FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Harmonic Change Detection from Musical Audio

Pedro Ramoneda Franco

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Gilberto Bernardes

July 28, 2020

Harmonic Change Detection from Musical Audio

Pedro Ramoneda Franco

Mestrado Integrado em Engenharia Informática e Computação

July 28, 2020

Abstract

In this dissertation, we advance an enhanced method for computing Harte et al.'s [31] Harmonic Change Detection Function (HCDF). HCDF aims to detect harmonic transitions in musical audio signals. HCDF is crucial both for the chord recognition in Music Information Retrieval (MIR) and a wide range of creative applications. In light of recent advances in harmonic description and transformation, we depart from the original architecture of Harte et al.'s HCDF, to revisit each one of its component blocks, which are evaluated using an exhaustive grid search aimed to identify optimal parameters across four large style-specific musical datasets. Our results show that the newly proposed methods and parameter optimization improve the detection of harmonic changes, by 5.57% (f-score) with respect to previous methods. Furthermore, while guaranteeing recall values at $> 99\%$, our method improves precision by 6.28%. Aiming to leverage novel strategies for real-time harmonic-content audio processing, the optimized HCDF is made available for Javascript and the MAX and Pure Data multimedia programming environments. Moreover, all the data as well as the Python code used to generate them, are made available.

Keywords: Harmonic changes, Musical audio segmentation, Harmonic-content description, Music information retrieval

Resumo

Nesta dissertação, avançamos um método melhorado para computar Harte et al.'s [31] Harmonic Change Detection Function (HCDF). O HCDF visa detectar transições harmónicas em sinais áudio musicais, fundamentais para a tarefa de estimativa automática de acordes dentro da Recuperação de Informação Musical (MIR) para um vasto âmbito de aplicações criativas que vão desde a harmonização a transformações harmónicas. À luz dos recentes avanços na descrição harmónica e transformação do audio musical, partimos da arquitectura original do HCDF de Harte et al. para revisitar cada um dos seus componentes, que são avaliados utilizando uma pesquisa exaustiva em grelha para identificar parâmetros óptimos em quatro grandes conjuntos de dados musicais específicos de estilos. Os nossos resultados mostram que os novos métodos propostos e a optimização de parâmetros melhoram a detecção de alterações harmónicas. Por $5,57\%$ (f-score) em relação aos métodos anteriores. Além disso, embora garantindo valores de recordação a $> 99\%$, o nosso outro método melhora a precisão em $6,28\%$. Com o objectivo de aproveitar novas estratégias para o processamento de áudio com conteúdo harmónico em tempo real, o HCDF optimizado é disponibilizado para Javascript e os ambientes de programação multimédia MAX e Pure Data. Além disso, estão disponíveis todos os resultados e o código python para gerar todos eles.

Keywords: Harmonic changes, Musical audio segmentation, Harmonic-content description, Music information retrieval

Acknowledgements

First of all, I want to thank my mum, my main music educator, for all my life. How can you teach an instrument to a child as a game, without the child regarding? I find it incredible. When the child understood it, it was too late. Thank you and dad for supporting me in every single moment of my life. I don't know what I'd do without you.

Many people would say I shouldn't put this here, because in the future this dedication may be blown away. However, it will be a reminder of everything we have lived. Thanks to Maria Valentina for supporting and inspiring me while I was doing all this work. Especially during the quarantine, I know that sometimes I am unbearable. I hope that you never get tired of my unbearably because I don't think I will get tired. And thank you also for all the help you gave me in making the graphics for this dissertation and the paper.

I especially want to thank Gilberto Bernardes. You suddenly received an Erasmus student from another university who spoke english badly and about which you had no obligation. And you gave him a very cool project, you helped him in everything you could and more, and you gave him all the opportunities he needed. Hats off, I've learned more from you than most of my professors. If I'm ever a teacher, I want to be like you. Even if we're not together next year, I know I have a mentor and a friend in Porto.

Thank you to my family in special to my grandma and Valentina to stay always there with me. To my other family, to all my childhood friends who are my friends now, thanks to God. For all, these years, for supporting me in every moment. For tolerating me, for studying two degrees at the same time, for your jokes and for your stickers. Thanks in particular to Alberto, Edu, Andrés and Carlos for helping me with my bad English in this dissertation. Also to all the friends I've made this year, Koreans, Laura and Pollo, my little kid, because our story doesn't end here. This year has been a great year.

Last but not least, Thank you to all the teachers who have taught me throughout my life. To those at the professional conservatory and the school. A lot of engineering teachers and very few from the superior conservatory. For all those who have learned and made me who I am, as a musician and an engineer. Thanks to the European Union for believing that the Erasmus do serve a purpose. And thanks to music and art, for making me enjoy myself and being the engine that moves the world.

Pedro Ramoneda

“Music, I feel, must be emotional first and intellectual second.”

Maurice Ravel

Contents

List of Figures

List of Tables

Abbreviations

HCDF	Harmonic Change Detection Function
SMC	Sound and Music Computing
MIR	Music Information Retrieval
ACR	Automatic Chord Recognition
HPSS	harmonic percussive source separation

Chapter 1

Introduction

The search for a mathematical basis in musical harmony has been a constant quest for thousands of years [23, 67, 68, 69, 44]. Ancient Greeks [26] believed music could be used as a tool to understand the natural world, an approach that has been sustained ever since[1]. Over the last two decades, information technologies have had a substantial impact on the newest findings. The computational processing of harmony has become a standard methodology across many disciplines, from musicology to audio signal processing. The topic under discussion in this dissertation, harmonic changes when chords change in time are understood from a Western perspective although this information is simpler in symbolic manifestations of music, such as a musical score, the retrieval of such information from audio poses significant challenges [35].

1.1 Motivation

In my longstanding classical piano studies, polyphony, and in particular harmony, has been instrumental in my formal education. Harmony can be regarded as a foundation part of music theory, composition, formal analysis, ear training, and choir, to cite a few.

On a broader scope, harmony is a central concept in Western music. It is distinctive to the composers' and artists' style, and as such, its importance while performing, improvising, and composing must be recognized. In the current view of computational creative tasks, where the computer is assuming a somewhat active role, harmonic change detection or, in other words, audio signal segmentation at the change of a chord, is the first phase for addressing a computational understanding of harmony.

Harmonic change detection has been addressed in the literature for the past two decades, in Sound and Music Computing (SMC) and Music Information Retrieval (MIR). Hainsworth et al. [28] and Harte et al. [31] are the main contributions to this problem. This work continues a line of

[1] Project Cosmos tries to model musical structures embedded in cardiac arrhythmias http://cosmos.ircam.fr

research of the Sound and Music Computing Lab at FEUP/INESC TEC on harmonic modelling, namely on the perceptually-inspired harmonic representation Tonal Interval Space [4, 6, 7, 61].

In Music Information Retrieval (MIR), the field of scientific research in which this dissertation can be framed, Harmonic Change Detection is a fundamental block in much Automatic Change Detection Recognition (ACR) algorithms. Hence, it is crucial to many tasks such as Cover Song Identification, Structural Segmentation, Genre Classification or Music Generation. Moreover, it is likely to provide many future applications, ranging from technical to creative tasks.

1.2 Objectives

The main objective of this dissertation is to revisit harmonic change detection from musical audio, a fundamental phase in the classical MIR task, automatic chord recognition. Furthermore, it can support various creative-related tasks associated with harmonisation, and other harmonic-related audio transformations, where a chord segmentation is needed. Aiming to improve current state-of-the-art methods for harmonic change detection, we will follow three specific goals:

1. To investigate the use of the Tonal Interval Space as a perceptually-enhanced representation for chord distances.

2. To perform a critical quantitative assessment of the state-of-the-art methods for the multiple component modules of the HCDF by Harte et al. [31], namely inspecting methods than have been recently proposed in the literature, such as novel chromagram representations and the adoption of harmonic percussive source separation.

3. Focusing on particularly challenging musical examples and genres and trying to avoid over-fitting problems. In the previous researches [31, 21], only the 16 songs presented in the first article [21] were taken into account. The use of small datasets can lead to significant biases, making it impossible to generalise the model to any musical audio, even to generalise the model to musical audio with the same genre as the 16 songs.

4. To explore visualisations and mining techniques which can promote better computational musicology and retrieval tasks in large audio datasets.

1.3 Approach

To meet the objective defined in Section 1.2, we established a methodological path which covers the following phases. To address the purpose, first of all, we will revisit the HCDF proposed by Harte et al. [31]. Specifically, we consider multiple parameterisations in its component modules, and we examine the implication of different tonal spaces. In particular, the recently proposed Tonal Interval Space by Bernardes et al. [4] in improving a perceptual spatial representation of tonal pitch. Several elements of Harte et al.'s original pipeline have been improved during the last 15 years, and in this dissertation, we test them, as well as we try to bring possible new improvements.

Moreover, Examining HCDF on different genres and with various settings could produce new research outputs and allow us to understand better the method. Datasets of different genres were proposed by Harte et al. 15 years ago. Given that it is possible to understand how HCDF works across various genres, we believe it is possible to develop a robust generalized HCDF method.

Another objective is comparing different tonal spaces and their performance in different scenarios and analysing how the different blocks of the pipeline and parameterizations relate to each other. As it is stated in objective 3, it is important to make a critical analysis from a more musical perspective of how the algorithm works. We use a musicological approach with the aim to understand concepts already existing in western music.

To sum up, in general, the conducted research seeks to ascertain whether the Tonal Interval Space, a novel mathematical framework which captures perceptual affinity in tonal pitch as distances, outperforms the current state of the art in harmonic change segmentation and function analysis.

1.4 Structure of the dissertation

The following paragraphs try to give an overview of how this dissertation will be organised. In this first chapter 1, we have made a small approximation of why and how we are going to approach to harmonic change detection.

In Chapter 2, we will deal with everything that has been done in the field of the study of harmonic change, starting with the approaches from the Music Technology field. And all the background on which it is based, that is, technological, musicological or cognitive. The previous proposals and structures will also be dealt with to try to analyse harmony automatically.

In Chapter 3, we will analyse all the new possibilities that we propose to extend the pipeline proposed by Harte et al [31].

In Chapter 4 shows how the problem has been solved from a more technical perspective. After this, we will analyse the results obtained from a data mining and musicology perspective.

In Chapter 5, we propose application scenarios for harmonic change detection. Finally, in Chapter 6, we review everything that has been done in this dissertation, and we propose what is going to be done in the future.

Chapter 2

State of the Art

The computational processing of musical harmony has been a topic of interest since the early days of computer music [87, 43, 81]. Over the last 20 years, we have witnessed the creation of many associations and communities at the intersection of computing and music, such as as the Sound and Music Computing Network and the Society for Musical Information Retrieval. Not only these societies have vastly incremented the visibility of this area, but also have promoted international venues for the dissemination and discussion of research outputs. Like many other areas, the growth of music technology is due mainly to the evolution of computing, information technologies, signal processing, artificial intelligence, machine learning and, in recent years, deep learning. On the other hand, the longstanding musicological tradition has equally responded with different methods and approaches with access to computational advances. Computational musicology is a branch of the early musicological area that exists at the intersection between music and technology [20, 48, 17].

2.1 State of the Field

In the early 1950s, a dedicated group of composers, engineers and scientists began to explore the use of new digital technologies to create new types of music. Music and technology have had a fruitful history ever since. Many terms, such as *computer music* and *music technology*, have been used for what today is commonly referred to as *sound and music computing* [74]. In 1974, it was established the International Computer Music Association and the International Computer Music Conference. In 1977, the Computer Music Journal was founded. The Center for Computer Research in Music and Acoustics (CCRMA) at Stanford was created in the mid 1970s, and the Institute for Research and Coordination Acoustic/Music (IRCAM) in Paris was established shortly after, as the principal department of the Pompidou Center. These two centres, together with the nowadays inactive composition department of Princeton University, lead research in music technology throughout the 20th century. Over the past 25 years, many other centres have emerged throughout the world, with different proposals and objectives, trying to bring together music and technology. Such is the case of our laboratory, the Sound and Music Computing group, at the Faculty of Engineering of the University of Porto (FEUP) and INESC TEC.

2.1.1 Music Information Retrieval

Music Information Retrieval (MIR) is the interdisciplinary application of information retrieval to music. MIR is part of the Sound and Music Computing (SMC) field, that encompasses all-new technologies applied to music. It is one of the most fruitful fields within SMC over the last 20 years. MIR is a small but growing field of research with many real-world applications ranging from multimedia apps to recommendation systems in big streaming companies. The MIR community includes experts in musicology, psychology, cognitive sciences, music, data science, digital signal processing, machine learning, theoretical computing, and some combination of these, making it a very interdisciplinary field. It covers a lot of topics, such as audio beat tracking, automatic chord recognition, audio classification tasks (e.g., genre, year, composer, and mood), cover song identification.

MIR tasks which aim to extract information from audio signals, such as the topic under research in this dissertation, have captured a lot of attention from the community. The challenge to abstract higher-level information from audio signals is due to the low-level representation of audio signals.

On the specific domain of harmony, extracting relevant information from audio promotes important lines of research within MIR such as structural segmentation, score following, audio cover song identification, audio chord estimation, audio melody extraction, automatic multiple fundamental, and automatic chord recognition.

SMC uses multi-disciplinary methodologies across the technological, scientific and artistic domains. Furthermore, SMC shares common objectives with many other fields of knowledge, such as the relationship between perception and action and the integration of different human senses in the way of relating to machines, both individually and collectively [62, 86, 8].

The scientific and technological outputs aim at modelling, measuring, analysing and creating music technology tools. Artistic methodologies focus on exploring human experience, aesthetics and expression. Art and technology can enhance each other [75].

2.2 Fundamentals of Harmony: A Music Theory Perspective

In this section, we review concepts and terminology deeply rooted in Western music harmony as backgroud for the remainder of the dissertation.

In music, the study of harmony involves chords and their construction, as well as chord progressions and the connecting principles that carry them out. Harmony is the combination of notes simultaneously to produce chords and successively, to produce chord progressions. The term is used descriptively to denote combination of chords and notes, and prescriptively to denote a system of structural principles governing their combination. In the latter sense, harmony has its own body of theoretical literature, as is defined in the New Grove Dictionary of Music [50]. In Riemann's theory of harmonic function, harmony is the essence of all chords having a likehood function, and thus exists at a much more abstract level than chords with their inversions and notes 'foreign to the harmony'. This last definition fits with the computational approach made in this dissertation.

It should be noted that the concept of harmony refers to the structure of the intervals and their combinations or chords and their relationships, not so much to the structure of the piece itself as explained below.

Harmony: Other points of view

Apart from the purely musical perspective discussed in the previous section, the topic of harmony has been analyzed from other points of view. This section covers them other points of view.

McDermott and Oxenham [45] present a study about music and pitch from a perceptual perspective; they propose the need of a new way to represent chords from a neurological perspective. In their research, they argue that in some pitches cases listeners perceive the different pitches individually, whereas in other cases they perceive pitches as a composited sound. McDermott and Oxenham proposed that chords with more than 3 notes are perceived as a composited sound by most listeners.

In [45] they explore consonance from a neurological perspective as well. Consonance is another widely studied property of chords. From a Western perspective, there is consonance when it is pleasant for the listener, and dissonance (non-consonance) when there is no pleasure for the listener. From Helmholtz dissonance theory, the dissonance is very related to beating or fluctuation of the amplitude of superposition of two tone's partials with adjacent frequencies. This theory is very extended among string musicians, and it is a popular way to tune.

For harmonic sounds, the pitch is a perceptual correlate of tonotopic representations that are observed from the cochlea to the auditory cortex. This part of the brain allows us to understand quickly the fundamental partial (F0) discriminating the remaining partials. This phenomenon enables humans to sense music as we do. Humans have an inherent pitch relative ear skills from a young age, but these skills are not too much studied, it is said in [45].

In the context of MIR, harmony is a fundamental search, in many domain areas as Audio Chord Estimation (ACE), Audio Cover Song Identification, Audio Key Detection, Discovery of Repeated Themes and Sections or Structural Segmentation. ACE software systems process audio recordings and predict chord labels for time segments. One of the current objectives is to incorporate the knowledge of music and neurology into the existing methods of machine learning and deep learning applied to music.

Notes

A note is the symbol used to represent the duration of a sound and, when it is on a music pentagram, to indicate the pitch of the sound. Notes are the basic musical unit defined by a height, a duration, an intensity and an instrumental timbre. The note concept helps us to understand music in terms of harmony, rhythm, melody, etc. However, the sound reality is more complex than the parameters expressed in the notes; tensions, artist performance, even partials and other parameters presented in a musical sound are very important too. From this section's perspective, we are focusing on the symbolic way. Discrete symbolic information of western music representation allows people to understand a complex phenomenon such as music or harmony.

Interval

Interval is the difference in pitch between two sounds [63]. However, the term used in this dissertation is contemplated from a western tempered music system and with a symbolic perspective. In this system, intervals are noted with two names. The first name is the distance in absolute notes (with no count of sharps and flats) taking in count in which key is contained this interval. The second name is a traditional notation in classical music for representing the number of semitones, as is shown in Figure 2.1. As it can be seen in Figure 2.1 each number of semitones have two possible nomenclatures. That is because the same interval can be represented in different ways on the score being different names first.

Number of semitones	Name of interval
0	Unison
1	Minor second, Augmented unison
2	Major second, Diminished third
3	Minor third,Augmented second
4	Major third,Diminished fourth
5	Perfect fourth, Augmented third
6	Tritone|Diminished fifth, Tritone|Augmented fourth
7	Perfect fifth, Diminished sixth
8	Minor sixth, Augmented fifth
9	Major sixth, Diminished seventh
10	Minor seventh, Augmented sixth
11	Major seventh, Diminished octave
12	Octave|Perfect octave, Augmented seventh

Table 2.1: *Number of semitones and corresponding interval name.*

We name root the first note from which the interval is counted. An interval is said to be an inversion of another interval if it is obtained from it by subsequently moving notes one octave above or below. There is a set of rules to compute the name of the inverted interval from the name of the original one. For example, if the first name of the interval changes after the inversion procedure, the second name changes according to the rule: *major and minor are exchanged, augmented and diminished are exchanged, and perfect keeps.*

Moreover, major and perfect intervals are the intervals presented between the tonic note and any note of the scale, as is shown in Figure 2.1 with C major as an example.

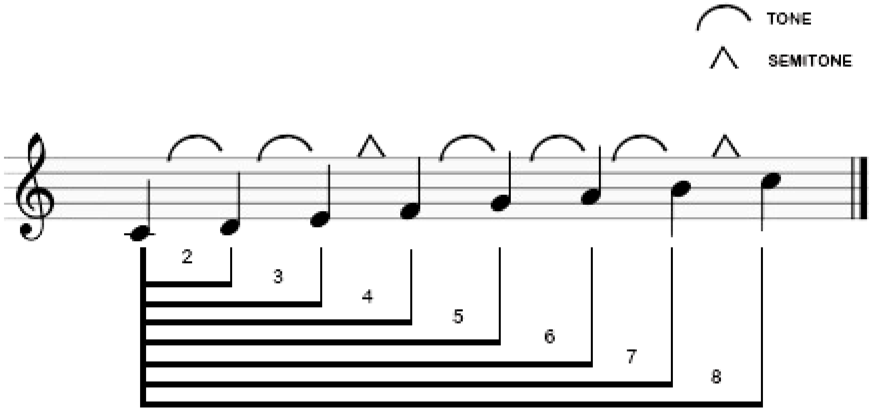


Figure 2.1: *Chromatic scale with intervals from Do (C).*

Chords

A chord is the simultaneous sounding of two or more notes. In other words, a set of three or more notes that form structure by overlapping their sounds directly, indirectly (as an arpeggio) or a mixture between these two methods. Whereas melody implies the horizontal aspect of music, harmony refers to the vertical dimension of music

A chord is a fundamental structure to perceive more high-level music structures. It tries to explain the behaviour and relationships of various sounds at the same types. Throughout history, several chord notations have been developed, and different types of compound sounds and their relationship have been theorized between different chords (chord progressions). All of these topics will be discussed below.

Chord Types

There is a very large number of chords. Any superposition of sounds is a chord, even a cluster (three adjacent notes). That is why in this section, we are going to discuss chords from a Western musical theory in general terms. Later, this chapter will focus on chord types, which are were related to MIR is centred [59]. All the chords are going to be named with their English/American notation, explained in the next section.

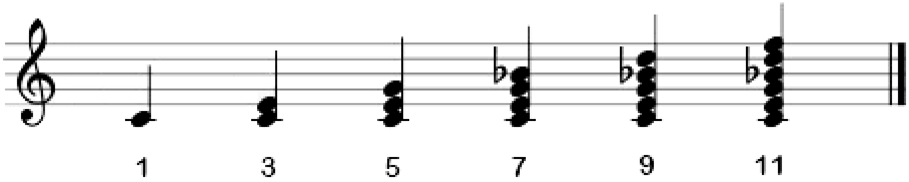


Figure 2.2: *Illustrative example of how to build chords adding thirds.*

Tonal chords are formed from a note called the fundamental note or root note, usually the lowest. The rest of the chord is compounded usually adding consecutive thirds from the root note, as is shown in Figure 2.2. In the tonal system, chords are formed by superimposing third notes. In the context of a given tonality or mode, a chord is named with a Roman numeral and called *a* *degree*. This is a functional notation that forgets the particular notes of the chord and only recalls its function on the tonality. It is not necessary for all the notes to be present in order to understand the chord. The third or fifth, for example, can be implied. Under certain conditions (dominant chords), the chord can be understood even if its fundamental is not present.

The type of chord depends on the intervals between the notes of the chord. Another name for this is the quality of the chord. The major is a type of chord quality, as for example minor.

Now, let is discuss the most common types of chords, how to build them and where to find them. The following chords are Major, Diminished, Major Seventh, Minor Seventh, Dominant Seventh, Suspended, Augmented and Extended.

- A major chord is structured by a root note (1^{st}), a major third (+4 semitones), and a perfect 5^{th} (+7 semitones). An example with root note C can be found in Figure 2.3 1 chord.
- A minor chord is structured by a root note (1^{st}), a minor third (+3 semitones), and a perfect 5^{th}(+7 semitones). An example with root note C can be found in Figure 2.3 2 chord.
- A diminished chord is structured by a root note (1^{st}), a minor third (+3 semitones), and a diminished/flat fifth (+6 semitones). An example with root note C can be found in Figure 2.3 3 chord.
- A major seventh chord is structured by a root note (1^{st}), a major third (+4 semitones), a perfect 5^{th} (+7 semitones), and a major 7^{th} (+11 semitones). They can be understood as major triads (i.e. major chord) with a major 7^{th} on top. An example with root note C can be found in Figure 2.3 4 chord.
- A minor seventh chord is structured by a root note (1^{st}), a minor third (+3 semitones), a perfect 5^{th} (+7 semitones), and a minor 5^{th} (+10 semitones). They can be understood as minor triads with a minor 7^{th} on top. An example with root note C can be found in Figure 2.3 5 chord.
- A dominant seventh chord is structured by a root note (1^{st}), a major third (+4 semitones), a perfect 5^{th} (+7 semitones), and a minor 7^{th} (+10 semitones). They can be understood as major seventh chords with the top note lowered by one semitone. An example with root note C can be found in Figure 2.3 6 chord.
- A sus4 chord is structured by a root note (1^{st}), a major fourth (+5 semitones), and a perfect fifth (+7 semitones). They can be understood as major chords with a perfect fourth instead of a major third. An example with root note C can be found in Figure 2.3 7 chord.
- An augmented chord consists of a root note (1^{st}), a major third (+4 semitones), and an augmented 5^{th} (+8 semitones). They can be understood as major chords with the top note with going down one semitone.
- An extended sound is structured by major or minor chord with additional triads. With 7^{th}, 9^{th}, 11^{th} and 13^{th}. An example with root note C can be found in Figure **??** 8 chord.

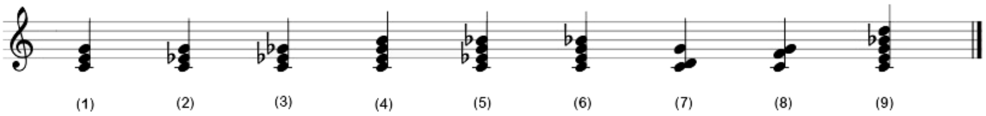


Figure 2.3: *Different types of chords.*

Chord Notation

Chords can be represented in various ways. The most common notation systems are described by Benward and Saker [3].

Macro analysis or Plain staff notation, write the chord in fundamental position, that is, ordered by triads starting from the fundamental note. Macro analysis is an analytical process that can be used in conjunction with more conventional methods of analysis. Macro (meaning large) refers to the basis of the method: to reveal the most basic function of the music. More complex patterns look much better thanks to this technique. It gives a more general perspective of a composition than the use of Roman numerals. All this notations are showed in Figure 2.4.

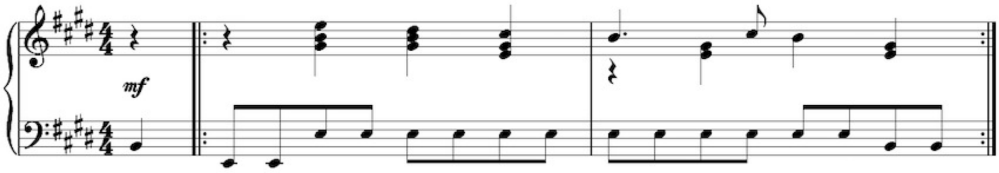
Roman numerals are commonly used in harmonic analysis to denote the scale step on which the chord is built. In short, Capital Roman numerals are Major triads, for example, I, IV, V. Lowercase Roman numerals are Minor triads, for example, ii, iii, vi. Lowercase Roman numerals with ^{o} are Diminished triads. For example, ii^{o}, vii^{o}. Capital Roman numerals with $^{+}$ are Augmented triads. For example, III^{+}.

Figured bass, much used in the Baroque era, uses numbers added to a bass line written on a staff, to enable keyboard players to improvise chords with the right hand while playing the bass with their left hand. In this system of musical notation, only the bass are written and the inversion in which the chord has to be developed.

English/American notation is sometimes used in modern musicology to denote chord root and quality. Quality defines which type of chord is describing, but it does not describe the inversion. This type of notations is used in popular music lead sheets, fake books, and chord charts, to easily allow musician improvise, to jam and vamp it. Chord letters are the system used Harte [29], and it has been adopted by the MIR community [59]. English/American notation is formed following the next rules:

- A letter (in capital letters) denoting a root note, such as D.
- An abbreviation or symbol denoting the quality of the chord (e.g. minor, aug or o) If no symbol or abbreviation is indicated, it is understood that the chord is a major triad.
- Numbers that indicate the intervals of the highest triads on which the chord is structured, such as 7 or 13.
- For special alterations of particular intervals of the chord, such as $5b, 5\#$ or add13.

- If the slash symbol is between two notes "/" and means that the lowest note is different from the root. For example, C/F indicates that a C major triad with an F added to the bass should be played.



TAKING IN COUNT MELODY						
MACRO ANALYSIS	E	–	–	–	–	–
ROMAN NUMERALS	I	–	–	–	–	–
AMERICAN NOTATION	E	–	–	–	–	–
FIGURED BASS	5	–	–	–	–	–
NO TAKING IN COUNT MELODY (AS HCDF)						
MACRO ANALYSIS	E					
ROMAN NUMERALS	I		I_7	I	VI_6	I
AMERICAN NOTATION	E		E_7	E	6	E
FIGURED BASS	5		$\frac{5}{7}$	5	6	5

Figure 2.4: *Different notations comparison.*

Chord Progressions

In a musical composition, a chord progression or harmonic progression is a succession of chords. Chord progressions are the foundation of harmony in Western musical tradition from the common practise era of Classical music to the 21st century. Chord progressions are the foundation of Western popular music styles (e.g., pop music, rock music) and traditional music (e.g., blues and jazz). The best way to study harmonic progression is to consider progressions in groups according to the interval produced by the roots of two adjacent chords. The following general categories will form the basis of our study of harmonic progression.

Undoubtedly the most common of all harmonic progressions is the circle progression, shown in Figure 2.5. More than any other, this progression has the capability of determining a tonality, giving direction and thrust, and providing order in a section or phrase of music. It is indeed the basis of all harmonic progression [3].

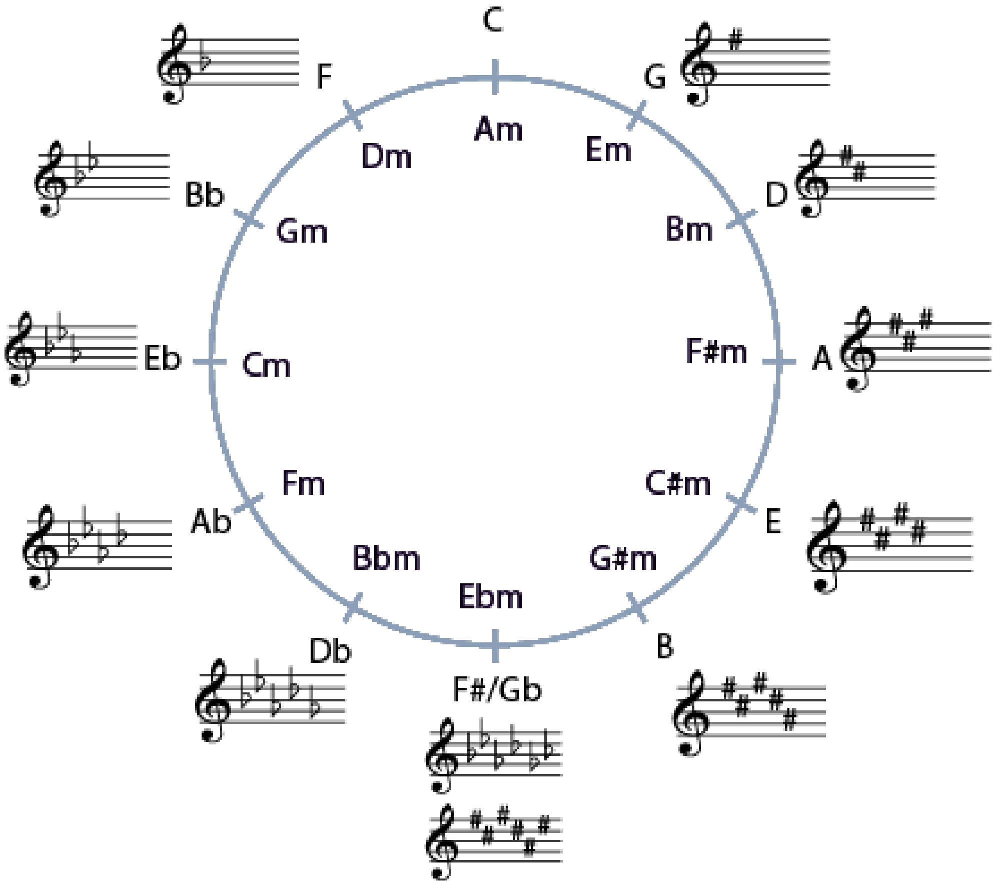


Figure 2.5: *Traditional circle of fifths diagram.*

Two concepts widely used in this dissertation are **harmonic change** and **harmonic rhythm**. On the one hand, harmonic change refers to transitions from one chord to another on musical audio. On the other hand, harmonic rhythm is the rate at which the chords change, in a musical composition. It can be considered from a merely time perspective or from the relation of chords per note.

Keys

In music theory, the key of a piece is the group of tones, or scale, which forms the basis of a musical composition from classical music, through pop or rock music to traditional music. From a reduced perspective and in relation to the chords of the 18^{th} century onwards, the key of a piece has been explained as the result of a reduction of the three main chords, the tonic, the dominant and the subdominant.

The key is composed of the tonic note and its corresponding chords also called the tonic chord. They provide a subjective sense of arrival and rest, and also have a relationship to its nearby tonalities, its corresponding chords, and the tonalities and chords outside the group. Notes and chords different from the tonic create different tensions, which are resolved when the note or the tonic chord returns.

In the Western music tradition, the tonality can be in the major or minor mode. Popular songs are usually in a particular major or minor key, and so is classical music during the period of common practice, around 1650-1900. Longer pieces in the classical repertoire may have sections in contrasting keys.

Modulations

In music, modulation is the change from one tonality (tonic or tonal centre) to another. This may or may not be accompanied by a change in the key of the tonality, modulations structure the form of many pieces, as well as add interest. Treating a chord as a tonic for no more than one phrase is not considered a modulation. Modulation describes the process in which a piece of music changes from one key to another completely.

When you start writing a piece of music, one of the first things you do is choosing a key to compose. This choice of tonality determines which scale is used, how many sharps and flats there are, and which chords can be used. This tonality is sometimes called the "starting tonality".

Many songs and pieces remain in this starting key and do not change. However, to make a piece more interesting composers often change to a different key at some point in the piece. This change is a modulation.

Musical Structure

Form in music is the result of the interaction of all structural elements. It consists of by different phrases and periods, but structure refers as a whole, the organisation of a complete composition, and takes into account harmonic, rhythmic and melodic elements.

A piece of music can generally be divided into two or more main sections, and the boundaries between these sections are called formal divisions. The formal divisions are the result of strong harmonic and melodic cadences and rhythmic factors and have been widely studied the centuries. Formal divisions define the sections of a composition, and these sections are labelled with capital letters: A, B, C, etc. If a musical section is repeated, the same letter is used: A, A, B, B, etc., and if it contains similar material, it is designated by adding cousins to the previous letter: A, A’, A", etc. Popular songs, song and rock usually have a chorus that is repeated for example A, A’, A", etc., and several different stanzas B, C, C’, D, etc.

2.3 Audio Content-Based Processing

Music can be represented differently, from musical scores, at a symbolic level, to the registration of its performance, in audio formats. Audio content-based processing aims to address the latter case and extract or abstract information, named descriptions or features, from raw audio. Audio Content-based processing supports many audio-driven applications, such as automatic speech recognition, audio segmentation, music recommendation or environmental sound retrieval. Accessing information by processing the content of an audio signal has been addressed by many disciplines from musicology to audio signal processing to find a new way of modelling raw audio as a more complex phenomenon. Content is a very used term in multimedia retrieval. In this chapter, content from a music perspective is the central axis. The other term analysed in this chapter is processing from a digital signal processing perspective.

First of all, we are going to focus on the content term. Information science and linguistics offer the meaning of term content. But we are going to focus on a Multimedia perspective cover on several past research [62, 49]. The Society of Motion Picture and Television Engineers (SMPTE) and the European Broadcasting Union (EBU) defined content as the combination of two concepts: metadata and essence. The essence is the raw program material itself, the data that directly encodes images, text, video, etc. or as in our case raw audio. Essence information can be encoded for directly represent the actual message, and it is usually presented ordered by time. On the other hand, metadata is the descriptors of the essence and its varieties. SMPTE/EBU presented a classification for metadata:

- Essential (meta-information that is necessary to reproduce the essence, like the incompatibilities, the number of audio channels, the Unique Identifier, the video format which is encoded raw data, etc.).
- Access (who can access to the essence, as legal access, i.e. copyright and licenses);
- Parametric (how the essence have been captured, number of capture machine, types, location).
- Relational (how to synchronize essence encoding, e.g. time-code).
- Descriptive (descriptors of the essence that can allow users catalogue, search, retrieval and administration of content).

Other classification presented by the National Information Standards Organisation only considers three different metadata types:

- Descriptive metadata, which describes the essence.
- Structural metadata, which describes how parts of the essence have relationships, such as video and audio.
- Administrative metadata, which describes management information, permissions, the tier of security, date of creation, who have accessed it.

In general, meta-data is all the information that can be retrieved from a media essence. In other words, any information that can be annotated or extracted over a music piece in a meaningful way. MPEG-7 standard has a metadata slot which is defined as a content descriptor, "a distinctive characteristic of the data which signifies something to somebody" [39]. This approach to music analysis has a very big problem [79, 78], the semantic-gap problem, which arises from the discrepancy about extracted metadata and how is the metadata concept perceived by one user in a given situation. That is why it is essential to keep the data most objectively for instancing; it depends on the circumstance. A way to solve the semantic-gap problem is to divide metadata in a hierarchy [10], low-level, mid-level and high-level. It is easy to understand how different a content description of a music piece is if the targeted user was an amateur listener or an expert musicologist. The education of his designer can bias even a low-level descriptor such as the Spectral flux, and another digital audio designer would have designed it differently.

- Low-level descriptor can be computed from raw data indirect or derived way (i.e. after signal transformations like Fourier or Wavelet transforms, after statistical processing like averaging, after value quantisation like the assignment of a discrete note name for a given series of pitch values, etc.). Most of the low-level descriptors don't have the sense from a musical perspective. However, they can be easily computed from a computer.

- Mid-level descriptors require and induction from validated data to an abstract concept. In this category would be ideas typically musical like chords or harmonic change derived from other descriptors, or a Hidden Markov Model or a Deep Learning model that segment a song by timbre similarities. Machine learning, Deep learning and statistical modelling make mid-level descriptors possible. Mid-level descriptors are also sometimes referred to as object-centred descriptors.

- The difference from low-level and mid-level descriptors to high-level descriptors is that the is a subjectivity concept mixture with the technical concept. For example, piano and forte depend on the rest of the piece dynamics and how the listener perceives it. More abstract concepts can be retrieved from essentia as aggressivity, melancholy, dissonance, beauty, etc. High-level descriptors are named user-centred descriptors too.

Due to a wide variety of descriptors, have been presented some frameworks to build it. The Dublin Core and MPEG- 7 are currently the most relevant standards for music content descriptors.

The other big part of audio-content processing is **processing**. The formal definition is to subject a thing to a process of elaboration or transformation; however, usually denotes a functional or computational approach to solve some scientific problems automatically. "Signal processing" is the central processing in audio for processing and modelling it, there is another processing involved as language processing, visual processing, speech processing, or knowledge processing. It should be noted that the difference between prescription and algorithmic is functional; in processing, the critical thing is not how it is done but what is achieved by doing it. And all the processing of music is a synergy between Signal Processing, Artificial Intelligence, Information Retrieval, and Cognitive Science to be able to process and model music.

Audio Content-Based Processing has been a growing field in music technology [49]. There are many different descriptors developed to understanding audio as Temporal descriptors, Physical frequency descriptors, perceptual frequency descriptors, Cepstral descriptors, Rhythm (modulation frequency descriptors) eigendomain descriptors or phase space descriptors. Moreover, the state of research is mature and some databases can be useful in perceptual research, in music information retrieval and machine-learning approaches to content-based retrieval in large sound databases as some types of descriptors as timbre descriptors [60]. In addition, there are software libraries with several audio-content processing tools, such as *essentia* [11], *madmom* [9] and *librosa* [47], consolidated and supported by a large community, many of them research centres.

This dissertation covers two perspectives of audio content-based processing. The first is a musicological view. The musicological angle tries to follow the Western musical concepts developed for years. The second is computational. Computational perspective refers to content-based audio processing strategies, and this means that automatically give information about the musical signal thanks to descriptors. Musically motivated ones try to imitate western musical concepts.

2.4 Harmonic Description of Musical Audio

Harmony is a very abstract concept developed from human perception [83]. Descriptors have to be meaningful in order to generate valuable information. Melody (sequence of single pitches), fundamental frequency, pitch classes and chords (simultaneous combinations of pitches), and chord progressions, harmony and key (temporal combinations of chords) are descriptors very useful for understanding harmony [49].

The search for a robust harmonic sound representation has been continuous for the last 50 years. Harmonic features are noisy from a lot of perspectives. Musical instruments produce overtones summed to F0; percussive instruments and different timbres and instrumentation produce perversions in the representation of harmony.

Harmonic content-based audio descriptors are mainly vertical (i.e., chords), however, they have a deep relationship with horizontal part (i.e., melodic and voice-leading). Furthermore, higher-level descriptors, such as the concept of musical key or connotations and tensions, are in a strong relationship with mid-level harmonic descriptors. The first attempts to address the harmonic domain was from symbolic manifestations of music such as those encoded using the MIDI standard [16]. However, there are too many differences between the symbolic domain and audio domain. The latter requires dedicated methods. Furthermore, attempts to translate from the audio domain to symbolic have not been prolific [82]. Moreover, MIDI research is very biased towards piano transcription. The search for a robust set of descriptors and structures about sound, polyphony and harmony is central to the many MIR tasks. Furthermore, research to link harmonic descriptors to semantic information (high-level descriptors) or even other types of information have been essential [89].

Harmonic content-based audio descriptors on which we have focused in this dissertation are Mid-level descriptors, such as chromagram, chords, *Tonnetz* based descriptors and harmonic boundaries. Chromagrams are one of the most used methods in the MIR field. They are a simplification of chords, and different chromagrams have been proposed in the literature. Furthermore, the Harmonic Network or *Tonnetz* is a typical representation of the relationships between harmonies in musicology, typically attributed to Euler, used by music theorists such as Riemann and Oettingen in the 17th century [71, 70, 57]. The generalization of this led to the spiral array created by Elaine Chew, a mathematical model to try to understand tonality.

2.4.1 Spectrogram

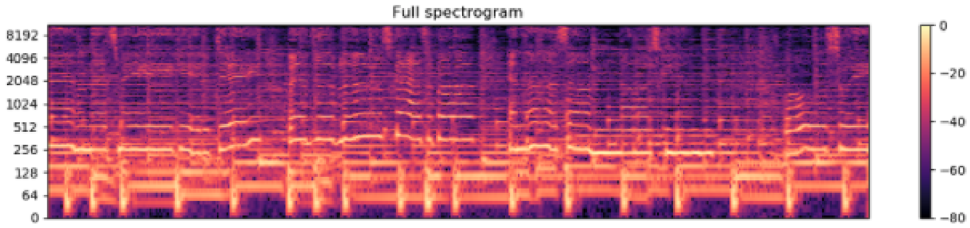


Figure 2.6: *Example of typical spectrogram. Computed with librosa [47].*

The spectrogram is the signal representation of splitting different ranges of frequencies as a histogram. The result is a three-dimensional graph that represents the energy of the frequency content of the signal as it varies over time, see Figure 2.6. It is a fundamental pillar trying to represent audio. Harmony is an example too, in particular, the first step of some chromagrams as is explained in the next section.

2.4.2 Chromagram

In what follows we detail the most used chromagrams. It includes from simpler and earlier methods for chromagram computation such as C_{STFT} [22] or C_{CQT} [15], that directly map the window-based spectral analysis from the preprocessing block into 12 chroma elements, to more robust chromagrams such as the C_{NNLS} [42] or C_{HPCP} [27], which include additional processing for enhancing the transcription or the tunning of the representation. Figure 2.8 shows the four chromagrams adapted for a major chord. Moreover, other types of chromagrams are mentioned in the following section.

One of the most prominent harmonic-content audio descriptors is the chromagram, which accumulates the energy of an audio signal as 12-element vectors, as is shown in Figure 2.7, representing the 12 notes of the chromatic scale across all octaves. It was first proposed in 1999 by Fujishima [25]. Many algorithms for chroma computation have followed, such as pitch class profiles [25], harmonic pitch class profiles (HPCP) [27], the CRP chroma [52] or the NNLS chroma [42]. Last Deep Learning chromagrams have attempted to get a binary chromagram with only the notes than could be activated. Their differences stem mostly from different degrees of invariance to a particular musical attribute (e.g., timbre) or some enhanced level of performance towards a symbolic-based representation (e.g. harmonics or transient noise).

Short Time Fourier Transform chromagram C_{STFT} [22] results from mapping and accumulating the energy from the frequency bins of a short-time Fourier transform representation into their corresponding pitch class.

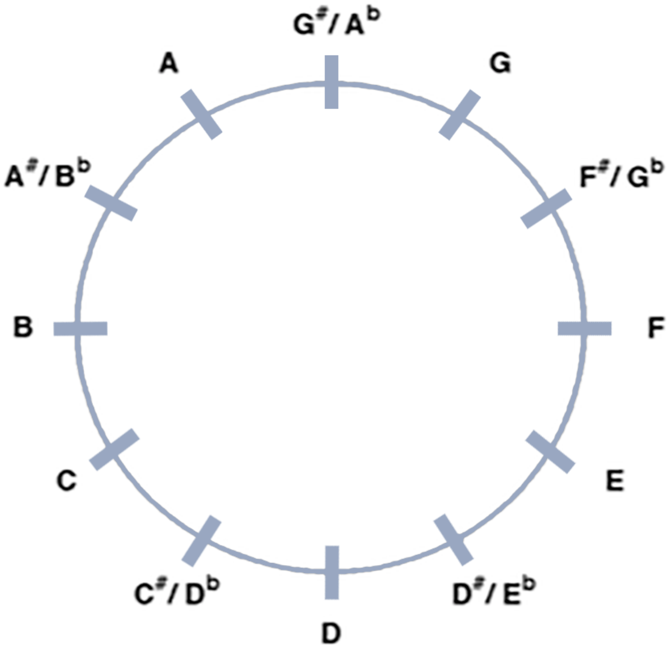


Figure 2.7: *Chomagram. A 12-pitch-class circular array for modelling chords and harmony.*

Constant-Q Transform chromagram C_{CQT} departs from the logarithmic scaled Constant-Q transform as the base representation for the mapping spectral bins to the 12 pitch classes of a chromagram. Due to its logarithmically spaced spectral basis, this chromagram is better aligned with human perception. However, the constant-Q transform [15] is heavier to compute when compared with the fast Fourier transform (FFT) used in C_{STFT}. Computing this algorithm with greater robustness involves the application of CQT (via FFT) octave-by-octave, using lowpass filtered and downsampled results for sequentially lower pitches. On the C_{CQT}, usually, the subsampled method and the direct FFT method are combined. In the end, the process is carried out with all the necessary frequencies.

Harmonic Pitch Class Profile chromagram C_{HPCP} [27] is tuning-independent and discards the presence of noisy transients by departing from a sinusoidal component analysis of the audio signal. The C_{HPCP} makes a first approximation of the tuning, based on the Western well-tempered system, to have a reference frequency. Furthermore, the accumulated energy in each pitch class of the C_{HPCP} is computed from the fundamental frequency and harmonically aligned partial frequencies. This procedure aligns with human perception as our hearing system typically fuses these partials into a unique auditory image.

Non-Negative Least-Squares chromagram C_{NNLS} [42] uses a Non-Negative Least-Squares (NNLS) optimization algorithm to approximate the transcription of the notes before the chroma computation. In the beginning, a log-frequency DFT spectrum is computed. Later, the global equal-tempered tuning frequency of all the piece is estimated from the spectrogram. Then, the previous log-frequency spectrogram is recalculated using linear interpolation and taking into account the global frequency. After that, the spectrum background is calculated and removed from

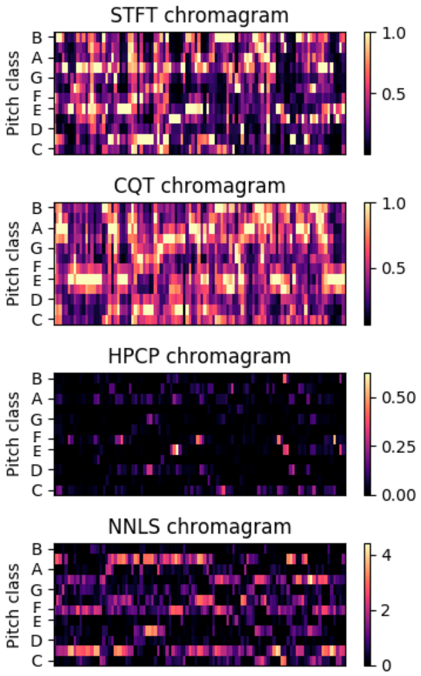


Figure 2.8: *Different chromagrams for an A major chord.*

the above spectrum, as indicated in [42]. After the NNLS decomposition, the spectrum mentioned above is mapped into a 12 bin chroma. Therefore, this chromagram should not have any non-tonal components like transient noise, since it is calculated in a symbolic way.

Deep Learning chromagram C_{DEEP} [42]. In the last few years there have been several attempts to do a chromagram with deep learning. Good examples are the deep chroma extractor [37] and crema-pcp [46].

Chroma DCT-Reduced log Pitch [52]. The general idea is to discard timbre-related information with certain Mel-frequency cepstral coefficients (MFCCs) methods. It's proven that the lower MFCCs have a big relationship to the aspect of timbre. Then, this method discards the part of the signal that is closely related to timbre (MFCC). These vectors are named as CRP (Chroma DCT-Reduced log Pitch) features.

Chroma Energy Normalized Statistics [52]. It is a chromagram based on short-time statistics over energy distributions bands, that creates a high level of abstraction about chromagram. CENS (Chroma Energy Normalized Statistics) constitute a family of scalable and robust audio features which have first been introduced in [53]. These features strongly correlate to the short-time harmonic content of the underlying audio signal, and they are very robust to dynamics, timbre, articulation, execution of note groups, and temporal micro-deviations. Moreover, this type of chromagram is one of the most efficient on time cost.

Beat synchronous chromagram [1]. Another alternative of chromagram that involves the use of centroid calculations to localise peaks on the time-frequency surface of spectrally sparse signals, as it is explained in [1], providing improved amplitude and frequency estimates. This refined method estimates an accurate computation of the associated chroma values.

2.4.3 Tonal Spaces

Several tonal pitch spaces have been presented in the literature since the Euler *tonnetz* [23]. These tonal spaces allow us to measure distances between sets of pitches. This distance is calculated based on how proximate the sets of pitches are perceived, according to Western musical tradition. Pitches, chords, and regions (or keys) are other models to describe the relationship between chords that are used as descriptors too.

Most are based on *Tonnetz*, the historical inspiration for all of them. It is a conceptual lattice diagram representing tonal space first described by Leonhard Euler in 1739 [23]. As shown in Figure 2.9, with the *Tonnetz* graph is suitable to represent most of the chord types, modelling over his space relationships (intervals) between different pitches.

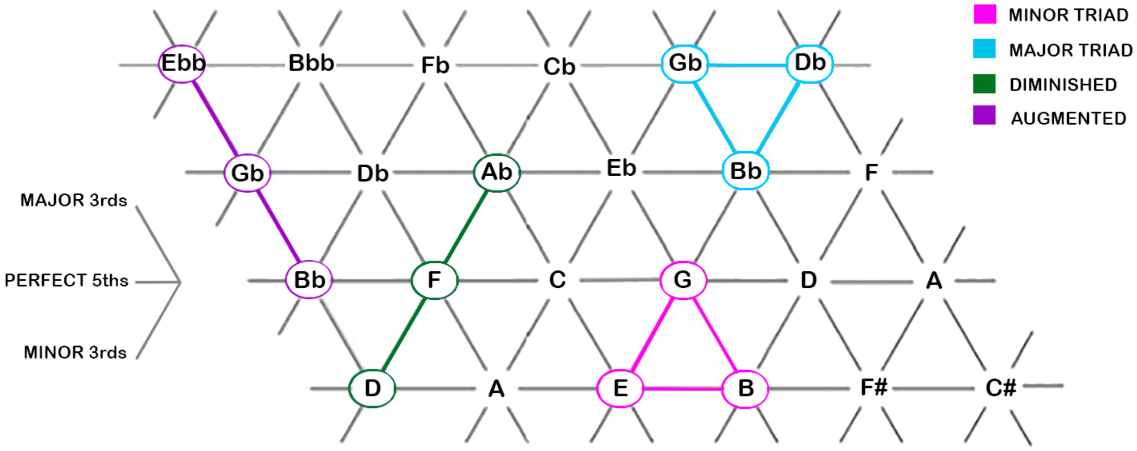


Figure 2.9: *Traditional* Tonnetz *representation.*

Tonnetz is a graph model that denotes, according to the classic Western harmony, the proximity between notes. It allows visualizing most of the chords in a geometrical way. The farther away one is in the graph, the more perceptually it is according to the classical harmony. The horizontal lines denote perfect fifths, the down right diagonals denote major thirds and up right diagonals, minor thirds, as shown in e 2.9.

Later, other more complex structures that modelled more characteristics of harmony were proposed. One of them is Shephard Helix [77] model, shown in Figure 2.10. Subsequently, models explained in the next sections are inspired by Sephard helix.

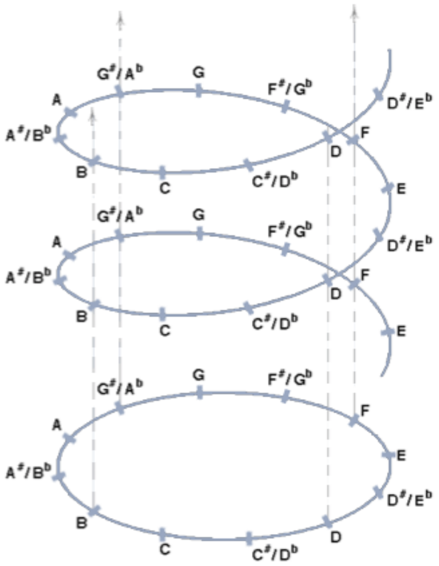


Figure 2.10: *Shepard Helix representation.*

Chew Spiral Array.

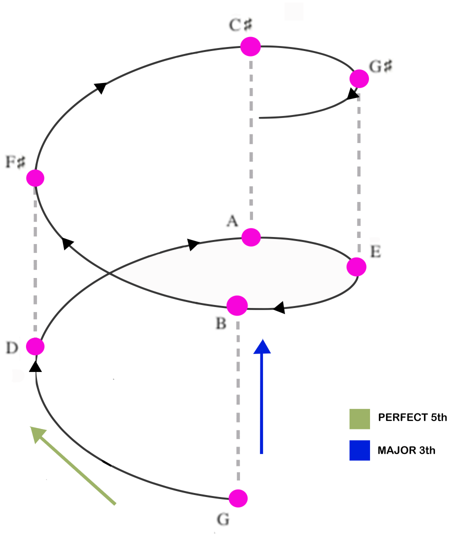


Figure 2.11: *Spiral Array. A mathematical model for modelling harmony.*

The spiral array model [19], shown in Figure 2.11, can be viewed as a generalized *Tonnetz*, which map pitches into a two-dimensional lattice (array) structure, pitches are mapped into a continuous spiral as is shown in Figure 2.11. The spiral array wraps up the two-dimensional Tonnetz into a three-dimensional lattice, and abstract concepts can be modeled such as chords and keys in the interior of the lattice space. This allows us to abstract high and low structures from the spiral array very useful for music analysis.

For example, it is possible to model the distance between two chords in a given key. Both are represented inside the spiral array space. Moreover, to preserve musically $F\# \neq Gb$ in, the spiral array does not assume enharmonic equivalence, i.e. it does not fold into an only torus its representation is like three overlapping hypertorus. There are spatial relationships between pitches, between chords, and between keys. In Figure 2.12 is shown how is modelled the relationship between pitches of 5^{th} perfect, 3^{th} major and 3^{th} minor.

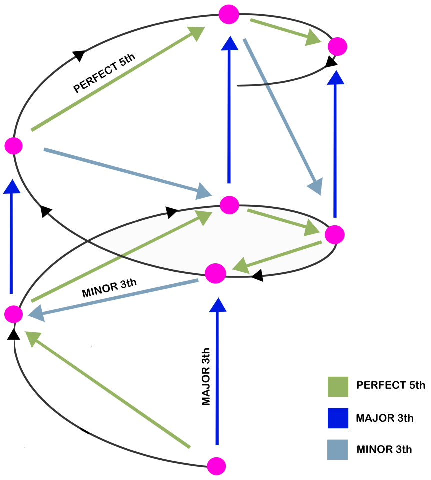


Figure 2.12: *Different intervals that can be measured in Elaine Chew Spiral Array.*

Tonal Centroid Space.

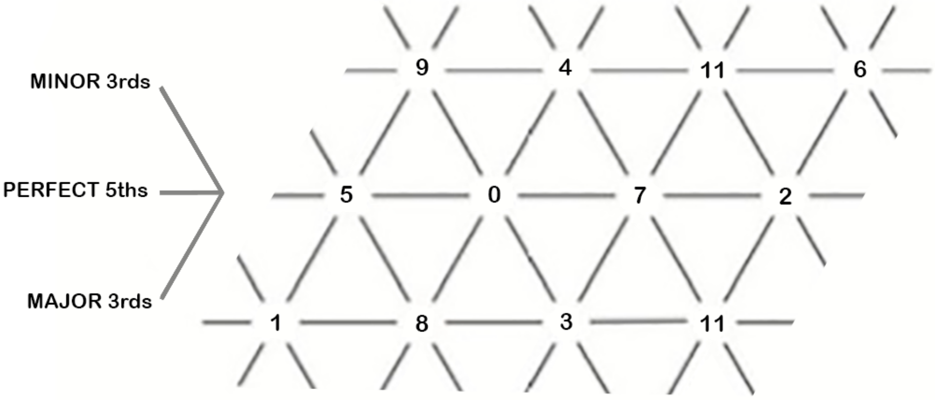


Figure 2.13: *Generalized* Tonnetz *Plane.*

Proposed by Harte et al. [31]. It introduces Enharmonic and Octave Equivalence, and it reduces the set of all notes to 12 pitch classes. Tonal Centroid Space can be represented as wrap the generalized *Tonnetz* plane, Figure 2.13, into a hypertorus, Figure 2.14.

$$\mathrm{TC}(d) = \frac{1}{\sum_b |C_f(b)|} \sum_{b=0}^{\beta-1} \Phi(d,b) C_f(b)$$

$$\Phi = [\phi_0, \phi_1, \ldots, \phi_{\beta-1}]$$

$$\phi_b = \begin{bmatrix} r_1 \sin(b\frac{7\pi}{6}) \\ r_1 \cos(b\frac{7\pi}{6}) \\ r_2 \sin(b\frac{3\pi}{2}) \\ r_2 \cos(b\frac{3\pi}{2}) \\ r_3 \sin(b\frac{2\pi}{3}) \\ r_3 \cos(b\frac{2\pi}{3}) \end{bmatrix}$$

The 6D interior space of the hypertorus, Figure 2.14, can be seen as three 2D circles: of fifths, major thirds and minor thirds. Chords can be described by their 6D centroids in this space [31].

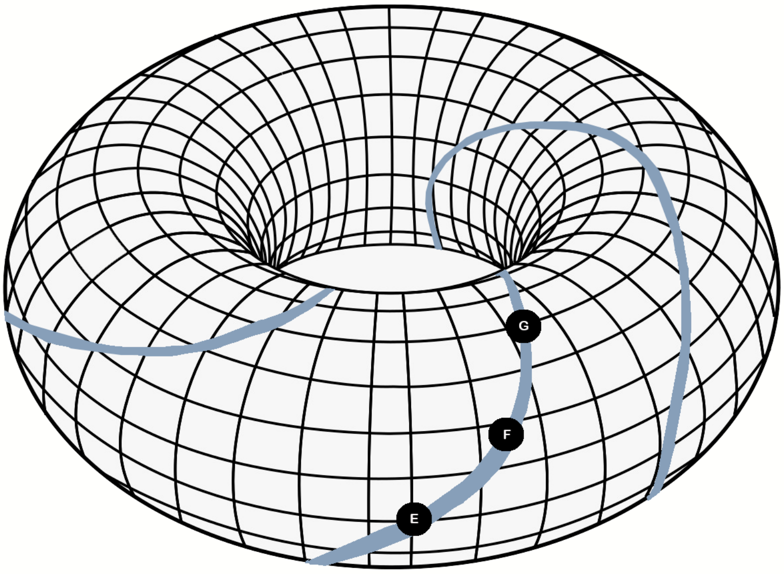


Figure 2.14: *Generalzied* Tonnetz *modelled as a toroid*

Tonal Interval Space.

Tonal Interval Space (TIS) proposed by Bernardes et al. [4] is a 12-dimensional tonal space in the context of the *Tonnetz* [23], Chew's Spiral Array [19], and Harte's Tonal Centroid Space [31].

The proposed Tonal Interval Space is calculated as the weighted Discrete Fourier Transform, as is shown in Equation 2.1 of normalized 12-element chroma vectors, which are representing as six circles covering the set of all possible pitch intervals in the chroma space, as is shown in Figure 2.15. Each one of the six circles has been interpreted from a musical perspective by Gilberto et Al. [4]. The contribution of each DFT coefficient (the circles in the visualization in x) is weighted

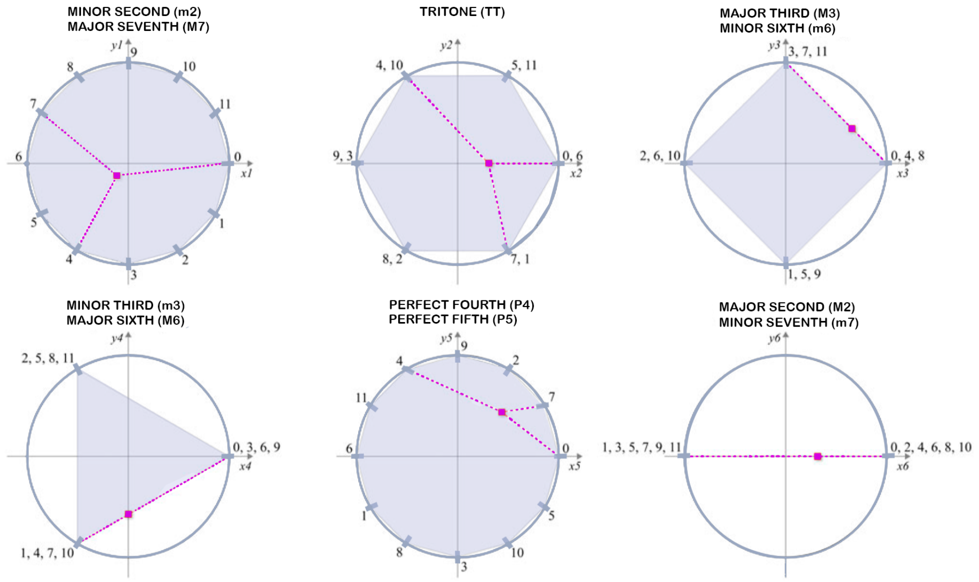


Figure 2.15: *Symbolic C major chord in all DFT components of TIV. Image edited from Gilberto Bernardes website.*

according to an empirical rating of consonance. By weighting the contribution of each circle (and hence pitch interval) independently, we can create a space in which angular (i.e., cosine) and Euclidean distances among pitches, chords, and regions concur with music theory principles [4, 61]. For example, TIV allows transposing a chord, as is shown in Figure fig:circulos2.

TIV 12-D space expands *Tonnetz* [23] by building a large range of pitch configurations beyond major and minor triads. Moreover, TIV expands Chew's Spiral Array [19] because the data of the space is more flexible in the sense that it allows the codification of any sonority that can be represented as a chroma vector although subject to enharmonic equivalence. Last but not least, TIV expands to Harte's Tonal Centroid Space [31] for including all possible intervals on one octave.

Mathematically, multi-level pitch configuration are represented by Tonal Interval Vectors (TIVs) T(k) in the Tonal Interval Space, chroma vector c(n) is computed by a DFT as follows:

$$T(k) = w(k) \sum_{n=0}^{N-1} \bar{C}(n) e^{\frac{-j2\pi kn}{N}}, \quad k \in Z \quad \text{with} \quad \bar{C}(n) = \frac{C(n)}{\sum_{m=0}^{N-1} C(m)} \tag{2.1}$$

where $N = 12$ is the dimension of the chroma vector; $w_{\star}(k)$ are weights derived empirically; $1 \leq k \leq 6$ is adopted by discarding the symmetrical components in $T(k)$; $T(k)$ uses $\bar{C}(n)$ which is $C(n)$ normalised by the DC component to allow the representation and comparison of different hierarchical levels of tonal pitch [4]. From the point of view of Fourier analysis, $T(k)$ is interpreted as a sequence of complex numbers, which we can visualize in Figure 1 as six circles, each corresponding to a complex conjugate. A musical interpretation relates each DFT component to complementary interval dyads within an octave (m2/M7 has the real part of $T(1)$ on the x-axis and

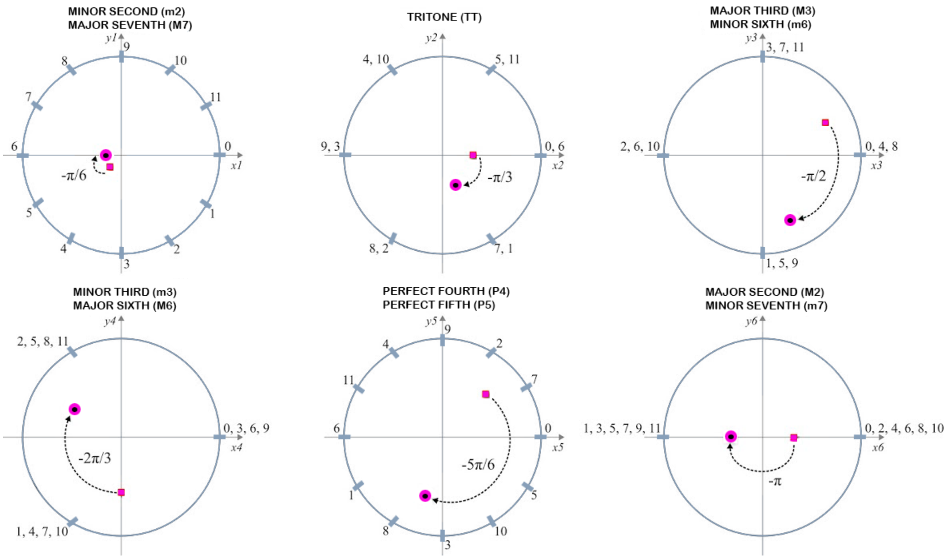


Figure 2.16: *Incrementing one semitone of Symbolic C major chord. Visualization of all DFT components of TIV. Image edited from Gilberto Bernardes website.*

the imaginary part of $T(1)$ on the y-axis and so on). The musical interpretation assigned to each coefficient corresponds to the musical interval that is furthest from the origin of the plane. The integers around each circle represent $0 \leq n \leq N1$ for $N = 12$, corresponding to the positions in the chroma vector $C(n)$.

Tonal Interval Space allows us to calculate two main metrics. The first metric explains the interval relationship between pitch configurations from the Western tonal music perspective theory principles by the cosine and Euclidean distances. The second one and most innovative aspect of the Tonal Interval Space is the possibility to compute a descriptor of tonal consonance from euclidean distance.

2.5 Harmonic Change Detection

The detection of harmonic changes from musical audio is a recognized task within Music Information Retrieval (MIR). It is important in the evolution of chord recognition systems as a preprocessing segmentation stage of the musical audio to detect chord boundaries [31, 28, 21]. The reason for such endeavour is that structurally -aware feature analysis has shown to improve accuracy when compared to the adoption of using equal -size frames [2].

To date, and to the best of our knowledge, Harte et al.'s [31] Harmonic Change Detection Function (HCDF) algorithm and its extension by [21] is recognized as state of the art. It includes an important harmonic representation for the tonal pitch which distorts the typical 12-element chroma representation, denoting the energy of each of the chromatic notes. In the resulting model, tonal pitch relations are captured as distances. The smaller the distances, the more related they are considered. The HCDF results from the temporal evolution of these distances. For a given time frame n the HCDF is computed from the Euclidean distance between neighbouring time frames

$n-1$ and $n+1$, where n is the frame index from a sliding analysis window across musical audio. Low values on the HCDF denote no substantial harmonic changes between consecutive frames. On the contrary, high peaks values (i.e. local maxima) in the HCDF denote harmonic changes. What exactly a harmonic change is depends on the context. For example, a harmonic change can be a note change, a chord change, or a modulation in the musical key.

Recognizing the diversity of models and the importance of the chromagram representation and the distortion introduced by the tonal model, as shown in [21], we advance new strategies for the computation of these blocks in the HCDF. Namely, we inspect recently proposed chromagrams – extensively reviewed in [42] – and the Tonal Interval Space [4, 6], which has been shown to expand Harte et al.'s [31] tonal pith distortion model with pitch class elements beyond 5th and 3rds. The tonal interval space adjusts weights of each interval to better match perceptual rankings of the tonal pitch at various levels – i.e., pitch, chords and keys. We inspect a large number of parameters in all component blocks of the HCDF and their interaction thereof on a large set of genre-specific datasets, which not only minimize overfitting problems but also inform about the generality of the method for musical expressions.

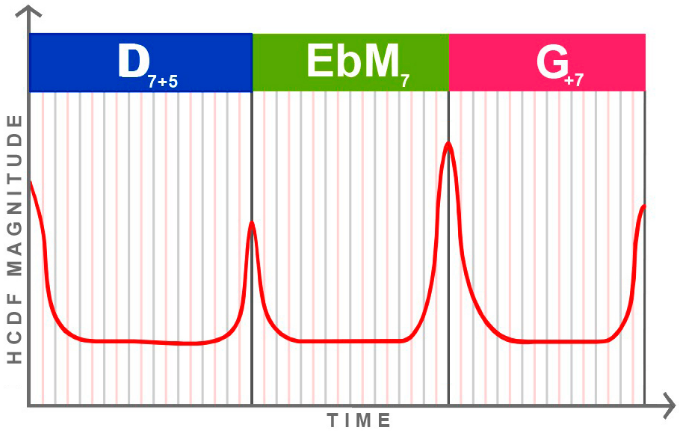


Figure 2.17: *Illustrative example of an "ideal" HCDF.*

2.5.1 Harmonic Change Detection Function in Musical Audio

The early research on harmonic change detection assumed the use of a likelihood function to evaluate the probability of the data generated by each of the annotated songs and to apply Bayes theorem to incorporate any prior knowledge. In this context, Hainsworth et al. [28] present a method for detecting musical change points which are mainly harmonic and more robust to transients. Roughly, the proposed method is able to detect half of the harmonic changes from musical audio with low computational cost and algorithmic complexity.

The algorithm proposed by Hainsworth et al. [28] consists in band-wise processing method to extract harmonic onsets from different bands. These different bands try to represent bass, low-mid and high-mid frequencies, with defined bands on 30-300Hz, 300Hz-1kHz and 1-5kHz . This

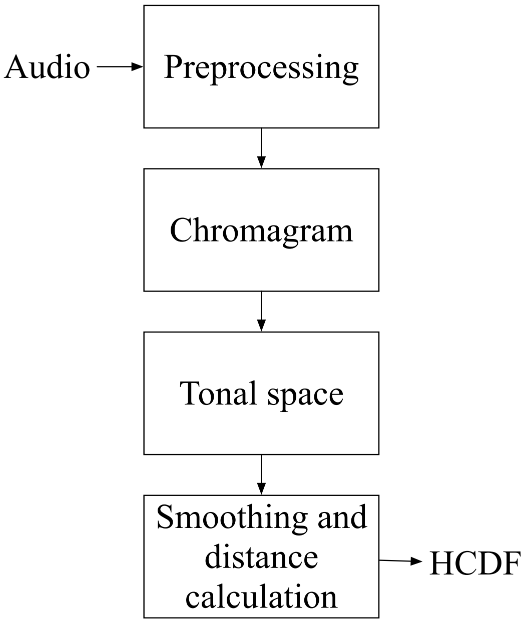


Figure 2.18: *Processing blocks of the HCDF.*

research assumes that above 5 kHz harmonic content is not interesting generally.

2.5.2 Detecting harmonic change in musical audio

The Harmonic Change Detection Function of Harte et al. [31] aims to extract chord boundaries from musical audio, taking advantage of the fact that distances can be measured between mapped audios on the model proposed in Harte et al's paper. The pitch space model projects audio information as Tonal Centroid Space cited above 2.4.3. Then it is theoretically possible to measure the distance betweeen tonal centroids, if the distance is wider enough then a harmonic chage has occured.

Figure 2.18 shows the four processing blocks of the Harte et Al. HCDF and the data flux between them. Harte et al. [31] HCDF *preprocessing* block includes the spectral analysis of 8192 sample windows ($\approx 743msec$) with a 50% overlap from a mono musical audio input at 11025 Hz sample rate. Constant-Q spectral analysis is adopted with 36 bins-per-octave across the 110-3520 Hz frequency range.

A 12-element *chromagram* is then extracted with the method described in [30]. Chromagrams are then mapped to a tonal space, to enhance the perceptual representation of the harmonic content as distances. The HCDF at a given time frame n is then computed using the Euclidean distance between the time frame $n-1$ and $n+1$. Smoothed by a 17-element Gaussian function with $5 \leq \sigma \leq 20$. Finally, as shown in Figure 2.17 peaks in the HCDF are considered harmonic changes.

2.5.3 Harmonic Change Detection for Musical Chords Segmentation

Degani et al. research [21] shows that parameterizations of different audio features with new methods proposed in the last years have a significant performance impact. In particular, in this research cosine distance, Non-Negative Least-Squares C_{NNLS} chromagram and Chroma DCT-Reduced log Pitch are tested among Harte et al. first pipeline [31]. This research outputs very good results. However, thresholding is used with a constant value over the HCDF generated by a little set of annotated songs. If Degani et al. HCDF is tested on other types of songs then the model does not generalize well, performing worse than Harte et al.'s HCDF.

Chapter 3

Revisiting Harmonic Change Detection

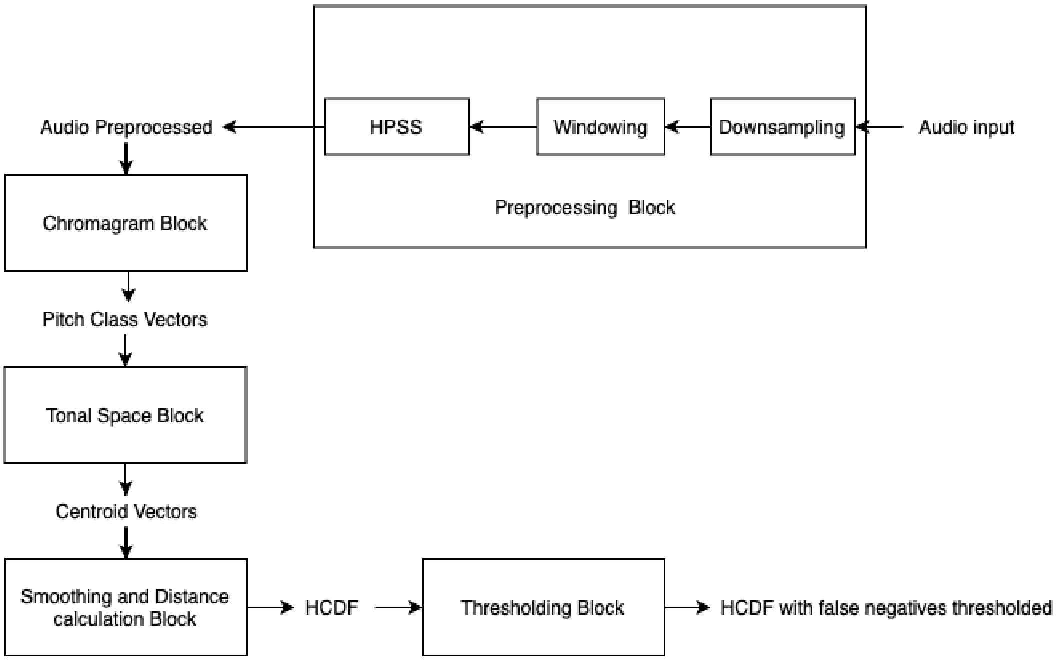


Figure 3.1: *HCDF diagram.*

In this Chapter, we revisit each of the component blocks of the HCDF (Figure 3.1) as defined by Harte et al. [31] in light of the recent advances in harmonic-content description and transformation, as well as their related signal processing methods. A comprehensive list of different algorithms per block is considered in each of the sections of this chapter.

3.1 Preprocessing

The preprocessing stage in the HCDF is responsible for creating a spectral representation from a time-domain audio signal (i.e., audio waveform). The audio under consideration can be either a single song in a digital audio format, such as a .WAV or .AIFF, or can be a collection of multiple audio files. In this section, we detail the spectral analysis and its parameterization (e.g., windowing, overlap) as well as a filtering processing stage which aims to enhance the harmonic content in the digital representation using the Harmonic-Percussive Source Separation (HPSS) [24]. Figure 3.2 shows the modular algorithmic structure of the preprocessing module.

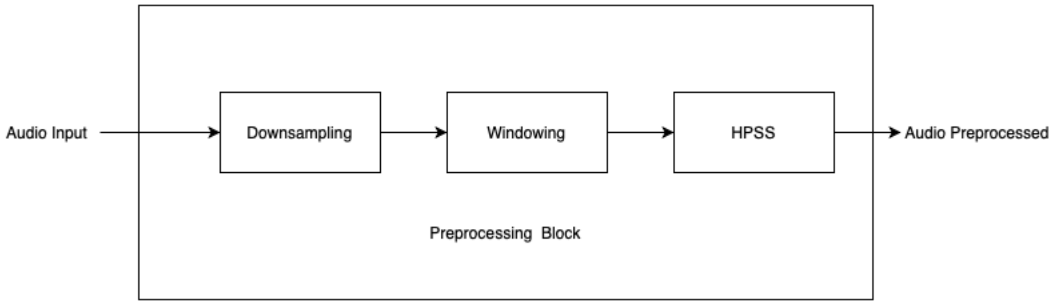


Figure 3.2: *Preprocessing block diagram.*

From the audio waveform representation, spectral analysis is typically done by applying the fast Fourier transform [55], and in some particular cases, notably when required by the chromagram computation, Constant-Q spectral analysis can be adopted. The Constant-Q transformation is better adapted to human perception as its frequency representation is logarithmic. Therefore, the lower notes are accumulated in a few frequencies while the higher the frequency the less the notes are accumulated. This means that if the same resolution is used in the low frequencies the resolution is poor and in the high frequencies it is excessive. Since the output of the transformation is effectively amplitude/phase versus logarithmic frequency, fewer frequency intervals are required to cover a given range effectively, and this is useful when frequencies span several octaves. In Figure 3.3 one can see the differences from a visual perspective the differences between FFT and Constant-Q transform.

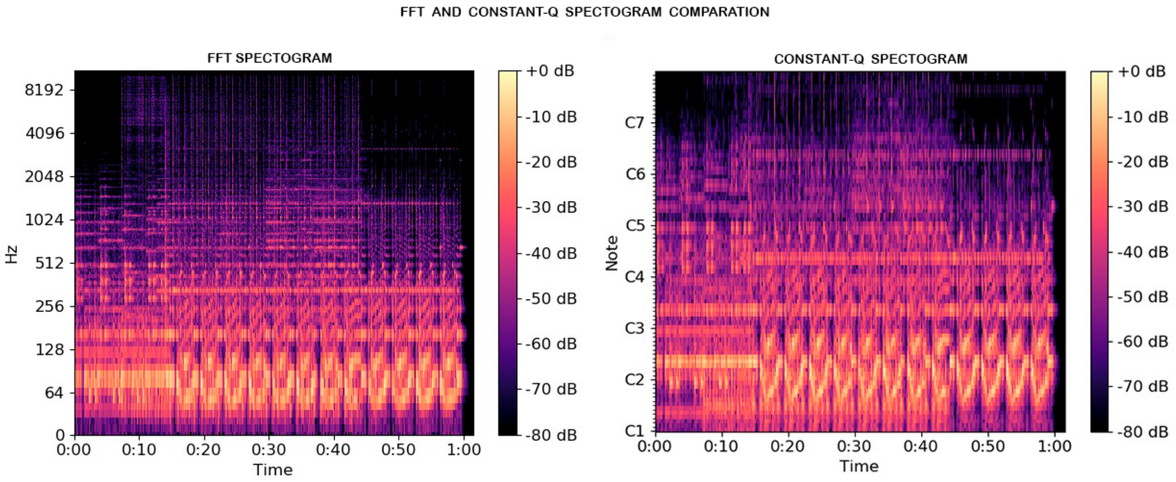


Figure 3.3: *FFT (left) spectrogram and Constant-Q transform (right) spectrogram. Computed from librosa [47] trumpet audio example.*

The range of human hearing covers approximately ten octaves from 20 Hz to about 20 kHz. Therefore in music processing only this range of data has to be analyzed. The transformation exhibits a reduction in the frequency resolution of higher frequency bins, which makes sense from an auditory perspective. As we have alredy mentioned, the Constant-Q transformation has a higher resolution at lower frequencies, which also comes closer to human perception. For the lowest notes of a piano (about 30 Hz), a semitone is a difference of about 1.5 Hz, while for the highest notes of the piano, at the 5 kHz range, a semitone can be a difference of about 200 Hz. Therefore, for musical data the exponential frequency resolution of the Constant Q transformation has advantages among the FFT.

In addition, the different tones structure a characteristic pattern in the Constant Q transform. Assuming the same relative power of each harmonic, if the F0 changes, the relative position is constant. This can help to segment instruments but also to find the fundamental sequences.

In relation to the Fourier transform, the implementation of this transformation is more complicated. This is because each interval requires a certain sample rate, so the window function is adaptive and has to vary according to the interval. Also, since the scale is logarithmic there is no zero/DC frequency, and this can be a problem in certain cases.

Transient noise [80] is one of the biggest problems [29] in HCDF algorithms [28, 31]. This produces sudden changes in the HCDF, as Harte mentioned on his thesis [29]. A signal is said to have a transient noise when its Fourier expansion requires an infinite number of sinusoids [80]. On the other hand, any signal expressible as a finite number of sinusoids is called as a steady-state signal. When there is waveform discontinuity there is a transient. However, in digital audio domain, to define transients is rather difficult. As stated in [80], one can ask which sounds should be "stretched" and which should be translated in time when a signal is "slowed down"? In the case of speech, for example, short consonants would be considered transients, while vowels and sibilants such as "ssss" would be considered steady-state signals. Percussive sounds and overtones are generally considered transients. More generally, almost any attack is considered a transient [80].

To this end, we consider the adoption of the Harmonic Percussive Source Separation (HPSS) algorithm using median filtering [24] to decompose an audio signal into two harmonic and percussive sources, from which we uniquely adopt the former for further processing. This step aims to exclude transient frequencies that have a negative impact on providing an optimal transcription of the audio signal's harmonic content. We adopt this algorithm after donwnsampling the musical audio and prior to the spectral audio analysis.

Different parameters for sample rate s, window size f, and overlap o, are considered in the preprocessing of the audio files. On the one hand, latter parameters have an impact on the efficiency of the system by changing the amount of audio sample data to be processed. Lower s, and higher values f and o result in more efficient HCDF computation. On the other hand, previous studies have shown that these parameters can have an influence on the outcomes of multiple tasks. Windowing and downsampling are techniques widely used [51] to simplify and standardize representations. They can be regarded as a bandwise lowpass filtering, which are used to attenuate fast fluctuations in the features representation. Downsampling is often used to effectively increase frequency resolution at lower frequencies, although one has to keep in mind that the maximum frequency analyzed will be half of the sample rate by Nyquist [56]. Often, these techniques are also used to reduce the cost of computing in exchange for the loss of information.

3.2 Chromagram

Chomagrams are pervasive across most harmonic-based audio-content description. They encompass multiple variants driven from a number of proposed algorithms, as we detailed in Section 2.4.2. In the context of our work on HCDF, we have selected four representative chromagrams: Constant-Q Transform chromagram C_{CQT}, Non-Negative Least-Squares chromagram C_{NNLS} [42], Harmonic Pitch Class Profile chromagram C_{HPCP} [27] and Short Time Fourier Transform chromagram C_{STFT} [22]. We have rejected Beat synchronous chromagram [1] because we could not find an open source implementation available. Chroma DCT-Reduced log Pitch [52] and Chroma Energy Normalized Statistics [52] have been rejected because they can be parameterized as the other chromagrams, due to them being implemented with a fix filter banks. The chromagrams implemented with deep learning have not been included in the study due to empirical character of the grid search, data generated in this dissertation is very large, and a song processed by deep learning takes about 160 seconds of computing time [89].

In chapter 2, a high level explanation is given of what the different chromagrams 2.4.2 are and what they are for, from a high level. In particular those used in this dissertation: C_{STFT}, C_{CQT}, C_{HPCP} and C_{NNLS}. Hence, in this section the working of chromagrams will be detailed from a low level perspective.

All the chromagram used in our research have some common parts. These chromagrams are implemented as a lineal digital signal processing pipeline. The input of this method is a digital audio that may have been preprocessed before 3.1. The output is a chromagram of 12 pitch classes where the different audio frequencies have been classified.

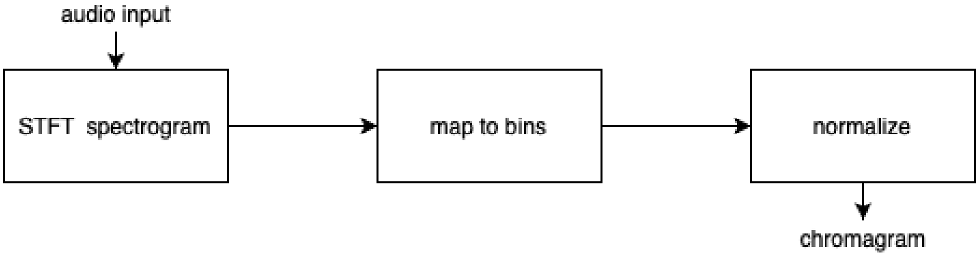


Figure 3.4: *Short Fourier Transform chromagram C_{STFT} pipeline diagram.*

The Short Time Fourier Transform chromagram C_{STFT} has been widely used in the last 20 years, in [25] a full explanation can be found. The lineal pipeline of this chromagram is shown in 3.4. The first block of this chromagram operates as a spectrogram, taking an audio input and generating a matrix of short-time spectrum frames. The second block uses immediate frequency estimates from the spectrogram to obtain the chroma profiles. Another option, with less computational complexity, is mapping each STFT bin directly to chroma classes, after selecting spectral peaks. In the last block, each class profile of the chromagram is normalized.

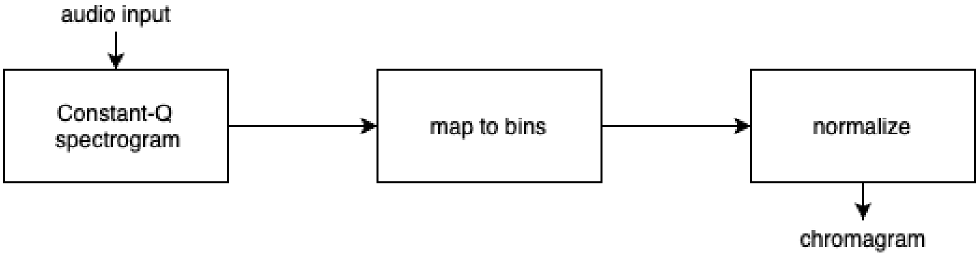


Figure 3.5: *Constant-Q Transform chromagram C_{CQT} pipeline diagram.*

The Constant-Q Transform chromagram C_{CQT} was proposed by Brown et al. [15]. The lineal pipeline of this chromagram is shown in 3.4 and is very similar to the previous chromagram, C_{STFT}. The first block computes a CQT spectrogram, in previous section 3.1 the difference between this type of transform and other types of Fourier transforms is explained and his musical advantages. This first block can be calculated by a full Constant-Q transform or by a hybrid one, with a lower computational cost. The output is also a matrix of Constant-Q spectrum frames. The second block is, as the previous chromagram, mapping the spectrogram into pitch class profiles. In the last block, as in the C_{STFT} pipeline, class profiles of the chromagram are normalized.

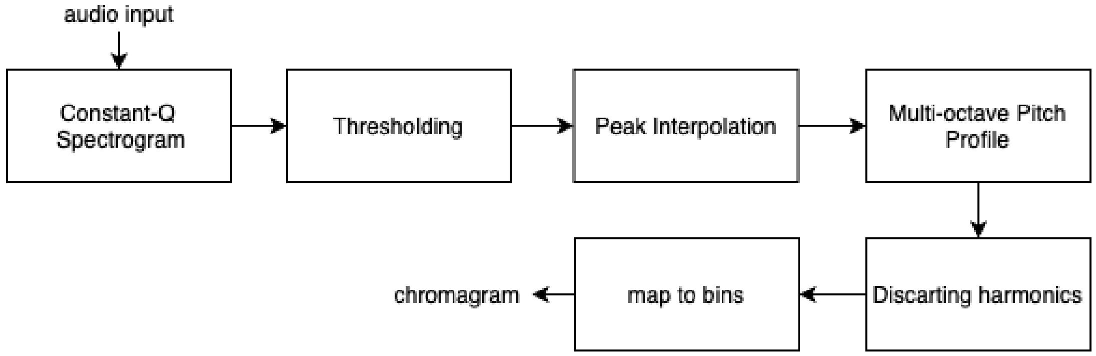


Figure 3.6: *Harmonic Pitch Class Profile chromagram C_{HPCP} pipeline diagram.*

The Harmonic Pitch Class Profile chromagram C_{HPCP} was proposed by Emilia Gomez et Al.[27], the pipeline of this chromagram is shown in Figure 3.6. On the first block the Constant-Q spectogram is computed from a tunning frecuency. Then, the second block removes redundant information, first discarding frequencies beyond a low pass threshold and a highpass threshold and later thresholding the spectrogram with a global and local (frame-wise) threshold. The third pipeline blocks to compute the peak interpolation to obtain spectral peaks with a high resolution. Then, in the fourth block, each of those peaks are assigned to the pitches of the octaves that were not discarded. In the fifth block, one tries to eliminate the energy redundancies produced by the harmonic partials. And finally, all the octaves are mapped in one octave, the Pitch Class Profiles.

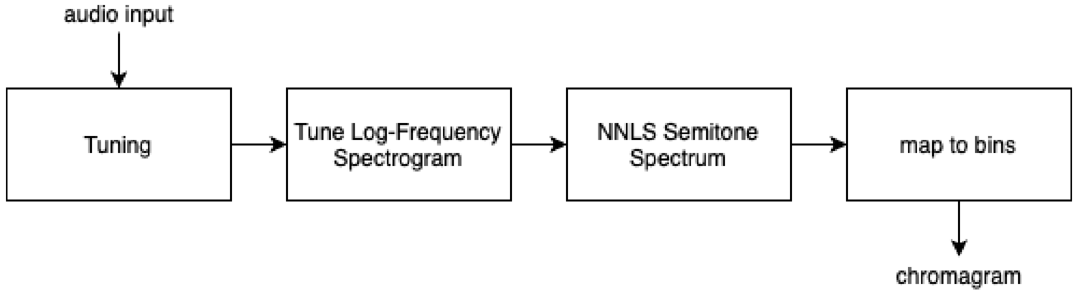


Figure 3.7: *Non-Negative Least-Squares chromagram C_{NNLS} pipeline diagram.*

The last chromagram is the Non-Negative Least-Squares chromagram C_{NNLS}. It was proposed by Mauch et al. [42] and on that research a full description of the pipeline shown in the Figure 3.7 can be found. The first block calculates tuning from NNLS, using the angle of the complex number defined by the cumulative mean of real and imaginary values. In the second block is computed the spectrogram, using the tuning estimated value, an approach to fundamental frequency for performing linear interpolation on the existing log-frequency spectrogram with all the partials. In the third block, a Semitone-spaced log-frequency spectrum derived from the tuned log-freq spectrum above is computed. The spectrum is inferred using a non-negative least squares algorithm. In the last block, three different kinds of chromagram are calculated, a treble chromagram in higher frequencies, a bass chromagram in lower frequencies, and a combination of both.

3.3 Tonal Space

Tonal Interval Vectors, $T(k)$, [4] extend the Tonal Space by Harte et al.'s [31] to the entire set of complementary intervals within the 12 chromatic notes of the equal tempered pitch class space (i.e. all complementary intervals resulting in the 12-elements chromagram representation space). The extended vector space projects the most salient pitch levels of tonal Western music – pitches, chords and keys – as unique locations in the space. The resulting spatial location of $T(k)$ ensures that perceptually-related pitch within the Western tonal music context correspond to small Euclidean distances [4].

Departing from a chromagram representation, C, we compute a 12-dimensional $T(k)$ as the L_1 normalized discrete Fourier transform (DFT), such that:

$$T(k) = w_\star(k) \sum_{n=0}^{N-1} \bar{C}(m) e^{\frac{-j2\pi km}{M}}, k \in Z \quad \text{with} \quad \bar{C}(m) = \frac{C(m)}{\sum_{n=0}^{M-1} C(m)} \tag{3.1}$$

where $M = 12$ is the dimension of the chromagram, C; k is set to $1 \leq k \leq 6$ since the remaining coefficients are symmetric; $T(k)$ uses $\bar{C}(m)$ which is $C(m)$ normalized by the DC component $T(0) = \sum_{n=0}^{M-1} C(m)$ to allow the representation and comparison of different hierarchical levels of tonal pitch [4]; and $w_\star(k)$ are weights which regulate the importance of each coefficient (or interpreted musical interval) in $T(k)$. Three sets of weights are adopted. $w_s(k) = \{2, 11, 17, 16, 19, 7\}$ was proposed for chromagrams driven from symbolic musical manifestations [4], $w_a(k) = \{3, 8, 11.5, 15, 14.5, 7.5\}$ was proposed for musical audio [7] and $w_h(k) = \{0, 0, 1, 0.5, 1, 0\}$ provides Harte et al.'s [31]. The two former set of weights result from empirical consonance ratings of dyads, used to adjust the contribution of each dimension k of the space (or interpreted musical interval), making it a perceptually relevant space in comparison to its non-weighted version [4].

3.4 Smoothing

To reduce the effects of transient frames and noise, the temporal sequence of $T(k)$ vectors are convoluted with a Gaussian smoothing function with $0 \leq \sigma \leq 20$ controlling the the standard deviation of the distribution, as proposed in [31]. The sequence of 12-D $T(k)$ vectors is convolved with a Gaussian on a row-by-row fashion.

3.5 Distance calculation

The HCDF, ξ, is then defined as the overall distance across the temporal sequence of the Gaussian smoothed $\widehat{T(k)}$ vectors. HCDF at frame ξ_n is computed as the distance between the smoothed vectors ξ_{n-1} and ξ_{n+1} (Equation 3.2). Euclidean distance ξ_n^{eucl} and cosine distance (i.e. the phase or angular distance) $\xi_n^{\cos}$ metrics are considered, where:

$$\xi_n^{\text{eucl}} = \widehat{\zeta}_{n+1} - \widehat{\zeta}_{n-1}, \qquad \xi_n^{\cos} = \frac{\langle \widehat{\zeta}_{n+1}, \widehat{\zeta}_{n+1} \rangle}{\widehat{\zeta}_{n+1}\widehat{\zeta}_{n+1}} \tag{3.2}$$

The Euclidean distance between $T(k)$ vectors denote parsimonious movements between sonorities, see [84, 66] for a comprehensive discussion on this topic. The cosine distance between the $T(k)$ is a good indicator of how well sonorities "mix" together. For example, it quantifies the degree of tonal proximity of $T(k)$ mixtures.

3.6 Adaptive Thresholding

Finally, we apply peak picking to the HCDF to identify transitions between regions that are harmonically stable, an approach inspired by Chew's key finding algorithm in detecting modulations [19].

Degani et al. [21] in revising Harte et al.'s [31] HCDF algorithm adopts a thresholding with a fixed value. However, it only performs well for the little dataset where they are training. They are training his model with only the 16 songs of the Beatles to perform that great result. Harte et al. [31] on the last mentioned paper section were already proposing that thresholdings with constant values did not work and that more complex models were needed.

Harte et al. [31] discuss the adoption of an adaptive thresholding whose parameterization would comply with the content of the musical audio. The reality is that some failures are structural, but others can be solved later. Training a typical fault detector or spurious detector should not be a complicated task if the distance generated by HCDF would contain sufficient information, although it is outside the objectives of this dissertation.

Chapter 4

Evaluation

Table 4.1: *Algorithms and parameters evaluated in the grid search method*

HPSS	{True, False}
Sample rate s	{8000, 11025, 22050, 44100}
Window size f	{1024, 2048, 4096, 8192, 16384}
Overlap o	$\{0\%, 50\%, 25\%, 12.5\%\}$
Chroma C	$\{C_{\mathrm{STFT}}, C_{\mathrm{CQT}}, C_{\mathrm{HPCP}}, C_{\mathrm{NNLS}}\}$
Tonal model $T(k)$	$\{w_s$, w_a, w_h, null $\}$
Gaussian filtering σ	{1, 3, 7, 9, 11, 13, 15, 17}
distance metric ξ	{Euclidean ξ_n^{eucl}, Cosine $\xi_n^{\cos}$}

We evaluate the HCDF ξ function in detetcting harmonic changes from audio using all combinations from the proposed algorithms for each component blocks, detailed in Section 3, as well as their parameterization and interaction. Table 4.1 lists all the algorithms and parameters under consideration. Ultimately, the evaluation ought to inform us how the different algorithms and their parameterization best perform in two different scenarios. On the one hand, we aim to optimize HCDF ξ to output the highest number of recognized harmonic changes, while minimizing the number of false positives. On the other hand, we equally aim to maximize the number of harmonic changes recognized, regardless of the increase in false positives. Ultimately, the evaluation ought to inform us how the different algorithms and their parameterization best perform in two different scenarios.

To minimize overfitting when finding the best set of algorithms and parameters in computing the HCDF ξ – mostly due to the use of a small set of audio examples for optimization [32] –, we adopt four style-specific datasets, thus expanding the previous evaluations [31, 21] that rely on a small set of 16 songs from a single band, the Beatles. In detail, we use the following four datasets: MTG-JAAH [73], Queen, Beatles and Zweieck [40], which include 113, 110, 20 and 15 songs, respectively; thus, a total of 258 songs are evaluated. The datasets cover four different genres –

jazz, rock, pop, and folk – , a broad range of timbres as a result of diversity in the instrumentation and a wide range of chord change rates as shown in Figure 4.1. These datasets have been adopted to evaluate MIR tasks, as there exist plenty of expert annotations for multiple musical structure elements, particularly including chord labels, which are used as the ground truth to contrast with the HCDF hits (i.e. peaks).

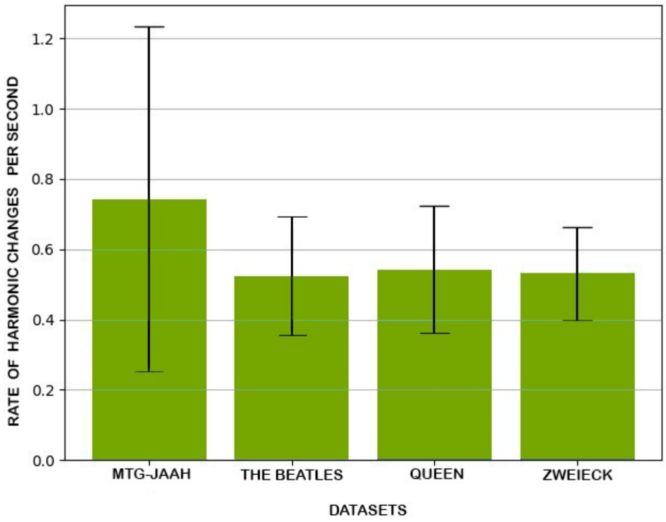


Figure 4.1: *Rate of harmonic changes per dataset. Standard deviation of the rate of harmonic changes is reported in the error bars.*

Due to the discrete nature of the parameters under study and their deterministic relationship, a grid search method has been adopted to inspect all possible combinations across the multiple proposed algorithms and parameterizations. Ultimately, we aim to find the set of algorithms and parameters that best match the ground truth annotations. We adopt dynamic programming for the efficient computation of the grid search method, since the HCDF ξ algorithms and parameterization under consideration share multiple processing blocks and data.

The evaluation of the quality of the hcdf has been carried out following previous studies [31], A harmonic change match is defined within ±3 frames ($\approx$ 278 ms) between the HCDF ξ predicted hits and the ground truth annotations. We adopt three performance measures as evaluation indicators: (1) *precision* P is the ratio of hits to (correctly) detected changes, (2) *recall* R is the ratio of hits to ground truth annotations of harmonic change changes, and (3) *f-score* F is the harmonic mean of precision and recall providing a balanced score for the task by combining the two previous indicators, such that:

$$P = \frac{\text{tp}}{\text{tp}+\text{fp}} \quad , \quad R = \frac{\text{tp}}{\text{tp}+\text{fn}} \quad , \quad F = \frac{2PR}{P+R} \tag{4.1}$$

where tp, fp, and fn are true positives, false positives and false negatives, respectively. A true positive is a hit in the HCDF ξ which matches a harmonic change; a false positive is a hit in the HCDF ξ which does not match a harmonic change in the musical audio, and a false negative is a harmonic change which has not been identified a hit in the HCDF ξ.

4.1 Datasets

Datasets are fundamental in any activity within the field of Machine Learning and Data Science, as well as in MIR. However, annotating songs, and in particular, chords, is a slow and costly process. On the other hand, without extensive datasets, it is challenging to reach generalizations without bias. In the field of harmonic recognition, proper chord notations are essential to generalize models. MIR and musicology communities have been cooperating for years to create datasets of audio recordings with annotated chord labels. These collections are fundamental for modelling harmony. Typically, annotations are openly available, yet the annotated musical audios is typically unaccessible due to copyright constraints.

Isophonics. The Isophonics is a collection of datasets first promoted by Christopher Harte in the course of his post-graduate studies at Queen Mary University of London[40] and followed by many other researchers from various research centres. The first dataset in this collection includes eight studio albums by The Beatles. The Beatles dataset seem quite appealing because of their diverse structure, recording techniques and chord progressions, while also being a central part of pop/rock culture. Moreover, the albums are readily available, and they are very well-known. Later, the corpus was extended to include music by Carole King, Queen, Michael Jackson, and Zweieck. However, chord annotations are limited to 113 songs for the Beatles, 7 songs by Carole king dataset, 20 songs by the Queens and 13 songs by Zweieck.

Isophonics family is available as ".lab" files. Each line of the chord annotations in the .lab files is divided into three columns: the first one is the instant of time when the chord begins in seconds, the second one is the instant of time when the chord ends and third one is the chord label. This format was proposed by Harte [29]. The format and dataset have been extensively used for modelling and evaluating chord recognition algorithms [13, 33, 58].

MTG-JAAH is a new dataset of time-aligned jazz harmony transcriptions [75].

4.2 Technical Architecture: Implementing an Efficient Grid-Search Method

An entire new evaluation system for computing the results of this dissertation has been developed. It has been designed to support the large amount of data required to test all the possible parametrizations.

We adopted a grid search to tune our HCDF proposal becouse for this study not only to get the best parametrization, the final objective is to understand how the different blocks of the algorithm relate to each other and to different genres. Therefore, it was decided to generate a large amount of data from the above relationships to be able to analyze them later. Moreover, in this case grid search is an optimal search, Exhaustive-search over specified parameter values for an estimator is an optimal search if the possible values are discrete. All the values that the different HCDF hyperparameters can take are discrete. There are none continuous. So this kind of brute-force search is possible. In this dissertation, all possible parameters, shown in the Table 4.1, are tested.

We adopted dynamic programming to solve the grid search over HCDF best parametrization optimization because it was possible for reducing computation time of the grid search. Therefore, to process the large amount of data that has to be processed, the following design decision has been made: any element of the grid search is computed at most one time. Moreover this algorithm have optimal substructure and Dynamic programming could be applied. Dynamic programming means simplifying a complicated problem by breaking it down into simpler sub-problems thanks to recursivity. The classic HCDF pipeline mimics this scenario. It has a solution tree in which all the children share the parent nodes of the solutions. For example, when the grid search is calculated with three different Tonal Spaces, the three possible target algorithms share the same chromagram. So, we don't have to calculate three times the same chromagram if it has been calculated before. The same occurs with all the blocks of the algorithm,

The evaluation system has been designed as a distributed among heterogeneous machines has been made to be able to process high amounts of data efficiently.

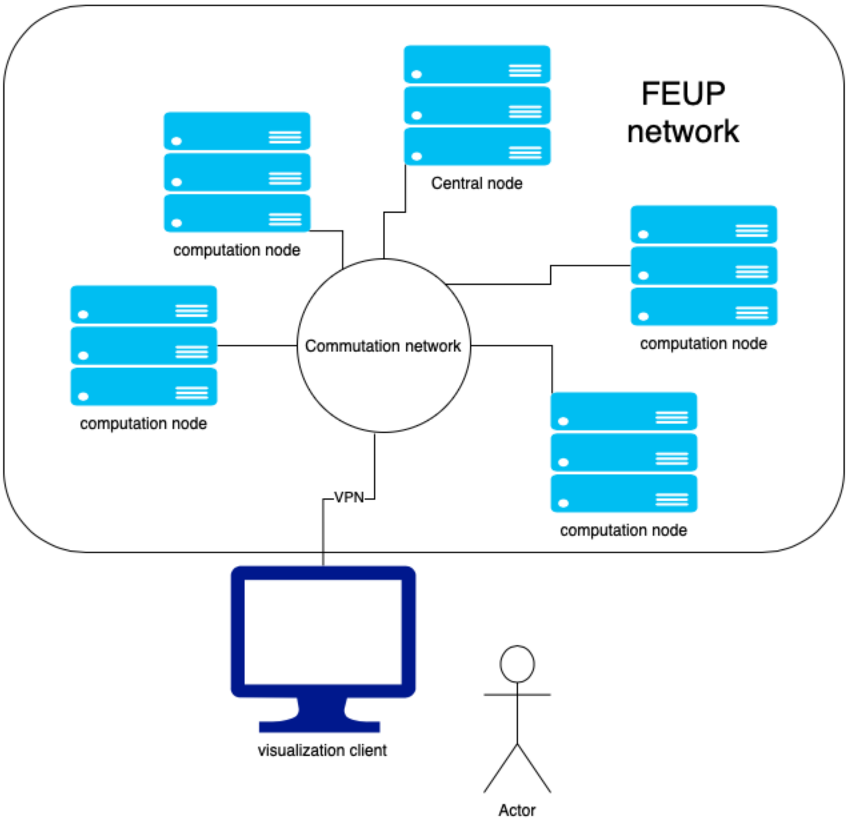


Figure 4.2: *Diagram of distributed architecture.*

The evaluation system has a distributed topology as is shown in this Figure 4.2. A central node, produces different parameterization tests to be processed across the distributed computation nodes. Each computation node stored processed data locally to run the dynamic programming grid search. In practice, the distributed system is very heterogeneous. It consists of six different computers, three desktop Linux and three iMacs, but it could be scaled to add more computational capacity.

Each computation node receives the parameterization of the algorithm and the song to be tested, performs the HCDF, evaluates the ground truth annotations and finally sends the results to the server. Each of the HCDF pipeline blocks produces partial results when executed with a given parameter setting. All partial results of each HCDF block computed are saved for following computations due to the nature of the dynamic programming.

The business layer of the computation nodes, where the HCDF is calculated, is developed in Python and Vamp. The HCDFs running in the computation nodes are the core component of the evaluation system, as they feature the algorithms described in Chapter 3. Downsampling, windowing, HPSS, the C_{CQT} and C_{STFT} chromagrams are computed with the Librosa library [47]. The C_{NNLS} and C_{HPCP} chromagrams are computed with Vamp Plugins [42, 27]. Unless stated, standard parameters were adopted.

Once a result is computed from one HCDF parametrization, it is sent to the central node to be stored. Last but not least, all the data generated by the system can be analyzed and visualized in the visualization client.

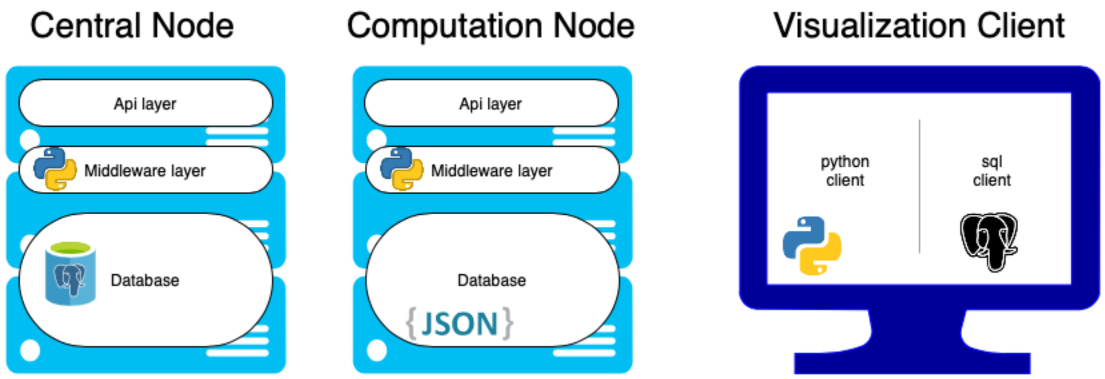


Figure 4.3: *Diagram of layers about each node.*

As shown in Figure 4.3, each type of node has a simple architecture. Both the Central Node and the Computation Nodes have an API communication layer that allows networking with other nodes and a middleware made on Python. The Visualization Node is built partially on SQL; and partially, in Python and SQL as a database interface.

Data processing and collection is fundamental to the evaluation system. All the values of the grid search are saved in the binary pickle format. In each computational node, the evaluation metrics for the HCDF are saved in a human-readable format, namely using JSON format.

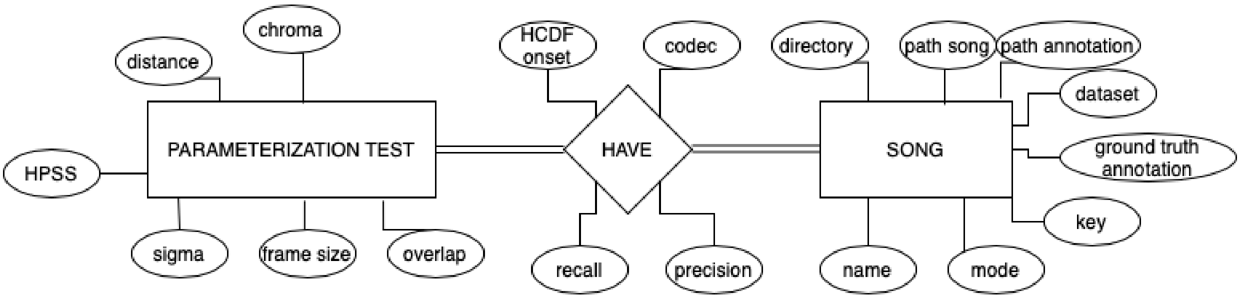
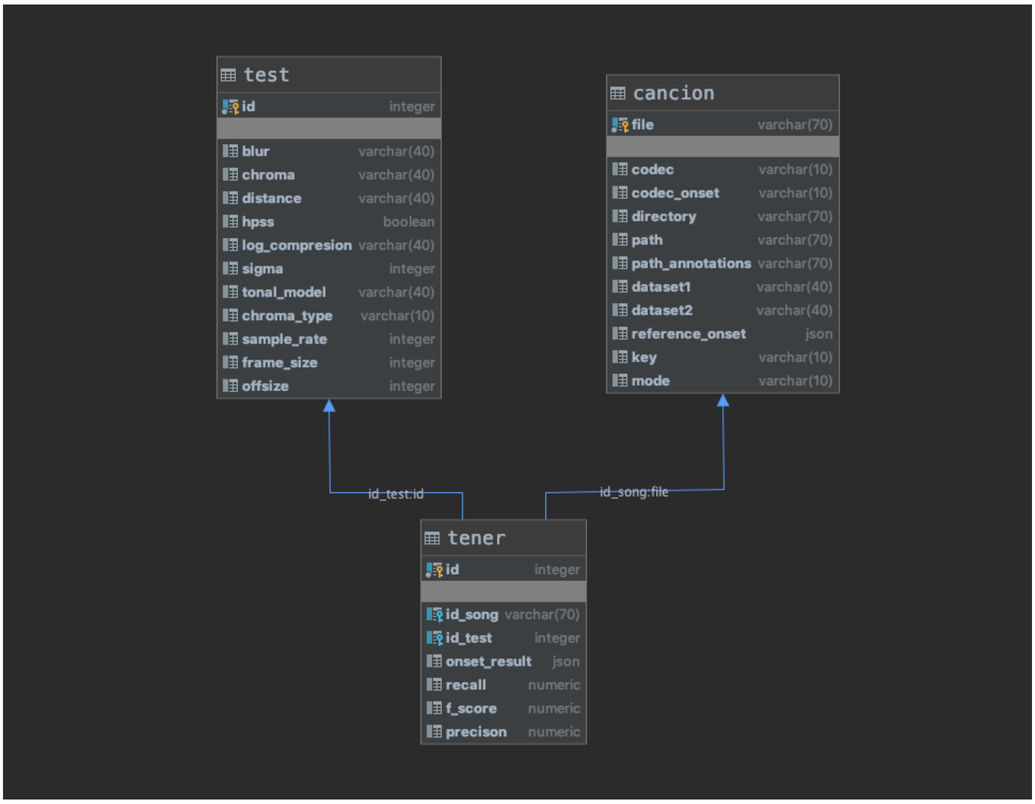


Figure 4.4: *Database entity/relationship model.*

All the data is sent to a Postgre SQL database, where it is stored in a structured manner in a simple database schema, as seen in the entity-relationship model (Figure 4.4).



test	
id	integer
blur	varchar(40)
chroma	varchar(40)
distance	varchar(40)
hpss	boolean
log_compresion	varchar(40)
sigma	integer
tonal_model	varchar(40)
chroma_type	varchar(10)
sample_rate	integer
frame_size	integer
offsize	integer

cancion	
file	varchar(70)
codec	varchar(10)
codec_onset	varchar(10)
directory	varchar(70)
path	varchar(70)
path_annotations	varchar(70)
dataset1	varchar(40)
dataset2	varchar(40)
reference_onset	json
key	varchar(10)
mode	varchar(10)

tener	
id	integer
id_song	varchar(70)
id_test	integer
onset_result	json
recall	numeric
f_score	numeric
precison	numeric

Figure 4.5: *Final database diagram.*

Final HCDF dataset is available at: `https://drive.google.com/drive/folders/1a-SzgmqPf7DmSnNaXAZM8b1MuBv1Id1A?usp=sharing`. It is compound by a README.md where its structure is explained and a directory file where all the data is available. Each file of data includes the parametrization results for each tested song. To easily parse the data, each file is named descriptively. The name file adopted is composed by the parametrization testes. On the file name, each parameter of the parameterization is separated by th character ".", with the order: HPSS, tonal model, chroma, sample rate, window size, overlap, blur and distance. For example, in Code 4.2 the file name would be hcdf.8000.1024.0.hpcp.T(w h).sigma11.euclidean.json. Inside the JSON file, the first eight entries are parameters with possible values defined in Table 4.1 with the name: sample rate, window size, overlap, HPSS, tonal model, chroma, σ smoothing filter, and distance. The JSON file includes the parametrization values adopted, followed by a list of songs with their respective HCDF evaluation metrics (i.e., f-score, recall, precision and the harmonic changes). Total size of the results amount to 128 GB.

```
{
    "hpss": false,
    "tonal_model": "T(w_h)",
    "chroma": "hpcp",
    "sample_rate": 8000,
    "window_size": "1024",
    "overlap": 0,
    "blur": "full",
    "distance": 11,
    "results": [
        {
            "song": "01_\\-_A_Hard_Day's_Night",
            "harmonic_change": [
                0.0,
                1.1542631509874295,
                ...
                144.57145966117554,
                146.68760877131916
            ],
            "f_score": 0.524390243902439,
            "precision": 0.6825396825396826,
            "recall": 0.42574257425742573
        },
        {
            "song": "01 A Kind Of Magic",
            "harmonic_change": [
                0.0,
                0.37655172413793103,
                ...
                250.21862068965518,
                253.13689655172413,
                257.93793103448274
            ],
            "f_score": 0.33136094674556216,
            "precision": 0.2978723404255319,
            "recall": 0.37333333333333335
        },
        ...
    ]
}
```

Visualisation scripts.

Above all, diverse functions have been implemented to find the different relationships between the hyperparameters and between these hyperparameters and specific data, such as the mode or key of a song. Various programs have also been built to be able to visualize either the HCDF or several HCDFs at the same time and be able to compare them.

Different scripts have been implemented, to perform the functions explained in the following paragraph. To measure the rate of harmonic change in all the different songs analysed and create various visualisualizations from the data of rate of harmonic change. To set thresholds to the HCDF to overfit it as done in previous articles or to test adaptive thresholding ideas. Or to fix various bugs that we had during the process.

Support scripts.

Scripts have been made to populate the database with data provided by the dataset and administer it later. Tools have been implemented to save backups of the final data or to monitor how much data was missing. Scripts have also been created to monitor if any test had not been performed correctly.

The distributed system has been maintained through scripting and monitoring. To be able to access each of the servers. To monitor their space availability, the lab network status and the system status. To reboot the system or just a specific node. Or to free up memory.

4.3 Results

The complete set of results for the grid search method can be accessed online at: https://drive.google.com/drive/folders/1a-SzgmqPf7DmSnNaXAZM8b1MuBv1Id1A?usp=sharing. In this section, we present and discuss the results for the algorithms and parameters which exhibit higher f-score and recall, as they play a prominent role in supporting applications within content-based digital audio processing (e.g. analysis, retrieval, and transformation), in detriment of higher precision. Higher f-measure provides a balanced prediction of chord boundaries, relevant to creative applications as harmonization [38], adaptive digital audio effects [61] or generative visuals from music [14]. On the other hand, higher recall guarantees an excellent

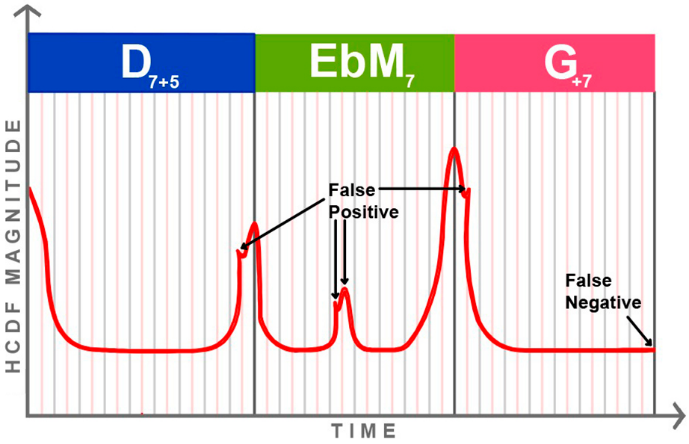


Figure 4.6: *Typical false positives caused by bass and percussive sounds.*

resolution as a preprocessing segmentation stage for tasks such as automatic chord recognition (ACR) [58, 88].

Table 4.2 shows the F, P and R evaluation metrics for the overall collection under consideration and for each dataset. These metrics are shown for the best f-score and recall results arising from the optimal combination of algorithms and parameters in the grid search method. For comparison purposes, we include in Table 4.2 the results from the Harte et al. 's method as presented in [31], which we evaluated in the grid search method in terms of best the Gaussian filtering σ parameter only, as it remained open in the original contribution.

The total number of combinations across all algorithms and parameters conditions adopted in each instance of the grid search method is 8890560. The best f-score results adopts a sample rate of $s = 8000$ Hz, a window size of $f = 1024$ and a window overlap of $o = 50\%$. Musical audio is preprocessed using HPSS. Harmonic content representation results from C_{NNLS} chromagram further encoded and projected on the tonal model $T(k)$ with w_a. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 5$. The best recall results adopts a sample rate of $s = 44.100$ Hz, a window size of $f = 2048$ and a $o = 25\%$ overlap. Processed harmonic content representation results from the harmonic source only from the HPSS algorithm, which is further encoded as an C_{STFT} chromagram and projected on the tonal model $T(k)$ with w_a. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 17$.

The best f-score results in the entire collection of four datasets improve by $5,57\%$ in comparison with the previous Harte et al. [31] method, with noticeable $6,28\%$ gains in terms of recall. The individual dataset results show considerable improvements in the Queen and The Beatles datasets, suggesting that in lower rates of harmonic change are simpler to achieve in this type of genres. If MTG-JAAH dataset is not taken into account in the scenario of the computing the best f-score, our method improves by 8.29% against Harte et al. method.

The adoption of the tonal model $T(k)$ with w_a has shown to improve the results across all

Table 4.2: *HCDF evaluation indicators for the best f-score and recall methods across the four datasets understudy, to which we compare with previous methods. The average is computed over all pieces.*

	Best f-score Proposed			Best Recall Proposed		
Dataset title	F	P	R	F	P	R
MTG-JAAH	63,3%	62,7%	72,2%	49,9%	35,8%	98,4%
Queen	65,2%	60,1%	75,8%	43,3%	28,5%	98,8%
The Beatles	69,1%	62,8%	81,9%	42,5%	27,7%	99,5%
Zweieck	68,6%	61,7%	81,0%	43,3%	28,0%	99,1%
Average	**66,4%**	**62,4%**	**77,4%**	**45,6%**	**31,1%**	**99,0%**

	Best f-score Harte			Best Recall Harte		
Dataset title	F	P	R	F	P	R
MTG-JAAH	61,7%	58,4%	74,5%	43,5%	29,6%	99,2%
Queen	57,9%	49,7%	74,71%	37,0%	23,3%	99,7%
The Beatles	60,5%	51,0%	79,9%	36,6%	22,9%	99,8%
Zweieck	61,0%	50,8%	79,3%	35,9%	22,2%	99,4%
Average	**60,8%**	**53,9%**	**77,2%**	**39,4%**	**25,6%**	**99,5%**

F f-score P precision R recall

datasets, as they prominently appear in the best-ranked f-score and recall results from the grid search (75 and 58 out of the 100 best-ranked parameterizations, respectively). The approximate transcription provided by the C_{NNLS} chromagram is equally identified in the top-ranked grid f-score search results (98 out of 100 best-ranked grid search's parameterizations).

Median-filtering harmonic percussive source separation filter [24] improves the f-score measure (60 out of 100 best-ranked grid search's parameterizations). In particular, by up to 2% on the 60 cited best ranked f-score grid search results with HPSS.

Window overlap of 50% overlap provide enhanced results. Sample rates of 8000 samples per second are enough for getting good HCDF performance. This allows, by being smaller the window, that HCDF has a computational cost in less time (more efficient).

Harte et al. [31] HCDF parametrization proposed 15 years ago scores with lower results, as shown in Table 4.3 than the other algorithms.

Our best results corroborate Harte et al. [31] findings that the adoption of a tonal model contributes to a high recall score R at the expense of precision P. Our results suggest that the new algorithms and parameters are better at discriminating significant harmonic changes in the signal, while significantly reducing the number of false positives in the HCDF ξ. Furthermore, as suggested in [31], these low precision scores P can be explained by the fact that the ground truth

Table 4.3: *Experimental results for HCDF peaks compared with hand labelled chord changes for 16 Beatles Songs (songs arranged in chronological order of release date). Please Please Me (1), Do You Want To Know A Secret (2), All My Loving (3), Till There Was You (4), A Hard Day'sDay's Night (5), If I Fell (6), Eight Days A Week (7), Every Little Thing (8), Help! (9), Yesterday (10), Drive My Car (11), Michelle (12), Eleanor Rigby (13), Here There And Everywhere (14), Lucy In The Sky With Diamonds (15), Being For The Benefit Of Mr Kite (16).*

	Best Precision Proposed			Best Recall Proposed			Best Harte		
Song #	F	P	R	F	P	R	F	P	R
1	76%	74%	77%	52%	35%	100%	65%	53%	87%
2	73%	89%	62%	70%	54%	98%	75%	72%	80%
3	79%	74%	84%	44%	28%	100%	63%	50%	86%
4	77%	78%	76%	56%	39%	100%	71%	59%	90%
5	81%	78%	83%	50%	34%	100%	54%	45%	70%
6	87%	86%	72%	56%	39%	100%	65%	79%	53%
7	75%	74%	76%	48%	32%	100%	61%	49%	82%
8	74%	80%	69%	55%	38%	94%	56%	49%	67%
9	65%	53%	84%	35%	21%	100%	41%	29%	74%
10	70%	73%	67%	62%	45%	98%	73%	64%	86%
11	72%	67%	78%	45%	29%	100%	62%	49%	86%
12	76%	71%	81%	47%	31%	98%	66%	53%	90%
13	75%	63%	90%	35%	21%	100%	47%	34%	81%
14	81%	77%	84%	55%	38%	100%	72%	65%	83%
15	78%	76%	79%	47%	31%	99%	63%	50%	88%
16	84%	86%	82%	54%	37%	100%	79%	68%	95%
Average	**75%**	**74%**	**78%**	**51%**	**34%**	**99%**	**64.9%**	**53%**	**84%**

F f-score *P* precision *R* recall

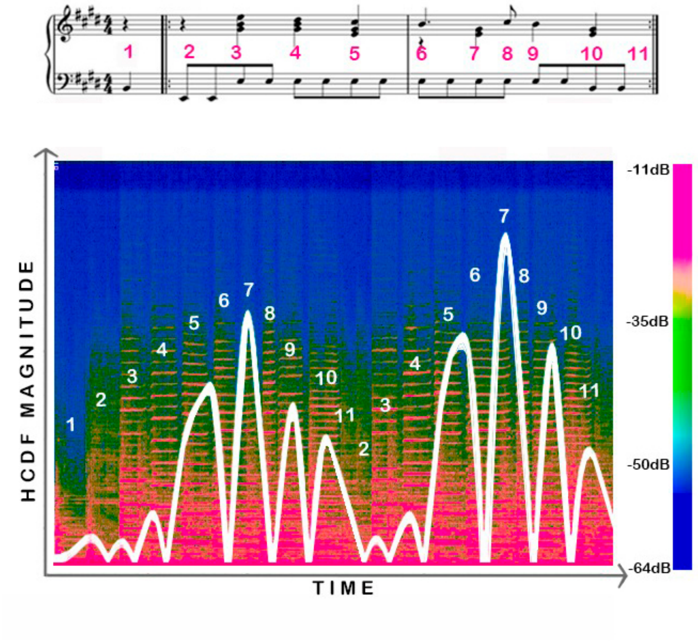


Figure 4.7: *Subjectivity in annotations. Beginning of The Beatles' song Please Please Me. Score and HCDF diagram over spectrogram. From 2 to 11 every boundary is a false positive. The numbers are in the same temporal position on the score as on the spectrogram.*

annotations only label chord changes. However, HCDF ξ is also sensitive to changes in harmonic content caused by strong melody or bass line movements that include non-chord tones. Thus, a high number of false positives is to be expected for this experiment and does not necessarily denote a perceptual phenomenon.

As shown in [54] and [36], besides the high complexity of polyphonic music and the subjectivity of its annotations, the adopted ground truth is not exactly prone to the task at hand from a perceptual viewpoint, yet not only for comparative, and legacy purposes are still adopted in our work, but also due to the importance of HCDF in supporting ACD systems. Most of the false positives in the HCDF ξ can be seen in Figure 4.7, one could argue that the harmonic change is correct when seen from a non-functional perspective. Indeed, a slice by slice analysis of the piece would yield the same results as the HCDF ξ: although functionally there is a sustained E chord, the notes that are being played at each point describe a succession of different chords, as one can see in Figure 4.7. When the bass line moves a lot, it can generate false positives, as shown in Figure 4.6. Sometimes similar errors can occur when quite large variations of percussive sounds are combined with the harmony. This is solved mainly by the HPSS filter even in chromagrams that should be robust to bass with a large number of harmonics.

If it is analyzed a small and biased dataset 4.3 as which was analyzed by Harte et al. [31] due to not existence of datasets 15 years ago, results are very impressive. Tonal interval space combined with C_{NNLS} outperforms Harte et al. results in precision oriented algorithm and in recall oriented algorithm.

4.4 Visualizing Harmonic Change Detection Function

In this section, we will discuss the HCDF departing from its graphical representation aiming to better grasp the implication of the multiple algorithms and parameterizations proposed. To this end, multiple visualizations for the 60 first seconds of song *Please Please Me* by The Beatles using different HCDF conditions are plotted.

4.4.1 Visualizing f-score results

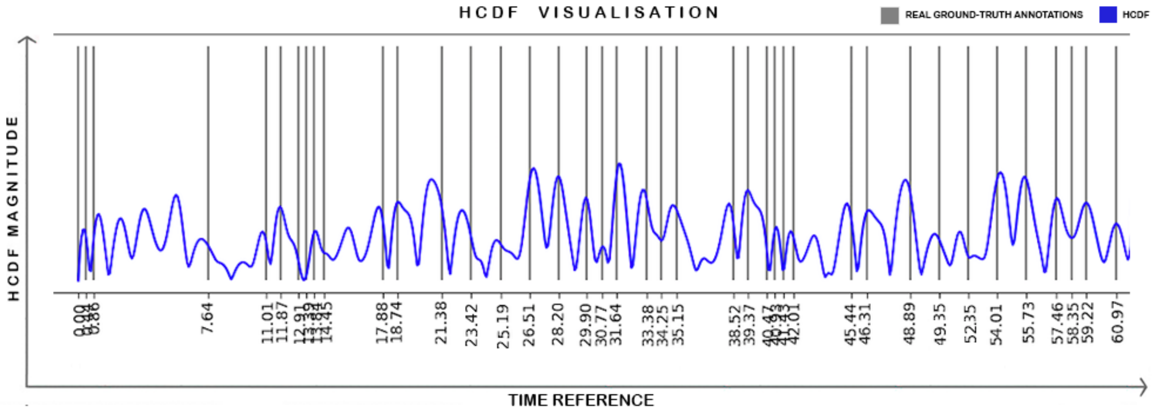


Figure 4.8: *HCDF diagram. First 60 seconds of Please Please Me. Best f-score result of grid search HCDF*

Figure 4.8 shows the HCDF for the best f-score result from the grid search 4.3. The parameterization whose output from the grid search have been selected and applied to Please Please Me. As it can see, in the above-mentioned HCDF diagram, the peaks represent harmonic changes. If the peaks are higher (of greater magnitude), it means that the harmonic change is higher from a tonal perspective. As it can be easily seen in the tonnetz diagram 2.9 thirds and fifths are closer to other intervals between chords, moreover, Tonal Interval Space proposed by Gilberto et Al. [4] expands the intervals recognized, as is explained in Chapter 2. Moreover, F-score measure evaluated with ground-truth annotations is $75,3\%$ with a balanced precision/recall; precision is $73,4\%$ and recall is 77.2%.

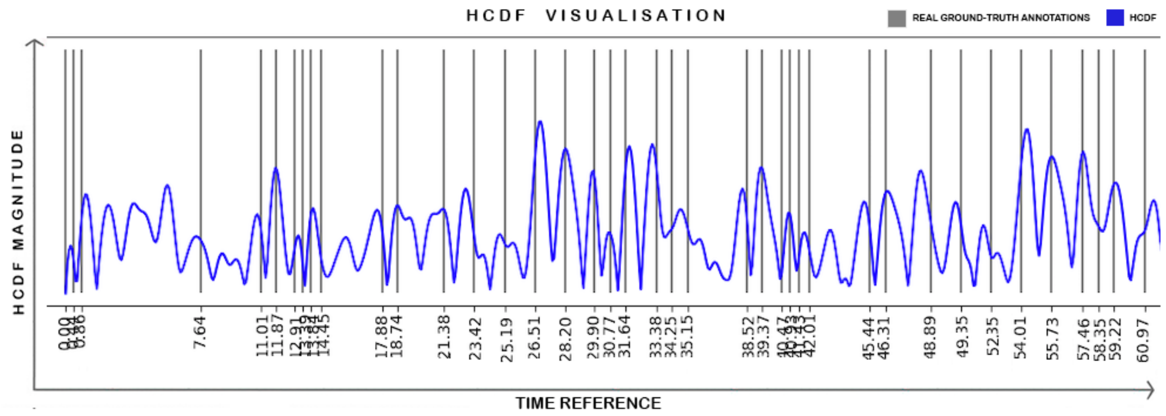


Figure 4.9: *HCDF diagram. First 60 seconds of Please Please Me. Best f-score result of grid search HCDF without HPSS method in preprocessing block*

Figure 4.9 shows an HCDF with a parameterization from the best f-score result without HPSS. This allows us to visually compare the behaviour of the HCDF when adopting the HPSS. The HCDF parameterizations, adopts a sample rate of $s = 8000$ Hz, a window size of $f = 1024$ and a window overlap of $o = 50\%$. Harmonic content representation results from C_{NNLS} chromagram further encoded and projected on the tonal model $T(k)$ with w_a. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 5$.

When HPSS is adopted, we can observe that the magnitude of the peaks is reduced, most probably as the result of the lack of percussive transients. Despite the results improvement of 2% when adopting HPSS, in this particular song f-score is 76,4%, precision 71,4% and recall is 82.2%. Although the results are better the difference between precision and recall is higher.

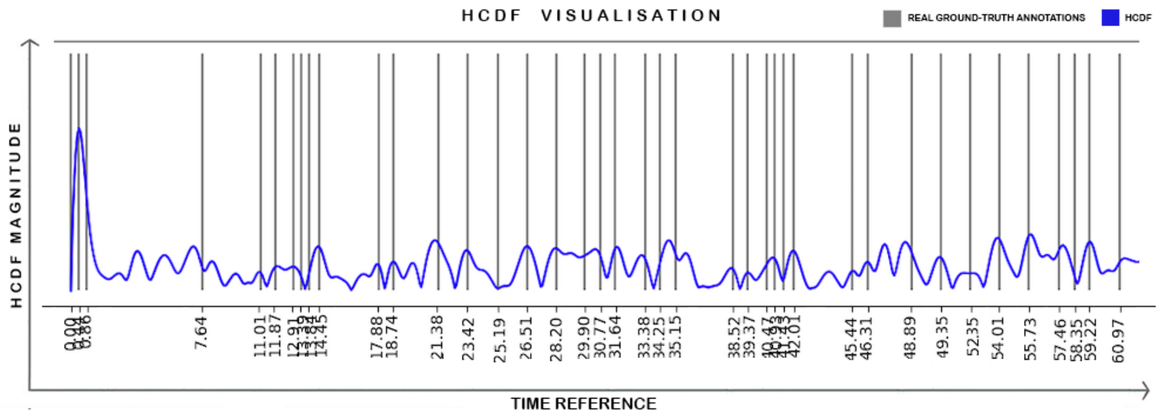


Figure 4.10: *HCDF diagram. First 60 seconds of Please Please Me . Best f-score result of grid search HCDF without tonal model.*

Figure 4.10 shows the best result for f-score without tonal space. Smoothing and distances are computed directly on the chromagram. This HCDF parameterizations adopts a sample rate of $s = 8000$ Hz, a window size of $f = 1024$ and a windows of $o = 50\%$ overlap. Musical audio is preprocessed using HPSS. Harmonic content representation results from C_{NNLS}. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 5$.

Conversely to Figures 4.9 and 4.8, the peaks in Figure 4.10 are very homogeneous in terms of magnitude. F-score is $72,1\%$, precision $67,1\%$ and recall is 77.2%. The f-score is 10%, lower than previous parameterizations (or visualizations), and the difference between precision/recall is three times wider than Figure 4.8. Chromagrams with a low computational cost like C_{CQT} or C_{STFT} perform well under this set of parameters.

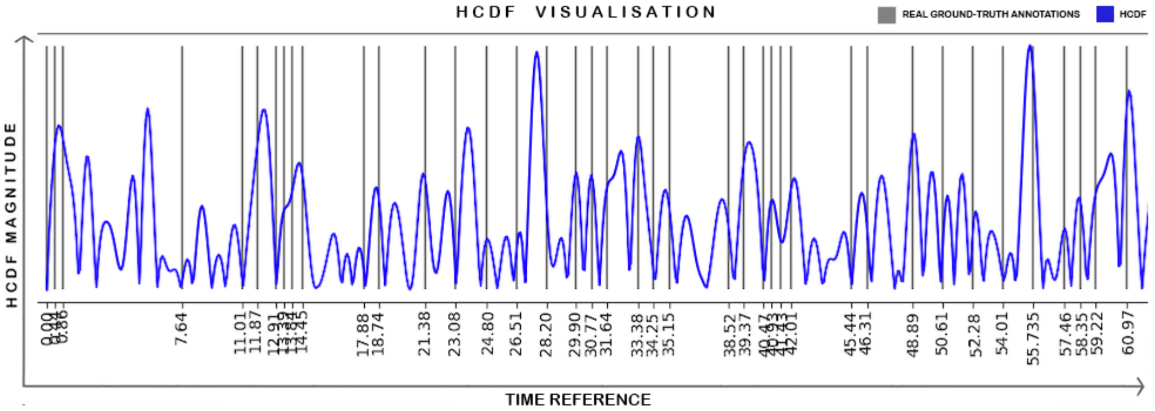


Figure 4.11: *HCDF diagram. First 60 seconds of Please Please Me. Best f-score result of grid search HCDF with $W(h)$ as a tonal model.*

Figure 4.11 adopts the parameterization originally proposed in Harte et al. In particular, it adopts a sample rate of $s = 44100$ Hz, a window size of $f = 2048$ and a window overlap of $o = 25\%$. Musical audio is preprocessed without HPSS. Harmonic content representation results from C_{STFT} chromagram further encoded and projected on the tonal model $T(k)$ with w_h. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 17$.

The resulting HCDF has in Figure 4.11 has less peak magnitude variance than tonal spaces based on Tonal Interval Space $w(a)$, as shown in the Figure 4.9 and 4.8. We believe that these results stem from the added intervallic relations in the tonal space $T(k)$ with w_s or w_a, which promote enhanced difference in triadic harmony above the sevenths chords. In this case f-score equals $63,7\%$, precision $55,6\%$ and recall 74.68%.

4.4.2 Visualizing recall results

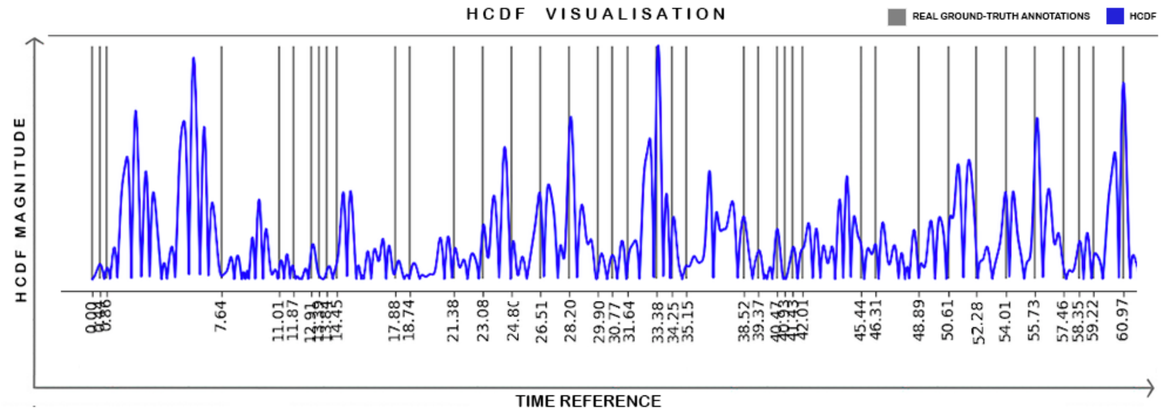


Figure 4.12: *HCDF diagram. First 60 seconds of Please Please Me. HCDF with parameterization adopted from best recall result of grid search.*

The HCDF shown in Figure 4.12 adopts the parameterization which provides the best recall results 4.3 in the grid search method. We can easily observe the increase in the number of false positives when compared with the best f-score results shown in Figure 4.8. To achieve these results, as discussed in Section 4.3, the ratio between windows size s and frame windows f have to be very high. F-score results equal to $51,8\%$, precision decreases significantly in comparison with f-score oriented methods with a value of $34,9\%$, and recall is $1,0\%$. The visualization shows than every time, when there is a (gray) chord change, a detected HCDF (blue) hit exists.

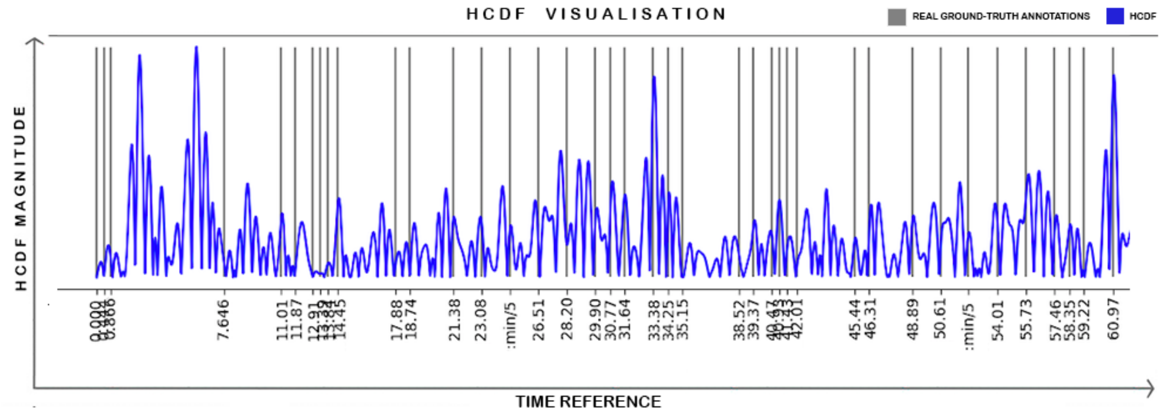


Figure 4.13: *HCDF diagram. First 60 seconds of Please Please Me. Best recall result of grid search HCDF without HPSS preprocessing block.*

Figure 4.13 shows an HCDF adopting the best parameterization result for recall. From the above-mentioned parameterization, the HPSS has been removed from the preprocessing block. This allows to visually compare the impact of the HPSS in the HCDF. Further parameterizatio include sample rate of $s = 44.100$ Hz, a window size of $f = 2048$ and a window overlap of $o =$

25%. Processed harmonic content representation is further encoded as an C_{STFT} chromagram and projected on the tonal model $T(k)$ with w_a. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 17$.

Figure 4.13 shows similar behavior in false positive as the f-score visualiztion without HPSS shown in Figure 4.9. F-score equals to 50,6% (1% lower than the HCDF shown in Figure 4.12. A similar decreased scores in precision is observed with a score of 33,9%. However, recall is 1,0%. For most of the recall-oriented applications, the absolute recall value is important to consider. But, the computational tradeoff cost of leaving 1% of precision can be useful, especially in real-time applications where lower computational cost can be a need.

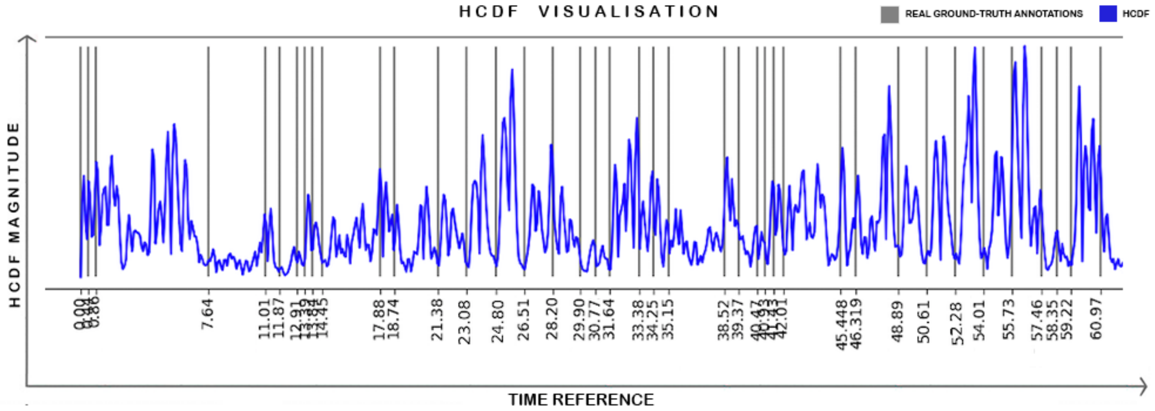


Figure 4.14: *HCDF diagram. First 60 seconds of Please Please Me. Best recall ranked HCDF parametrization without Tonal Space.*

Figure 4.14 shows the best ranked recall score for a HCDF without tonal space. There are several parameterizations with no tonal model than performs with a $> 99\%$. Most of the parameters used to compute this visualization change in comparison to the Figure 4.12 because it is searched a perfect recall. The HCDF parameterization without tonal model that provides best recall results adopt a sample rate of $s = 1050$ Hz, a window size of $f = 8192$ and a window overlap of $o = 12.5\%$. Musical audio is preprocessed without HPSS. Harmonic content representation results from C_{HPCP} chromagram further encoded. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing the raw chromagrams with $\sigma = 1$.

Visually, the HCDF is very are erratic. We believe that it may be due to the small value of $\sigma = 1$. Moreover, there is not a correlation between HCDF magnitude and the type of chord that have changed. However, rate of false positives is very similar to other recall visualizations as Figure 4.12 and Figure 4.13. F-score equals to 46,4%, precision equals to 30,35% and recall equals to 98.7%.

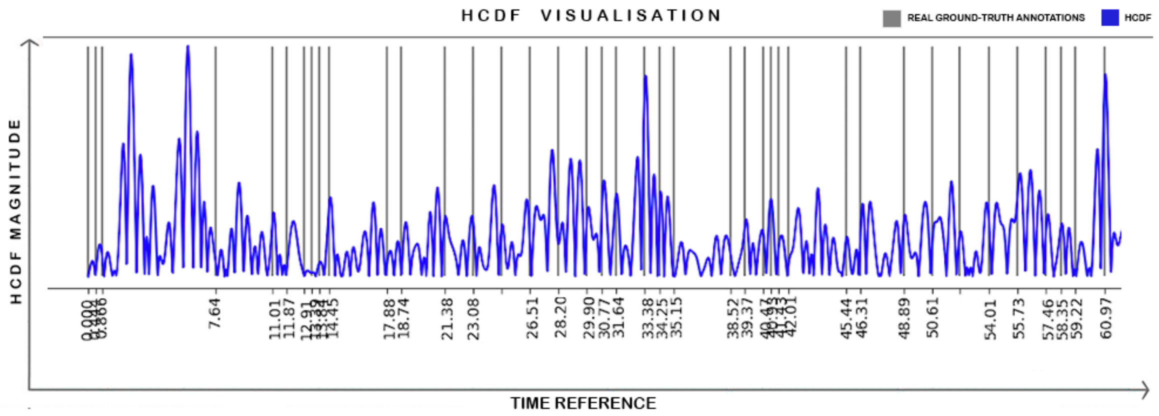


Figure 4.15: *HCDF diagram. First 60 seconds of Please Please Me. Best recall result of grid search HCDF with $W(h)$ as a tonal model.*

Figure 4.15 shows the best ranked HCDF with recall oriented optimization and $T(k)$ with w_h, i.e. the weights proposed by Harte et al. [31]. The HCDF parameterization, adopts a sample rate of $s = 44100$ Hz, a window size of $f = 1024$ and a window overlap of $o = 100\%$. Musical audio is preprocessed without HPSS. Harmonic content representation results from C_{HPCP} chromagram. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 9$.

HCDF behaves similarly to Figures 4.12 and 4.13. Moreover, it has more smooth curves than Figure 4.14. F-score performs similar to other HCDF parameterizations with a $50,48\%$ value. Precision equals to $33,9\%$ and recall equals to 98.73%.

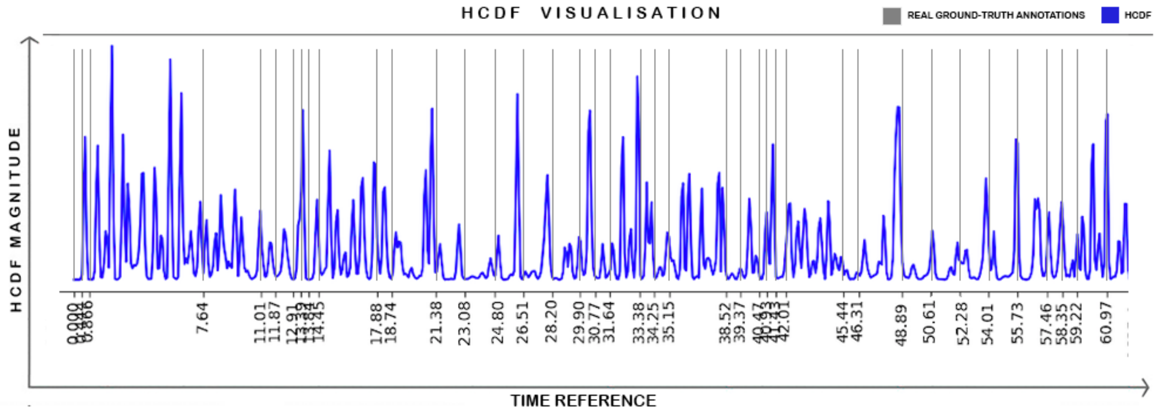


Figure 4.16: *HCDF diagram. First 60 seconds of Please Please Me. Best recall ranked on grid search HCDF with Cosine $\xi_n^{\cos}\}$ as a centroid distance.*

In the top ranking recall results of the grid search, we can find several parameterizations adopting the cosine distance $\xi_n^{\cos}\}$ [1]. The top ranking HCDF result adopting cosine distance $\xi_n^{\cos}\}$

[1] In the f-score comparison no visualizations of the algorithm with cosine $\xi_n^{\cos}\}$ distance are included because they perform not at the top of the grid search rank for f-score.

is shown in Figure 4.16. The HCDF adopts a sample rate of $s = 22050$ Hz, a window size of $f = 16384$ and a window overlap of $o = 12.5\%$. Musical audio is preprocessed using HPSS. Harmonic content representation results from C_{HPCP} chromagram further encoded and projected on the tonal model $T(w)$ with w_a. For computing the HCDF ξ, we adopt the Euclidean distance ξ_n^{eucl} after Gaussian smoothing with $\sigma = 1$.

This representation is very illustrative, chords that are "perceptually" related have a small HCDF magnitude and less related chords follow the opposite trend (i.e., larger magnitudes). Moreover, HCDF increments and decreases are very well defined. F-score and precision equal to $47,8\%$ and $31,6\%$, respectively, and the equals to 97.4%.

Chapter 5

Applications of HCDF

HCDF have multiple applications that range from more technical tasks within the scope of MIR to creative-oriented applications. In this chapter, we detail three of such applications: Automatic Chord Recognition, Audio Visualizations and Musical Audio Harmonization.

5.1 Automatic Chord Recognition

Automatic Chord Recognition (ACR) is a primary tasks within MIR which typically adopts harmonic change detection algorithms as a preprocessing stage. To the best of our knowledge, Fujishima [25] published the first system for ACR in 1999. Since then, multiple approaches have been proposed ranging from first knowledge-based systems to data-driven methods using deep learning architectures [58].

ACR was initially solved in a knowledge-based approach, by trying to approximate audio fragments to a template [25]. Nowadays, knowledge-based approaches such as the Tonal Interval Space [4] are still being tried for modeling harmony. The data-driven approaches has been predominant in recent computer science literature [12, 90]. This dissertation explores a data-driven approach for understanding the different HCDF parameterizations. In the previous five years, many proposals in MIR and SMC have been proposed from a data-driven approach. From more sophisticated chroma features to learnt Gaussian chord models. In ACR attacked from Deep Learning, pipelines become blurred as systems are more monolithic, as is shown in [46].

Probabilistic models such as Hidden Markov Models [18, 76, 85] or dynamic Bayesian networks [65, 41] have been the primary algorithms to integrate other musical concepts with chord detection, such as key, genre, bass or melody. However, these systems have many false positives because the chord labels do not align well with the real chord boundaries **??** and this problem is the one that HCDF focuses on. The musical concept most close to chord is key. In pop and rock music genres, some chords are very much probable than others, once we know the key of the analysed audio. These probabilities can be data-driven even deep learning models can be trained with this information [33, 13]. Another target is to get an ACR system close to functional harmony analysis, which has a higher information density [72], although this representation does not have demonstrated benefits in ACR.

HCDF is usually adopted early in the ACR algorithmic pipeline. It aims to segment the signal into structurally-aware units (or chord boundaries). Recent state-of-the-art methods for ACR use deep-learning architectures with HCDF [88]. Related ACR literature to our work can be found in the work [34], which adopt the *Tonnetz* as a base representation, in a similar fashion as the tonal space here proposed.

5.2 Creative applications

In this section, two creative applications of HCDF will be presented: Audio Visualizations and Musical Audio Harmonization. In order to give a robust idea of the multiple applications that HCDF can have.

5.2.1 Audio Visualizations

Visualizations of music have been pervasive across the widely popular music players. most humans recognize visualizations of Windows media player or SoundCloud, as is shown in Figure 5.1. However, to create these visualizations that change with music, usually, rhythm content has been used. With the analysis of parameterization and performance over different harmonic attributes as those detailed in this dissertation, it could be possible to expand these music visualisations platforms. For example, HCDF peak magnitudes, denoting harmonically proximity across time, can provide harmonically-aware segmentation and perceptually affinity across time that can be mapped to visual settings.

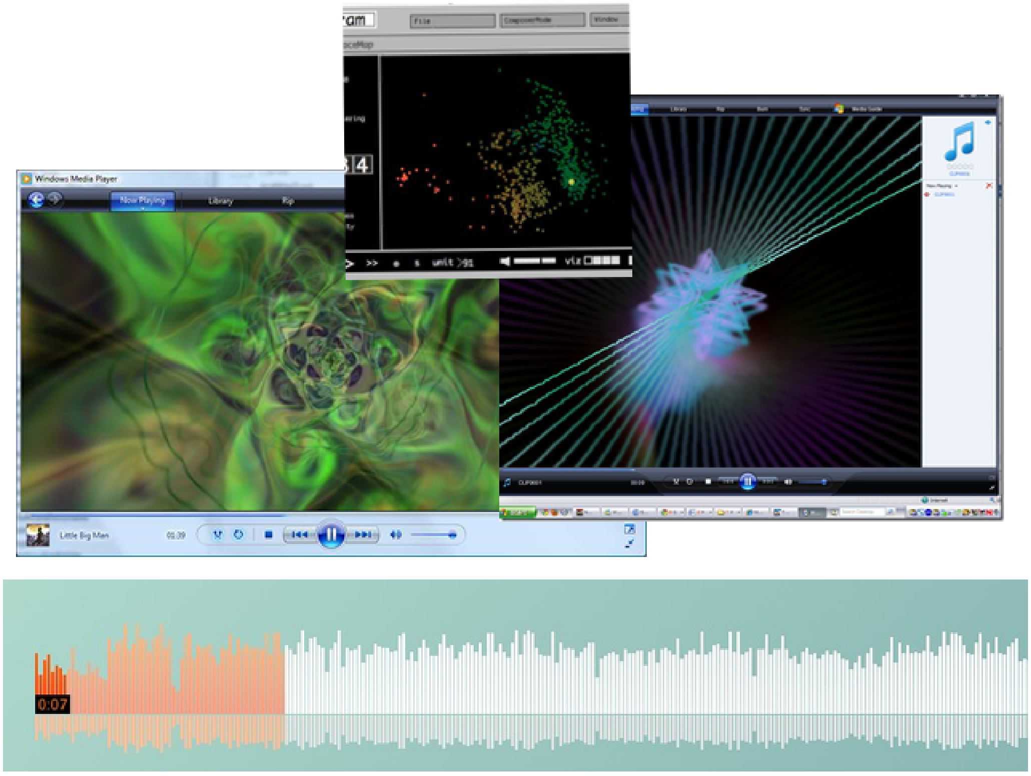


Figure 5.1: *Some music industry visualization systems.*

5.2.2 Musical Audio Harmonization: D'accord

Another creative application that can make use of HCDF is the automatic harmonization of music. D'accord is a representative example of a generative music system for creating harmonically compatible accompaniments [5]. It harmonizes musical audio by accompaniments with user-specified number of voices, instrumentation and complexity. On the backend of the system it relies the tonal space $T(k)$ for harmonic-aware segmentation, key detection and harmonization. D'accord was originally developed for Ableton Live, MAX, and Pure Data. Figure 5.2 shown the interface of D'accord in Pure Data.

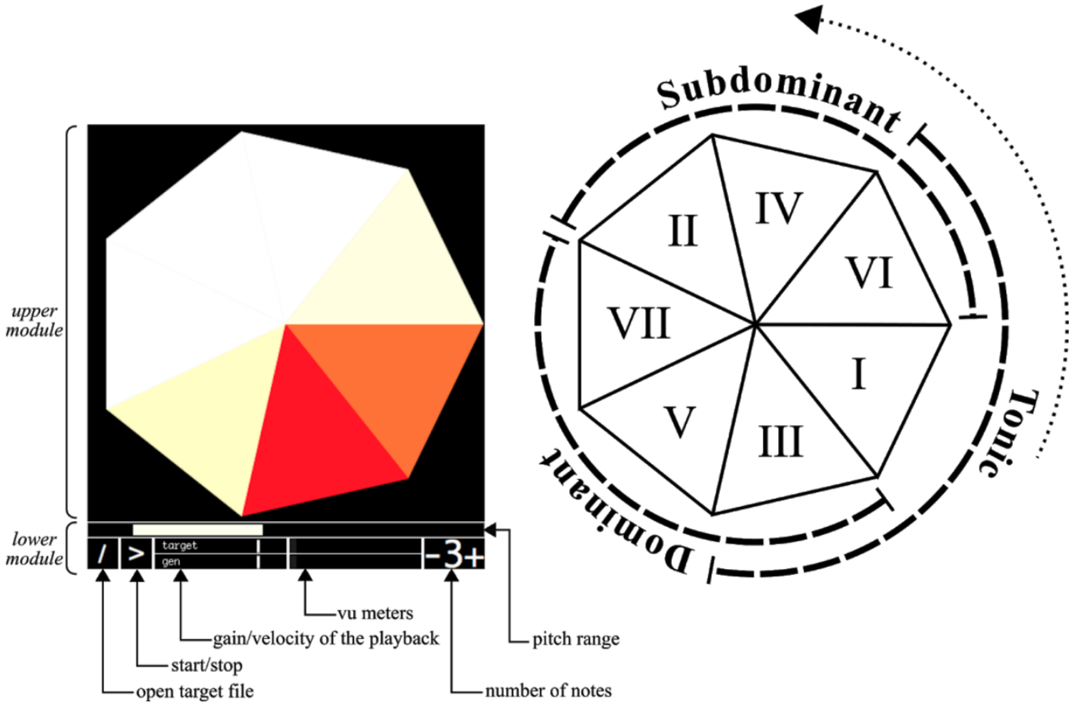


Figure 5.2: *D'accord desktop interface. Diagram extracted from Gilberto personal site.*

An ongoing revision of the tool is under development as a web-based application. The proposed method, and its javascript implementation, is supporting the harmonic change detection from musical audio in the browser.

Chapter 6

Conclusions

In this dissertation, we revisited harmonic change detection in light of recent advances in harmonic description and transformation. Stemming from Harte et al.'s [31] HCDF, we proposed novel algorithms for each of their processing blocks. We exhaustively inspected the parameters that best performs across the multiple algorithm combinations using a grid search method. We evaluated the proposed algorithms and its parameterization on four style-specific datasets (The Beatles, Queen, Zweieck and MTG-JAAH).

Different preprocessing strategies have been applied to vary the sample rate s, windows size f, and overlap o, as well the adoption of harmonic-only signal decomposition from the median-filtering HPSS algorithm. Four different chromagrams C_{STFT}, C_{CQT}, C_{NNLS} and C_{HPCP} and the recently proposed tonal space $T(k)$ with weights w_a, w_s and w_h have been tested. Finally, Euclidean ξ_n^{eucl} and cosine $\xi_n^{\cos}$ distance metrics and $1 \leq \sigma \leq 17$ values in Gaussian filtering have been evaluated in the HCDF performance.

Our results showed that the newly proposed algorithms and parameters improved previous HCDF computation in detecting harmonic changes. The adoption of the Tonal Space $T(k)$ with w_a and C_{NNLS} improved f-score and recall by 5.57% and 6.28%, respectively. An assessment of the HCDF ξ across a significantly larger number of musical examples from multiple styles not only improves the generality of the method for unknown musical audio sources but also suggests the link between harmonic change rates and parameterization of the HCDF.

6.1 Original Contributions and Model Implementations

With the objective of disseminating our contribution to improving HCDF, we made available the resulting HCDF for processing the best f-score and recall results. Furthermore, the models have been developed as functions for real-time audio processing in Pure Data [64] and for offline processing in Python [1], Javascript [2]. A Javascript implementation of TIVlib [3] has been made available. Furthermore, we also distribute a well-structured dataset containing all the resulting data from the grid search method. It has been uploaded to Google Drive[4] in a legible format, JSON.

6.2 Further work

In future work, we aim to tackle the adoption of parameters per genre (or for different rates of harmonic changes). Moreover, we believe that an adaptive threshold for detecting the peaks (i.e. harmonic changes) can improve the peak picking phase in detecting chord boundaries in the HCDF. Ultimately, this should improve precision by eliminating false positives.

In Future research, we would like to improve the use of the Tonal Interval Space. For example, to carry out Automatic Change Recognition methods on the basis of previous similar research [34].

We will develop a computationally-efficient harmonic rhythm detector. The harmonic rhythm or rate of harmonic change is very crucial. A computer vision approach could be used, songs' chromagrams would be as a 2d image with grey colours (energy have only one dimension). At first sight it is possible to observe a relationship between the harmonic rhythm and the chromagram in a song, as is shown in Figure 6.1.

[1] At: https://github.com/PRamoneda/HCDF.

[2] At: https://github.com/PRamoneda/HCDF.js.

[3] At: https://github.com/PRamoneda/TIVlib.js.

[4] At: https://drive.google.com/drive/folders/1a-SzgmqPf7DmSnNaXAZM8b1MuBv1Id1A?usp=sharing.

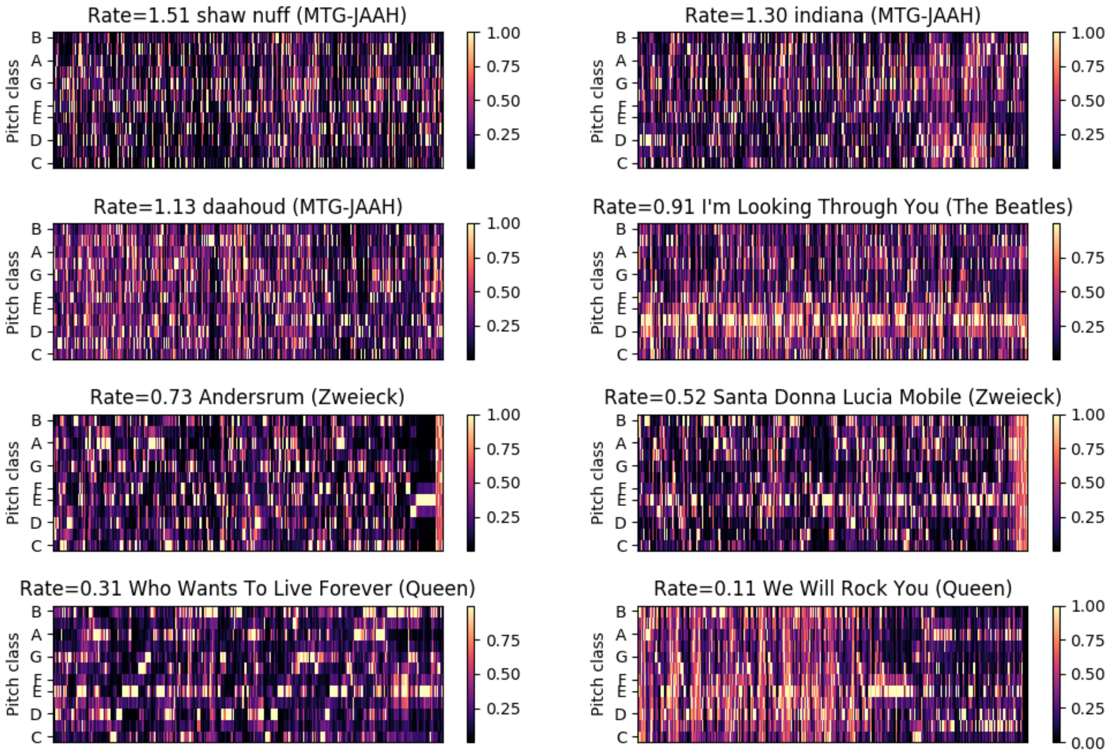


Figure 6.1: *Evolution of the chromagrams depending on the rate of harmonic change.*

It would be beneficial to apply neural networks to audio Transient/Steady-State separation for processing the musical audio before applying HCDF. It would also be interesting to research the possibility of developing a detector of spurious peaks in HCDF.

Some early experiments on the adoption of thresholding techniques in the peak picking stage have been pursued. One of them is to use a double harmonic change detection function. First one was optimized to generate many changes, and then the second one was over the sum of the chromagrams of the boundaries found by the first one HCDF. This second HCDF optimized to get the best possible results. We would use different algorithms in each of the blocks of each of the two algorithms. We understood that the overlap of the two solutions would get a better solution. However, the results of the experimental tests were not good. Last but not least, an optimal adaptive function could be developed through deep learning, machine learning or reinforcement learning combined with digital signal processing. This adaptive function would improve significantly HCDF and we will keep our research in this direction.

Appendix A

Anexo

A.1 Python library

A.1.1 README

~~~
# Harmonic Change Detection Function (HCDF) library

This library is used to compute HCDF. [Here]() is described the algorithm in det


## Installation
Install Vamp-plugins:
- (NNLSchroma) http://www.isophonics.net/nnls-chroma
- (HPCPchroma) http://mtg.upf.edu/technologies/hpcp
Install dependencies:
```BASH
        pip3 install requeriments.txt
```

## Usage
The library can be imported as a module with `import HCDF`. All the functions th
function. The rest are auxiliar functions.

HCDF.py act as script allowing the user to print to the console Harmonic Change
focus on maximizing recall or f-score. It is assumed that the first command line
the name file of the audio file located in audio_files and the second one is, if
recall or f-score.

### Example of use
~~~

~~~
With target on maximize f-score:

```BASH
python3 f-score file/name
```

With target on maximize recall:

```BASH
python3 recall file/name
```



## References

https://librosa.github.io
https://vamp-plugins.org
https://github.com/aframires/TIVlib
~~~

A.1.2 CODE

```
"""HCDF python implementation

Author: Pedro Ramoneda Franco
Year: 2020

This script allows the user to print to the console Harmonic Change Detection Functi
focus performance on recall or f-score. It is assumed that the first command line arg
the name file of the audio file located in audio_files and the second one is if is fo
recall or f-score.


This tool accepts comma separated value files (.csv) as well as excel
(.xls, .xlsx) files.

This script requires that `setup.py` requeriments be installed within the Python
environment you are running this script in. More over it is a need instal vamp plugi
NNLS and HPCP

This file can also be imported as a module. All the functions than begins by get are
function. The rest are auxiliar.
```

```
"""
import os
from os import path

import sys
from TIVlib import TIV
import librosa
import numpy

numpy.set_printoptions(threshold=sys.maxsize)


from librosa import display
from librosa.feature import chroma_cqt, tonnetz, chroma_cens, chroma_stft
from librosa.filters import get_window
from scipy.ndimage.filters import gaussian_filter
import matplotlib.pyplot as plt
from astropy.convolution import convolve, Gaussian1DKernel
from scipy.spatial.distance import cosine

from iotxt import load_real_onset, load_binary, get_name_harmonic_change, save_b
        get_name_tonal_model, get_name_gaussian_blur, get_name_audio
import vamp


def get_distance(centroids, dist):
        """
        Returns the quantity of centroids per second

        Parameters
        ----------
        centroids : list of floats
            The file location of the spreadsheet
        sr : bool
            A flag used to print the columns to the console (default is False)

        Returns
        -------
        float
```

```
                    centroids per second
                """
                ans = [0]
                if dist == 'euclidean':
                        for j in range(1, centroids.shape[1] - 1):
                                sum = 0
                                for i in range(0, centroids.shape[0]):
                                        sum += ((centroids[i][j + 1] - centroids[i][j - 1])
                                sum = numpy.math.sqrt(sum)

                                ans.append(sum)

                if dist == 'cosine':
                        for j in range(1, centroids.shape[1] - 1):
                                distance_computed = cosine(centroids[:, j - 1], centroids[:,
                                ans.append(distance_computed)
                ans.append(0)

                return numpy.array(ans)


def centroids_per_second(y, sr, centroids):
                """
                Returns the quantity of centroids per second

                Parameters
                ----------
                y : list of floats
                    The file location of the spreadsheet
                sr : bool
                    A flag used to print the columns to the console (default is False)

                Returns
                -------
                float
                    centroids per second
                """
                return sr * centroids.shape[1] / y.shape[0]
```

```
def get_peaks_hcdf(hcdf_function, c, threshold, rate_centroids_second, centroids

        changes = [0]
        centroid_changes = [[centroids[j][0] for j in range(0, c.shape[0])]]
        last = 0
        for i in range(2, hcdf_function.shape[0] - 1):
                if hcdf_function[i - 1] < hcdf_function[i] and hcdf_function[i +
                        centroid_changes.append([numpy.median(centroids[j][last+
                        changes.append(i / rate_centroids_second)
                        last = i
        return numpy.array(changes), centroid_changes


def everything_is_zero(vector):
        """Returns true if all the values of the vector are 0 if not return fals

        Parameters
        ----------
        vector : list
            vector of reals

        Returns
        -------
        bool
            true or false depending if everything is 0 or not
        """
        for element in vector:
                if element != 0:
                        return False
        return True


def complex_to_vector(vector):
        """transforms an array of i complex numbers in an array of 2*i elements
        odd indexes are the real part and even indexes are the imaginary part.

        Parameters
        ----------
        vector : list
            list of complex numbers
        Returns
        -------
        list
            list of real numbers with odd indexes as the real part and even indexes
        """
        ans = []
        for i in range(0, vector.shape[1]):
                row1 = []
                row2 = []
                for j in range(0, vector.shape[0]):
                        row1.append(vector[j][i].real)
                        row2.append(vector[j][i].imag)
                ans.append(row1)
                ans.append(row2)
        return numpy.array(ans)


def get_parameters_chroma(txt):
        """
                returns parameters of json "chroma-samplerate-framesize-overlap"

                Parameters
                ----------
                txt : str
                        chroma-samplerate-framesize-overlap

                Returns
                -------
                dictionary with keys: {chroma, samplerate, framesize, overlap}
        """
        rows = txt.split("-")
        return {"chroma": rows[0], "sr": int(rows[1]), "fr": int(rows[2]), "off": in


def tonal_interval_space(chroma, symbolic=False):
        """
                returns tonal interval space from a vector of chromagrams

                Parameters
                -----------
                chroma : list
                        list of chromagrams

                symbolic: bool
                        True for symbolic musical audio tonal interval space and

                Returns
                --------
                list of tonal interval space vectors
        """
        centroid_vector = []
        for i in range(0, chroma.shape[1]):
                each_chroma = [chroma[j][i] for j in range(0, chroma.shape[0])]
                # print(each_chroma)
                if everything_is_zero(each_chroma):
                        centroid = [0. + 0.j, 0. + 0.j, 0. + 0.j, 0. + 0.j, 0. +
                else:
                        tonal = TIV.from_pcp(each_chroma, symbolic)
                        centroid = tonal.get_vector()
                centroid_vector.append(centroid)
        return complex_to_vector(numpy.array(centroid_vector))


def check_parameters(chroma, blur, tonal_model, log_compresion, dist):
        chroma = get_parameters_chroma(chroma)["chroma"]
        chroma_type = {'nnls', 'hpcp', 'cqt', 'crp', 'stft', 'cens'}
        assert chroma in chroma_type, "Type of chroma is not correct ['nnls', 'h
        blur_type = {'none', '17-points', 'full'}
        assert blur in blur_type, "Type of blur is not correct ['none', '17point
        tonal_model_type = {'tonnetz', 'TIV2', 'TIV2_symb', 'without_tc'}
        assert tonal_model in tonal_model_type, "Type of tonal model is not corr
        log_compresion_type = {'after', 'before', 'none'}
        assert log_compresion in log_compresion_type, "Type of log_compresion is
        distance_type = {'euclidean', 'cosine'}
        assert dist in distance_type, "Type of distance is not correct ['euclidi


def get_nnls(y, sr, fr, off):
        """
```

```
                returns nnls chromagram

                Parameters
                ----------
                y : number > 0 [scalar]
                        audio

                sr: number > 0 [scalar]
                        chroma-samplerate-framesize-overlap

                fr: number [scalar]
                    frame size of windos

                off: number [scalar]
                        overlap

                Returns
                -------
                list of chromagrams
        """
        plugin = 'nnls-chroma:nnls-chroma'
        chroma = list(vamp.process_audio(y, sr, plugin, output="chroma", block_size=
        doce_bins_tuned_chroma = []
        for c in chroma:
                doce_bins_tuned_chroma.append(c['values'].tolist())
        return numpy.array(doce_bins_tuned_chroma).transpose()


def get_chromagram(y, sr, chroma):
        """
                returns chromagram

                Parameters
                ----------
                y : number > 0 [scalar]
                        audio

                sr: number > 0 [scalar]
                        target sampling rate
        chroma: str
            chroma-samplerate-framesize-overlap


        Returns
        -------
        list of chromagrams
    """
    params = get_parameters_chroma(chroma)
    chroma = params["chroma"]
    doce_bins_tuned_chroma = None
    if chroma == 'nnls':
        doce_bins_tuned_chroma = get_nnls(y, params["sr"], params["fr"],
    elif chroma == 'cqt':
        win = get_window('blackmanharris', params["fr"])
        doce_bins_tuned_chroma = chroma_cqt(y=y, sr=params["sr"],
                                            C=None,
                                            hop_length=params["off"],
                                            norm=None,
                                            # threshold=0.0,
                                            window=win,
                                            fmin=110,
                                            n_chroma=12,
                                            n_octaves=4 if params["chrom
                                            bins_per_octave=36)
    elif chroma == 'cens':
        win = get_window('blackmanharris', params["fr"])
        doce_bins_tuned_chroma = chroma_cens(y=y, sr=params["sr"],
                                             C=None,
                                             hop_length=params["off"],
                                             norm=None,
                                             window=win,
                                             fmin=110,
                                             n_chroma=12,
                                             n_octaves=5,
                                             bins_per_octave=36)
    elif chroma == 'stft':
        win = get_window('blackmanharris', params["fr"])
        doce_bins_tuned_chroma = chroma_stft(y=y, sr=params["sr"], hop_l
                                             n_chroma=12)
        return doce_bins_tuned_chroma


def chromagram(hpss, name_file, y, sr, chroma):
        """
                wrapper of get_chromagram for save all results for future same calcu

                Parameters
                ----------
                hpss : bool
                        true or false depends on hpss block

                name_file: str
                        name of the file that is being computed

                y : number > 0 [scalar]
                        audio

                sr: number > 0 [scalar]
                        target sampling rate

                chroma: str
                    chroma-samplerate-framesize-overlap



                Returns
                -------
                list of chromagrams
        """
        name_chromagram = get_name_chromagram(name_file, hpss, chroma)
        if path.exists(name_chromagram):
                dic = load_binary(name_chromagram)
        else:
                # if mutex_global.mutex is not None:
                #     mutex_global.mutex.acquire()
                doce_bins_tuned_chroma = get_chromagram(y, sr, chroma)
                # if mutex_global.mutex is not None:
                #     mutex_global.mutex.release()
                dic = {'doce_bins_tuned_chroma': doce_bins_tuned_chroma}
                # dic_save = {'doce_bins_tuned_chroma': doce_bins_tuned_chroma.tolis
```

```
            save_binary(dic, name_chromagram)
        # save_json(dic_save, name_chromagram + '.json')
        return dic['doce_bins_tuned_chroma']


def get_tonal_centroid_transform(y, sr, tonal_model, doce_bins_tuned_chroma):
                """
                returns centroids from tonal model

                Parameters
                ----------
                hpss : bool
                        true or false depends on hpss block

                name_file: str
                        name of the file that is being computed

                y : number > 0 [scalar]
                        audio

                sr: number > 0 [scalar]
                        target sampling rate

                chroma: str
                    chroma-samplerate-framesize-overlap

                tonal_model: str optional
                        Tonal model block type. "TIV2" for Tonal Interval space
                        "tonnetz" for harte centroids aproach. Default TIV2\

                doce_bins_tuned_chroma: list
                        list of chroma vectors

                Returns
                -------
                list of tonal centroids vectors
        """
        centroid_vector = None
        if tonal_model == 'tonnetz':
                centroid_vector = tonnetz(y=y, sr=sr, chroma=doce_bins_tuned_chr
        elif tonal_model == 'TIV2':
                centroid_vector = tonal_interval_space(doce_bins_tuned_chroma)
        elif tonal_model == 'TIV2_symb':
                centroid_vector = tonal_interval_space(doce_bins_tuned_chroma, symbo
        return centroid_vector


def tonal_centroid_transform(hpss, chroma, name_file, y, sr, tonal_model, doce_bins_t
        """
                wrapper of tonal centroid transform for save all results for future

                Parameters
                ----------
                hpss : bool
                        true or false depends on hpss block

                name_file: str
                        name of the file that is being computed

                y : number > 0 [scalar]
                        audio

                sr: number > 0 [scalar]
                        target sampling rate

                chroma: str
                    chroma-samplerate-framesize-overlap

                tonal_model: str optional
                        Tonal model block type. "TIV2" for Tonal Interval space focu
                        "tonnetz" for harte centroids aproach. Default TIV2\

                doce_bins_tuned_chroma: list
                        list of chroma vectors

                Returns
                -------
                list of tonal centroids vectors
        """
        name_tonal_model = get_name_tonal_model(name_file, hpss, chroma, tonal_model
        if tonal_model == 'without_tc':
                dic = {'centroid_vector': doce_bins_tuned_chroma}
        else:
                if path.exists(name_tonal_model):
                        dic = load_binary(name_tonal_model)
                else:
                        centroid_vector = get_tonal_centroid_transform(y, sr, tc
                        dic = {'centroid_vector': centroid_vector}
                        save_binary(dic, name_tonal_model)
        return dic['centroid_vector']


def get_gaussian_blur(centroid_vector, blur, sigma):
        """
                Apply gaussian smoothing to tonal model centroids

                Parameters
                ----------
                centoid_vector: list
                        tonal centroids of the tonal model

                sigma: number (scalar > 0) optional
                        sigma of gaussian smoothing value. Default 11

                Returns
                -------
                list
                        centroids blurred by gassuian smoothing
        """
        if blur == 'full':
                centroid_vector = gaussian_filter(centroid_vector, sigma=sigma)
        elif blur == '17-points':
                gauss_kernel = Gaussian1DKernel(17)
                i = 0
                for centroid in centroid_vector:
                        centroid = convolve(centroid, gauss_kernel)
                        centroid_vector[i] = centroid
        return numpy.array(centroid_vector)
```

```python
def gaussian_blur(hpss, chroma, tonal_model, name_file, centroid_vector, log_compres
        """
                Wrapper of get_gaussian_blur for save all results for future same ca
                have been computed before get_gaussian_blur is not computed.

                Parameters
                ----------
                name_file: str
                        name of the file that is being computed

                hpss: bool optional
                        true or false depends is harmonic percussive source separati

                sr: number > 0 [scalar]
                        target sampling rate

                chroma: str optional
                    "chroma-samplerate-framesize-overlap"
                        chroma can be "CQT","NNLS", "STFT", "CENS" or "HPCP"
                        samplerate as a number scalar
                        frame size as a number scalar
                        overlap number that a windows is divided

                tonal_model: str optional
                        Tonal model block type. "TIV2" for Tonal Interval space focu
                        "tonnetz" for harte centroids aproach. Default TIV2

                centoid_vector: list
                        tonal centroids of the tonal model

                sigma: number (scalar > 0) optional
                        sigma of gaussian smoothing value. Default 11


                Returns
                -------
                list
                        sample of audio
        """
        gaussian_blur = get_name_gaussian_blur(name_file, hpss, chroma, tonal_model,
        if path.exists(gaussian_blur):
                dic = load_binary(gaussian_blur)
        else:
                centroid_vector = get_gaussian_blur(centroid_vector, blur, sigma
                dic = {'centroid_vector': centroid_vector}
                # dic_save = {'centroid_vector': centroid_vector.tolist()}
                save_binary(dic, gaussian_blur)
        # save_json(dic_save, gaussian_blur + '.json')
        return dic['centroid_vector']


def get_audio(filename, hpss, sr):
        """
                Get audio as list

                Parameters
                ----------
                filename: str
                                name of the file that is being computed witout f

                hpss: bool optional
                        true or false depends is harmonic percussive source sepa

                sr: number > 0 [scalar]
                        target sampling rate

                Returns
                -------
                list
                        sample of audio
        """
        y, sr = librosa.load(filename, sr=sr, mono=True)
        if hpss:
                y = librosa.effects.harmonic(y)
        return y, sr


def audio(filename, name_file, hpss, sr):
        """
                Wrapper of get audio for save all results for future same calcul
                have been computed before get audio is not computed.

                Parameters
                ----------
                filename: str
                                name of the file that is being computed witout forma

                name_file: str
                        name of the file that is being computed

                hpss: bool optional
                        true or false depends is harmonic percussive source separati

                sr: number > 0 [scalar]
                        target sampling rate

                Returns
                -------
                list
                        sample of audio
        """
        name_audio = get_name_audio(name_file, hpss, sr)
        if path.exists(name_audio):
                dic = load_binary(name_audio)
        else:
                y, sr = get_audio(filename, hpss, sr)
                dic = {'y': y, 'sr': sr}
                # dic_save = {'y': y.tolist(), 'sr': sr}
                save_binary(dic, name_audio)
        # save_json(dic_save, name_audio + '.json')
        return dic['y'], dic['sr']


def get_harmonic_change(filename: str, name_file: str, hpss: bool = False, tonal_mode
                        chroma: str = 'cqt',
                        blur: str = 'full', sigma: int = 11, log_compresion: str = 'r
        """
                Computes Harmonic Change Detection Function

                Parameters
```

```
            ----------
            filename: str
                            name of the file that is being computed witout f
            
            name_file: str
                    name of the file that is being computed
            
            hpss : bool optional
                    true or false depends is harmonic percussive source sepa
            
            tonal_model: str optional
                    Tonal model block type. "TIV2" for Tonal Interval space
                    "tonnetz" for harte centroids aproach. Default TIV2
            
            chroma: str optional
                "chroma-samplerate-framesize-overlap"
                    chroma can be "CQT","NNLS", "STFT", "CENS" or "HPCP"
                    samplerate as a number scalar
                    frame size as a number scalar
                    overlap number that a windows is divided
            
            sigma: number (scalar > 0) optional
                    sigma of gaussian smoothing value. Default 11
            
            distance: str optional
                    type of distance measure used. Types can be "euclidean"
            
            
            Returns
            -------
            list
                    harmonic changes (the peaks) on the song detected
            list
                HCDF function values
            number
                windows size
    """
    # audio
    y, sr = audio(filename, name_file, hpss, get_parameters_chroma(chroma)["
        # chroma
        doce_bins_tuned_chroma = chromagram(hpss, name_file, y, sr, chroma)

        # tonal_model
        centroid_vector = tonal_centroid_transform(hpss, chroma, name_file, y, sr, to

        # blur
        centroid_vector_blurred = gaussian_blur(hpss, chroma, tonal_model, name_file,
                                                sigma)

        # harmonic distance and calculate peaks
        harmonic_function = get_distance(centroid_vector_blurred, dist)
        windows_size = centroids_per_second(y, sr, centroid_vector_blurred)
        changes, centroid_changes = get_peaks_hcdf(harmonic_function, centroid_vecto
                                                   centroid_vector)

        return changes, harmonic_function, windows_size, numpy.array(centroid_change


def harmonic_change(filename: str, name_file: str, hpss: bool = False, tonal_model: s
                    blur: str = 'full', sigma: int = 11, log_compresion: str = 'none
        """
                Wrapper of harmonic change detection function for save all results fo
                have been computed before HCDF is not computed.

                Parameters
                ----------
                filename: str
                                name of the file that is being computed witout forma

                name_file: str
                        name of the file that is being computed

                hpss : bool optional
                        true or false depends is harmonic percussive source separati

                tonal_model: str optional
                        Tonal model block type. "TIV2" for Tonal Interval space focu
                        "tonnetz" for harte centroids aproach. Default TIV2
                chroma: str optional
                    "chroma-samplerate-framesize-overlap"
                        chroma can be "CQT","NNLS", "STFT", "CENS" or "HPCP"
                        samplerate as a number scalar
                        frame size as a number scalar
                        overlap number that a windows is divided

                sigma: number (scalar > 0) optional
                        sigma of gaussian smoothing value. Default 11

                distance: str optional
                        type of distance measure used. Types can be "euclidean"


                Returns
                -------
                list
                        harmonic changes (the peaks) on the song detected
                list
                    HCDF function values
                number
                    windows size
        """
        centroid_changes = []
        check_parameters(chroma, blur, tonal_model, log_compresion, distance)

        name_harmonic_change = get_name_harmonic_change(name_file, hpss, tonal_m
                                                        distance)
        if path.exists(name_harmonic_change):
                dic = load_binary(name_harmonic_change)
        else:
                changes, harmonic_function, windows_size, centroid_changes = get




                dic = {'changes': changes, 'harmonic_function': harmonic_functic

                save_binary(dic, name_harmonic_change)
        return dic['changes'], dic['harmonic_function'], dic['windows_size']
```

```
def main():
        """ This program computes HCDF function focus performance on recall or preci

                Arguments
                ----------
                first one :
                    recall or f-score
                second one :
                   The name file of the audio file located in audio_files

                PRINTS
                -------
                    a list of harmonic changes (the peaks) on the song detected
                    a list with the HCDF function
                    windows size

                Typical use
                Harmonical use

        """
        absolute_path = "./audio_files/"
        file = "07_-_Please_Please_Me.wav"
        # file = argv[2]
        if argv[1] == "f-score":
                print(harmonic_change(absolute_path + file,
                                                          file,
                                                          chroma='nnls-8000-1024-2',
                                                          hpss=True,
                                                          tonal_model='TIV2',
                                                          blur='full',
                                                          sigma=5,
                                                          euclidean='euclidean'
                                                          ))

        elif argv[1] == "recall":
                print(harmonic_change(absolute_path + file,
                                                          file,
                                                          chroma='stft-44100-2048-4',
                                                          hpss=True,
                                                          tonal_model='TIV2',
                                                          blur='full',
                                                          sigma=17,
                                                          euclidean='euclidean')




if __name__ == '__main__':
        main()
```

A.2 Javascript library

A.2.1 README

```
# Harmonic Change Detection Function (HCDF) Javascript library HCDF.js

This library is used to compute HCDF. [Here]() is described the algorithm in det

## Installation

For using this library have to been added the following CDNs:
```
<script src="https://unpkg.com/essentia.js@0.0.9/dist/essentia-wasm.web.js"></sc
<script src="https://unpkg.com/essentia.js@0.0.9/dist/essentia.js-core.js"></scr
<script src="https://unpkg.com/essentia.js@0.0.9/dist/essentia.js-plot.js"></scr
```

## Usage
The library can be imported as a module with `import HCDF from './hcdf.js'` and
```

A.2.2 CODE

```
let essentia

/* "https://freesound.org/data/previews/328/328857_230356-lq.mp3"; */

let audioData;
// fallback for cross-browser Web Audio API BaseAudioContext
const AudioContext = window.AudioContext || window.webkitAudioContext;
let audioCtx = new AudioContext();
let plotChroma;
let plotContainerId = "plotDiv";

let isComputed = false;


/*
* returns an audio buffer downsampled from sample rate old_sr to sample rate new_sr
*
* Parameters
* ----------
* buffer : number > 0 [scalar]
*   audio
*
* old_sr: number > 0 [scalar]
*
*
* new_sr: number > 0 [scalar]
*
*
* hopsize: number [scalar]
*   overlap
*
* Returns
* -------
* audio downsampled
*/
function downsample(buffer, old_sr, new_sr) {
    if (new_sr == old_sr) {
        return buffer;
    }
    if (new_sr > old_sr) {
        throw "downsampling rate show be smaller than original sample rate";
    }
    var sampleRateRatio = old_sr / new_sr;
    var newLength = Math.round(buffer.length / sampleRateRatio);
    var result = new Float32Array(newLength);
    var offsetResult = 0;
    var offsetBuffer = 0;
    while (offsetResult < result.length) {
        var nextOffsetBuffer = Math.round((offsetResult + 1) * sampleRateRatio);
        var accum = 0, count = 0;
        for (var i = offsetBuffer; i < nextOffsetBuffer && i < buffer.length; i+
            accum += buffer[i];
            count++;
        }
        result[offsetResult] = accum / count;
        offsetResult++;
        offsetBuffer = nextOffsetBuffer;
    }
    return result;
}


/*
* returns nnls chromagram
*
* Parameters
* ----------
* frames : number > 0 [scalar]
*   audio
*
* sampleRate: number > 0 [scalar]
*   chroma-samplerate-framesize-overlap
*
* framesize: number [scalar]
*     frame size of windos
*
* hopsize: number [scalar]
*   overlap
*
* Returns
* -------
* list of chromagrams
*/
function chromaNNLS(frames, frameSize, hopSize, sampleRate){
  let logSpectFrames = new essentia.module.VectorVectorFloat();
  for (var i=0; i<frames.size(); i++) {
      // default hanning window (you can change it according to your need)
      let windowing = essentia.Windowing(frames.get(i), false, hopSize, 'hann');
      let spect = essentia.Spectrum(windowing.frame, frameSize); // frameSize
      let logSpectrum =  essentia.LogSpectrum(spect.spectrum,
                                             3, // bins per semitone
                                             frameSize,
                                             0, // rollon
                                             sampleRate);// sample rate

      logSpectFrames.push_back(logSpectrum.logFreqSpectrum);
      // console.log(essentia.vectorToArray(logSpectrum.logFreqSpectrum));

      meanTuning = logSpectrum.meanTuning;
      localTuning = logSpectrum.meanTuning;
  }

  let nnlsChroma = essentia.NNLSChroma(logSpectFrames,
                                       meanTuning,
                                      localTuning,
                                      "none",
                                      frameSize,
                                      sampleRate,
                                      0.7,
                                      1,
                                      "global",
                                      false).chromagram;

  delete windowing;
  delete spect;
  delete logSpectrum;
  console.log(nnlsChroma);
  var chroma_list = [];
  for (var i = 0; i < nnlsChroma.size(); i++){
      console.log(essentia.vectorToArray(nnlsChroma.get(i)));
      chroma_list.push(essentia.vectorToArray(nnlsChroma.get(i)));
  }
  return chroma_list;
}


function everything_is_zero(vector){
  let is_zero = true;
  for (var i = vector.length - 1; i >= 0 && is_zero; i--) {
     if (vector[i] !== 0){
        is_zero = false;
     }
  }
  return is_zero;
}


/**
 * Discrete Fourier Transfrom
 */
function DFT(input, zero = 1e-10) {
  // Discrete Fourier Transform
  const N = input.length;
  const signals = [];
  // Each discrete frecuenciy
  for (let frequency = 0; frequency < N; frequency += 1) {
    //complex(frequencySignal)
    let frequencySignal_re = 0;
    let frequencySignal_im = 0;
    // Each discrete time
    for (let timer = 0; timer < N; timer += 1) {
      const amplitude = input[timer];

      //rotation angle.
      const angle = -1 * (2 * Math.PI) * frequency * (timer / N);

      // Remember that e^ix = cos(x) + i * sin(x);

      let point_re = Math.cos(angle) * amplitude;
      let point_im = Math.sin(angle) * amplitude;
      // Add this data point's contribution.
      frequencySignal_re += point_re;
      frequencySignal_im += point_im;
    }

    // If is close to zero.... zero
    if (Math.abs(frequencySignal_re) < zero) {
      frequencySignal_re = 0;
    }
    if (Math.abs(frequencySignal_im) < zero) {
      frequencySignal_im = 0;
    }

    // Average contribution at this frequency.
    // complex(frecuencySignal) / N
    frequencySignal_re = (frequencySignal_re * N) / (N*N);
    frequencySignal_im = (frequencySignal_im * N) / (N*N);

    // Add current frequency signal to the list of compound signals.
    signals.push(frequencySignal_re);
    signals.push(frequencySignal_im);

  }

  return signals;
}


function division(vector, energy){
  for (var i = vector.length - 1; i >= 0; i--) {
    vector[i] = vector[i]/energy;
  }
  return vector;
}


function multiply(vectorA, vectorB){
    var ans = new Array(12);
    for (var i = vectorA.length - 1; i >= 0; i--) {
        ans[i] = vectorA[i] * vectorB[i]
    }
    return ans;
}

function TIV(pcp, weights){
  // Tonal Interval Vectors
  let fft = DFT(pcp);
  let energy = fft[0];
  let vector = fft.slice(2, 14);
  if (weights === "symbolic"){
      let weights_symbolic = [2, 2, 11, 11, 17, 17, 16, 16, 19, 19, 7, 7]
      vector = multiply(division(vector, energy), weights_symbolic);
  }
  else if (weights === "audio"){
      let weights_audio = [3, 3, 8, 8, 11.5, 11.5, 15, 15, 14.5, 14.5, 7.5,7.5];
      vector = multiply(division(vector, energy), weights_audio);
  }
  else if (weights === "harte"){
      let weithts_harte = [0, 0, 0, 0, 1, 1, 0.5, 0.5, 1, 1, 0, 0];
      vector = multiply(division(vector, energy), weithts_harte);
  }
  return vector;
}

/*
* returns tonal interval space from a vector of chromagrams
*
* Parameters
* ----------
* chroma : list
*  list of chromagrams
*
* weights: str
*  "audio", "symbolic" or "harte"
*
* Returns
* -------
* list of tonal interval space vectors
*/
function tonal_interval_space(chroma, weights="audio"){
  // Tonal Interval Space
  let centroid_vector = [];
  for (var i = 0; i < chroma.length; i++){
    let each_chroma = chroma[i];

    let centroid = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    if (!everything_is_zero(each_chroma)){
        centroid = TIV(each_chroma, weights)
    }
    centroid_vector.push(centroid);

  }
  return centroid_vector;
}


function avg (v) {
  return v.reduce((a,b) => a+b, 0)/v.length;
}


/*
* Apply gaussian smoothing to tonal model centroids
* Parameters
* ----------
* vector: list
*   tonal centroids of the tonal model
* sigma: number (scalar > 0) optional
*   sigma of gaussian smoothing value.
* Returns
* -------
* list
*   centroids blurred by gassuian smoothing
*/
function gaussian_smoothing_vector(vector, sigma) {
  var t_avg = avg(vector)*sigma;
  var ret = Array(vector.length);
  for (var i = 0; i < vector.length; i++) {
    (function () {
      var prev = i>0 ? ret[i-1] : vector[i];
      var next = i<vector.length ? vector[i] : vector[i-1];
      ret[i] = avg([t_avg, avg([prev, vector[i], next])]);
    })();
  }
  return ret;
}
```

```
function gaussian_smoothing(tis, sigma){
  var ans = [];
  for (var i = tis.length - 1; i >= 0; i--) {
      ans.push(gaussian_smoothing_vector(tis[i], sigma));
  }
  return ans;
}


/*
* Returns the quantity of centroids per second

*   Parameters
*   ----------
*   centroids : list of floats
*       The file location of the spreadsheet
*   Returns
*   -------
*   float
*       centroids per second
*/
function distance(centroids){
    var ans = [0];
    for (var i = 1; i < centroids.length - 1; i++) {
        var sum = 0;
        for (var j = 1; j < centroids[i].length - 1; j++) {
            sum += Math.pow((centroids[i][j + 1] - centroids[i][j - 1]), 2)
        }
        sum = Math.sqrt(sum)
        ans.push(sum);
    }
    return ans;
}


/*
* Returns the quantity of centroids per second
*
* Parameters
* ----------
* y : list of floats
*     The file location of the spreadsheet
* sr : bool
*     A flag used to print the columns to the console (default is False)
*
* Returns
* -------
* float
*     centroids per second
*/
function centroids_per_second(y, sr, centroids){
  return sr * centroids.length / y.length;
}


function peaks(hcdf_function, rate_centroids_second){
    let changes = [0];
    for (var i = 0; i < hcdf_function.length; i++) {
      if (hcdf_function[i - 1] < hcdf_function[i] && hcdf_function[i + 1] < hcdf_func
          changes.push(i / rate_centroids_second)
      }
    }
    return changes;
}


/*
* Computes Harmonic Change Detection Function

* Parameters
* ----------
* id_audio: str
*     id of HTML element <audio>

* Returns
* -------
* list
*   harmonic changes (the peaks) on the song detected
*/
export async function HCDF(id_audio) {
```

```
let audioURL = document.getElementById(id_audio).currentSrc;
console.log(audioURL);

// load audio file from an url
let audioData = await essentia.getAudioChannelDataFromURL(audioURL, audioCtx,

if (isComputed) { plotChroma.destroy(); };

const frameSize = 2048;
const hopSize = 512;
const sampleRate = 8000;

console.log("audio antes downsampling", audioData);
audioData = downsample(audioData, 44100, sampleRate);
console.log("audio despues downsampling", audioData);
let frames = essentia.FrameGenerator(audioData,
                                    frameSize,
                                    hopSize)


let chroma = chromaNNLS(frames, frameSize, hopSize, sampleRate);
console.log("chroma", chroma);
let chroma = [[0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0]]

let tonal_centroids = tonal_interval_space(chroma, "symbolic");
console.log("tonal centroids", tonal_centroids);

let smoothed_centroids = gaussian_smoothing(tonal_centroids, 5);
console.log("gaussian smoothing", tonal_centroids);

let harmonic_function = distance(smoothed_centroids);
console.log("distance", distance);

let cps = centroids_per_second(audioData, sampleRate, smoothed_centroids);
let harmonic_changes = peaks(harmonic_function, cps);
console.log("harmonic_changes", harmonic_changes);

return await harmonic_changes;
}


/*
* Function for loading essentia wasm module
*
*/
export async function loadEssentia(){
  // Now let's load the essentia wasm back-end, if so create UI elements for computi
  EssentiaModule().then(async function(WasmModule) {
      essentia = new Essentia(WasmModule);
  });
};


export default {HCDF, loadEssentia}
```

References

[1] Mark A Bartsch and Gregory H Wakefield. Audio thumbnailing of popular music using chroma-based representations. *IEEE Transactions on multimedia*, 7(1):96–104, 2005.

[2] Juan Pablo Bello and Jeremy Pickens. A robust mid-level representation for harmonic content in music signals. In *ISMIR*, volume 5, pages 304–311. Citeseer, 2005.

[3] Bruce Benward. *Music in Theory and Practice Volume 1*, volume 1. McGraw-Hill Higher Education, 2014.

[4] Gilberto Bernardes, Diogo Cocharro, Marcelo Caetano, Carlos Guedes, and Matthew EP Davies. A multi-level tonal interval space for modelling pitch relatedness and musical consonance. *Journal of New Music Research*, 45(4):281–294, 2016.

[5] Gilberto Bernardes, Diogo Cocharro, Carlos Guedes, and Matthew EP Davies. Harmony generation driven by a perceptually motivated tonal interval space. *Computers in Entertainment (CIE)*, 14(2):1–21, 2016.

[6] Gilberto Bernardes, Matthew EP Davies, and Carlos Guedes. Automatic musical key estimation with adaptive mode bias. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 316–320. IEEE, 2017.

[7] Gilberto Bernardes, Matthew EP Davies, and Carlos Guedes. A hierarchical harmonic mixing method. In *International Symposium on Computer Music Multidisciplinary Research*, pages 151–170. Springer, 2017.

[8] Nicola Bernardini and Giovanni De Poli. The sound and music computing field: present and future. *Journal of New Music Research*, 36(3):143–148, 2007.

[9] Sebastian Böck, Filip Korzeniowski, Jan Schlüter, Florian Krebs, and Gerhard Widmer. Madmom: A new python audio and music signal processing library. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 1174–1178, 2016.

[10] Dmitry Bogdanov, Joan Serra, Nicolas Wack, and Perfecto Herrera. From low-level to high-level: Comparative study of music similarity measures. In *2009 11th IEEE International Symposium on Multimedia*, pages 453–458. IEEE, 2009.

[11] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez Gutiérrez, Sankalp Gulati, Herrera Boyer, Oscar Mayor, Gerard Roma Trepat, Justin Salamon, José Ricardo Zapata González, Xavier Serra, et al. Essentia: An audio analysis library for music information retrieval. In *Britto A, Gouyon F, Dixon S, editors. 14th Conference of the International Society for Music Information Retrieval (ISMIR); 2013 Nov 4-8; Curitiba, Brazil.[place unknown]: ISMIR; 2013. p. 493-8.* International Society for Music Information Retrieval (ISMIR), 2013.

[12] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[13] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, pages 335–340. Citeseer, 2013.

[14] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[15] Judith C Brown and Miller S Puckette. An efficient algorithm for the calculation of a constant q transform. *The Journal of the Acoustical Society of America*, 92(5):2698–2701, 1992.

[16] Emilios Cambouropoulos. From midi to traditional musical notation. In *Proceedings of the AAAI Workshop on Artificial Intelligence and Music: Towards Formal Models for Composition, Performance and Analysis*, volume 30, 2000.

[17] Soubhik Chakraborty, Guerino Mazzola, Swarima Tewari, and Moujhuri Patra. *Computational musicology in Hindustani music*. Springer, 2014.

[18] Ruofeng Chen, Weibin Shen, Ajay Srinivasamurthy, and Parag Chordia. Chord recognition using duration-explicit hidden markov models. In *ISMIR*, pages 445–450. Citeseer, 2012.

[19] Elaine Chew. The spiral array: An algorithm for determining key boundaries. In *International Conference on Music and Artificial Intelligence*, pages 18–31. Springer, 2002.

[20] Eric Clarke and Nicholas Cook. *Empirical musicology: Aims, methods, prospects*. Oxford University Press, 2004.

[21] Alessio Degani, Marco Dalai, Riccardo Leonardi, and Pierangelo Migliorati. Harmonic change detection for musical chords segmentation. In *2015 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2015.

[22] Dan Ellis. Chroma Feature Analysis and Synthesis. Available at https://labrosa.ee.columbia.edu/matlab/chroma-ansyn/, accessed April 18, 2020.

[23] Leonhard Euler. *Tentamen novae theoriae musicae ex certissismis harmoniae principiis dilucide expositae*. Saint Petersburg Academy, 1739.

[24] Derry Fitzgerald. Harmonic/percussive separation using median filtering. In *Proc. of DAFX*, volume 10, 2010.

[25] Takuya Fujishima. Real-time chord recognition of musical sound: A system using common lisp music. *Proc. ICMC, Oct. 1999*, pages 464–467, 1999.

[26] Barbara R Gaizauskas. The harmony of the spheres. *Journal of the Royal Astronomical Society of Canada*, 68:146, 1974.

[27] Emilia Gómez. Tonal description of music audio signals. *Department of Information and Communication Technologies*, 2006.

[28] Stephen W Hainsworth, Malcolm D Macleod, et al. Onset detection in musical audio signals. In *ICMC*, 2003.

[29] Christopher Harte. *Towards automatic extraction of harmony information from music signals*. PhD thesis, 2010.

[30] Christopher Harte and Mark Sandler. Automatic chord identifcation using a quantised chromagram. In *Audio Engineering Society Convention 118*. Audio Engineering Society, 2005.

[31] Christopher Harte, Mark Sandler, and Martin Gasser. Detecting harmonic change in musical audio. In *Proceedings of the 1st ACM workshop on Audio and music computing multimedia*, pages 21–26, 2006.

[32] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[33] Eric J Humphrey and Juan P Bello. Rethinking automatic chord recognition with convolutional neural networks. In *2012 11th International Conference on Machine Learning and Applications*, volume 2, pages 357–362. IEEE, 2012.

[34] Eric J Humphrey, Taemin Cho, and Juan P Bello. Learning a robust tonnetz-space transform for automatic chord recognition. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 453–456. IEEE, 2012.

[35] Elyor Kodirov, Sejin Han, Guee-Sang Lee, and YoungChul Kim. Music with harmony: Chord separation and recognition in printed music score images. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '14, New York, NY, USA, 2014. Association for Computing Machinery.

[36] Hendrik Vincent Koops, W Bas de Haas, John Ashley Burgoyne, Jeroen Bransen, and Anja Volk. Harmonic subjectivity in popular music, 2017.

[37] Filip Korzeniowski and Gerhard Widmer. Feature learning for chord recognition: The deep chroma extractor. *arXiv preprint arXiv:1612.05065*, 2016.

[38] Mathieu Lagrange, Graham Percival, and George Tzanetakis. Adaptive harmonization and pitch correction of polyphonic audio using spectral clustering. In *Proceedings of DAFx*, pages 1–4, 2007.

[39] Bangalore S Manjunath, Philippe Salembier, and Thomas Sikora. *Introduction to MPEG-7: multimedia content description interface*. John Wiley & Sons, 2002.

[40] Matthias Mauch, Chris Cannam, Matthew Davies, Simon Dixon, Christopher Harte, Sefki Kolozali, Dan Tidhar, and Mark Sandler. Omras2 metadata project 2009. In *Proc. of 10th International Conference on Music Information Retrieval*, page 1, 2009.

[41] Matthias Mauch and Simon Dixon. Simultaneous estimation of chords and musical context from audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1280–1289, 2009.

[42] Matthias Mauch and Simon Dixon. Approximate note transcription for the improved identification of difficult chords. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010.

[43] HJJ MAXWELL. An artificial intelligence approach to computer-implemented analysis of harmony in tonal music. 1986.

[44] Guerino Mazzola. *The topos of music: geometric logic of concepts, theory, and performance*. Birkhäuser, 2012.

[45] Josh H McDermott and Andrew J Oxenham. Music perception, pitch, and the auditory system. *Current opinion in neurobiology*, 18(4):452–463, 2008.

[46] Brian McFee and Juan Pablo Bello. Structured training for large-vocabulary chord recognition. In *ISMIR*, pages 188–194, 2017.

[47] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. 2015.

[48] David Meredith. *Computational music analysis*, volume 62. Springer, 2016.

[49] Dalibor Mitrović, Matthias Zeppelzauer, and Christian Breiteneder. Features for content-based audio retrieval. In *Advances in computers*, volume 78, pages 71–150. Elsevier, 2010.

[50] Philip Morrison. The new grove dictionary of music and musicians, 1981.

[51] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*, chapter Section 3.1.2.3 and Section 7.2.1. Springer, 2015.

[52] Meinard Müller and Sebastian Ewert. Chroma Toolbox: MATLAB implementations for extracting variants of chroma-based audio features. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR)*, Miami, USA, 2011, to appear.

[53] Meinard Müller, Frank Kurth, and Michael Clausen. Audio matching via chroma-based statistical features. In *ISMIR*, volume 2005, page 6th, 2005.

[54] Yizhao Ni, Matt McVicar, Raul Santos-Rodriguez, and Tijl De Bie. Understanding effects of subjectivity in measuring chord estimation accuracy. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(12):2607–2615, 2013.

[55] Henri J Nussbaumer. The fast fourier transform. In *Fast Fourier Transform and Convolution Algorithms*, pages 80–111. Springer, 1981.

[56] H. Nyquist. Certain topics in telegraph transmission theory. *Proceedings of the IEEE*, 90:280 – 305, 03 2002.

[57] Arthur Joachim von Oettingen. Das duale harmoniesystem [the dual harmony system], 1913.

[58] Johan Pauwels, Ken O'Hanlon, Emilia Gómez, Mark Sandler, et al. 20 years of automatic chord recognition from audio. Ismir, 2019.

[59] Johan Pauwels and Geoffroy Peeters. Evaluating automatically estimated chord sequences. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 749–753. IEEE, 2013.

[60] Geoffroy Peeters, Bruno L Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916, 2011.

[61] João Paulo Caetano Pereira, Gilberto Bernardes, and Rui Penha. Musikverb: A harmonically adaptive audio reverberation. In *Proceedings of the 21st International Conference on Digital Audio Effects*, Aveiro, Portugal, 2018.

[62] Pietro Polotti. *Sound to Sense, Sense to Sound: A state of the art in Sound and Music Computing*. Logos Verlag Berlin GmbH, 2008.

[63] Ebenezer Prout. *Analytical Key to the Exercises in Harmony: Its Theory and Practice*. Augener & Company, 1903.

[64] Miller Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the second intercollege computer music concerts*, pages 37–41, 1996.

[65] Stanisław A Raczyński, Emmanuel Vincent, and Shigeki Sagayama. Dynamic bayesian networks for symbolic polyphonic pitch modeling. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(9):1830–1840, 2013.

[66] Bernardes G. Davies M.E.P. Serra X. (2020) Ramires, A. Tiv.lib: An open-sounce library for the tonal description of musical audio. international conference on digital audio effects. vienna, austria. 2020.

[67] Hugo Riemann. *Skizze einer neuen Methode der Harmonielehre*. Leipzig, 1880.

[68] Hugo Riemann. *Handbuch der Harmonielehre*. Leipzigy, 1887.

[69] Hugo Riemann. *Vereinfachte Harmonielehre*. London/New York, 1893.

[70] Hugo Riemann. *Harmony simplified, or the theory of the tonal functions of chords*. Augener Ltd., 1896.

[71] Hugo Riemann and John Comfort Fillmore. *The nature of harmony...* Presser, 1882.

[72] Ricardo Scholz, Emmanuel Vincent, and Frédéric Bimbot. Robust modeling of musical chord sequences using probabilistic n-grams. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 53–56. IEEE, 2009.

[73] Xavier Serra. Audio-aligned jazz harmony dataset for automatic chord transcription and corpus-based research. In *In: Gómez E, Hu X, Humphrey E, Benetos E. Proceedings of the 19th ISMIR Conference; 2018 Sep 23-27; Paris, France.[Canada]: ISMIR; 2018. p. 483-90.* International Society for Music Information Retrieval (ISMIR), 2018.

[74] Xavier Serra, Roberto Bresin, and Antonio Camurri. Sound and music computing: Challenges and strategies. *Journal of New Music Research*, 36(3):185–190, 2007.

[75] Xavier Serra, Marc Leman, and Gerhard Widmer. A roadmap for sound and music computing. 2007.

[76] Alexander Sheh and Daniel PW Ellis. Chord segmentation and recognition using em-trained hidden markov models. 2003.

[77] Roger N Shepard. Geometrical approximations to the structure of musical pitch. *Psychological review*, 89(4):305, 1982.

[78] Clifford W Shults, David Oakes, Karl Kieburtz, M Flint Beal, Richard Haas, Sandy Plumb, Jorge L Juncos, John Nutt, Ira Shoulson, Julie Carter, et al. Effects of coenzyme q10 in early parkinson disease: evidence of slowing of the functional decline. *Archives of neurology*, 59(10):1541–1550, 2002.

[79] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12):1349–1380, 2000.

[80] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*. W3K Publishing, http://www.w3k.org/books/, 2007.

[81] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

[82] Robert J Turetsky and Daniel PW Ellis. Ground-truth transcriptions of real music from force-aligned midi syntheses. 2003.

[83] Dmitri Tymoczko. *A geometry of music: Harmony and counterpoint in the extended common practice*. Oxford University Press, 2010.

[84] Dmitri Tymoczko and Jason Yust. Fourier phase and pitch-class sum. In *International Conference on Mathematics and Computation in Music*, pages 46–58. Springer, 2019.

[85] Yushi Ueda, Yuki Uchiyama, Takuya Nishimoto, Nobutaka Ono, and Shigeki Sagayama. Hmm-based approach for automatic chord detection using refined acoustic features. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5518–5521. IEEE, 2010.

[86] Gerhard Widmer, Davide Rocchesso, Vesa Välimäki, Cumhur Erkut, Fabien Gouyon, Daniel Pressnitzer, Henri Penttinen, Pietro Polotti, and Gualtiero Volpe. Sound and music computing: Research trends and some key issues. *Journal of New Music Research*, 36(3):169–184, 2007.

[87] Terry Winograd. Linguistics and the computer analysis of tonal harmony. *journal of Music Theory*, 12(1):2–49, 1968.

[88] Yiming Wu and Wei Li. Automatic audio chord recognition with midi-trained deep feature and blstm-crf sequence decoding model. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(2):355–366, 2018.

[89] Furkan Yesiler, Joan Serrà, and Emilia Gómez. Accurate and scalable version identification using musically-motivated embeddings. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 21–25. IEEE, 2020.

[90] Junping Zhang, Fei-Yue Wang, Kunfeng Wang, Wei-Hua Lin, Xin Xu, and Cheng Chen. Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1624–1639, 2011.