MASTERARBEIT

Automatic Chord Detection in Polyphonic Audio Data

Ausgeführt am Institut für
Softwaretechnik und Interaktive Systeme
der Technischen Universität Wien

unter der Anleitung von
ao.Univ.Prof. Dr. Andreas Rauber

durch

VERONIKA ZENZ
Waxriegelgasse 9
A-2700 Wr. Neustadt

Wien, Februar 2007 ....

Abstract

Automatic analysis of digital audio data has a long tradition. Many tasks that humans solve easily, like distinguishing the constituting instrument in polyphonic audio or the recognition of rhythm or harmonies are still not solved for computers. Especially the development of an automatic transcription system that computes the score out of a music recording is still a distant prospect despite decades-long research efforts. This thesis deals with a subproblem of automatic transcription – automatic chord detection. Chord detection is particularly interesting as chords are comparatively simple and stable structures, and at the same time completely describe the harmonic properties of a piece of music. Thus musicians are able to accompany a melody solely by provided chord symbols. Another application of this thesis is automatic annotation of music. An annotated music database can then be searched for specific chord sequences and harmonic or emotional characteristics.

Previous approaches to chord detection often severely restricted the types of analysable music by considering for example only performances without percussion or vocals. In addition a large part of existing approaches does not integrate music theoretical knowledge in their analysis, thus renouncing helpful additional information for chord detection. The goal of this thesis was to design an algorithm that operates on musical pieces of arbitrary instrumentation and considers music theoretical knowledge. Thus the developed algorithm incorporates rhythm, tonality and knowledge about the common frequencies of chord-changes. An average accuracy rate of 65% has been achieved on a test set of 19 popular songs of the last decades and confirms the strength of this approach.

The thesis starts with an overview followed by an introductory chapter about acoustical and music theoretical fundamental principles. Design and implementation of the chord detection algorithm constitute the two central chapters of this thesis. Subsequently, setup and results of the performed evaluation are described in detail. The thesis ends with a summary of the achieved insights and an outlook on possible future work.

Kurzfassung

Die automatische Analyse digitalisierter Musikaufnahmen hat eine lange Tradition. Viele Aufgaben die für einen Menschen leicht lösbar sind, wie das Unterscheiden verschiedener Instrumente, das Erkennen des Rhythmus oder der Harmonien sind für den Computer jedoch noch nicht gelöst. Speziell von einem automatischen Transkriptionssystem, das aus einer digitalen Musikaufnahme eine Partitur erstellt, ist man trotz jahrzehntelanger Forschung noch weit entfernt. Diese Arbeit beschäftigt sich mit einem Teilproblem der automatischen Transkripition - der Akkorderkennung. Diese ist besonders interessant, weil Akkorde vergleichsweise simple und robuste, also über längere Zeitspannen gleich bleibende Strukturen sind, gleichzeitig aber die harmonischen Eigenschaften eines Musikstücks vollständig beschreiben. So können Musiker eine Melodie allein anhand vorgegebener Akkordsymbole begleiten. Ein weiteres Anwendungsgebiet dieser Arbeit stellt die automatisierte Annotation von Musikdaten dar. In einem Musikarchiv kann so nach bestimmten Akkordfolgen, harmonischen und emotionalen Eigenschaften gesucht werden.

Bisherige Ansätze zur Akkorderkennung machen teilweise große Einschränkungen auf die zu analysierenden Musikdaten, indem sie sich zum Beispiel auf die Analyse von Musikstücken ohne Schlagzeug oder Gesang beschränken. Weiters bezieht ein Großteil der bestehenden Arbeiten musiktheoretische Regeln nicht in die Analyse mit ein und lässt dadurch hilfreiche Zusatzinformation zur Akkorderkennung unberücksichtigt. Das Ziel dieser Arbeit ist es, einen Algorithmus zu entwickeln, der auf Musikstücken mit beliebiger Instrumentierung arbeitet und dabei musiktheoretisches Wissen zu berücksichtigen. So fließen in den hier entworfenen Algorithmus Rhythmus, Tonart und das Wissen um Häufigkeiten von Akkordwechseln in die Akkorderkennung ein. Eine durchschnittliche Erkennungsrate von 65% auf 19 Teststücken aus dem Gebiet der Unterhaltungmusik der letzten Jahrzehnte wurde erreicht und untermauert die Stärke dieses Ansatzes.

Die Arbeit beginnt mit einer Übersicht und einem einleitenden Kapitel zu akustischen und musiktheoretischen Grundlagen. Kapitel 3 gibt einen Überblick über bestehende Akkorderkennungssysteme und deren Eigenschaften. Der Entwurf eines eigenen Algorithmus zur Akkorderkennung und dessen Implementierung bilden die zwei zentralen Kapitel dieser Arbeit. Im Anschluss wird der Aufbau und die Ergebnisse der durchgeführten Evaluierung beschrieben. Die Arbeit schließt mit einer Zusammenfassung der gewonnenen Erkenntnisse und einem Ausblick auf mögliche zukünftige Arbeiten.

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Arbeit beigetragen haben. Besonderer Dank gebührt hierbei meinem Betreuer ao. Univ. Prof. Andreas Rauber, der mir mit vielen Anregungen und großem Engagement die Chance gegeben hat, mein eigenes Wunschthema zu bearbeiten. Bedanken möchte ich mich auch bei meinen Kollegen am Institut für Technische Informatik für zahlreiche Hilfestellungen und interessante Diskussionen.
Diese Arbeit wäre nicht möglich gewesen ohne den Rückhalt in meiner Familie und das Vertrauen, das mir meine Eltern entgegenbringen. Ganz besonders möchte ich mich bei Elisabeth bedanken, der besten Schwester, die man sich wünschen kann, bei Walter, für gute Ideen für das Poster und zahlreiche Anregungen in fachlichen Diskussionen, bei Hemma und Birgit für durchwachte Nächte und hinterfragte Systeme, sowie bei Markus, ohne den ich dieses Studium vielleicht gar nicht begonnen hätte.

Abschließend sei noch jenen gedankt, die mich überhaupt erst auf die Idee zu dieser Arbeit gebracht haben (in zufälliger Reihenfolge): The Weakerthans, Lambchop, Red Hot Chili Peppers, Harri Stojka, R.E.M., Tori Amos, Peter Licht, Titla, Tocotronic, Conor Oberst, Muse, Goran Bregovic, Element of Crime, den Beatsteaks, Nada Surf, Belle & Sebastian, Radiohead, Elliott Smith, Fiona Apple, The Notwist, K's Choice, Nirvana, Noir Desir, Travis, Nick Drake, Dobrek Bistro, Roland Neuwirth, Damien Rice, Madeleine Peyroux, Tom Waits, Rilo Kiley, die Schröders, Smashing Pumpkins, den Toten Hosen, Riccardo Tesi, Flaming Lips, Blur, Starsailor, The Velvet Underground, T.V. Smith, Ani di Franco, Vincent Delerm, Bratsch, Cat Power, Pete Yorn, Django Reinhardt, Emilio Tolga, Gunshot u.v.m.

Table of Contents

1 Introduction....1

2 Acoustic and Music Theoretical Background....4

2.1 Acoustics....4

2.1.1 Audio Signal....4

2.1.2 Harmonic Series....5

2.2 Music Theory....7

2.2.1 Basic Terms and Definitions....8

2.2.2 Tonality and Chords....10

2.2.3 Tuning and Temperament....14

2.3 Conclusion....15

3 Related Works....16

3.1 Chord Detection Algorithms....16

3.2 Hidden Markov Models....17

3.3 Self-organized Maps....18

3.4 Hypothesis Search....19

3.5 Comparison and Summary....20

3.6 Conclusion....21

4 Conceptual Design....22

4.1 Overview....22

4.2 Frequency Detection by Enhanced Autocorrelation....24

4.3 PCP and Reference PCPs....27

4.3.1 PCP Generation....28

4.3.2 PCP Analysis....31

4.3.3 Reference PCPs....31

4.4 Key Detection....33

4.5 Tempo and Beat Tracking....35

4.6 Chord Sequence Optimization....37

4.7 Conclusion....40

5 Implementation....41

5.1 System Overview....41
5.2 The Chord Analyser – Genchords....43
5.2.1 User Interface....43
5.2.2 Output formats....47
5.2.3 Performance....50
5.2.4 Parameter Summary....51
5.3 Tools....52
5.3.1 Learnchords....52
5.3.2 Labeldiff....53
5.3.3 Transpose....56
5.3.4 Chordmix....57
5.4 Conclusion....58
6 Evaluation....59
6.1 Test Set....59
6.2 Accuracy....60
6.2.1 Key Detection Accuracy....62
6.2.2 Influence of Key Detection....63
6.2.3 Influence of Beat Tracking....64
6.2.4 Influence of Chord Sequence Optimization....65
6.2.5 Accuracy of the entire Algorithm....66
6.2.6 Accuracy Limits....68
6.2.7 Comparison with other Algorithms....69
6.3 Confusion Matrix....70
7 Conclusion....71
7.1 Summary....71
7.2 Future Work....71
Bibliography....74

List of Figures

Figure 1.1: Chord Sequence: The Beatles - Yesterday........1
Figure 1.2: Chords and Lyrics........2
Figure 2.1: Sinus tone........4
Figure 2.2: Fundamental frequency and first five overtones........6
Figure 2.3: Fundamental frequency, first five overtones and Resulting........6
Figure 2.4: Triads........10
Figure 2.5: Scale & Chords........11
Figure 2.6: Scottish Traditional "Auld lang syne"........12
Figure 2.7: J.S. Bach, "Nun laßt uns Gott, dem Herren"........12
Figure 2.8: Cycle of Fifths........13
Figure 3.1Chord Detection Algorithm Classifications........16
Figure 3.2: Hidden Markov Model for Chord Sequences........17
Figure 3.3: Self-organized map used by Su and Jeng in [8] for chord detection........18
Figure 4.1: Flow Chart of the Chord Detection Algorithm........23
Figure 4.2: Enhanced Auto Correlation: Input........24
Figure 4.3: AC, EAC........26
Figure 4.4: EAC: original data, lowpass filtered data........26
Figure 4.5: Pitch Class Profile (PCP)........27
Figure 4.6: Peak Picking........29
Figure 4.7: Comparison of different PCP Generation Algorithms........30
Figure 4.8: Generated Reference PCPs........32
Figure 4.9: BeatRoot: System Architecture........35
Figure 4.10: BeatRoot: Inter-onset intervals and clustering to groups C1-C5........35
Figure 4.11: Chord Change Penalty........38
Figure 5.1: Implementation Overview........41
Figure 5.2 Interactive Genchords Session........46
Figure 5.3: Example scorefile........48
Figure 5.4: PCPFile Entry........49
Figure 5.5: Labeldiff........55

Figure 5.6: Labeldiff: Confusion Matrix..55
Figure 5.7: Transpose..56
Figure 5.8: Chordmix..57
Figure 6.1: Test Results for Key Detection..63
Figure 6.2: Test Results for Beat Detection..64
Figure 6.3: Labelfile Comparison: short span versus beat tracking - (Elvis Presley, Devil in Disguise)..64
Figure 6.4: Test Results for Optimization..65
Figure 6.5: Test Results for all modules combined..66
Figure 6.6: Accuracy results for different number of output chords per time span...........................68

List of Tables

Table 2.1: Overtones (assumed fundamental frequency C)....7
Table 2.2: Intervals....8
Table 2.3: Major Scale....9
Table 2.4: Natural Minor Scale....9
Table 2.5: Chord Functions....11
Table 3.1: Related Works Summary....20
Table 4.1: fmax....28
Table 4.2: Chords for C-Major....34
Table 4.3: Keys and associated chords....34
Table 4.4: Exemplary Chordsequence....39
Table 4.5: Example Optimization Algorithm....39
Table 5.1: Genchords Performance (in seconds for a 1 minute song)....50
Table 6.1: Test Set....61
Table 6.2: Key Detection Results....62
Table 6.3: Accuracy Overview....67
Table 6.4: Result Comparison....69
Table 6.5: Confusion Matrix: Elton John - Sorry Seems To Be....70
Table 6.6: Shared pitches of confused chords....70

1 Introduction

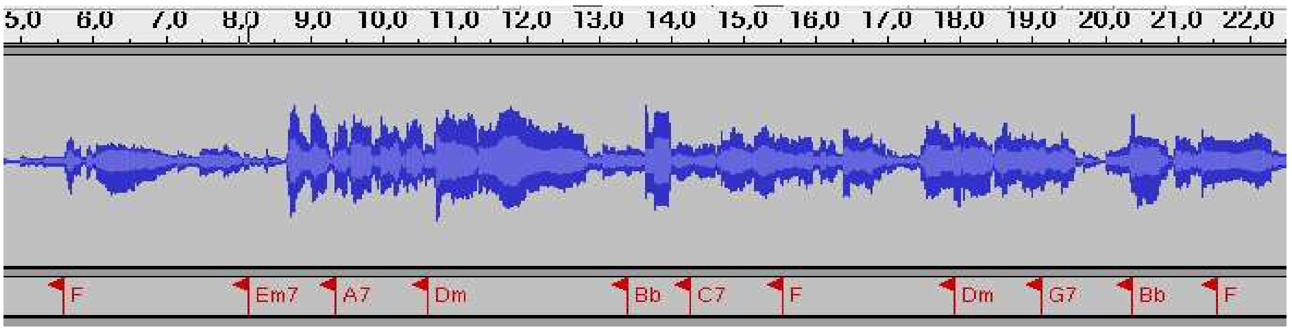


Figure 1.1: Chord Sequence: The Beatles - Yesterday

Automatic chord detection is part of the large research field of computer audition (CA) which deals with all kinds of information extraction from audio signals. Chord detection extracts the harmonies that occur over the time of a piece of music. Figure 1.1 depicts an exemplary result of chord detection visualized with Audacity.

Motivation and Applications

The main goal of computer audition has long been transcription of speech or music. In spite of decades-long research effort, automatic music transcription is still a distant prospect. Chord detection is a special form of lossy music transcriptions, that captures only harmonic properties of the audio signal. It is particularly interesting as chords are comparatively simple and stable structures, and at the same time completely describe a piece of music in terms of occurring harmonies. The great interest of musicians in chord sequence is perhaps best demonstrated by pointing out the large number of websites, newsgroup and forum messages on this topic. Newsgroups like rec.music.makers.guitar.tablature, or alt.guitar.tab offer a platform to request and publish chord sequences and tablatures together with the lyrics of songs. Many websites that offered large chord-databases, like olga.net (currently offline), chordie.com or azchords.com have evolved in the last decade but many of them are currently offline due to legal reasons. These platforms provide chord information usually not as time-chord pairs but give the timing information indirect by stating the lyrics that are sung during the duration of each chord. An example for such a chord sequence file is shown in Figure 1.2. It shows lyrics and chords for the first measures of the Beatles' song "Yesterday", the same measures that have been used for Figure 1.1.

```
 F           Em7   A7                 Dm
Yesterday all my troubles seemed so far away

Bb    C7              F
now I need a place to hide away, oh

Dm  G7     Bb    F
I believe in yesterday
```

Figure 1.2: Chords and Lyrics

Besides transcription as an end in itself, a new application field has evolved in the last years: Music Information Retrieval (MIR). The upcoming of compression formats, especially mp3, and the decrease of cost of memories capacities, lead to the rise of large private and public digital music archives. In order to search these archives for music with special properties, each music file has to be annotated with this information. Such properties are commonly artist, title and genre but could as well be mood, melody, harmonies, lyrics and so on. Manual information extraction and annotation is rather time consuming, thus implying the need to design algorithms that compute these features automatically. The chord-sequence of a song does not only describe its harmonic properties but can also be used to draw conclusions on its genre and emotions that are evoked at the listener. Thus, having a music database annotated with chord information, users could search for specific chord-sequences, music with complex or simple chord structures, slow or fast chord progressions, rather sad (minor) or lively (major) music and so on.

Goals of this thesis

Previous approaches to chord detection often severely restricted the types of analysable music by considering for example only performances without percussion or vocals. In addition a large part of existing approaches does not integrat theoretical music knowledge in their analysis, thus renouncing helpful additional information for chord detection.

The goal of this thesis is to design, implement and evaluate an algorithm that detects the chord sequence from arbitrarily instrumented music. We want to evaluate the possibilities of integrating music theoretical knowledge into the algorithm and whether or to what extend detection quality can be increased in this way. We further aim to support precise evaluation the same as immediate feedback by means of resynthesizing the detected chord sequence.

Input restrictions

As input data we accept sampled audio signals. Other music formats like the musical instrument digital interface (MIDI), that contain precise pitch and instrument information, are not covered by this thesis. The music has to contain chords in the first place, that means it must not be monophonic or atonal, as it is impossible to accurately detect chords where there are no chords. We further assume that the key of the input data does modulate. Only short, transient modulations are allowed. This assumption excludes many pieces of music, for example most music from the romantic period, where modulations are very frequent. This restriction has been necessary to hold complexity down – key detection itself and modulation detection in particular form a separate research field. The restriction is not as severe as it may seem in the first place as it can be circumvented using a modulation detection tool that splits the song into parts of constant keys which can then be passed to our chord detector.

Overview

The structure of this thesis follows the chronology of the performed research. First basic concepts and definitions that go beyond computer-science are summarized. Afterwards existing approaches to chord detection are discussed. Based on this knowledge and the requirements stated above, an algorithm has been developed that integrates beat structure and the key of the song in addition to the common frequency-feature and uses these features to raise its detection accuracy. This design is described in Chapter 4. Guided by the conceptual design the chord detection program ***genchords*** has then been implemented in C++ together with a set of evaluation and transformation tools, described in Chapter 5. A test set of 19 songs has then been assembled, labelled and was used to evaluate our approach. Details on the test set and evaluation results can be found in Chapter 6. The thesis closes with a summary of the achieved insights and an outlook on possible future work.

2 Acoustic and Music Theoretical Background

Automatic chord detection, as computer audition in general, overlaps several disciplines: Acoustics, or the study of sound, music theory and computer engineering. A basic knowledge of acoustics is necessary to understand the principles of what sound in general and music in particular are physically, and which properties they have. The study of music theory is necessary to define the problem of chord detection itself. For this thesis it is even essential, as the rules of harmonization, defined by music theory, shall be integrated in our algorithm and help us to interpret the audio signal.

Though it is impossible to treat these two disciplines, acoustics and music theory, in detail, this chapter tries to give an introduction to their basic terms and concepts so that a computer-scientist without expertise in those fields can understand the chord detection algorithms described in the subsequent chapters. The interested reader can find further information on acoustics in [1], respectively may consult the text books [2] and [3] for detailed information on music theory.

2.1 Acoustics

Acoustics is a part of physics and is concerned with the study of sound and its production, control, transmission, reception, and effects. Section 2.1.1 defines the basic concepts of sound. Section 2.1.2 then introduces harmonic series which are fundamental to understand the complexity of fundamental frequency and pitch detection.

2.1.1 Audio Signal

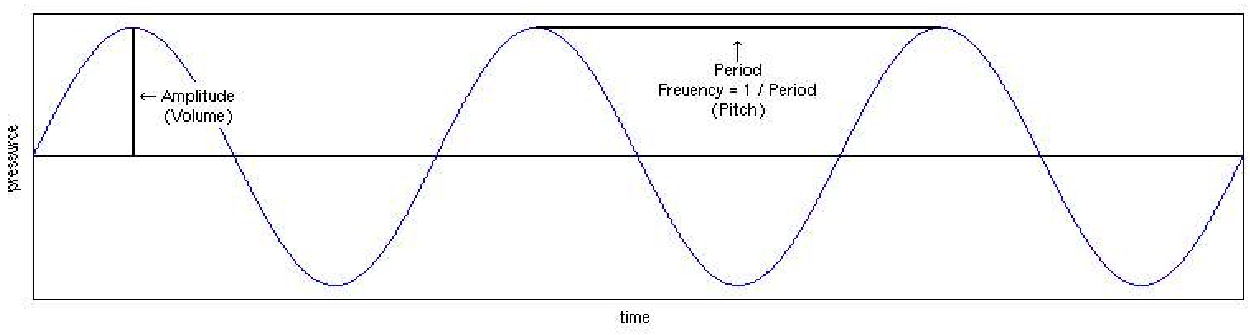


Figure 2.1: Sinus tone

Sound is the vibration of a substance, commonly the air. It is initiated by a vibrating source, e.g. vocal cords or a plucked guitar string and transmitted over the air or another medium. Sound propagates as a wave of alternating pressure, that causes regions of compression and regions of rarefaction. It is characterized by the properties of this wave, which are frequency (measured in Hertz, short Hz) and amplitude (measured in db(SPL) relative to the threshold of human hearing, which is 20µPa). Sound waves are commonly depicted in the form pressure over time (see Figure 2.1. The amplitude of the sound wave determines the intensity of the sound and the frequency determines the perceived pitch or pitches.

2.1.2 Harmonic Series

Real-world tones do not consist of only one sinus wave. When a body is started to vibrate it does not only vibrate as a whole, but at the same time vibrates in all its parts. E. g. an air column vibrates as a whole and in its halves, thirds, quarters and so on. Every tone generated by human voice or acoustic instruments thus consists of the sinus wave at the fundamental frequency, that usually corresponds to the perceived pitch, and various differently strong waves at integer multiples of this frequency. The multiple frequencies are called overtones. The fundamental frequency and its overtones are called partials or harmonics. Their strength and number characterizes the timbre of the tone and different instruments have differently strong overtones. Figure 2.2 depicts a sinus wave representing a fundamental frequency and its first 5 overtones in separate graphs. Figure 2.3 shows those functions in one graph. The fundamental frequency is illustrated as continuous bold line, the first 5 overtones are depicted with smaller line width and with smaller amplitudes than the fundamental frequency. The dashed red line represents the resulting curve. Both figures illustrate the theoretic concept of harmonic series and do not state the harmonic series of one specific instrument. The concrete ampliudes of the various harmonics depend on the instrument and the played pitch.

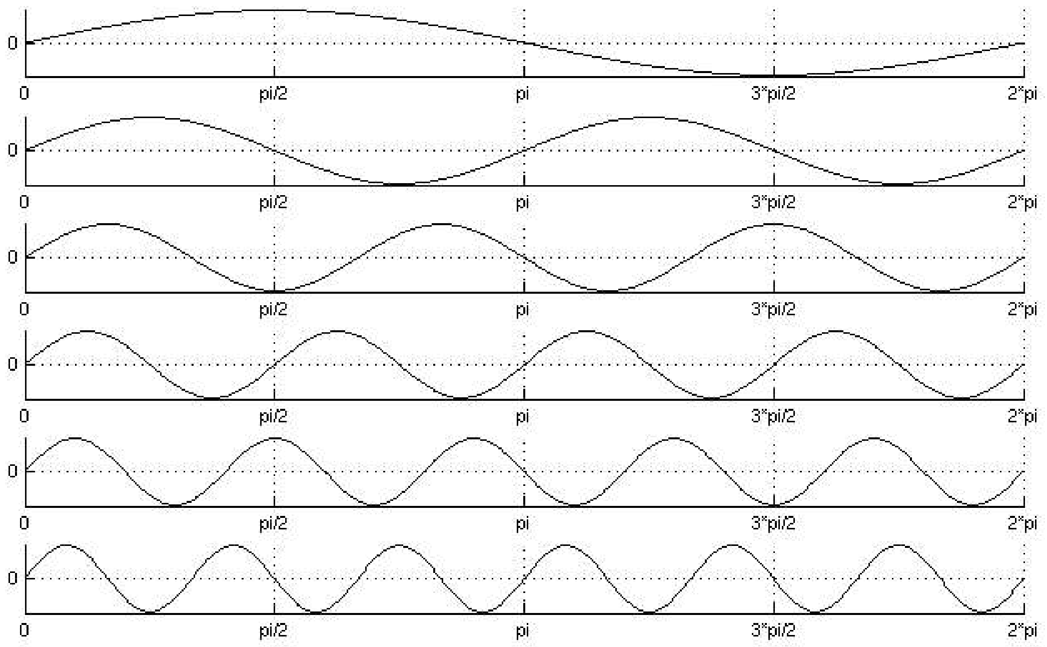


Figure 2.2: Fundamental frequency and first five overtones

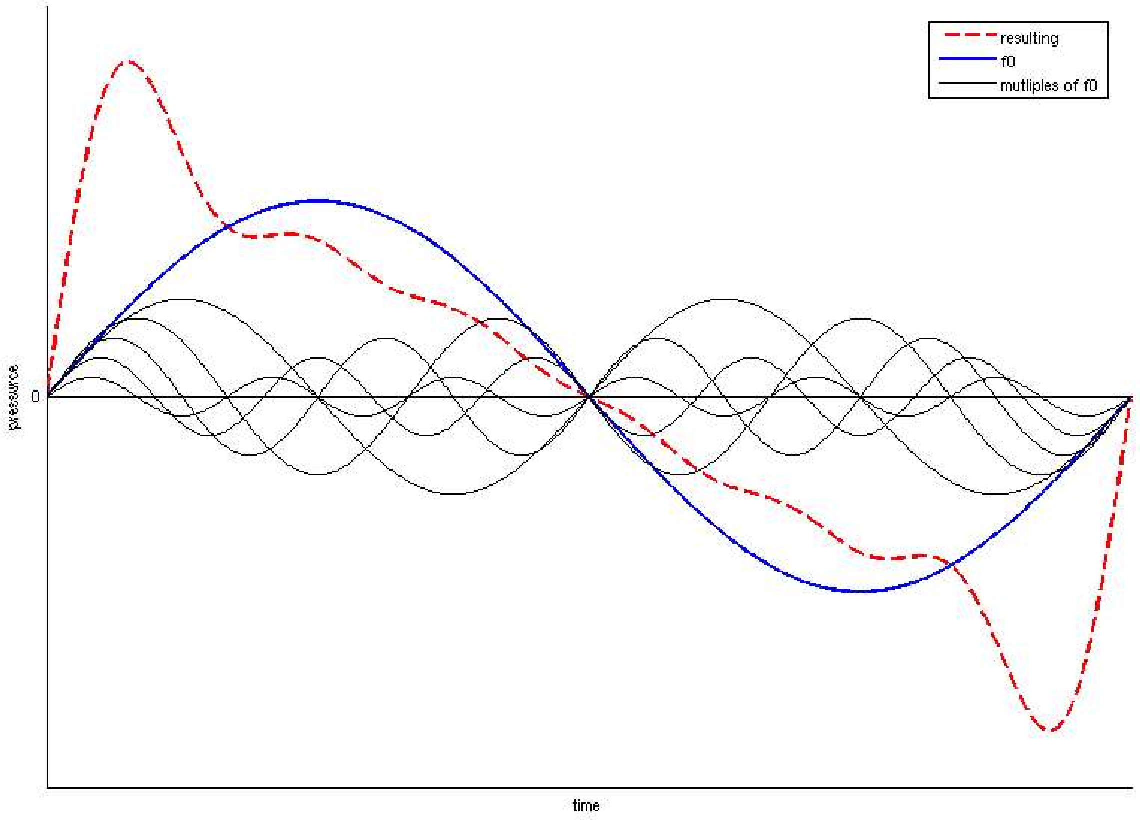


Figure 2.3: Fundamental frequency, first five overtones and Resulting

Overtone		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Frequency	f	2f	3f	4f	5f	6f	7f	8f	9f	10f	11f	12f	13f	14f	15f	16f
Interval	I	I	V	I	III	V	VII-	I	II	III	IV+	V	VI	VII-	VII	I
Pitch	c_2	c_3	g_3	c_4	e_4	g_4	bb_4	c_5	d_5	e_5	$f\#_5$	g_5	a_5	bb_5	b_5	c_6

Table 2.1: Overtones (assumed fundamental frequency C)

Table 2.1 lists the first 15 overtones. Note that the first 5 overtones and 9 of the first 15 overtones (highlighted in the table) are part of the major chord built on the fundamental frequency. For each overtone its frequency and interval class relative to the fundamental frequency are listed. As an example the fundamental frequency c_2 is used and the pitch names of its overtones are shown in the last row of the table. Overtones are theoretically unlimited and the first 43 have been verified [2]. After the 15th overtone the distances between the overtones become smaller than semitone steps and the overtones are no more educible in our staves. Nevertheless they are relevant to the timbre.

The physical phenomenon of the overtones has often been used to explain music theoretical buildings, for example by Riemann in [4] or Helmholtz [5]. The fact that the first five overtones form a major chord prove for some the "natural" foundation of major tonality. However, this approach has many critics, whose main point is the lack of a simple deduction of the minor tonality from the overtones.

2.2 Music Theory

Music theory is the entirety of theories that build the foundation of understanding and composing music. This section gives an overview over those concepts of music theory that are necessary to understand the task of chord detection and the solution proposed in this thesis. We will concentrate on the one branch of music theory, called harmonic theory, that deals with harmonies, chords and tonality and is of special relevance for this thesis. Following the assumptions made on the input data in chapter 1, we will restrict our overview on classic major/minor tonality theory and leave out other contemporary approaches like twelve-tone or atonal music.

This section is divided into three subsections. First the basic terms used in music theory like pitch class, enharmonics or scale are defined. Then the fundaments of major/minor tonality are introduced focusing on the concepts of keys, chords and function theory. The section closes with an explanation of tuning and temperament.

2.2.1 Basic Terms and Definitions

Human pitch perception is periodic as that pitches with the double frequencies (octaves) are perceived as being very similar. In western music the octave is divided into 12 ***semitones***, named with the first seven letters of the Latin alphabet and an optional ***accidental***. A sharp accidental (#) raises the pitch by one semitone, a flat accidental (b) lowers it by one semitone. The 12 semitones in ascending order beginning with c are [c, c#, d, d#, e, f, f#, g, g#, a, a#, b].

The naming convention is not injective, so that the same pitch can be named with several note names, called ***enharmonics***. The chromatic scale above could thus also be noted with the enharmonic equivalent [c, db, d, eb, f, gb, ab, a, bb, b].

All pitches that stand in octave relationship are grouped into a set, called a ***pitch class***. More precisely a pitch class is an equivalence class of all pitches that are octaves apart. The pitch class 'a' is a set containing the elements {..., a_1, a_2, a_3, a_4, a_5, ... }.

In order to differentiate two notes that have the same pitch class but fall into different octaves a number specifying the octave is added to the pitch name. According to standard tuning a_4 is set to 440 Hz, thus a_5 is one octave higher than a_4 and a_3 is one octave lower than a_4.

english name	***german name***	***nr of semitones***	***example***
perfect unison	Prim	0	c-c
minor second	kleine Sekund	1	c-c#
major second	große Sekund	2	c-d
minor third	kleine Terz	3	c-d#
major third	große Terz	4	c-e
perfect fourth	Quart	5	c-f
augmented fourth diminished fifth	übermäßige Quart verminderte Quint, Tritonus	6	c-f#
perfect fifth	Quint	7	c-g
minor sixth	kleine Sext	8	c-g#
major sixth	große Sext	9	c-a
minor seventh	kleine Septim	10	c-a#
major seventh	große Septim	11	c-b
perfect octave	Oktave	12	c_1-c_2

Table 2.2: Intervals

The relationship between two notes is called ***interval***. Intervals may occur vertical (harmonic) if the two notes sound simultaneously or linear (melodic) if they sound successively. Table 2.2 outlines the English and German names of the intervals the distance between the semitones and two exemplary pitches that form this interval.

Notes are arranged into ***scales***. A scale is an ordered series of notes that provides the material for part or all of a musical work. Common scales are the major and the minor scale, latter having three forms, natural minor, harmonic minor and melodic minor. The scales are characterized by the intervals either between two subsequent pitches (whole-step, semi-step) or between the first and the nth degree. The major third is characteristic for the major scale, the minor third for all minor scales. The minor scales differ in the interval between the fifth, sixth and seventh degree. Table 2.3 shows intervals, inter-pitch steps and example pitches of the major scale. The natural minor scale is listed in Table 2.4.

The natural minor scale equals the major scale shifted by a major sixth. Such scales, that have the same key-signature are called ***relative***; C-Major for example is relative to A-Minor, C-Minor is relative to Eb-Major and so on.

degree	1	2	3	4	5	6	7	8
step	whole	whole	semi	whole	whole	whole	semi	
interval	perfect unison	major second	major third	perfect fourth	perfect fifth	major sixth	major seventh	perfect octave
C-Major	c	d	e	f	g	a	b	c
Eb-Major	eb	f	g	ab	bb	c	d	eb

Table 2.3: Major Scale

degree	1	2	3	4	5	6	7	8
step	whole	semi	whole	whole	semi	whole	whole	
interval	perfect unison	major second	major third	perfect fourth	perfect fifth	major sixth	major seventh	perfect octave
C-Minor	c	d	eb	f	g	ab	bb	c
A-Minor	a	b	c	d	e	f	g	a

Table 2.4: Natural Minor Scale

2.2.2 Tonality and Chords

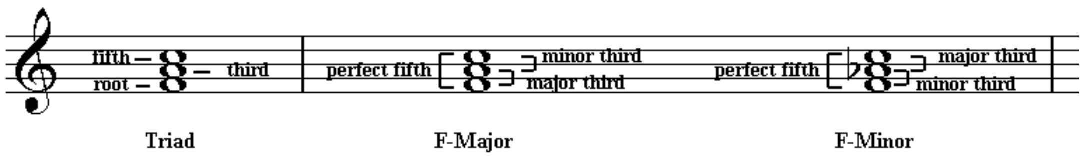


Figure 2.4: Triads

We have defined intervals as combinations of two notes. A collection of three or more notes that appear simultaneously or near-simultaneously is called ***chord***. Chords are characterized by the intervals they contain and the number of distinct pitch classes. The most fundamental chords in major-minor tonality are triads, consisting of three notes, the first, which is called the ***root*** note, a third and a fifth, respectively two third layered above each other. A triad containing, root note, major third and minor third is called a ***major chord***. ***Minor chords*** consist of the root note, a minor third and a major third (see Figure 2.4).

By alteration, suspension and addition of certain pitches new chord with a different structure can be generated from the basic triads. The most important of these chords are the seventh chord which originates by addition of a third third (c-e-g-bb) and the "sixte-ajoutée" which consists, as the name suggests, of a triad with an additional sixth over the root note (f-a-c-d). Entirely differently structured chords exist too, which are no more based on thirds but on layered fourths, fifths or clusters, that contain many or all notes in a marked region.

Chord symbols start with the root note (e.g. C, Db) followed by the letter 'm' for minor chords and optional additions (e.g. additional intervals) that are usually noted as superscripts. Other notations for minor Chords add a minus sign (e.g. C-) or use lower-case pitch names for minor chords. The chord symbol for C-Major (c-e-g) is C, for C-Minor (c-eb-g) Cm, for the major seventh chord on C (c-e-g-bb) C^7. and for a diminished C chord (c-eb-gb) C° or C^{dim}.

A piece of music usually has one major or minor chord, that represents the harmonic center. This special chord is called the ***key*** of the piece of music. ***Tonality*** is the system of composing music around such a tonal center. The word tonality is frequently used as a synonym for key. In classical music the key is often named in the title (e.g. Beethovens 5th Symphony in C-Minor).

The major or minor scale that starts on the root note of this chord defines the main set of pitches. Given this scale, it is possible to build a triad on every chord of that scale with notes that are proper to the scale. Figure 2.5 shows the resulting chords for C-Major.

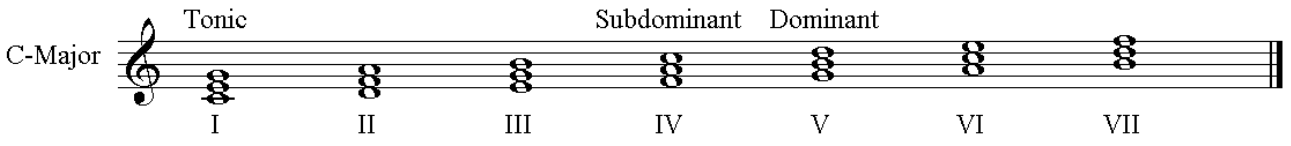


Figure 2.5: Scale & Chords

The resulting chords stand in certain relationships to each other and assume certain ***roles or functions*** in regard to the key. Table 2.5 lists the function names and common abbreviations. In this thesis we will use the Funktionstheorie by Riemann, which is commonly used in Germany. For completeness and comparison the Stufentheorie that is also frequently referred to is also listed in Table 2.5. For more details on Stufentheorie and Funktionstheorie see [3]. The most important functions are the tonic which generates a feeling of repose and balance, the dominant, which generates instability and tension and the subdominant which acts as a dominant preparation. According to Riemann the other chords are only substitutes for the one of those three main chords with which they share the most notes. E. g. the chord on the sixth degree, the tonic parallel (A-Minor (a-c-e) for C-Major) acts as a substitute to the tonic (C-Major (c-e-g)) or to the subdominant (F-Major (f-a-c)) with both of which it shares two of its three notes.

Funktionstheorie		***Stufentheorie***		***Example***
Function	***Abbreviation***	***Function***	***Roman Numeral***	***(C-Major)***
Tonic	T	Tonic	I	C
Subdominant Parallel	Sp	Supertonic	II	Dm
Dominant Parallel	Dp	Mediant	III	Em
Subdominant	S	Subdominant	IV	F
Dominant	D	Dominant	V	G
Tonic Parallel / Subdominant Contrast	Tp / Sg	Submediant	VI	Am
Dominant Seventh	D^7	Leading/Subtonic	VII	$B°/G^{7\ \text{omit}\ 1}$

Table 2.5: Chord Functions

The characteristics of the chords do not evolve from their absolute pitches but from their functions and relationships to each other. Some standard ***chord sequences*** (also called chord progressions) with characteristic properties have established themselves and reoccur in many pieces of music. The most common chord progression in popular music is based on the three main degrees tonic, subdominant and dominant and is I-IV-V-I.

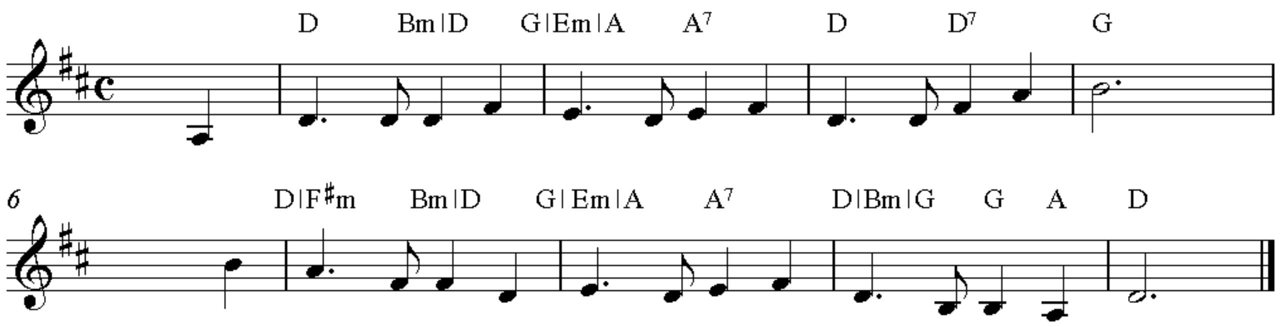


Figure 2.6: Scottish Traditional "Auld lang syne"

Figure 2.6 shows the melody of the first measures of the Scottish traditional "auld lang syne" in D-major. Above the notes possible accompanying chords are noted, separated by a vertical slash. Note that there is very often more than one possible chord that could accompany the melody. One could chose only one chord per measure (D-A-D-G-D-A-G-D), or change the chord every second quarter note (D,Bm-G,A^7–D,D^7–G-D,Bm-G,A^7-Bm,G,A-D) The rhythm in which the chord changes occur defines the ***harmonic rhythm*** which is independent from the melodic rhythm. The harmonic rhythm is essential for the character of a piece of music, whether it is perceived to be strongly structured or large-scaled, short-winded or lengthy, to progress fast or to dwell. Generally there is a strong interaction between tonal and rhythmic phenomenons. The position and length of a chord determines to a great extent its function and intensity.

The act of finding chords to an existing melody (Figure 2.6) is called ***harmonization***. Though there are several rules-sets for harmonizing melodies, there is rarely one "true" chord-progression and harmonizing remains an artistic act.

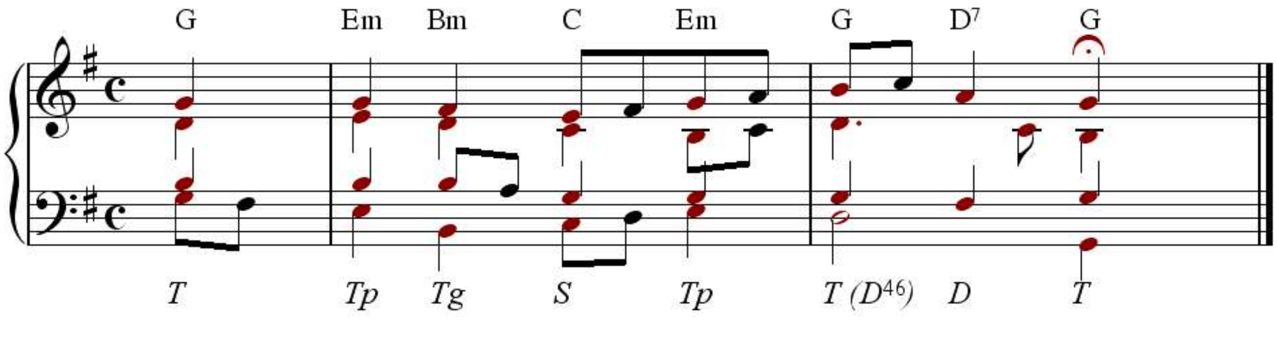


Figure 2.7: J.S. Bach, "Nun laßt uns Gott, dem Herren"

The process of identifying the chords and functions of a polyphonic piece of music is called ***harmonic analysis***. Figure 2.7 shows the final measures of a Bach Choral and the results of its harmonic analysis in the form of absolute chords (above the staves) and functions (below).

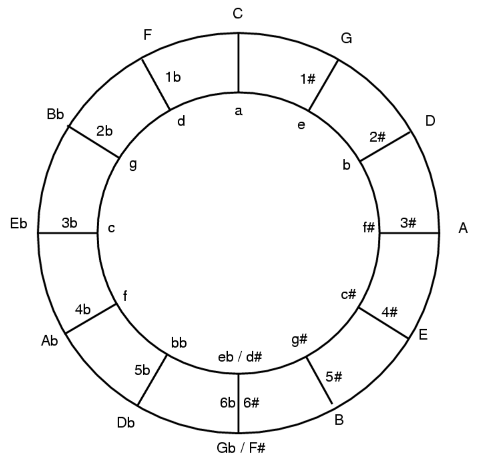


Figure 2.8: Cycle of Fifths

Closely related keys are those, of which the scales share many common tones. A common way to depict the closeness of keys is the circle of fifth shown in Figure 2.8. This figure shows the names of the major keys in the outer circle and their relative minor keys on the inner circle. Adjacent Keys have a fifth (to the right) of fourth (to the left) relationship and share 6 of 7 scale notes. The number of sharps accidentals is also depicted and changes by one for each segment of the circle.

A piece of music need not remain in one key over the whole time. The process of changing the tonal center of a piece of music is called ***modulation***. The most common modulations are those to closely related keys as they share many common tones. Thus modulation to the dominant or subdominant is very frequent, the same as modulation to the relative major or minor. An example for a modulation to the subdominant is the chord-sequence: [C-F-G-C-C^7-F-Bb-C-F–d-Bb-C-F]. The first 4 chord establish the first tonal center C, the next 5 chord modulate to the new tonal center F to which the chord sequence then sticks to.

Short changes of the tonal center are called ***transient modulations*** or "Ausweichungen".The shortest change of tonic center is produced by preceding a chord that is not the tonic with its dominant. This dominant is called ***secondary dominant***. The chord the dominant leads to, is for this short time the tonic. An example for a secondary subdominant is the second chord in the chord progression [C-E-a-F-G-C]. This progression has the tonic center C that changes for the second chord to the tonic center a. The functional symbols for this progression are [T-(D)-Tp-S-D-T], where the braces around the dominant mark it as a secondary dominant relative to the function that follows the closing brace. In the same manner as secondary dominants, ***secondary subdominants*** are subdominants of chords other than the tonic.

2.2.3 Tuning and Temperament

As described in Section 2.1 a tonal signal can be decomposed into several sine with certain frequencies and amplitudes, where the amplitude determines the volume and the frequency determines the pitch. The mapping of frequencies to musical pitches is determined by two factors: the standard pitch (also called concert pitch) and the tuning system.

The ***standard pitch*** is a universal frequency that all instruments are set to. The need for a standard pitch arises when several musicians want to play together on different instruments. Today's standard pitch is a_4 set to 440 Hz.

Different tuning systems exist, these are among others just intonation, meantone temperament, well temperament and equal temperament. In this work we assume that the audio data is in equal temperament which holds for most of contemporary music. In ***equal temperament*** the octave is divided into twelve parts with equal frequency rates. As the octave has a ratio of two, the ratio of frequencies between two adjacent semitones is the twelfth root of two.

To find the frequency of a certain pitch or calculate the pitch that matches a given frequency, each pitch is represented with an integer value, and consecutive semitones have consecutive integer numbers. We use a pitch number of 57 to represent the reference pitch a_4. To find the frequency of a certain pitch the following formula is applied:

$$P_n = P_a \cdot 2^{\frac{n-a}{12}} \tag{2.1}$$

where n is the number assigned to desired pitch and a the number of the reference pitch. P_n is the frequency of the desired pitch and P_a the frequency of the reference pitch.

Example: The frequency of c_4 (57 – 9 semitones = 48) is thus

$$P_{48} = 440 \cdot 2^{\frac{48-57}{12}} = 440 \cdot 2^{\frac{-9}{12}} = 261.626\,Hz \tag{2.2}$$

Given a certain frequency P_n, the associated pitch number n is computed using

$$n = a + {}_2\log\left(\frac{P_n}{P_a}\right) \cdot 12 \tag{2.3}$$

Example: The pitch number of 261.626 Hz is thus $n = 57 + {}_2\log\left(\frac{261.626}{440}\right) \cdot 12 = 48 = c_4$

2.3 Conclusion

This chapter has introduced the reader to the basic acoustic concepts necessary to understand music analysis. We have given an overview of the properties of sound waves and the relationship between pitch, frequency, amplitude and loudness. Special focus was given to harmonic series, describing the properties of acoustic instruments, that never produce one isolated sinus wave – but waves at the integer multiples of the fundamental frequency.

Following the introduction to acoustics, the basic concepts of music theory have been introduced in the second part of this chapter. First we defined the terms pitch, enharmonics, interval and scale. We then have focused on the definition of chords, explaining their notation and structure and giving examples of the most important chord types – major and minor. We have recognized the importance of the key or tonal center, and have investigated on function theory, that describes the relationships between chords and their tonal center. Transient and permanent modulation have been explained, the same as relationships between keys. Finally tuning systems have been defined as mappings between pitch names and frequencies and today's most common tuning system, equal temperament has been described in detail.

We have now gathered the necessary background knowledge to understand existing approaches and design our own chord detection algorithm.

3 Related Works

Automatic chord transcription has been a field of research since the 90s. This chapter gives an overview over existing approaches. It starts with a general section on chord detection algorithm and categorization criteria. Considering these criteria three algorithms have been chosen, that provide a good overview over current approaches to chord description. Each of these algorithms is explained in detail in the subsequent sections. A comparison of the different approaches and their results completes the chapter.

3.1 Chord Detection Algorithms

- Accepted input
  - Input format:
    - Musical Instrument Digital Interface (MIDI)
    - Pulse Code Modulated Data (PCM)
  - Instrumentation:
    - single instruments
    - polyphonic without percussion
    - polyphonic with percussion
- Algorithm
  - Context awareness:
    - short-span methods
    - boundary detection
  - Methods
    - Machine Learning Methods
    - Cognitive Methods
  - Used features

Figure 3.1Chord Detection Algorithm Classifications

Figure 3.1 lists the most important aspects by which chord detection algorithms can be categorized. Restriction on MIDI format or single instruments audio data facilitates chord recognition but makes the developed algorithms applicable only to a limited set of audio data. In this work we want to do chord detection on "real world" signals, so this chapter only treats related works that operate on PCM data of polyphonic music with no restriction on percussive sound. The detection of chord boundaries has been identified in [6] as essential for effective chord detection. Analysis tools with the focus on chords thus consider most often the temporal dimension and chord boundaries. However short-span chord detection often build the foundation for the more complex chord detection algorithm and will thus also be considered here.

3.2 Hidden Markov Models

This section describes the approach of Chord Segmentation and Recognition using EM-Trained Hidden Markov Models introduced by Alexander Sheh and Daniel P.W. Ellis in [7]. This approach uses hidden Markov Models to represent the chord sequences. Given a certain output sequence of pitch class profiles this module allows to estimate the corresponding sequence of chords, that have generated this output. A hidden Markov model (HMM) is a statistical model of a finite state machine in which the state transitions obey the Markovian property, that given the present state, the future is independent of the past.

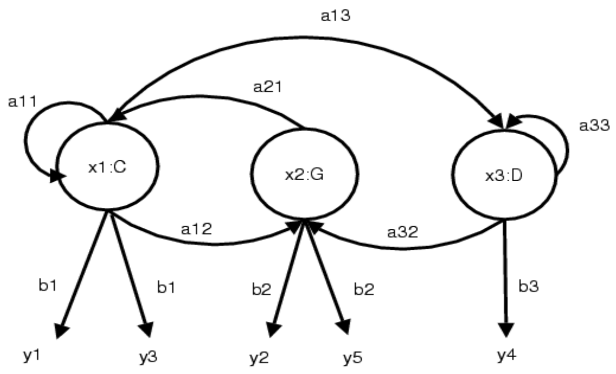


Figure 3.2: Hidden Markov Model for Chord Sequences

Figure 3.2 outlines the hidden Markov model for chord sequences. Each state (x) generates an output (y) with a certain probability (b). Transition probabilities (a) are assigned to each pair of states. While the sequence of states (path) is not visible ("hidden"), the outputs can be observed and used to draw conclusions on the current state.

Sheh and Ellis apply transitions every 100ms. The model has the following parameters:

Each chord that shall be distinguished by the system is one state. The output is defined as the Pitch Class Profile (PCP) of the current 100ms interval. An output probability consists of a Gaussian curve for every pitch class. Transition and output probabilities are unknown at the beginning. To obtain these parameters an expectation maximization algorithm (EM) is applied using hand-labelled chord sequences. Once all parameters of the model have been defined, the Viterbi algorithm is used to find the most likely sequence of hidden states (that is the most likely sequence of chords) that could have generated the given output sequence (PCP sequence).

The authors trained the algorithm with 18 Beatles songs and evaluated it using two other songs from the Beatles with a result of 23% chord recognition accuracy.

3.3 Self-organized Maps

This section describes the approach of Multi-timber Chord Classification Using Wavelet Transform and Self-organized Map Neural Networks introduced by Borching Su and Shyh-Kang Jeng in [8]. This is one of the few approaches that do not use Pitch Class Profiles but evaluate the frequency spectrum directly. The results of a wavelet transform are directly sent to a neural-network chord-classification unit without note identification. The neural network consists of a self-organized map (SOM) with one node (neuron) for every chord that shall be recognizable (Figure 3.3). The nodes are arranged in such a way that adjacent nodes are with high similarity (vertically) or strong relationships (horizontally). Before learning the initial synaptic weights of each neuron on the SOM are set according to music theory. The SOM then learns from a set of training data without supervised information.
The authors report an accuracy rate of 100%. As their test set consists only of 8 measures of a Beethoven Symphony this result can not be regarded neither as representative, as the test set is too homogeneous nor as highly meaningful as the test set simply is too small.

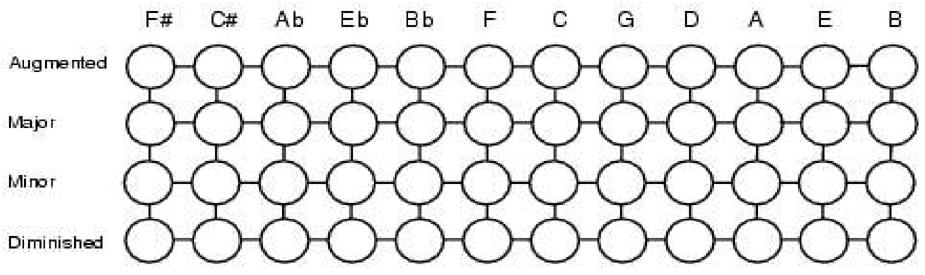


Figure 3.3: Self-organized map used by Su and Jeng in [8] for chord detection

3.4 Hypothesis Search

This section describes an approach of Automatic Chord Transcription with Concurrent Recognition of Chord Symbols and Boundaries introduced in [6] by Takuya Yoshioka, Tetsuro Kitahara, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G. Okuno. The emphasis of the work of Yoshioka et al. is on the mutual dependency of chord-boundary detection and chord symbol identification. They identify this problem as crucial to chord detection. For their solution they do not only use frequency based features but also beat detection and a high-level database of common chord sequences. The heart of this algorithm is a hypothesis-search algorithm that evaluates tuples of chord symbols and chord boundaries. First, the beat tracking system detects beat times. In order to hold execution time down, hypotheses span over no more than one measure-level beat interval, at which time they are either adopted or pruned. The eighth-note level beat time is used as clock to trigger feature extraction, hypothesis-expansion and hypothesis-evaluation. Evaluation considers three criteria:

- Acoustic-feature-based certainty: Pitch Class Profiles (PCP) are generated from the input audio. They are compared to trained mean PCPs. The product of the Mahalanobis distances and a span extending penalty form the acoustic score.
- Chord-progression-pattern-based certainty: The hypothesis is compared to a database of 71 predefined chord-function-sequences, that have been derived from music theory (e.g. V-I = Dominant-Tonic = G-C for key C).
- Bass-sound-based certainty: The authors argue that bass sounds are closely related to musical chords, especially in popular music. Bass sound based certainty is high if the predominant low-frequency pitch is part of the given chord. The higher its predominance the greater the score.

The hypothesis with the largest evaluation value is adopted.

The system was tested on excerpts of seven Pop-songs taken from the RWC Music Database[1] for which an average accuracy of ~77% has been achieved.

1 http://staff.aist.go.jp/m.goto/RWC-MDB/

3.5 Comparison and Summary

Table 3.1 summarizes the different approaches. Sheh and Su (Section 3.2) both use machine learning approaches, while Yoshioka (Section 3.4) implements a hypothesis search algorithm and focuses on additional music theoretical knowledge. Yoshioka operates on chord sequences, while Su operates only on isolated chords. The Hidden Markov Model used by Sheh (Section 3.3) contains transition probabilities but the Markov property restricts the context awareness to the last state, thus chord sequences of maximal two chords are analysed. This combined with the short interval of 100ms makes the algorithm liable to non-chord tones and arpeggio sounds. All algorithms detect major, minor, diminished and augmented chords. Sheh and Ellis also consider seventh chords. Unfortunately we have no information on whether and to what extent the test sets also used these chord types.

Although accuracy rates are reported for all three of these algorithms they can not be compared directly, as they have been observed on different test sets and using different chord types.

Paper	**[7]: Sheh, Ellis**	***[8]: Su, Jeng***	***[6]: Yoshioka et. al.***
Technique	HMM, EM	Neural Networks	Hypothesis, Music Theory
Learning	yes	yes	no
Context awareness	very limited	no	yes
Chordtypes	maj, min, aug, dim, maj7, min7, dom7	maj, min, dim, aug	maj, min, dim, aug
Test set	2 songs (Beatles)	8 measures (Beethoven)	7 songs (various Pop)
Test set genre	Pop	Classic	Pop
Accuracy	20,00%	100,00%	77,00%
Representative?	limited (Beatles)	no	yes

Table 3.1: Related Works Summary

The least helpful approach surely is the one by Su and Jeng, not because of the approach itself but because of its inaccurate evaluation and the shortness and lack of detail of the paper. The approach by Sheh and Ellis is interesting as it tunes itself and learns from its test set. As it does not use any music theoretical principles it still is of only limited interest for our work. Yoshioka et al. have developed the most interesting algorithm for us as it widely uses music theory and has been evaluated on a comparatively large test set with good results. They do not use key information directly but operate on a database of common chord progression. A less complex algorithm that filters the chords directly according to the computed key might bring just as good results.

3.6 Conclusion

In this chapter we have given an overview over existing types of chord detection algorithms. We have picked out three algorithms and have presented them in detail. Finally the different algorithms and their evaluation results have been compared, as far as this was possible considering the strong differences between the test sets. We defined the approach of Yoshioka et al. as the most interesting one for this thesis, as it incorporates music theoretical knowledge in its detection method. Considering the smallness of the used test sets of all described approaches it becomes apparent that creating test sets for chord detection is a quite unpopular task. The true chord sequences have to be detected manually and each chord has to be assigned to a specific time within the piece of music, which is very time consuming. Nevertheless a large test set is of course desirable as it raises the significance of evaluation. As a consequence, one goal of this thesis is to assemble a large test set of heterogeneous data and to make the hand labelled chord sequence files available to other researchers.

4 Conceptual Design

The previous chapter gave an overview over different existing approaches to chord detection and described their strengths and weaknesses. This chapter now introduces a new chord detection algorithm that has been designed for this thesis and that reuses the idea of chord boundary detection introduced by Yoshioka et al. [6]. The first section gives an overview over the developed chord detection algorithm. Subsequently each module of the algorithm is described in detail: Section 4.2 deals with frequency detection, Section 4.3 describes the Pitch Class Profile, its generation and evaluation. The key detection algorithm is described in Section 4.4 and the approach to tempo and beat detection is outlined in Section 4.5. Section 4.6 finally deals with chord sequence optimization (smoothing).

4.1 Overview

Figure 4.1 depicts the flow chart diagram of our chord detection algorithm. The boxes in the first and last row represent the input and output data of the algorithm. The modules in the second and third row deal with feature extraction while the subjacent modules analyse these features.

The algorithm takes audio data as input and outputs a sequence of time-chord pairs. Before computing the chords themselves, the audio data is used to extract two other relevant characteristics: the beat structure (see Section 4.5) and the key of the song (Section 4.4). Former is used to split the audio data into blocks of sizes that correspond to the computed beat structure. Each of the obtained blocks is passed to an enhanced autocorrelation-algorithm (Section 4.2) which calculates the frequency spectrum and returns a sequence of frequency-intensity pairs. These are then used to compute the intensity of each pitch class (see definition in Section 2.2), the so called Pitch Class Profile (PCP) (Section 4.3).

The calculated PCP's are compared to a set of reference chordtype-PCP's using only those reference chords that fit to the key of the song. The possible chords are sorted according to the distance of their reference PCP to the calculated PCP. The chords and their probabilities are accumulated and stored in the chord sequence data structure. A smoothing algorithm (Section 4.6) is applied to the chord sequence which rates each chord according to the number of chord changes around it. Finally for each timespan the chord with the highest score is taken.

The algorithm has a modular design. The modules beat detection, key detection and smoothing can

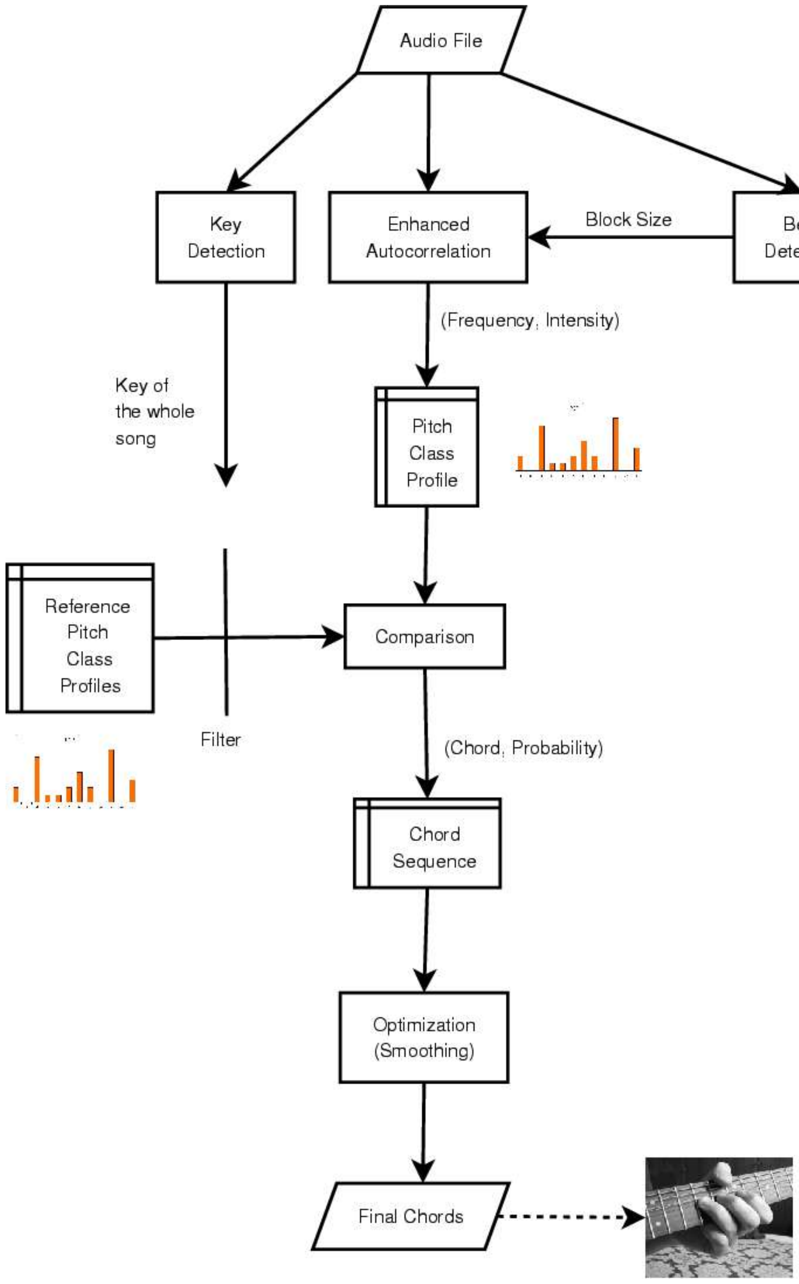


Figure 4.1: Flow Chart of the Chord Detection Algorithm

be turned on and off, thus allowing to measure their influence on the result. If the beat detection module is turned off, the audio file is split into equally sized blocks of configurable length (e.g. 100 ms). If the key detection module is switched off, the reference pitch filter is turned off and all possible chords are considered. Finally turning off the smoothing algorithms leads to the output of those chords with the minimal distance between observed PCP and reference PCP, that is the first row of the chord sequence matrix.

4.2 Frequency Detection by Enhanced Autocorrelation

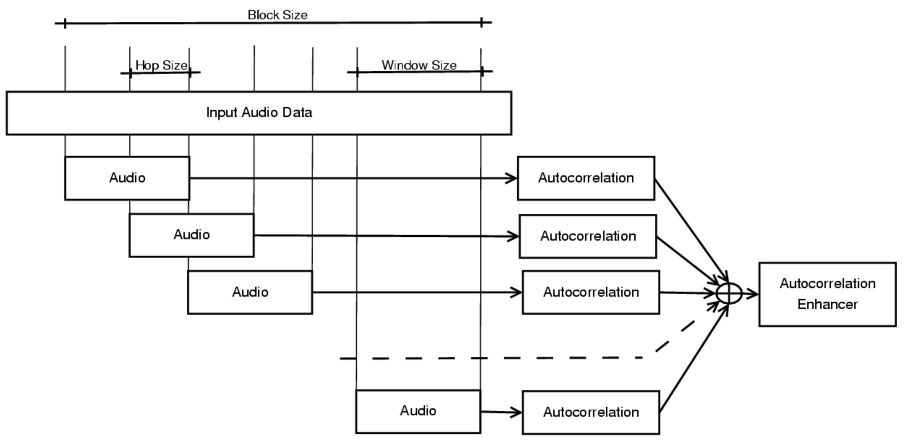


Figure 4.2: Enhanced Auto Correlation: Input

As described in Section 4.1, the frequency detection lies at the basis of our chord detection algorithm. A chord is an accord of specific notes that last for a certain time. The detection of the pitches, that is at the first instance the detection of the frequencies, that occur over the time is therefore fundamental for our chord detection algorithm. While there exist a variety of works on primary frequency detection ([9], [10], [11], [12], [13]), the research on multipitch analysis is still rather limited. From the existing approaches described in [14], [15] and [16] we have chosen the approach of Tolonen [16], which offers a good trade-off between complexity and quality.

This algorithm calculates an enhanced autocorrelation (EAC) of the signal, taking the peaks of the EAC as the pitches of the signal. In this algorithm, the signal is first split into two channels, one below and one above 1000 Hz. The high-channel signal is half-wave rectified and also lowpass filtered. Frequency detection is then performed on both channels using time-lag correlation, called "generalized autocorrelation" (AC). More specifically the computation consists of a discrete Fourier transformation (DFT), magnitude compression of the spectral representation, and an inverse transformation (IDFT): $AC = IDFT(|DFT(x)|^k)$ where k is 2/3. The autocorrelation results are then summed up and passed to the autocorrelation enhancer which clips the curve to positive values and prunes it of redundant and spurious peaks. For instance, the autocorrelation function generates peaks at all integer multiplies of the fundamental period. By subtracting a time-scaled version from the original autocorrelation function, peaks of a multiple frequency that are lower than the basic peak are removed. Furthermore the extreme high frequency part of the curve is clipped to zero. More details about this algorithm and its calculation steps can be found in [16].

The integration of the EAC function into our system is illustrated in Figure 4.2. The EAC function is applied to non-overlapping successive blocks of the input data. The size of each block is calculated using the Beat Detection Algorithm described in Section 4.5. Each input block is split into equally sized overlapping windows. In our tests a Hamming window of 46.4 ms (that is 1024 samples for a sampling rate of 22050 Hz) and a window-overlap of 50% which equals to a hop size of 23.2 ms gave the best results. The autocorrelation function is calculated for each window, but in contrast to Tolonen we only consider the lowpass filtered signal in order to reduce the complexity of the algorithm. The sum of the autocorrelation results is then enhanced as described above and passed to the Pitch Class Profile Generator described in Section 4.3.

An example output of the enhanced autocorrelation (EAC) is shown in Figure 4.3 At the top the autocorrelation curve is depicted. The graph on the bottom shows the enhanced autocorrelation function. You can see that the EAC is clipped to positive values and does not contain the peak around the origin. The peak at 110Hz in the AC curve has been removed by the enhanced algorithm as it a a result of the peak at 55 Hz. Figure 4.4 compares the EAC of the original unfiltered data to the EAC of the data passed through a lowpass filter. You can see that the lowpass filtered signal produces a calmer, smoothed autocorrelation result than the original signal and that all peaks with frequencies greater or equal to 440 Hz have been removed by the lowpass filter.

The enhanced autocorrelation function has a logarithmic frequency scale. Thus it provides less information for high pitches than for low ones. Assuming a sampling rate of 44100 Hz, the spacing between two semitones varies from over 100 indexes for frequencies beneath 30 Hz to less than one index for frequencies above 4000 Hz. The frequency range depends on the window size and the sampling rate:

$$frequency\,range = \left[\frac{samplerate}{windowlength/2}, samplerate \right] \tag{4.1}$$

The mapping form the indices in the EAC vector to frequencies are calculated by the following formula:

$$frequency = \frac{samplerate}{index} \tag{4.2}$$

The conversion from frequency to pitch name is done using the formula 2.3. For more details on frequency to pitch mapping and tuning systems see Section 2.2.3.

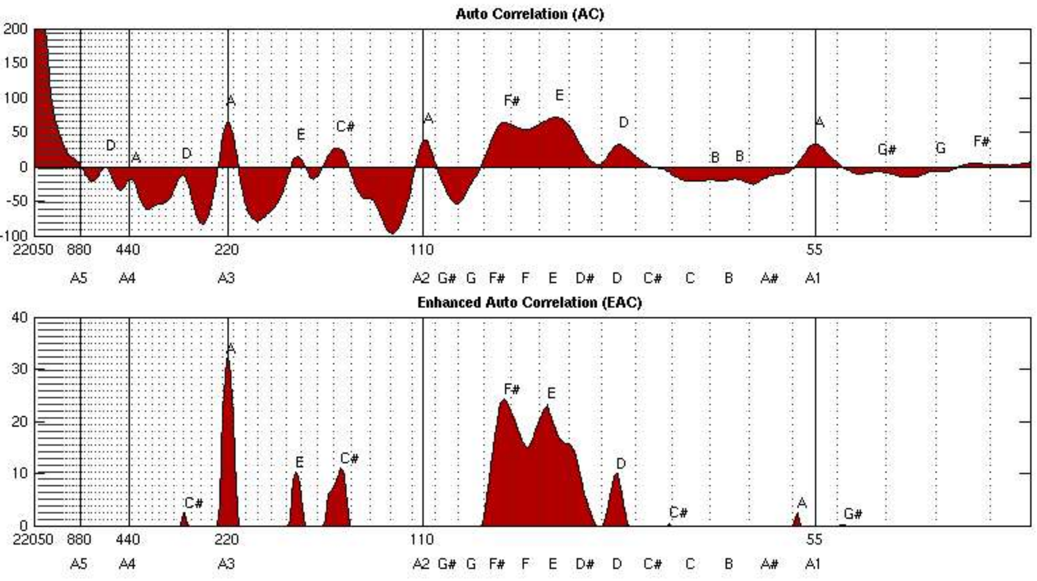


Figure 4.3: AC, EAC

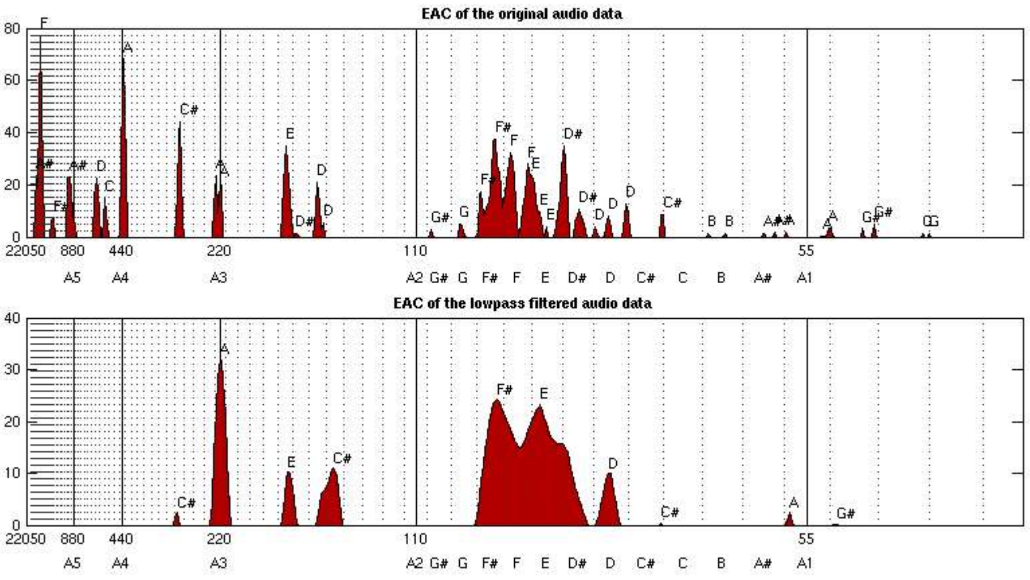


Figure 4.4: EAC: original data, lowpass filtered data

4.3 PCP and Reference PCPs

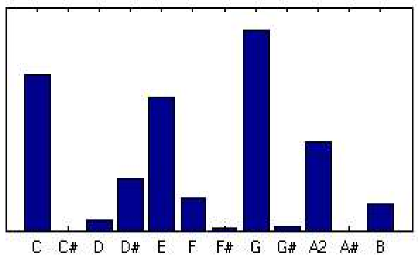


Figure 4.5: Pitch Class Profile (PCP)

This section covers the conversion of the frequency spectrum into a Pitch Class Profile (PCP) and its analysis. A PCP is a vector of twelve elements, each representing the energy of the signal at the pitch class of one semitone. It concisely characterizes an audio signal in terms of occurring notes and harmonies. The PCP is sometimes also called Folded Pitch Histogram ([17]) or Chroma Vector ([6]).

The previous section explained how to convert an audio-signal into a frequency spectrum. The frequency spectrum gives us very detailed information of the energy distribution over frequency at the price of a high data volume. Before analysing its content we thus transform it into a smaller and more compact representation, the PCP. The conversion from the frequency spectrum to a PCP introduces two layers of abstraction:

- Abstraction from the exact frequency: We now consider frequency bands, where each band represents one semitone.
- Abstraction from the octave: All pitches are mapped to a single octave, the pitch class.

The twelve pitch classes are numbered in ascending order from 0 (c) to 11 (b) according to the MIDI note numbering scheme. A reference frequency of 440 Hz is used, which conforms to today's concert pitch a_4. We assume that the analysed music is in equal temperament, which is universally adopted today in western music (for detailed information on temperament and tuning see Section 2.2.3). The conversion from frequency to pitch class is done using the following equation:

$$PitchClass(freq) = Pitch(freq)\, modulo\, 12 \quad (4.3)$$

$$Pitch(freq) = \left(57 + {}_2\log\left(\frac{freq}{440}\right) \cdot 12 \right) \quad (4.4)$$

where *freq* is the frequency in Hertz. The mapping of the pitches to one single octave is performed by modulation of the pitch by 12. An example of a PCP is shown in Figure 4.5.

4.3.1 PCP Generation

The EAC for a window of size *windowlength* outputs a vector of size *windowlength/2*. For the computation of the PCP we do not consider the whole frequency range of the EAC but use a minimal frequency f_{min} of 52 Hz (g#$_1$) and a maximal frequency f_{max} of max((*samplerate*/25),3520) Hertz, thus covering 4 octaves for a samplerate of 22050 Hz. For orientation, the average human can hear from 16 to 20000 Hz and the standard range of a piano covers 7 octaves from 27.5 Hz (a$_0$) to ~4186 Hz(c$_7$). The restriction to 4 octaves is due to the inaccuracy of the EAC at extreme frequencies. This inaccuracy can be explained by the logarithmic scale of the EAC, where the distance between two semitones decreases continually. (e.g. for a sample rate of 22050, b$_6$ is at index 11, but index 10 corresponds already to c#$_7$, so c$_7$ would be skipped). Table 4.1 lists the values of f_{max} for common sampling rates.

samplerate	f_{max}	***pitch***
11025	441	a$_4$
22050	882	a$_5$
44100	1764	a$_6$

Table 4.1: fmax

For the computation of the PCP we propose two algorithms: an integration approach and an algorithm that only considers the peaks in the spectrogram.

Algorithm1: Integration

In this algorithm, the PCP is computed by summing up the spectral energy at the frequencies that correspond to the same pitch class. As discussed, only the spectral information at frequencies in [f_{min}, f_{max}] is considered.

$$PCP[i] = \sum_{j \in PitchClassIndices(i)} EAC[j], i = 0 \ldots 11 \tag{4.5}$$

$$PitchClassIndices(i) = \left\{ x \in \left\{ round\left(\frac{samplerate}{f_{max}}\right), \ldots, round\left(\frac{samplerate}{f_{min}}\right) \right\} \mid PitchClass(Frequency(x)) = i \right\} \tag{4.6}$$

The PCP Generation in pseudo code:

```
for i in {0 ... windowlength/2}: PCP[PitchClass(Frequency(i))] += EAC[i]
```

Algorithm 2a: Simple Peak Detection

Instead of summing up the whole spectrum, this algorithm only considers the spectral peaks. The local maxima of the spectrum are computed and the energies at the local maxima are summed in the corresponding PCP entries:

$$PCP(i) = \sum_{j \in \text{PCPeaks(i)}} EAC[j], i = 0 \ldots 11 \tag{4.7}$$

$$PCPeaks(i) = PitchClassIndices(i) \cap arg\,localmax(EAC) \tag{4.8}$$

Algorithm 2b: Neighbouring Peak Detection Algorithm

This strategy however is highly prone to tuning errors, as peaks at the borders of two pitches are always fully assigned to only one pitch. To overcome this weakness an improved algorithm considers *N* left and right neighbours of the peak using:

$$PCP[i] = \sum_{j \in PCPeakNeighbourhood(i)} EAC[j], i = 0 \ldots 11 \tag{4.9}$$

$$PCPeakNeighbourhood(i) = PitchClassIndices(i) \wedge j\, near\, arg\,localmax(EAC) \tag{4.10}$$

$$x\, near\, S : \{x - N, \ldots, x, \ldots, x + N\} \cap S \neq \emptyset \tag{4.11}$$

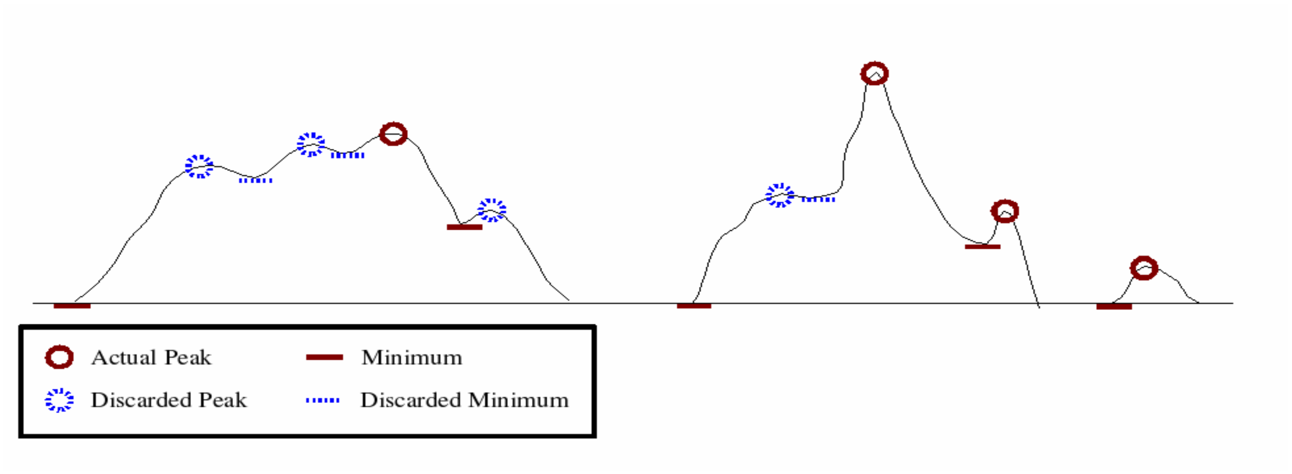


Figure 4.6: Peak Picking

where *N* is set to 3 but could also be a function of the index x.

The algorithm is further adopted to consider only peaks that stand alone or have a certain height relative to the surrounding local minima (Figure 4.6). A dynamical threshold that is 10 percent of the absolute maximum in the spectrum is used.

Comparison of the different generation algorithms

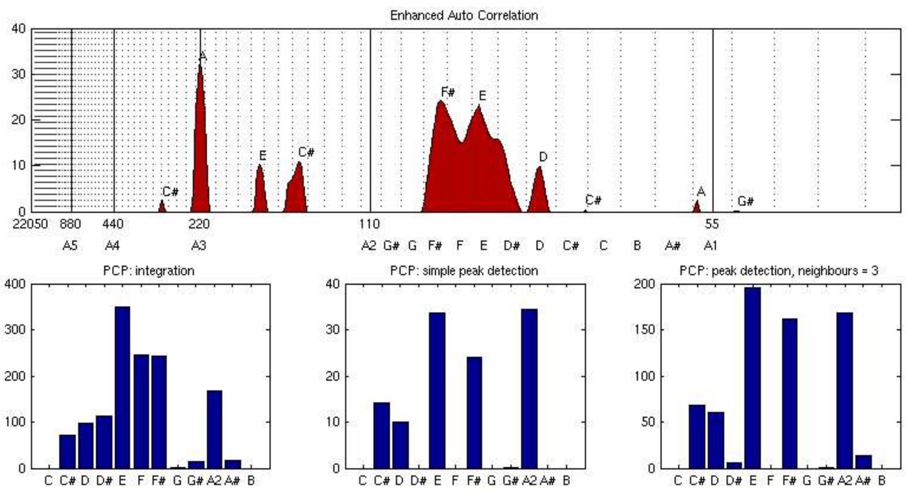


Figure 4.7: Comparison of different PCP Generation Algorithms

Each of the described algorithms has its advantages: the integration algorithm is more resistant to mistuning, but is blind to the context of the values. It does not make any differences between high values that are peaks and those that are only in the wider surrounding of a peak. Besides it favours low frequencies, as they have a greater range in the spectrogram than high frequencies. The simple peak algorithm, on the other hand, is context sensitive but ignores tuning errors. Finally the neighbouring peak detection algorithm favours broad peaks, as their neighbourhood adds to a higher sum than that of sharp peaks.

Figure 4.7 shows the EAC for a timespan of 2 seconds in the song “Abba – Dancing Queen” and the corresponding PCPs that have been calculated with the described algorithms. You can see that the integration algorithm produces a high value at 'f' whereas the other algorithms do not register any energy at this pitch class. Comparing the PCP of the two peak algorithms, you can see that the advanced peak algorithm assigns the peak at 56 Hz partly to a#, as it is at the border of the two pitches whereas the simple algorithm doesn't record any activity at a#.

No matter which of the PCP generation algorithms is used, the resulting PCPs are always normalized to values between zero and one. The normalized PCPs are then passed to the Analysis module.

4.3.2 PCP Analysis

The PCP are compared to a set of reference chord PCPs. This is done by calculating the linear distance between the two normalized vectors. The smaller the distance, the higher the resemblance of the two PCPs. The reciprocal of the distance is then stored as a reference value for the probability that the chord at this timespan matches. Finally, n chords with the highest probability are stored in the chordsequence-matrix.

4.3.3 Reference PCPs

A reference PCP is a typical PCP for one special chord. There are different methods to obtain reference PCPs, the most prominent being derivation from music theory, derivation from probe tone ratings and derivation from training data. In the first case the PCP is deduced from theoretical rules. According to music theory a major chord contains major third and perfect fifth. Thus the reference PCP would be (1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0). In the second method, also called "cognitive method", the PCP is built using the perceived stability of each semitone within the context of a particular chord or key. This is obtained by confronting listeners with a key context followed by probe tones, which have to be rated according to their perceived fittingness to the key context. Results from experiments conducted by Carol Krumhansl, and further information on cognitive methods can be found in [18]. For reference PCP derivation from training data, correct chord names are manually assigned to a set of music, the training data. From the training data the PCPs are generated in the same way as for chord detection itself. For each PCP its chordtype is looked up in the hand-labelled chord files. All PCPs of the same chordtype are transposed to the same root (c) and summed up in one chordtype PCP, which normalized is the new reference PCP.

In this work we used the training approach to obtain the reference PCPs, as it performed better than derivation from music theory and cognitive PCPs. The trained reference PCPs have been adapted manually to remove biases for special chordtypes using the following rules:

- The total sum of the different chord-type PCPs must not vary.
- If only the root note is present in a PCP, the reference PCPs of all chordtypes shifted to this root must have the same distance. Hence the value for the root pitch must not vary.

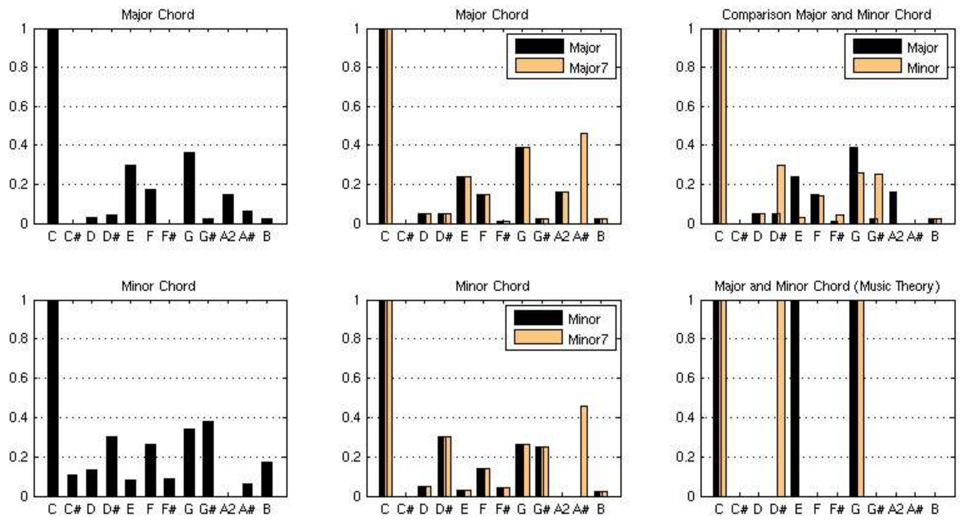


Figure 4.8: Generated Reference PCPs

This work distinguishes two chord types: Major and Minor. Reference PCPs of the major and minor seventh chord are also used, though in evaluation stage the confusions between the standard and the seventh version of the chords are not counted as an error. Figure 4.8 shows the reference PCPs for C-Major and C-Minor and the corresponding values deduced from music theory. A clear predominance of the root note can be observed, the same as a surprisingly high value at the minor sixth (g#) in the minor chord.

For each chord type one single reference PCP is stored. The 12 chords of this chord type can then be obtained by shifting the PCP by one step for each semitone.

$$PCP[(i+root)\, modulo\, 12]=cPCP[i], i=0..11$$

4.4 Key Detection

The key detection works similar to the first stage of the chord detection, in that it reuses the calculation of the enhanced autocorrelation and the generation of a Pitch Class Profile. As described in Section 2.2.2 a song of a special key is characterized by the scale of this key. That is, if a song is performed in Key c, the pitches of the scale c, namely 'c d e f g a b c' will be the dominant pitches. Pitches that are not part of the scale like 'c# or g# ' are much more unlikely to occur as they produce dissonance.

Assuming this the reference PCPs for major and minor keys are derived from the major and minor scale of key c. In contrast to the reference chord PCPs the reference key PCPs have not been trained but determined empirically. The values proposed by Krumhansl in [18] have been tested, but brought a slightly worse result. Starting from the two reference PCPs, the 24 possible keys (12 major, 12 minor ones) are again derived by shifting the reference PCPs by the number of semitones between c and the root note of the desired key.

As, according to the assumptions made in Chapter 1, the key of the input data does not modulate, the whole song can be used for the calculation of the one key. Nevertheless, in order to speed up calculation, we only use certain chunks of the audio file, namely the first and last 15 seconds. Another possibility would have been to take x random parts of size y, where x*y results in the same amount of analysed time. We preferred the first method, as empirically the beginning and end of a song usually establish and enforce the harmony of the song, and thus stick more rigidly to the scale than middle parts, where small modulations are encountered more often. This theory, though, has not been tested and it would certainly be interesting to compare the accuracy of key detection of these two approaches and of a computation that uses all of the available data.

Once the key of the song has been determined the set of possible chords is filtered according to harmonics. The following chords pass this filter: chords that are proper to the scale, except the diminished minor chord on VII, which is heard as V7. Furthermore all secondary dominants except the dominant of III and all secondary subdominants. So from 24 possible chords (12 major and 12 minor chords) 10 chords are preselected to build the set of possible chords for this song. Table 4.2 shows the table of chords for Key C-Major. The chords are highlighted dark grey when they are first selected. Chords that have already been selected are highlighted light grey. Chords that are not selected remain black on white.

pitch(function)	***c(T)***	***d(Sp)***	***e(Dp)***	***f(S)***	***g(D)***	***a(Tp)***	***b***
chords proper to the scale	C	d	e	F	G	a	G7(b-)
Secondary dominants	G	A	B	C	D	E	
Secondary subdominants	F	G	A	Bb	C	D	

Table 4.2: Chords for C-Major

By adding the secondary dominants and subdominants even modulations in the neighbouring of the key do not disturb our algorithm.

Note, that confusions between a major and its corresponding minor key (e.g. C-Major and A-Minor) do not affect the chord detection algorithm, as both keys are connected to the same set of chords. Also note that confusions of a key with its neighbours in the circle of fifth are not as severe as confusions with keys that are wider away, as in the first case the two keys have more chords in common. Table 4.3 illustrates the similarities of nearby keys. The columns represent the chords in quint-ordering (minor chords in lower-case, major chords upper-case), the rows are the keys, which are also in quint ordering. Each association between a key and a chord is marked. You can see that neighbouring keys as C and G have 8/10 chords in common whereas C and D share only six chords, C and A four chords, C and E three chords, F and E only two and Bb and E finally only one chord, that in addition has only a secondary function for both keys.

	Eb	***eb***	***Bb***	***bb***	***F***	***f***	***C***	***c***	***G***	***g***	***D***	***d***	***A***	***a***	***E***	***e***	***B***	***b***	***F#***	***f#***	***C#***	***c#***	***G#***	***g#***
Bb	🟧		⬛		🟧		▨	🟧	▨	🟧	▨	🟧											▨	
F	▨		🟧		⬛		🟧		▨	🟧	▨	🟧	▨	🟧										
C			▨		🟧		⬛		🟧		▨	🟧	▨	🟧	▨	🟧								
G					▨		🟧		⬛		🟧		▨	🟧	▨	🟧	▨	🟧						
D							▨		🟧		⬛		🟧		▨	🟧	▨	🟧	▨	🟧				
A									▨		🟧		⬛		🟧		▨	🟧	▨	🟧		🟧		
E											▨		🟧		⬛		🟧		▨	🟧	▨	🟧		🟧
legend			⬛	Tonic			🟧	proper to scale					▨	secondary chords										

Table 4.3: Keys and associated chords

4.5 Tempo and Beat Tracking

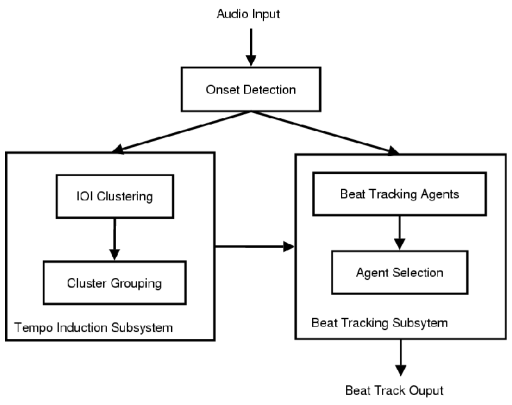


Figure 4.9: BeatRoot: System Architecture

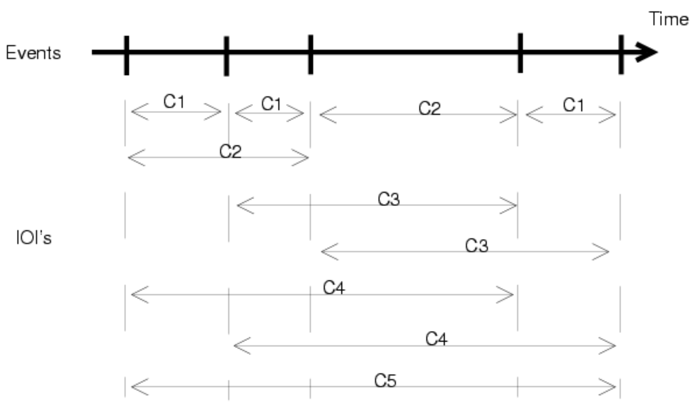


Figure 4.10: BeatRoot: Inter-onset intervals and clustering to groups C1-C5

Instead of slicing the song into equally-sized blocks of data, a tempo and beat tracking algorithm is used to obtain the beat of the song. The complex task of tempo detection or beat tracking is best described as correspondent to the human activity of foot-tapping in time with music. The song is then sliced into blocks spanning from one beat time to the next. As we know that chord changes usually occur at beat time, this is an efficient method to lengthen the span of the audio-blocks without risking windows that span over two or multiple chords.

Simple beat detection algorithms as described in [19] detect changes in the amplitude of music data. More precisely, they detect sound energy variations by computing the average sound energy of the signal and comparing it to the instant sound energy. More advanced algorithms detect energy variations not in the time domain but in frequency subbands. However, these algorithms, further called onset detection algorithms, do not bring these events into a common context, the tempo.

For this purpose the BeatRoot[2] program, is utilized, which rated best in the "MIREX 2006 Audio Beat Tracking Evaluation[3]". The system architecture of this algorithm is shown in Figure 4.9. The onset time of musical events (notes, percussive beats) is calculated by finding peaks in the spectral flux. More specifically, the signal is passed through a short time Fourier transformation and the spectral flux is calculated by summing up the changes in magnitude for each frequency bin where the energy is increasing. The local maxima of the spectral flux then represent the onset and possible beat times.

2 http://www.ofai.at/~simon.dixon/beatroot/index.html

3 http://www.music-ir.org/mirexwiki/index.php/MIREX_2006

The onset times are passed to a tempo induction algorithm that calculates initial hypotheses for the tempo. This is done by computing the inter-onset intervals (IOIs), that is the time interval between any pair of, not necessarily successive, onsets. The IOIs are then clustered into groups with approximately the same length (Figure 4.10), generating first indications for the tempo.

Finally control passes to the beat tracking subsystem. This consists of multiple beat tracking agents: Each agent is initialized with a tempo from one IOI cluster and a first beat time (phase) from the first few onset times. From the beginning of the music to the end the agent then predicts the next beat. Onsets which correspond to an inner window of predicted beat times are taken as actual beats and used to adapt the agent's tempo and phase.

An evaluation function is used to rate each agent according to its ability to predict beats correctly, the salience of the matched events and on how evenly the beat times are spaced. The beat track of the agent with the highest score is finally taken as the solution to the beat tracking problem.

More details about the algorithm and its evaluation can be found in [20], [21] and [22].

4.6 Chord Sequence Optimization

The previous sections described how to obtain for a certain timespan a list of possible chords and their probabilities. In this section we will introduce an algorithm which selects from these chords not simply the most probable, but the best fitting. This is achieved by analysing the context of each chord. We have obtained the chords at beat time interval, which we declared as the shortest timespan in which a chord change happens. Although chord changes may happen at each beat time they most often last for a longer period of time. Thus, the idea behind this algorithm is to select the chord sequence with high probabilities for each single chord and few chord changes.

First we assemble the detected chords to a chord sequence matrix where each column represents the possible chords for one specific timespan. Row n contains the nth best chords of all segements. $C_{2\,5}$ for example is the 5th best chord at timespan two.

$$Chordsequence = \begin{matrix} cp_{11} & cp_{21} & \cdots & cp_{m1} \\ cp_{12} & cp_{22} & \cdots & cp_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ cp_{1k} & cp_{2k} & \cdots & cp_{mk} \end{matrix} \qquad (4.12)$$

Then for each timespan i we calculate all possible permutations with repetition of chords around i. The length of the permutation is always odd, as we take w neighbours to the left and to the right of i. Thus the length W is w*2+1. Say k is the number of chords that we take into account for each timespan, then the number of permutations with repetition ^{R}P is

$$^{R}P_{k}^{W} = \mathrm{k}^{W} \qquad (4.13)$$

and the set of permutations is defined as

$$^{R}P^{W}(\{1\ldots k\}) = \{(x_1, \ldots, x_W) \mid x_i \in \{1\ldots k\}\} \qquad (4.14)$$

$$^{R}P^{W}(Chordsequence, i) = \{(cp_{(i-w)v_0}, \ldots, cp_{iv_w}, \ldots, cp_{(i+w)v_W}) \mid v \in {}^{R}P^{W}(\{1,\ldots,k\})\} \qquad (4.15)$$

$$\text{example}: {}^{R}P^{3}(\{1,2\}) = \{\{1,1,1\}\{1,1,2\}\{1,2,1\}\{1,2,2\}\{2,1,1\}\{2,1,2\}\{2,2,1\}\{2,2,2\}\}$$

For length W=3 and k=2 for example one possible permutation is to chose the 2nd best chord at i-1, the best chord at i and the 2nd best chord at i+1. The greater W is, the greater the context sensitivity of the algorithm. In this work we found a W of 5 to give the best trade off between quality and execution time. With increasing Ws execution time increases exponentially due to the number of permutations that have to be computed (see Formula 4.13).

Now that we have constructed the possible permutations, we calculate a total acoustic probability for each permutation. This is done by multiplying the probabilities of each chord with a chord-change penalty.

$$p(cp_1,\dots,cp_n)=\prod_{i=1}^{n} Prob(cp_i)\cdot \mathrm{C}^{SumChordChanges(cp_1,\dots,cp_n)} \tag{4.16}$$

$$SumChordChanges(cp_1,\dots,cp_n)=\sum_{i=1}^{n-1} ChordChange(Chord(cp_i),Chord(cp\,i+1)) \tag{4.17}$$

$$ChordChange(c_1,c_2)=\begin{Bmatrix} c_1=c_2\colon & 1 \\ c_1\neq c_2\colon & 0 \end{Bmatrix} \tag{4.18}$$

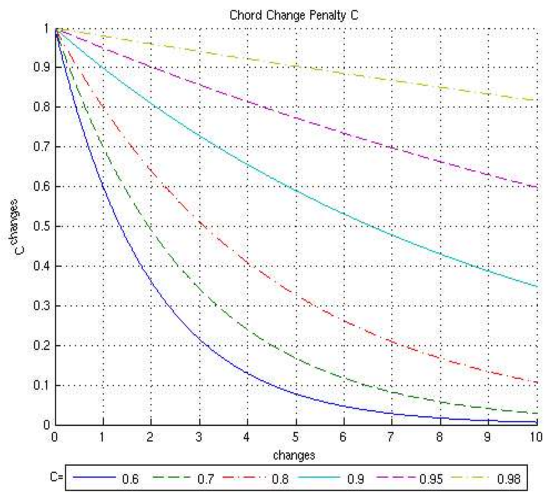


Figure 4.11: Chord Change Penalty

Since we want to penalize chord changes, C must be between zero and one. Figure 4.11 depicts the penalty curves for different values of C. For a W of 5 we found a C of 0.9 (turquoise line) to give good results. Note that the number of chord changes is between zero and W-1, as between W chords there can not be more than W-1 changes.

From the permutation with the highest total probability the element in the middle is chosen to be the final chord at timespan i. This procedure is repeated to cover the whole song except the first and last w time spans.

$$opt(Chordsequence, i) = arg\,max_w\, p(cp_{(i-w)v_0}, \ldots, cp_{iv_w}, \ldots, cp_{(i+w)v_w}), \\ v \in {}^{R}P^{W}(\{1, \ldots, k\}), i = w, \ldots, (m-w) \tag{4.19}$$

Example

We illustrate the optimization algorithm with a short example. For clarity we set k only to 2 and W=5. Table 4.4 shows an an excerpt of eight spans of a Chordsequence. The cells contain the chord names and the probabilities as percentage.

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***
Dm = 75	Dm = 88	Bb = 79	F = 89	C = 88	F = 89	C = 89	F = 90
F = 72	F = 80	Dm = 74	Bb = 80	F = 85	C = 75	C = 82	F = 80

Table 4.4: Exemplary Chordsequence

Table 4.5 lists some of the permutations around i=3, the respective number of chord changes, and their probabilities. The chordsequence 'Dm-Dm-Dm-F-F', highlighted in Table 4.5, which has only one chord change between Dm and F ranks best. For timespan i the final chord is thus Dm.

1	***2***	***3***	***4***	***5***	***Chord Changes***	**Π *Prob***	***p***
1, Dm	1, Dm	1, Bb	1, F	1, C	3	0.408	0.297
1, Dm	1, Dm	1, Bb	1, F	2, F	2	0.394	0.319
1, Dm	1, Dm	1, Bb	2, Bb	1, C	2	0.367	0.297
1, Dm	1, Dm	2, Dm	1, F	1, C	2	0.383	0.310
1, Dm	1, Dm	2, Dm	1, F	2, F	1	0.369	0.332
1, Dm	2, F	1, Bb	1, F	1, C	4	0.371	0.244
2, F	2, F	1, Bb	2,Bb	1, C	2	0.320	0.259
2, F	2, F	2, Dm	1, F	2, F	2	0.322	0.261
...	...	...	...	...	...	...	...

Table 4.5: Example Optimization Algorithm

4.7 Conclusion

We have presented a new design for a chord detection algorithm. This algorithm operates on audio signals from which it calculates frequency spectra using enhanced autocorrelation. We have shown how the input can be split into blocks of data that correspond to the beat-level interval of the song. We have introduced different algorithms for converting the frequency spectra to pitch class profiles (PCP) and an algorithm that computes the key from those PCPs. Afterwards we have proposed a filter based on music theoretical knowledge that filters possible chords according to the detected key. Finally a chord sequence optimization algorithm was introduced that reduces chord changes by rating each chord-possibility by its neighbouring chords.

5 Implementation

Starting from the conceptual design described in Chapter 4, a chord detection program named *genchords* and a set of helper tools have been implemented. This chapter describes the performed implementation. It is divided into three sections: Section 5.1 provides an overview of the system, the different implemented software tools and their interaction. Section 5.2 gives detailed information on the chord detection program genchords itself. The subsequent section 5.3 describes tools that have been designed to facilitate evaluation and exploitation of the detected chords.

5.1 System Overview

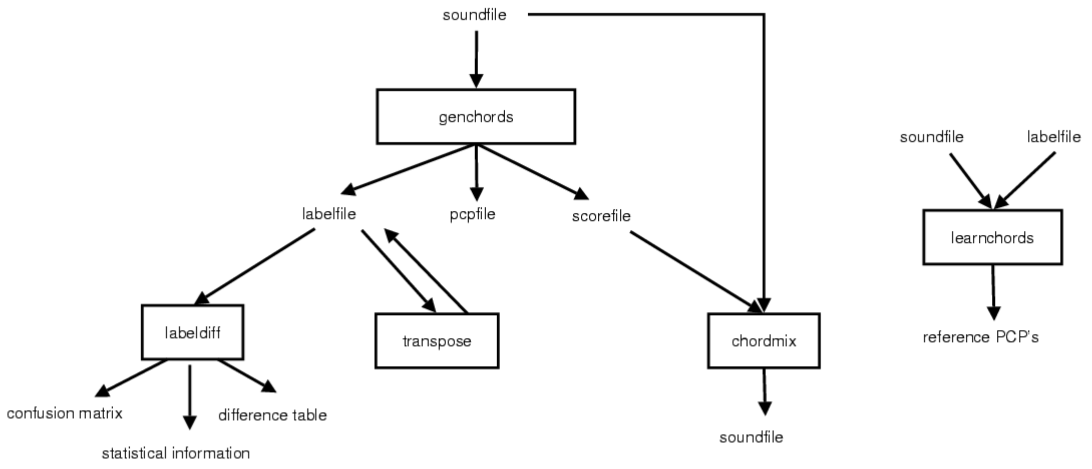


Figure 5.1: Implementation Overview

The chord detection algorithm designed in Chapter 4 has been implemented on a Linux platform using C++, Python and Shell scripts. The implementation consists of the main chord detection program, called genchords, and four helper programs called *labeldiff*, *transpose*, *chordmix* and *learnchords*. The way these programs interact is depicted in Figure 5.1: Genchords takes a soundfile and computes the corresponding chord sequence. The chord sequence is outputted in two different formats: A Labelfile, that consists of lines containing starting time and chord name, and a scorefile of the chords in the syntax used by the synthesis software csound[4]. The labelfile can then be transposed to another key with the script tranpose. Chordmix and labeldiff serve evaluation purpose. Labeldiff compares two labelfiles in terms of alikeness. Chordmix processes the scorefile generated by genchords. It outputs a soundfile containing the synthesized scorefile on the left and the original soundfile on the right channel. Finally learnchords can be used to obtain mean reference PCPs from a given soundfile and its chord sequence.

4 http://www.lakewoodsound.com/csound/

Implementation issues

Careful considerations have been given to the choice of the implementation approach. Essentially three different options have been considered: Usage of a computer audio research framework like Marsyas[5] or CLAM[6], usage of a numerical computation environment like MATLAB[7] or usage of a general purpose language like C++ or Java. Each of these approaches has its own advantages and disadvantages. The use of a numerical computation environment provides a large number of libraries and functions, supports rapid development by providing a high abstraction level and a syntax tailored for numerical computations. The main disadvantage of MATLAB is that it is proprietary, cost-intensive software. Compared to hardware-near languages like C or C++ it is slow and memory expensive. The use of a standard programming language like C++ overcomes the performance problems and licensing issues of MATLAB. For C++ there also exist various libraries for standard functions like reading and writing audio data or FFT computation. Nevertheless the program has to be written mainly from scratch. Audio Frameworks on the other hand offer a large base of algorithms that are commonly used for audio processing, like FFT or Histogram computation, which are designed to interact smoothly. They also offer data structures and visualisation mechanisms. The main advantage of frameworks is, that the code must not be written from scratch. Instead existing modules, structures, and interfaces can be reused, and only the new ideas must be implemented. The primary disadvantage of this approach is the steep learning curve that these frameworks require. To complicate things further the documentation is often imprecise, obsolete or simply not existent. There are no printed books or manuals for these frameworks as far as the author knows and much of the information is hidden in forums, mailing lists and newsgroups. MATLAB was excluded from the begin, since I wanted the final program to be usable for a broad audience, and MATLAB is widely known and used only among scientists and professionals. The first approach was thus to use a framework. Installation problems, the hardly foreseeable period of vocational adjustment and the wish to do and thus understand things from scratch finally led to the direct use of C++, supported by various libraries.

5 http://opihi.cs.uvic.ca/marsyas/

6 http://clam.iua.upf.edu/

7 http://www.mathworks.com

5.2 The Chord Analyser – Genchords

Our chord-detection program is named genchords and is implemented on a Linux platform in C++. It consists of approximately 3500 physical lines of code. The source code is structured into eleven classes, the most important being Chordtype, Chord, Chordsequence, Key, Labelfile, PCP, PCPTrack, Sounddata.

The open source Library libsndfile-dev[8] is used to read in the audio data. For the computation of the FFT source-code from Audacity[9] has been reused. The program lame[10] is used to convert mp3 to wave files. BeatRoot[11] is called for beat tracking and sox[12] is used for downsampling and filtering. Libsndfile, sox, lame and BeatRoot must be installed to get the full capabilities from genchords.

The following section describes the user interface and operation modes of genchords. It is followed by three sections that describe the different output formats of genchords, namely labelfiles, PCPfiles and scorefiles. Finally the performance of the algorithm is discussed and a summary of the finally used values for all parameters that have occurred in the previous chapters is given.

5.2.1 User Interface

Genchords is a command-line program. It has three operation modes: interactive, batch and file based mode. File and batch mode normally are the modes you want to use. They operate automatically and can be integrated in shell scripts. Interactive mode has been developed mainly for debugging reasons.

Batch and File Mode

The syntax for batch and file mode is

```
genchords [OPTIONS] sound [outdir]
```

Sound must either be a wav, aiff or mp3 file, or a directory. If it is a directory, all soundfiles in the directory are processed (batch mode). An output directory outdir can be specified to which the output (labelfiles, scorefiles, pcpfiles) is written. If outdir is omitted output is written to ./tmp.

8 Libsndfile - C library for reading and writing files containing sampled sound; http://www.mega-nerd.com/libsndfile/

9 Audacity – A free digital audio editor: http://audacity.sourceforge.net/

10 Lame – Lame ain't an MP3 encoder. Program to create compressed audio files; http://lame.sourceforge.net/

11 BeatRoot – An interactive Beat Tracking Program; http://www.ofai.at/~simon.dixon/beatroot/index.html

12 sox - universal sound sample translator; See http://sox.sourceforge.net/

Genchord accepts the following options:

-a PCPAlgoNr	Chose the algorithm that is used to convert the spectrum to a PCP. 1: one peak per mountain; 2: all peaks; 3: integrate. For a description of these algorithms see Section 4.3.1.
-b Beatfile | MillisecondsPer Beat	If the argument is an integer it is used as the span between two chords, else take the timespan of the chords from the specified beatfile. For each beat, the beatfile must contain one line with the start time of the beat in seconds. If this option is omitted a default span of 100 ms is used.
-o	Optimize the chord sequence to have less chord changes (see Section 4.6).
-p	Add the probability of each chord as comment in the labelfile
-n Numchords	The x best chords will be printed for each time period to the labelfile. Default=1 (only the best). Cannot be used with together with option -o.
-w Windowlegnth	Specifies the windowlength of the autocorrelation as number of frames. Must be a power of 2. The smaller the window size, the less low frequencies will be detected. If this options is omitted a defaults of the equivalent of 46.4 ms is used (see Section 4.2).
-k	Compute the key of the song, and only use chords that correspond to this key (see Section 4.4).
-h	print help
-v level	Verbosity level: 1=labelfile, 2=scorefile, 4=pcpfile; default=1 To output more than one file make level the sum of the files you want to output. E.g. in order to output labelfile and scorefile set level to 3 (= 1+2). All output files have the same base name as the corresponding soundfile, except for the extension, which is .txt for the labelfile, .sco for the scorefile and .pcp for the pcpfile.

The best results are usually achieved using the option " -a3 -b1 -k -o".

Examples of other useful option settings:

- -a3 -b100

  Perform short span analysis: all enhancement modules are switched off. Uses input spans of 100ms length. These options have been used to compute the short-span accuracies in Section 6.2.
- -p -n3

  Output the three best chords and their probabilities for each timespan.
- -b beatfile.txt -v7

  use the timestamps of beatfile.txt to split the input. Output score-, label-, and pcpfiles.

Interactive Mode

The interactive mode is entered by calling genchords without any options or arguments.

It provides the following interactive options:

```
************* Menu ****************
(l)oad          load a new song.
(i)nfo          print song information (length, channels, ...)
(k)ey           print key of the song
getsample       print a sample value from the audio data
pcp             print the Pitch Class Profile of a sequence of blocks
chord           print the chord of a sequence of blocks
p&c             print PCP and chord of a sequence of blocks
label(f)ile     write chords in a label file that can be imported to Audacity
(s)corefile     write chords to a scorefile that can be processed by csound
(m)enu          print the menu
(q)uit          quit the program
**********************************
```

In interactive mode computations are delayed until their results are really needed. That is, the PCPs are not computed until the command key, pcp, chord, p&c, labelfile or scorefile is entered. The chord sequence is not computed until the command chord, p&c, labelfile or scorefile is entered and so on. This speeds up debugging of the different stages of the algorithm. An exemplary interactive session is shown in Figure 5.2.

```
zenz@snoopy:~$ genchords
filename:./REM_-_Everybody_Hurts.wav
File loaded successfully
*************** Menu ****************
 (l)oad  load a new song.
[...]
 (q)uit  quit the program
************************************
i
/home/zenz/src/Genchords/testset/22050/low
pass/REM_-_In_Time_-_Everybody_Hurts.wav
********* File Information *************
length: 318.8s
frames: 7030104
samplerate: 22050
channels: 2
format: 10002
sections: 1
seekable 1
****************************************

getsample
...init sounddata...
start block: 10000
nr of blocks: 1
Sample[10000] = -0.140854

getsample
start block: 20000
nr of blocks: 3
Sample[20000] = 0.13327
Sample[20001] = 0.132309
Sample[20002] = 0.130508
```

```
pcp
...init pcptrack...
start block: 24
nr of blocks: 1
 PCP[24 = 12000ms ] =
296.584
        ^
        |              x
        |              x
        |              x
        |              x x     x
        |        x   x x x     x
0       |------------------------
        |C C#D D#E F F#G G#A A#B
C       C#      D       D#      E       F
F#      G       G#      A       Bb      B
10.18   0.00    0.00    0.00    88.91
0.00 42.64   296.58  101.76  0.39    5.19
98.36

chord
...init chordsequence...
start block: 24
nr of blocks: 1
Chord[24] = GMaj
k
********* Key of the song ***********
Ddur
*********************************


q
good bye ....
zenz@snoopy:~$
```

Figure 5.2 Interactive Genchords Session

5.2.2 Output formats

Genchords supports three different file formats to output the detected chord sequence: Labelfiles, Scorefiles and PCPFiles. The following subsections describe the syntax of each of these output formats and their applications.

Labelfile

One of the possible output formats for the detected chord sequence is a labelfile. Labelfiles have the following syntax:

```
labelfile = (labelline<LINEBREAK>)*
labelline = timestamp<BLANKS>label
timestamp = (0-9)+(,(0-9)*)?
label = <CHORDNAME>(/CHORDNAME)*(%Probability(/Probability)*)?
```

The timestamps are given in milliseconds and must be in ascending order. A label is said to be valid for a timespan x, where the duration of x is given indirectly by subtracting the current timestamp from the next timestamp. The label in line n is thus valid for (timestamp_{n+1}-timestamp_{n}) milliseconds. In order to assign a length to the last label a final labelline containing a dummy or blank label completes each labelfile.

In our case the label is the observed chord, but generally it can be an arbitrary string. A labelline produced by genchords could be:

20,432 AMaj

If genchords is started using the option -n numchords, every label consists of numchords chords separated by slashes. For example (n=3):

35,801 AMaj/AMin/F#Min

if the option -p is set, genchords additionally appends the probability of each chord to the label

35,801 AMaj/AMin/F#Min %% 91,544/90,563/87,664

Scorefile

The scorefiles comply to the syntax of csound scorefiles[13] and are used to resynthesize the extracted chord sequence. This allows the user to evaluate the correctness of the detected chords by listening to them instead of having to manually detect the chords himself and compare them to the automated result. An exemplary scorefile is shown in Figure 5.3. The first line is just a comment describing the contents of the columns. The second to fourth line generate a B Minor chord (b, d, f#). The second line generates the root pitch at the frequency 7.11 (b) with a length of 0.5 seconds using instrument one an orchestra file. The third line generates the third (8.02 = d) and the fourth line the quint (8.06 = f#) all with the same start and length. Line four to six generate a F# Minor chord (7.06 = f#, 7.09 = a, 8.01 = c#). In order to synthesize one chord, each of its pitches is triggered at a specified time (start) for a specified time (len). The root pitch is always set as the lowest pitch. the other pitches are ordered according to their number in the MIDI Notation.

```
;ins    start   len     amp     freq
1       0.0     0.50    3600    7.11    ;BMin
i1      0.0     0.50    3600    8.02    ;BMin
i1      0.0     0.50    3600    8.06    ;BMin
i1      0.50    0.50    3600    7.06    ;F#Min7
i1      0.50    0.50    3600    7.09    ;F#Min7
i1      0.50    0.50    3600    8.01    ;F#Min7
```

Figure 5.3: Example scorefile

The scorefile can be synthesized and added to the original soundfile with chordmix (Section 5.3.4). To directly synthesize the scorefile without mixing it to the original soundfile you can call csound with the following command:

```
csound -W -o $chordsoundfile -d -m 0 $ORC_FILE $scorefile
```

13 http://kevindumpscore.com/docs/csound-manual/index.html

PCPFile

```
------------------ 12.47 sec ------------------
136.067
        ^
        |        x
        |        x
        |        x
        |        x         x
        |        x x       x
        |        x x       x
        |  x     x x x     x
        |  x     x x x     x
        |x x   x x x x     x
        |x x x x x x x     x
0       |------------------------
        |C C#D D#E F F#G G#A A#B

C       C#      D       D#      E       F       F#      G       G#      A       Bb      B
29.02   56.42   24.01   29.28   136.07  89.95   54.49   0.00    3.82    99.73   1.11    0.00
------------------ 12.83 sec ------------------
```

Figure 5.4: PCPFile Entry

The PCPFile contains all the computed pitch class profiles in a textual and graphical representation. Each entry starts with a line containing the beginning time of the PCP. A bar diagram of the not-yet normalized PCP is followed by the PCP vector. An exemplary PCPFile entry is shown in Figure 5.4. It shows the pitch class profile computed at the timespan beginning at 12.47s and ending at 12.83s. The profile has local maxima at the pitches C#, A and E, which would indicate an A-Major chord(a-c#-e). However, the final detection output might also be for example C#-Minor (c#-e-g#) or F-Major (f-a-c) depending on the key of the song and the surrounding PCPs.

5.2.3 Performance

On a 1.5 GHz pentium computer chord detection with the ideal parameters (-o -k -a3 -b1) takes under 30% of the length of the music. A three minute song thus takes less than one minute to process. From those 30% about 30% are needed by BeatRoot for beat detection.

Table 5.1 shows execution times in seconds of genchords on a 1.5 GHz pentium for a one minute song measured with the linux command time. You can see that the -o option which triggers the smoothing optimization algorithm dramatically increases execution time. The smoothing algorithm is not optimized and effective time enhancements are probably possible at this module.

		-b100	***-b200***	***-b1***	***-b filename***
-oka3	***real***	31.628	17.943	15.720	11.415
	user	24.456	15.048	13.993	9.656
-ka3	***real***	5.490	5.493	10.523	5.691
	user	5.078	5.278	9.819	5.500
-a3	***real***	2.817	2.797	7.623	2.918
	user	2.587	2.622	7.167	2.767

Table 5.1: Genchords Performance (in seconds for a 1 minute song)

The algorithm is fast enough to operate in real time, which means that it can process the data while the music is playing. Thus the algorithm could be integrated in a music player and compute and visualize the chords that fit the music that is currently played. However the current implementation doesn't support real-time analysis, because currently each module processes the whole song, and passes the entire results to the next module. Nevertheless, the implementation could be easily modified to operate in real-time. Also BeatRoot could be redesigned to work in real-time (see [20]). Only key detection really has to be performed in advance as it works on the start and end of the song and its result is needed for all further computation. With the current settings and if the song length is greater than 30 seconds, the execution time of key detection is independent of the song length, as it always uses a 30 seconds excerpt to compute the key. On a 1.5 GHz pentium key detection takes about 2.5 seconds. BeatRoot takes less than 10 % (4.3 seconds for a 1 minute song). Thus if genchords would be redesigned to work in real time it would need a start up time of less than 2.5 + 0.1*length(song) seconds (8.5 seconds for a 1 minute song).

5.2.4 Parameter Summary

This section summarizes the parameters that have been introduced in Chapter 4 and the values that we have assigned used for evaluation.

For enhanced autocorrelation described in Section 4.2 the following parameters are used:
windowSize = 46.4 ms (1024 samples for samplerate 22050, 2048 samples for samplerate 44100)
blockSize (if the Beat Detection Module is turned off): 100 ms
hop size = windowSize/2

For the Generation of the PCP (Section 4.3.1) the following parameters are used:
number of neighbours for the advanced peak algorithm: 3

The applied reference PCPs (Section 4.3.3) for the chord types major, minor and seventh chords are:
pcp_maj = {1.0, 0.00, 0.05, 0.05, 0.24, 0.15, 0.01, 0.39, 0.02, 0.16, 0.00, 0.02};
pcp_min = {1.0, 0.00, 0.05, 0.30, 0.03, 0.14, 0.04, 0.26, 0.25, 0.00, 0.00, 0.02};
pcp_maj7 = {1.0, 0.00, 0.05, 0.05, 0.24, 0.15, 0.01, 0.39, 0.02, 0.16, 0.46, 0.02};
pcp_min7 = {1.0, 0.00, 0.05, 0.30, 0.03, 0.14, 0.04, 0.26, 0.25, 0.00, 0.46, 0.02};

Our reference PCPs (Section 4.4) for the major and minor key are:
pcp_major_key = {1.0, 0.0, 0.6, 0.0, 0.7, 0.6, 0.0, 0.9, 0.0, 0.6, 0.0, 0.6};
pcp_minor_key = {1.0, 0.0, 0.6, 0.8, 0.0, 0.5, 0.0, 0.9, 0.6, 0.0, 0.7, 0.3};

For the optimization algorithm (Section 4.6) we use the following parameters:
number of chords per timespan: 5
neighbours: 2
ChordChangePenalty: 0.9

Ideal invocation options for genchords:
-a3 -b1 -k -o

5.3 Tools

In addition to the main program described in Section 5.2, a variety of support-tools have been implemented. These tools are not necessary to perform chord analysis but help the implementer to find good parameters for genchords and the end-user to interpret and evaluate the outputs of genchords.

Learnchords takes sound files and labelfiles as input and computes the average PCPs of all occurring Chordtypes. It has been used to find the reference PCPs for this work. The obtained reference PCPs are listed in Section 5.2.4. Labeldiff compares two labelfiles and prints out a detailed comparison table as well as confusion matrix and statistical data. By providing it with generated and hand labelled files it has been used to evaluate the accuracy of our chord detection algorithms. Transpose provides methods for transposing and converting labelfiles to different formats, including enharmonic equivalent representations. Chordmix finally is used to process the scorefiles generated by genchords. With the help of different open source products it generates a sound file that contains both the synthesized generated chords and the original audio data. It's nice tool to get a quick first estimation of the correctness of genchords output.

5.3.1 Learnchords

Learnchords is a tool to compute the reference PCPs for different Chordtypes from audio data of which we know the Chordsequence. It reuses the classes implemented for genchords and has a very similar syntax:

```
learnchords –a algonr –b beat(file|dir) Label(file|dir) Sound(file|dir)
```

The options a and b follow the behaviour of genchords and are described in Section 5.2.1. The labelfile must contain the chords for the soundfile. If directories are given, the files in the directories are processed one after the other. Each soundfile in sounddir must have a corresponding labelfile in labeldir, which has the same name as the soundfile but the extension ".txt".

The soundfile is split into chunks of data according to the beat option. For each chunk the PCP is generated and the chordtype is looked up in the labelfile. The PCP is then shifted to pitch C. All PCPs of the same chordtype are accumulated in a reference PCP. After processing all the inputfiles the reference PCPs are normalized and printed to stdout.

5.3.2 Labeldiff

`labeldiff [-m (0-2)] labelfile1 labelfile2`

In the easiest case "Labeldiff" compares the content of two labelfiles. If labelfile1 is a directory labelfile2 must also be a directory. In this case labeldiff compares corresponding files in the two directories. Two labelfiles are not compared line-by-line but according to their timestamps. Two Labels are said to be equal for a certain period of time if the two intervals overlap and one of the following statements is applies:

- The labels match character by character
- at least one of the labels is an asterisk ('*'). Asterisks match any label and are for example used in the truth labelfiles to label periods of silence.
- at least one of the labels has the form <sublabel>("/"<sublabel>)* and at least one of the sublabels matches the label or one of the sublabels of the other file.

Labeldiff can compare any labelfiles that comply to the syntax described in Section 5.2.2, but it offers some special features for chord labelfiles:

- Enharmonic equivalent Chords match: e.g.: G#Maj and AbMaj
- Chord Style Comparison can be switched on or off. When switched off, only the pitch and the primary type (major and minor) are compared. e.g. GMaj7 and GMaj match, but GMaj and GMin do not. By default chord style comparison is off.

Labeldiff outputs a detailed comparison table, a confusion matrix and statistical information which we will now explain in detail.

Comparison Table

The comparison table gives detailed information on the Labels of both files at any time. Figure 5.5 shows two input labelfiles and the resulting comparison table. The first two columns *pos1* and *pos2* contain the current line number of the two labelfiles. The third column *time* specifies the start time and is followed by the labels of the two labelfiles at this time. The next column *len* specifies the length of time in seconds for which these labels are valid. The *diff* column finally is 0 if the two labels match and 1 otherwise. The last two columns contain debugging information about the start time of the last time the labels fell apart and the duration of difference since the beginning not including the current state. Lines with differing labels are highlighted.

Confusion matrix

The confusion matrix displays information about the kind of confusions. The Labels of the first inputfile are listed horizontally, the labels of the second inputfile vertically. The cell at row x and column y specifies how often or how long labelfile1 contained label x while labelfile two contained label y. The cells in the diagonal axis hold the information about label matches. The option -m of the command labeldiff specifies how the confusion matrix shall be printed: It is set to zero to suppress output of the confusion matrix. If set to one, duration of the confusion in seconds are printed. When set to two, the number of confusion is outputted, which is the default setting.
Figure 5.6 displays the confusion matrix for the input files of Figure 5.5 The left matrix shows the numbers of confusions and has been generated using the option -m 2, while the right matrix lists the durations.

Statistical Information

The statistical information contains absolute hits in seconds and relative hits as percentage value.

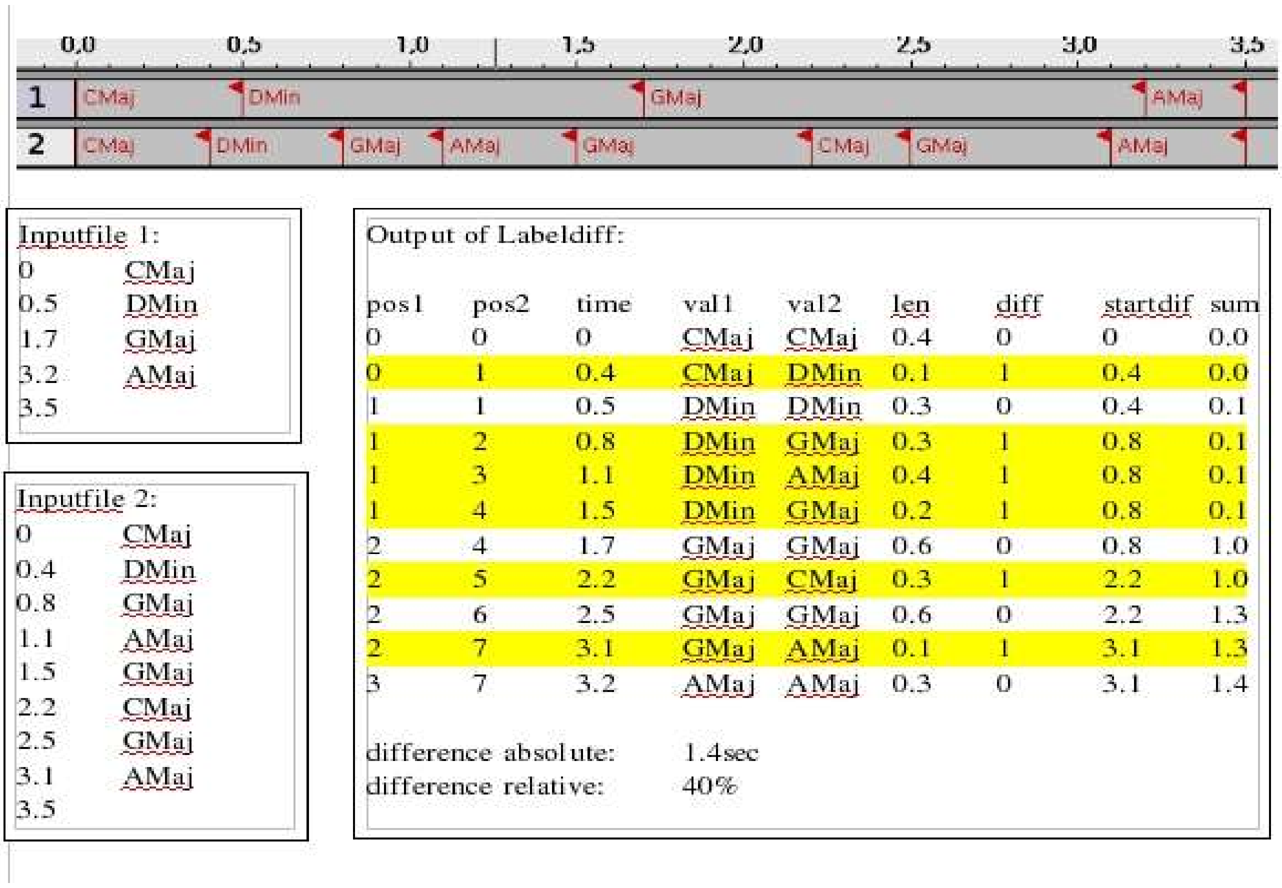
$$relativehits(f) = \frac{absolutehits(f)}{length(f)} \tag{5.1}$$

If labelfile1 is a directory, statistic information for each file is printed as well as a summary hit percentage which is calculated using:

$$\frac{\sum_{f \in files} relativehits(f)}{|files|} \tag{5.2}$$

and a summary hit percentage that rates the hits according to the durations of the labefiles:

$$\frac{\sum_{f \in files} absolutehits(f)}{\sum_{f \in files} length(f)} \tag{5.3}$$



```
Inputfile 1:
0       CMaj
0.5     DMin
1.7     GMaj
3.2     AMaj
3.5
```

```
Inputfile 2:
0       CMaj
0.4     DMin
0.8     GMaj
1.1     AMaj
1.5     GMaj
2.2     CMaj
2.5     GMaj
3.1     AMaj
3.5
```

```
Output of Labeldiff:

pos1   pos2   time   val1   val2   len   diff   startdif   sum
0      0      0      CMaj   CMaj   0.4   0      0          0.0
0      1      0.4    CMaj   DMin   0.1   1      0.4        0.0
1      1      0.5    DMin   DMin   0.3   0      0.4        0.1
1      2      0.8    DMin   GMaj   0.3   1      0.8        0.1
1      3      1.1    DMin   AMaj   0.4   1      0.8        0.1
1      4      1.5    DMin   GMaj   0.2   1      0.8        0.1
2      4      1.7    GMaj   GMaj   0.6   0      0.8        1.0
2      5      2.2    GMaj   CMaj   0.3   1      2.2        1.0
2      6      2.5    GMaj   GMaj   0.6   0      2.2        1.3
2      7      3.1    GMaj   AMaj   0.1   1      3.1        1.3
3      7      3.2    AMaj   AMaj   0.3   0      3.1        1.4

difference absolute:   1.4sec
difference relative:   40%
```

Figure 5.5: Labeldiff

	AMaj	Cmaj	DMin	GMaj
AMaj	**1**	0	0	0
CMaj	0	**1**	1	0
DMin	1	0	**1**	2
GMaj	1	1	0	**2**

	AMaj	CMaj	DMin	GMaj
AMaj	**0.30**	0.00	0.00	0.00
CMaj	0.00	**0.40**	0.10	0.00
DMin	0.40	0.00	**0.30**	0.50
GMaj	0.10	0.30	0.00	**1.10**

Figure 5.6: Labeldiff: Confusion Matrix

5.3.3 Transpose

“Transpose” is a tool for transposing and reformatting chord-labelfiles generated by *genchords*.

```
transpose.py [-b  [-l | -s ] ] steps filename
```

Filename must be the name or path of a labelfile, steps indicates by how many semitones the supplied labelfile shall be transposed. Negative steps are not allowed. To transpose the chords two semitones downwards, set steps to 12-2 = 10. A step of zero doesn't change the chords and can be used if you just want to apply a format change using the format options “-l” or “-s” or apply enharmonic change using -b and leave the pitch as it is.

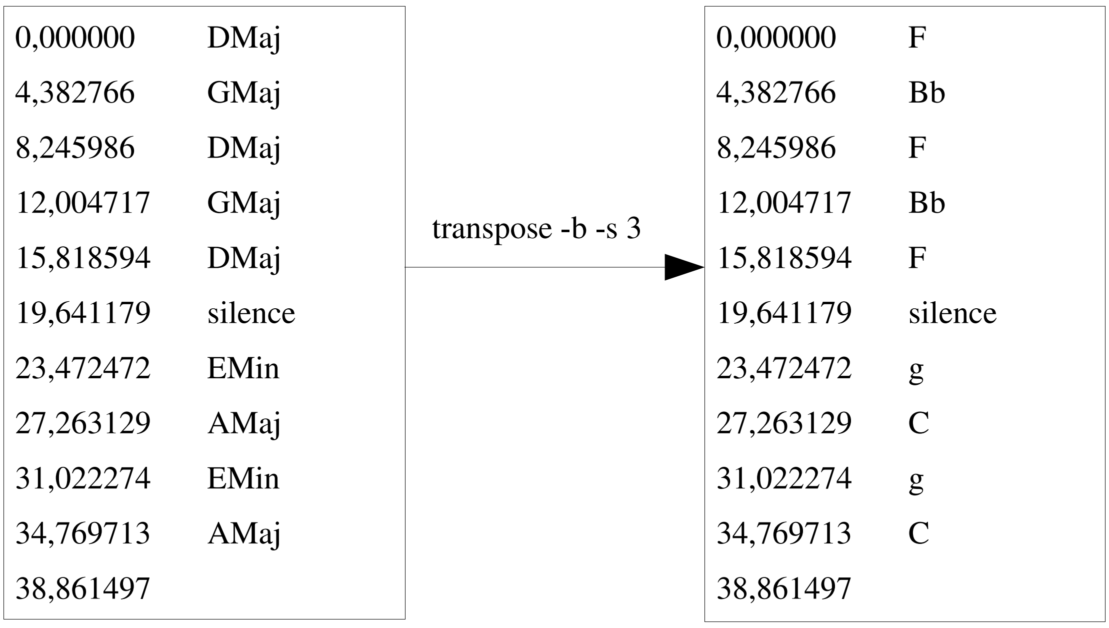
options:

-b use b instead of sharps (sharps are default)

-l use long names: longname = (pitchname)(Maj|Min).*

-s use short names = pitchname.*, where pitchname is uppercase if the chord is a Major chord, and lowercase for Minor chords.

The output is written to *stdout* and thus can be redirected to a file using the “>” operator. Lines that cannot be converted because they don't comply with the labelfile format are copied as is and a summary containing the line numbers that couldn't be converted is written to stderr. Figure 5.7 shows exemplary input and output data of transpose envoced with the options "-b -s 3".



```
0,000000    DMaj
4,382766    GMaj
8,245986    DMaj
12,004717   GMaj
15,818594   DMaj
19,641179   silence
23,472472   EMin
27,263129   AMaj
31,022274   EMin
34,769713   AMaj
38,861497
```

```
transpose -b -s 3
```

```
0,000000    F
4,382766    Bb
8,245986    F
12,004717   Bb
15,818594   F
19,641179   silence
23,472472   g
27,263129   C
31,022274   g
34,769713   C
38,861497
```

Figure 5.7: Transpose

5.3.4 Chordmix

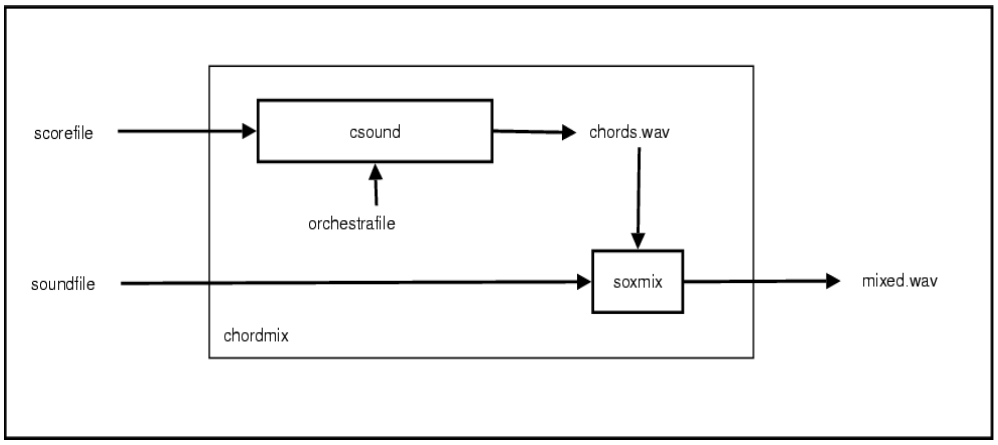


Figure 5.8: Chordmix

"Chordmix" is a shell script that synthesizes the chords generated by *genchords.* Furthermore it creates a stereo sound file, which contains the original song on the right channel and the synthesized chords on the left channel. The volume of the two audio streams can thus be controlled separately. Chordmix is executed using the following syntax:

```
chordmix scorefile soundfile.
```

The scorefile is passed to *csound* together with an orchestra file. The sound file generated by *csound*[14] is our chordsoundfile. The chordsoundfile and the soundfile are both converted to mono using *sox*[15]. The length of both files is calculated using *shntool*[16]. If necessary the longer file is trimmed so that both files have exactly the same length. Finally the two mono files are mixed to one stereo file using *sox*. The resulting wavfiles are saved in *`</path/to/soundfile>/chords/`* and are named *`<soundfile>__chords.wav`* and *`<soundfile>__mixed.wav.`*

Listening to the synthesized chord sequence offers the user the possibility to assess its quality in an informal, direct way. This is of course not a formal method of evaluation, as it is subjective, imprecise and impractical for large amounts of data. Nevertheless when dealing with the subjective medium of music, the listening test may be the most convincing way to demonstrate the capabilities of the system. Another advantage is, that no additional labelfile that represents the truth is needed, and the evaluation can be performed without tedious preparations.

14 csound - music and sound synthesis software; See http://www.lakewoodsound.com/csound/

15 sox - universal sound sample translator; See http://sox.sourceforge.net/

16 shntool - multi-purpose WAVE data processing and reporting utility; See http://www.etree.org/shnutils/shntool/

5.4 Conclusion

In this chapter we have discussed the developed automatic chord detector genchords and its tools in detail. Finally, we want to give you a short tour of the whole tool chain.

Tour of the implementation

Suppose you have a sound file called *Track1.wav* that you have ripped from an audio CD. Before analysing it you should downsample and lowpass-filter it, to enhance and speed up analysis:

```
sox Track1.wav -r 22050 Track1a.wav resample -qs filter 0-1000
```

This generates sound file "Track1a.wav" that we can now pass to our chord detector:

```
genchords -a3 -b1 -k -o -v7 Track1a.wav ./chords
```

The files Track1a.txt, Track1a.pcp and Track1a.sco are generated in the directory ./chords.

First, lets mix the detected chords to the original song and listen to the result:

```
chordmix.sh ./chords/Track1a.sco Track1.wav
play ./chords/Track1__mixed.wav &
```

Sounds good? Well, lets have a detailed look at the detected chords. You can either just read the labelfile using any text reader, or better, visualise it using Audacity:

```
audacity Track1.wav &
In Audacity: Menu Project -> import textlabel -> ./chords/Track1a.txt
```

The chord labels are aligned with the audio signal, and you can select a section and look at the current chord labels, that scroll with by while the song is played.

In order to precisely evaluate the quality of the labels, you can detect the chords manually or look them up in a scorefile of the song, if you have one available. You need to assign the chords to precise timespans in the song and create a labelfile (say Track1_truth.txt). If you don't like typing the full names and want to speed up the hand-labelling process, name the chords with their short names (e.g. 'g' instead of 'GMin') and transpose the whole labelfile to long chord names when you have finished using transpose:

```
transpose -l 0 Track1_truth_short.txt > Track1_truth.txt
```

Now you can compare your labelfile with the automatically generated one using labeldiff:

```
labeldiff Track1_truth.txt ./chords/Track1a.txt
```

98% accuracy? - Perfect!

6 Evaluation

The previous two chapters described the design and implementation of an automatic chord detector. This chapter summarizes the tests that have been performed in order to evaluate this design and its implementation. It starts with an overview over the test set (Section 6.1). Section 6.2 gives detailed information on the achieved accuracies. As stated in Chapter 4, our chord detector has a modular design, where each module can be switched on and off independently. In order to prove the effectiveness of each module the tests have been performed separately for each module. Afterwards the limits of frequency-based chord detection are highlighted and our results are compared to those achieved by the algorithms described in Chapter 3. Finally Section 6.3 gives an insight on the properties of the chord-confusions that have occurred.

6.1 Test Set

In order to evaluate chord detection algorithms a representative test set is most essential. The compilation of a test set and especially the generation of truth files is a time-consuming task, thus it would be desirable that a standard test set for chord detection would be defined. Evaluation results of different chord detectors could then be compared directly leading to more transparency.
As such a standardized test set doesn't exist yet, and the test set used by other researches were either not available at that time or not general enough (see Sections 3.5 and 3.6), we have defined our own test set. The songs have been carefully selected to represent a variety of different styles. Attention has been paid to chose widely known artists so that others who wish to compare their results with those stated here have easier access to the test data. All songs have been manually labelled using Audacity. Table 6.1 lists the artist and title of the songs that have been chosen for evaluation, their key and the set of chords that occur in the first 60 seconds. All tests have been performed on the first minute of each song. Though the songs themselves are under copyright restrictions and thus cannot be published, the publishing of the truth files is planned in order to facilitate the work for future research.

6.2 Accuracy

This section gives detailed information on the detection accuracy of our algorithm. All results are compared to those of what we further call a "simple short-span algorithm". This is an algorithm, that splits the input into 100ms spans and does no key detection or chord optimization.

For evaluating the output we compared the generated labelfile to a hand-labelled truth file using labeldiff (Section 5.3.2). We have defined accuracy as

$$accuracy = \frac{hits(ms)}{total\ length(ms)} \tag{6.1}$$

First we will describe the accuracy of key detection. Afterwards the influence and effectiveness of each module is evaluated: Section 6.2.2 states the influence of key detection on the accuracy rate, Section 6.2.3 describes the influence of beat tracking and Section 6.2.4 the influence of chord sequence optimization. Section 6.2.6 then lists the results we achieved with all modules turned on. A description of the limits of frequency-based chord detection follows. Section 6.2.7 finally compares our results to those achieved by other chord detection algorithms.

Nr.	***Artist***	***Title***	***Key***	***Chords (first 60s)***
1	ABBA	Dancing Queen	A major	A, D, E, c#, f#, B
2	Bob Dylan	Blowin In The Wind	D major	D, G, A
3	Cat Stevens	Wild World	C major	a, D, G, C, F, E
4	Elton John	Sorry Seems To Be The Hardest Word	G minor	g, c, D, F, A#, B
5	Elvis Presley	Devil In Disguise	F major	Bb, C, F, d
6	Eva Cassidy	Fields Of Gold	f# minor	f#, D, E, A, b,
7	Green Day	Basket Case	Eb major	Eb, Bb, c, g, Ab, Db,
8	Jack Johnson	Sitting Waiting Wishing	A minor	a, G, F, C, E
9	Mando Diao	God Knows	D major ?	B, D, E, G, f#, b
10	Muse	Thoughts Of Dying Atheist	G minor	g, Eb, D, g, c, F, Bb
11	Norah Jones	Come Away With Me	C major	C, a, F, e
12	Radiohead	Karma Police	A minor	a, e, G, F, D, b, C
13	Reinhard Fendrich	I Am From Austria	G major	G, C, D, e, b, a
14	REM	Everybody Hurts	D major (+B Minor)	D, g, e, A,
15	Red Hot Chili Peppers	Californication	A minor (+ F# minor)	a, F, C, G, d
16	The Beatles	Help	A major	b, G, E, A, c#, f#, D,
17	The Beatles	Yesterday	F major	F, e, A, d, Bb, C, G,
18	Tina Turner	What's Love Got To Do With It	B major (+ C# major)	g#, d#, E, F#, B,
19	Travis	Sing	F# minor	f#, b, A, E,

Table 6.1: Test Set

6.2.1 Key Detection Accuracy

Table 6.2 lists the true and the detected keys for all test songs. Matches are marked with a ✓, confusion between relative keys with an 'R' and other confusions with X. 13 of 19 keys have been detected correctly. From the six confusions five have been between major and relative minor key. As explained in Section 2.2 major and relative minor key contain the same pitches and our algorithm preselects the same chords for them. Thus these confusions do not affect our chord detection algorithm. The only serious mistake has been made on song 2 (Bob Dylan – Blowin in the wind) where the key was mistakenly identified as A Major instead of D Major. Again, this mistake is one between closely related keys (see 2.2), that share 6 of 7 pitches and 8 of 10 associated chords (see Section 4.4).

Nr.	***Artist***	***Title***	***Key***	***Key detected***	***Accuracy***
1	ABBA	Dancing Queen	A major	A major	✓
2	Bob Dylan	Blowin In The Wind	D major	A major	X
3	Cat Stevens	Wild World	C major	C major	✓
4	Elton John	Sorry Seems To Be ...	G minor	G minor	✓
5	Elvis Presley	Devil In Disguise	F major	F major	✓
6	Eva Cassidy	Fields Of Gold	f# minor	A major	R
7	Green Day	Basket Case	Eb major	Eb major	✓
8	Jack Johnson	Sitting Waiting Wishing	A minor	A minor	✓
9	Mando Diao	God Knows	D major ?	B minor	R
10	Muse	Thoughts Of Dying Atheist	G minor	Bb major	R
11	Norah Jones	Come Away With Me	C major	C major	✓
12	Radiohead	Karma Police	A minor	A minor	✓
13	R. Fendrich	I Am From Austria	G major	G major	✓
14	REM	Everybody Hurts	D major (+B Minor)	D major	✓
15	RHCP	Californication	A minor (+ F# minor)	C major	R
16	The Beatles	Help	A major	A major	✓
17	The Beatles	Yesterday	F major	F major	✓
18	Tina Turner	What's Love Got To Do ...	B major (+ C# major)	G# minor	R
19	Travis	Sing	F# minor	F# minor	✓

Table 6.2: Key Detection Results

6.2.2 Influence of Key Detection

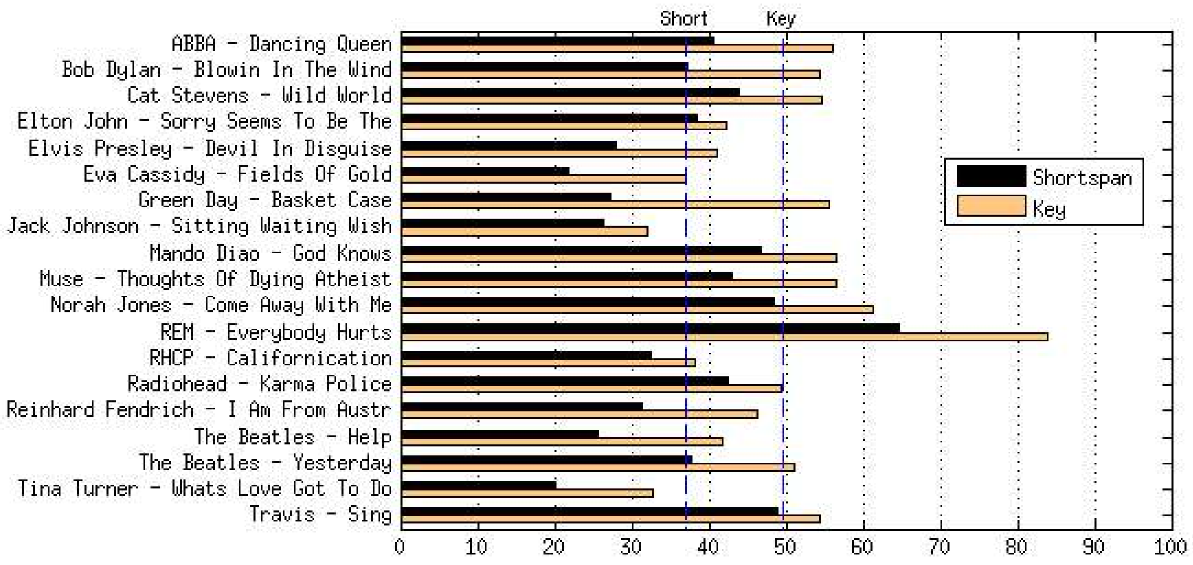


Figure 6.1: Test Results for Key Detection

Figure 6.1 compares the results of the short-span algorithm to those of our algorithm where only key detection is turned on. Key detection enhanced the detection quality for all songs of the test set. The wrong key detection for song 2 had no consequences, as only primary chords are used in this song. The mistakenly identified key A-major has a quint-relationship to the real key D-major, and all primary chords of D-major are primary or secondary chords of A-major and thus not filtered out. This indicates a high robustness of our algorithm against related-key confusions.

When the detected key of the song was taken into account, accuracy increased by an average of 13%. The greatest enhancement has been achieved for song 7 (Green Day – Basket Case) of which the accuracy rate doubled from 27% to 55%. With the short-span method many confusions are made between major and minor chords on the same root note (specially on the tonic (eb-g-bb versus eb-gb-bb) and the dominant (ab-c-eb versus ab-cb-eb). This is, because the third, that differentiates minor from major chords is often missing. Having identified the key being major, a minor tonic or dominant is excluded from the possible chords resulting in the described accuracy enhancement.

6.2.3 Influence of Beat Tracking

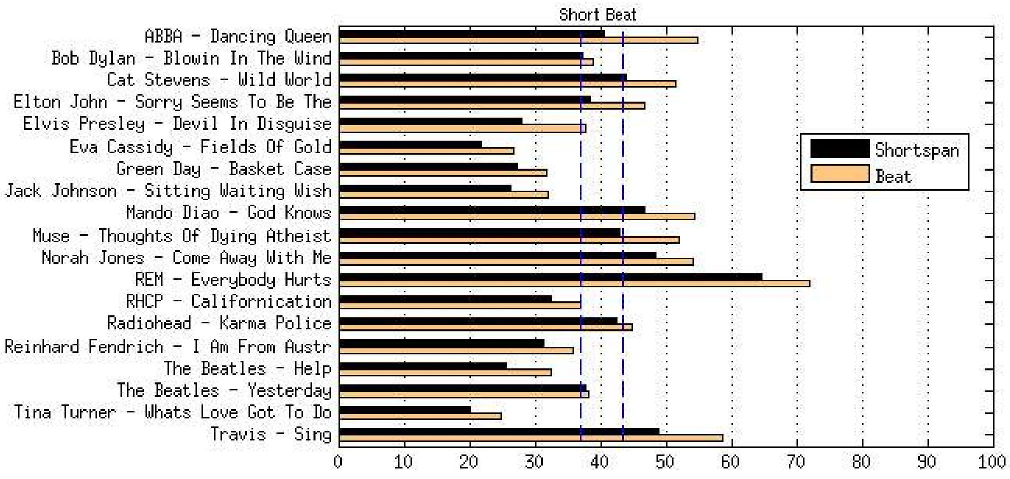


Figure 6.2: Test Results for Beat Detection

Figure 6.2 compares the results of the short-span algorithm to those of our algorithm where only beat detection is turned on. Beat detection enhanced the detection quality for all songs of the test set. On an average accuracy increased by 6% when the detected beats of the song were taken into account. The greatest enhancement was achieved for song 2 (Abba – Dancing Queen) (14%), the smallest enhancement for song 17 (The Beatles – Yesterday) (0.3%). Beat spans had an average length between 300ms for song 18 (Tina Turner – What's Love Got To Do) and 760 ms (Norah Jones – Come Away With Me). Compared to the spans of 100ms, that the short-span algorithm uses, these larger spans are more robust against short non-chord tones. Figure 6.3 shows an excerpt of song 5, the manually recognized chords and those recognized by the short-span algorithm respectively those recognized by our algorithm using beat tracking. The flags in the last two lines demarcate the analysed time spans of the short-span respectively the beat tracking enhanced algorithm. Chord names are shortened for better readability.

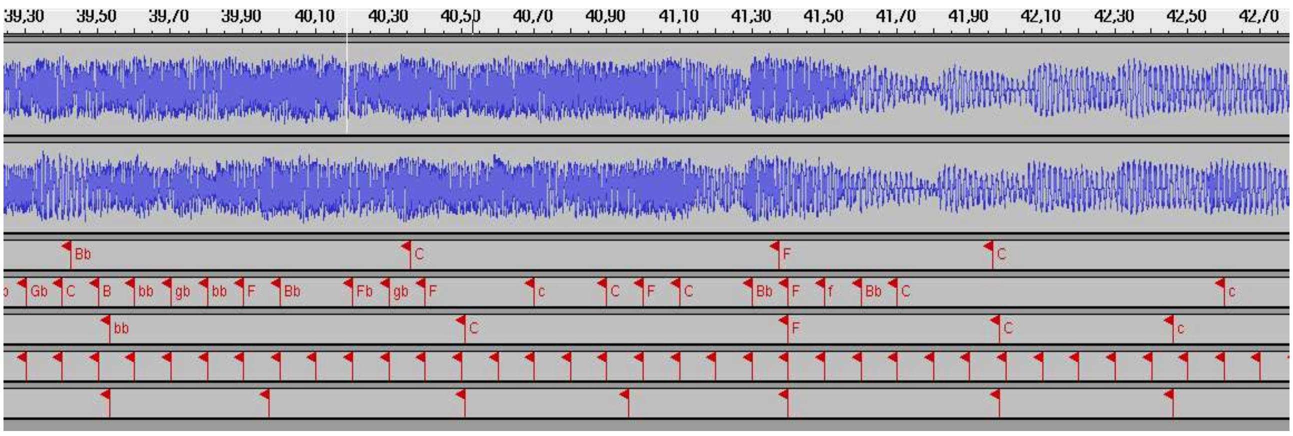


Figure 6.3: Labelfile Comparison: short span versus beat tracking - (Elvis Presley, Devil in Disguise)

6.2.4 Influence of Chord Sequence Optimization

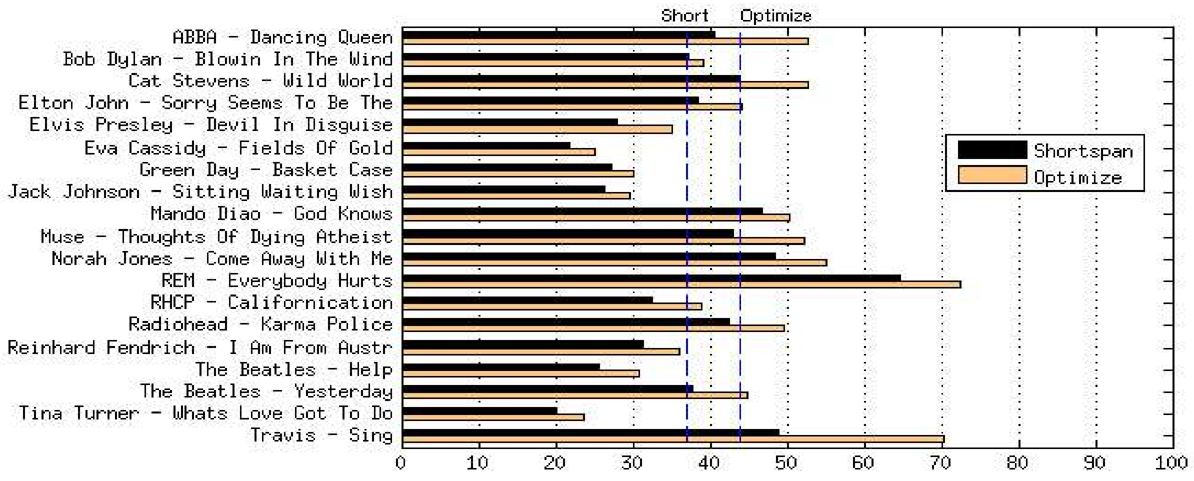


Figure 6.4: Test Results for Optimization

Figure 6.4 compares the results of the short-span algorithm to those of our algorithm where only chordsequence optimization (smoothing) is turned on. Chordsequence optimization enhanced the detection quality for all songs of the test set. On an average accuracy increased by 7% when the chord sequence was passed to our optimization algorithm. The greatest enhancement has been achieved for song 19 (Travis – Sing) for which accuracy rate was increased by 21% from 49% to 70%. The smallest enhancement was 3% and was encountered for song 2 (Bob Dylan – Blowin in the wind).

These tests were made with a neighbourhood n of 2 considering the best 5 chords per timespan. Further increase of the neighbourhood or the number of chords resulted in an accuracy enhancement of only some tenths of a percent while boosting execution time exponentially.

On the average, smoothing changed nearly 30% of the chords. For every third to fourth span, the algorithm did not chose the chord that scored best. Of course, the most frequent change was the one to the second best chord (58% of all changes) but also the 5th best chord has been taken at an average of 24 times per song (13% of all changes). Half of the changes were changes from the major to the minor chord with the same root, or from the minor to the major chord with the same root.

6.2.5 Accuracy of the entire Algorithm

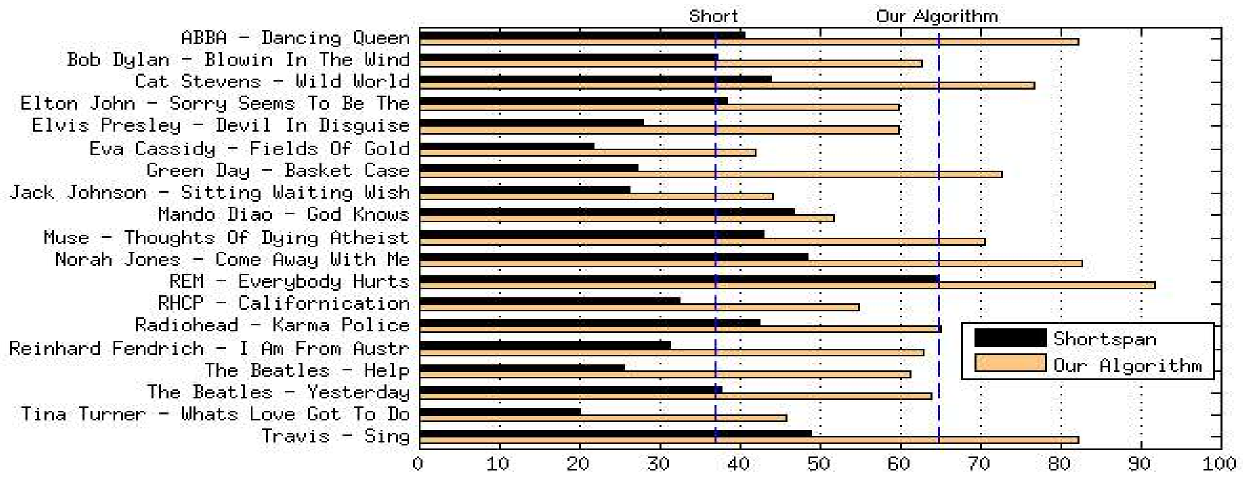


Figure 6.5: Test Results for all modules combined

The simple short-span algorithm, that detects chords every 100ms, and performs no key detection, beat tracking or chord sequence optimization, had an average accuracy rate of 37%. Our system achieved an average accuracy rate of 65% which is an improvement of 28% compared to the short-span algorithm. Figure 6.5 shows the results for each song of the test set. A significant increase of accuracy can be noted for all songs except song 9 (Mando Diao – God Knows), which uses the chord sequence that sticks least to its key. The song for which our algorithm performed best were song 14 (REM – Everybody Hurts) with 95% accuracy. It has a very dominant accompaniment in the form of arpeggio sounds. Percussion doesn't seem to have a great influence, as both the best and the worst recognition results (song 6 Eva-Cassidy – Fields of Gold 42%) have occurred for songs without (song 6) or with very decent percussion (song 14).

The main factors that lead to wrong chord identification are:

- non-chord tones: neighbour tones, anticipation, passing tones, suspension and escape tones are just some examples for notes that can occur mainly in the melody, that are not members of the chord and produce intended dissonances. The stronger these tones are, the more they confuse the algorithm. Thus songs where the melody is much more dominant than the accompaniment are more difficult to analyse.
- misleading frequency spectrum due to the harmonic series, mistuning (especially with human voice) or percussive sounds.

- use of other chord types: Our system currently differentiates only major, minor and seventh chords. Other chord types, like augmented and diminished chords, suspended or extended chords are not recognized.

Table 6.3 summarizes the accuracy results for the whole test set and all modules.

Song			***Accuracy***				
Nr.	***Artist***	***Title***	***Shortsp.***	***Key***	***Beat***	***Optim.***	***Algo***
1	ABBA	Dancing Queen	41	56	55	53	82
2	Bob Dylan	Blowin In The Wind	37	54	39	40	63
3	Cat Stevens	Wild World	44	55	52	53	77
4	Elton John	Sorry Seems To Be ...	38	42	47	44	60
5	Elvis Presley	Devil In Disguise	28	41	38	35	60
6	Eva Cassidy	Fields Of Gold	22	37	27	25	42
7	Green Day	Basket Case	27	55	32	30	73
8	Jack Johnson	Sitting Waiting Wishing	26	32	32	30	44
9	Mando Diao	God Knows	47	57	54	50	52
10	Muse	Thoughts Of Dying Atheist	43	57	52	52	71
11	Norah Jones	Come Away With Me	49	61	54	55	83
12	Radiohead	Karma Police	43	49	45	50	65
13	R. Fendrich	I Am From Austria	31	46	36	36	63
14	REM	Everybody Hurts	65	84	72	73	92
15	RHCP	Californication	32	38	37	39	55
16	The Beatles	Help	26	42	33	31	61
17	The Beatles	Yesterday	38	51	38	45	64
18	Tina Turner	What's Love Got To Do ...	20	33	25	24	46
19	Travis	Sing	49	54	59	70	82
Total			**37**	**50**	**43**	**44**	**65**

Table 6.3: Accuracy Overview

6.2.6 Accuracy Limits

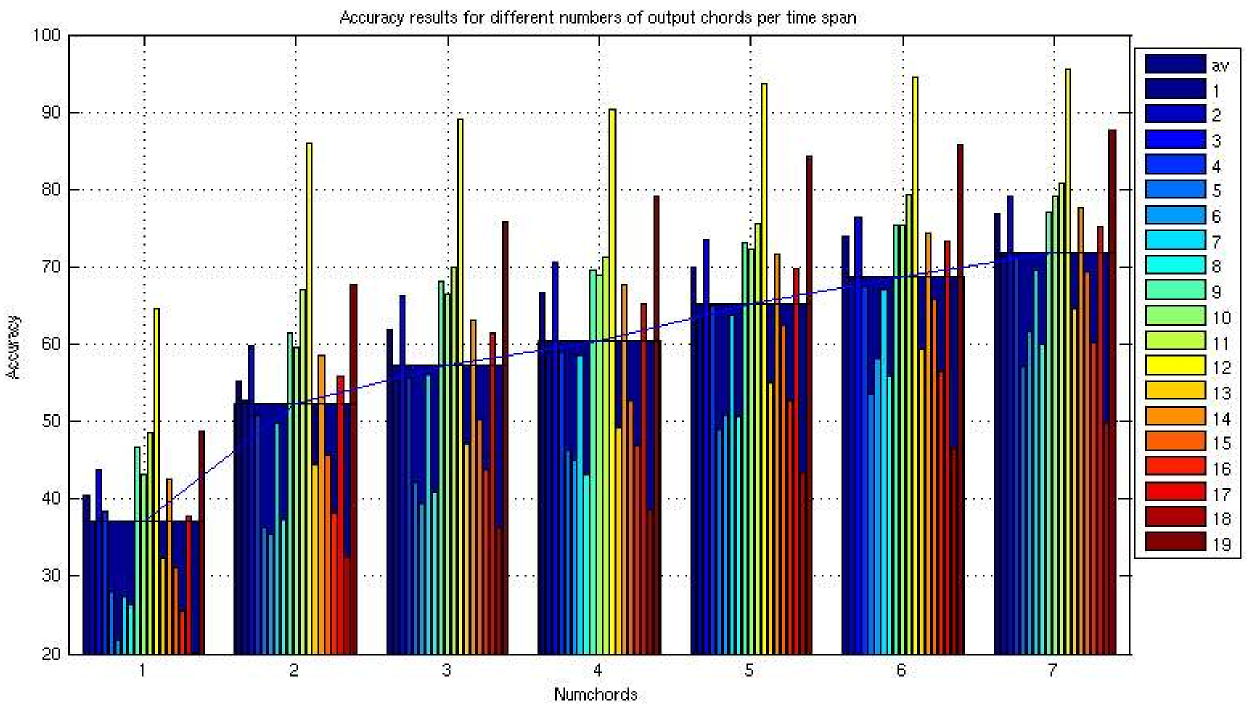


Figure 6.6: Accuracy results for different number of output chords per time span

As described in Section 5.2 genchords can be configured to output not only the best but the n-best chords. For each timespan the detection is then defined to be correct if one of these n-chords matches the real chord. Figure 6.6 shows the achieved accuracy results for number of output chords (numchords) ranging from 1 (standard) to 7. All enhancements modules have been switched off for this computation. The average result is shown as a broad bar over witch the individual results for each chord are drawn. You can see that the best song (song 12) still did only reach 98% even when 7 chords were considered. Part of the remaining error rate results from inaccuracies in the hand-labelled truth file, where chord changes are always a bit out of time (on average about 50ms early or late). What is very interesting to observe, is that even when the 7 best chords are considered, the average accuracy rate doesn't rise over 80%. This means that, assuming that the calculation of the PCP has not simply been erroneous, the PCP and the detected frequencies are misleading in about 20% of the time. As a conclusion, a chord detector that wants to cross this 80% mark needs to make hypothesis on the quality of the PCPs and needs to deduce the chords of low-quality-PCP chord spans from the surrounding higher quality spans. In a nutshell: sometimes it is necessary to ignore the frequency-based feature.

6.2.7 Comparison with other Algorithms

A direct comparison of our results with that of others is difficult, as the reported results have been achieved on different test sets. In order to have a significant comparison, effort has been taken to obtain the testing data including songs and truth files from other researches. While the search for truth files was not successful, we were able to obtain the sound files used by Yoshioka et al. in [6]. The songs have been taken from the popular section of the RWC Music Database[17] (RWC-MDB-P-2001). Table 6.4 compares the results of Yoshioka with our results.

The algorithm designed by Yoshioka et. al. outperforms our algorithm in total and also on each of the songs. This is probably due to their use of a chord-sequence database and a better frequency spectrum. Compared to an acoustic version of the algorithm of Yoshioka that does not use chord sequences and bass tone information, our algorithm is still 6 percent inferior on average. It is interesting that the results for the individual songs do not correlate. Song number 44, for which Yoshioka measured the biggest inaccuracy of this test set, lies in the midfield of our evaluation results. The song that posed the most problems to our algorithm (Nr. 40) was the best rated song for Yoshioka.

Differences in the accuracies might also be due to different truth chord files and different extracts of the songs: Yoshioka reportedly use one-minute extracts. As we did not know which extract they chose, we computed our results simply on the first minute.

Piece Nr.	***Yoshioka***			***Genchords***	
	Beat Span	***Acoustic***	***Algorithm***	***Beat Span***	***Algorithm***
14	42%	74%	84%	40%	63%
17	57%	64%	76%	49%	68%
40	38%	76%	80%	37%	48%
44	34%	46%	67%	43%	63%
45	53%	68%	74%	52%	72%
46	57%	69%	80%	45%	66%
74	45%	71%	80%	38%	56%
Total	***46%***	***69%***	***77%***	***44%***	***63%***

Table 6.4: Result Comparison

17 http://staff.aist.go.jp/m.goto/RWC-MDB/

6.3 Confusion Matrix

		Guessed						
		Bbmaj	**Cmaj**	**Cmin**	**Ebmaj**	**Dmaj**	**Fmaj**	**Gmin**
	Bbmaj	**3.6**						0.1
T	**Cmaj**		**0.0**					
r	**Cmin**		2.3	**7.8**	0.9	0.1	1.3	4.0
u	**Ebmaj**				**0.0**			
t	**Dmaj**		1.8			**3.7**		0.3
h	**Fmaj**	2.0					**5.4**	

Table 6.5: Confusion Matrix: Elton John - Sorry Seems To Be ...

Confusion matrices give us an insight on the types of mistakes the chord detector makes. Table 6.5 shows the confusion matrix for Elton Johns "Sorry seems to be the hardest word" (song 4). The matrix that has been computed using labeldiff (Section 5.3.2) and show the confusion lengths in seconds. The true labels are listed horizontally and the recognized chords vertically. The diagonal (bold and black) shows the chords that have been recognized correctly. Mistakes are depicted in red. Two chords (C-Major and Eb-Major) have been recognized that do not appear at all in the song.

The more pitches two chords share the more easily the tend to be confused: The chord G-Minor (g-bb-d, last row) for example has been confused most often with Bb-Major (bb-d-f) and Eb Major (eb-g-bb) with both of which it shares two pitches. C-Minor and G-Minor, the chords that have been confused for the longest time, share one pitch. But also chords that seem to use totally different pitches sometimes are confused: for example D Major (d-f#-a) and C-Major (c-e-g). This normally has one of the following reasons:

- The detector is influenced by the previously played chord (C-D-G for example is a very common chord sequence (in functions IV-V-I), from which the confusion above could have resulted).
- The played chord is not a simple triad but uses additional pitches or alterations. Thus the two chords in reality do share some pitches. In our example the D-Major might be a seventh-chord. (d-f#-a-c). As the seventh is usually played very strong, this might very well be the reason for the confusion with C-Major.

0 pitch	*1 pitch*	*2 pitches*
2.4 seconds = 12.2 %	9.2 seconds = 46.9 %	8.0 seconds = 40.8 %

Table 6.6: Shared pitches of confused chords.

Table 6.6 summarizes the lengths of these confusion types calculated for song 4 and their contribution as percentage to the total inaccuracy.

7 Conclusion

7.1 Summary

In this thesis we have presented a sophisticated chord detection analysis model that incorporates music theoretical knowledge. It detects the key of the input and uses it to filter out those chords that fit the key. It further uses beat tracking to define the boundaries of possible chords and a smoothing algorithm to further enhance chord boundaries and chord change properties.

Tests

In order to get representative results a test set of 19 songs of various styles and genres has been assembled and hand-labelled. The effectiveness of each enhancement module has been evaluated against this test set independently of the other modules. Each of the modules has raised the average quality by several percent. In the final integration test accuracy ranged from 42% to 92%, with an average accuracy rate of 65%. Overall, the enhancements have improved the chord detection accuracy by 28% compared to a simple short-span detection algorithm, confirming our approach of integrating music theory in the chord detection process.

Tools

In addition to the chord detector itself a set of tools has been created to facilitate evaluation and use of the chord detector. These tools include programs to transpose and compare chord labelfiles. Special focus has been given to make the detection output directly evaluable, without the tedious work of hand-labelling truth files. This was achieved by resynthesizing the detected chord sequences. The user can thus get a direct notion of the detection quality by listening to a generated sound file that consists of the original file mixed with the detected and resynthesized chords.

7.2 Future Work

Throughout implementation and evaluation phase, possible enhancements came to our mind, that would have been too time-consuming to be integrated in this version of the chord detector. In this final section we want to state possible detection optimizations of our algorithm followed by interesting user interface enhancements.

Detection enhancements

One important enhancement of our chord detector would be to lift the restriction on key-modulation. Our system assumes that the key of the song does not change for the whole length of the input. In order to lift this restriction a modulation recognition module would have to be integrated into the chord detector. The chord filter could then be applied to whatever key currently dominates.

The system might be further improved by detecting not only the beat but also the meter of the song. Chord changes are most probable on the strong times of the meter (for example at beat one and three of a 4/4 meter). Knowing the meter, the chord boundary detection could be enhanced and analysis spans could further be stretched which would make the algorithm more robust against non-chord tones.

Another enhancement could be achieved by integration of a chord-sequence database that stores common chord sequences in the form of functions (e.g. I-IV-V-I). A set of possible chord sequences could then be evaluated against this database; an approach that is already successfully used by Yoshioka et. al. ([6]).

As we have seen in Section 6.2.6, the frequency based feature is misleading at about 20% of the time. As a consequence other additional features need to be integrated. The chord-optimization algorithm (smoothing) might be enhanced in this direction by allowing it to chose a chord independent of its acoustic score, if the surrounding chords suggest so.

Our algorithm uses enhanced autocorrelation for frequency detection. A variety of other pitch detection algorithms exist and it would be interesting to evaluate their integration in the PCP generation module. In the same way other PCP generation algorithms and PCP distance methods might further enhance the chord detection algorithm.

Implementation enhancements

In addition to the possible enhancements of the detection algorithm, several implementational enhancements are possible, including a platform independent implementation (e.g. using Java), a more efficient implementation of the smoothing algorithm and sequential processing of the input that supports real time analysis.

Improving the user interface and usability

The user interface too could be enhanced in various ways: A plugin for popular audio players could be implemented that displays the detected chords of the currently player music. Usability could be further improved by the implementation of a graphical front-end to the chord detector and its tools: A GUI could support visualisation of the PCP, the frequency spectrum, the detected chords and their computed probabilities. An editor could be integrated that supports manual adoption of the generated chord labelfile. Comparison and evaluation of the confusion matrices could be facilitated if their columns and rows were sorted by the circle of fifths and function names were displayed instead of absolute chord names.

Bibliography

[1]: René Brüderlin, Akustik für Musiker; Gustav Bosse Verlag; 2003

[2]: Reinhard Amon, Lexikon der Harmonielehre; Doblinger Metzler; 2005

[3]: Erich Wolf, Harmonielehre; Breitkopf&Härtel; 1972

[4]: Hugo Riemann, Handbuch der Harmonielehre; Leipzig; 1918

[5]: Hermann Von Helmholtz, Die Lehre von den Tonempfindungen als physiologische Grundlage für die Theorie der Musik; Minerva-Verlag; 1863

[6]: T. Yoshioka, T.Kitahara, K. Komatani, T. Ogata, H.G. Okuno, "Automatic Chord Transcription with Concurrent Recognition of Chord Symbols and Boundaries". In Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR), Barcelona, pages 100-105, 2004

[7]: A. Sheh, Ellis D.P.W, "Chord Segmentation and Recognition Using EM-Trained Hidden Markov Models". In Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR), Baltimore, USA, pages 183-189, 2003

[8]: B. Su, S. Jeng, "Multi-timber Chord Classification Using Wavelet Transform and Self-organized Map Neural Networks". In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Salt Lake City, USA, pages 3377 - 3380, 2001

[9]: L. Rabiner, M. Cheng, A. Rosenberg, C. McGonegal, "A Comparative Performance Study of several Pitch Detection Algorithms". In IEEE Transactions on Acoustics, Speech, and Signal Processing, 24(5): 399-418, 1976

[10]: J. C. Brown, M.S. Puckette, "A High Resolution Fundamental Frequency Determination Based on Phase Changes on the Fourier Transform". In Journal of the Acoustical Society of America, 94(2): 662 - 667, 1993

[11]: B. Doval, X. Rodet, "Estimation of Fundamental Frequency of Musical Sound Signals". In IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Toronto, Canada, pages 3657-3660, 1991

[12]: Xudong Jiang, "Fundamental frequency estimation by higher order spectrum". In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Istanbul, Turkey, pages 253-256, 2000

[13]: T. Tanaka, T. Kobayashi, D. Arifianto, T. Masuko, "Fundamental frequency estimation based on instantaneous frequency amplitude spectrum". In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Orlando, USA, pages I-329- I-332, 2002

[14]: Y. Zhu, M. S. Kankanhalli, "Precise Pitch Profile Feature Extraction From Musical Audio for Key Detection". In IEEE Transactions on Multimedia, 8(3): 575-584, 2006

[15]: A. P. Klapuri, "Multiple fundamental frequency estimation based on harmonicity and spectral smoothness". In IEEE Transactions on Speech and Audio Processing, 11(6): 804- 816, 2003

[16]: T. Tolonen, M. Karjalainen, "A Computationally Efficient Multipitch Analysis Model". In IEEE Transactions on Speech and Audio Processing, 8(6): 708-716, 2000

[17]: G. Tzanetakis, "Manipulation, Analysis and Retrieval - Systems for Audio Signals", PhD thesis, Princeton University, 2002

[18]: Carol L. Krumhansl, Cognitive Foundations of Musical Pitch; Oxford University Press; 2001

[19]: Frédéric Patin, "Beat Detection Algorithms",
Available at http://www.yov408.com/beat/BeatDetectionAlgorithms.pdf

[20]: S. Dixon, "Automatic Extraction of Tempo and Beat from Expressive Performances". In Journal of New Music Research, 30(1): 39-58, 2001

[21]: S. Dixon, "MIREX 2006 Audio Beat Tracking Evaluation: BeatRoot", Available at www.ofai.at/~simon.dixon/pub/2006/mirex-beat.pdf

[22]: S. Dixon, "Onset Detection Revisited". In Proceedings of the 9th International Conference on Digital Audio Effects (DAFX), Montreal, pages 133-137, 2006