observable how much of the signal lies within each frequency band over a range of frequencies, which is an important observation for the purposes of the present study. Frequency units used in DSP are *Hertz* (cycles/sec), *cycles/sample*, *radians/sample* and *radians/sec*. We can obtain the

signal's numerical computation of its frequency spectrum by applying the *Fourier Transform* to the original signal. Since our work is using discrete-time signals, the *Discrete-Time Fourier Transform*, or *DTFT* of a discrete-time signal *x[n]*, is given by:

$$X(f) = \sum_{n=-\infty}^{\infty} x[n] e^{-j2\pi fTn} \tag{2.5}$$

where*X(f)* is computable only from the knowledge of the sample values *x[n]*. It is also a periodic function of *f*, with period $f=f_s$, $X(f+fs)=X(f)$, meaning that we get replicas of the spectrum with center each $f=nf_s$(where*n*=1,2,…) frequencies. We can restrict our spectrum to only one period, from $[-\frac{f_s}{2}, \frac{f_s}{2}]$. We can recover the original signal *x[n]* from *X(f)* by using the *Inverse Discrete-Time Fourier Transform (IDTFT)*:

$$x[n] = \frac{1}{f_s} \int_{-\frac{f_s}{2}}^{\frac{f_s}{2}} X(f)\, e^{j2\pi fTn} df \tag{2.6}$$

If *x[n]* is a sampled version of a continuous-time signal *x(t)*, then *X(f)* is an approximation of the frequency spectrum of the continuous-time signal *x(t)*. This is why the DTFT is often used to compute the frequency spectra of analog signals. Though we have the means to compute a frequency spectrum with a computer, it is still not possible, since *X(f)* requires infinite calculations resulting from an infinite number of samples, $-\infty < n < \infty$. To make it computable two other, practical approximations are made to *X(f)*. Firstly, only a finite set of samples are kept and computed, say $0 \leq n \leq L-1$, known as *time-windowing* and, secondly, *X(f)* is evaluated only on a finite set of frequencies. A DTFT of a length-*L* signal evaluated at *N* equally spaced frequencies between 0 and f_s is called a*N*-point *Discrete Fourier Transform* or *DFT*. A faster implementation for the *N*-point DFT of a signal is the *Fast Fourier Transform* (*FFT*). The FFT of a signal has the exact output as its DFT, but its algorithm is based on a divide-and-conquer approach which saves a lot of computational time. Maximum efficiency of the FFT computation is gained when its number of points is selected to be any positive integer number of two, e.g. 1024, 2048, 4096. In addition, the first half of the frequency range (from 0 to the Nyquist frequency $f_s/2$) is sufficient to identify the component frequencies in the data, since the second half is a reflection of the first half.

Since we are dealing with music signals, we should point out a particular scale of frequency bands, called the *mel-scale*. The name 'mel' is derived from the word melody indicating that the scale is based on pitch comparisons. The mel-scale maps each *Hertz*-frequency to a *mel*-frequency, expressing what a human's ear perceives to be an equal pitch interval. The relationship between frequency in *Hertz* and frequency in *mel* is given by the equationsbelow, as shown in Fig. 2.2:

$$f_{mel} = 2595 \log_{10}\left(1 + \frac{f_{Hz}}{700}\right) \tag{2.7}$$

$$f_{Hz} = 700(10^{\frac{f_{mel}}{2595}} - 1) \tag{2.8}$$

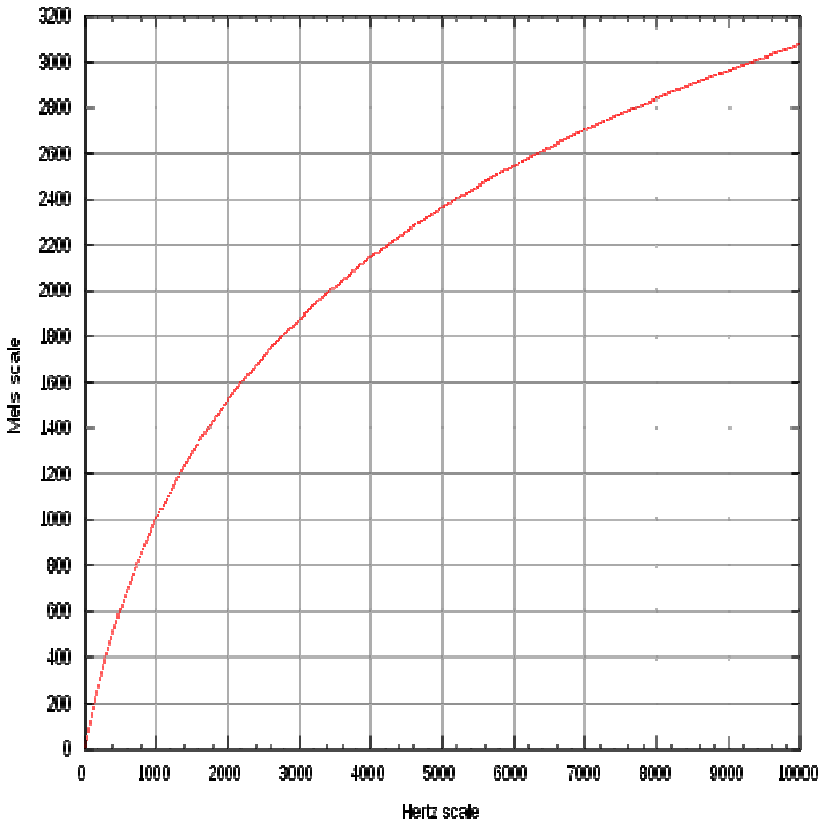


Figure 2.2: The relationship between Mel and Hertz scale

An examination of the relationship between *Hertz*-frequency and *mel*-frequency reveals that it approximates linearity below 1 KHz and exhibits a logarithmic spacing beyond that threshold, which reflects the logarithmic nature of sound as it is perceived by human ear.

2.1.3. Windowing and the time-frequency domain

It is often useful to have a representation of the signal's frequency (or energy) over time. This is derived by computing the *Short-Term Fourier Transform* (or *STFT*) of the signal. The process by which a STFT of a signal is computed is through 'cutting' (splitting) the sampled signal into time segments (frames), then calculating each segment's frequency spectrum by using either the DFT or FFT and, finally, appending the resulting spectra to build the signal's frequency over time representation. If, then, we square the magnitude of the STFT we obtain the *spectrogram* function, a common time-varying spectral representation. Such representations are often useful for signals whose characteristics change quickly over time, as i.e. music signals, and are to be displayed or processed in both time and frequency.

The process of cutting the sampled signal into segments of, say, L samples, is known as *windowing* and is achieved by multiplying our sampled signal with a series of shifted windows. The simplest window is a *rectangular window* and is defined as:

$$w(n) = \begin{cases} 1, & if\ 0 \le n \le L-1 \\ 0, & otherwise \end{cases}$$

When the window is multiplied to the sampled signal it becomes:

$$x_L(n) = x(n)w(n) = \begin{cases} x(n), & if\ 0 \le n \le L-1 \\ 0, & elsewhere \end{cases}$$

An example of which is presented in Fig. 2.3 below:

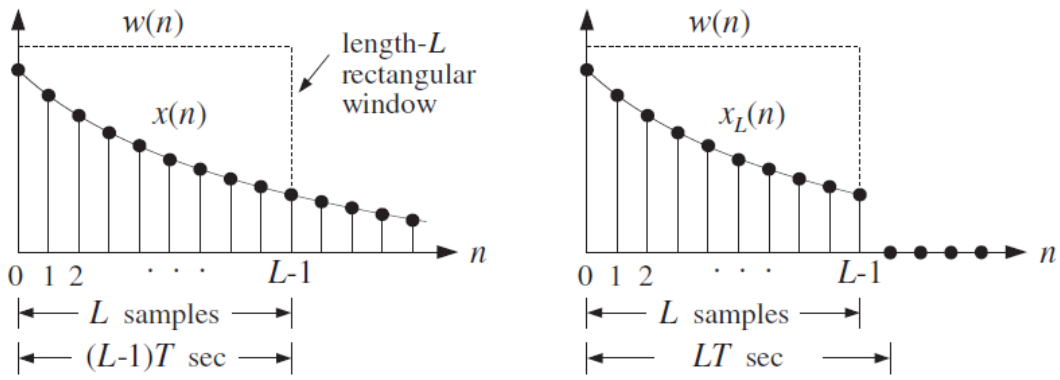


Figure 2.3: Example of windowing process

Two major effects occur with the use of time-windowing when considering the frequency domain. Firstly, the frequency resolution is reduced as a direct consequence of the "uncertainty principle" and, secondly, spurious high-frequency components are produced into the spectrum, a phenomenon known as *'frequency leakage'*. Concerning the first and in the context of signal processing, the uncertainty principle indicates that a function cannot be both time-limited and band-limited. Alternatively stated, one cannot achieve high temporal resolution and frequency resolution at the same time because the window size has a direct impact on both. A wide window achieves high frequency resolution at the cost of temporal resolution, while a narrow window has the opposite trade-off effect. Hence, the reduction in frequency resolution is a consequence of the change of the signal's duration, since after windowing the minimum resolvable frequency difference becomes $\Delta f = \frac{1}{LT}$ as the new signal duration is $T_L = LT$. Concerning the second effect, the spurious high-frequency components are produced as a consequence of the sharp ends of the signal caused by the (rectangular) window. The presence of these components becomes clear by examining the magnitude spectrum of *w(n),* $|W(\omega)|$,as shown in Fig. 2.4 below.The figure shows the existence of several smaller side-lobes in addition to the main-lobe.

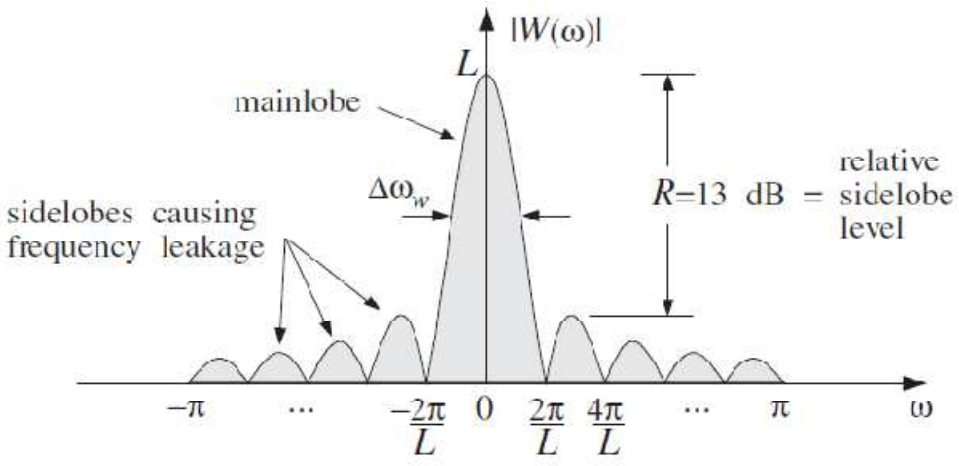


Figure 2.4: Magnitude spectrum $|\boldsymbol{W(\omega)}|$ of rectangular window

The main-lobe width determines the achievable frequency resolution. The side-lobes are an undesirable artifact of the windowing process referred to above as frequency leakage. They are to be suppressed as much as possible, as they can be confused with main-lobes of weaker sinusoids. To avoid such phenomena we can use other non-rectangular windows, such as the *Hamming window*, which is defined as:

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right), & for\ 0 \le n \le L-1 \\ 0, & elsewhere \end{cases} \quad (2.9)$$

As we can see in Fig. 2.5, the use of a hamming window suppresses the side-lobes by cutting off to zero less sharply than the rectangular one. The trade-off is that the main lobe becomes wider and shorter, reducing the frequency resolution capability. In the present work we are using a *hann-window*, otherwise known as *raised cosine window,* which is quite similar to the Hamming-window and is given by the equation below:

$$w(n) = \begin{cases} 0.5\left(1 - \cos\left(\frac{2\pi n}{L-1}\right)\right), & for\ 0 \le n \le L-1 \\ 0, & elsewhere \end{cases} \quad (2.10)$$

An additionalaspect to consider when windowing is that if we append windows one after the other there is a risk that some of the signal's information could be excluded. This occurs in cases where the signal's information appears when a window is decaying.This can be seen in Fig. 2.6. To overcome this issue, instead of appending the windows we shift them in a way that they overlap with each other so that no information is omitted.

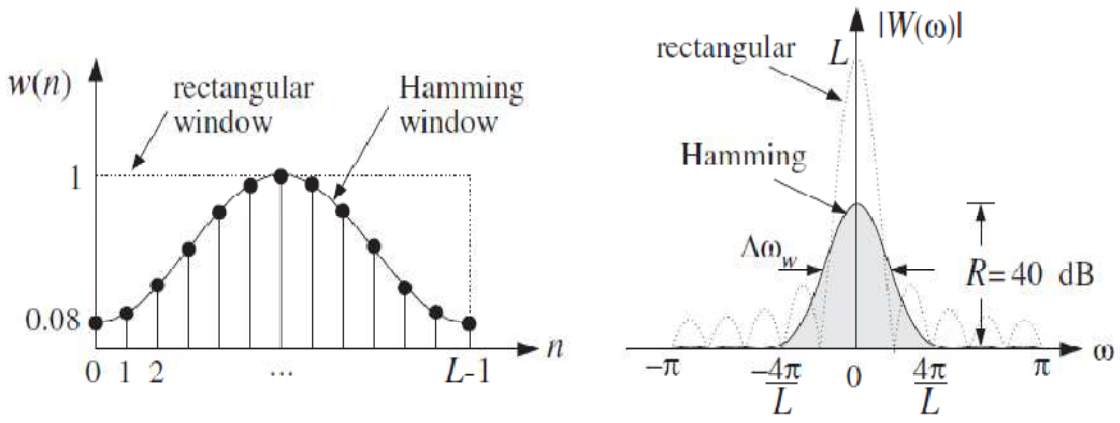


Figure 2.5: Differences between rectangular and Hamming window in time and frequency

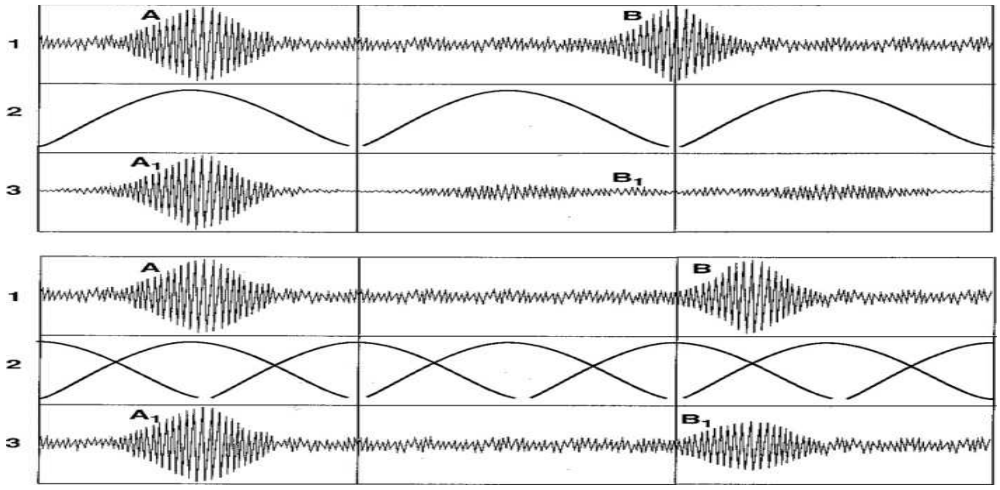


Figure 2.6: Example of the use of overlapping windows. In the above figure the information of the signal (B) is not included (B_1)

2.1.4. Filters

The process of removing a signal's unwanted components is known as *filtering*. Usually this means the removal of certain frequencies and the non-removal of others, with a view to suppressing interfering signals or reducing background noise. Digital filters can be classified into *FIR*-filters and *IIR*-filters based on their *impulse response.* An impulse response sequence *h(n)* (Fig. 2.7), is defined as the response of a system to a *unit impulse δ(n)*, where the latter is defined as the discrete-time analog of the *Dirac delta functionδ(τ)* and is defined as:

$$\delta(n) = \begin{cases} 1 \ , & if \ \ n = 0 \\ 0 \ , & if \ \ n \neq 0 \end{cases} \tag{2.11}$$

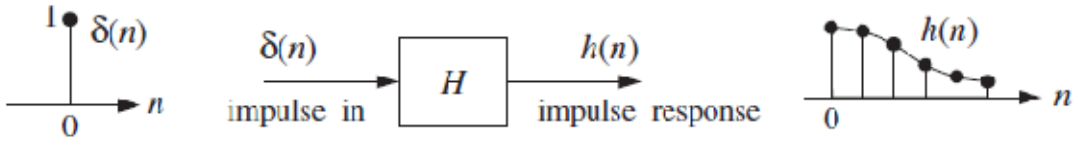


Figure 2.7: Impulse response sequence of a system

Consequently, if the impulse response is finite meaning that it extends only over a finite time interval, say $0 \leq n \leq M$, and it is equal to zero beyond that, then it is the case of a *finite impulse response*, or *FIR*-filter. Alternatively, the impulse response is infinite in duration and it is the case of an *infinite impulse response*, or *IIR* filter (Fig. 2.8).

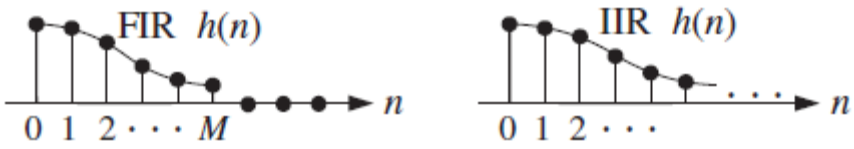


Figure 2.8: The impulse response of a FIR (left) and an IIR (right) filter

For the purposes of the present work only FIR-filters will be used. For a discrete-time FIR-filter the I/O equation is obtained as a weighted sum of the present input sample *x(n)* and the past *M* samples:

$$y(n) = \sum_{m=0}^{M} h(m)x(n-m) \tag{2.12}$$

where *x(n)* is the input signal, *y(n)* is the output signal, h_i are the *filter coefficients* (otherwise known as *filter taps* or *filter weights*) and *M* is the order of the filter having *M*+1 terms on the right-hand side. For example, a 3rd order *FIR*-filter is described by its four filter coefficients $\boldsymbol{h} = [h_0, h_1, h_2, h_3]$ and is equal to:

$$y(n) = h_0x(n) + h_1x(n-1) + h_2x(n-2) + h_3x(n-3)$$

Digital filters are further divided into several categories, some of which are described and displayed below (Fig.2.9):

- Low-pass filters: where low frequencies are passed and high frequencies are suppressed
- High-pass filters: where low frequencies are cut-off and high frequencies are passed
- Band-pass filters: where only a specified range of frequencies is passed and the rest is cut off
- Band-stop filters: where only frequencies in a specified range are cut-off and the rest is passed.

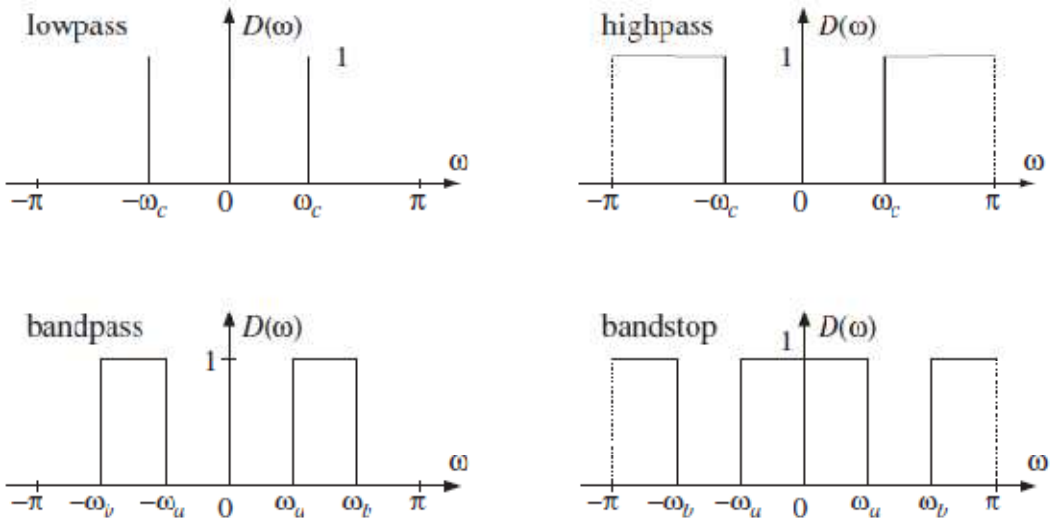


Figure 2.9: Ideal lowpass, highpass, bandpass and bandstop filters

2.1.5. Cross-correlation

Cross-correlation is a method used to measure similarity between two signals. One signal is shifted in time and is then multiplied by the other signal resulting in a function which displays similarity over lag. Cross-correlation is often used when a short signal is to be found within a longer one. The values of the cross-correlation at the lags at which the shorter signal has a good alignment with the longer one will result into high peaks which denote good correlation between the two signals at this particular lag. The output values for each lag are normalized to range from -1 to 1, so that the closer the cross-correlation is to 1, the more similarity there is between the two signals.

The cross-correlation between two discrete-time signals *x(n)* and *y(n)* is a sequence $R_{xy}(l)$ and is given by:

$$R_{xy}(l) = \sum_{n=-\infty}^{\infty} x(n)y(n-l) \qquad l = 0, \pm 1, \pm 2, \ldots \qquad (2.13)$$

It is noted that the nature of cross-correlation is similar to the one of *convolution*. Whereas convolution involves reversing a signal, then shifting it and multiplying by another signal, correlation only involves shifting it and multiplying (no reversing).

A special case of cross-correlation is *auto-correlation* whereby a signal is cross-correlated with itself. Obviously the autocorrelation of a signal will have a maximum peak at the lag equal to zero. A periodic signal will have maxima in its autocorrelation at lags which correspond to the period of the signal. Autocorrelation is often used to find repeated patterns within a signal. The auto-correlation of a discrete signal *x(n)* is given by [38]:

$$R_{xx}(l) = \sum_{n=-\infty}^{\infty} x(n)x(n-l) \qquad l = 0, \pm 1, \pm 2, \ldots \qquad (2.14)$$

2.1.6. Self-similarity matrix

The *self-similarity matrix (SSM)* is the opposite concept to the *distance matrix*. For the purposes of the present work the distance is set to be the *Euclidian distance*. The similarity matrix of a length *N* signal is a symmetric $N * N$ matrix, containing elements which display the similarity of the points in the signal to each other, taken pair wise. The greater the similarity between two points in the signal, the greater the numerical value of the point in the similarity matrix corresponding to that particular pair. There are five characteristics which describe a similarity matrix *M*. 1) *M* must have the same number of rows and columns, 2) all elements in *M* must be real, non-negative numbers, 3) all elements in *M* must have values between 0 and 1, 4) all elements that are located on the main diagonal must all have the value 1, and 5) each element located at the point *(i,j)* must have the same value with the element found at the point *(j,i)*. If any of these characteristics is not met, then it is not the case of a SSM. The fact that the SSM is symmetric implies that all information is calculated twice, once for each side of the diagonal. For practical reasons, implementations of the SSM usually calculate only one side of the diagonal resulting in a triangular matrix.

The similarity matrix described above has both axes representing absolute time. A different visualization of the similarity matrix arises when one of the axes is changed to correspond to a time-lag rather than time itself. Thus one of the axes' indexing is relative to the other. The resulting matrix is a *time-lag* similarity matrix $L(l, t)$ and is derived from the *time-time* similarity matrix $S(t_1, t_2)$ using the equation below:

$$L(l, t) = S(v_t, v_{t-l}) \tag{2.15}$$

The time-time SSM and the time-lag SSM are presented in figure 2.10.

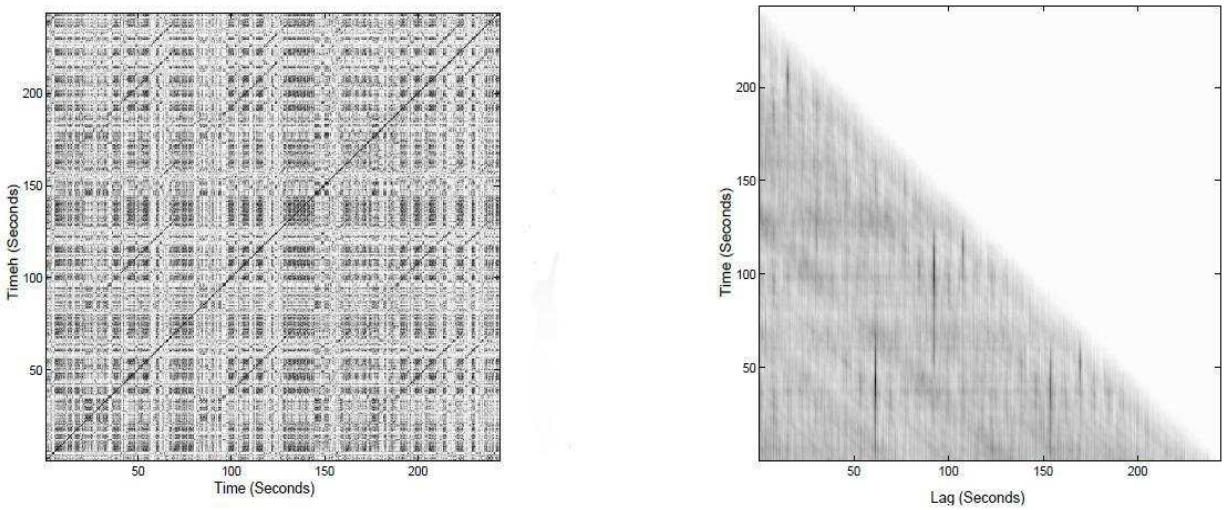


Figure 2.10: The time-time SSM (left) and the time-lag SSM (right) of an example song

2.2. Dynamic programming

The term “programming” does not refer to writing code for computer programs, but rather it relates to the word “planning” and to the description of a set of rules which anyone can follow in order to solve a problem. In mathematics and computer science *dynamic programming* is a method for solving problems much like the *divide-and-conquer* method. These types of methods break down the problem into smaller, simpler sub-problems, solve the latter and combine the solutions of the sub-problems to reach an overall solution to the initial problem. In cases where sub-problems overlap, a divide-and-conquer algorithm would solve every sub-problem regardless if they are the same, whereas a dynamic programming algorithm would solve each sub-problem only once, and then store its solution for future use. This results in a significant computation number reduction, especially if the number of repeating sub-problems is exponentially large. Dynamic programming is usually used to solve a broad range of search and optimization problems which exhibit the characteristics of overlapping sub-problems and optimal sub-structure. Such an optimal sub-structure means that an optimal solution of a problem can be derived from the optimal solutions of its sub-problems. *Top-down* dynamic programming means storing the results of certain calculations which are later used again since the completed calculation is a sub-problem of a larger calculation. *Bottom-up* dynamic programming involves formulating a complex calculation as a recursive series of simpler calculations [39] [40].

A dynamic programming algorithm can be analyzed into four steps:

1. Characterize structure of an optimal solution
2. Define value of optimal solution recursively
3. Compute optimal solution values either *top-down* or *bottom-up*
4. Construct an optimal solution from computed values.

2.3. Music feature extraction

A feature is a characteristic part of something. It is a way to describe the nature and properties of a subject making it possible to decide if two subjects are similar or not, and/or to assemble subjects with similar attributes into particular groups. When dealing with music, measurable content-based features that describe the audio signal are known as *acoustic features*. Many acoustic features have been used in different music information retrieval systems with the aim of measuring the similarity between two songs. Acoustical features could be classified in 3 categories: rhythmic content features, timbral textural features and pitch content features. Rhythmic content features describe the speed of the music signal in time. They try to catch the rhythm of a song, as a human would try to catch it by tapping with his hand or foot. If we think about it, the rhythm is also a good feature to describe the mood of a song, since for a low speed we have a relaxing song, and for a faster speed we have a more uplifting song. The tempo and the beat of a signal which are features of this category are described in section 2.3.3. Timbral textural features are used to distinct similar sounds. Timbre, or 'tone color', is what makes a sound unique. As quoted by [41], page 49, the American Standards Association (ASA) defines timbre as "that attribute of sensation in terms of which a listener can judge that two sounds having the same loudness and pitch are dissimilar [42]." For example, if two different instruments play the same note we can still understand which sound came from which instrument. The disadvantage of timbral features is that they do not include the melody and harmony of the signal in their representations. Known timbral features are the *spectral centroid*, the *spectral rolloff*, the *spectral flux*, the *zero-crossings*, the *low-energy* and the widely used *mel-frequency cepstral coefficient* or else known as *MFCC* [43]. MFCCs are short-term spectral-based features commonly used for state-of-the-art speech recognition systems. Since the present work will not get more into detail regarding this particular musical feature, more information can be found in [44] [45]. Finally, the pitch content features are a means to keep track of the melody and/or harmony of a musical signal. These features capture sound as a pitch regardless of where it came from, making them as a result unsuitable for timbral analysis.Such pitch content

features are, for example, tonal sequence representations which can be understood, in a broad sense, as a sequentially-played series of different note combinations. These notes can be either unique for each time slot, representing the main melody of a song, or they can be played jointly with others, expressing in this way the chord or the harmonic progressions.

2.3.1. Pitch and musical notes

Musical sound signals can be characterized by their pitch. The pitch perceived by a listener is his ear's response to the sound's frequency. Thus pitch is directly dependant on the sound's frequency, though it is not to be confused with the scientific term of frequency itself. Pitch is measured in *Hertz(Hz)* and defines how high or low a musical tone is.

In music, only a selected number of pitches are ever played and thus notations for these particular pitches and their duration are represented by *notes*. The range of pitches is divided into *octaves*, in a way so that if a note's frequency value is double to another note's frequency, then these two notes are separated by one octave. This is valid for any ratio of power of two between their frequencies. For example, if three notes, say *N1, N2 and N3*, are equal to $N1 = x, N2 = 2x, N3 = 4x$, then *N1* and *N2* are separated by one octave and *N1* and *N3* by two. An octave is further divided into twelve *semitones* which correspond to the different notes (pitch values). Thus a semitone (or *half-step interval*) can be defined as the distance between each successive white and black piano key, and its frequency difference is equal to $2^{(1/12)}$ times the initial frequency.

The twelve distinct notes in an octave are generally depicted by the following symbols: C, C#, D, D#, E, F, F#, G, G#, A, A# and B. The result is a set of 12 pitches, named individually, which repeat themselves for each upper or lower octave. In table 2.1 in page 28we can see the first nine octaves (starting from A=27.5 Hz) with their 12 distinct notes and their frequency values.

2.3.2. Pitch class – chroma

On the basis of the above information it is possible, given an octave number and a note's name (i.e. the note A in the 4th octave, or simpler, A4), to 'play' the desired pitch (440Hz). The fact that the note's names repeat for each octave helps us create a new term called the *pitch class*, or else named *chroma*. The pitch class (chroma) of a note's name is simply a set of all pitches corresponding to this note's name, independent of its octave. For example the pitch class of A corresponds to all the pitches of A0, A1, A2 etc. From this observation derives the *Shepard's pitch helix* [46] which is a model that maps all the pitches based on the following two criteria: firstly the chroma, which indicates the cyclical position of the desired pitch on the model (helix); and secondly the height, indicating the vertical height in the model (helix), which is measured in octaves. Fig. 2.11 shows the above mentioned model.

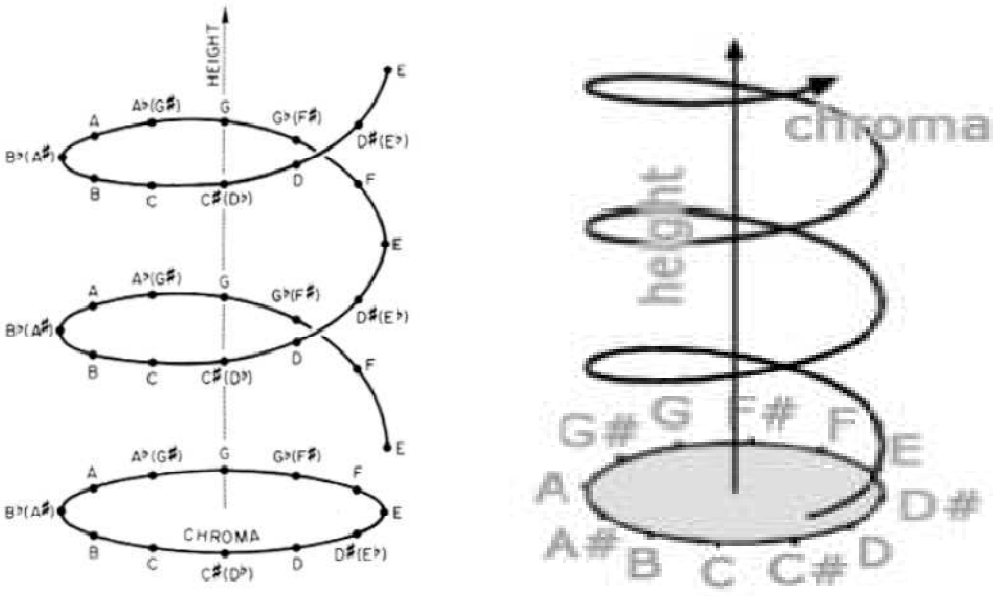


Figure 2.11: Shepard's pitch helix

Shepard gave birth to a logarithmic unit of measure for music intervals, called *cents*. He divided the interval between two semitones in 100 cents, and the interval between two octaves in 1200 cents respectively. The distance *n,* in cents, between a note*a* and a note *b* given in *Hertz* is provided by the following equation:

$$F_{cent} = 1200 \log_2 \left(\frac{b}{a}\right) \tag{2.16}$$

And given a note *a* and the cents of the interval between *a* and *b*, then *b* is provided by:

$$b = a * 2^{\frac{F_{cent}}{1200}} \tag{2.17}$$

Thus, if we center our helix so that its first note is a=A0=27.5 Hz, then we are able to measure each note's distance from the first note (A0) in *cents*. The equations (2.16) and (2.17) thus become:

$$F_{cent}(F_{Hz}) = 1200 \log_2 \frac{F_{Hz}}{27.5} \tag{2.18}$$

and

$$F_{Hz} = 27.5 * 2^{\frac{F_{cent}}{1200}} \tag{2.19}$$

Shepard's equation (2.20) below is able to map any note's distance (in *cents*) from the initial note by using variables *c* and *h*, where *c* stands for the 12 distinct pitch classes and takes values from 1 to 12, and *h*corresponds to the height and takes positive integer values starting from 0.

$$F_{Shepard}(c, h) = 1200h + 100(c - 1) \tag{2.20}$$

The tables below indicate in *Hertz* and in *cents* the 12 notes for the nine first octaves.

c\h	0	1	2	3	4	5	6	7	8
A	27.5	55	110	220	440	880	1760	3520	7040
A#	29.1	58.3	116.5	223.1	466.16	932.33	1864.66	3729.31	7458.6
B	30.9	61.7	123.5	246+.9	493.88	987.77	1975.53	3951.07	7902.1
C	32.7	65.4	130.8	261.6	523.25	1046.5	2093	4186.01	8372
C#	34.6	69.3	138.6	277.2	554.37	1108.73	2217.46	4434.92	8869.8
D	36.7	73.4	146.8	293.66	587.33	1174.66	2349.32	4698.64	9397.3
D#	38.9	77.8	155.6	311.13	622.25	1244.51	2489.02	4978.03	9956.1
E	41.2	82.4	164.8	329.63	659.26	1318.51	2637.02	5274	10548.1
F	43.7	87.3	174.6	349.23	698.46	1396.91	2793.83	5587.6	11175.3
F#	46.2	92.5	185	369.99	739.99	1479.98	2959.96	5919.9	11839.8
G	49	98	196	392	783.99	1567.98	3135.96	6271.9	12543.8
G#	51.9	103.8	207.7	415.3	830.61	1661.22	3322.44	6644.9	13289.7

Table 2.1 Notes' frequencies over nine octaves in *Hz*

c\h	0	1	2	3	4	5	6	7	8
1(A)	0	1200	2400	3600	4800	6000	7200	8400	9600
2(A#)	100	1300	2500	3700	4900	6100	7300	8500	9700
3(B)	200	1400	2600	3800	5000	6200	7400	8600	9800
4(C)	300	1500	2700	3900	5100	6300	7500	8700	9900
5(C#)	400	1600	2800	4000	5200	6400	7600	8800	10000
6(D)	500	1700	2900	4100	5300	6500	7700	8900	10100
7(D#)	600	1800	3000	4200	5400	6600	7800	9000	10200
8(E)	700	1900	3100	4300	5500	6700	7900	9100	10300
9(F)	800	2000	3200	4400	5600	6800	8000	9200	10400
10(F#)	900	2100	3300	4500	5700	6900	8100	9300	10500
11(G)	1000	2200	3400	4600	5800	7000	8200	9400	10600
12(G#)	1100	2300	3500	4700	5900	7100	8300	9500	10700

Table 2.2: Notes' distances in *cents* in relation to A0 over nine octaves

2.3.3. Tempo and beat

Tempo and beats are two important attributes of an audio signal which determine the rhythmic content of a music piece, and is thus a relevant task to create techniques such as tempo estimation and beat tracking systems in order to be able to measure and use those features. A number of applications are based on these features such as video editing, audio editing, stage lightning control and many others.A beat is an event occurrence or a wave pulse driven usually by instruments which operate in lower frequencies (drums, base) and is considered to be a base time unit for songs. A metronome is a tool that keeps track of this time units and by listening to it we perceive regular ticks, where each tick corresponds to a beat. Humans perceive beats as binary regular pulses determining the rhythm of a song. By calculating the number of beats at a given time interval we are able to determine its *tempo*, or in other words the global speed of playing a piece of music. Tempo is typically measured in *beats per minute*, or *BPM*. When describing musical tempo for an audio signal, it is important to differentiate between two different meanings of tempo, *'notated'* tempo and *'perceptual'*tempo [47] [48]. Notated tempo is the tempo derived from the music's notation in the music score, and it is a mark on the top of the general music staff which has a single value over the whole song; perceptual tempo is the tempo which is perceived by a listener unfamiliar with the notated tempo and may exist at different metrical levels during the song.It is of interest to create a representation of the perceptual tempo rather than the notated tempo, especially in cases where the latter is unknown (as is in our case). This is because for many applications perceived tempo is considered to be a more relevant feature than the notated tempo, since it captures the 'feel' of the music.The perceptual tempo is sometimes equal to the notated tempo, but not always, since a musical piece may 'feel' faster or slower than its notated tempo. This happens if the perceived tempo is a metrical level higher or lower than the notated tempo.

But how do we capture the perceived tempo? Having listeners, including expert musicians, tap along to music excerpts is a good way to annotate the perceived tempo, but there are some drawbacks. Not all people tap exactly the same way, because

different people might have different interpretations for what they are ‘feeling’ to be the perceptual tempo and therefore are tapping at different metrical levels. For some excerpts, the perceptual tempo is less ambiguous and listeners tap at the same metrical level, whilst for other excerpts the tempo can be quite ambiguous and listeners tap at different metrical levels resulting in a complete split across the listener’s tapping. Thus an explicit answer cannot be obtained, though previous studies have shown that listeners tend to tap at tempi near a ‘resonance’ of around 120 *BPM* [49] [50].

For an audio signal’s beat analysis we first estimate the tempo and then we try to determine the temporal position of each individual beat. Such information may be useful as melody and chord changes of a music piece tend to coincide with the beat times.The exact method used in this work is described in section 4.2.

CHAPTER 3

Query-By-Example

In this chapter a basic *Query-By-Example (QBE)* system is described. As already mentioned in the introductory chapter, a QBE system provides the means to a user to navigate a large music database by using audio inputs as queries. It is understandable how such a system may be useful to a large number of applications. A typical scenario of such a system's use in an application would be a user who wishes to retrieve information on a piece of music that is playing at that time by recording a portion of that music with his cell phone and querying a database with the recorded data in order to obtain information concerning that particular song. It is noted that this recording could be very noisy, proportional to the noisy environment it was recorded from and could be encoded in a very poor form (lossy cellular phone encoding as GSM 6.10 compression). Theoretically, a query in a QBE system could be either a recording of a cover-version of a song, or a user-sung melody, or a possibly degraded portion of the desired recording itself. The present chapter considers the latter case whereby we would like to find an exact instance of the given query inside a database, reflecting the example of the application mentioned above. A naive approach would be to store the whole query and compare it with each entry in the database on a sample-by-sample basis. Obviously this is not an effective method, since it would be very expensive to search and even more expensive to keep as the database would be very large, considering music pieces with an average duration time of 4 minutes each and a sampling rate of 44.1KHz. Instead, an *audio-fingerprinting* approach is used for this purpose and the above expensive drawbacks cease to exist.

3.1.Audio fingerprinting

An *audio fingerprint* (AFP) is a compact set of features derived from the signal that uniquely identifies the signal [1]. This representation uses a much smaller amount of data and can therefore be used and stored instead of the whole signal. The prime aim of multimedia fingerprinting is to develop an efficient mechanism for establishing the perceptual equality of two multimedia objects by comparing the associated fingerprints (small by design), rather than by comparing the (typically large) objects themselves [51]. This is a reminder of the hash function known in cryptography, whereby a usually large object is mapped to a smaller object using a hash function. The larger object can then be retrieved by searching the smaller one. This similarity in the systems is the reason why audio fingerprinting is also referred to as *robust or perceptual hashing*. One could ask why we are not using hash functions for audio fingerprinting. It must be pointed out that we are not interested in mathematical equality of two pieces of songs but rather to their perceptual similarity. For example, an original CD quality of a particular song and its MP3 version may sound the same (perceptual similarity) but mathematically they are quite different.

Our system is therefore transformed into a system where each audio file in the database is represented by its AFP, forming a *fingerprint database*. By querying an unknown audio sample in the system we may subsequently compute its AFP in the same way. The fingerprint from the unknown audio sample is then matched against the fingerprints found in the fingerprint database. Thus, a fingerprint-based system consists of two methods: first, of a fingerprint extraction one and, second, a method that efficiently searches for matches of a fingerprint in a fingerprint database. This system presents the following advantages: less memory needed for storing the database as the fingerprints are relatively small; more efficient searching since the dataset to be searched is also much smaller; and more efficient comparison between fingerprints as the fingerprints have been stripped down of perceptual irrelevancies. Such a system is depicted in Fig. 3.1 below. An AFP should be *robust* to noise and distortion. This means that the fingerprint extracted from an original ‘clean’ track should be the same with the fingerprint extracted from a degraded copy of the same

track. This is achieved by using perceptual features which are invariant to signal degradation. Robustness is measured using the *false negative rate*, which corresponds to the case when two fingerprints of the same audio segment are too different to lead to a positive match. Furthermore, an AFP should be *reliable*, meaning that we do not wish incorrect matches. Reliability is measured using the *false positive rate*, which measures the rate at which songs are incorrectly identified.

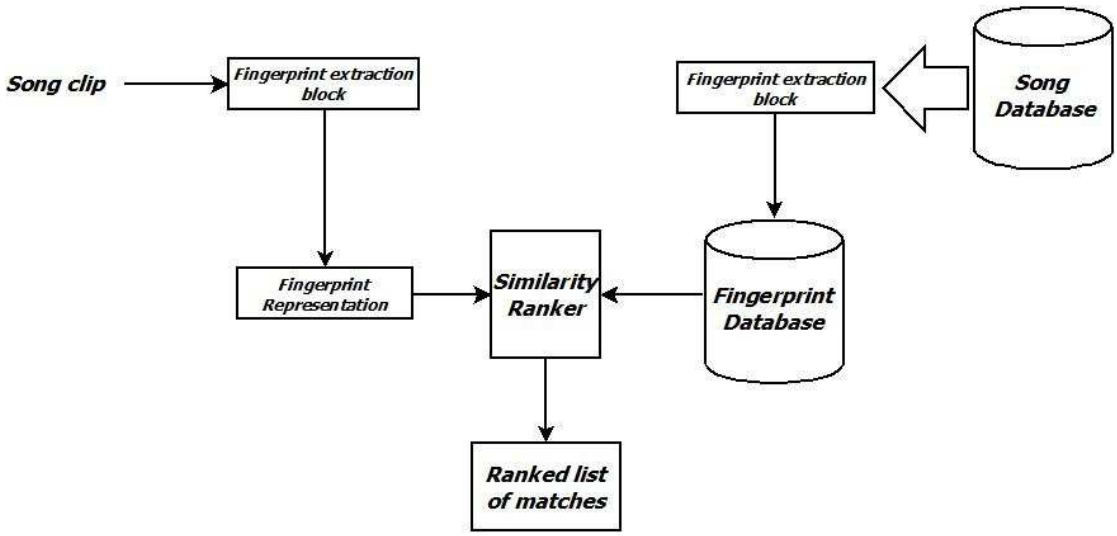


Figure 3.1: Overview of a QBE system

In what follows two approaches describing the method of extracting an AFP are presented. The first corresponds to a *frame-based approach* used inMusiwave developed by J. Haitsma and A. Kalker [51] and the second, more recent, corresponds to a *landmark-based approach*developed by A. Wang and used by Shazam [52]. In the present work we experiment with the latter case.

3.2.Frame-based analysis

The natural approach in calculating a feature representation of a music signal would be a standard frame-based analysis in which we divide the signal into short-time frames, calculate a feature vector for each of those frames, and obtain a sequence of feature vectors which represent the signal. Subsequently we would try to match our music piece on this basis. The method presented in [51] does exactly that.

The audio signal is first segmented into overlapping frames with a length of 0.37 seconds and an overlapping factor of 31/32, resulting in a feature vector every 11.6 milliseconds. The feature vector extracted from each frame is denoted as a *sub-fingerprint*. A single sub-fingerprint does not contain enough data to permit identification of an audio clip. The unit that allows such identification consists of 256 subsequent sub-fingerprints andis denoted as a *fingerprint-block*. As the most important perceptual audio features lie within the frequency domain, the Fourier Transform is applied to every frame. The range of frequencies taken into consideration is from 300 *Hz* to 2000 *Hz*. This range is further split into 33 non-overlapping and logarithmically spaced frequency bands. It was experimentally verified by the authors that the sign of energy differences along the time and the frequency axes simultaneously is a property that is very robust to many kinds of processing. This can be expressed by the equation below where *E(n,m)* denotes the energy of band *m* of frame *n* and *F(n,m)* denotes the *m*-th bit of the sub-fingerprint of frame *n*:

$$F(n,m) = \begin{cases} 1 & if\ E(n,m) - E(n,m+1) - \big(E(n-1,m) - E(n-1,m+1)\big) > 0 \\ 0 & if\ E(n,m) - E(n,m+1) - \big(E(n-1,m) - E(n-1,m+1)\big) \leq 0 \end{cases} \tag{3.1}$$

This results in a 32-bit sub-fingerprint which contains the sign of the energy difference between adjacent bands and windows. The overview of the fingerprint extraction scheme is presented in Fig. 3.2.

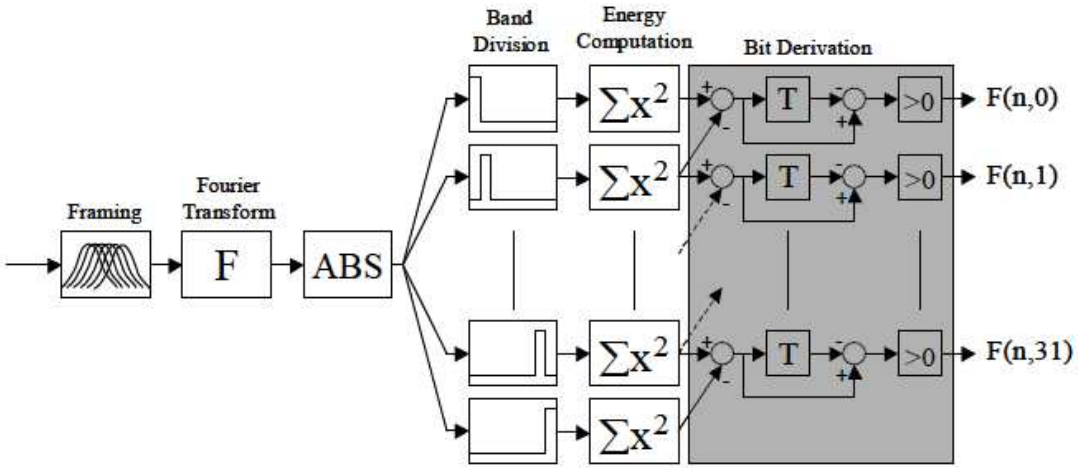


Figure 3.2: Overview of the fingerprint extraction scheme

A signal is then represented by a train of sub-fingerprints. Next, to measure the similarity between two recordings, we calculate the bit difference between the fingerprint of the query recording and the fingerprint of the reference song. The track which holds a fingerprint-block with the least bit differences is considered as the target recording. To avoid a brute force of comparison of an immense number of sub-fingerprints, [51] presents an efficient and very fast search algorithm for his system.

3.3.Landmark-based analysis

The idea behind the landmark-based approach developed by Shazam (A. Wang) [52] is to use the signal itself in trying to find some unique structure within it which can be used as a fingerprint, rather than creating a representation based on each frame of the signal. This analysis is based on the observation that prominent onsets within a signal's time-frequency representation are to be sustained under GSM encoding and the addition of noise. Spectrogram peaks are used as a feature in this method. These peaks correspond to points in the time-frequency representation with energy higher than all neighboring points in a region centered around that particular point [52][53]. Candidate points are then selected based on a density criterion and an amplitude criterion. These points are found for the entire signal, reducing its representation to a sparse set of peak-coordinates which is denoted as *constellation-map.*The constellation map of a spectrogram example can be seen in Fig. 3.3.Next, *landmarkhashes* are formed from the constellation map by forming the peaks into pairs and parameterize them by their frequencies and their time difference. In more detail, we choose an *anchor point* within the constellation and sequentially pair it with other points within a certain target zone associated to that anchor point. The information stored for each pair of peaks is the start-frequency, the end-frequency, and the time difference between them. These values tend to remain constant between the original and the distorted query signal and are thus the values to be used to represent the signal's identity. The values are quantized in order to give a relatively

large number of distinct landmark hashes. Each song in the database and the sample query song are

subjected to the same fingerprint analysis. The landmark hashes are subsequently held in an inverted index which lists the songs of the database in which they occur, and when they occur in the songs. The fingerprint of the query is then compared to the fingerprint for each database element by counting the number of matching anchor point pairs (landmarks).

A match can be declared once a sufficient number of landmarks have been identified. Since exact matches are very unlikely to take place, a small number of matches (e.g. 5) are sufficient to declare a match. Even in the presence of spurious time-frequency peaks injected due to noise, or when some peaks are missing, or even when the query example is strongly filtered, a sufficient number of hashed landmarks are to be sustained in order to correctly identify a match.

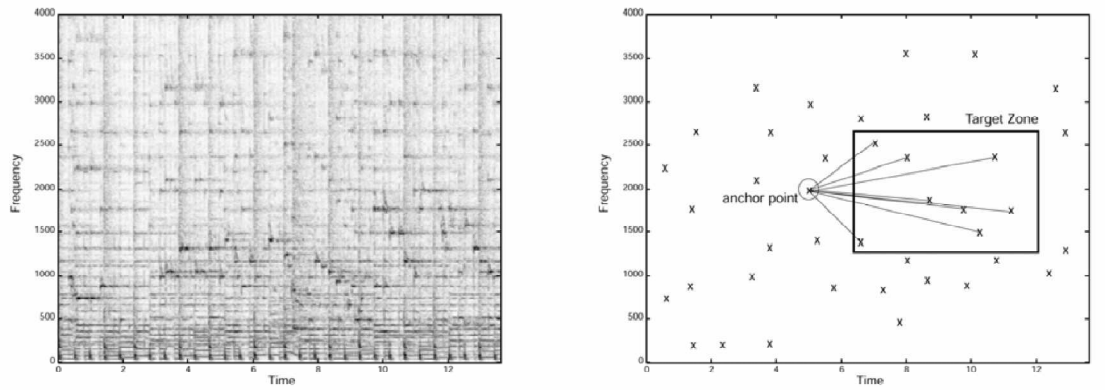


Figure 3.3: Spectrogram example (left) and its constellation map (right)

Even though a fingerprint-based system is very efficient in identifying a recording from a database, it is also very 'fragile' in as far as it is not able to identify alternate versions of a song (cover songs). In such cases fingerprints will differ significantly. Querying with one version will not return the other version, since they will not both have similar fingerprints because they will not have peaks at the same positions. Even if a human ear would mistake a song for another because of their similarity, an audio fingerprinting system would be able to tell them apart. For the case of cover song identification, features other than audio fingerprints should be used for matching. Such a system is described in the following chapter.

CHAPTER 4

Automatic Cover Song (ACS) Identification Method

This chapter describes the methodology of the cover song identification task. For each system it is essential to find the feature that maximizes the description of the desired attribute. Since our work is focused on cover-song detection we are neither interested whether the song is played with a particular instrument, nor if it is sung by a male or a female voice, nor if the song has a fast rhythm or not. Quite the contrary, we seek a feature which is robust in the above characteristics, reflecting changes in tempo, instrumentation and transposition. Moreover this feature should be able to track the music signal's melody which, by and large, remains the same in different renditions of the same song. In other words we are not interested in using timbral features, since these features are to differ the most. Instead, for the purposes of our research use is made of rhythmic and pitch content features for building a signal representation which satisfies our needs. First, a time-by-chroma representation of the signal is computed which is then averaged by the beat temporal locations derived from a beat-tracking module. Next the similarity between two songs is measured by cross-correlating these representations in respect to lag and transposition (key shifts) (Fig. 4.1).

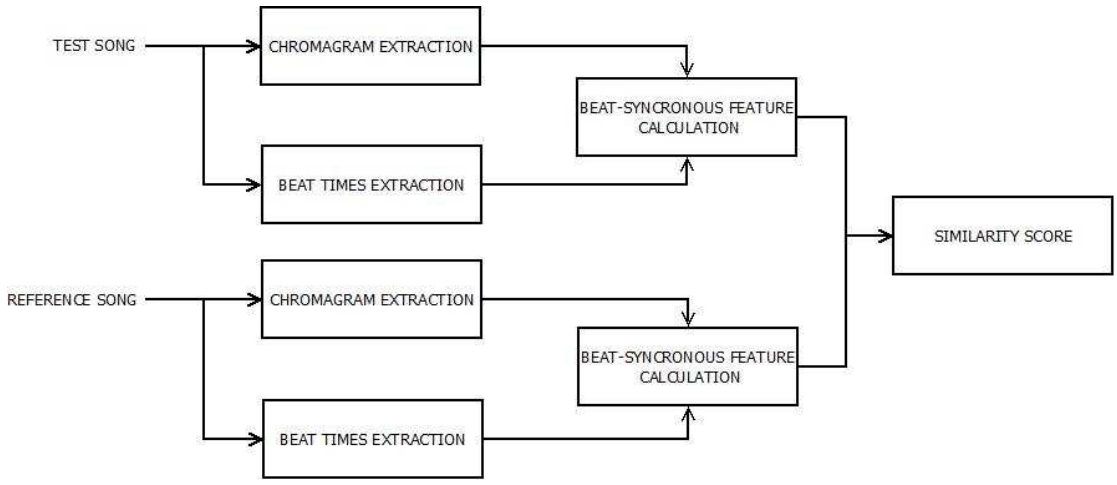


Figure 4.1: General block diagram of the cover song identification system

4.1. Extraction of chroma-vectors and chromagram

In section 2.3.2.we introduced the concept of *chroma*. As demonstrated by [54] it is useful to construct a signal's representation of chroma over time, known as *chromagram*. With its use we are able to track an audio signal's pitch, and thus melody (dominant note) and harmony over time, since both refer to a combination of pitches, either sequential or simultaneous respectively [55][32] [56]. An audio signal's *chroma-vector* is a representation of the spectral magnitude's distribution on the 12 different pitch-classes. To extract a music signal's chroma-vectors, which in their turn form the chromagram, one has to first look at a signal's frequency domain. Thus, we first split our music signal into time segments (frames) by multiplying our signal with a shifted hann-window of length *N=2048* and a hop-size of 1024 samples; we then compute theFFT for each resulting frame using 4096 FFT-bins. As a result we obtain the frequency spectrum as shown in Fig. 4.2.

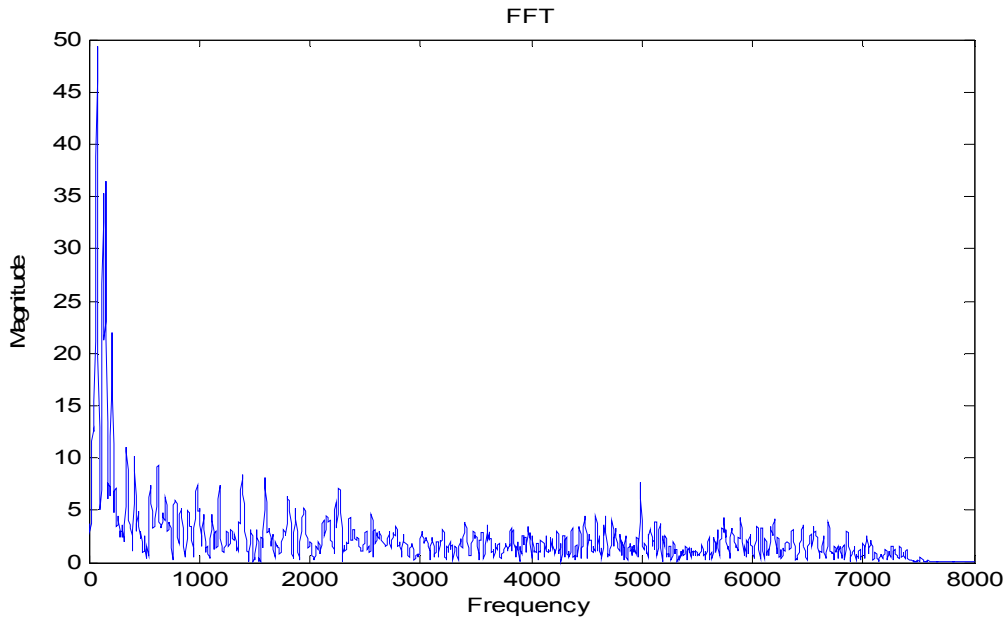


Figure 4.2: Frequency spectrum of an example signal

As proposed by many authors e.g. [14] [56], and as it turns out to result in better performance, only local maxima of the frequency spectrum are used for further

processing. These local maxima are found when a point has a higher value than both its preceding and its following points (Fig. 4.3).

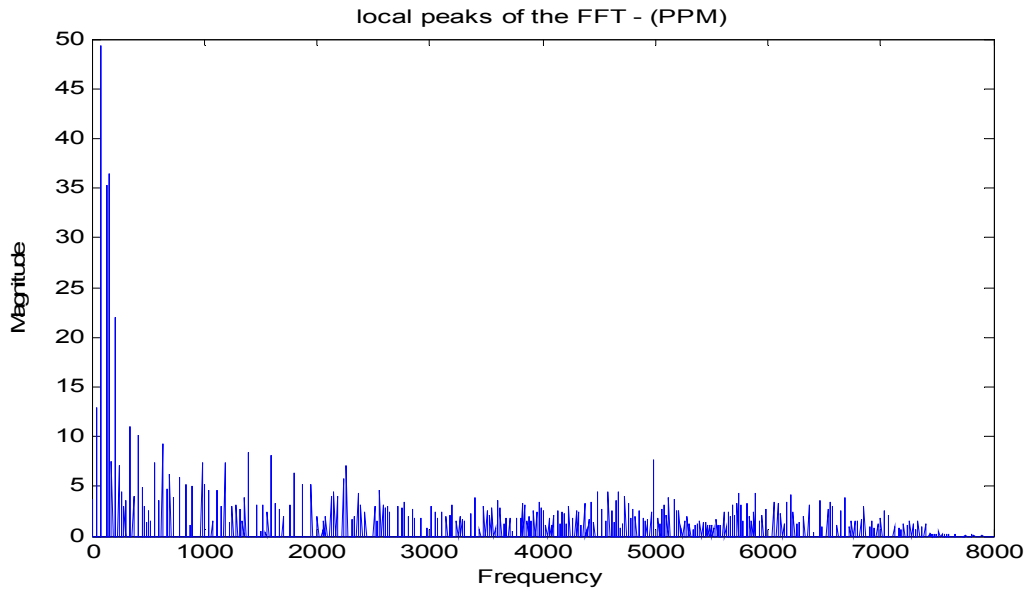


Figure 4.3: Peak picking of the example frequency spectrum

Given the local maxima of the frequency spectrum, the next stage is to map each frequency bin (fft-bin) to one of the 12 corresponding chroma-bins. The basic idea is to coil the magnitude spectrum around the pitch helix model (Fig. 2.11) and squash it flat to project the frequency axis to the chroma[30]. This is achieved by designing twelve different *band-pass filters* (*BPF*), one for each pitch class, that collect the frequencies corresponding to the correct chroma. Those BPFs are constructed by placing hann-windows centered on every pitch of the desired pitch-class. For example, the band-pass filter for the pitch-class C would have hann-windows centered at C0,C1,C2…etc, as shown in Fig. 4.4. The equation representing one 'lobe' created of a hann-window centered at frequency $F_{Shepard}$ given in *cents*, is provided by:

$$BPF_{c,h}(f) = 0.5 * \left(1 - \cos\frac{2\pi\left(F_{cent}(f) - \left(F_{Shepard}(c,h) - 100\right)\right)}{200}\right) \qquad (4.1)$$

where*f* is the frequency in *Hertz* converted into *cents* according to equation (2.16). Thus, the final equation for the BPF of a pitch-class *c* is:

$$BPF_c(f) = \sum_{h=0}^{8} BPF_{c,h}(f) \qquad (4.2)$$

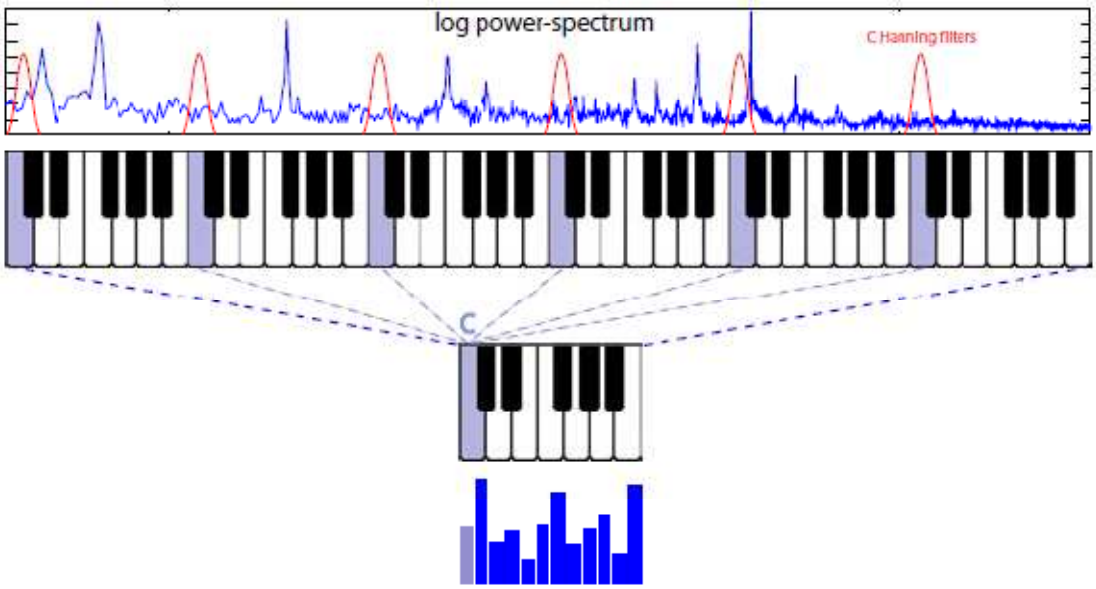


Figure 4.4: Placing of Hann-windows in the logarithmic power spectrum

An example of the band-pass filter corresponding to the pitch class A is presented in the figure below:

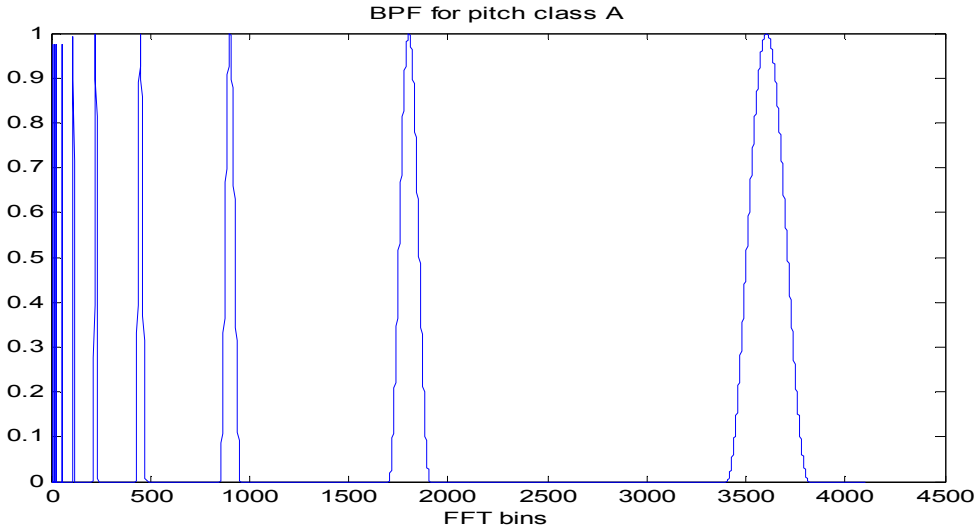


Figure 4.5: Band-pass filter for the pitch-class A

The BPFs so obtained are then zero-padded for FFT-bins greater than 2048 and a scaling process is applied to the bins in order to favor lower frequencies. The scaling process is achieved by multiplying each BPF with a Gaussian function defined as:

$$P(F_{cent}) = e^{-\frac{1}{2}\left(\frac{F_{cent}-F_{cent_{ctr}}}{1200}\right)^2}$$

where$F_{cent_{ctr}}$=*4635 cents* corresponding to $F_{Hz}ctr$=*400 Hz* (Fig. 4.6), and a width of one octave corresponding to 1200 *cents*. The scaling function and the result of its application to the initial filter are shown in Fig. 4.6 and Fig. 4.7 below:

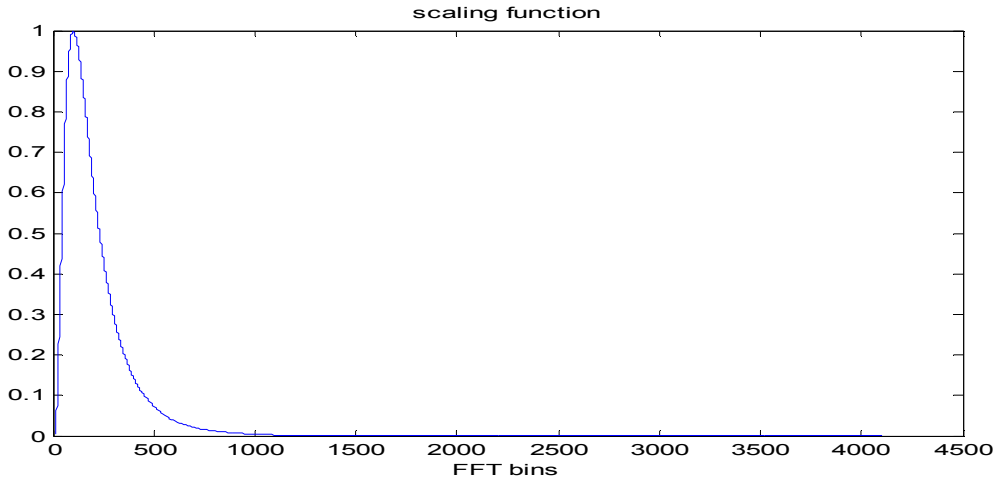


Figure 4.6: Scaling function centered at 400 Hz and base width of 1 octave

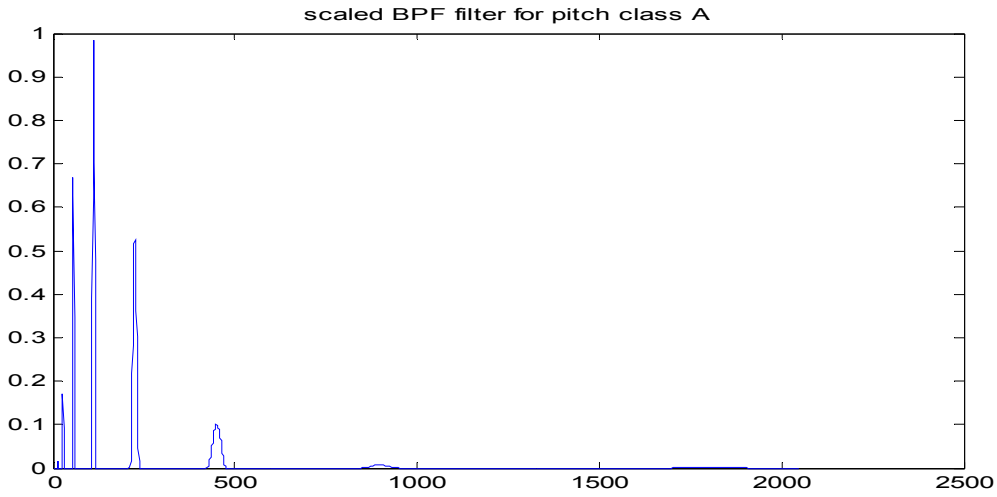


Figure 4.7: Result of the application of the scaling function to the BPF of pitch class A

By multiplying the scaled BPFs with the function corresponding to the local maxima of the frequency spectrum we obtain the values which need to be summed together in order to measure the magnitude distribution on each chroma-bin. This results in a 12x1 chroma vector which represents the spectral magnitude's distribution on the 12 different pitch-classes for the particular frame. This vector is subsequently stored. A representation of a chroma-vector is displayed in Fig 4.8. By repeating this process for

each frame of the song and by appending the resulting chroma-vectors one after the other we produce a 12x*N* (where *N*=number of frames) matrix, the chromagram. In Fig. 4.9 and Fig. 4.10 we may observe the resulting chromagram representations of two different audio signals, one of which follows the note sequence of an octave and the other is an example song (Madonna - Like a virgin).

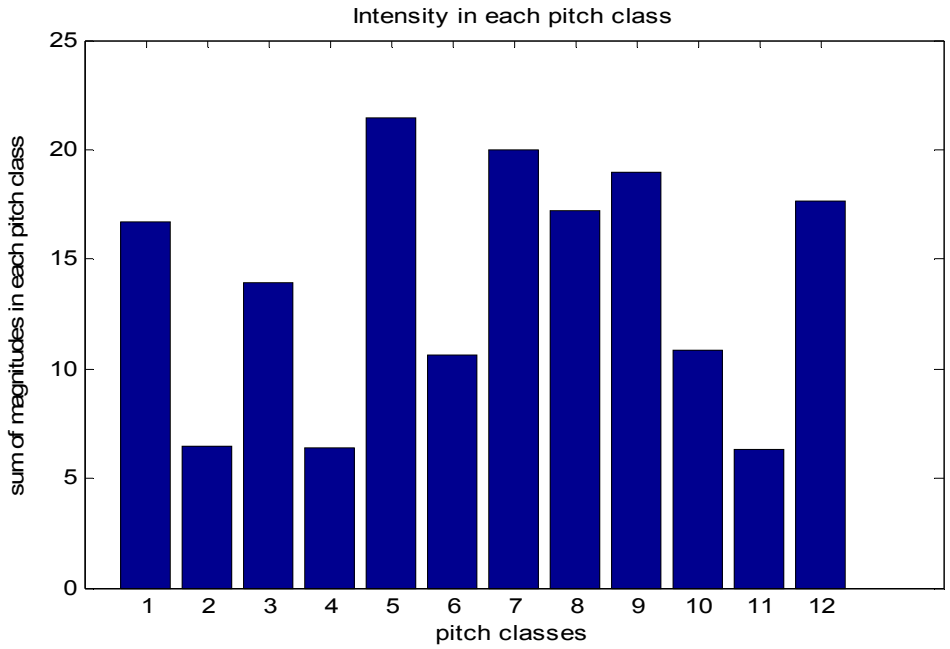


Figure 4.8: Visualization of a chroma-vector

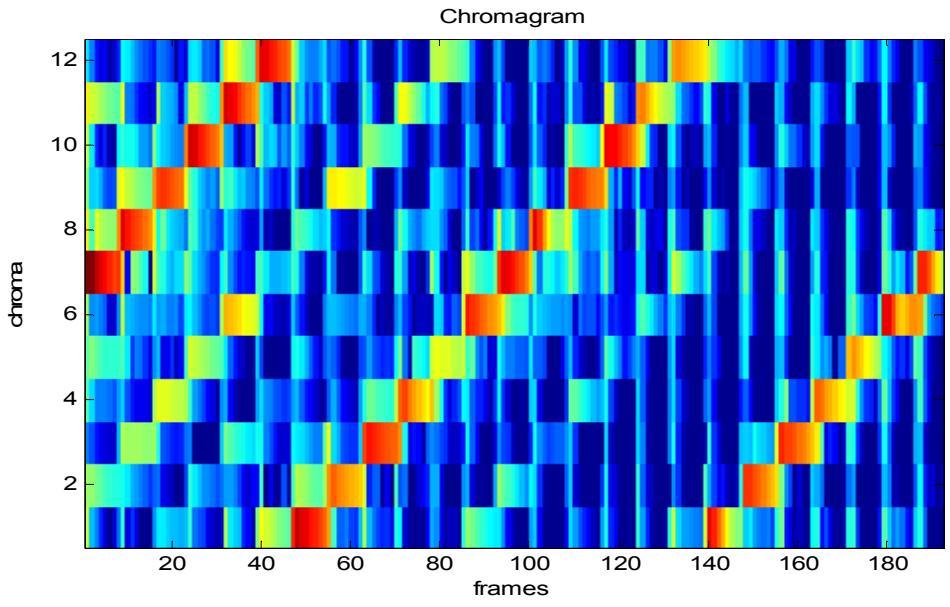


Figure 4.9: Chromagram of an audio signal which follows the note sequence of an octave

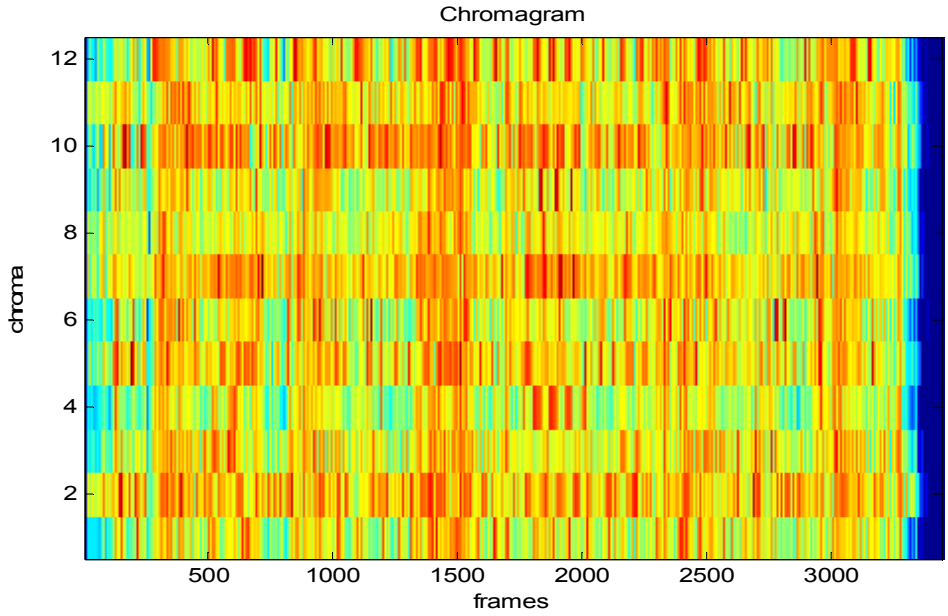


Figure 4.10: Chromagram of an example song

An interesting aspect in the use of chroma representations is that key transpositions are simply circular shifts of the rows of the chroma. Since cover versions are commonly transposed in respect to the main key, achieving key transposition invariance is usually reached by examining each and every possible key transposition of the feature. This method will be further examined in section 4.4.

It is worth noting that in the work performed by Ellis and Poliner [14], instead of using a coarse mapping of FFT bins to one of the 12 chroma classes, they use the phase derivative (instantaneous frequency)[57] [58] within each FFT bin to identify strong tonal components in the spectrum as well as to get a higher-resolution estimate of the underlying frequency. For the purposes of the present work,however, instantaneous frequency was not taken into consideration.

4.2. Tempo estimation and beat tracking

As mentioned in chapter 2.3.3. tempo estimation and beat tracking systems describe the temporal structure of a music signal, and thus are fundamental processes in automatic music processing. For the task of tracking a music signal's tempo and beat locations we use Ellis's method of *beat-tracking with dynamic programming* [29]. A more thorough survey on different approaches used for beat tracking can be found in [59].The method used in this work consists of three stages (Fig. 4.11). First, we extract from the signal an *onset-strength signal* which is a representation of the succession of discrete acoustic events such as beats, or note changes in the music signal. This onset-strength signal is further processed in order to derive an estimation of the *global tempo* of the musical piece which is based on periodicity estimation algorithms. Finally, by using dynamic programming we find the temporal locations of each beat, so that it is placed on a moment which has high onset-strength and agrees with the spacing of the global tempo simultaneously.

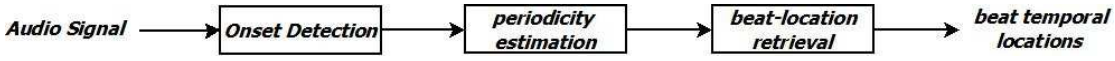


Figure 4.11: Overview of the beat tracking algorithm

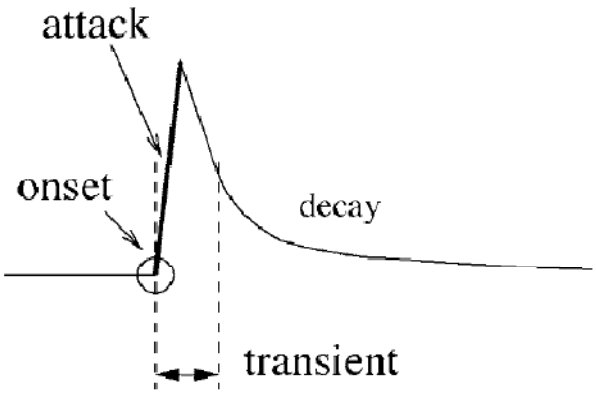


Figure 4.12: Ideal sound wave signal and its four constituted parts

4.2.1. Extraction of the onset-strength signal

An ideal sound wave signal and its four constituted parts (onset, attack, transient and decay) as presented by [60], is displayed in Fig. 4.12 above. Audio is often considered to be a succession of discrete acoustic events (such as beats, pitch changes etc.) and an onset detection process is a way to determine the temporal position at which those events begin. In a real polyphonic music signal where different sound objects occur simultaneously, it is highly unlikely to get an ideal audio wave and thus onset positions found by different algorithms might not be exactly the same. The desired system should take an audio signal as an input and extract a detection function which will indicate the location of the most salient features, such as note changes, harmonic changes or percussive events, since beats tend to occur at these time instants. The fact that note onsets or percussion hits are easily masked in the overall energy of the time-domain signal by continuous tones of higher frequency (such as bass notes) encourages frequency-domain processing [61], where such events are detected even if they are of much lower energy than other continuous signals in the audio. This leads to a system that tries to locate areas in time and frequency where there is a sudden energy increase. This can be achieved by building a function which represents the energy of a sound at a particular time, or in other words this function is a result of summing energy across frequency.

In order to construct the onset-strength signal, the analyzed music input signal is first downsampled to 8 *KHz* and then divided into frames of length *N=512* and an overlapping factor of *N/8.* Each frame is then multiplied by a *hann-window* of the same length and is then processed by a FFT-function resulting in the frame's frequency spectrum.The fact that these onsets are clearly audible by a human's ear leads to the use of an auditory model using a mel-scaled frequency representation of the signal's frequency, in which the relationship between *Hertz*-frequency and *mel*-frequency are given by the equations (2.7) and (2.8) referred to above in section 2.1.2. This is done with a view to better approximating the beats from a human's auditory system perspective. A driving function could be derived at that point from the

resulting representation by summing the magnitudes across frequency, resulting to a plot which represents the running energy of the audio signal in respect to time.

Another aspect that must be considered, though, when summing energy across frequency is where energy shifts in frequency, as for example when an instrument (i.e. saxophone) plays a note and then plays another note continuously. The total energy in that case might not change that much, though a human's ear still hears that as an onset. To overcome this particular case of energy shift, we divide the frequency range into multiple bands and calculate an onset function within each band individually.Just like the mapping of FFT-bins into chroma-bins in the previous section, we map the FFT-bins into 40 different mel-bands by multiplying the frequency representation with a suitable BPF expressing each separate band resulting in a 40x*N*mel-spectrogram (where *N*=number of frames of the analyzed sampled signal). Next, we convert the magnitude into *decibels* and look only at the top 80 dB of the representation.

Then a first-order difference from frame to frame is calculated for each separate band and the result is half-wave rectified. This is done in order to keep only positive values and leave only onset information resulting in a driving function for each band. These driving functions exhibit sharp maxima (peaks) at the desired temporal locations (onset times).Next, by summing the differences, or by taking the vertical mean of the differences for each frame across bands (frequency), we obtain a function which exhibits large values at onset times and corresponds to the final onset-strength signal.We finally filter this driving function with a high pass filter corresponding to global gain variations in the original signal. The resulting driving function can be depicted in Fig. 4.13 below.

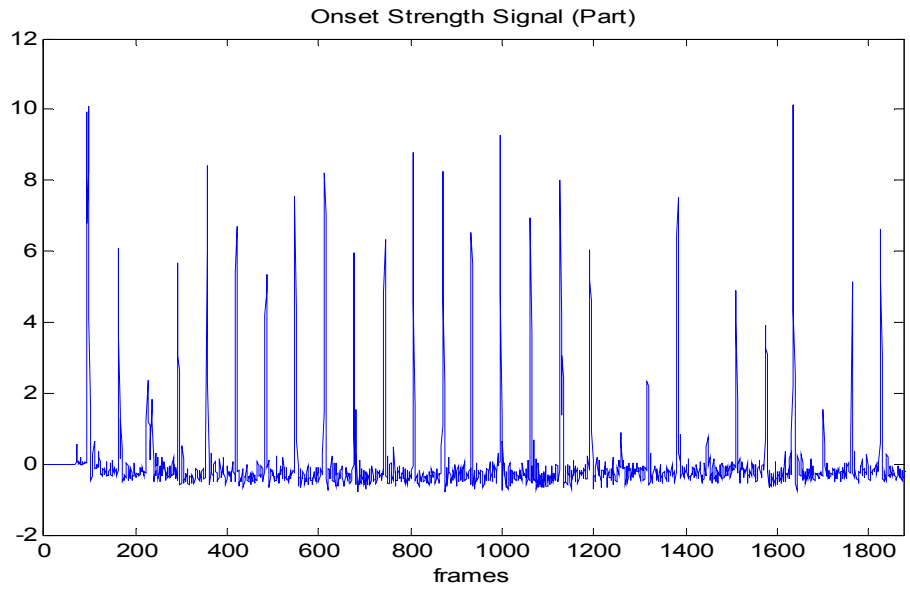


Figure 4.13: Onset-strength signal from a part of a song

4.2.2. Estimating the global tempo

The onset-strength signal derived from the previous stage is then passed to a periodicity estimation block in order to try and find a sequence of onsets in the signal which are on a regular pattern. Given that the onset function can be seen as a quasi-periodic and noisy pulse-train that has large peaks at note attacks, a natural approach to compute the tempo would be to compute a *beat histogram* by taking the difference in time between each successive pair of onsets and build a histogram of those differences. In this case, if there is a value in the histogram which indicates that pairs are closely spaced around a specific value, then the tempo can be derived from that information. Other methods in calculating the periodicity from the onset locations are the *spectral product* and the *autocorrelation function* (ACF), which is a classical method in periodicity estimation. Results calculated using the autocorrelation function are usually better than those stemming from the spectral product (and the histogram) and therefore we choose to use autocorrelation for this purpose as well. The autocorrelation function is a special case of the cross-correlation described in chapter (2.1.5), and can be regarded as a cross-correlation of a signal with itself. In other words it is a signal that is multiplied by itself while it is being shifted in time. The signal that we are going to correlate with itself is the output of the onset detection

block and since it can be regarded as a kind of periodic pulse train, we expect to see peaks in the autocorrelation spectrum at the times where the shifted signal aligns well with its original. To give an example, if the onset function has sharp periodic pulses, or pulses which form repeated patterns, its autocorrelation will exhibit peaks at times which correspond to one period or any integer number of periods of those events (or pattern of events), since for any number of periods the original and the shifted version of the onset function will line-up well. Consequently, to estimate the driving function's periodicity and hence the audio's global tempo, we auto-correlate the onset-strength signal with itself out to a maximum lag of 4 sec. It can be pointed out that it is not necessary to auto-correlate the entire signal and that only a portion of it, which retains the basic information, can be used instead. A peak picking process is then applied to the autocorrelation in order to find local maxima within the function. An autocorrelation example and its result after finding its local maxima are depicted below in Fig. 4.14 and Fig. 4.15:

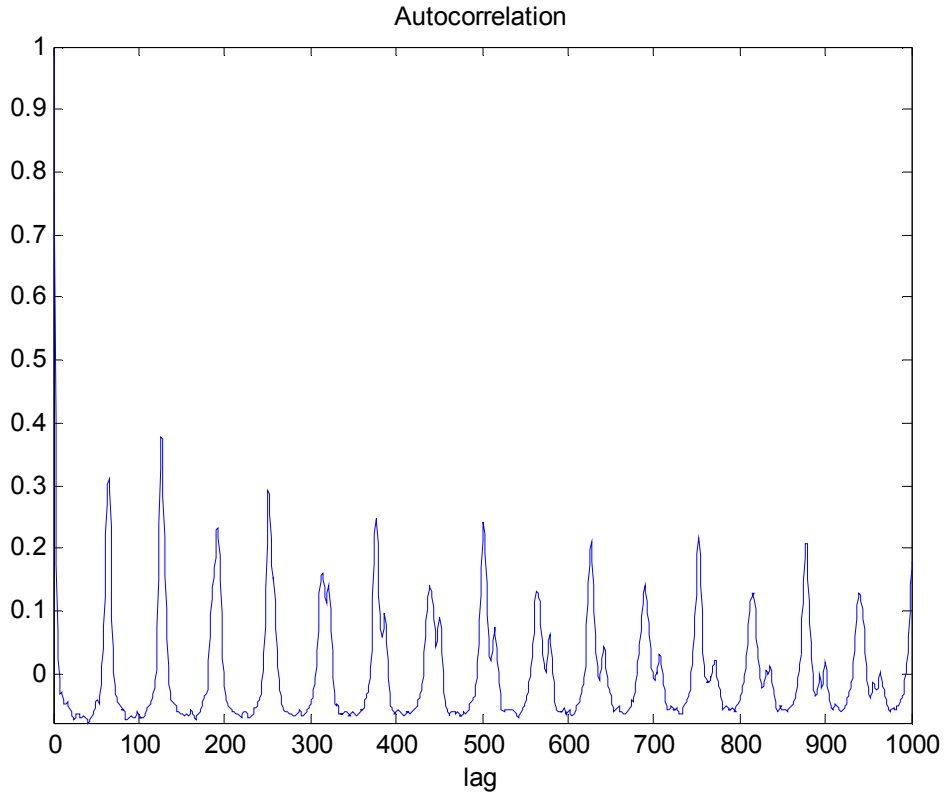


Figure 4.14: Autocorrelation function of an example onset-strength signal

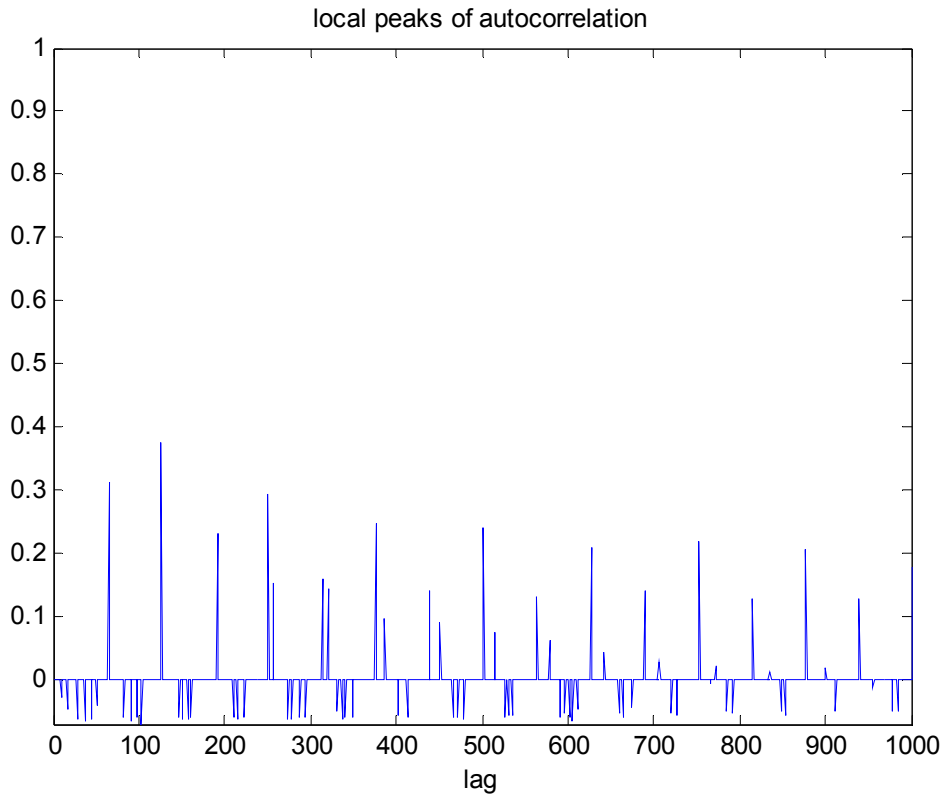


Figure 4.15: Peak picking applied to the autocorrelation function

By considering the human tapping experiment [49] [50] and the fact that humans tend to tap toward a particular range of tempi, we scale the resulting autocorrelation signal which has typically multiple peaks in order to enhance periodicities preferred by listeners and extract one dominant peak, which also indicates the most likely tempo. To this effect a 'human preference' weighting window is created. This window is a Gaussian on a log-time axis and is characterized by its center (the BPM at which it is largest) and its half-width (the sigma of the Gaussian, in units of octaves on the BPM scale, since the axis is in fact logarithmic) [29].In other words, we use the weighting window to adjust the autocorrelation peak's heights so that these favor the peaks that are located around the right tempo. The best center of the window was found to be at 120 BPM, agreeing with the results of the human tapping experiment, and the width was set to be 1.4 octaves. Two weighting windows centered at 120 and 240 BPM respectively are shown in Fig. 4.16.

Finally, the lag in the scaled autocorrelation (Fig. 4.17) which has the maximum value is chosen to correspond to the main tempo. Taking into account that humans

might tend to tap in different metrical levels and that those levels are usually separated by a ratio of 2 or 3, a secondary tempo is computed in addition to the main tempo, which is chosen to be the largest value of the peaks closest to 0.33, 0.5, 2 and 3 times the main tempo, respectively.

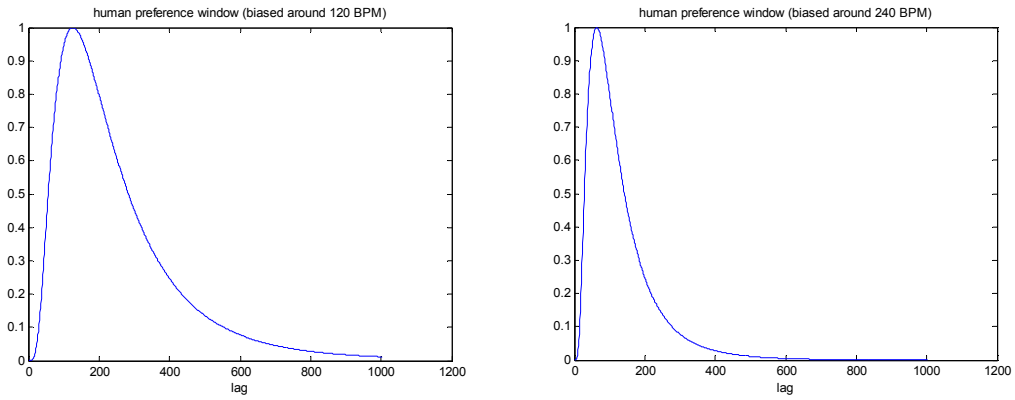


Figure 4.16: 'Human preference window' biased around 120 BPM (left) and 240 BPM (right)

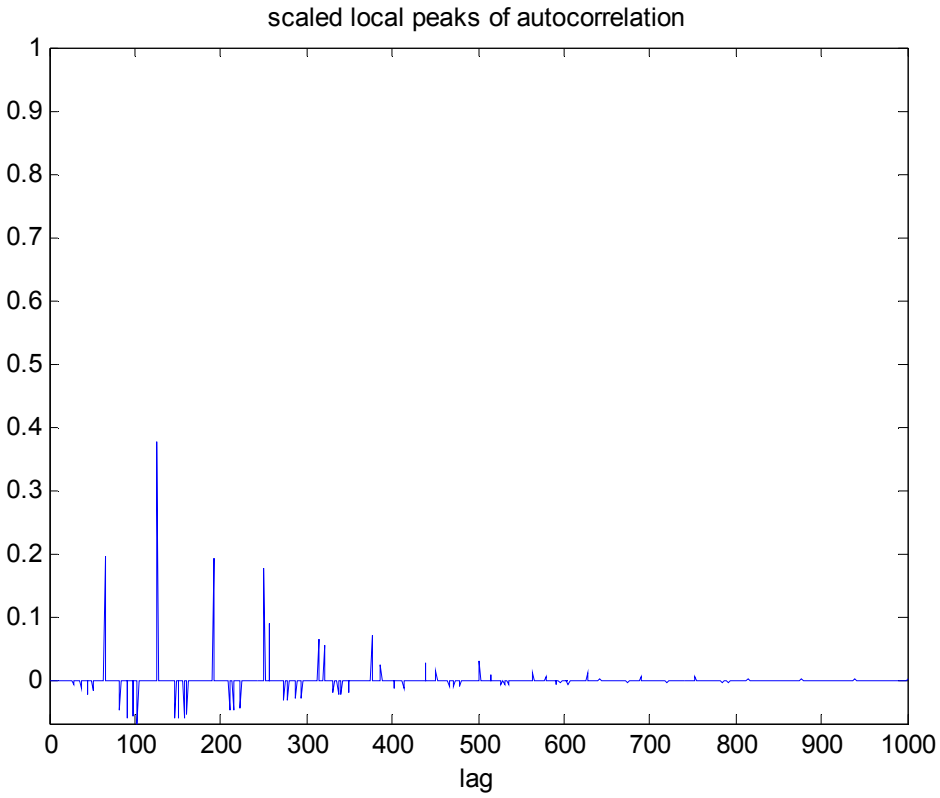


Figure 4.17: Scaled local maxima of the autocorrelation favoring peaks around 120 BPM

4.2.3. Temporal beat-location retrieval

In order to obtain the beats position in time we use a beat tracking module that will try to detect and mark all the beats temporal locations using the information obtained from the previous stages (onset-strength signal extraction stage and global tempo estimation stage). The idea is to create an algorithm which will decide for any given time *t*, if there is a beat or not and find a sequence of those beat times $\{t_i\}$that all correspond to large values in the onset function and their spacing agrees to the estimated global tempo. Meaning, we want the sum of the onset function at those times to be as high as possible and that the differences between successive beat times to be consistent. These conditions can be expressed with the equationsbelow, where *O(t)* is the onset-strength signal, *F(Δt,τ)* is the *tempo consistency score* and *a* is a balance function whichdefines how rigidly we stick to the tempo τ_p and how well we hit the high score times on the onset function:

$$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + a \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p) \qquad (4.3)$$

$$F(\Delta t, \tau) = \exp\{-\frac{1}{2}\left(tightness * log\frac{\Delta t}{\tau}\right)^2\} \qquad (4.4)$$

The tempo consistency function *F*is just a way to compute if the spacing between two beat times agrees (or not) with a given ideal spacing (τ_p), in our case the global tempo. If *Δt*and *τ*are equal, then the ratio within the log will be equal to 1 and hence the log will be equal to zero, making this the best case of *F* being one.If, on the other hand, *Δt* and *τ* are different, then the log of their ratio will have any value other than zero and by taking the negative value of its square the final value of *F* will be a real number between 0 and 1.

So in theory, if we could calculate the above cost function for every possible beat sequences, which is an exponential set, and find the sequence that has the maximum score, then that set of beat times would be the most probable beat sequence. This is not feasible, however, since coming up with all possible time sequences is an exponential problem and is not solvable for a large amount of data. Dan Ellis [29] formulated the above problem intoa dynamic programming one, in a way that onset

times correspond to local information subject to a long-term constraint reflecting that we want to form a regularly spaced set of events. In dynamic programming we are trying to find an optimal combination of a local cost function and a transition function of a path cost within a space. Hence by decomposing the problem as mentioned above and by using dynamic programming we are able to solve it in linear time. In our case, since any sequence of beat times corresponds to a different path in space, by taking a particular time-point we observe that several sequences lead up to that point, but there is only one that is optimal. This optimal sequence is found by calculating each sequence's score and by keeping the one that has the maximum score. Given that the cost function's structure is using only past information in time, any path that leads to a particular point ends its influence to the total cost at that point and thus anything that happens after that time is not affecting the score. So basically we calculate the score for every possible *t*, even though the majority of the times will not correspond to beat times.We do so by adding the local cost score (onset function), by searching over some preceding window the best score up to that time and by multiplying some transition cost which shows how well we can move from the preceding point to the new one (tempo consistency cost). The preceding window covers a range of 2 to 0.5 beat periods into the past. This can be expressed by the equation below, where $C^*(t)$ corresponds to the best score up to time *t*, $O(t)$ is the onset signal, $F(\Delta t,\tau)$ is the tempo consistency cost function and *a* is the balance function that scales the best score at the preceding beat in order to keep balance between past scores and local match:

$$C^*(t) = (1-a)O(t) + \max_\tau\{aF\left(t-\tau,\tau_p\right) * C^*(\tau)\} \tag{4.5}$$

This process can be depicted in the figure below:

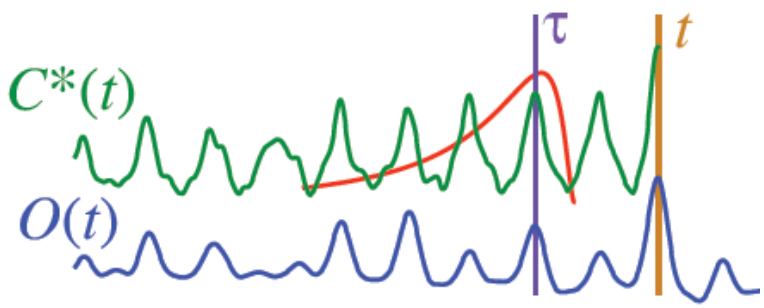


Figure 4.18: Finding the best predecessor beat for a given time *t*

The largest scaled value by the transition weight corresponds to the best predecessor beat of the current time *t* and is then added to the onset value of *t* which makes the best cumulative score $C^*(t)$, which is stored along with the best predecessor beat. Doing so for all the times, and finding the most probable last beat time, we can trace back through the stored beat times and acquire the entire final beat sequence. The back-trace function can be defined as:

$$P(t) = \text{argmax}_\tau\{aF(t-\tau, \tau_p) * C^*(\tau)\} \quad (4.6)$$

The fact that the algorithm uses back-trace makes the above beat-tracking system intrinsically non-real-time. In addition it relies only on a single global tempo making it unable to track large (>10%) tempo drifts. As our work is mainly focused on popular music corresponding to music genres with a more or less invariant straightforward rhythm, the current beat-tracker indicates high performance.

4.3. Beat-synchronous chromagram

As it turns out tempo, and therefore beat times, are a useful time-normalization metric and thus through averaging our chromagram (the output of our chromagram extraction stage) according to the beat times (the output of the beat tracker) in order to have one chroma-vector per beat, we may overcome the variability in time (time-shifts, tempo) between two songs. This is achieved by implementing a function which takes the chromagram and the beat times as an input and by creating an 12x*B* (where *B* are the number of beats) matrix representation which consists of *B*chroma-vectors whose values are derived by averaging the originalchromagram in time intervals defined by the beat times.As a result we obtain a feature representation of the song which is invariant to instrumentation due to the use of 12-dimensional chroma vectors which collect spectral energy supporting each semitone of the octave, and invariant to tempo changes, as a result of the use of beat-synchronous features. The case of variation in transposition (key shift) will be examined during the matching module.

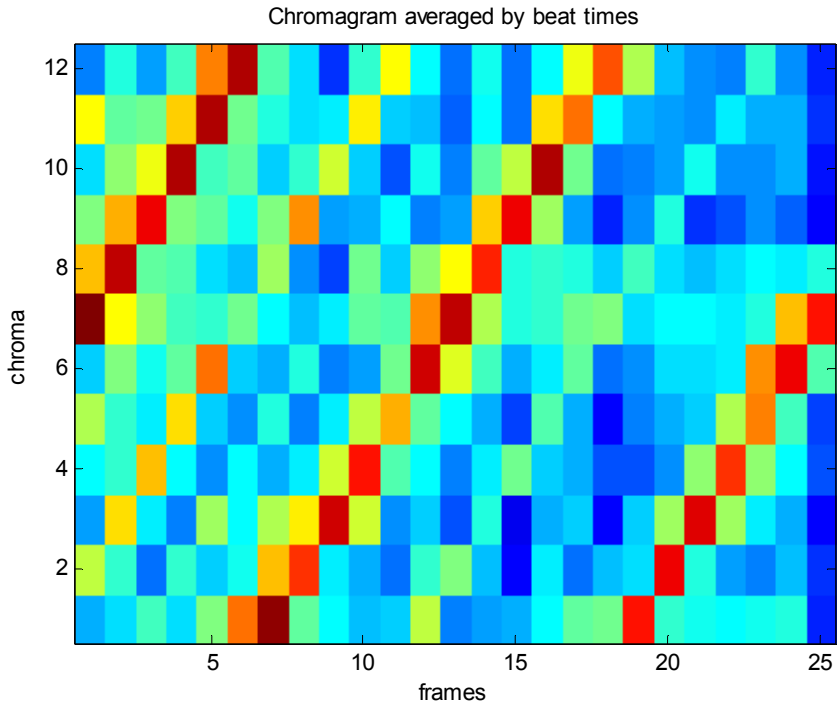


Figure 4.19: Beat-synchronous chromagram of the example chromagram in Fig.4.9

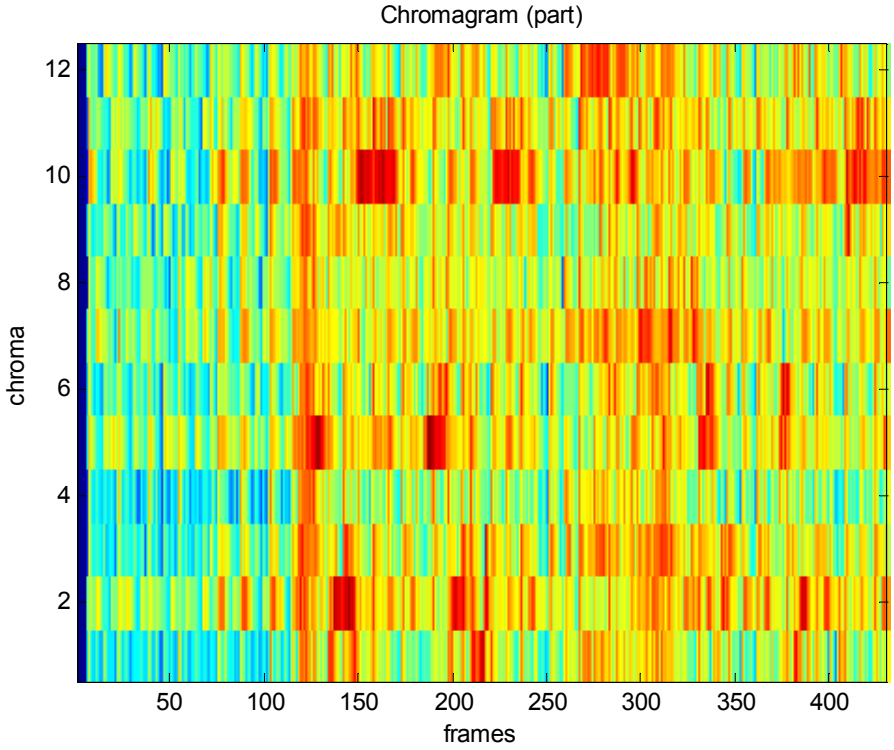


Figure 4.20: Part of a chromagram of an example song

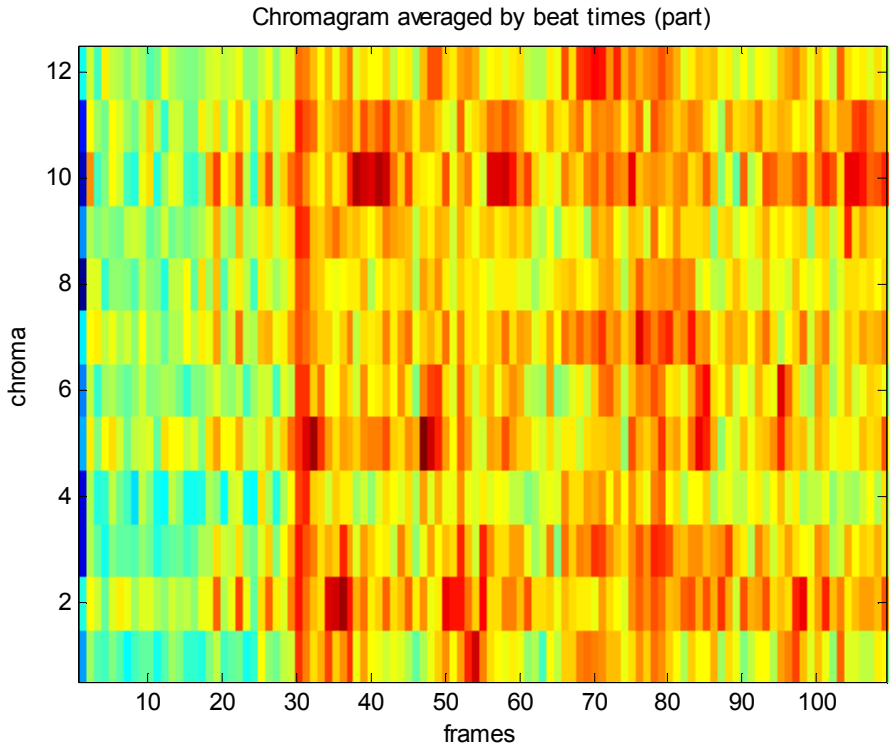


Figure 4.21: Beat-synchronous chromagram of the above example

4.4. Matching

To compare two music pieces we follow the method of cross-correlating their entire beat-by-chroma matrix representations. This correlation will exhibit sharp peaks expressing good local alignment of the two pieces at a precise lag. In other words, if there is a subsequence of chroma-vectors in the reference song which is similar to a subsequence of the test song, this will result in a local peak in the cross-correlation at the appropriate lag. The length of the matching subsequence and the degree of their similarity are directly related to the size of the peak in the cross-correlation, meaning that the bigger the length of the subsequence or the bigger the degree of similarity, the higher the height of the peak. The disadvantage of this method is that it is not possible to reward the case where multiple parts align at different relative positions. On the

other hand, a small fragment of the total track which has high similarity will still contribute in a peak high enough to correctly point out a match between two tracks.

Before proceeding to cross-correlating the representations a normalization step is involved. This normalization consists of square-root compressing the amplitude of the representations followed by scaling each chroma-vector of the beat-synchronous chromagrams to have unit norm. Consequently, to overcome variability in key transposition the cross-correlation is performed between all twelve possible semitone transpositions (circular rotations in respect to the pitch axis) of the chromagram and the rotation for which we have the highest score is selected. Through experimentation it was found that many tracks appear to have false large correlations due to sustained sequences of a single chroma-bin with a large value. If we think about it, if we cross-correlate in relation to time and key transpositions, sustained blocks of a single chroma value (any chroma) in both pieces will result in a correlation peak, without witnessing any 'true' similarity. To overcome this case of false similarity, we de-emphasize prolonged chroma values and instead are focusing on the changes of the representation which is more informative. This is done by high-pass-filtering the beat-by-chroma representations for each song. Last but not least, it was observed that representations which are averaged based on beats with different metrical levels, having for example twice as many beats per song phrase as its other version, prevented a correct match. In order to accommodate this case, we created two representations of each song. The first representation averages the chroma-vectors using beats derived from the beat tracker which makes use of a 120 BPM biased window, whereas the second representation is using a 240 BPM biased one. Cross-correlation is then performed between the two representations of the query song and the two representations of the reference song and the single largest value of the four cross-correlations is taken to be the final similarity score.

CHAPTER 5

Automatic Chorus Detection Method

In the work presented by Serra et al. [15][26] it has been demonstrated that song structure is a key factor to consider for cover song similarity measurement. To this endwe opted to present an approach which consists in extracting a music summary of a given song representing its most representative or repeated parts [29][49]. Bartsch and Wakenfield [32] argue that, in the context of popular music, extracting a music summary amounts to picking up the chorus, which is likely to be recognized or remembered by a listener. Such a summary representation is thought to provide the main characteristics of a title in a more compact form allowing for instance a faster music similarity search amongst a set of music pieces. Therefore, a hypothesis can be made that by making use of such summary information one could increase a cover song identification system's performance [15]. Moreover, a system which identifies repeated parts within a song has interesting applications in other systems, as for example in an enhanced music player where the sense of an 'intelligent fast-forward' mode is introduced. For instance, a user may move automatically to the next chorus of the song or move to the next occurrence of what is currently being played. Given the above attractions and the fact that a structure analysis system can be developed by using the features already described in the present work, it was considered useful to make also reference to *automatic chorus detection* (ACD) systems. Having said this, it should also be pointed out that trying to achieve structure invariance in a cover song identification system by using a song summarization approach may be error-prone. This is because it cannot be ascertained that the most representative part of a song is included in the extracted music summary i.e. the chorus, as this may be located in another part of the song, for example the introduction, the bridge, or other. Also it is acknowledged that music structure systems are in need of further refinement [4][34].

In what follows a method which automatically detects a chorus for a given song with no prior information on acoustic features unique to choruses is presented. The basic idea of this method is to find sections in a song that repeat themselves and output the one that is most likely to correspond to the chorus. To identify the repetitions in a song we make use of its beat-synchronous chromagram representation examined in the previous Chapter (section 4.3) and a self-similarity matrix (SSM).

The SSM calculates the similarity between each pair of frames so that repeated sections are shown as diagonal lines with high similarity. These diagonal lines, which correspond to precise music segments, are then listed and only one line segment is selected as the chorus, utilizing a novel heuristic scoring scheme presented by [31]. It is important to mention that a frame segmentation of the given song is necessary. We follow the method used in [31] and [32] by making use of dynamic beat-synchronous frame segmentation, having one feature vector per beat and thus measuring time in beat units. This is a crucial process which lessens greatly the computational load of the system. The use of chroma vectors to represent the song is proposed by many authors [30] [31] [32] [33], since its ability to encode harmonic relationships is critical for this task.

AntiiEronen [31] used a combination of two SSM, one corresponding to beat-synchronous chroma-vectors and the other corresponding to beat-synchronous MFCC-vectors. However, for the purposes of the present work we limit our analysis to the use of only one SSM derived from the beat-synchronous chroma feature on the grounds that the information retrieved is sufficient. Nevertheless extending the model to include information derived from MFCC features could enhance its overall performance.

5.1. Obtaining the SSM

As we are working with chroma-vectors, two audio frames with similar harmonic content will have "similar" feature vectors. We define the similarity between two chroma vectors *v(i)* and *v(j)* as:

$$s(i,j) = 1 - \frac{\left\| \frac{\vec{v}(i)}{\max_c v_c(i)} - \frac{\vec{v}(j)}{\max_c v_c(j)} \right\|_2}{\sqrt{12}} \quad (5.1)$$

where$1 \leq i,j \leq N$ and *N* is the total number of beats in the song.

Each vector is normalized by dividing it with its maximum element. Then the normalized vectors are subtracted from each other and the Euclidean norm is calculated taking the squared root of the sum of the squared elements of the vector, resulting to the Euclidean distance of the two vectors. Next, we divide the Euclidean distance with $\sqrt{12}$ which is the diagonal line of a 12-dimensional hypercube with edge length equal to 1, guaranteeing that the condition $0 \leq s(i,j) \leq 1$ is satisfied. Finally, the result is subtracted from 1 in order to obtain the similarity measure.

Doing so for all *i,j*$\in \{1, N\}$ results in a $N * N$ SSM (where *N* is the number of chroma vectors) which represents the similarities of each set of points taken pair-wise. This can be observed in Fig. 5.1. For example $s(i,j)$ is an element in the similarity matrix that represents the similarity between the ith and the jthchroma-vector of the song's representation. The higher the similarity between those two frames the darker the point in the visualization of the SSM. Hence, extended regions of high similarity such as choruses or other repeated sections, result in extended areas of high similarity in the SSM and are pointed out by darker diagonal line segments. Our objective is to detect and list all those line segments in the SSM and examine which one is most suitable for the system's output. It is important to point out that the Euclidean distance is symmetric and thus the similarity matrix is symmetric as well. Therefore, all the following operations take into account only the lower triangular part of the SSM, resulting in a significant computational time reduction. In order to process the similarity matrix it is strongly advised by [30][31] to enhance the similarity matrix.

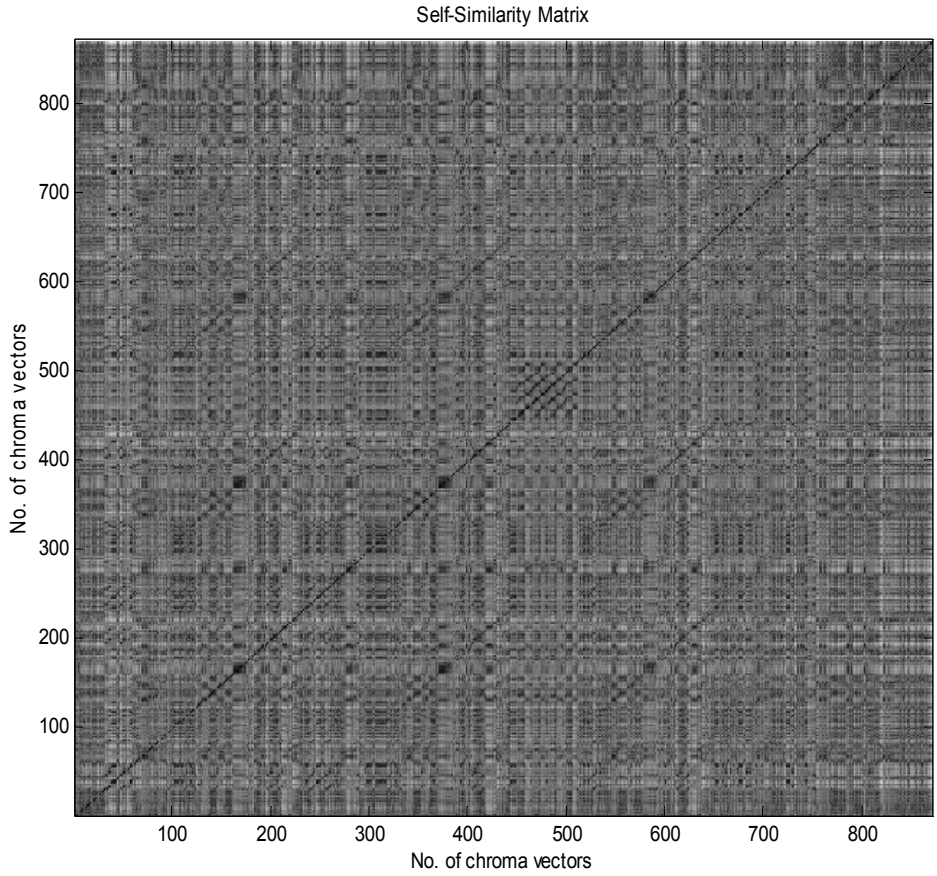


Figure 5.1: SSM of a particular song (Madonna: 'Like a virgin')

5.2. Enhancing the similarity matrix

Due to variations in the performance within a song at different times, such as articulation, improvisations, changing instrumentation or even key modulation, diagonal stripes in the SSM corresponding to repeated sections are not always clearly visible. On the contrary, the SSM is often very noisy and contains irrelevant line segments that do not correspond to repetitions. Therefore the enhancement of the SSM is a vital process. The aim of this enhancement process is to suppress the unwanted line segments and emphasize the desired ones to the extent possible. This is done by examining each point in the similarity matrix to see if it is a part of a desired line segment or not. A *5x5 kernel* is used for this purpose. When the kernel is centered

at a point, say *(i,j)*, six directional local mean values are calculated along the right, left, upper, lower, upper-right and lower-left dimensions of the kernel and the maximum and minimum values are obtained. Since we are working with the lower part of a time-time similarity matrix, the desired line segments are parallel to the main diagonal. Thus, if the maximum of the local mean values happens to be either the upper-right or lower-left mean, then the point *(i,j)* is considered to be part of a desired line segment and must be emphasized. Otherwise, it is considered noise which tends to form lines along the left, right, upper and lower directions and must be suppressed. The process of suppressing a point is done by subtracting the maximum of the mean values from the point *(i,j)* and the process of emphasizing a point is done by subtracting the minimum of the mean values from the point *(i,j)*. In addition, noise flooring is performed by setting the value of *(i,j)* equal to zero if its new value doesn't exceed a given threshold (in our case *threshold=0.6*). To avoid over-indexing, the SSM is divided into nine different parts, six if we consider only the lower triangular part of the SSM, which are defined as A, B, C, D, H and J (the rest are omitted). As shown in Fig. 5.2 below the enhancement process is carefully calculated for each section.The resulting enhanced SSM is displayed belowin Fig. 5.3.

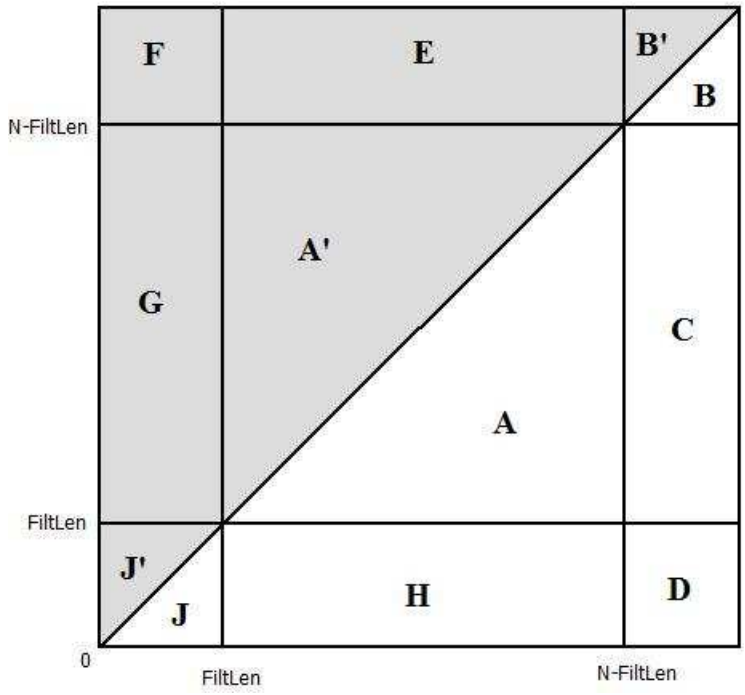


Figure 5.2: SSM division into 9 parts. Gray parts are omitted for the enhancement process

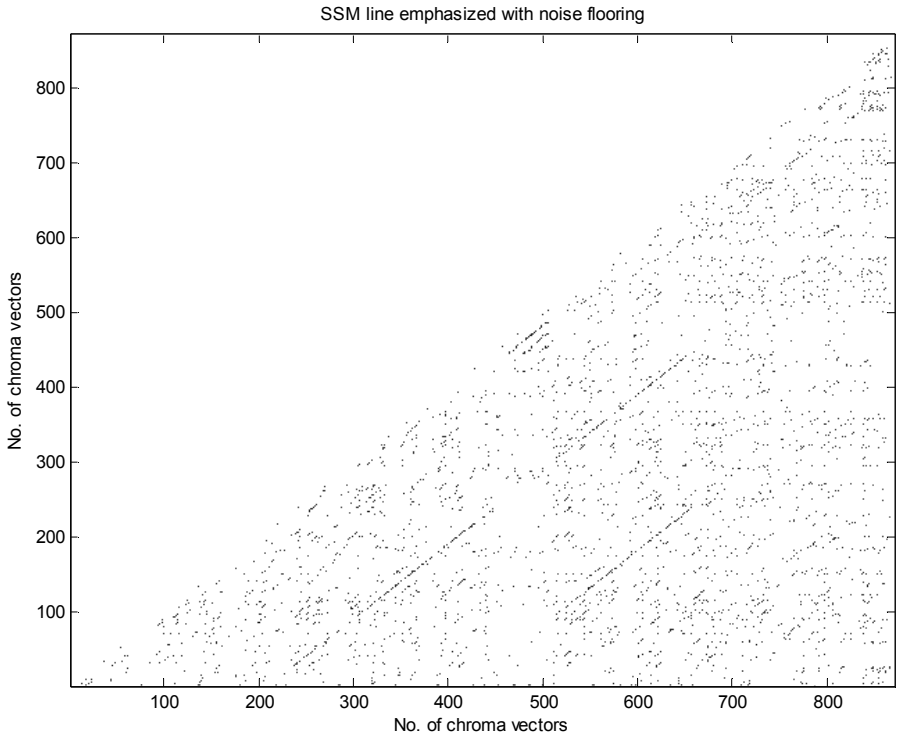


Figure 5.3: Enhanced SSM of the example song

5.3. Detecting repetition

The enhanced SSM is then further processed in order to examine whether a diagonal line in the matrix is likely to contain a line segment or not. To do so, an average in magnitude is calculated along each diagonal corresponding to the *possibility* of a diagonal containing one or more line segments [30][31]. Diagonals which contain line segments will have a higher *possibility* than the ones which do not. This *possibility* for a diagonal containing a line segment is defined as follows:

$$P(k) = \frac{1}{N-k}\sum_{c=1}^{N-k} s(c+k, c) \quad , where\ k = 1, \dots, N-1 \tag{5.2}$$

where*N* is the number of beats in the song.

Thus *P(1)* corresponds to the first diagonal below the main, *P(2)* corresponds to the second diagonal below the main diagonal, and so on. The resulting function *P(k)* contains higher peaks at the diagonals *k* below the main, where the possibility of finding line segments is high. The only disadvantage of this equation is that there is a possibility that some high-similarity values along a diagonal are masked by lower-similarity values at the same diagonal.However this possibility is relatively small and the system performs well as it is.

The method subsequently selects a certain number of diagonals for further processing which correspond to maxima in *P(k)*. The selection of those diagonals is done as follows. First, cumulative noise is removed from the function *P(k)*. This is done by calculating a lowpass-filtered version of *P(k)*, say $P_{lp}(k)$, which is then subtracted from *P(k)*. Next the *smoothed differential* of *P(k)* is calculated. This is provided by the equation:

$$sd(k) = \sum_{w=-K_{size}}^{K_{size}} wR_{all}(t, l+w) \qquad ,where\ K_{size} = 2\ points \tag{5.3}$$

The points where the smoothed differential changes sign from positive to negative correspond to the positions of the peaks in *P(k)*. These peaks are then examined and only peaks higher than a threshold are selected. This threshold is automatically set for each song and is based on a *discriminant criterion* [62].This criterion is based on the number of peaks above and below the threshold and is given by the equation below:

$$\sigma_B^2 = \omega_1\omega_2(\mu_1 - \mu_2)^2 \tag{5.4}$$

where$\omega_1$and ω_2 represent the number of peaks in each class divided by the total number of peaks and $\mu_1$and $\mu_2$correspond to the means of the peak heights in each class. The threshold is selected to be the one maximizing the discriminant criterion.
The sequence contained in Fig. 5.4 below provides an overview of the process by which we select the diagonals to be used for further processing, where *P(k)* stands for the function which contains the possibility of a diagonal containing a line segment.

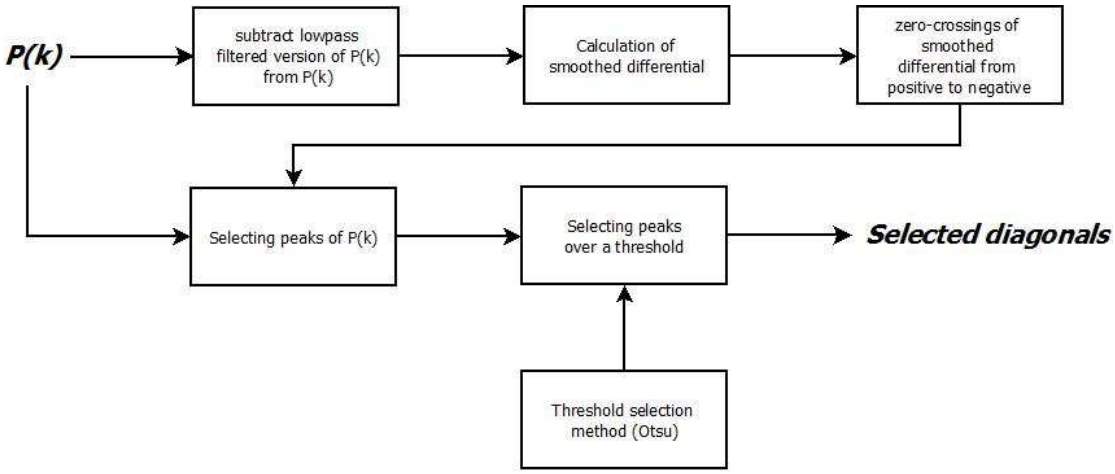


Figure 5.4:Overview of selecting diagonals for further processing

The diagonals of the SSM which are selected for further processing are then smoothed using a low pass filter to 'repair' small imperfections in the diagonals. A new threshold is then calculated in order to detect line segments from the diagonals. All the points contained by the selected diagonals are compared to this threshold. Points which exceed the threshold are set to one; otherwise they are set to zero. The result of this process is *a binarized matrix* where the value one indicates that a particular point is a part of a line segment (repeated section) and zero indicates that it is not. Goto[30] performed another threshold selection based on the Otsu method [62], though in the present work we follow the method used in [31]. We select a threshold so that 20% of all the processed values are considered parts of a repetitive segment. This is done by concatenating the points of all the processed diagonals in a large array which is then sorted in a descending order. We then simply adjust our threshold so that 20% of the array's values are greater than the threshold. The binarized SSM of the example song is displayed in Fig. 5.5.

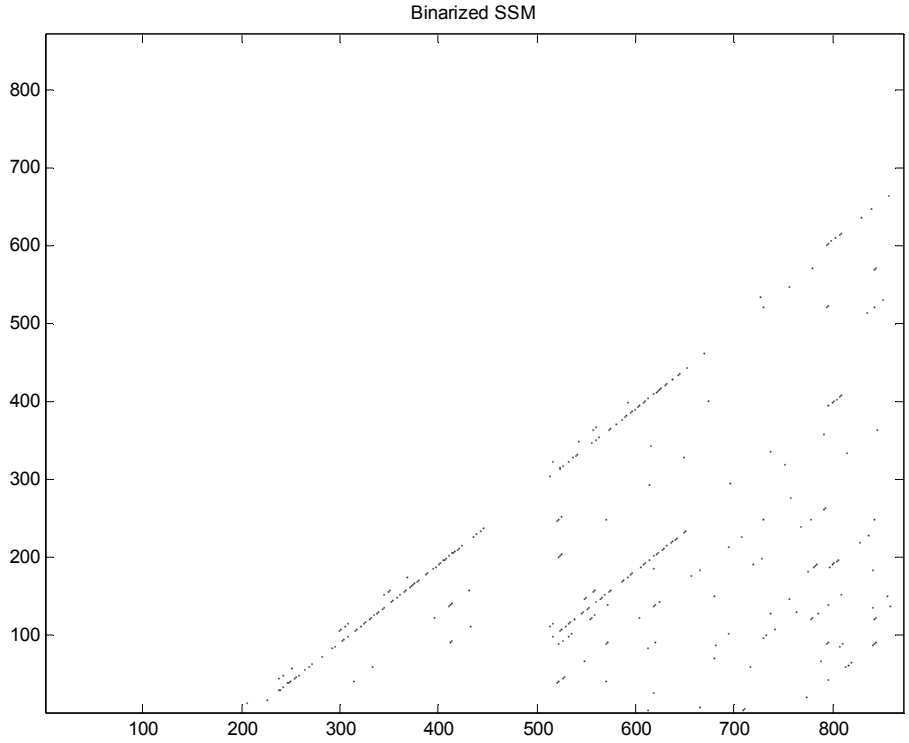


Figure 5.5:Binarized SSM of the example song

The binarized matrix is then processed by an enhancement method presented in [31] which tries to remove gaps of a small number of beats. In other words, for a line segment which is long enough and where most of its values are ones, enhancing this line segment would lead to all the points within the line segment having the value one. This enhancement process is performed only when certain conditions are met. Hence, each point in the binarized SSM, *B(i,j)*, is examined to establish whether an enhancement step is approved. The conditions are: first, at least 65% of the values between *B(i,j)* and *B(i+FiltLen , j+FiltLen)* are ones; second, that *B(i,j)=1*; and third, that either of *B(i+FiltLen-2 , j+FiltLen-2)=1* or *B(i+FiltLen-1 ,j+FiltLen-1)=1* must be true. *FiltLen*is a parameter of the system and is chosen to be 25 points of length. Although [30] did not mention the need for such a process, [31] found it vital to include an enhancement process to the binarized matrix. The resulting enhanced binarized SSM is presented in Fig. 5.6 below:

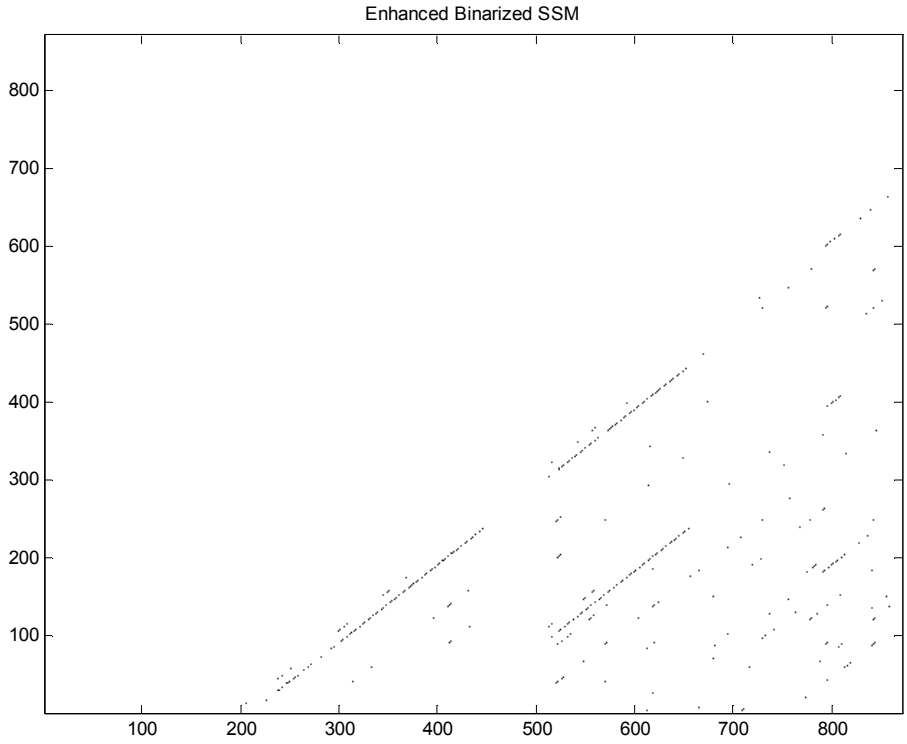


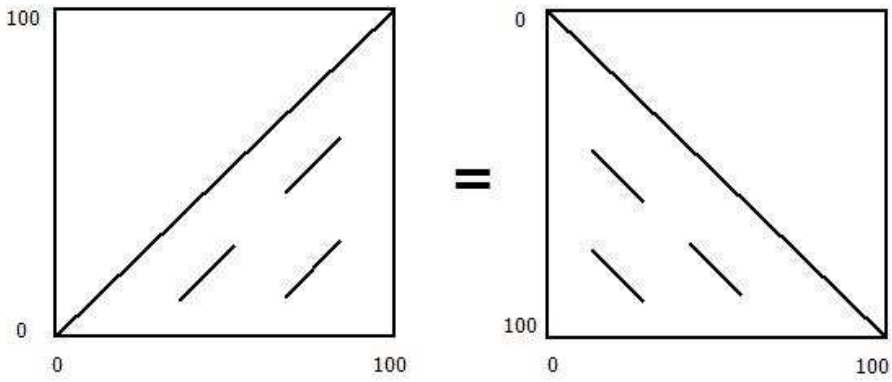
Figure 5.6: Enhanced binarized SSM of the example song

The resulting enhanced binarized matrix is then processed point by point in order to extract and list all the line segments contained in the matrix. We are interested in line segments which have a possibility of corresponding to the chorus section of the song, hence line segments indicating too short repetitions (i.e. shorter than 6.4 seconds) are omitted. Thus, the enhanced binarized SSM is searched for line segments longer than 6.4 seconds, and each line segment found is stored as $\underline{x} = [i, j, i', j']$, where *(i,j)* denotes the diagonal segment's start point and *(i',j')* denotes the segment's end point. The length of the segment's duration is given by:

$$\Delta(\underline{x}) = j' - j + 1 \tag{5.4}$$

In case where no segments longer than 6.4 seconds are found the time threshold is decreased until a segment is detected. Finally, the line segments which have been identified throughout the above process are passed for further processing.

At this point it is noted that by using the matrix' symmetric characteristics we may change the matrix as shown below, without changing its content. The following equations and calculations take into account the right form of the similarity matrix.



5.4. Selecting the desired repetition – chorus

For cases whereby too many line segments are detected, a line removal process is performed to remove lines which are considered to be too close to each other. This is achieved by using a combination of conditions defined by [31], which examine whether two line segments, x_1 and x_2, are consideredto be too close to each other or not. These conditions are:

$$\underline{x}_2(1) \geq \left(\underline{x}_1(1) - 5\right) \quad and \quad \underline{x}_2(3) \leq \left(\underline{x}_1(3) + 20\right) \quad and$$

$$\left|\underline{x}_2(2) - \underline{x}_1(2)\right| \leq 20 \quad and \quad \underline{x}_2(4) \leq (\underline{x}_1(4) + 5) \tag{5.5}$$

All the different pairs of line segments are then searched one by one examining whether the above conditions are satisfied, and all the close segments for each segment are listed. Next, if a line segment happens to have more than three close segments, then these extra segments are to be removed. The only case where an extra segment is not removed is if that particular segment has also more than three close segments by itself.

The remaining line segments are considered the chorus candidates and are subsequently examined with a view to selecting only one line segment to output as the most probable chorus section. This is done by utilizing the novel heuristic scoring scheme used by [31], with the difference that we are working with similarities and not distances. Aspects considered in the final score of each line segment are the following: position of the repetition in the SSM; position of the repetition in relation to other repetitions; average energy and the average similarity in the SSM during the repetition; and the number of times the repetition occurs in the musical piece. In what follows these aspects are considered separately. The line segment with the maximum score is then output as the most likely chorus section.

5.4.1. Scoring the position of the repetition in the SSM

This score examines whether a specified line segment $\underline{x}$ is found to be close to an expected chorus position. In pop music, the chorus is usually located at approximately one quarter, or three quarters, of the song's length. Consequently, we measure the difference between the middle column of the segment and the column corresponding to one quarter of the song's length (Eq. 5.6). In addition we calculate the difference between the middle row of the line segment and the location corresponding to three quarters of the song's length (Eq. 5.7). In so doingwe favor line segments that are found in the lower-left hand corner of the SSM which corresponds to the first occurrence of the chorus which matches to the chorus located at three quarters of the song's length.

$$s_1(\underline{x}) = 1 - \frac{\left|\left(j+\frac{\Delta(\underline{x})}{2}\right) - round\left(\frac{M}{4}\right)\right|}{round\left(\frac{M}{4}\right)} \tag{5.6}$$

$$s_2(\underline{x}) = 1 - \frac{\left|\left(i+\frac{\Delta(\underline{x})}{2}\right) - round\left(\frac{3M}{4}\right)\right|}{round\left(\frac{M}{4}\right)} \tag{5.7}$$

where$\Delta(\underline{x})$ corresponds to the line segment's length given by (4.4), and M corresponds to the number of beats within the song.

5.4.2. Scoring the position of the repetition in relation to other repetitions

For this score we examine the position of a repetition in relation to other repetitions by searching for possible groups of three line segments (*triads*) within the SSM, which may correspond to three repetitions of the chorus. An ideal case of the three segments described is presented in Fig. 5.7. This search is done for each candidate line segment ($\underline{x}_u$). First, the search method examines whether a line segment is found below $\underline{x}_u$, by searching for a line segment $\underline{x}_b$ which satisfies the condition given by $\underline{x}_b(1) > \underline{x}_u(3)$. This condition denotes that there should be no overlap between the rows of $\underline{x}_u$ and $\underline{x}_b$ and that there must be some overlap between the columns of $\underline{x}_u$ and $\underline{x}_b$ (we used an overlap greater than 25%). If such a segment is found, the method then searches a line segment $\underline{x}_r$ located on the right hand side of the below segment and whose rows are overlapping with the rows of the segment $\underline{x}_b$ (by at least 25%).

All the groups of three segments found are listed as a triad, say $\underline{m}_z = [u, b, r]$, and are then scored based on how close to the ideal case each group is. This is done by computing the average value of four different partial scores which are presented below:

1. The first partial score $\sigma_1$examines the end columns of the upper ($\underline{x}_{u(4)}$) and below segments ($\underline{x}_{b(4)}$) and gives a high score when those columns are found to be close to each other. The score $\sigma_1$is given by:

$$\sigma_1(z) = 1 - 2 * \frac{|\underline{x}_u(4) - \underline{x}_b(4)|}{\Delta(\underline{x}_b) + \Delta(\underline{x}_u)} \tag{5.8}$$

2. The second partial score $\sigma_2$considers the vertical alignment of $\underline{x}_u$and $\underline{x}_b$, whereby if the start column of the below segment is under the upper segment then the score gets a value of 1. Otherwise, if the below segment starts before the upper segment, it will get penalized and obtain a score smaller than 1. This scoring is handled by the equationbelow:

$$\sigma_2(z) = \begin{cases} 1 - \frac{\underline{x}_u(2) - \underline{x}_b(2)}{\Delta(\underline{x}_b)} & if\ \underline{x}_b(2) < \underline{x}_u(2) \\ 1 & otherwise \end{cases} \tag{5.9}$$

3. The third partial score σ_3 examines the length of the below segment $\underline{x}_b$ and the right segment $\underline{x}_r$ and awards a higher score if $\underline{x}_b$ and $\underline{x}_r$ are both of equal length. This can be implemented as follows:

$$\sigma_3(z) = 1 - \frac{|\Delta(\underline{x}_r) - \Delta(\underline{x}_b)|}{\Delta(\underline{x}_b)} \tag{5.10}$$

4. Finally, the fourth partial score σ_4 depends on the difference in the position of the below and right segments and is given by:

$$\sigma_4(z) = 1 - \frac{2*\min(|\underline{x}_b(1) - \underline{x}_r(1)|, |\underline{x}_b(3) - \underline{x}_r(3)|)}{\Delta(\underline{x}_b) + \Delta(\underline{x}_r)} \tag{5.11}$$

The final score to be awarded to each group of three segments found is then calculated by taking the average value of the four partial scores described above and given to the below segment $\underline{x}_b$. The score is given to the below segment, as this segment's length is often closer to the exact length of the chorus. Since $\underline{x}_b$ may be a below segment for many groups of threes, the maximum value is retained and stored as its value. If no groups of three are found, then $s_3(\underline{x}_b) = 0$.

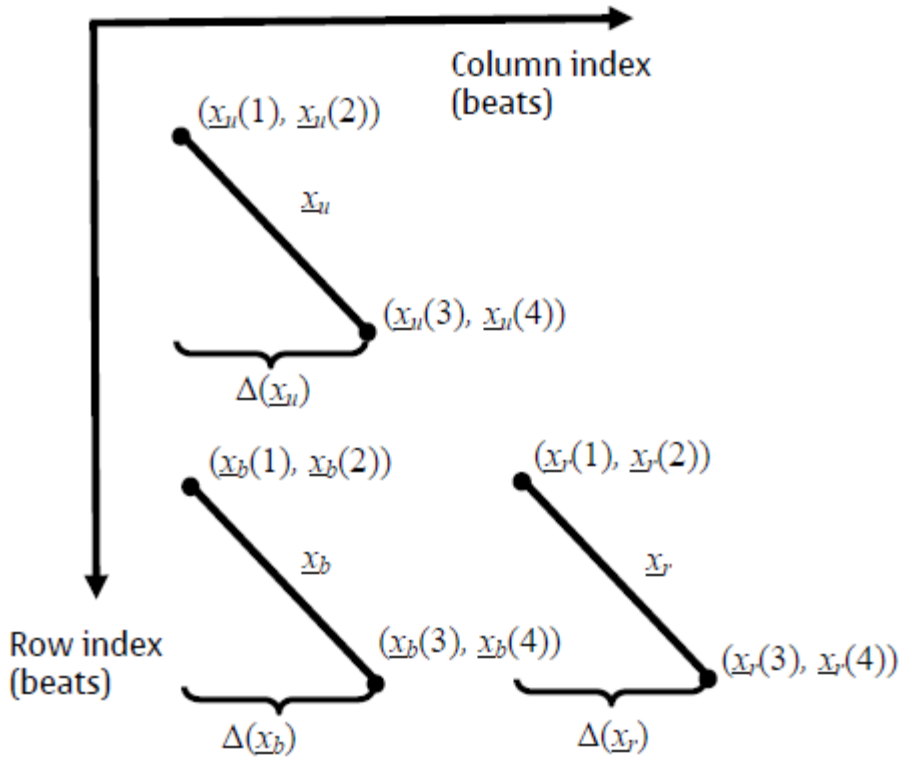


Figure 5.7: Ideal case of the position of three line segments, corresponding to three repetitions of the chorus

5.4.3. Scoringthe average energy of the line segment

For this score, [31] measured the average logarithmic energy with the use of the *zerothcepstral coefficient* over the segment. Since in this work we are not using MFCC features, we approximated it by computing the vertical sum of each frame from the beat-synchronous representation of the song across frequency and then calculated the mean of that resulting function, and denoted the mean value as E_{av}.Next, we repeat the process but only for the frames defined by the line segment's columns and denoted it as E_{seg}. The ratio E_{seg} / E_{av} is taken to be the score (Eq. 5.12). The score thus obtained favors segments which have high energy, often a characteristic of chorus sections.

$$s_4(\underline{x}) = \frac{E_{seg}}{E_{av}} \tag{5.12}$$

5.4.4. Scoring the average similarity of the line segment

This score takes into consideration the average similarity in the SSM and the average similarity in the line segment. The greater the similarity during the line segment, the higher the chance that it corresponds to a chorus section. Consequently, if SSM_{avg} is the average similarity of the SSM and LS_{median} is the median similarity value of the line segment, then the score is taken to be:

$$s_5(\underline{x}) = 1 - \frac{1-LS_{median}}{1-SSM_{avg}} \tag{5.13}$$

5.4.5. Scoring the number of times the repetition occurs

This score looks for line segments located above or below the examined line segment. If a line segment ($\underline{x}_2$) is found to be on top of, or below, the examined segment ($\underline{x}_1$) it is considered an indication that this repetition is repeated more than once. The condition which examines whether two line segments ($\underline{x}_1$and $\underline{x}_2$) are on top of, or below, each other is given by:

$$\left|\underline{x}_1(2) - \underline{x}_2(2)\right| \leq 0.2 * \Delta(\underline{x}_2) \;\; and \;\; \left|\underline{x}_1(4) - \underline{x}_2(4)\right| \leq 0.2 * \Delta(\underline{x}_2) \qquad (5.14)$$

Hence, a count is performed for each segment to establish how many segments are above or below it, and this count is denoted as N_{seg}. The maximum count obtained for all segments is denoted as N_{seg_max}. The score is then taken to be:

$$s_6(\underline{x}) = \frac{N_{seg}}{N_{seg\_max}} \qquad (5.15)$$

5.4.6. Chorus selection

The above partial scores are then combined in order to obtain a final score *S* which will describe the likelihood of a line segment being the chorus. A. Eronen[31] combined the scores as follows:

$$S(\underline{x}) = 0.5 * s_1(\underline{x}) + 0.5 * s_2(\underline{x}) + s_3(\underline{x}) + 0.5 * s_4(\underline{x}) + s_5(\underline{x}) + 0.5 * s_6(\underline{x}) \qquad (5.16)$$

In the case where no groups of threes where found, which means that for each segment $s_3(\underline{x}) = 0$ is true, the segment which has the maximum value *S* is chosen to be the chorus segment. Otherwise, the selection is performed among the line segments that have $s_3(\underline{x}) \neq 0$.At this point a search for the exact pin point temporal location of the chorus might be implemented. Such a search method could be achieved by making use of image processing techniques such as 2D kernel filters as presented in [31]. However this processing step is omitted to avoid further complication.

CHAPTER 6

Implementation

The present chapter describes the experimental part of this work in which the systems of AFP matching, ACS identification and ACD are implemented, reflecting the models described in Chapters 3, 4 and 5. Implementation is performed mainly using the MATLAB programming language with the intension of examining the systems' performance and accuracy. To do so, each system is handled and examined separately.

6.1. Experimental Setup

6.1.1. AFP matching

With a view to examining an audio-fingerprinting based music matching system we make use of the MATLAB code provided by D. Ellis in [53]. This implementation is a close approximation to the landmark-based approach used by Shazam [52] presented in section 3.3. A music collection is formed selecting random pieces of our own music collection. A total of 1062 songs were selected representing a wide range of musical genres. As our implementation does not handle .mp3 formats, each and every song was converted into a .wav format using the *lame decoder* [63], which is an open source tool. To avoid converting each song individually an executable (.exe) file named *convertmp3stowavs* was created using MATLAB's compiler, which carries out the conversion for all files automatically. Another executable file was created in the

same way in order to overcome an 'incorrect chunk size information' issue while reading the data, and was denoted as *fixWavs*. These songs are then placed at the system's root directory.

Landmark pairs are formed in a 4-tuples basis corresponding to start-time, start-frequency, end-frequency and time-difference using the function *find_landmarks*. These landmarks are then quantized using the function *landmark2hash*, and the inverse action is performed using *hash2landmark*. Songs are integrated into the system's music database by using the function *add_tracks*, whereby each song (found in the root directory) is converted into landmarks, which in turn are converted into hashes and are stored within an 'inverted index' hashtable representing the music database. Next, given an audio query, a search for a possible match is done throughout the music database using the function *match_query*. A successive match is depicted in Fig. 6.1 below, where the green connected dots are referring to matching landmark pairs in the two audio recordings. This particular example demonstrates a match of an audio snippet which is found to be an excerpt of the song 'Slogans' by Bob Marley at precise 59.904 sec within the song.

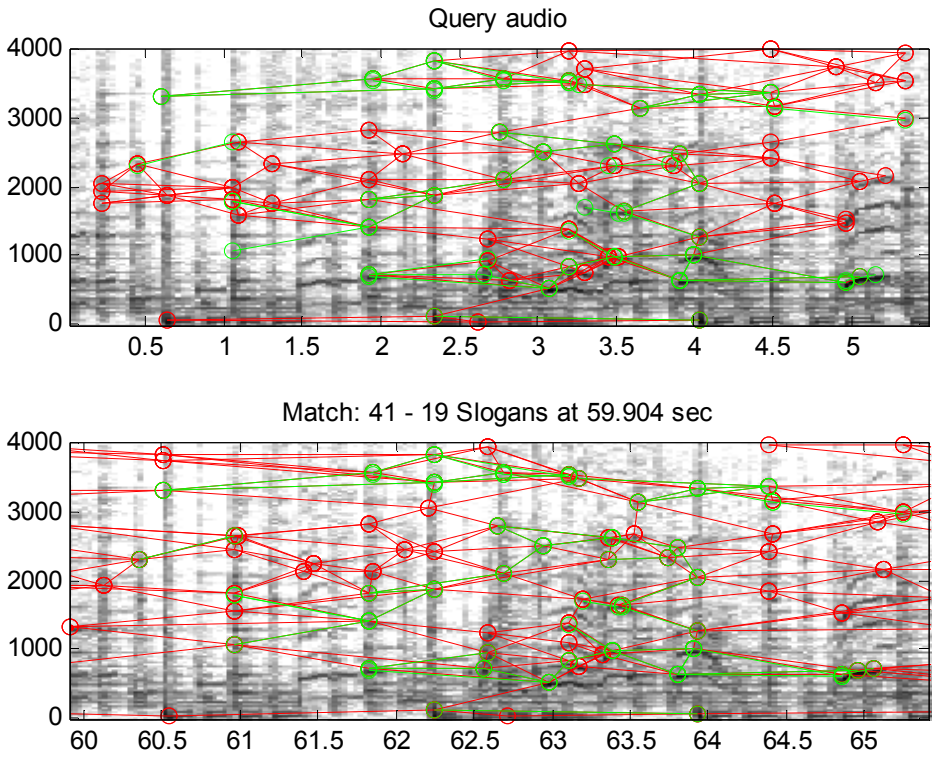


Figure 6.1: Example of the AFP matching system's output

To make this program more interactive a *graphical user interface*(*GUI*) was developed using the ECLIPSE software development environment. The GUI allows a user to either record a new audio recording with the use of a microphone pressing the *'Capture sound from mic...'*- and *'Stop recording...'*-buttons (Fig. 6.2), or to load an already existing audio recording into the system using the *'Open a File...'*-button (Fig. 6.3). As soon as an audio recording is loaded or captured it is played back to the user. The user can then either select or record a new audio recording, or proceed in finding a match for his previous recording. This is done by clicking on the 'Match your clip.'-button.

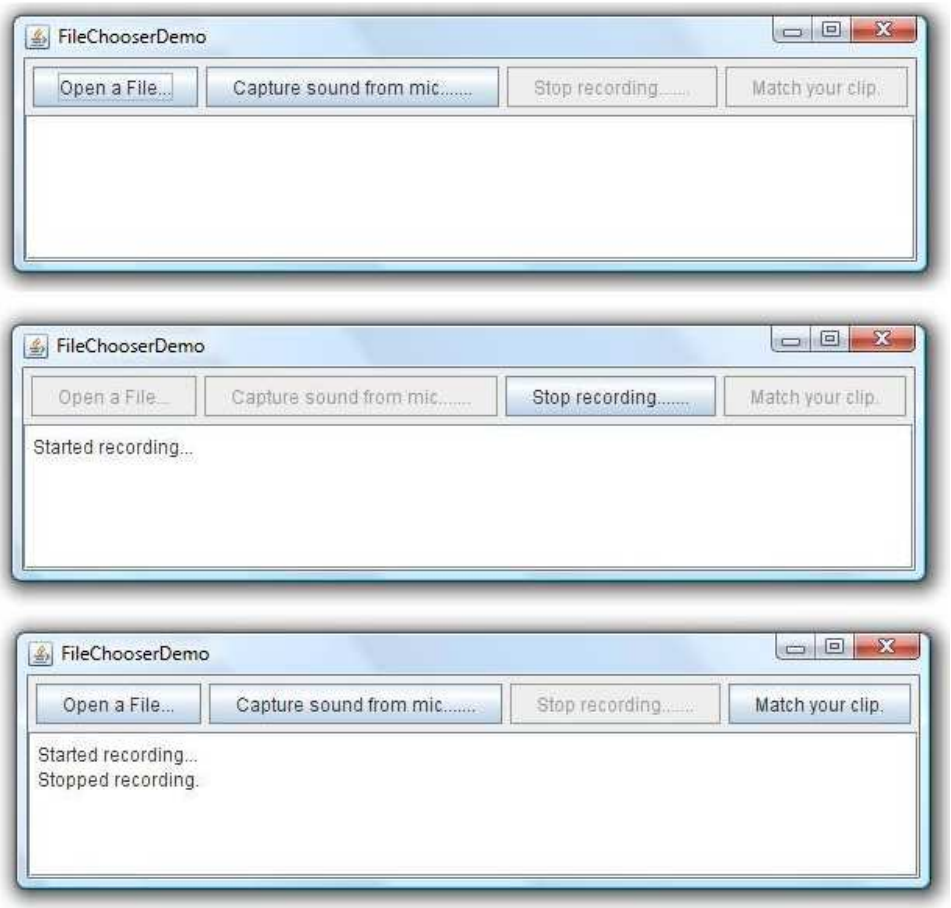


Figure 6.2: Recording a new audio recording using the GUI

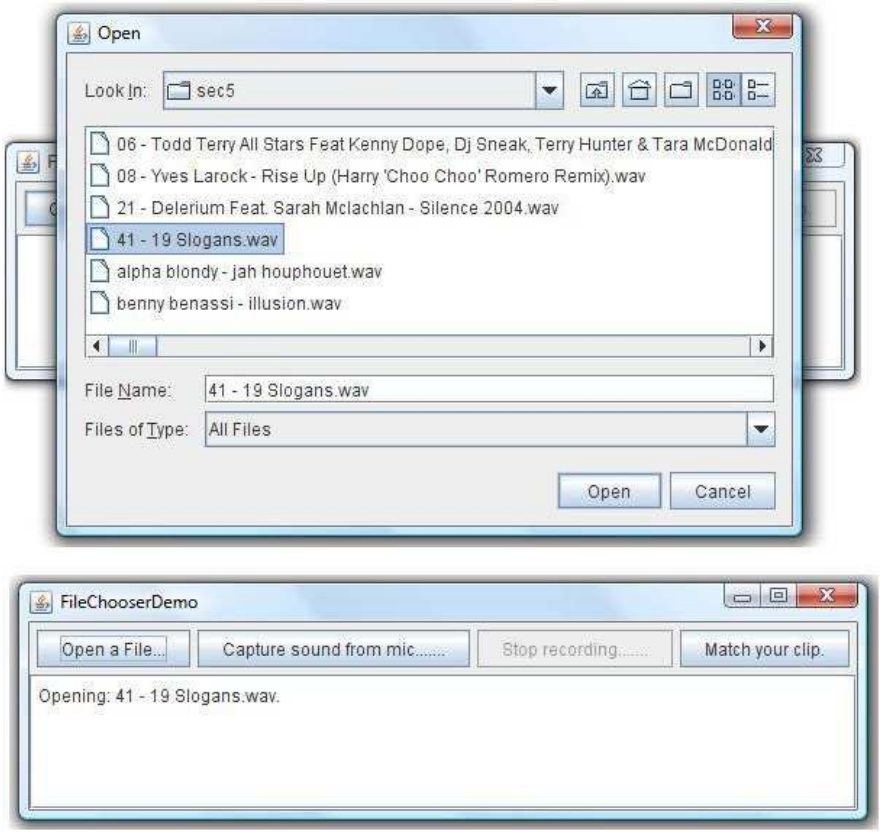


Figure 6.3: Opening an existing audio recording using the GUI

Clicking on the *'Match your clip.'*-button is equivalent to running the MATLAB script described earlier. To link the ECLIPSE and MATLAB environments we make use of an open source tool called *JAMAL* (JAvaMAtlab Linking). This tool is based on a client-server approach which makes it possible to call MATLAB functions from Java programs without reading and writing to a temporary file [64]. In doing so, the results of the MATLAB function are presented in the GUI's text box, as seen in Fig. 6.4.

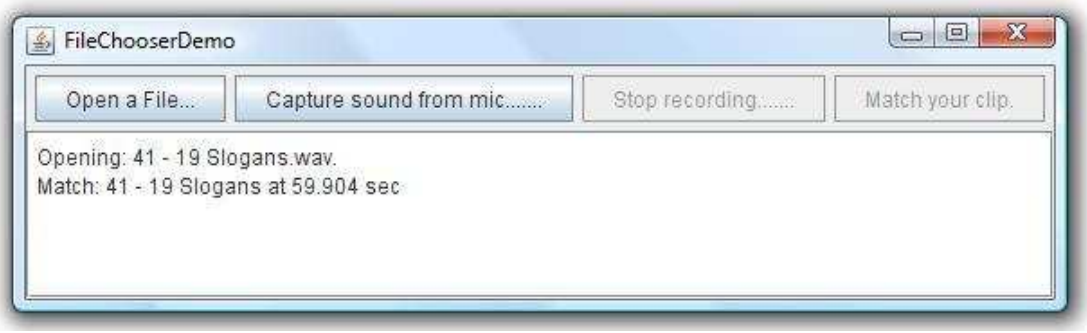


Figure 6.4: Matching a clip using the GUI

6.1.2. ACS identification

The ACS identification system was implemented following the steps described in Chapter 4. A chromagram extraction function, as well as a beat-location extraction, an averaging function of these two and a similarity measure function were created using MATLAB. These functions where denoted as *chromagramExtr, beatLocationExtr, beatSynchChromRepr*and *search4CSinDB* respectively. The similarity function has the ability to handle three different approaches, namely the one described by Ellis in [14], its upgrade in [13] and a combination of the latter case usingmultiple tempi. Hence, the function is given a parameter which dictates which approach will be followed.

It should be pointed out here that currently there exists no standard music collection available which is used for evaluation purposes. A close approximation to an 'all-modulation-including' dataset would be the one used during the MIREX contest [2]. This data evaluation set, however, is being kept secret in order to prevent over-tuning of approaches. Consequently as an evaluation dataset we use the so-called '*covers80*' cover song dataset [65] which is a musical collection consisting of 80 different cover-sets, each having two different renditions of each song. This gives a total of 160 music tracks covering a wide range of different modulation in several musical features.

Next, two experiments are carried out. The first creates two lists of 15 songs. The first list includes the first version of 15 different songs, whilst the second contains their alternate versions. A search is performed for each song in the first list searching for its cover version in the second. This results in a total of 15 queries. A correct match of the system is defined as the case whereby the song that is outputted as the one having the highest similarity happens also to be the cover version of the particular query. The number of correct matches amongst the 15 queries is then stored. This experiment is conducted 250 times and the mean of correct matches is outputted as an evaluation measure. The second experiment takes into account all 160 songs which are also divided into two lists of 80 songs, each containing one version of each pair of songs. Next, for each song in the first list a search is performed within the second to find its cover. The system's *rank* defines the tolerance of what is considered to be a

correct match. If *rank=N*, then if for a query the correct cover is found to be amongst the first *N* results, it is considered to be a successful match. In addition the system's *mean reciprocal rank* (*MRR*) is calculated. MRR is simply the average of the ranks corresponding to the correct match. This measure can be given by the equation below:

$$MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i} \tag{7.1}$$

6.1.3. ACD

With a view to analyzing the song's structure an automatic audio summary (thumbnail) extraction system has been implemented, following the steps described in Chapter 5. Since for popular music this usually corresponds in identifying the chorus of a song, the system is denoted as anautomatic chorus detection system respectively. For some songs, though, it is not possible to find the chorus, as a song might not have one, or might have one which is not clearly standing out to be identified. In addition a chorus might not be anaccurate representation for the song as it might be too short or not clearly audible. In such cases, the verse or even the basic short music theme often identified in an intro can be more informative than the chorus. For the above reasons this system is thought of providing the most representative part of the song as a person would pick it out, trying to put more weight on selecting the chorus part if such one exists.

For the implementation of the system described above, three basic functions were created: *createSSM*, *enhanceSSM*, and *lineSelection*. Function *createSSM*generates the self-similarity matrix of the song, where function *enhanceSSM*enhances it and lists all the line segments found. The line segments are then passed to function*lineSelection* where they are examined in order to select the segment corresponding to the chorus.

Further tuning was performed to the system in order to retrieve not one, but two possible chorus candidates. The second chorus candidate was chosen to correspond to the line segment with the next maximum score value which also has no temporal overlap with the first line segment. If such a condition was not met for any of the line

segments, then it was omitted and the segments with the second largest value was selected. We believe that this condition improved the system's performance, since if the correct part was not selected in the first part, usually it was selected in the second.Moreover, when selecting the two chorus candidates we increased the segments duration by 2 seconds by moving the end-point of the segment. This was performed following the observation that the chorus' section identified cut off approximately 2 seconds from the end. This effect could have been caused by smoothing functions used in the algorithm.

It was observed that by using the ACD system described in this study in order to find the most representative part within a song resulted to acceptable results.For songs with a straight-forward music structure, such as clearly distinguishable intro, verse, chorus, bridge and outro sections, the system could trace the chorus section in most of the cases. If no chorus was present in the song, then usually the verse or the basic tune of the song is returned as the most representative part.Only on very few occasionsdid the system select a poor part of the song as being most representative. In the case where a chorus was present in the song but the system outputted another part of being more representative, we observed that the system's choice is considered reasonable, as in many of those cases the catchiest part of the song is the verse or the melody in the intro or bridge,rather than the chorus itself.

Due to all the above considerations and the fact that the investigation of a music structure analysis system is of a preliminary nature, no exhaustive evaluation of the described system was performed, however some examples and comments are presented in section 6.2.3.

6.2. Evaluation Results

6.2.1. AFP matching

In order to evaluate the AFP matching system's performance we randomly chose 50 music pieces out of the music collection. For each piece three segments of different

duration were extracted. The segments duration were 5, 10 and 15 seconds respectively. Each of these segments where used as queries in the AFP matching system in order to retrieve their origin music piece. Without the addition of any noise, even for the 5 seconds segments we achieve a 100% accuracy of music matching. Next, several types of noises were added to the segments in order to evaluate the system's robustness to noise. The noise types where selected from a noise database called *NOISEX* [66]. The noise types that were used are: *speech babble, destroyer engine room noise, factory floor noise 1, car interior noise (Volvo), white noise, pink noise, F-16 cockpit noise and leopard (tank) noise*. These noises were added to each and every music segment achieving a *signal-to-noise ratio* (*SNR*) equal to -15 dB, -12 dB, -9dB, -6 dB, -3 dB, 0 dB, 3 dB, 6 dB, 9 dB, 12 dB and 15 dB respectively. This process is implemented in the function denoted as *addNoise*. It is understood that in order to add noise of a specific SNR amount to a signal we scale the signal appropriately. The equation providing the SNR value is given below:

$$SNR = 10\log_{10}(\frac{\sigma_{sig}^2}{\sigma_{Noise}^2}) \tag{6.1}$$

whereσ_{sig}^2 corresponds to the variance of the initial signal and σ_{Noise}^2 to the variance of the noise signal. Rearranging the above equation results in:

$$\sigma_{sig}^2 = \sigma_{Noise}^2 10^{\frac{SNR}{10}} \tag{6.2}$$

Consequently, our scaled signal is given by:

$$scalsig(t) = \frac{\sigma_{Noise}\sqrt{10^{\frac{SNR}{10}}}}{\sigma_{sig}} sig(t) \tag{6.3}$$

Finally the output signal with the required amount of SNR is given by:

$$noisesig(t) = scalsig(t) + Noise(t) \tag{6.4}$$

The possibility of correct recognition rate of the pre-described evaluation are shown in table 6.1 below.

NOISE \ SNR level (dB)		-15	-12	-9	-6	-3	0	3	6	9	12	15
Type	Segment's duration											
Babble	5 sec	0	0,04	0,12	0,24	0,38	0,5	0,66	0,82	0,94	1	1
	10 sec	0,02	0,08	0,12	0,22	0,48	0,74	0,86	1	1	1	1
	15 sec	0,02	0,14	0,18	0,42	0,72	0,9	0,96	0,98	1	1	1
Destroyerengine	5sec	0	0,02	0,08	0,18	0,36	0,44	0,54	0,68	0,7	0,84	0,94
	10 sec	0	0,08	0,12	0,32	0,66	0,76	0,9	0,94	1	1	1
	15 sec	0,04	0,1	0,22	0,52	0,7	0,86	0,98	0,98	1	1	1
Factory1	5 sec	0	0	0,06	0,04	0,18	0,38	0,56	0,66	0,76	0,84	0,88
	10 sec	0	0	0,04	0,14	0,28	0,56	0,76	0,9	0,98	1	1
	15 sec	0	0	0,06	0,18	0,42	0,68	0,92	0,94	1	1	1
Volvo	5 sec	0,94	0,98	0,98	0,98	0,98	0,98	0,98	0,98	1	1	1
	10 sec	1	1	1	1	1	1	1	1	1	1	1
	15 sec	1	1	1	1	1	1	1	1	1	1	1
White	5 sec	0	0	0,02	0,06	0,16	0,32	0,46	0,62	0,68	0,78	0,82
	10 sec	0	0	0,06	0,14	0,3	0,5	0,64	0,82	0,94	1	1
	15 sec	0	0,02	0,1	0,22	0,4	0,6	0,82	0,96	0,96	1	1
Pink	5 sec	0	0,02	0,04	0,08	0,24	0,4	0,62	0,76	0,78	0,8	0,94
	10 sec	0,02	0,04	0,06	0,18	0,46	0,6	0,84	0,92	0,94	1	1
	15 sec	0,02	0,04	0,12	0,3	0,54	0,76	0,9	0,96	0,96	1	1
F16	5 sec	0	0,02	0,1	0,12	0,26	0,34	0,64	0,7	0,88	0,92	0,94
	10 sec	0	0,06	0,14	0,24	0,48	0,66	0,84	0,92	1	1	1
	15 sec	0,02	0,08	0,2	0,44	0,62	0,82	0,96	1	1	1	1
Leopard	5 sec	0,66	0,78	0,84	0,94	0,98	0,98	1	1	1	1	1
	10 sec	0,9	0,96	0,96	0,98	1	1	1	1	1	1	1
	15 sec	0,96	0,98	1	1	1	1	1	1	1	1	1

Table 6.1: AFP matching evaluation results

As can be observed from the experimental results we may conclude that the system performs satisfactorily, even in the presence of noise. In particular, in cases where either no noise or noise with an amount of SNR equal to 12 or 15 dB is added in segments of 10 seconds or longer, the accuracy approximates 100%. In general it may be argued that the higher the degree of noise, the lower the system's accuracy in finding the correct origin song, with the exceptions of car interior (Volvo) and leopard (tank) noises which have no effect on accuracy results. Also, as expected, longer-time segments help counterbalance the impact of noise in respect to accuracy. This can be seen in the graph below, indicating how longer-time segments improve performance in accuracy by using the example of babble noise which is closer to 'real-life' situations.

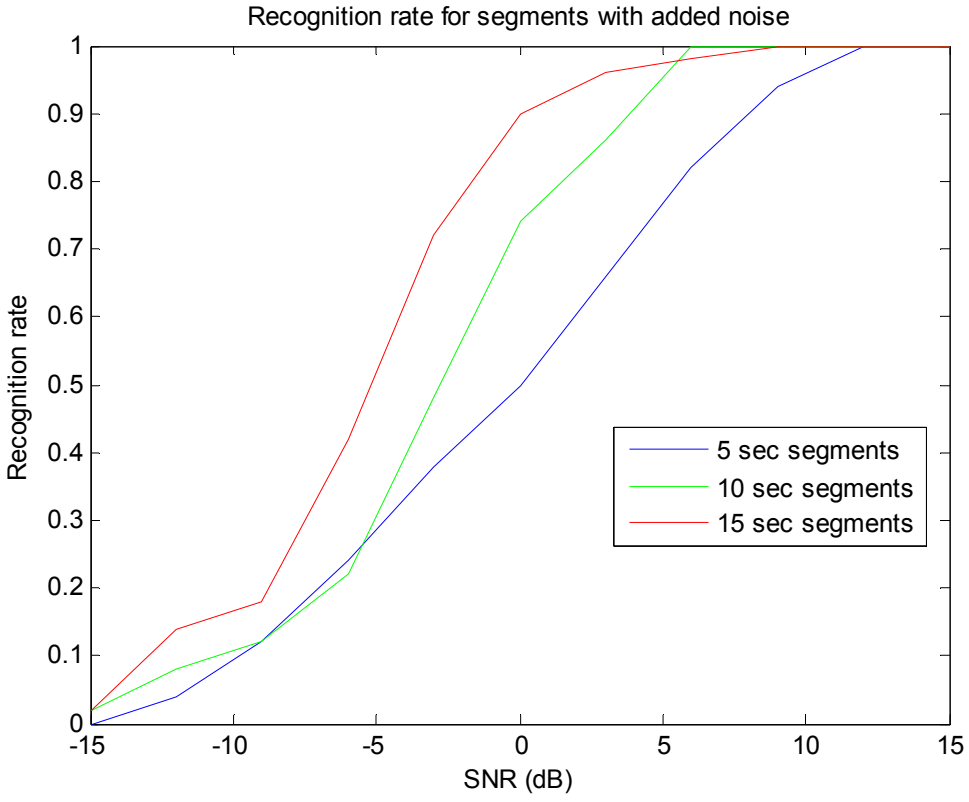


Figure 6.5: Recognition rate for segments of 5, 10 and 15 seconds with additive babble-noise

6.2.2. ACS identification

In section 6.1.2 we described two experiments performed in order to evaluate the ACS identification system's performance. The results of those experiments are shown in table 6.2 below.

		1 TEMPO		2 TEMPO
		120 BPM	240 BPM	
15 cover sets	rank=1	8,32 / 15	7,88 / 15	9,936 / 15
80 covers ets	rank=1	40 / 80	35 / 80	42 / 80
	rank=3	43 / 80	39 / 80	51 / 80
	rank=5	44 / 80	45 / 80	55 / 80
	rank=10	48 / 80	48 / 80	59 / 80

Table 6.2: Results of ACS identification system

Examining the evaluation results we draw the conclusion that the system described throughout this work, while not perfect, has an acceptable performance. This in turn implies that the use of beat-synchronous chroma features is a correct method of approaching the implementation of an ACS identification system. Moreover, weobserve that better results are obtained when using chroma features averaged by beat times extracted from a beat-tracker biased around 120 BPM than from one biased around 240 BPM. Using the information of both tempi, though, results in a slight improvement of the system's performance.

For the first experiment, where cover songs are searched within a music collection of 15 tracks, an average of 9.9 correct matches is achieved. This number of correct matches greatlyexceeds the case of random matching, since guessing the right cover song would give an average of only 1 correct match. This verifies the above implication.

For the second experimentthe system resulted in a 52.5% accuracy considering only the first rank. We also examined the accuracy for different ranks. We observe that by considering more ranks as correct results we get a higher performance. As music similarity is difficult to measure in some examples of cover songs due to their radical modifications, the correct cover may not be selected as the most similar song (rank=1).Considering more ranks as correct results may result to higher accuracy. The mean reciprocal rank of the system tested on the music collection of 80 cover sets is equal to: *MRR=0.6070*, which indicates the rank of the correct match based on the ordered returns of similarity.

However, sometimes cover versions are simply too different to be considered similar based on the features described in this work. As the cover80 dataset covers a wide range of modulations between cover versions, sometimes to a radical degree, we believe that the results provided by the system tested on this dataset may be considered as satisfactory.

6.2.3. ACD

In this section a rough evaluation of the ACD system is given based on a subset of the covers80 music dataset (125 out of 180 songs), containing the music tracks which were considered as easy to comment. Extracted chorus parts were divided into 3 categories: a good match, a medium match and a low match (Table 6.3). Consequently 6 query-examples are given into the system and its outputis presented.

Good match (±5 sec) (in case of no chorus in song, the verse = good match)	**Medium match** (e.g. half-chorus found / or verse identified instead of chorus)	**Low match** (e.g. very short snippet that doesn't make sense,basic tune or solo part)
85 cases	28 cases	12 cases

Table 6. 3: rough evaluation results

The example query-songs presented in this section are listed in Table 6.3.

Name of artist	Song's title	Genre	Duration
Steve Miller Band	Abracadabra	popular music	3:42
Aerosmith	I don't want to miss a thing	slow rock music	4:57
Run dmc	It's tricky	hip hop music	3:03
Beatles	Let it be	pop/roch music	3:50
The Corrs	Little Wings	celtik/pop/folk	5:05
Depeche Mode	Never let me down again	popular music	4:48

Table 6.4: Example songs tested for the ACD system

Each and every song of the above table-list was given to our ACD system as input and 2 segments were selected as the most probable chorus representations. Consequently we compare the times outputted by the system with the chorus start and end times extracted by manually segmenting each song.

Abracadabra by Steve Miller Band:

This particular song belongs to the *popular dance music* genre. It has a straight-forward structure with notable chorus sections repeating several times. The verses are also considered similar to each other, making the selection of the chorus more complicated. The song's structure is displayed in Table 6.4.

Abracadabra	**times (mm:ss)**
Intro	00:00 - 00:14
Verse 1	00:15 - 00:29
Verse 2	00:29 - 00:43
Chorus 1	00:43 - 00:57
Verse 3	00:58 - 01:11
Chorus 2	01:12 - 01:26
Verse 4	01:27 - 01:41
Verse 5	01:42 - 01:56
Chorus 3	01:56 - 02:10
Verse 6	02:11 - 02:29
Bridge	02:30 - 03:00
Chorus/Outro	03:01 - 03:42

Table 6. 5: Song parts of "Abracadabra" by Steve Miller Band

The song is then represented by a 12x475 beat-synchronous chromagram which has one single chroma-vector for each beat of the song. The two outputted chorus start and end times extracted for the given song by our ACD systems are:

Chorus 1: 43.2 sec – 62.4 sec Chorus 2: 17.4 sec – 36 sec

It is observed that the first chorus start and end times extracted by the system includes the complete first chorus of the song. The exact start and end times extracted from the system differ from the manually selected chorus times by only a few seconds. The second chorus part, as it is selected to not overlap with the first chorus-times represents a part of the song which is found between verse 1 and verse 2, a part which also repeats in the song.Having correctly identified a chorus section with the first extracted possible chorus-part times is considered a satisfactory result.

I don't want to miss a thing by Aerosmith:

This song is found under the genre *slow rock* or *blues rock* music. It has a well-defined chorus which repeats three times throughout the song. The song is then represented by a 12x596 beat-synchronous chromagram. The manual segmentation of the song is depicted in Table 6.5.

The two outputted chorus parts are:

Chorus 1: 73.4 sec – 100.1 sec

Chorus 2: 131.9 sec – 157.2 sec

I don't want to miss a thing	times (mm:ss)
Intro	00:00 - 00:30
Verse 1	00:31 - 01:11
Chorus 1	01:12 - 01:41
Verse 2	01:42 - 02:15
Chorus 2	02:15 - 02:39
Bridge	02:40 - 03:14
Chorus 3	03:15 - 03:54
Outro	03:55 - 04:57

Table 6. 6: Song parts of "I don't want to miss a thing" by Aerosmith

For this example song, both extracted possible chorus parts correspond to two chorus repetitions within the song ('Chorus 1' and 'Chorus 2'). The start and end times extracted differ only for a couple of seconds. Having identified not one, but two choruses within the song is considered as a highly satisfactory result.

It's tricky by Run-D.M.C.:

This song belongs to the category of *hip-hop/rap* music. It has many repetitions including choruses, verses as well as short-time melodic or rhythmic patterns. The song has four repetitions of the chorus which are distinguishable through the fact that these parts are more "energetic". The musical form of the song is depicted in Table 6.6. The song is represented by a 12x391beat-synchronous chromagram.

It's tricky	times (mm:ss)
Intro	00:00 - 00:09
Chorus 1	00:10 - 00:24
Verse 1	00:25 - 00:39
Chorus 2	00:40 - 00:54
Verse 2	00:55 - 01:09
Chorus 3	01:09 - 01:24
Interlude	01:25 - 01:38
Verse 3	01:39 - 02:09
Chorus 4	02:10 - 02:24
Verse 4	02:24 - 02:39
Outro	02:40 - 03:03

Table 6. 7: Song parts of "It's tricky" by Run D.M.C.

The outputted chorus parts from our ACD system are:

Chorus 1: 68.5 sec – 84 sec

Chorus 2: 18.7 sec – 30.9 sec

It is observed that the first chorus part corresponds to the 'Chorus 3' part of the song whereas the second outputted chorus corresponds to a segment between the first chorus and the first verse. Having correctly identified a chorus section with the first extracted possible chorus start and end times is considered a satisfactory result.

Let it be by The Beatles:

This particular song belongs to the genre of *popular rock* music. It has three choruses that stand out and one more chorus repeating at the end of the song followed by the outro. Moreover it has three verses that are also similar with each other and a bridge part which contains a solo. The times of each part are shown in the Table 6.7 below. The song is then converted into a 12x534 beat-synchronous chromagram.

The chorus parts extracted from our ACD system are:

Chorus 1: 30.4 sec – 50.8 sec

Chorus 2: 14.2 sec – 26 sec

Let it be	times (mm:ss)
Intro	00:00 - 00:12
Verse 1	00:13 - 00:37
Chorus 1	00:38 - 00:51
Verse 2	00:52 - 01:17
Chorus 2	01:18 - 01:44
Bridge (+ solo)	01:45 - 02:25
Chorus 3	02:26 - 02:40
Verse 3	02:41 - 03:08
Chorus/outro	03:09 - 03:47

Table 6. 8:Song parts of "Let it be" by The Beatles

The first extracted chorus surrounds the 'Chorus 1' part of the manual segmentation. The reason for including 8 seconds from 'Verse 1' as well in the representation is due to the similarity of the verses. We conclude that the system provided reasonable results for the particular example.

Little Wings byTheCorrs:

This song belongs to the *Celtic/pop/rock* music. It is an example of a song which does not contain a chorus part. Instead it has two different instrumental parts with two identical verses which repeat themselves throughout the song. The times of the different parts are presented in the Table 6.8. It was considered a challenge whether the system would pick the instrumental part as the most representative for the song or if it would select the identical verses. The song is consequently represented by a 12x717 beat-synchronous chromagram matrix.

Little Wing	times (mm:ss)
Intro	00:00 - 00:33
Verse 1	00:34 - 01:05
Verse 2	01:06 - 01:38
Instrumental 1	01:39 - 02:10
Verse 1	02:11 - 02:42
Verse 2	02:43 - 03:15
Instrumental 2	03:16 - 03:49
Instrumental 1	03:50 - 05:05

Table 6. 9: Song parts of "Little Wings" by The Corrs

The outputted chorus parts from our ACD system given the particular song are:

Chorus 1: 102.2 sec – 129.2 sec

Chorus 2:77.1 sec – 88.7 sec

Interestingly the first extracted chorus part corresponds to the instrumental part within the song. This is considered as reasonable as for the particular song the instrumental part would be characterized as the most representative one.

Never let me down again by Depeche Mode:

This song is considered to belong in the *popular* music genre. It is a song with a not so straight-forward song structure as it has several interlude-parts with the same melodic background as well as a whole new section in the end of the song which consists of loud dynamics, and repeatable chanting ('B section'), followed by the outro. The complete song segmentation can be seen in Table 6.9. Next the song is converted into a 12x1017 beat-synchronous chromagram matrix.

The outputted chorus parts from our ACD system given the particular song as input are:

Chorus 1:78.4 sec – 87.5 sec

Chorus 2:87.6 sec – 96.9 sec

Never let me down	**times (mm:ss)**
Intro	00:00 - 00:31
Verse 1	00:32 - 01:04
Interlude	01:05 - 01:17
Chorus 1	01:18 - 01:35
Interlude	01:36 - 01:44
Verse 2	01:45 - 02:21
Interlude	02:22 - 02:29
Chorus 2	02:30 - 03:06
Bridge	03:07 - 03:24
B section/outro	03:25 - 04:48

Table 6.10: Table 6. 11: Song parts of "Never let me down again" by Depeche Mode

For this example, we observe that both extracted chorus parts correspond to the first repetition of the chorus within the song, where one represents the first half and the other the second half. Although these results were not considered as optimal, it was thought that the results were acceptable on the grounds that the summing of the two parts result in the complete chorus.

CHAPTER 7

Conclusion and Future Work

The aim of the present study was to examine whether it is possible for a system to be able to recognize cover versions of a given query-song by making use of automatically extracted audio features. Such work might be of relevance on the grounds that content-based search techniques are of particular interest for handling large music databases as their main purpose is to make music or information about music easier to find. Moreover, cover song identification may supply knowledge to improve measurement and modeling of musical similarity which in turn may be used for organizing large music collections. Managing musical rights and licenses can also benefit by making use of such works since similar techniques could facilitate identification of unauthorized moneymaking renditions of a particular song. Extending the idea of cover song music similarity systems to the degree of a QBH-ones which share by and large the same principles, whereby music similarity is performed between reference songs within a database and a user's humming, might have benefits from a user's point of view, as someone facing a large database (YouTube, Amazon, Google) is given a choice to query this database using audio as an input. Such queries would have a wide range of applications as for example in a music pre-listening station found in a music store or in portable devices such as smartphones, portable music players, or even in car music stereo devices. Consequently, examining ACS identification is considered to be a good starting point for music similarity research, providing an analysis of which features are considered relevant for quantitative musical similaritymeasurement.

Since ACS identification systems follow a more general QBE framework, a basic introduction to QBE systems was performed by presenting two AFP matching methods, one of which was implemented and rapidly provided very satisfactory

results, even in the presence of noise. Real-world example applications of such systems are Shazam and Musiwave. However, such systems are 'fragile' to musical changes because they are based on an exact matching scheme and as a result they are unable to identify altered versions of a song.

Cover songs are known to differ in several musical dimensions, as for example instrumentation, tempo and key transposition. The only feature that is more or less retained between two different versions of a particular song is melody and/or harmony. Therefore to accommodate these characteristics use was made of beat-synchronous chroma features. Chroma features were chosen in order to overcome variability in instrumentation. Beat averaging of the chroma features by making use of beats locations derived from the use of a beat tracker dealt with variability in tempo. Matching was then performed by cross-correlating the entire representations by taking into account all possible key transpositions.

The above approach has demonstrated that the use of beat-synchronous chroma features contributes to the task of identifying cover versions. This conclusion derives from the experimental results provided in the present work, whereby it is shown that the use of beat-synchronous chroma features grants better results in identifying the correct cover version than when songs are selected randomly within a database. However, results are farfrom perfect due to the abstract nature of the similarity defined for this task. Sometimes cover versions are simply too different to be considered similar based on the above mentioned features. This in turn creates demand for multi-level musical descriptors which take into account more information about a song. One example of such an enhancement process could be the consideration of the musical structure of a particular title within its representation, as for example a summarization of its most representative parts.

To this end it was found useful to include a presentation of a music structure analysis algorithm which identifies the most repeated part of a song, usually the chorus. For songs which are well-structured, meaning that a clear intro, verse, chorus, bridge and outro are identifiable, the system was quite accurate in identifying the song's chorus. However, for other songs with a less straight-forward structure, which for instance might not even have a chorus, the system's performance was not so precise. Having said this it can be concluded that trying to achieve structure invariance by this means

is prone to errors for mainly two reasons: firstly because automatic structure analysis systems are not totally accurate, and secondly because the part of a song which might be the most representative and identifiable might not be included in the most repeated part, but in another part of the song such as the intro, outro, solo or other. As a result the link of these two systems was not further examined.It would, however, be useful if this topic becomes a subject for further research.

Cover song identification is still an open research topic which needs improvement. Future work might for example focus on examining the impact of additional musical features, such as timbre features in the computation of musical similarity granting a higher similarity measurement to a cover with the same timbre than to a cover with a different timbre. However, such an addition of extra features might be misleading. Another observation in current ACS identification systems is that in order to find a potential cover version of a query song a comparison between each and every song's representation in the music collection and the query's representation must be made. This can amount to a large number of comparisons for a large music collection, decreasing such a system's speed of interaction. Work could be focused on finding a collection of musical motifs which would cover the range of a large music database making these motifs suitable for indexing and therefore allowing for faster search methods. Moreover, excluding karaoke versions future work may concern cover song identification systems furthered by automatic lyrics extraction systems. Identified lyrics might then serve to ease the process of indexing the songs within a database and to speed up search methods. Translation of these lyrics may help finding translated cover versions as well. In addition, for remixed or other cover versions which make short quotations of other musical pieces, already existing algorithms may be altered in order to cater for such cases.

Finally, it could be argued that what had begun as a technologically-based trend in the musical sphere has now developed into a serious scientific endeavor.

References

[1] B. Pardo (2006). Finding structure in audio for music information retrieval. *IEEE Signal Processing Magazine* vol. 23, no. 3, pp. 126-132.

[2] J. S. Downie, M, Bay, A. F. Ehmann and M. C. Jones (2008). Audio cover song identification MIREX 2006-2007 results and analyses. *The International Music Information Retrieval Systems Evaluation Laboratory (IMIRSEL)*, pp. 468-473

[3] MIREX homepage, available at:http://www.music-ir.org/mirex/wiki/2011:Main_Page

[4] J. Serrà, E. Gòmez, P. Herrera (2009). Audio cover song identification and similarity. Background, approaches, evaluation and beyond. *Spirnger*at press.

[5] M. Casey, R. C. Veltkamp, M. Goto, M. Leman, C. Rhodes and M.. Slaney (2008). Content-based music information retrieval: current directions and future challenges. *Proceedings of the IEEE* 96(4), 668-696.

[6] A. Markaki (2008). Music genre classification using temporal and spectral features. *Diploma dissertation,* Technical university of Crete.

[7] M. Marolt (2006). A mid-level melody-based representation for calculating audio similarity. *Int. Symp. on Music Information Retrieval (SMIR)*, pp. 280-285.

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

[9] C. Sailer and K. Dressler (2006). Finding cover songs by melodic similarity. *MIREX extended abstract.*

[10] W. H. Tsai, H. M. Yu and H.M. Wang (2005). A query-by-example technique for retrieving cover versions of popular songs with similar melodies. *Int. Symp. on Music Information Retrieval (ISMIR)*, pp. 183-190.

[11] W. H. Tsai, H. M. Yu and H.M. Wang (2008). 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.

[12] A. Egorov and G. Linetsky (2008).Cover song identification with IF-F0 pitch class profiles. *MIREX extended abstracts*.

[13] D. P. W. Ellis and C. Cotton (2007). The 2007 labrosa cover song detection system. *MIREX extended abstracts.*

[14] D. P. W. Ellis and G. Poliner (2007). Identifying 'cover songs' with chromafeatures and dynamic programming beat tracking. In *Proc. Int. Conf. on Acoustics, Speech & Sig. Proc. ICASSP-07,* pages IV-1429-1432, Hawai'i.

[15] E. Gòmez and P. Herrera (2006). The song remains the same: identifying versions of the same song using tonal descriptors. *Int. Symp. on Music Information Retrieval (ISMIR)*, pp. 180-185.

[16] E. Gòmez, B. Ong and P. Herrera (2006). Automatic tonal analysis from music summaries for version identification. *Conv. of the Audio Engineering Society (AES)*, paper no. 6902.

[17] J. H. Jensen, M. G. Christensen, D.P.W. Ellis and S. H. Jensen (2008). A tempo-insensitive distance measure for cover song identification based on chroma features. *IEEE Int. Conf. on Acoustics, Speech and Singal Processing (ICASSP)*, pp. 2209-2212.

[18] J. H. Jensen, M. G. Christensen and S. H. Jensen (2008). A chroma-based tempo-insensitive distance measure for cover song identification using the 2D autocorrelation. *MIREX extended abstract.*

[19] S. Kim and S. Narayanan (2008). Dynamic chroma feature vectors with applications to cover song identification.*IEEE Workshop on Multimedia Signal Processing (MMSP)*, pp. 984-987.

[20] S. Kim, E. Unal and S. Narayanan (2008). Fingerprint extraction for classical music cover song identification. *IEEE Int. Conf. on Multimedia and Expo (ICME)*, pp. 1261-1264.

[21] Y.E. Kim and D. Perelstein (2007). MIREX-2007: audio cover song detection using chroma features and hidden markov model. *MIREX extended abstract*.

[22] F. Kurth and M. Müller (2008). Efficient index-based audio matching. *IEEE Trans. on Audio, Speech, and Language Processing* 16(2), 382-395.

[23] M. Müller, F. Kurth and M. Clausen (2005). Audio matching via chroma-based statistical features. *Int. Symp. on Music Information Retrieval (ISMIR)*, pp. 288-295.

[24] H. Nagano, K. Kashino and H. Murase (2002). Fast music retrieval using polyphonic binary feature vectors. *IEEE Int. Conf. on Multimedia and Expo (ICME)*, vol. 1, pp. 101-104.

[25] J. Serrà, X. Serra and R. G. Andzejak (2009). Cross recurrence quantification for cover song identification. *New Journal of Physics 11*, art. 093017.

[26] J. Serrà, E. Gòmez, P. Herrera and X. Serra (2008). Chroma binary similarity and local alignment applied to cover song identification. *IEEE Trans. on Audio, Speech, and Language Processing* 16(6), 1138-1152.

[27] J. P. Bello (2007). Audio-based cover song retrieval using approximate chord sequences: testing shifts, gaps, swaps and beats. *Int. Symp. on Music Information Retrieval (ISMIR)*, pp. 239-244.

[28] K. Lee (2006). Identifying cover songs from audio using harmonic representation. *MIREX extended abstract*.

[29] D. P. W. Ellis (2007). Beat tracking by dynamic programming. *J. New Music Research*. Special Issue on Tempo and Beat Extraction.

[30] M. Goto (2003). A chorus-section detecting method for musical audio signals. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICCASP),* pp V-437-440.

[31] A. Eronen (2007). Chorus detection with combined use of mfcc and chroma features and image processing filters. In *Proc. Of the 10th Int. Conference on Digital Audio Effects (DAFx-07)*, Bordeux, France.

[32] M. A. Bartsch and G. H. Wakefield (2001). To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics,* Mohonk, New York.

[33] M. A. Bartsch and G. H. Wakefield (2005). Audio thumbnailing of popular music using chroma-based representations. *IEEE Transaction on multimedia*, vol.7, no. 1, pp.96-104.

[34] B. S. Ong (2007). Structural analysis and segmentation of music signals. *PhD thesis*, IniversitatPompeuFabra, Barcelona, Spain. Available at: http://mtg.upf.edu/node/508

[35] B. S. Ong, E. Gómez and S. Streich (2006). Automatic extraction of musical structure using pitch class distribution features. In *Proceedings of the Workshop in Learning the Semantics of Audio Signals (LSAS),* Athens, Greece, pp. 53-65.

[36] J. Paulus, M. Müller and A. Klapuri (2010). Audio-based music structure analysis. *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, Utrecht, Netherlands, pp. 625-636.

[37] S. J. Orfanidis (1996). Introduction to signal processing. *Pearson Education Inc.*, available at: http://www.ece.rutgers.edu/~orfanidi/intro2sp/orfanidis-i2sp.pdf

[38] S. R. Taghizadeh (2000). Digital signal processing - part 3: Discrete-time signals and systems case studies. *School of communications technology and mathematical sciences*. Available at: http://www.abiscus.com/resources/Signals/matlabsignal.pdf

[39] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein (2001). Introduction to Algorithms (2nded.). *MIT Press and McGraw-Hill*, P.323.

[40] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani (2006). Algorithms. *McGraw-Hill*, p.169, available at: http://www.cs.berkeley.edu/~vazirani/algorithms.html

[41] T. Jehan (2005). Creating music by listening. *PhD thesis*, MIT Media Lab, Cambridge, MA.

[42] A. S. Association (1960). American standard acoustical terminology. Definition 12.9. timbre.

[43] G. Tzanetakis (2002). Manipulation, analysis and retrieval systems for audio signals. *PhD thesis*, Princeton University.

[44] B. Logan (2000). Mel frequency cepstral coefficients for music modeling, *Proc. Int. Symp. on Music Information Retrieval (ISMIR)*.

[45] L. R. Rabiner and B. H. Juang (1993). Fundamentals of speech recognition, *Prentice Hall*, New Jersey.

[46] R. Shepard (1964). Circularity in judgments of relative pitch. *J. Acoust. Soc. Am.,* 36, pp. 2346-2353.

[47] M. Y. Kao, C. B. Yang and S. H. Shiau (2009). Tempo and beat tracking for audio signals with music genre classification, *Int. Journal of Intelligent Information and Database Systems (IJIIDS)*, 3(3): 275-290

[48] M. F. McKinney and D. Moelants (2004). Extraccting the perceptual tempo from music, *ISMIR, 5th International Conf. on Music Information Retrieval*, Barcelona, Spain

[49] M. F. McKinney and D. Moelants (2004). Deviations from the resonance theory of tempo induction. *Conference on Interdisciplinary Musicology*.

[50] M. F. McKinney and D. Moelants (2004). Ambiguity in tempo perception: What draws listeners to different metrical levels? *Music Perception*, vol. 24 no. 2 pp. 155-166.

[51] J. Haitsma and A. Kalker (2002). A highly robust audio fingerprinting system, in *Proc. 3rd Int. Conf. Music Retrieval (ISMIR)*, Paris.

[52] A. Wang (2003). An industrial strength audio search algorithm, in *Proc. 4th Int. Conf. Music Retrieval (ISMIR)*, Baltimore, MD.

[53] D. Ellis (2009). Robust landmark-based audio fingerprinting, web resource, available at: http://labrosa.ee.columbia.edu/matlab/fingerprint

[54] G. H. Wakefield (1999). Mathematical representation of joint time-chroma distributions. *Int. Symp. on Opt. Sci., Eng., and Instr.,SPIE'99*, Denver Colorado.

[55] T. Fujishima (1999). Realtime chord recognition of musical sound: A system using common lisp music. In *Proc. ICMC,* pages 464-467, Beijing.

[56] E. Gòmez (2006). Tonal description of music audio signals. *PhD thesis*, UniversitatPompeuFabra, Barcelona, Spain. Available at: http://mtg.upf.edu/node/472

[57] T. Abe and M. Honda (2006). Sinusoidal model based on instantaneous frequency attractors. *IEEE Tr. Audio, Speech and Lang. Proc.,* 14(4): 1292-1300.

[58] F. J. Charpentier (1986). Pitch detection using the short-term phase spectrum. *In Proc. ICASSP-86,* pages 113-116, Tokyo.

[59] M. F. McKinney, D. Moelants, M. Davies and A. Klapuri (2007). Evaluation of audio beat tracking and music tempo extraction algorithms. *Journal of New Music Research*.

[60] J. P. Bello, L. Daudet, S. Abadia, C. Duxbury, M. Davies and M. B. Sandler (2005). A tutorial on onset detection in music signals, *IEEE Transactions on Speech and Audio Processing,* Vol. 13, Nr. 5, pp.1035-1047

[61] J. Laroche (2003). Efficient tempo and beat tracking in audio recordings. *J. Audio Eng. Soc.,* 51(4): p. 226-233.

[62] N. Otsu (1979). A threshold selection method from gray-level histograms. *IEEE Trans. Syst., ManmCybern,* vol. SMC-9, no.1, pp. 62-66

[63] The lame project. Download available at: http://lame.sourceforge.net/

[64] M. Khadkevich (2010). JAMAL Manual, available at: http://jamal.sourceforge.net/documentation.shtml .

[65] D. P. W. Ellis (2007). The "covers80" cover song data set. Web resource, available: http://labrosa.ee.columbia.edu/projects/coversongs/covers80/ .

[66] NOISEX-92. Download available at: http://spib.rice.edu/spib/select_noise.html