Escuela Politécnica Superior de Linares

Telecommunication Engineering Master

Universidad de Jaén

Escuela Politécnica Superior de Linares

Master Thesis

EVALUATION OF THE TEMPO OF A MUSICAL PERFORMANCE USING MUSIC/SCORE ALIGNMENT WITH A DATABASE OF INTERPRETATIONS FOR SAME WORK

Student: **Raúl Carrillo Alguacil**

Supervisors: Prof. D. Pedro Vera Candeas
Prof. D. Julio José Carabias Orti

Department: Telecommunication Engineering

Junio, 2020

A mi familia, por creer siempre en mí.

Agradecimientos

Este Trabajo Fin de Máster, basado en el desarrollo de un algoritmo de alineamiento y estimación de estadísticas musicales para comparar entre interpretaciones, ha supuesto una gran motivación para mí. Principalmente, ha hecho que me afiance en la idea de que el camino que escogí en su momento, de estudiar y ser Ingeniero de Telecomunicación, es el correcto y, por otro lado, me ha hecho darme cuenta de que trabajaré en lo que me gusta. También, me ha demostrado, una vez más, que la música es una de mis fuentes de inspiración y trabajar en este ámbito junto a las áreas de imagen y sonido y procesado de señal de audio es lo que me apasiona.

Mi agradecimiento profundo y sincero al Dr. Pedro Vera Candeas, mi tutor en este TFM, por sus aportaciones, sus consejos, su día a día,… En definitiva, agradecerle la cantidad de tiempo invertido y su implicación para que este trabajo tenga sentido. Sus conocimientos y sus explicaciones han sido muy importantes para llevarlo a cabo durante todo este tiempo. Sin sus brillantes ideas, su enfoque profesional y su claridad mental este trabajo no habría sido posible.

También quiero agradecer su ayuda al Dr. Julio José Carabias Orti, mi otro tutor en este TFM. Ha sido todo un placer haber podido recibir sus consejos y aportaciones antes de embarcarnos en este proyecto y durante el mismo.

Quiero también agradecer a D. Pablo Cabañas Molero sus aportes técnicos, sus conocimientos y opinión sobre el tema cuando la he necesitado. Muy agradecido también a Antonio J. Muñoz Montoro por su ayuda, conocimientos y opiniones que han servido para que este trabajo salga adelante. Agradecer también su disposición y aportaciones a Juan de la Torre Cruz.

Todo esto no hubiera sido posible sin la constancia y trabajo que me llevó hasta aquí y, sobretodo, sin los compañeros que tuve la suerte de encontrarme en el camino. Por ello, agradecer todos los momentos vividos a mis compañeros: Andrés, Maier y Antonio.

Agradecer al Departamento de Ingeniería de Telecomunicación de la Universidad de Jaén, por el apoyo y las facilidades prestadas durante este tiempo, a la Escuela Politécnica Superior de Linares y a todo el profesorado por haberme inculcado los conocimientos en Ingeniería de Telecomunicación.

Raúl Carrillo Alguacil

Junio 2020

Acknowledgements

This Master's Thesis, based on the development of an algorithm of alignment and estimation of musical statistics to compare between performances, has been a great motivation for me. Mainly, it has made me consolidate in the idea that the path that I chose at the time, to study and be an Engineer in Telecommunications, is the correct one and, on the other hand, it has made me realize that I work in what I like. Also, it has shown me, once again, that music is one of my sources of inspiration and working in this area together with the areas of image and sound and audio signal processing is what I am passionate about.

My deep and sincere thanks to Dr. Pedro Vera Candeas, my tutor in this Master's Thesis, for his contributions, his advice, his day to day… In short, thank him for the amount of time invested and his involvement in making this work make sense. His knowledge and explanations have been very important to carry it out during all this time. Without his brilliant ideas, his professional approach and his mental clarity this work would not have been possible.

I also want to thank Dr. Julio José Carabias Orti for his help. It has been a pleasure to have been able to receive his advice and contributions before working on this project and during it.

I also want to thank Pablo Cabañas Molero for his technical contributions, his knowledge and opinion on the subject when I have needed it. Also very grateful to Antonio J. Muñoz Montoro for his help, knowledge and opinions that have helped this work to go ahead. Also thank their willingness and contributions to Juan de la Torre Cruz.

All this would not have been possible without the perseverance and work that led me here and without the colleagues who were lucky enough to find me on my way. For this, thank all my colleagues for their moments: Andrés, Maier and Antonio.

Thanks to the Department of Telecommunications Engineering of the University of Jaén, for the support and facilities provided during this time, the Higher Polytechnic School of Linares and all the teaching staff for having instilled in me the knowledge in Telecommunications Engineering.

Raúl Carrillo Alguacil

June 2020

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMT	Automatic Music Transcription
CHARM	Center for the History and Analysis of Recorded Music
DTW	Dynamic Time Warping
EUC	Euclidean Distance Divergence
FFT	Fast Fourier Transform
FT	Fourier Transform
GT	Ground-Truth
GUI	Graphical User Interface
HMM	Hidden Markov Model
IS	Itakura Saito Divergence
ISMIR	International Society for Music Information Retrieval
ISS	Informed Source Separation
KL	Kullback-Leibler Divergence
LTI	Linear and Time-Invariant
MIDI	Musical Instruments Digital Interface
MIR	Music Information Retrieval
MIREX	Music Information Retrieval Evaluation eXchange
MPO	Musical Plus One
NMF	Non-negative Matrix Factorization
RWC	Real World Computing Database
SPL	Sound Pressure Level
STFT	Short-Time Fourier Transform
URMP	University of Rochester Multi-modal Music Performance

LIST OF FIGURES

LIST OF TABLES

1 Resumen

1.1 Resumen

A lo largo de la historia, aunque la historia del procesado de señales musicales es relativamente corta, han existido muchos investigadores que se han dedicado al estudio de este campo de la música. Un campo de la investigación, bajo mi punto de vista, poco explotado hasta la fecha con respecto a la cantidad de sistemas que se pueden integrar. A día de hoy, se pueden desarrollar aplicaciones para hacer más fácil el estudio de la música o incluso desde el punto de visto turístico, aplicaciones que muestren información a cada persona sobre lo que está tocando una orquesta en un concierto o lo que el compositor quiere transmitir en cada momento. Aunque el ámbito de la música está todavía por explotar, es importante indicar que es un área un poco tediosa, ya que necesita invertir gran cantidad de tiempo a la vez que requiere potencia en los procesadores sobre los que se ejecuta una determinada aplicación.

Este trabajo se basa en la hipótesis de que los instrumentos que se usan para interpretar una determinada pieza musical son armónicos, es decir, no tratamos instrumentos percusivos. Estos últimos, son los instrumentos más difíciles de analizar y los métodos actuales no trabajan con ellos, aunque algunos investigadores hayan obtenido resultados con algunos algoritmos.

Siendo esto así, el objetivo de este Trabajo Fin de Máster es la estimación de estadísticas musicales como tiempo de beat, tempo y duración de las notas musicales aplicadas al desarrollo de un sistema de comparación entre músicos. El objetivo final del trabajo es desarrollar una aplicación que permita al usuario analizar la grabación de su actuación y compararse con otro músico de la base de datos o con él mismo si así lo desea. Nuestra aplicación se basa, principalmente, en cuatro etapas: entrenamiento (preprocesado), alineamiento, extracción de estadísticas musicales y comparación.

En este trabajo, proponemos un método de alineamiento basado en Carabias y otros [22], que permite estimar el camino óptimo, es decir, el alineamiento entre tiempo real y tiempo MIDI con una función de coste variable guiada por una constante de regularización (λ). Además, se han desarrollo el algoritmo que permite extraer las estadísticas musicales a partir del camino óptimo y un sistema de comparación entre interpretaciones. El desarrollo del algoritmo de alineamiento se ha llevado a cabo en MatLab, para pruebas, y en C, para la aplicación final. Y los demás módulos que componen la aplicación están desarrollados en MatLab.

La aplicación, desarrollada con la herramienta GUIDE de MatLab y que está compuesta por dos interfaces, trabaja principalmente con dos bases de datos: **URMP** y **Proyecto Mazurka**. Sin embargo, la base de datos que compone la aplicación es dinámica e interactiva con el usuario. Esto es así porque la aplicación permite al músico insertar en la base de datos tantas grabaciones, de las piezas musicales correspondientes a las dos bases de datos mencionadas, como quiera.

1.2 Abstract

Throughout history, although the history of audio signal processing is relatively short, there have been many researchers who have dedicated themselves to the study of this field of music. A field of research, in my view, little exploited to date regarding the number of systems that can be integrated. Today, applications can be developed to make the study of music easier or even from a tourist point of view, applications that show information to each person about what an orchestra is playing at a concert or what the composer wants to transmit in every moment. Although the field of music is still to be exploited, it is important to indicate that it is a somewhat tedious area, it needs to invest a large amount of time while requiring power in the processors on which a certain application runs.

This work is based on the hypothesis that the instruments used to interpret a certain musical piece are harmonic, which is, we do not treat percussive instruments. The latter are the most difficult instruments to analyze and current methods do not work with them, although some researchers have obtained results with some algorithms.

This being so, the objective of this Master's Thesis is the estimation of musical statistics such as beat time, tempo and duration of the musical notes applied to the development of a comparison system between musicians. The final objective of the work is to develop an application that allows the user to analyze the recording of his performance and compare himself with another musician in the database or with himself if he so wishes. Our application is mainly based on four stages: training (pre-processed), alignment, extraction of music statistics and comparison.

In this work, we propose an alignment method based on Carabias et al. [22], which allows estimating the optimal path, that is, the alignment between real time and MIDI time with a variable cost function guided by a regularization constant (λ). In addition, the algorithm that allows extracting musical statistics from the optimal path and a comparison system between performances has been developed. The development of the alignment algorithm has been carried out in MatLab, for tests, and in C, for the final application. And the other modules that make up the application are developed in MatLab.

The application, which has been developed with MatLab's GUIDE tool and which consists of two interfaces, works mainly with two databases: URMP and Mazurka Project. However, the database that makes up the application is dynamic and interactive with the user. This is so because the application allows the musician to insert into the database as

many recordings, of the musical pieces corresponding to the two mentioned databases, as he wants.

2 Introduction

Without a doubt, music is one of the most incredible expressions of the human being since it manages to immediately transmit sensations and feelings that other artistic forms are not capable of doing. Music is a complex system of sounds, melodies and rhythms that the human being has been discovering. Music allows the human being to express happy moments, sadness, deep feelings and others.

In this way, music is of vital importance for its beauty and aesthetic value but also as a support from which the human being can communicate with others and with himself.

The human auditory system is capable of processing and discriminating the sounds present around it. If we understand the auditory scene as a set of sounds concurrent in time, the human being can carry out an analysis of this scene and establish a priority based on the sound that most interests it.

Technology advances at a rate that is difficult to achieve, a project always arises capable of developing a service that meets a need or simply makes life easier for us. Therefore, technology is very important for human beings. It is easy to understand that if a person is able to analyze the performance of a musician, a machine can also do it with the appropriate technology.

2.1 Outline of the thesis

The document is divided into three different blocks:

- First, in ***section 2.2*** we review audio signal processing, explaining concepts and attributes, and a description of the key points of music theory.
- Later, in ***section 2.3*** the state of the art of automatic alignment (score tracking) is exposed, where the main algorithms introduced by different researchers are explained.
- In ***section 3***, we explain the different objectives and hypothesis of this work.
- In ***sections 4.1*** **and 4.2** we detail the algorithms used and the application development process. Some diagrams are also included, such as the schematic showing the relationship between the algorithm modules and the schematic with the relationship between the algorithm and the graphical interface, in order to help the reader to understand the explanation.
- In ***sections 5*** **and *6***, we show and evaluate the results obtained and the conclusions that can be drawn from them.

- Finally, in ***section 7***, we show an appendix with the user manual and, in ***section 8***, the bibliography that we have used to prepare this document.

2.2 Music Signal Processing

The methods for processing audio signals are widely used in the music field. These techniques are tools that can help the musician to improve his/her skills, increasing the ability of interpretation of a certain musical piece, or in the instrument technique, or analyzing his/her evolution.

To understand the problem and the development of the solution, we begin doing a brief review of the background about sound, music theory and audio signal processing. These are basic concepts, but are especially relevant because they are closely related to this work.

The key of this work is to analyze the audio signal, produced by the musician when interpreting a musical piece, together with the score, and from there, estimate certain musical features such as beat time, tempo and note duration, which are very interesting for improving the musician's skills. In addition, the musician can also compare with the performances of other musicians in comparative plots. But, before showing this application, we are going to explain other theoretical concepts of music, their relationship with the environment, and the most useful concepts of digital signal processing.

2.2.1 Sound concept

Sound is a mechanical vibration that is transmitted through an elastic medium (such as air), producing small pressure fluctuations (compressions and rarefactions) around the static pressure of the medium. These pressure changes generate an auditory sensation when they are picked up by the ear.

At each point in the medium by which the sound wave propagates, it is described by two fundamental physical quantities: **pressure and molecular speed** (speed of vibration of the molecules in the medium). Furthermore, like any other wave, sound is characterized by other physical parameters, such as its propagation speed, frequency, power or wavelength.

The most used measure to quantify the intensity of a sound is the **Sound Pressure Level** (*SPL*). This is a logarithmic measurement of the pressure generated by a sound (in dB), relative to a reference pressure of 20μPa (human hearing threshold), as indicated by equation (1). The use of logarithmic units instead of Pascals to quantify the sound pressure is due to the wide range of variation of the same in Pascals (range of hearing from 10^{-6} to 10^{2} Pa) [1].

$$Lp(dB_{SPL}) = 20\log\left(\frac{P_{rms}[Pa]}{20 \cdot 10^{-6}\,[Pa]}\right) \quad (1)$$

A sound wave can be considered as composed of sine functions; therefore, it can be characterized with three mathematical terms: **frequency, amplitude and phase**. In *Figure 2.1*, we can see an **example** of a sine wave.

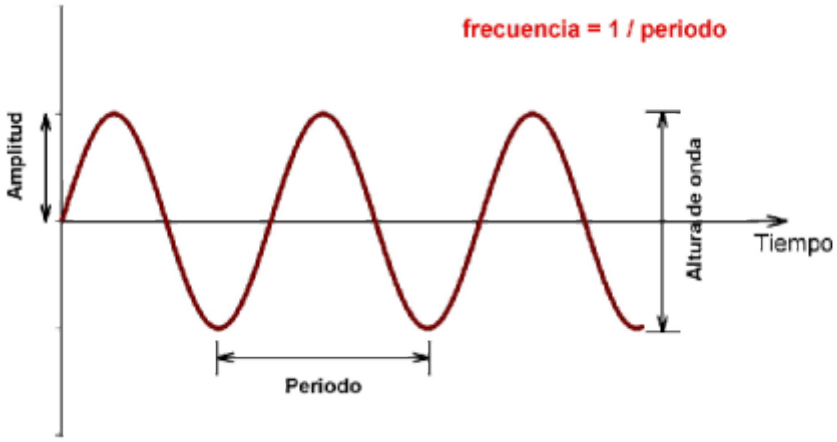


Figure 2.1 – Sound signal like a sine wave. Source https://fisica.laguia2000.com/acustica/descripcion-de-la-onda-sonora

There is a small relationship between the **physical characteristics** (frequency and amplitude) and the **perceptual properties** (pitch and loudness) as we can see in *Figures 2.2* and *2.3*, where the large amplitudes of a signal correspond to a high loudness level and high frequencies are associated with the perception of higher pitches [2].

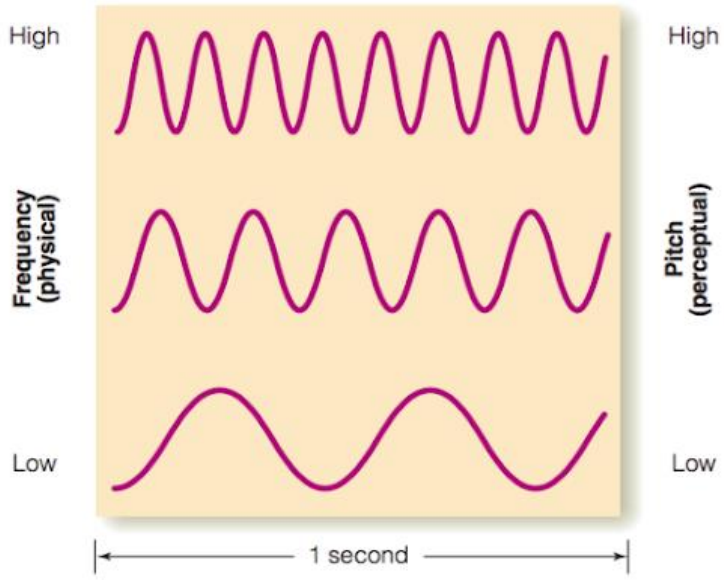


Figure 2.2 – Different frequencies of a pure tone. Source [2]

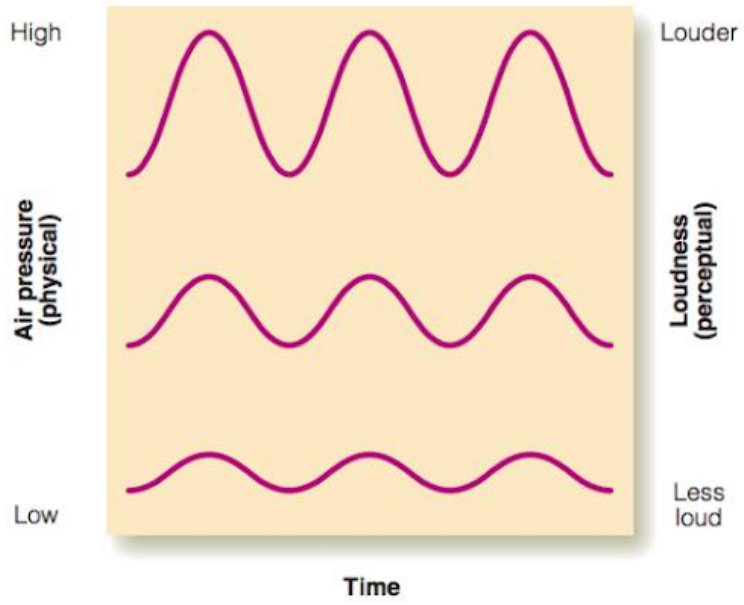


Figure 2.3 – Different amplitudes of a pure tone. Source [2]

The human ear perceives the oscillation frequency of a signal as pitch. The amplitude of the sound signal is obtained by the difference between the highest and lowest pressure that exists in the signal. The phase of a signal determines when a wave begins in time.

2.2.2 Sound attributes

Previously, the parameters that characterize a sound have been exposed, in a physical and psychoacoustic way, and we have concluded that perceiving a sound wave is not the same as observing it. There are more sound attributes, apart from those mentioned above, so, it is convenient to correctly define the parameters already explained and others that are necessary to know in this project.

2.2.2.1 Loudness

It is the attribute of the sounds that allows us to order them from stronger to weaker. Although it is directly related to the sound power, it also depends on the frequency and duration of the sound. Frequency dependency is due to the fact that the sensibility of the ear varies according to the frequency of sounds: the maximum sensibility is around 4 KHz and it changes as we move towards very low-pitched or very high-pitched sounds.

The loudness can be measured in phons, taking as reference a tone of 1 kHz. The sound pressure level of 1 kHz tone (in dB SPL) matches with its loudness in phons. Thus, a sound of n phons is perceived with the same loudness as a tone of 1 kHz and n dB SPL. The loudness according to the frequency can be estimated by means of graphs called isophonic curves, which join pairs **intensity (dB) - frequency (Hz)** that are perceived with the same loudness. *Figure 2.4* shows the isophonic curves of Fletcher and Munson. The number that appears on each curve is its loudness in phons.

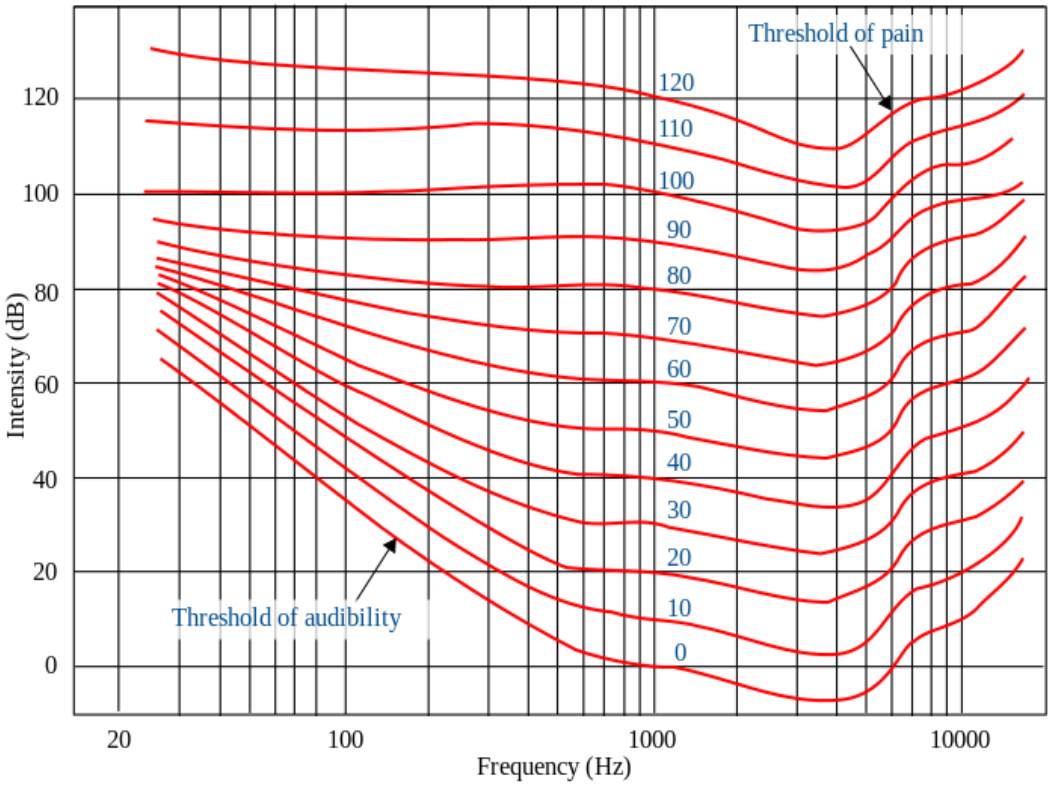


Figure 2.4 – Fletcher and Munson's isophonic curves.

A music concept related to the loudness is the **musical dynamics**. Dynamics normally refer to the loudness of a note or sound even. However, it can also refer to every aspect of the execution of a given musical piece, either stylistic (staccato, legato, etc.) or functional (velocity). In *Table 2.1* the relation between dynamics and loudness is represented.

Loudness interpretation	Notation	Dynamic indication
Very soft	pp	pianissimo
Soft	p	piano
Moderately soft	mp	mezzopiano
Moderately loud	mf	mezzoforte
Loud	f	forte
Very loud	ff	fortissimo
Forceful	sfz	sforzando
Becoming louder	<	crescendo
Becoming softer	>	decrescendo

Table 2.1 – Relation between the types of dynamics and loudness. Source [8]

2.2.2.2 Timbre

It is the **perceptual quality** of sound that allows us to recognize and discriminate different sound sources, even if they emit sounds of equal tone and loudness. Each musical instrument is characterized by a specific timbre, thanks to which we recognize the instrument when we listen to it. The timbre depends mainly on the distribution of energy in the sound spectrum (see *Figure 2.5*). Therefore, sounds with different timbre, although having the same pitch and volume, it differs in the spectral envelope and the distribution of harmonics amplitudes.

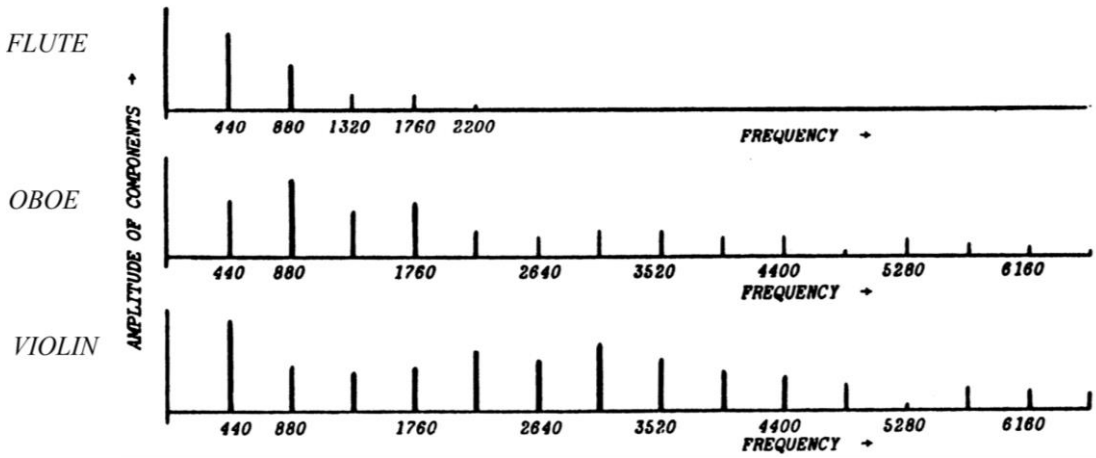


Figure 2.5 – Amplitude of the spectral energy distribution for flute, oboe and violin.

Unlike other attributes of sound, timbre cannot be defined by a single characteristic because there are multiple features that affect auditory perception, such as the degree of **inharmonicity** of the partials, the material of construction of the musical instrument (for example, wood or metal), the type of musical instrument (for example, string or brass) or the excitement produced by sound (for example, plucked or rubbed).

Among the characteristics that most affect the timbre is the number of partials, the spectral envelope, the temporal envelope, the attack time or the duration.

An example is shown in *Figure 2.6*, showing the difference, in the time and frequency domain, between sounds of two different instruments with the same duration, pitch and loudness.

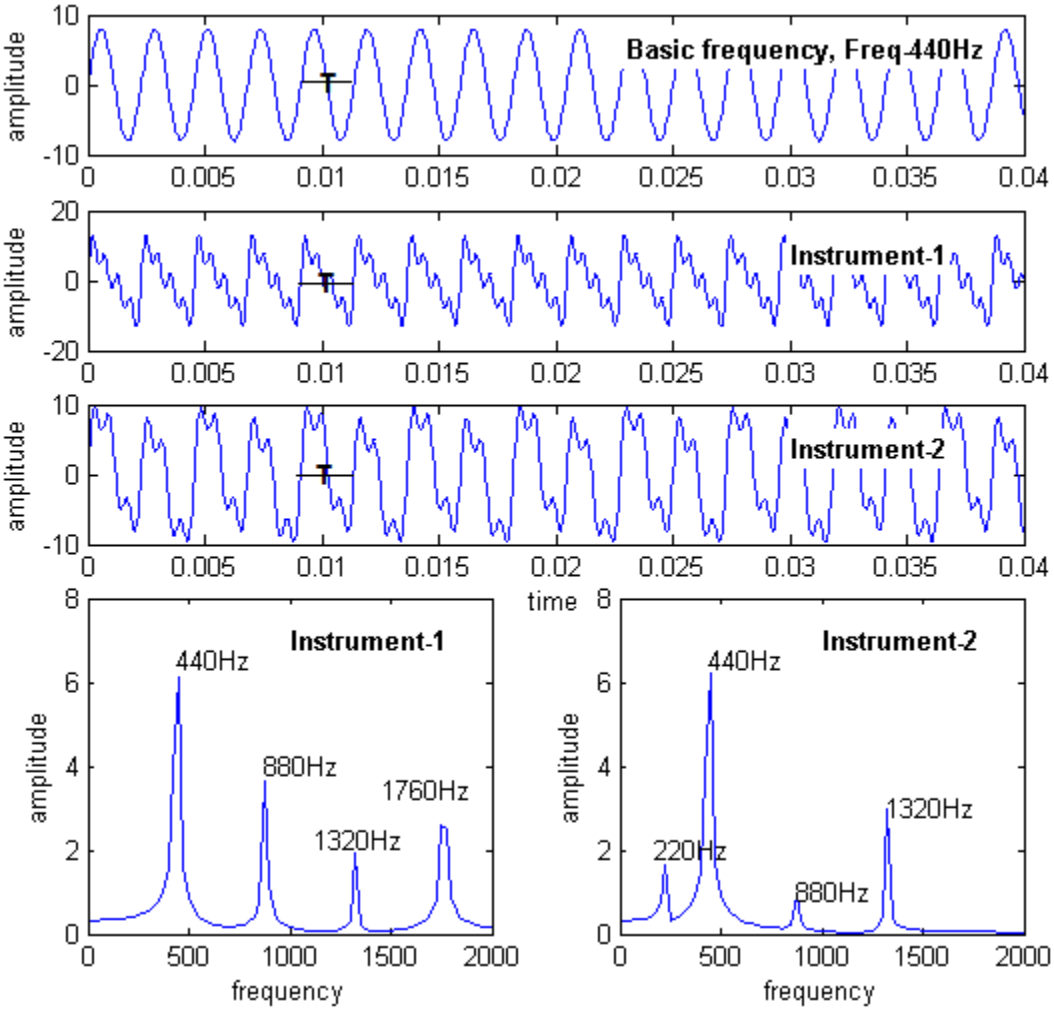


Figure 2.6 – Sound of two instruments with f_0 = 440 Hz in the time and frequency domain.

2.2.2.3 Duration

The **duration** of a sound is the time interval in which an event or note is active in the audio signal, that is, the time that elapses from the beginning of being perceived a sound until it is extinguished or ceases to be perceived.

This parameter is associated with the terms **onset** and **offset**, which indicates the instant in which the event or note begins and is no longer active respectively.

The signals, in practice, are not stationary during the time they are active, that is, they do not keep their statistical properties constant. In order to analyze stationary signals, these are usually divided into temporal fragments called **frames**, in which their characteristics are maintained (quasi-periodic signals).

2.2.2.4 *Fundamental frequency (f_0)*

When a human hears a musical note played on an instrument, he perceives it as a clear and unambiguous musical tone. That is because the instrument produces a sound pressure wave that he repeats regularly. For example, if you look at *Figure 2.7,* we have four instrument waveforms (violin, trumpet, flute and oboe) recorded with a microphone and we can see that each one of them has a different waveform that repeats regularly, that is a periodic waveform.

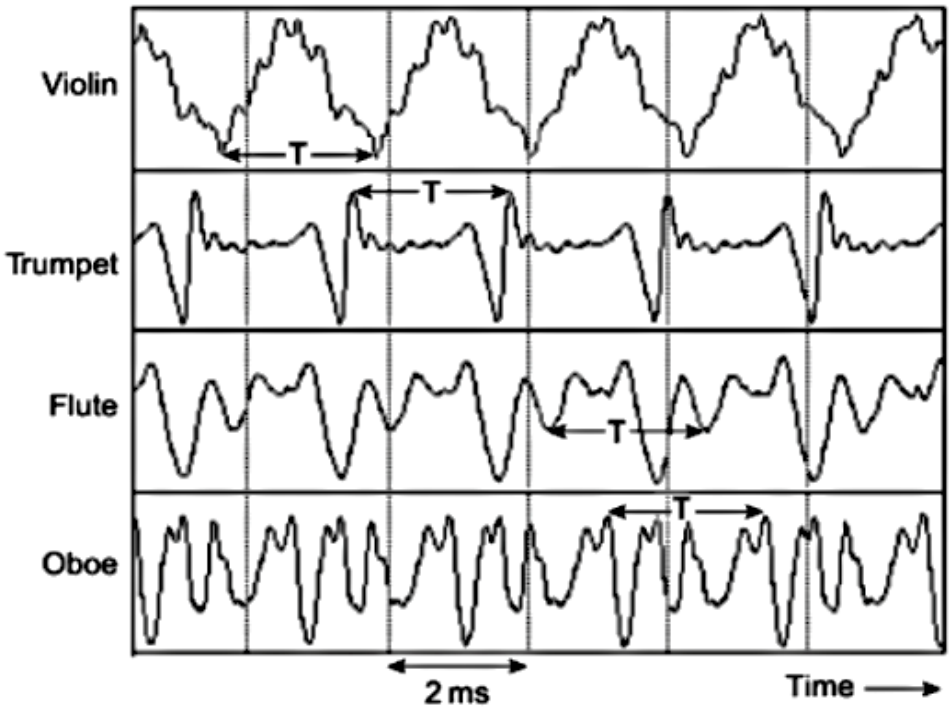


Figure 2.7 – Sound pressure wave of the note A4 (440 Hz) played on a violin, trumpet, flute and oboe.

Because of that, a signal x(t) is periodic when it is repeated from time to time. The **fundamental period (T_0)** is the smallest positive real value that equation (2) satisfies. If the signal x(t) has a fundamental period, then there are infinite periods for which x(t) is periodic (integer multiples of T_0).

$$x(t) = x(t + T_0) = x(t + nT_0), \qquad \forall t, n \in (-\infty, \infty), n \in Z \tag{2}$$

In this way, the **fundamental frequency (f_0)** is defined as a physical term of the sound that is known as the inverse of the signal period, referring to the period as the number of cycles in a second. The fundamental frequency, in Hertz, is defined as we can see in the following equation:

$$f_0[Hz] = \frac{1}{T_0[s]} \tag{3}$$

2.2.2.5 *Harmonics of the fundamental frequency*

There are other features of sound that can only be determined from the spectral point of view, **harmonicity** is one of them.

A sound is harmonic when the signal presents a spectral structure where the spectral components are spaced in the frequency domain.

According to the **Fourier Theorem**, every periodic wave can be decomposed into a sum of pure sine waves, whose frequencies are multiples of the fundamental frequency ($f_k = k * f_0$) and are called **harmonics**.

$$x(t) = \sum_{k=-\infty}^{\infty} a_k \cdot e^{j2\pi k f_0 t} \tag{4}$$

The sinusoidal signals are ideal for modeling musical signals, thanks to the property that, if at the entrance of an *LTI system* (Linear and Time-Invariant) we have a linear combination of sinusoidal signals, at the output we have the same combination scaled in complex amplitude.

In practice, the musical signals are not stationary, but if we divide them into very small temporary frames (millisecond order), we achieve them to be quasi-periodic. So, the harmonics are replaced by **"partial frequencies"** [4], which are sinusoidal components of a quasi-periodic signal whose frequencies may deviate slightly from the integer multiples of the fundamental frequency.

$$x(t) = \sum_{p=1}^{M} A_p \cdot \cos(2\pi f_p t + \varphi_p) \quad (5)$$

The characteristic spectrum of the musical signals presents peaks in the fundamental frequency and in its multiples (harmonic or partial). In *Figure 2.8* we can see the waveform and the spectrum of the tuning-fork, flute, violin and the human voice. In the spectrogram of the signals we see how the different harmonics are spaced in frequency.

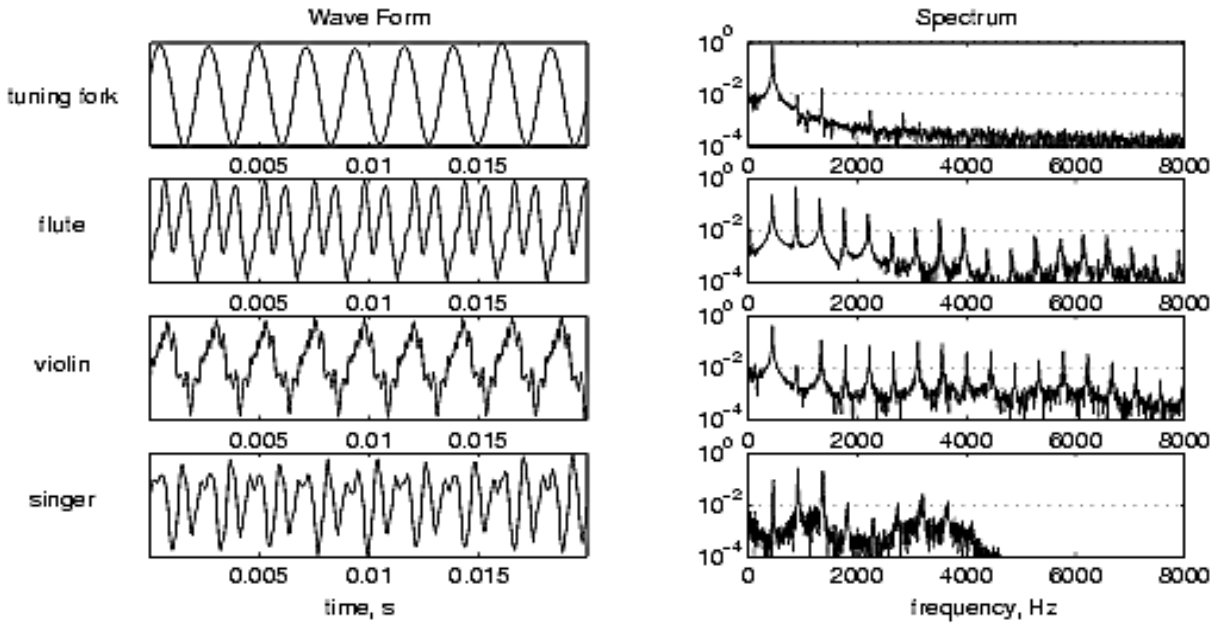


Figure 2.8 – Waveform (time) and spectrum (frequency) from tuning-fork, flute, violin and human voice.

At the same time as there is the phenomenon of harmonicity, there is also the **lack of harmony** of the signal, which is usually found in stringed instruments where the partial harmonics are defined according to the following equation:

$$f_m = m\, f_0 \sqrt{1 + \varepsilon\, (h^2 - 1)} \quad (6)$$

where m is the harmonic number and ε is the inharmonicity coefficient.

2.2.2.6 Pitch

We can define the **pitch** of a sound as the perceptual characteristic that allows us to order it in a logarithmic musical scale, from bass to treble. The pitch is, in other words, the "musical note" perceived by the human ear which allows our brain to associate that sound with a sinusoidal signal of a certain frequency. Although, conceptually, they are not

the same, the pitch is closely related to the fundamental frequency of sound. Therefore, we use these two concepts interchangeably throughout this report.

Despite the number of definitions that exist for the concept of pitch, they all refer to the same thing; this attribute allows sounds to be ordered on a scale.

2.2.2.7 Envelope or musical articulation

It shows the shape of sound in the time domain based on the way a musical instrument is played, including the human voice. Articulation refers to what happens in the first milliseconds of a sound, and to the amount of energy that is applied to the note. For example, with a trumpet, a large amount of energy can be applied by playing a "sforzando" note, if a higher initial pressure is exerted with the mouthpiece on the trumpet. Then, most of the pressure is released causing the sound to stabilize until it gently disappears [4].

With a piano, the articulation is different, because initially a hammer blow is heard, and then, the note gently disappears. Thus, the piano has a percussive articulation, which graphically represents an immediate impulse.

2.2.3 Introduction to music theory

In this section of the report, I detail the fundamental features regarding the fundamental theory and the appropriate concepts that allow us to enter in this wonderful art.

A musical piece can be represented with symbolic notation on a score (see *Figure 2.9*). At the same time that the letters represent the notation of the human speech, the musical notes form the symbolic notation used in a score. The representation of the sounds with the notes is not enough to describe a composition, it is necessary to include information on other aspects such as the speed of interpretation, the intensity or dynamics, among others, so that the melody is fully described.

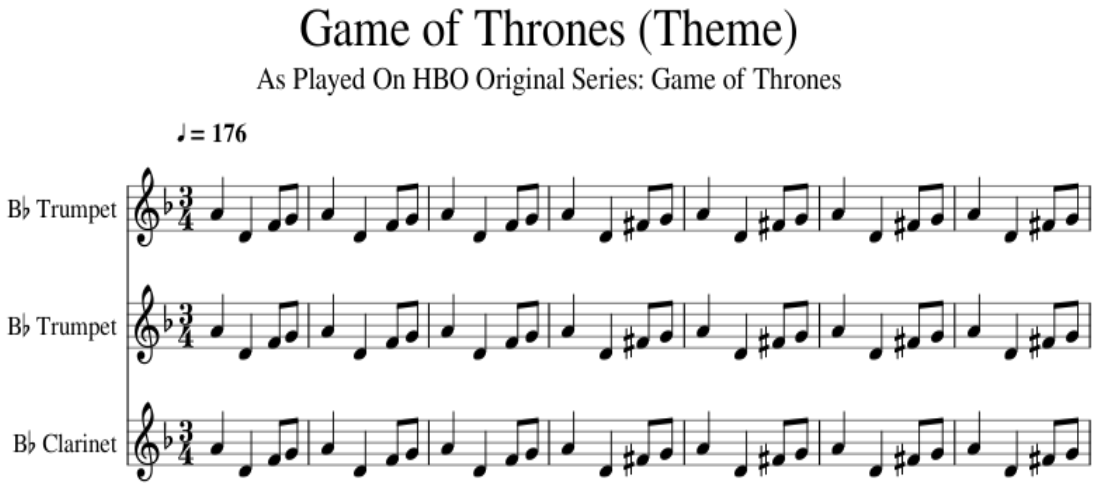


Figure 2.9 – Fragment of the musical piece "Game of Thrones" composed for trio (trumpet, trumpet and clarinet).

It is important to understand some basic concepts about music theory that we define in the next subsections [4].

2.2.3.1 Traditional musical notation

- An **event or musical note** is the basic unit of music that represents a pitch that remains active for a period of time. According to their pitch or fundamental frequency (f_0), the notes are organized in musical scales. Western music considers scales that divide an octave into 12 musical notes separated by a semitone (see *Figure 2.10*).

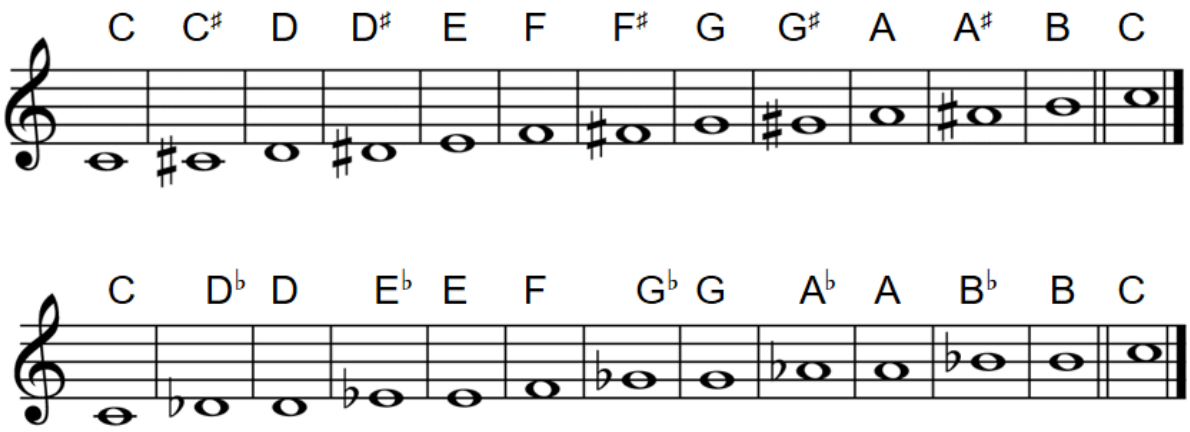


Figure 2.10 – Chromatic scale example (sharp and flat, respectively)

The 7 notes of the diatonic scale are, in Latin notation: Do, Re, Mi, Fa, Sol, La and Si, and are separated by intervals of 2,2,1,2,2,2,1 semitones respectively. In English notation, the system is similar, but the notes are represented with letters from A to G, starting with the note La (A = La, B = Si, C = Do, ... G = Sol). To these 7 notes, we can add an alteration to modify their pitch a semitone up (sharp - #) or down (flat - ♭). For example, Do# (C#) is a semitone above Do (C) and a semitone below Re (D).

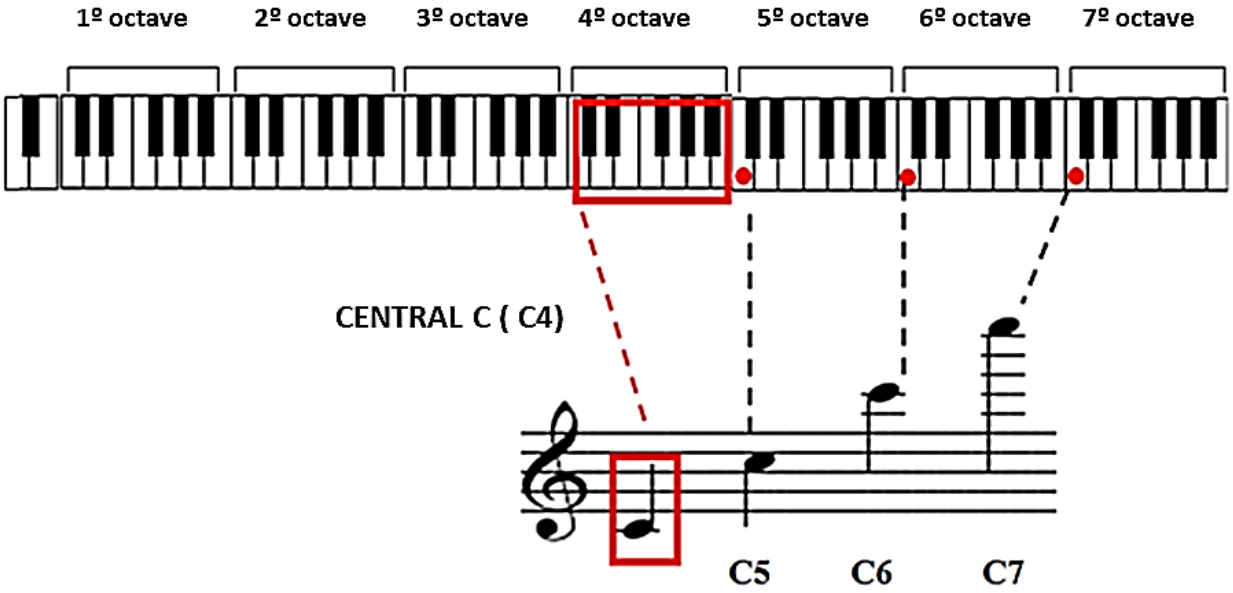


Figure 2.11 – Piano keyboard with 88 keys including from note A0 to C8. Each scale is made up of 7 white keys (C, D, E, F, G, A, B) and 5 black keys (C # or D ♭, D # or E ♭, F # or G ♭, G# or A ♭, A # or B ♭).

The **duration** of each musical note is represented by musical figures, which give us information about how many times the note has. Each figure has a fixed duration respect to the reference figure. For **example**, if the reference figure is crotchet, the rest of figures have the following relationships: semibreve (4), minim/half note (2), crotchet (1), quaver/8th note (1/2), semiquaver/16th note (1/4), demisemiquaver/32th note (1/8) and semidemisemiquaver/64th note (1/16).

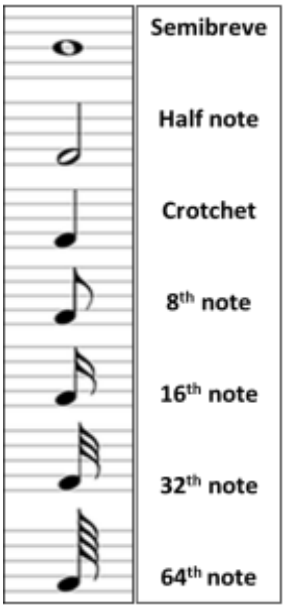


Figure 2.12 – Musical figures. Source http://www.simplifyingtheory.com/musical-figures-in-sheet-music/musical-figures/

- An **octave** is the interval between two frequencies f_1 and f_2, where $f_2 = 2 \cdot f_1$. There is a musical scale (12 notes in each octave), and the octave/scale to which each note belongs is indicated with a number. For **example**, C3 is the C note of the 3rd octave, C4 is the C note of the 4th octave, and among them there is an octave of separation, as we can see in *Figure 2.13*.

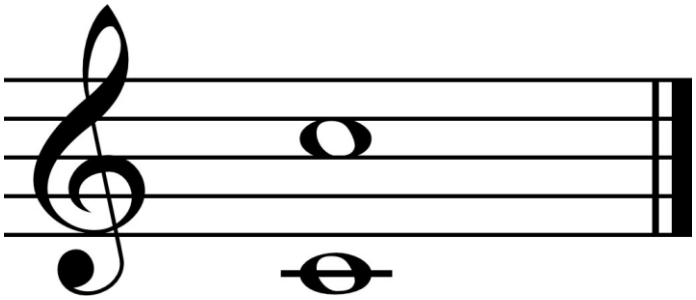


Figure 2.13 – Example with the same musical note in different octave. Source https://www.britannica.com/art/octave-music

- A **semitone** is the minimum musical interval considered in Western music between consecutive notes. Between two frequencies f_1 and f_2 there is a semitone if $f_2 = 2^{\frac{1}{12}} \cdot f_1$. In an octave there are 12 semitones. We could say that the resolution in frequency of a score is 1 semitone.

- A **staff** is a set of 5 parallel horizontal lines in which musical notes are represented as a function of time. The vertical position in which the note is represented determine its pitch. The pitch of each note is determined by its vertical position on the staff and the notes on the left are played before the notes on the right.

Although, the exact start time of each note is not directly proportional to its horizontal position, it is determined based on the musical symbol used for each particular note.

- In addition, the staff is divided by a vertical line, which divides the piece into fragments, each of them is called **bar** or **measure** and we use them to make reading the score more comfortable. A thicker and double vertical division indicates the beginning and the end of the score. The bar is a tool to indicate the rhythm of the music. At the beginning of the staff, next to the clef, the rhythm of the bar is indicated by two numbers (see *Figure 2.14*). The numerator indicates the number of times (beats) that the bar have and the denominator indicates the unit of time, that is, the musical note that occupies a full time of the bar. For example, a 4/4 measure represents that the unit of subdivision is crotchet figure and each measure is composed of 4 beats.

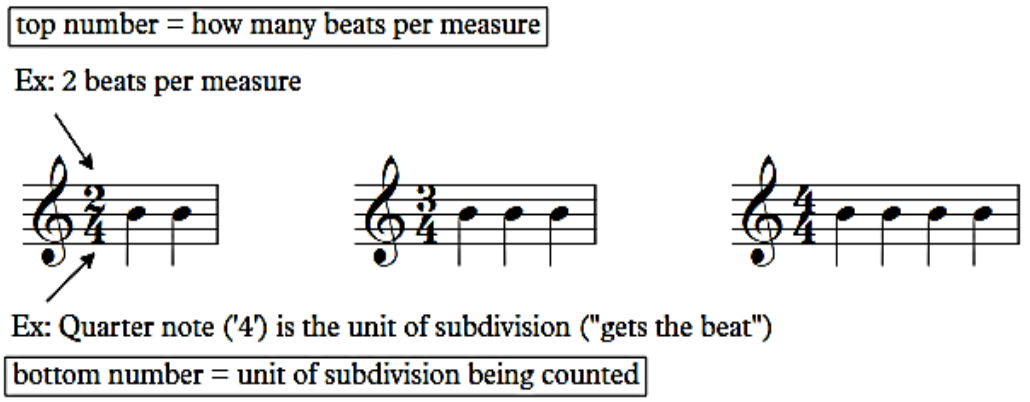


Figure 2.14 – Example of measure types. Source https://brainly.lat/tarea/1505807

- The **clef** is a musical symbol that is used to indicate the tone of the notes written on the staff. It is placed on one of the lines at the beginning of the score and indicates the name and pitch of the notes placed on that line. The three most important clefs are: G or treble clef, F or bass clef and C or alto clef, as we can see in *Figure 2.15*.

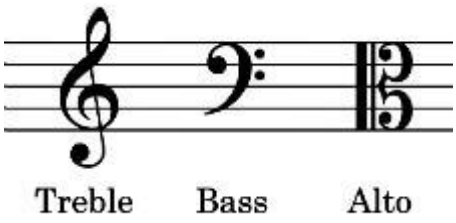


Figure 2.15 – Types of musical clefs.

- Also, the next concept is important, a **chord** is a combination of two or more notes playing simultaneously or almost simultaneously. The most frequent chords are the triads, which are chords of three notes. A chord can be consonant or dissonant depending on the relationship between the frequencies of the partials that make up these harmonic sounds.

- Other aspects of the performance of a musical piece are also included in the scores, such as the **intensity** of the notes (piano, forte, crescendo,...), and **the tempo or the execution rate**, which can be indicated subjectively (andante, allegro,...) or exactly (beats per minute - bpm - where each beat corresponds to a specific musical figure, and is indicated at the beginning of the score).

- According to Oxford University Press, a **melody** is an "ordered and coherent linear succession of musical sounds of different heights that form a structured unit with a musical sense, independent of the accompaniment" [5].

2.2.3.2 MIDI protocol

MIDI (**Musical Instruments Digital Interface**) is a digital communications protocol originally created to allow the exchange of information between electronic musical instruments and audio equipment or computers [6]. MIDI protocol opens up great possibilities in the processing of musical signals, since it allows the generation of scores that are understandable by computers, which indicate, among other things, the notes that must be activated at each moment and their intensity. This allows a MIDI synthesizer to generate the synthetic signal using the instructions from the MIDI score.

Within the General MIDI standard, each musical instrument is identified by a number between 1 and 128, called **MIDI Program**. For example, the trumpet is identified by the number 57 and the clarinet by the number 72. This allows the synthesizer to be able to activate the appropriate sounds when orders come to play notes from different instruments. In addition, MIDI supports up to 16 simultaneous channels (instruments).

In MIDI protocol, each musical note or pitch is identified with an integer (MIDI note). The relationship between fundamental frequency (f_0) and MIDI note (N_{MIDI}) is given by equations (7) and (8). Note A4 (f_0 = 440Hz) corresponds to MIDI note 69. To convert a frequency f_0 into MIDI notation, the next equation is used:

$$N_{MIDI} = \left[69 + 12\log_2\left(\frac{f_0\ (Hz)}{440}\right)\right] \quad (7)$$

To convert from MIDI notation to Hertz the following equation is used:

$$f_0\ (Hz) = 440 \cdot 2^{\frac{N_{MIDI}-69}{12}} \quad (8)$$

In *Table 2.2* we can see the fundamental frequency of each musical note depending on the scale to which it belongs.

Note events (Hz - MIDI)									
Nota / Octava	0	1	2	3	4	5	6	7	8
C	16.35 (12)	32.70 (24)	65.41 (36)	130.8 (48)	261.6 (60)	523.3 (72)	1047.0 (84)	2093.0 (96)	4186.0 (108)
C#	17.32 (13)	34.65 (25)	69.30 (37)	138.6 (49)	277.2 (61)	554.4 (73)	1109.0 (85)	2217.0 (97)	4435.0 (109)
D	18.35 (14)	36.71 (26)	73.42 (38)	146.8 (50)	293.7 (62)	587.3 (74)	1175.0 (86)	2349.0 (98)	4699.0 (110)
D#	19.45 (15)	38.89 (27)	77.78 (39)	155.6 (51)	311.1 (63)	622.3 (75)	1245.0 (87)	2489.0 (99)	4978.0 (111)
E	20.60 (16)	41.20 (28)	82.41 (40)	164.8 (52)	329.6 (64)	659.3 (76)	1319.0 (88)	2637.0 (100)	5274.0 (112)
F	21.83 (17)	43.65 (29)	87.31 (41)	174.6 (53)	349.2 (65)	698.5 (77)	1397.0 (89)	2794.0 (101)	5588.0 (113)
F#	23.12 (18)	46.25 (30)	92.50 (42)	185.0 (54)	370.0 (66)	740.0 (78)	1480.0 (90)	2960.0 (102)	5920.0 (114)
G	24.50 (19)	49.00 (31)	98.00 (43)	196.0 (55)	392.0 (67)	784.0 (79)	1568.0 (91)	3136.0 (103)	6272.0 (115)
G#	25.96 (20)	51.91 (32)	103.80 (44)	207.7 (56)	415.3 (68)	830.6 (80)	1661.0 (92)	3322.0 (104)	6645.0 (116)
A	27.50 (21)	55.00 (33)	110.0 (45)	220.0 (57)	440.0 (69)	880.0 (81)	1760.0 (93)	3520.0 (105)	7040.0 (117)
A#	29.14 (22)	58.27 (34)	116.50 (46)	233.1 (58)	466.2 (70)	932.3 (82)	1865.0 (94)	3729.0 (106)	7459.0 (118)
B	30.87 (23)	61.74 (35)	123.50 (47)	246.9 (59)	493.9 (71)	987.8 (83)	1976.0 (95)	3951.0 (107)	7902.0 (119)

Table 2.2 – Fundamental frequencies (Hz) and corresponding MIDI notes [4].

2.2.4 Music signal processing systems

The applications of music signal processing are very varied, and they include from information extraction and content analysis of musical pieces to automatic transcription or sound source separation. It should be noted that this work is a direct application of the

information extraction and content analysis of musical pieces, which are explained in detail in the following section. Below, we briefly discuss some problems in which music signal processing is applied to and the relationship with these applications [7].

2.2.4.1 Automatic music transcription

We can define *AMT* (Automatic Music Transcription) as the analysis of a musical signal in order to identify the source, the pitch, the start time and the duration of all the sounds that occur throughout it. In this way, the system is able to transfer the obtained information to symbolic notation, generating a score or an equivalent representation (for example, a MIDI file).

In the same way as multi-pitch estimation, when the degree of polyphony increases, the complexity of the problem increases. In addition, the music transcription goes a step further, because, in addition to identifying the instruments and notes active at each moment, it also requires estimating tempo, clef, nuances, differentiating melody and accompaniment, etc.

Traditionally, the transcription of musical pieces has been carried out manually by professional musicians. In fact, at present, the Automatic Transcription systems have not reached a level of development that allows them to overcome the human being in the performance of this task.

2.2.4.2 Multi-pitch estimation

Multi-pitch estimation is a technique that allows you to find out which pitches or fundamental frequencies are active at each moment in a polyphonic signal and their amplitudes. This makes it possible to estimate the number of instruments or sound sources and opens the door to more advanced systems, such as automatic music transcription.

It is a complex problem, but essential in a large part of the applications of music signal processing, which is why it has been an open field of research in recent years and continues to be so today. A definitive solution to the problem has not yet been found, since the error rates when increasing the degree of polyphony are still very high.

2.2.4.3 Audio to score alignment

Today, thanks to formats such as MIDI, a computer may be able to analyze and process the information in a score. An electronic score contains information on the defined times at which each note should be played. However, when a musician interprets the score, it is impossible for these times to coincide with those of the interpretation, since

deviations always occur due to changes in tempo, musician's failures or voluntary modifications.

Alignment is the technique that allows the recording of a musical piece to be synchronized with the corresponding score. In other words, the alignment techniques allow us to obtain the relationship between the playing times and those defined in the score. There are two approaches to the problem: **offline** and **online** [8]. In offline alignment, the system has the complete recording before starting the alignment process, allowing information from the “future” to be used, so we can obtain better results. On the other hand, online approach is intended for real-time applications, as it processes data as the signal is acquired.

There are many interesting applications of alignment, such as automatic accompaniment systems, capable of generating accompaniment in real time according to the position of the score in which the musician is. Another example is score tracking, which allows us to evaluate a performance, automatically turn pages or synchronize visual effects with music.

Alignment is an essential part of the application that has been developed in this work and then we have dealt with the extraction of musical statistics such as tempo, beat time and note duration. For this reason, it is necessary that the score is synchronized with the recording to be processed and we have used an algorithm based on **Dynamic Time Warping** (*DTW*). This system is explained in *section 4.1* of this report.

2.3 State-of-the-art automatic alignment

2.3.1 Overview

The **alignment** between audio and score is the task of synchronizing an audio recording of a musical piece with the corresponding score. We assume that the efficiency in the performance of a musical piece has variations in tempo due to musical expression or errors. This alignment or synchronization is accomplished by extracting some features from the audio signal and the score. Then, the system finds the best match between the MIDI and real time sequence. A basic diagram of an alignment system is shown in *Figure 2.16*.

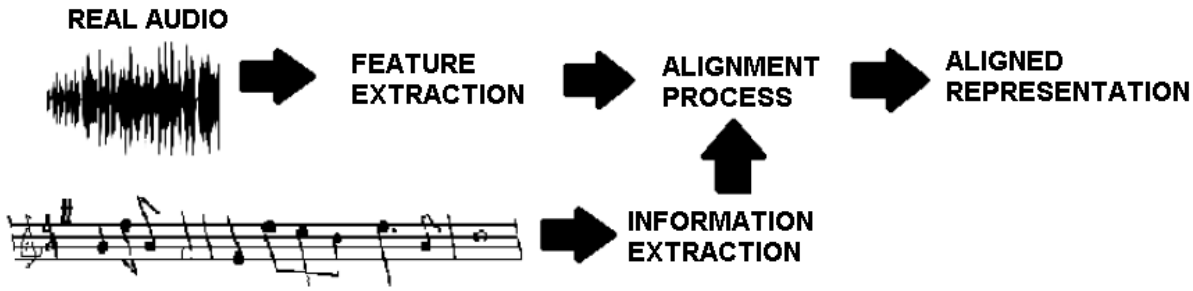


Figure 2.16 – Basic Audio-Score Alignment Diagram.

There are two different approaches depending on how the alignment is processed: **offline** and **online**. In the offline version, all the information required for the alignment process is collected and this allows the system to process the information and establishes the pairing. This approach is used when the application does not require a real-time alignment process and can achieve greater precision. In contrast, in the online version (score following), the alignment is performed as the signal is acquired.

The development of audio-to-score alignment dates back to the **early 1980s** (**Dannenberg**, 1985; **Vercoe**, 1984) [9]. During this period, due to the time calculation restrictions, sampling was carried out on the received monophonic audio signal, that is, on which there is only a single melodic line (this can only work for monophonic instruments). In addition to the restrictions on the processing capacity, some type of system was necessary to increase the output value of the instrument and consequently the application could detect it. The processing capacity over time, together with the standardization of the MIDI format, made it necessary to adapt the developments by accepting inputs in MIDI format.

The next evolutionary step in the use of this technology was linked to the improvement of audio processors, which spawned the first tone-based alignment systems. Later, the **1990s** introduced probabilistic methods for speech and audio recognition, such

as the **Hidden Markov Models** (*HMM*). These intelligent systems took new factors into account such as the uncertainty of the machine's performance, so the use of these probabilistic methods allowed guaranteeing the robustness of the application.

In addition to Hidden Markov Models technique, there is another technique that is the most used together with *HMM* and it is **Dynamic Time Warping** (*DTW*), an algorithm that measures the similarity between two temporal sequences. Today, even those are the most common approaches to tackle this issue.

Based on these additions, the latest variations of audio-to-score alignment systems consist of advanced alignment, which allows musicians to perform actions synchronously with accompaniment during concerts. The advance paradigm allowed the development of **Antescofo** [44], an audio-to-score alignment system considered today the standardized platform for most compositions.

The latest innovation in these systems is synchronous programming. Audio-to-score alignment based on synchronous programming allows concurrency and parallelism techniques to be introduced and polyphony, more than one melody line at the same time, to be added to audio-to-score detection [9].

2.3.2 First approaches of score following

One of the first systems was designed by **Dannenberg** (1985) to follow monophonic MIDI input using dynamic programming and symbolic representation. Both the input signal and the score are converted to character strings to best match each other. The objective of this system was to create a real-time musical accompaniment application for solo musicians. This approach was not able to deal with polyphonic signals because it was designed to deal with monophonic instruments and it was later expanded to include piano performances [10].

The same year, **Vercoe** introduced another score tracking system for the same purpose, an automatic accompaniment system. In this method, which was called **"Synthetic Performer"**, the developer wanted to use pitch as the main audio feature, but the pitch detection was not fast enough in time to provide real-time results. For this reason, Vercoe added to the algorithm the real information extracted from the musician, that is, the real audio signal played by the performer. The information was acquired through a series of optical sensors installed on the keys of a flute. Later, the alignment is performed using pattern matching techniques [11].

These two primary systems showed slight differences in performance and greater differences in focus. **Vercoe's system** is more responsive, assuming professional musicians change the score tempo while performing; Dannenberg achieved greater robustness by being less precise with the musicians' skills.

Another approach that is also based on pitch information was introduced by **Puckette** in the system that was called **"Explode"**. The technique used in this score follower is based on a list of unpaired notes. The algorithm uses a pointer to the current note to find the maximum match in three steps. First, it tries to match the played note coming from the performance with a note from the list. In the next step, it tries to match that note to the current note, and finally a note in the near future. This method was used for live music concerts [12].

2.3.3 Currently used techniques for score alignment

In the last two decades, there have been two different approaches to speech processing research that have emerged in the audio field to qualify alignment. This section describes these two methods and reviews some alternative well-known approaches.

2.3.3.1 Statistical methods (HMM)

A **Markov process** is a stochastic process based on a statistical model in which the system being modeled satisfies the **Markov property**. The property is true when the future values of the system output signal depend only on the current state of the system and not on any of the previous states.

Hidden Markov Models (*HMM*) are widely used for statistical modeling of non-stationary stochastic processes and are widely used for speech and music. An HMM can be described as a finite state machine where the transition between states has rules which are defined by probability functions. In every transition, the next state has a value associated to a given probability. The probabilities from one state to another depend on a limited previous state that is generally established to one. The states are not directly observable and we can only know the value emitted by each state, which is called observation. In *Figure 2.17* we can see a basic diagram about an HMM system.

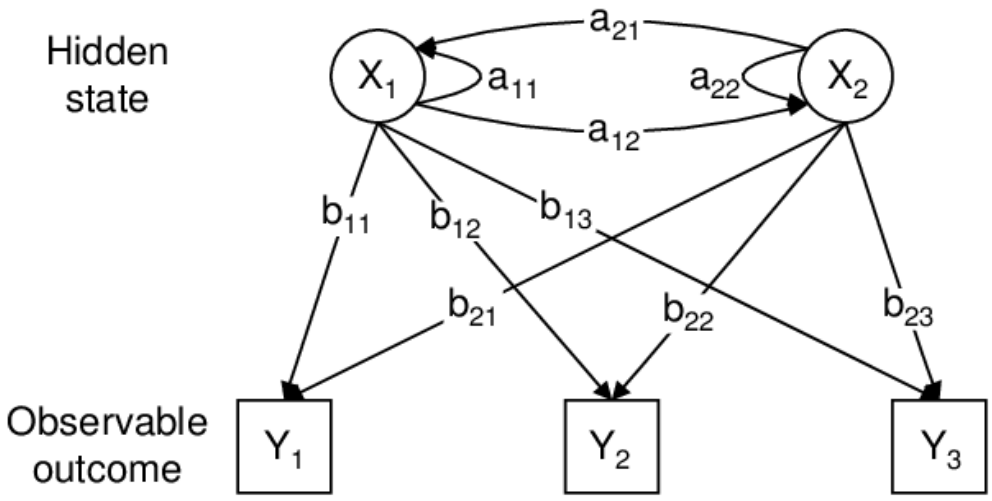


Figure 2.17 – HMM Basic Diagram.

The set of parameters of an HMM system can be trained to maximize the probability of a given set of observation sequences. Decoding is the search for the optimal sequence of states given a sequence of observations. A more detailed discussion of HMM theory can be found in the work of **Lawrence R. Rabiner**. This researcher was one of the first to investigate and study the Hidden Markov Models applied to speech processing. For this purpose, Rabiner developed a pattern recognition approach to continuous speech recognition system, whose block diagram is represented in *Figure 2.18* [13].

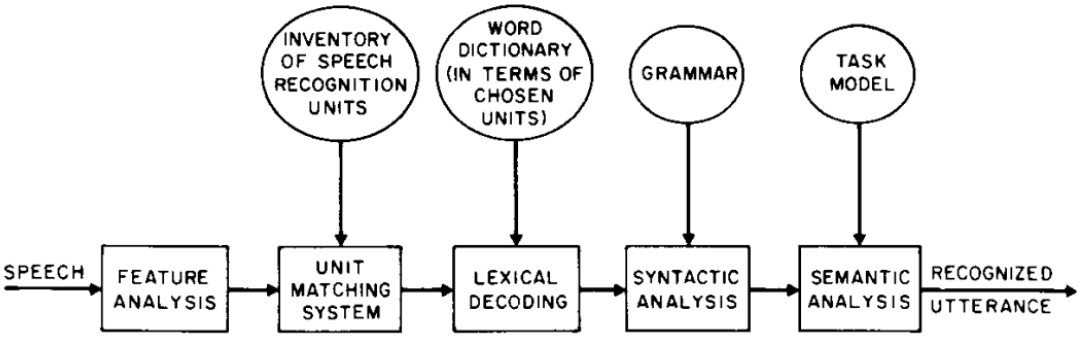


Figure 2.18 – Block diagram of a continuous speech recognizer based on a pattern recognition approach. Source [13]

This system collected human speech and made a previous analysis to extract its features. Then he applied a matching system to determine all the sub-words of speech. Next, the system was able to decode the lexicon of the sequence from a dictionary of words (in terms of the units chosen in the previous stage) and it also applied a syntactic analysis based on grammar. Finally, the approach made a semantic analysis and we can extract the recognized speech [13].

One of the first approaches, applying HMM, to achieve audio-score alignment was introduced by **Pedro Cano, Alex Loscos and Jordi Bonada**. Their system is built with **three different HMM models** according to each observation type: a note, a no-note, and a silence. The "note" model is constructed with three different states (attack, sustain and release) that correspond roughly to the envelope of the energy of a high-pitched sound. The "no-note" model, related to non-pitched sounds, is built the same way. Finally, the "silence" model has only a state. The length of the notes is modeled with auto-transitions and the **Viterbi algorithm** is performed to achieve the alignment [14].

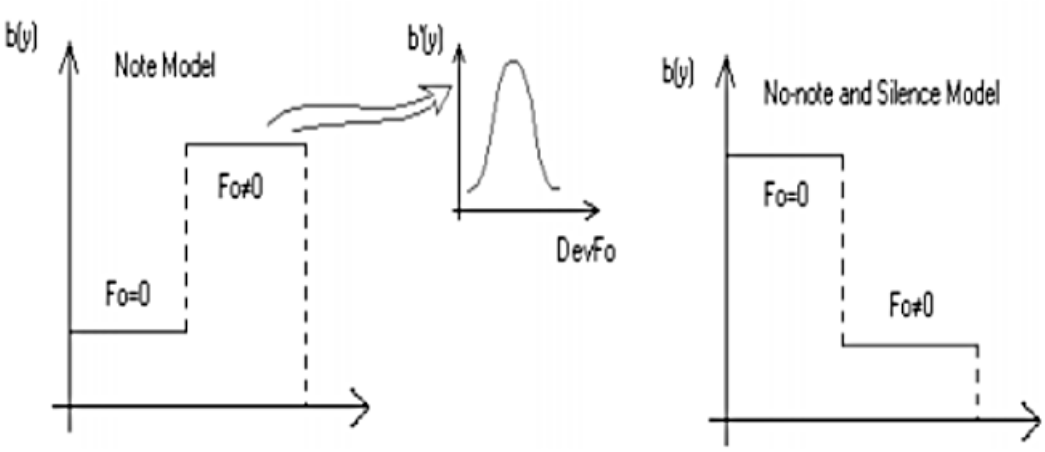


Figure 2.19 – Probability function f_0.

Another influential method in this type of algorithms was the one created by **Nakamura** and **Sagayama**, where they talk about an audio-to-score alignment system in real-time. The main goal of this method is to align the audio signals of a music performance to the corresponding score, which can handle tempo changes, errors and arbitrary repeats and/or skips (repeats/skips) in performances [15]. This type of score tracking is particularly useful in automatic accompaniment for practices and rehearsals, where mistakes and repeats/skips are often made. This algorithm takes into account four typical sources of variety in monophonic audio performance: acoustic variations, temporal fluctuations, performance errors and repeats/skips. The method considers **two levels** in the state machine: the **top level** is the one that describes the progression of interpretation events, which is described as transitions between the top states and the bottom level; and, the **bottom level**, which corresponds to subevents in an event. The method also considers the pause state in which the score does not contain a musical note. In *Figure 2.20,* we can see the basic diagram about this system.

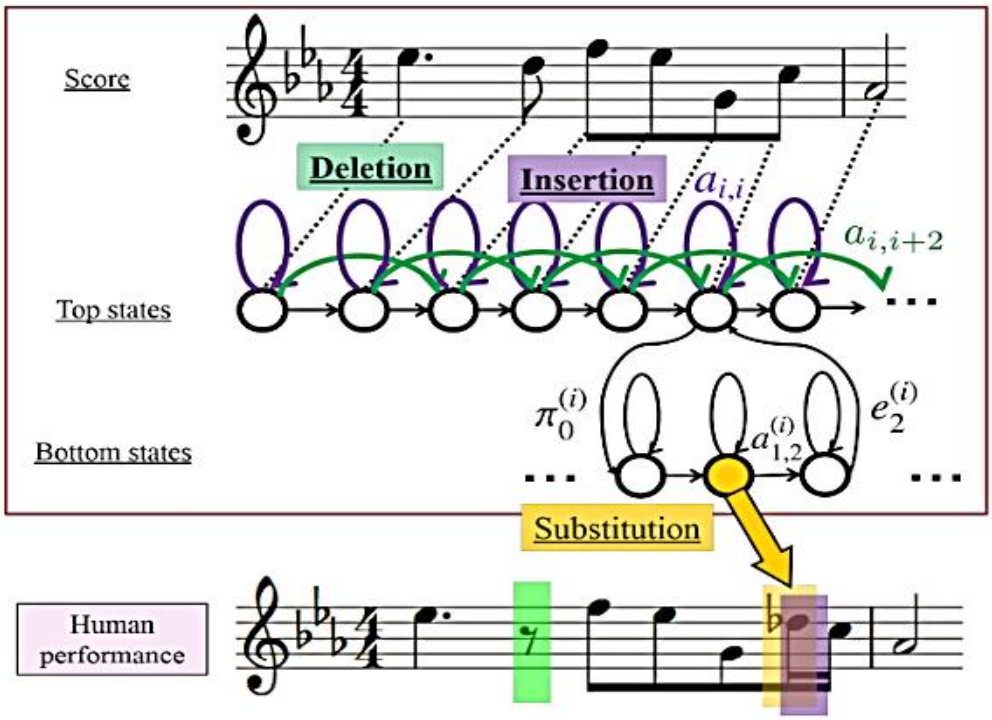


Figure 2.20 – Hierarchical Hidden Markov Model with two levels that describes a music performance with different errors (deletion, insertion and substitution) [15].

Another similar algorithm, created for score tracking, was the system developed by **C. Raphael**, where a particular **decoding technique** is used, instead of the Viterbi algorithm. According to the score information, each different note is modeled using various topologies due to its behavior over time based on: long notes, short notes and ornamentations [16].

Another method, created by **Nicola Orio**, considers two categories of states that represent notes: **normal states** (n-states) and **ghost states** (g-states). The g-states are used to model three different types of errors depending on the score: wrong note, extra note and omitted note. Then, the performance is modeled using two levels HMM. The sequence of states is decoded using an algorithm based on **dynamic programming** [17].

Some of the previous algorithms were designed for monophonic signal inputs or polyphonic instruments with the same timbre (mainly piano). As in previous approaches, the score is defined as a state string representing sequential events. The pitch classes used for the alignment process are learned from the use of databases with sound examples.

Considering polyphonic music as input, **C. Raphael** developed a system for automatic musical accompaniment called **Musical Plus One** (*MPO*). This approach is

made up of **three subtasks** called "Listen", "Predict" and "Play". The first step is to know the note start times that are identified using the same approach as the Markov model.

The algorithms presented previously are some of the many that exist and use the Hidden Markov Models (HMM) for the alignment of the input audio signal and the corresponding score. However, this type of algorithms is relatively current; there are many researchers who have thought it convenient to study in this field of research.

2.3.3.2 Dynamic Time Warping (DTW)

2.3.3.2.1 Overview

Dynamic Time Warping is a technique used to align time series used in speech recognition, data extraction and information retrieval. In general, DTW is a method that computes the optimal similarity between two given sequences (for example, time series) with certain restrictions.

The series are presented by two vectors **U = u_1, ..., u_m** and **V = v_1, ..., v_n** that contain feature vectors at each position. The alignment of the two sequences is done by computing the distances between the different positions. These distances are represented in an **m x n matrix** whose positions are the **cost**, generally calculated with the **Euclidean distance**.

Therefore, the first stage in the DTW algorithm is to compute the local distance matrix (a.k.a. cost matrix) $M(\tau, t) = \varphi(u_\tau, v_t)$, where the matrix M has m x n elements which correspond with the match cost between every two points in the time sequences. The cost function φ could be any cost function that returns cost 0 for a perfect match [23].

The second stage is to compute the warping matrix D as follows:

$$D(\tau,t) = min \begin{cases} D(\tau-1, t-1) + M(\tau,t) \\ D(\tau-2, t-1) + \gamma_{2,1}\, M(\tau,t) \\ \vdots \\ D(\tau-\alpha_\tau, t-1) + \gamma_{\alpha_\tau,1}\, M(\tau,t) \\ D(\tau-1, t-2) + \gamma_{1,2}\, M(\tau,t) \\ \vdots \\ D(\tau-1, t-\alpha_t) + \gamma_{1,\alpha_t}\, M(\tau,t) \end{cases} \quad (9)$$

where the step size at each dimension has a range from 1 to $\boldsymbol{\alpha_\tau}$, and from 1 to $\boldsymbol{\alpha_t}$, $\boldsymbol{\alpha_\tau}$ and $\boldsymbol{\alpha_t}$ are the maximum step size at each dimension. Parameter **σ** controls

possible diagonal steps. **D(τ,t)** is the accumulated cost of the minimum cost path from (1,1) to (τ,t).

Finally, the DTW algorithm finds the minimum cost path W = W_1,..., W_L. Each W_l represents an ordered pair (τ_l, t_l) of vector positions aligned according to the cost matrix [18].

The original definition for *equation (9)* considers γ = 1, so the path is stretched towards diagonal steps with respect to vertical or horizontal steps. On the contrary, there are other algorithms that use a different weight for diagonal steps, this is the case of our work.

The path with the minimum cost is computed by dynamic programming. So, the alignment of the DTW is done in **three steps** essentially:

- Extraction of comparable features from the 2 time series.
- Calculation of local distances between the feature vectors of the two time series.
- Computation of the optimal path with respect to the global distance.

In the following *Figure* we can see how the DTW technique behaves in the field of alignment.

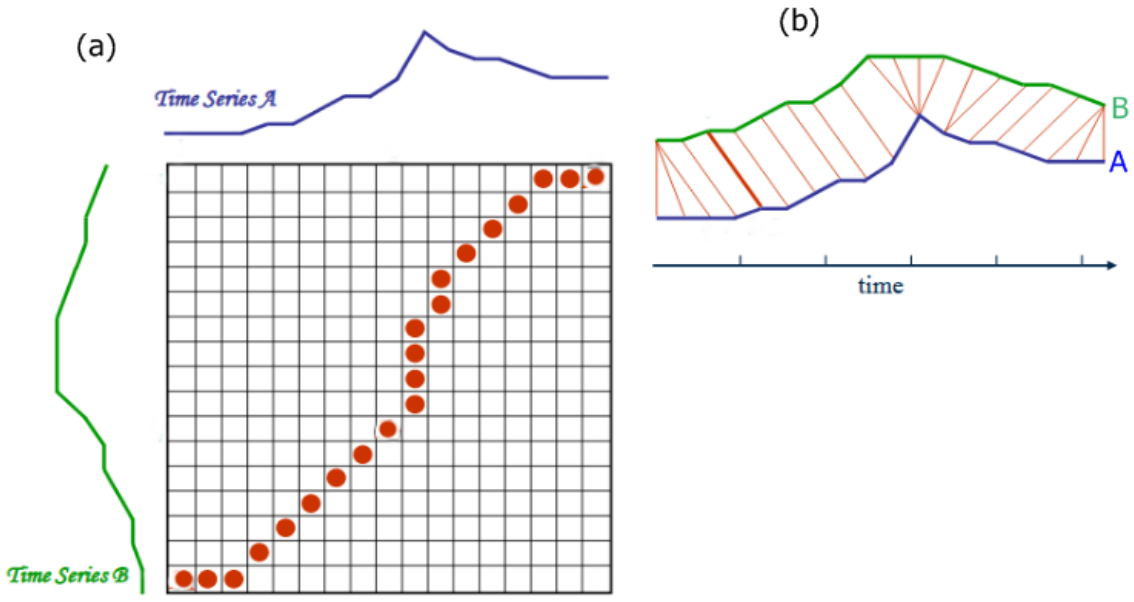


Figure 2.21 – Example of DTW online algorithm. The axes represent the variables t (real time) and τ (MIDI time) respectively. The cells with red point are the optimal path that represents the synchronization between two time series.

As we can see in the *Figure 2.21*, we have two time series: one referring to real time (t) and the other referring to MIDI time (τ). Therefore, the DTW concept applied to audio-to-score alignment is based on aligning real time series with score time series.

2.3.3.2.2 DTW for score alignment

DTW algorithm needs two sequences with the same type of data to perform the alignment process. The score is often treated as an audio signal by converting it into MIDI, and then a synthesizer is used to create a synthesized audio file. Once we have both audio signals, feature extraction processing is performed [19].

An example of this type of algorithm was created by **Ning Hu et al.** [20], in which alignment is achieved with acoustic rather than symbolic characteristics. MIDI data is mapped with corresponding audio characteristics and DTW is used to align the resulting sequences. The result not only shows us the optimal alignment between audio and MIDI data, but also gives us a score that reflects the alignment precision. Basically, the algorithm is based on representing the MIDI score on a graph called **Chroma** and later, the alignment of audio and MIDI is done [20].

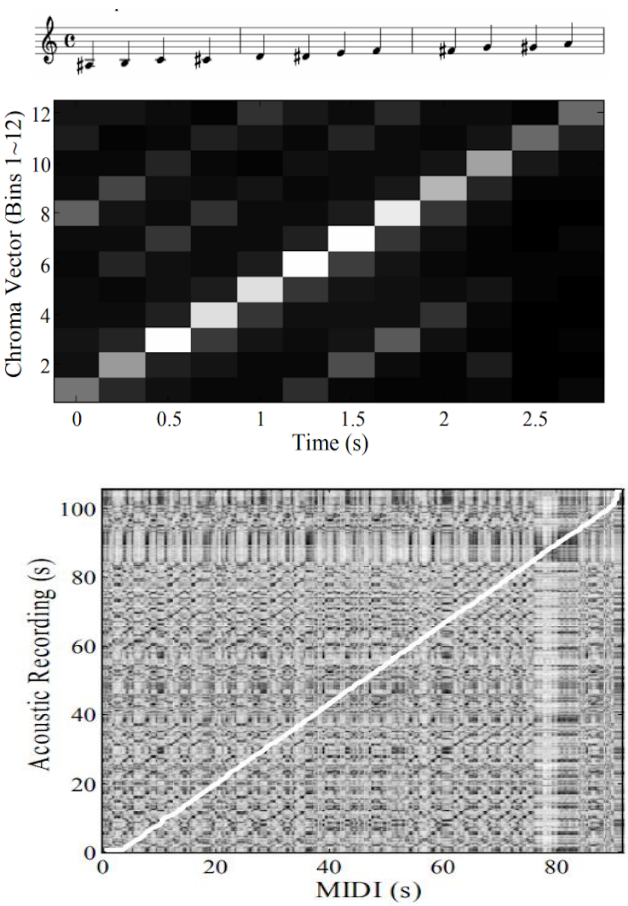


Figure 2.22 – First, the Chroma graph which relates the score with the Chroma vector. Second, the audio-score alignment matrix is represented. Source [20]

Most of the methods take advantage of the spectral representation of the audio data provided by the **Fast Fourier Transform** (*FFT*). This representation is usually mapped into a more compact version like the one developed by **Dixon**, where the spectrum is compressed into **84 frequency bins**.

Another algorithm that uses mapping in the frequency spectrum using the **Chroma concept** was also created by **Ning Hu et al**. This vector is made up of 12 elements that contain the envelope of spectral energy of a tone. In order to compute the **chroma vector** we map the frequency bins to the nearest step on the musical color scale. This method was shown to perform somewhat better than other algorithms of the same type due to the independence of the **chroma timbre**, and is still used today for **orchestral accompaniment**.

Other approaches for feature extraction have been presented depending on the application and the type of input data.

2.3.3.2.3 Graphical DTW implementation

There are algorithms that use **chroma** and **delta chroma** as characteristics from the DTW implementation, for example, the method proposed by **Kosuke Suzuki et al** [21]. This method describes a score follower in several steps. First, MIDI data is converted into an audio signal using a synthesizer. When generating the synthesized audio signal, the audio to MIDI match problem is solved with an audio to audio match. The performance and synthesized audio signals are transformed into **“chromagrams”**. Finally, the alignment is obtained by DTW between the two **"chromagrams"**. A summary of the tracking system and an example of alignment are shown in *Figure 2.23* [21].

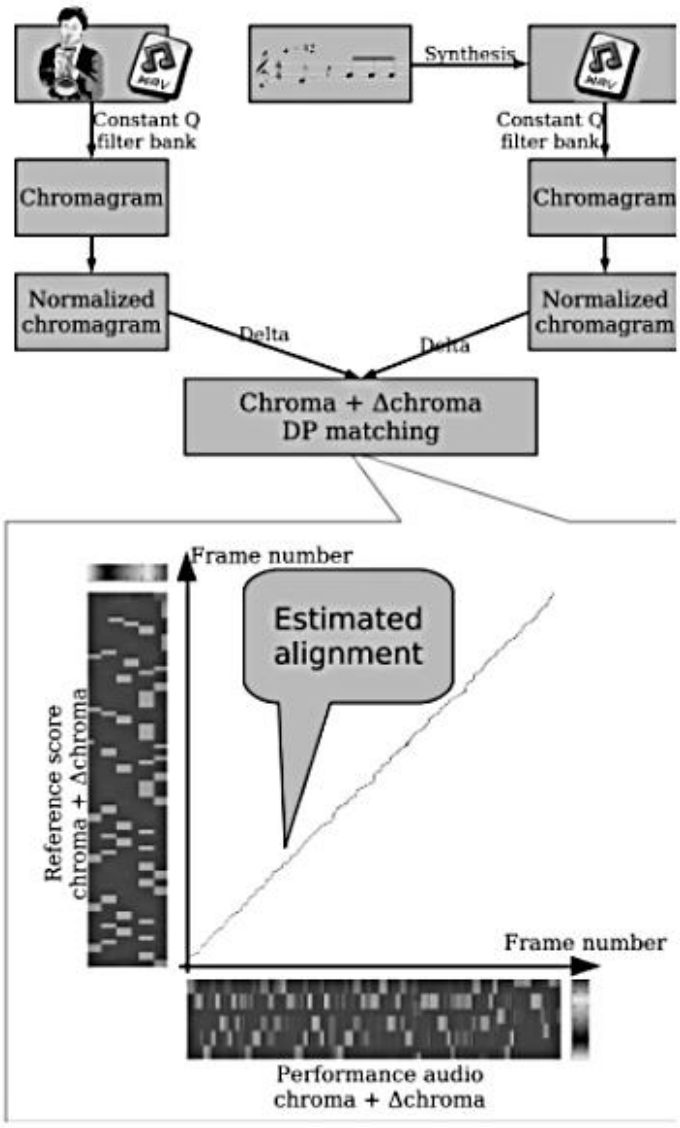


Figure 2.23 – Block diagram and example of the method explained.

The term **chromagram** is closely related to the twelve different pitch classes that form a chromatic scale, that is, this graph is achieved from the audio and score.

2.3.3.2.4 DTW implementation using spectral factorization

Highly accurate score tracking results are also achieved using an algorithm based on DTW created by the research group to which my tutors belong. This system has **two separate stages**: **pre-processing** and **alignment**. In the first step, the different states are defined as a unique combination of notes in a given period of time. These states are used as base functions for alignment. An NMF-based method is used to learn the base functions and then find these states in each frame, the result of which is a distortion matrix that shows the cost of each state in each frame. Finally, alignment is achieved by applying DTW to the distortion matrix [22].

Another method in which a spectral factorization process is implemented is the one developed by **Rodriguez-Serrano et al** [23]. This algorithm consists of **two stages**: **preprocessing** and **real-time alignment**. In the first stage, the given MIDI score is analyzed and all the information is extracted from it. The MIDI file is synthesized with software and the training spectral patterns are extracted from the synthetic audio signal using a method based on **Non-negative Matrix Factorization** (*NMF*) with **Beta-divergence**. Finally, once we have learned all the instrument and note spectral patterns, the second stage is activated, which is based on an online DTW algorithm, and the audio-score alignment is obtained [23]. In *Figure 2.24* we can see the diagram of this method.

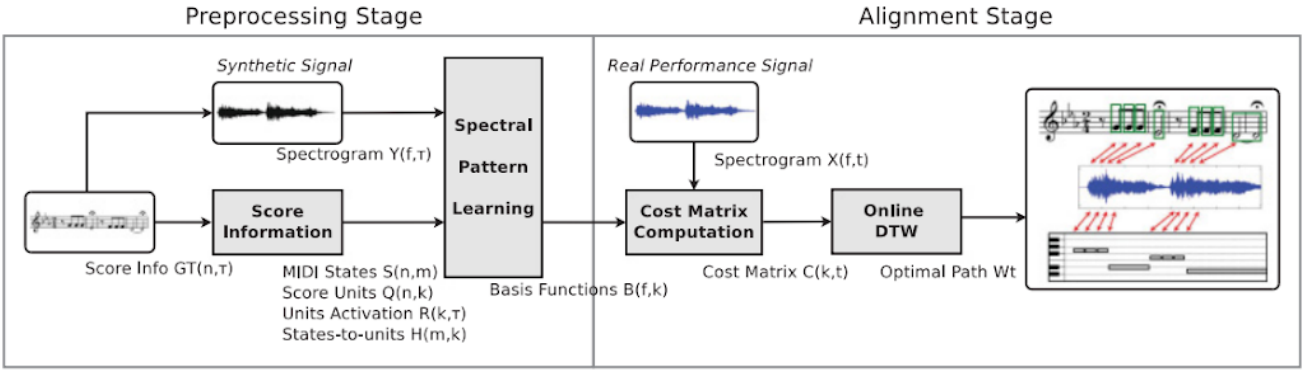


Figure 2.24 – Block diagram about the proposed score follower. Source [23]

The methods shown above are just a few of the many algorithms out there, developed by researchers that help advance technology.

2.3.4 Algorithms for comparison

During the development of this work we have always had in mind the main objective that we wanted to achieve and it was to develop a new audio-to-score alignment system that was robust against possible errors that might exist during the interpretation of a musical piece. For this reason, in this section we are going to review the methods that have served as a guide in this difficult task and which we have compared with our method. These comparisons can be seen in successive sections of this report. The different approaches compared are the following:

2.3.4.1 Ellis method

The first algorithm we took into account in order to do this work was the one developed by **D. Ellis et al.** [24]. This method was created mainly for the use in the field of speech recognition, but this purpose had some difficulties. The most important thing is that, although different recordings of the same words include more or less the same sounds in the same order, the precise time of each syllable of a word will not match.

Although Hidden Markov Models have replaced other systems, the first technique used in speech recognition was **Dynamic Time Warping** (*DTW*), as we have already seen in previous sections.

This method takes advantage of MIDI transcripts to synthesize them because the most faithful or closest interpretation that can be had of a score is the synthesis of the MIDI file itself. Once we have the synthesized audio and the real audio of the musician's performance, we can compare both signals, that is, we can compare synthetic audio with real audio.

In order to align a song with its transcription, he created the concept of similarity matrix, where each point gives us the cosine distance between the short-time spectral analyses of particular frames, analyzing features such as pitch and beat/note onset times. The following algorithm is based on using dynamic programming to find the path with the minimum cost between the starts and the ends of the sequences through the similarity matrix. Finally, this path is used to match real time with MIDI time and, if all goes well, the alignment has been successful.

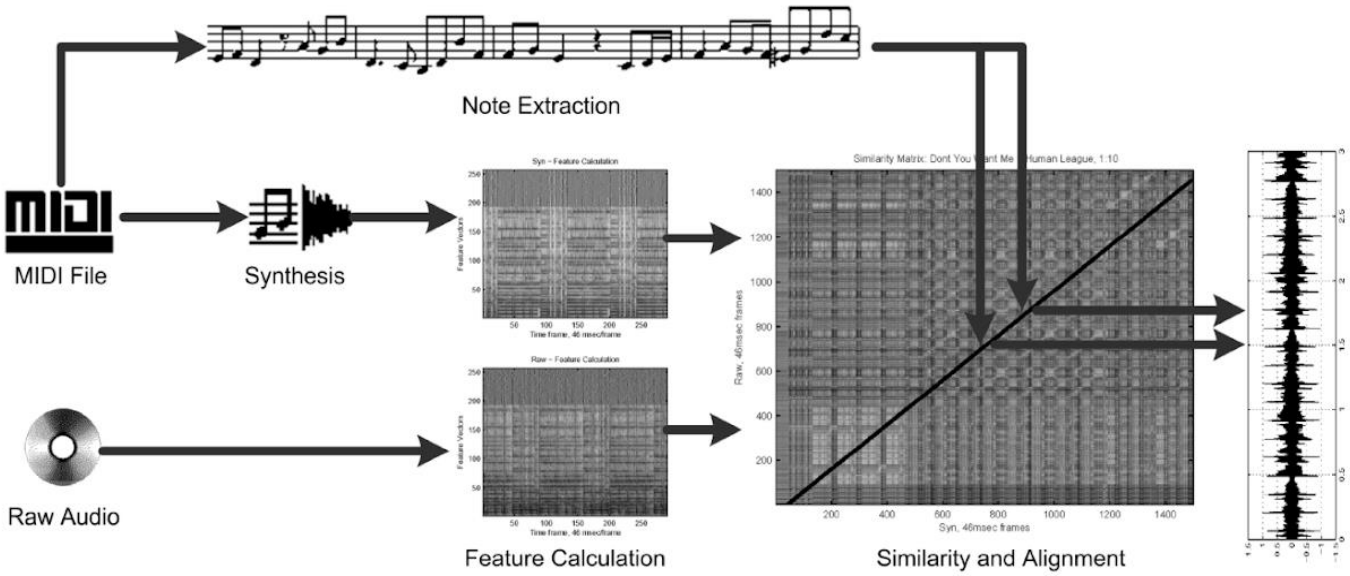


Figure 2.25 – Alignment process of Ellis' method. Features computed on real audio and synthesized MIDI are compared in the similarity matrix. The best path through SM is a warping between the real audio and the MIDI file. Source [24]

So, as we can see in *Figure 2.25,* and we have previously explained, the MIDI file is obtained from the score and synthesized. Next, the system does a spectral analysis of the synthetic audio and the real audio signal, from this spectral analysis the similarity matrix is generated and, from it, the path is calculated with the minimum cost. Once we have the path, we have the real audio aligned with the score [24].

2.3.4.2 Carabias algorithm

Highly accurate score tracking results are achieved using a **DTW-based algorithm** created by the **research group TIC-188** from UJA. This system has two separate stages: **pre-processing** and **alignment**, as shown in *Figure 2.26*. In the first step, the different states are defined as a unique combination of notes in a given period of time. The score is then converted to a reference audio signal using synthesizer software and uses a method based on **Non-Negative Matrix Factorization** (*NMF*) with **Beta-divergence** to learn the spectral patterns for each combination of notes.

In the second stage, the alignment is carried out in two steps. First, the matching measure between events in the score and in the performance is defined. More specifically, a divergence or cost matrix is estimated using a low latency signal decomposition method. Finally, DTW technique is applied to find the path of minimum cost and then determine the relationship between the performance and the times of the score [22]. This method is implemented in two versions, **online** and **offline**.

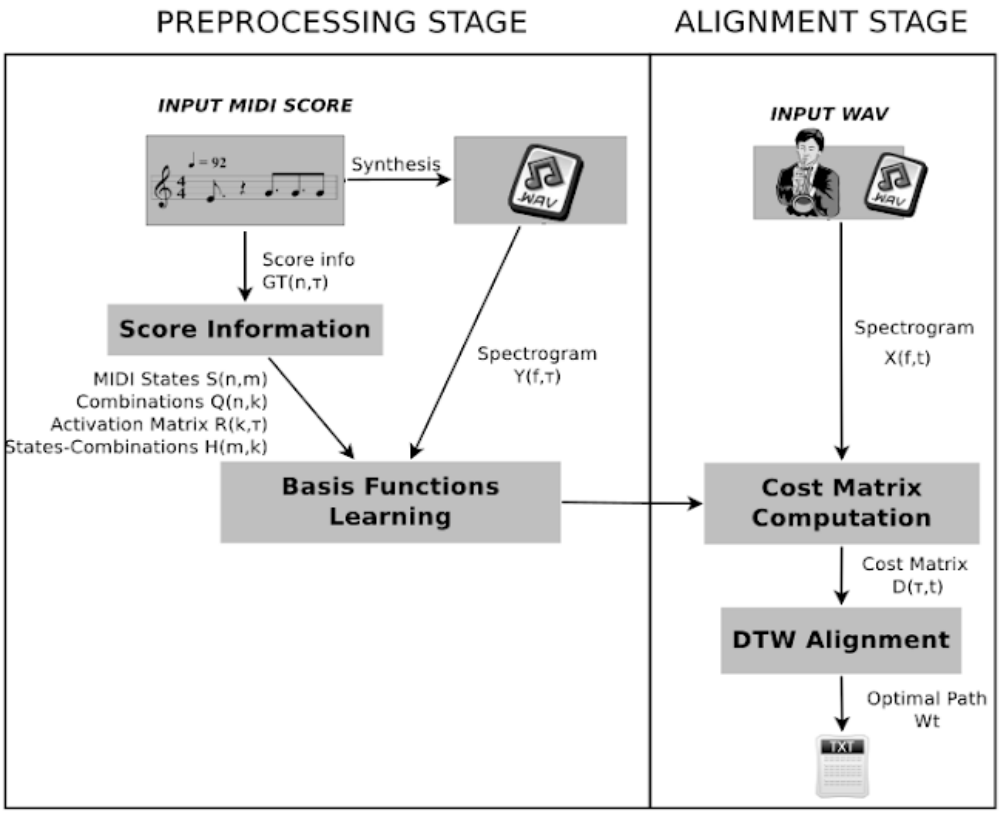


Figure 2.26 – Block diagram of the system. Source [22]

The main difference that can exist between this algorithm and the previous one is based on the fact that the previous system does a spectral analysis of the real and synthetic signal. From here, the system generates a similarity matrix to determine the

optimal path. However, this algorithm first learns the spectral bases of the information extracted from the score and then, with the synthetic bases learned and the real audio, it generates a distortion or cost matrix to determine the path with the minimum cost.

2.3.4.3 Soundprism system

The next algorithm we have analyzed is an online system created for **Score-Informed Source Separation** (*ISS*) and developed by **Z. Duan et al.** [25]. This system separates single-channel polyphonic music played by harmonic sources into source signals online.

This method consists of two parts: a **score follower** that associates a score position with each frame of the performance audio and a **source separator** which reconstructs the source signals for each frame, informed by the score. We are only going to look at the first part, since it is the main objective of our work.

As we know, an algorithm that works online is one that can process piece-by-piece the inputs serially, in the sense that the input is fed to the algorithm, without having the entire input available from the beginning, that is, the system is processing audio as it arrives.

In the score follower, the **Hidden Markov Model** (*HMM*) technique is used, where each audio frame is associated with a state (position in the score and tempo). After analyzing an audio frame, a multi-pitch observation model is used, which indicates the probability that an audio frame contains the pitches in a hypothetical position in the score. The inference of the position of the score and the tempo of the current frame is accomplished by filtering. Then, the second stage starts, which is the score-informed source separation at the aligned score position.

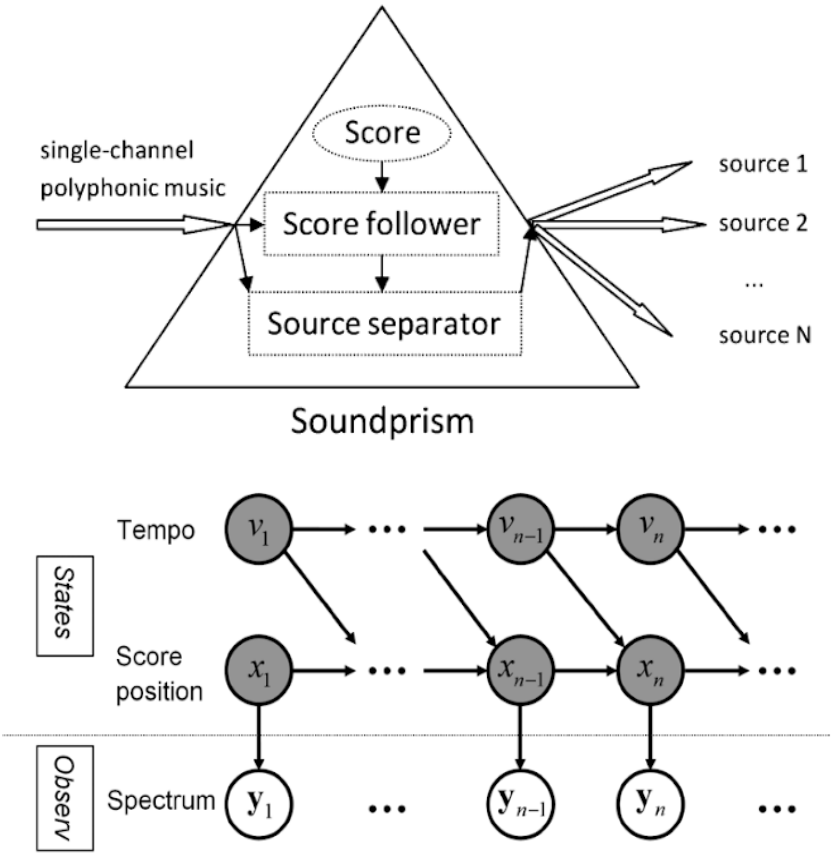


Figure 2.27 – System overview of Soundprism system and representation of the state space model for online audio-score alignment. Source [25]

As we can see in the *Figure 2.27*, each audio frame is associated with an observation, which is the spectrum of the audio. The main goal is to infer the current score position x_n from current and previous observations y_n. To achieve this, a process model is defined to describe how the states are transitioning, an observation model to evaluate the hypothetical score positions for the current audio frame, and find a way to make the inference online.

2.3.4.4 Munoz system

The next method was developed by **Munoz-Montoro et al.** [26], and the purpose of it is to achieve a score informed music signal decomposition, specifically an application to **minus one**, but this system suits for both online and offline applications.

The proposed method uses a dictionary learned a priori made up of spectral templates for all the instruments that can be presented in a score. However, this work shows that the score information is encoded with a signal model in which it is assumed that a musical signal can be described as a single sequence of notes for each instrument. Then, the cost function can be obtained by calculating the projection of each note on the

spectrum at the frame level and DTW is used to determine the path of least cost. In addition, the NMF technique is used to correct possible misalignments. As we can see in *Figure 2.28*, the proposed system consists of three stages.

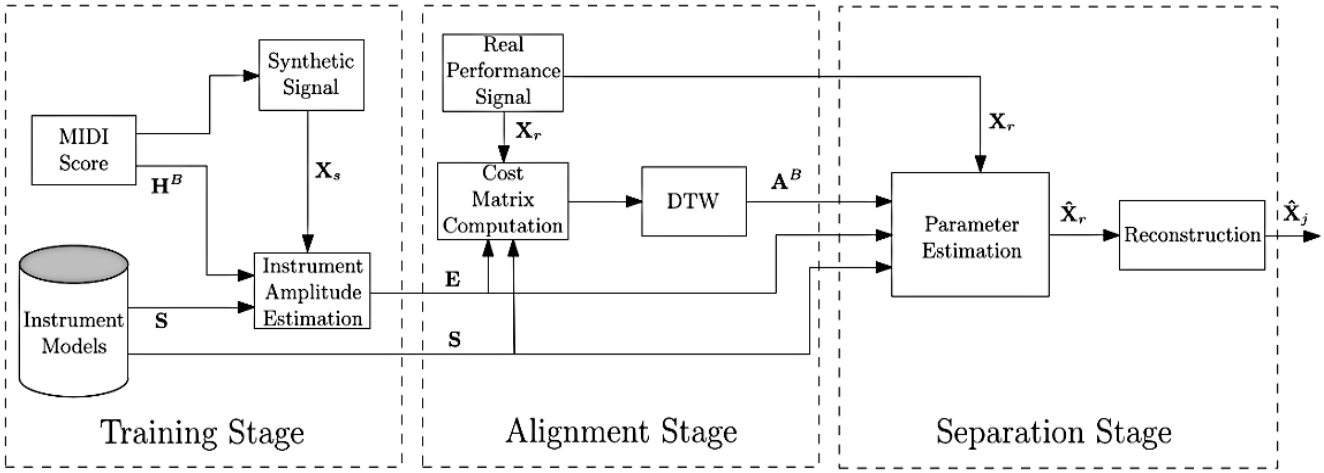


Figure 2.28 – Block diagram of the system. Source [26]

In this case, we only take into account the first two stages because the third consists of a source separation stage and our system is based on alignment. That is why we are only going to look at the alignment result of this algorithm.

First, in the training stage we initialize the bases functions using spectral patterns which were pre-trained for each of the instruments that may appear in a score and estimate the relative amplitude between the instruments and the input synthetic score. Second, in the alignment stage, an NMF-based signal decomposition method is used to estimate the model parameters minimizing the signal reconstruction error. Once at this step, we apply a *DTW* technique to the resulting cost matrix and this is used to estimate the alignment between the performance audio and the synthetic signal. Finally, we come to the third stage that is based on source separation.

The main difference between Munoz and Carabias methods is that Carabias trains with synthetic sound (created by a synthesizer software) and Munoz determines the spectral patterns from the training of real signals (from the *RWC* database) whose mixture is estimated with an NMF technique. *RWC* (**Real World Computing**) Music Database is a copyright-cleared music database that is available to researchers as a common foundation for research [42].

2.3.5 Applications

In this section we review some applications that are based on audio-score alignment in two different sections: online and offline.

2.3.5.1 Online applications

The online score alignment approach has a wide variety of applications. These applications are based on a real-time alignment algorithm and it is important that the system have a fast response, computational efficiency and robustness [34].

Score-informed Source Separation (Online)

The first system that was created under the name of score-informed source separation in an online approach is **Soundprism**, a system presented by **Duan** and which we have discussed in previous sections. As we know, the main goal of the system is to separate single-channel polyphonic music into source signals. It is essential to have the performance correctly aligned with the score in order to carry out the source separation stage [27].

Automatic musical accompaniment

The first system created for automatic musical accompaniment for soloist musician was developed by **Dannenberg** using matching techniques and dynamic programming. In this algorithm, the computer calculates the matching between the score and the soloist's performance and, from this, generates the musical accompaniment [28].

Another newer approach to automatic musical accompaniment is the **Antescofo** system introduced earlier in this report. This system is able to anticipate the response of events that are triggered in real time while playing scores electronically [29].

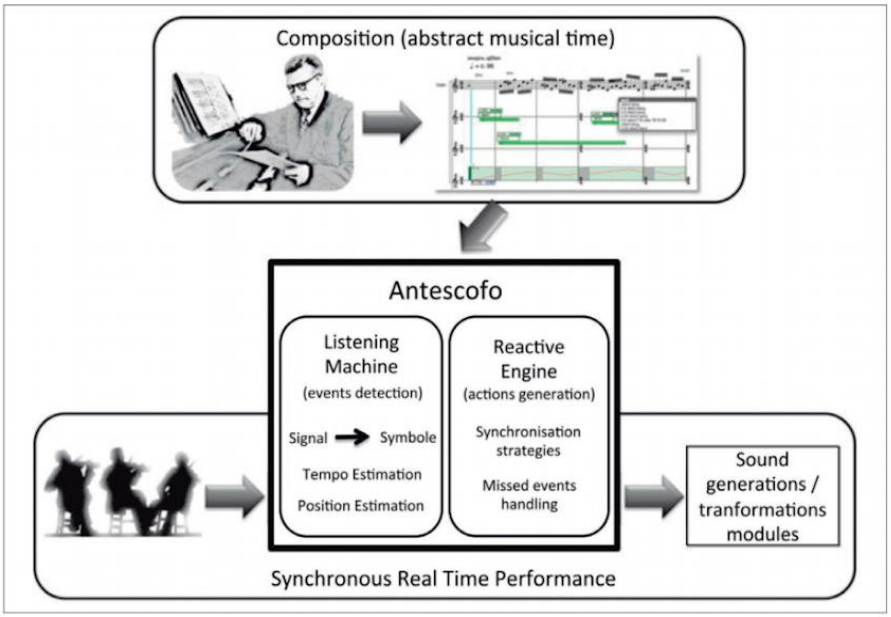


Figure 2.29 – Block diagram of Antescofo system.

Automatic page turning

Musicians have long been hampered by the challenge of turning the score page while their hands are busy playing an instrument. The Polyphonic Score Following used in **Tonara Ltd. System** is one of the most successful applications of score following. In [43] we can see an automatic page-turning based on Dixon's work [18]. In this system, the MIDI file is converted into an audio file using a software synthesizer. Then the matcher process receives one audio frame from the performance, computes its feature vector, hands it over to the matching algorithms, waits until they are done and then start with the next frame. We can see the architecture of the system in *Figure 2.30*.

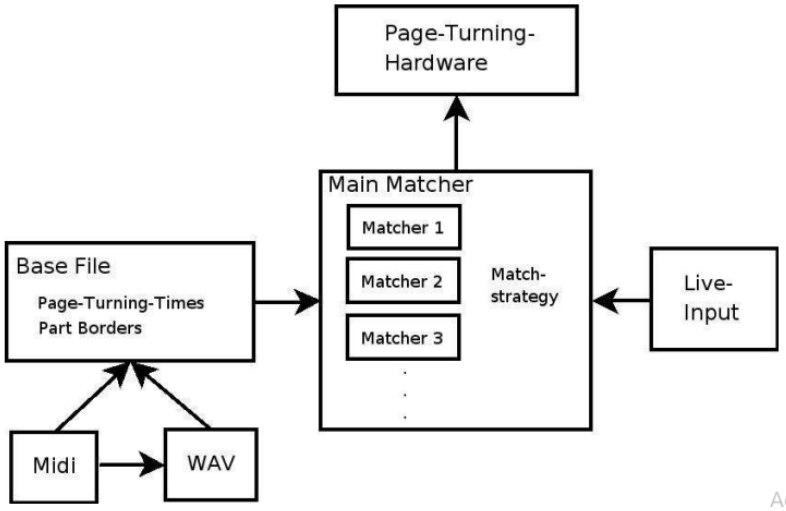


Figure 2.30 – Architecture of the Automatic Page-Turner. Source [43]

Also, this type of automatic page-turning systems is used by other applications such as **Beatik** from the company Revinclassic S.L. This application allows you to turn the page just using the sound of an instrument in real time.

2.3.5.2 Offline applications

As we already know, in the offline score alignment the whole performance is available for the alignment process. This is why offline alignment systems perform better and are more accurate than online algorithms. It is important to have a good processing unit because, if the interpretation is very long, the device may find itself in the situation of not being able to process it. In this section we review the main applications.

Score-informed Source Separation (Offline)

The source separation approach is widely used nowadays in many areas such as noise and music separation, noise and speech separation.

One of the main methods presented is the one developed by **Woodruff et al**. In this method they implemented a source separation system to separate stereo sound mixes into separate sound sources to allow remixing of the recordings. The system performs the alignment by converting the score or MIDI file and the audio signal into a **chromagram** representation. Finally, use *DTW* to find the best alignment using the Euclidean distance between the **chroma vectors** [30].

A more recent approach is related to **Fritsch** using the **NMF technique**. This method is based on two phases consisting of two NMF routines: one of them learns the spectral components of each instrument in the score and the other adjusts those components in the current mix. As in other algorithms, a MIDI file is used to synthesize each instrument separately. At the end, *DTW* algorithm is applied to align the synthesized signals in the mix [31].

Music retrieval applications

In this area there is a challenge called **Music Information Retrieval** (*MIR*) to find correspondences between different representations of the same music composition.

The audio matching and alignment system developed for music retrieval by **Hu et al.** is a clear example of this type of application. The algorithm is based on DTW and extractor of audio features. In addition, it converts the MIDI database to chroma and obtains two comparable sequence vectors. So, it does a normalization phase and computes a similarity matrix by computing the Euclidean distance between the vectors. Finally, the DTW of the matrix is computed to achieve the alignment [32].

3 Objectives

3.1 Objectives

The main objective of this work is to develop a software that allows us to obtain an evaluation of the musical statistics from a performance and, in addition, to compare with another performance of the same musical piece. In order to achieve this goal, we have implemented a **musical feature extraction** and a **comparison** algorithm, along with an **alignment** method based on a variable cost function guided by a **regularization constant (λ)**. Also, an interface has been created to visualize the musical statistics plots and compare them with other performances. The user is able to select his/her own recordings and add them into the database.

This main objective is divided into the following secondary objectives necessary to be able to reach the previous one, which are detailed below.

The initial objective of this work is to develop an automatic audio-score alignment algorithm (online and offline). It is a new proposal in this type of methods that is based on the regularization of the performance tempo.

Once the previous objective has been achieved, the following is the development of a method that allows estimating the musical statistics of the musician's performance. In this case, we have statistics like beat time, tempo and duration or rhythm.

The next objective is based on the implementation of an algorithm that allows the user to compare his/her performance with the performance of another musician.

The final objective of this work is to integrate the above algorithms into an application with a user-friendly interface. This application allows us to visualize comparison charts of the musical statistics of both performances.

The following points detail the objectives that have been met to carry out this Master's Thesis:

- Analysis of the state of the art in automatic score tracking algorithms
- Implementation in MatLab / C of a module to obtain music / score alignment library in an offline approach.
- Implementation in MatLab / C of a module to obtain music / score alignment library in an online approach.

- Implementation of a statistics viewer to compare the played tempo with the tempo of previous interpretations or with other users.
- Testing and promotion of the tool.

4 Materials and methods

4.1 System proposed

This chapter proposes an evaluation of performances based, mainly, on four stages: **training** stage, **alignment** stage, stage of **extraction of musical statistics** and **comparison** stage with another performer from the database. This system has as input, the audio file with the real recording of the user musician and the training file previously generated and stored in our database. Consequently, the user must give to the system the comparison file that contains the information of the performance to compare. From this, the system is able to align the score with the audio, estimate the musical statistics (beat time, tempo and rhythm) and generate comparison information between the two performances. In other words, from the real audio recording, score training and comparison information of other musicians, the user obtains information about his/her interpretation of a certain piece of music and a comparison with others who have played the same.

4.1.1 Previous concepts

Next, we are going to define some concepts of this work that we believe are very useful to understand the operation of the application and to introduce the nomenclature used.

4.1.1.1 Chords, states and onset/offset frames

A score is composed of N musical notes in total, where N is defined as the number of different notes that the musical piece is composed of. In most cases, N is a fairly high number and, therefore, working with isolated musical notes is not efficient.

Due to this, our approach works with combinations of notes or chords, so that the system is more robust and efficient. Each unique note / instrument combination that occurs simultaneously in a score is called a chord or combination, and is denoted by the integer index k = 1,..., K; where K is the total number of different chords that occur throughout the score. For example, suppose a score with three instruments: violin, clarinet, and flute. The chord k = 20 could be made up of the MIDI note 43 of the violin, the MIDI note 56 of the clarinet and the MIDI note 63 of the flute; while the chord k=15 could be composed only of the MIDI note 57 of the clarinet, or the chord k=1 could be a general silence of the three instruments.

A score can be viewed as a consecutive sequence of states, where M is the total number of states. Each state m = 1,..., M is defined by a k chord or a note played isolated, and there can be several states defined by the same chord, so $M \geq K$. If we continue with the previous example, the chord k = 20 could appear several times in the score, for example, in states m = 50, m = 98 and m = 100 and in the start and end frames associated with each onset state = 50, offset = 65, onset = 153, offset = 200, onset = 265 and offset = 275.

In summary, the score is divided into states, so each state have an associated onset and offset frame and in each state a unique combination of notes is given.

The score itself provides us with ideal information on the onset and offset times of each state, but these times are not the same as the performance of the musician. To determine the specific moment that we pass from one state to another in the real performance of the musical piece, we use the audio-score alignment algorithm. In this way, the information of the states that has been obtained from the score can be applied.

4.1.1.2 Regularization constant (λ)

Before detailing the parameters that we have used for the development of the algorithms in this work, I explain one of the most important concepts.

As explained throughout this Master's Thesis, the automatic alignment algorithm that we have developed is based on a regularization constant (**λ**). This constant allows the system to compute the optimal path of the real input signal by applying a regularization of the performance's average tempo. In other words, the value of the constant is more or less high depending on the average tempo of the path. In this way, we ensure that the path estimated by the algorithm is guided by the interpretation tempo.

However, we detail this concept more extensively in the following sections.

4.1.1.3 Parameters used

- In order to obtain the time frames, we have used a **window length** of 128 ms, which is equivalent to 5644 samples if the **sampling frequency** is 44100 Hz. This is a frame length short enough in order to consider it a quasi-periodic signal. The **hop** between frames has been set to 32 ms (1412 samples). For this we have used the **Hanning window**.

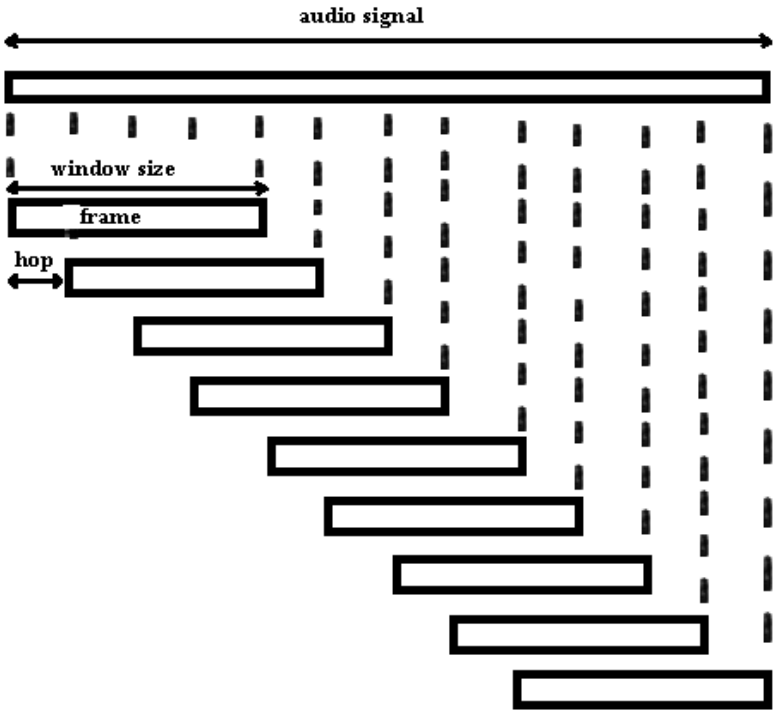


Figure 4.1 – Decomposition into frames of an audio signal.

- The time/frequency representation of the spectrogram is carried out using the **FFT** with 16384 **points** in frequency.

- We work on a **logarithmic frequency scale** with **a semitone resolution**. This is obtained by integrating each and every one of the bins of the transform that belong to the same semitone interval in a single spectral bin.

- When working with **MIDI notes**, the lowest MIDI note considered is 24, and the highest is 137, since most musical instruments are not capable of producing sounds outside this range. In total, a scale of 114 MIDI notes is considered.

- Regarding the parameters for NMF decomposition used in our training stage, we have set the maximum **number of iterations** to 50, which guarantees the convergence of the cost function minimization problem. The cost function used to measure the error between the actual and the estimated spectrogram is the **Beta-Divergence**, with the parameter β = 1.5, between the Euclidean distance (β = 2) and the Kullback-Leibler divergence (β = 1).

- The **optimal regularization constant (λ)** that we have used is: online (λ = 2.4) and offline (λ = 3).

4.1.2 *Summary of the implemented system*

In the application that we have developed, we have used a first stage based on a training block. With this block we are able to extract features and learn the synthetic basis functions from the MIDI score.

When we have completed the training stage, we move on to the alignment stage, which is supported by Carabias' algorithm developed by the **research group TIC-188 from UJA** [22] and explained in the previous section of this report. As indicated previously, this alignment method is based on two parts: **pre-processing (training)** and **alignment**. The algorithm utilizes some processing tools such as **FFT** or **STFT**, **non-negative matrix factorization** (*NMF*) or **dynamic time warping** (*DTW*).

At the output of the alignment block, we introduce the stage of extraction of musical statistics. In this part of the system, we have the optimal minimum cost path that we have computed in the previous stage and it estimates the musical statistics such as beat time, tempo and duration of each beat of the score Each of these statistics is displayed on a plot next to the triangle and tree plots that we explain in more detail in the following sections. It should be noted that, apart from visualizing the musical statistics of our performance, we can compare ourselves with the performance of another musician from our database. In order to see the comparison plots, we must select the file with the musical statistics obtained by the other musician. These statistics are previously stored in the database and the system is in charge of loading them and generating the comparison plots.

The following is the general diagram of the algorithm developed:

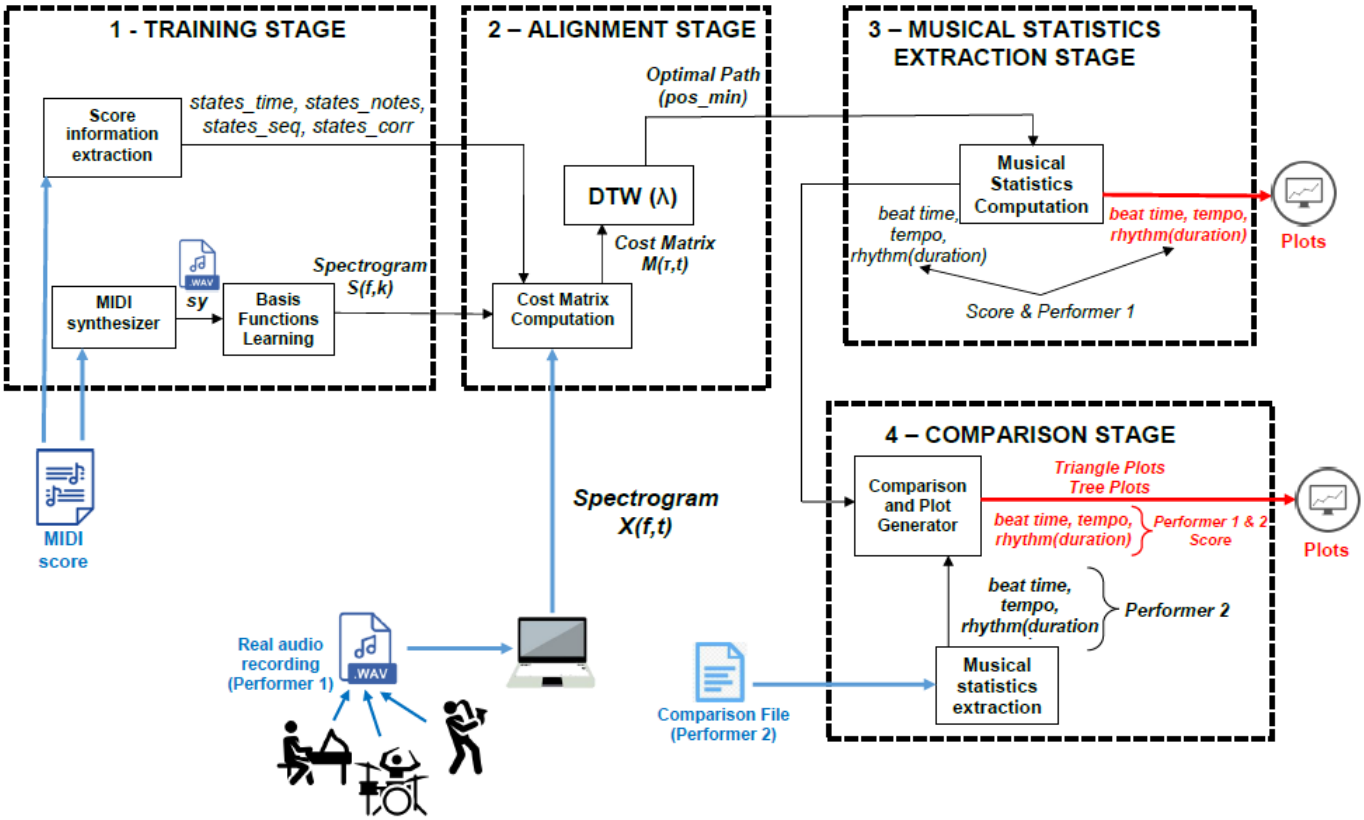


Figure 4.2 – General scheme of the algorithm used with four stages: training, alignment, extraction of musical statistics and comparison between musicians. The input data are: MIDI score, real audio recording and comparison file; and the output data is the plots with the musical statistics.

Finally, it should be said that in this project, the implementation of all the algorithms explained here has not been carried out. In particular, the training modules and part of the alignment have been provided to me by the **TIC-188 research group** of the University of Jaén. I have had to understand how these modules work and how they were implemented in order to make some modifications and integrate them into the rest of the application. The alignment method, in online and offline version, the stage of extraction of musical statistics and comparison between musicians has been fully implemented by me for this work.

4.1.3 Training stage

In this section, I am going to introduce the training block that I have used in this application.

The main purpose of this stage is to obtain the matrix with the basis functions trained with information from the score and synthetic audio and then, we can use all this information in the alignment block.

As previously commented, all the musical pieces of our database are trained and store in advance. Then, the user can choose the training or score information as input in the application.

Inputs:

- *Score* in MIDI format.

Outputs:

- Score information: *states_time, states_seq, states_notes, states_corr*, number of states and chords and *synthetic basis S(f,k).*

Training algorithm summary
- Load the score in MIDI format - Generate the synthetic signal from the score and compute the spectrogram of it - Learn states basis for each combination of notes → S(f,k) - Extract information from the score → states_time, states_seq, states_notes and states_corr

Explanation of the training method:

In this stage, we analyze the MIDI score provided in order to define the set of combinations of simultaneous notes and the transitions between them, I mean, the different states from the MIDI score that we have entered as input. Then, we convert the score into audio signal (synthetic audio) using **synthesizer software** (*Fluidsynth*). When we already have the synthetic signal we use an **NMF-based method with Beta-Divergence** to learn the spectral patterns of each combination of notes.

Next, we have to make a definition of states, that is, we compute each of the note combinations and then define the states as the sequence of these chords. The score is defined as a consecutive sequence of **M** states, where each **m** state is defined by its note combinations (for all instruments). In addition, the score informs about time changes from one state to the next. In fact, any score follower must determine the time of all transitions between states. In a score, there are only **K** unique combinations of notes, this condition being fulfilled $\boldsymbol{M} \geq \boldsymbol{K}$ because some states could represent the same combination of notes.

A **GT(n, τ)** matrix is created that represents the notes that are active in each time frame (see example in *Figure 4.3*) and is decomposed into two binary matrices as follows:

$$GT(n,\tau) = Q(n,k)R(k,\tau) \tag{10}$$

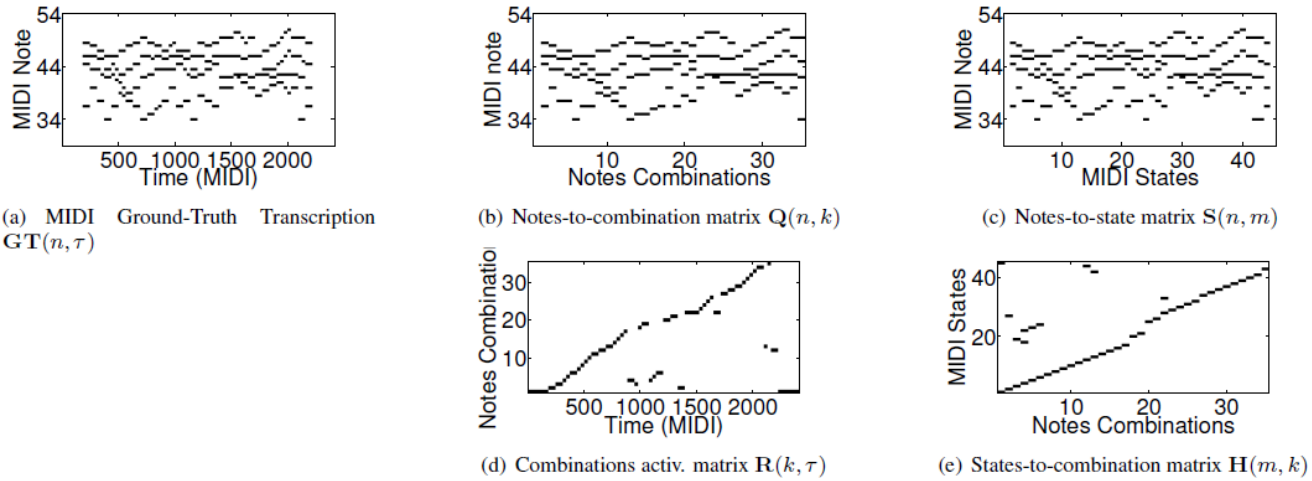
where **Q(n, k)** corresponds to the matrix representing the notes (**n**) and each combination of them (**k**), and **R(k, τ)** represents the matrix with the combination of notes (**k**) in each MIDI frame (τ).

In order to obtain the state information, we have to compute the notes-to-combination matrix **Q(n, k)**:

$$Q(n,k) = S(n,m)H(m,k) \tag{11}$$

where **S(n, m)** is the matrix that represents the notes (**n**) of each state (**m**), and **H(m, k)** represents the matrix that defines the state (**m**) which each combination of notes belongs (**k**) to.

The defined matrices are used in the following stages to perform the alignment and are computed from the MIDI score. In *Figure 4.3,* we can see these matrices described above from an article of the **16th ISMIR Conference (Malaga, 2015)** for the musical piece **"01-AchGottundHerr"** extracted from **The Bach Choral Datasets** [33].



(a) MIDI Ground-Truth Transcription $\mathbf{GT}(n,\tau)$

(b) Notes-to-combination matrix $\mathbf{Q}(n,k)$

(c) Notes-to-state matrix $\mathbf{S}(n,m)$

(d) Combinations activ. matrix $\mathbf{R}(k,\tau)$

(e) States-to-combination matrix $\mathbf{H}(m,k)$

*Figure 4.3 – Example of each matrix for the musical piece "01-AchGottundHerr": **(a)** represents a transcript of notes with the GT matrix, showing the combinations of active notes and the duration (in frames) that remain active. **(b)** represents the notes that belong to each combination of them. **(c)** corresponds to the matrix indicating the combination of notes corresponding to each state. **(d)** shows the combination of notes that is active in each frame in MIDI time. **(e)** represents the state to which each combination of notes belongs.*

After having computed the combination of notes and in which frame each one is active, we move on to the spectral learning stage. At this stage of the training, the system computes the combination of each frame, knowing the different combinations of notes defined in the score. A spectral pattern is defined as a fixed spectrum that is learned from a signal with certain characteristics. The use of a single spectral pattern per combination of notes allows computing distortions with a not very complex signal decomposition method. This means that this method must learn in advance the spectral pattern associated with each unique combination of notes in the score. For this, a method based on **non-negative matrix factorization** (*NMF*) with **Beta-Divergence** is used.

First of all, we define the signal model as follows:

$$Y(f,\tau) = B(f,k)G(k,\tau) \tag{12}$$

where **Y(f, $\boldsymbol{\tau}$)** is the magnitude of the synthetic signal spectrogram, the matrix **G(k, $\boldsymbol{\tau}$)** represents the gain of the spectral pattern of each combination of notes (**k**) and in each frame ($\boldsymbol{\tau}$), and the matrix **B(f, k)**, for k = 1, ..., K, represent the spectral patterns for all note combinations defined in the score.

In *Figure 4.4,* we can see the spectrogram of the synthetic signal of the musical piece **"03_Dance_fl_cl"** created for flute and clarinet extracted from *URMP* database. It should be noted that the successive examples of *Figures* that are shown in the following subsections are about this musical piece.

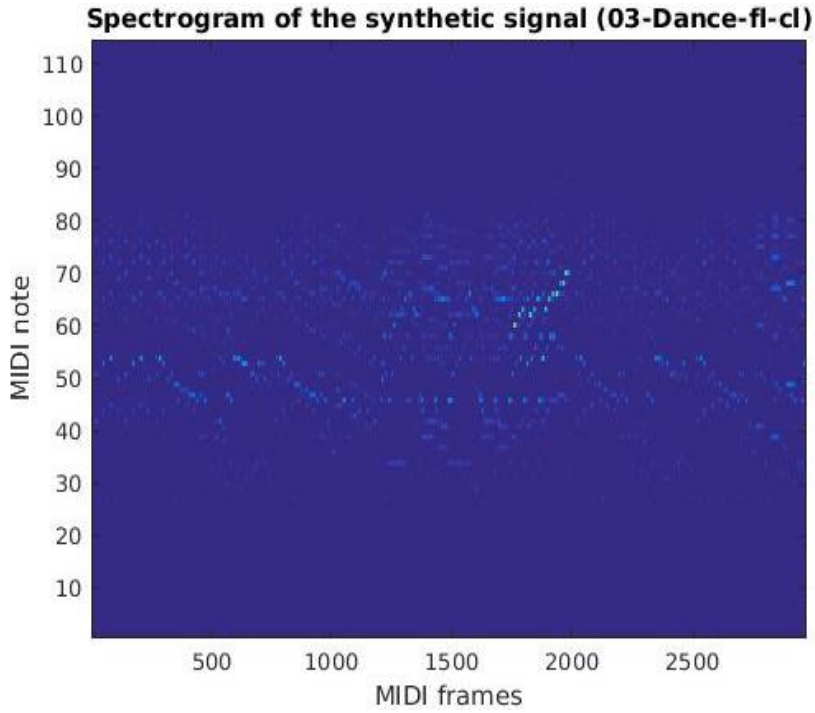


Figure 4.4 – Spectrogram of the synthetic signal for the musical piece "03-Dance-fl-cl".

When the parameters are limited to be non-negative, as in the case of spectral magnitude, a common way to compute factorization is to minimize the reconstruction error between the observed spectrogram and the modeling. The most popular cost functions are the **Euclidean distance** (*EUC*), the one created by **Kullback-Leibler** (*KL*) and the divergences of **Itakura-Saito** (*IS*). In addition, in Beta-Divergence other commonly cost functions are used, which include in their definition the three previously mentioned EUC (β = 2), KL (β = 1) and IS (β = 0). For our application we use a β = 1.5.

The general definition for Beta-Divergence function is:

$$D_\beta(x|\hat{x}) = \frac{1}{\beta(\beta-1)}(x^\beta + (\beta-1)\hat{x}^\beta - \beta\, x\, \hat{x}^{\beta-1}) \quad (13)$$

Finally, the method learns the spectral patterns for each state and for each combination of notes.

Figure 4.5 shows an example of learning the state bases for the previously mentioned musical piece. In this *Figure,* we can see the synthetic bases of each combination of notes associated with each state, where the first state corresponds to the state of silence of the score.

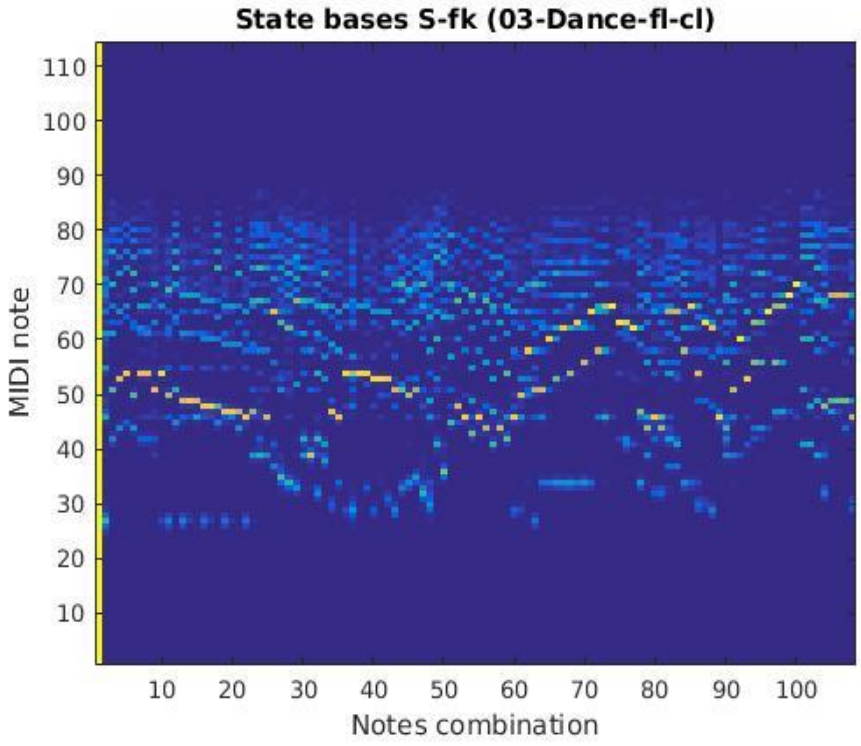


Figure 4.5 – State bases (S(f,k)) for the musical piece "03-Dance-fl-cl".

Apart from defining the states of the score and estimating spectral patterns, all the necessary information is extracted from the MIDI score and organized into **states/chords/frames**. The information is divided into four tables, which is very useful for our application:

- **states_notes**: indicates the combination of notes that make up each unique chord (k). Size: 1 x number of chords (K)
- **states_time**: indicates the start frame and end frame of each state. Size: 2 x number of states
- **states_seq**: shows, for each state, the chord (k) that belongs to it. Size: 1 x number of states
- **states_corr**: indicates with a binary 1 when a state is an anchor point. Size: 1 x number of states

The following *Figure* shows an example of a part of the mentioned tables of signal "03_Dance_fl_cl" from the URMP database.

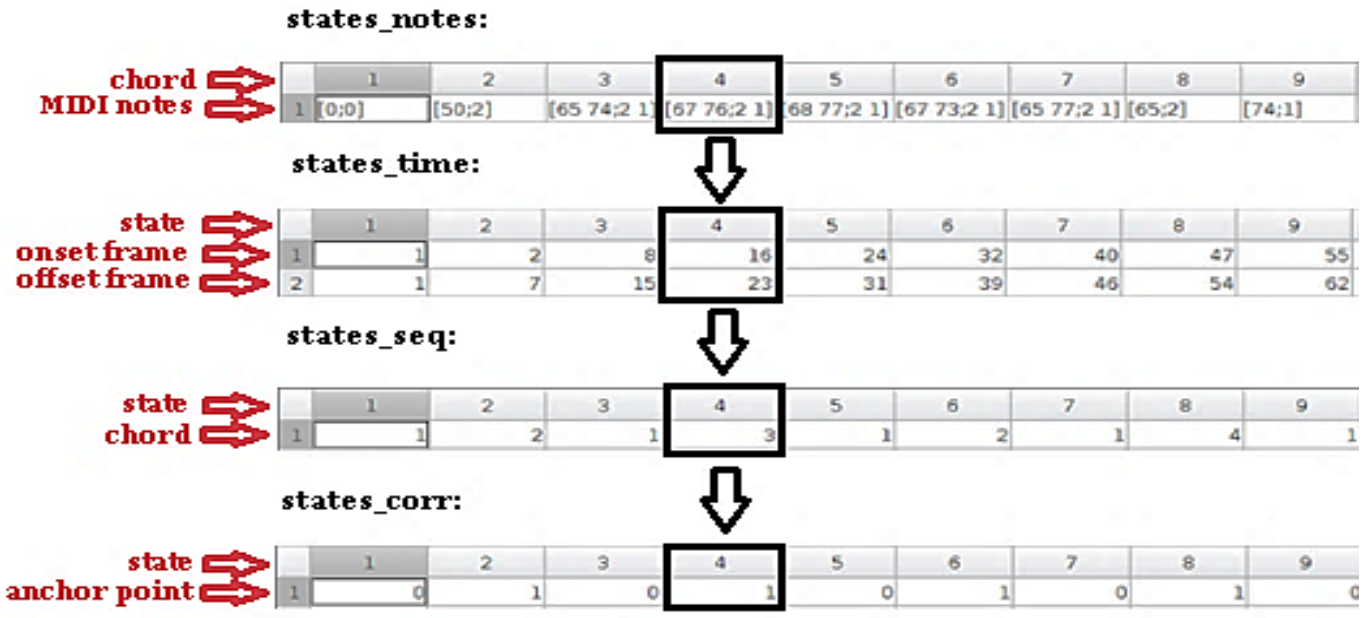


Figure 4.6 – Example of states_notes, states_time, states_seq and states_corr tables. If we focus on the 4th state, we can see that the onset frame is 16 and the offset frame is 23, this state is an anchor point, it corresponds with the chord k=3 and is composed by the MIDI notes 67 and 76.

These variables have the same information as the matrices in *Figure 4.3*, but from a programming point of view they require less memory for their implementation.

After having obtained the score information that we need, we can move on to the alignment stage.

4.1.4 Alignment stage

In this part of the report, I am going to show the audio-score alignment stage used and implemented in this application. This algorithm is based on **spectral factorization** and **Dynamic Time Warping** (*DTW*) with a **regularization value (λ)** that allows us to make the cost variable depending on a series of constraints.

The main purpose of this stage is to achieve the relationship between the times set by the musician's performance the score. This is determined by the minimum cost path computed by our **variable cost DTW algorithm**.

The method that we have implemented, in **online** and **offline** version, is based on another algorithm developed by **Carabias et al.** [22]. The only difference between these two methods is: the one that we have developed in this work computes the alignment guided by a **λ** value that regularizes the cost.

Inputs:

- Real audio signal (musician performance): *x(t)*
- Score information extracted in the training stage: *states_time, states_seq, states_notes, states_corr* and *synthetic basis S(f,k)*

Outputs:

- Optimal path: *pos_min*

Alignment algorithm summary
- Compute transform of the real input signal with beta normalization - Compute distorsions matrix for each combination - DTW algorithm to obtain the optimal path

Explanation of the alignment method:

The alignment stage receives as input the learned and trained parameters of the score: **states_notes, states_time, states_seq, states_corr** and the **synthetic bases S(f,k)** with **semitone resolution**. In addition, it receives as input the real audio signal **x(t)** that contains the musician's performance.

The alignment algorithm used and explained in *section 2.2.4.2*, has two different stages: **pre-processing** and **alignment**. The first stage has been carried out during our training stage. In the second stage, the alignment between the real audio and the score is carried out using a *DTW* method.

4.1.4.1 Algorithm behavior

Throughout this subsection I am going to show plots of a specific example in order to explain how this algorithm works: the musical piece **"03_Dance_fl_cl"** extracted from the *URMP* database.

First, the system takes the audio file and computes the Fourier Transform to convert it from time domain to frequency domain (**spectrogram**). In the following *Figure,* we can see the spectrogram of the real signal of the example:

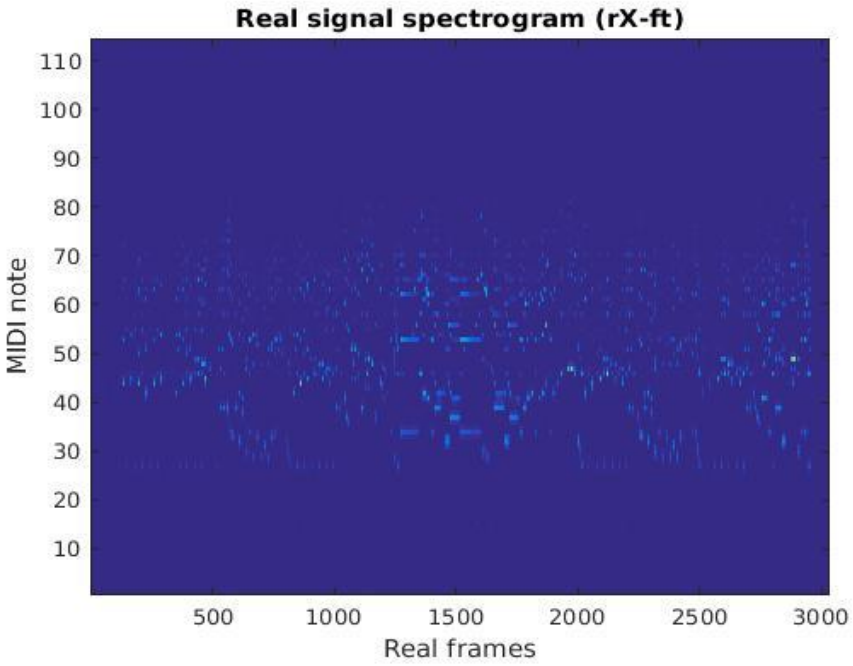


Figure 4.7 – Real signal spectrogram of the example.

When we have the spectrogram of the real signal, it is important to apply a Beta normalization. Beta normalization consists of estimating a norm of each frame to make sure that the Beta value is the one we have chosen. For this, the norm value is computed according to the following equation:

$$norm = \left[\sum_{i=0}^{N_{MIDI}} \left(X[i]^{\beta} \right) \right]^{\frac{1}{\beta}} \tag{14}$$

where X is the vector associated to each frame of the matrix, i is the MIDI note and β is the Beta value (in our case β = 1.5).

Once we have the norm value, we can normalize the matrix, that is, we divide element-by-element the vector associated to each frame by this norm value.

In the following *Figure,* we can visualize the spectrogram with Beta normalization.

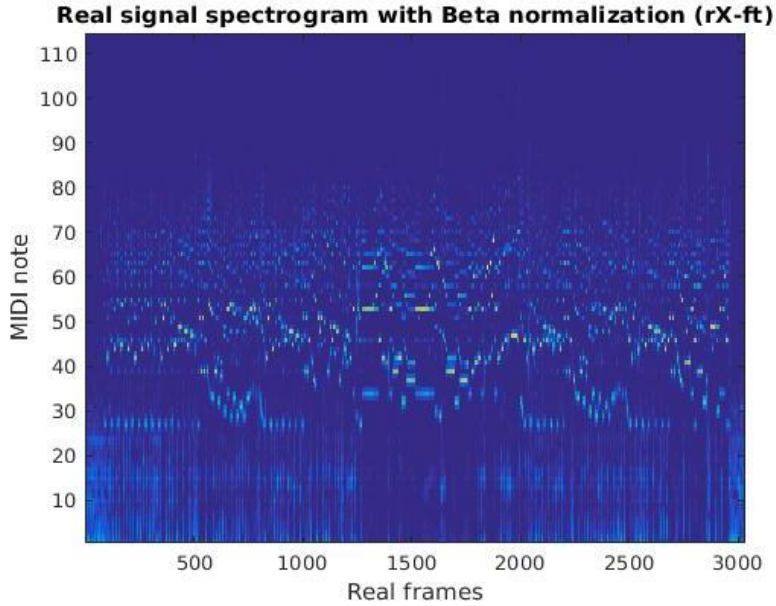


Figure 4.8 – Real signal spectrogram with Beta normalization (β = 1.5) of the example.

Then, we also apply the Beta normalization to the synthetic basis (*S(f,k)*) as shown in *Figure 4.9*.

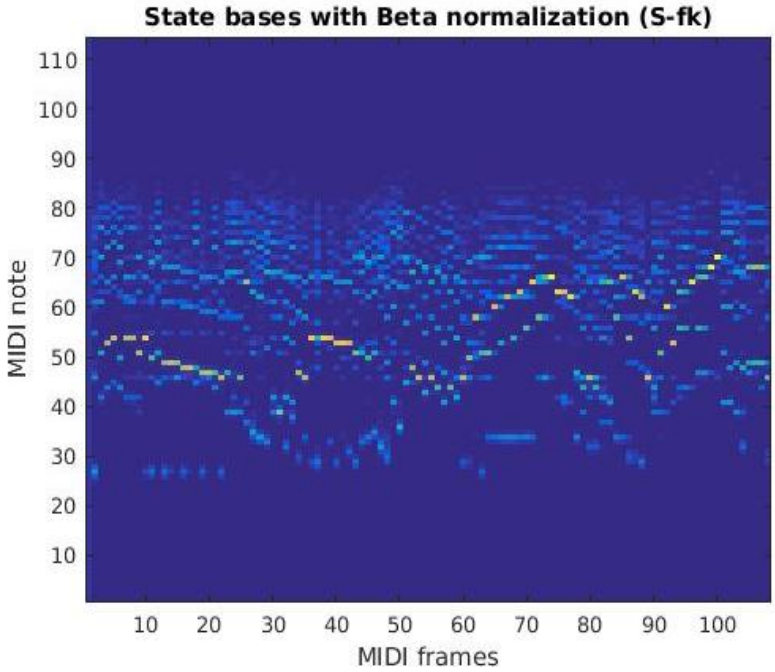


Figure 4.9 – Synthetic bases with Beta normalization (β = 1.5) of the example.

After this, the system computes the amplitude and distortion matrix of the monophonic decomposition model for each frame of the real signal **spectrogram** (*X(f,t)*) and for each of the **synthetic basis** (*S(f,k)*) in order to obtain the distortion between the real and synthetic signal.

Once we have each previous matrix computed, we can start the *DTW* process. It should be noted that we have developed this *DTW* process in different versions and in different programming languages as indicated in the list below:

- **DTW online** version in MatLab/C programming language.
- **DTW offline** version in MatLab/C programming language, which processes the input signal in blocks of N frames, that is, it does not process the entire signal at the same time.

In the previous section, we assumed that we had a temporal alignment between the score and the performance times. In order to automate this process, there are many methods that we can use to estimate a temporal alignment between audio and score. To date, *DTW*-based methods have been shown to provide the best alignment results in the *MIREX* Real-time Audio-to-Score Alignment task, and for this reason, we have chosen this method as the basis for our alignment process. Next, I am going to explain the DTW process in general and later I emphasize the difference between online and offline version.

DTW process

The signal model we use assumes that when combination k is active all the other combinations are inactive. For that reason, we can obtain the optimum combination at each frame directly computing the divergence from the input data X(f,t) and the trained spectral patterns S(f,k).

We are going to define two time vectors that represent the time series to be aligned: $\mathbf{U = (u_1, ..., u\tau, ..., u_{Tm})}$ and $\mathbf{V = (v_1, ..., v_t, ..., v_{Tr})}$, where τ represents the MIDI time series and t the real time series.

The first step in the DTW process is to estimate the **local distance matrix** (**cost matrix**) $\mathbf{M}(\tau, t)$, where **M** represents the match cost between every two points in the time series. The function associated with matrix **M** could be any cost function that returns 0 for a perfect match and a positive value otherwise.

We know that we have computed the input signal transform for each real frame t and the synthetic signal transform for each MIDI frame τ. The latter shows the spectral pattern associated with each combination of notes in each MIDI frame τ.

From this data, the system goes through all the states of the score and replicates the spectral pattern associated with each active note combination at each moment of the performance. In other words, the system goes through all the MIDI frames of each state and inserts in the matrix the spectral pattern corresponding to the combination of notes active in each part of the score for all the real frames.

The above cost function provides us with information about the similarity of the spectral pattern of each combination with the spectrum of the real signal at each frame t. In this way, we can compute the cost matrix M between MIDI time and real time. In the *Figure 4.10*, we can visualize the cost matrix of the example.

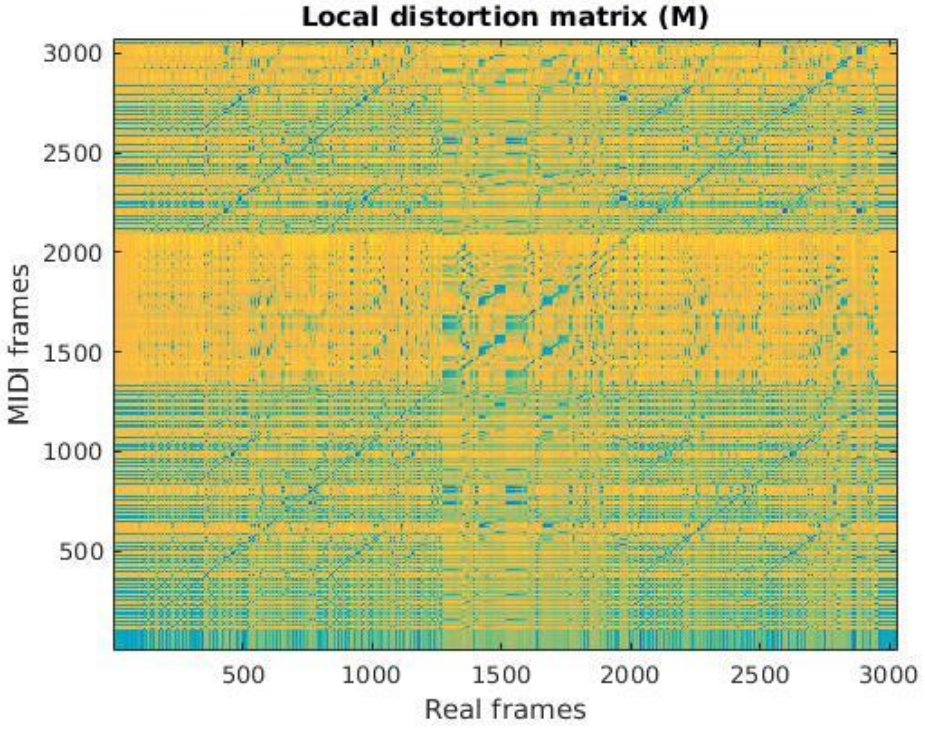


Figure 4.10 – Local distortion matrix M (τ, t) of the example.

Secondly, the **warping matrix D** is computed as follows:

$$D(\tau,t) = min \begin{cases} D(\tau-1,t-1) + \gamma_{1,1} M(\tau,t) \\ D(\tau-2,t-1) + \gamma_{2,1} M(\tau,t) \\ \vdots \\ D(\tau-\alpha_r,t-1) + \gamma_{\alpha_r,1} M(\tau,t) \\ D(\tau-1,t-2) + \gamma_{1,2} M(\tau,t) \\ \vdots \\ D(\tau-1,t-\alpha_t) + \gamma_{1,\alpha_t} M(\tau,t) \end{cases} \quad (16)$$

where the step size at each dimension has a range from 1 to α_r and from 1 to α_t respectively. $\boldsymbol{\alpha_r}$ and $\boldsymbol{\alpha_t}$ are the maximum step size in each dimension. Parameter **γ** is the value for the cost model and controls steps on the diagonal, it is the parameter that regulates the cost of the system. **D** is the accumulated cost of the minimum cost path.

In *Figure 4.11*, we can see an example of the accumulated cost matrix D:

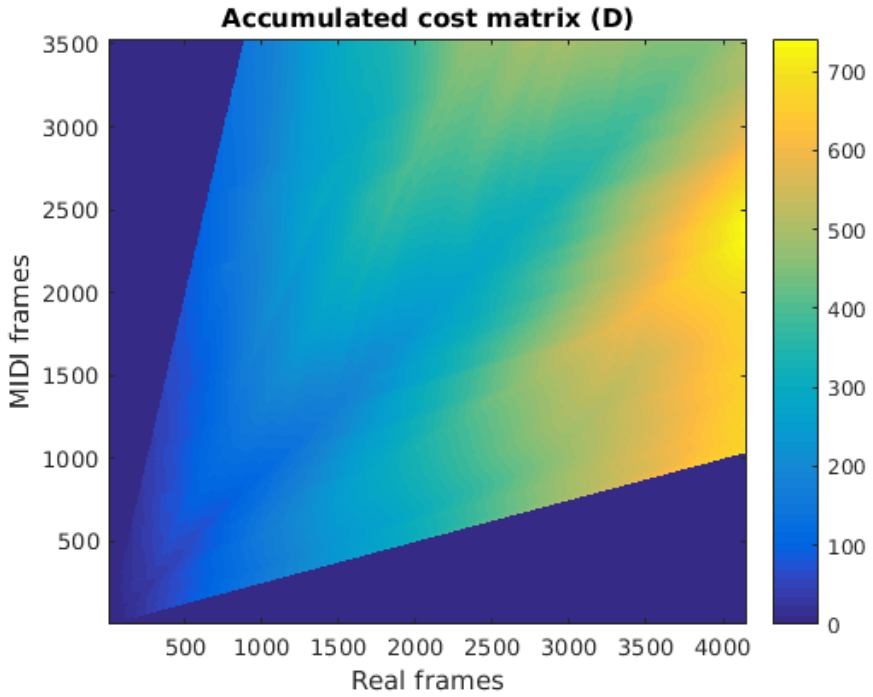


Figure 4.11 – Accumulated cost matrix D (τ, t) of the example.

Finally, the minimum cost path is an ordered pair **(τ,t)** which means that an instant **τ** corresponds to an instant **t**. In order to compute the optimal path with the minimum cost, it is necessary to define the **cost model** on which our system is based, since it is a regulated or variable cost algorithm depending on a **regularization parameter** (**λ**). In

addition, this algorithm is guided by the tempo of the interpretation, that is, those cells that follow the tempo I am performing have priority or advantage over the others.

Now, I have to explain the concept of step. The steps are defined in the real time axis and in the MIDI time axis and mark the possible cells that can jump our path to obtain the minimum cost. Basically, the steps define the possible cells that the path can jump to and depending on the cell that the algorithm decides, it has one restrictive weight or another. The penalties are given by the weights that are applied to the real time of the path depending on whether the system moves away or approaches the correct one. If the system moves away from the correct one, a higher restrictive weight is applied and if it approaches, a lower restrictive weight is applied. The weights are defined by the following vector, which is called the cost of the geometry: **C = [1 1 1 1 2 3 4]**. Depending on the optimal step chosen by the system, it multiplies by one or the other of the values of vector C in order to achieve a more or less high distortion value, as shown in *equation (16)*. The values of vector C correspond with the times 1,1,1,1,2,3,4 of the columns (t) of *equation (16)* to apply it to *equation (17)*.

The steps that we allow in our method are detailed in the *Figure 4.12*:

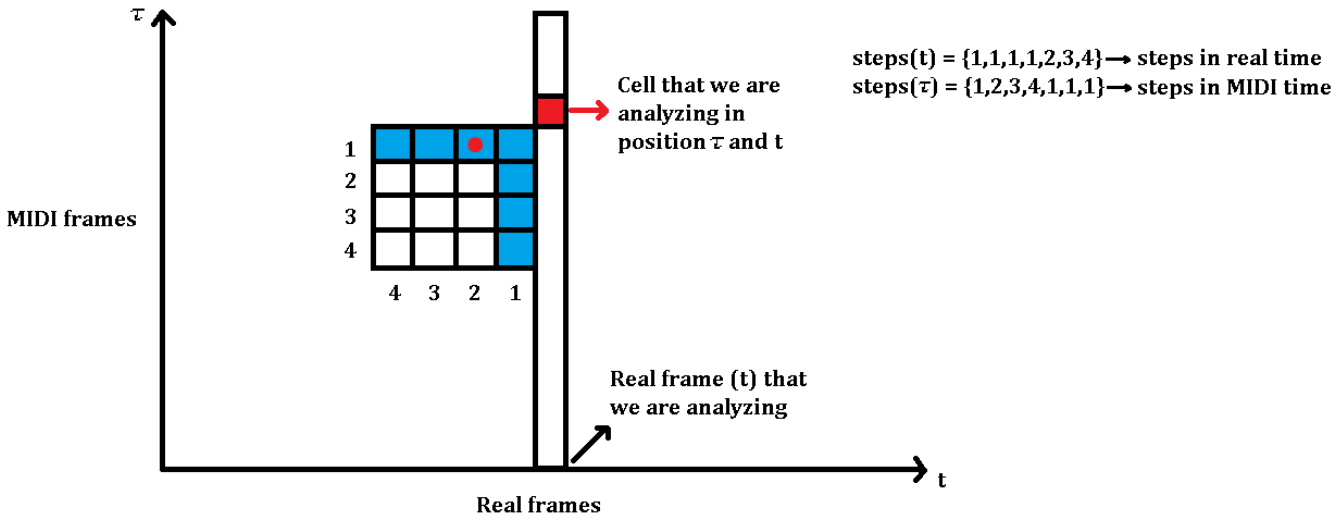


Figure 4.12 – Steps between cells that our system allows. The blue cells are the possible cells depending on the steps allowed and the red cell is the one that our system is analyzing.

The definition of the **cost model** that we propose in this work is:

$$\gamma(m,n) = n + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \qquad (17)$$

where m is the steps (in frames) in MIDI time from 1 to α_r, n is the steps (in frames) in real time from 1 to α_t, **λ** is the regularization constant and it defines the inertia of the system, **σ** and **μ** is the standard deviation and the average tempo of the path respectively, and **p** is the probability of the estimated instant velocity that follows a Gaussian distribution.

If we develop equation (17) for the different cost values, we obtain the following expression:

$$\begin{cases} \gamma(1,1) = 1 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(2,1) = 1 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(3,1) = 1 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(4,1) = 1 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(1,2) = 2 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(1,3) = 3 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \\ \gamma(1,4) = 4 + \lambda \frac{\sigma}{\mu} \frac{1}{10} + \lambda (1-p) \end{cases} \quad (18)$$

If we look at the previous definition of the cost model in the *equation (17)*, it consists of three terms. The first term sets the cost imposed by the geometry that the system must follow. The second term allows the system to penalize changes in rhythm, that is, it increases when the variance of the path increases. And the third term is the probability model that is 0 when the instant velocity coincides with the average tempo of the path.

As explained previously, the optimal path that our algorithm follows is with the minimum possible cost like any DTW algorithm. By contrast, in this work we propose a method in which, in addition to finding the path with the minimum cost, it is also regulated by changes in rhythm and by a probability model. This allows the system to be robust against possible errors.

For example, if we observe *Figure 4.12*, we can see that we are analyzing the red cell, so, we always have to find the blue cell with the minimum possible cost in each iteration and in each cell of the **warping matrix**, assuming that the array is made up of **MIDI frames x real frames cells**. In this way, we build the optimal path.

As we already know, **p** is the probability that the instant velocity coincides with the average tempo. This probability follows a Gaussian distribution and must meet certain conditions based on the partial tempo or instantaneous speed of the performance and the average tempo (**μ**) that the path has stored. That is, we define a function that returns a value 1 when the partial tempo of the cell I am analyzing exactly matches with the average tempo (**μ**) that the path takes and return a smaller value if those values do not match.

Partial tempo or instant velocity is defined as the ratio between the possible step in MIDI and real time:

$$Vi = \frac{steps\ (\tau)}{steps\ (t)} = \frac{MIDI\ time}{Real\ time} \tag{19}$$

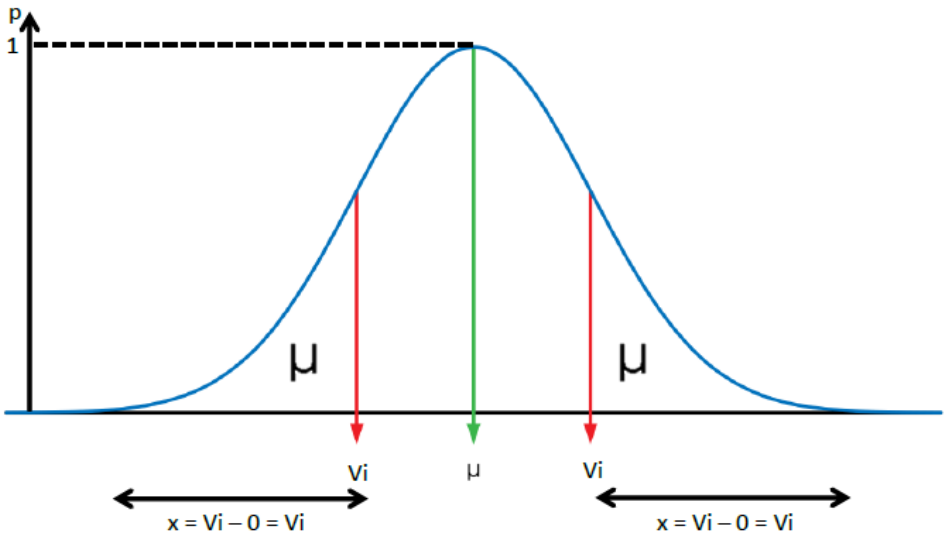
We can define probability (**p**) as follows:

$$p = 2 * CDFGAUSS(x, \mu, \sigma) \qquad \begin{matrix} \mu \in R \\ \sigma > 0 \end{matrix} \tag{20}$$

$$CDFGAUSS\ (x, a, s) = \frac{1}{2}\ erfc\left((a - x)\frac{\sqrt{2}}{2\ s}\right) \tag{21}$$

where **erfc** is the complementary error function, **a** and **s** are the average tempo and the standard deviation of each cell, respectively, and **x** is related to **Vi** and **μ** as we can see in the *Figure 4.13*.

When Vi < μ:



When Vi > μ:

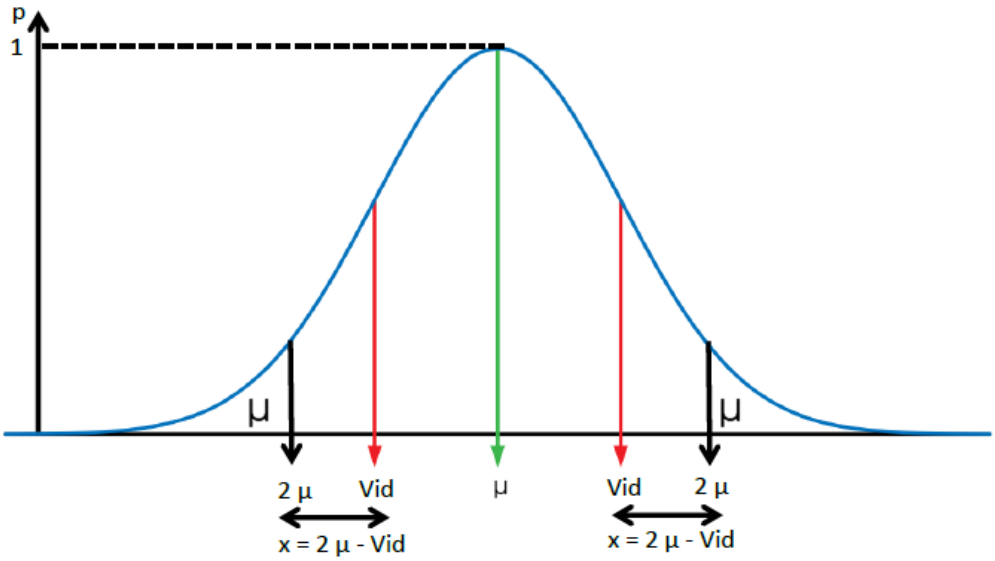


Figure 4.13 – Definition of probability of our system (Gaussian distribution).

Next, we have to compute the difference in tempo when the path goes from one cell to another. To do this, we apply the definition of the slope of the line in which we have two pairs of coordinates and measure its slope (see *Figure 4.14*).

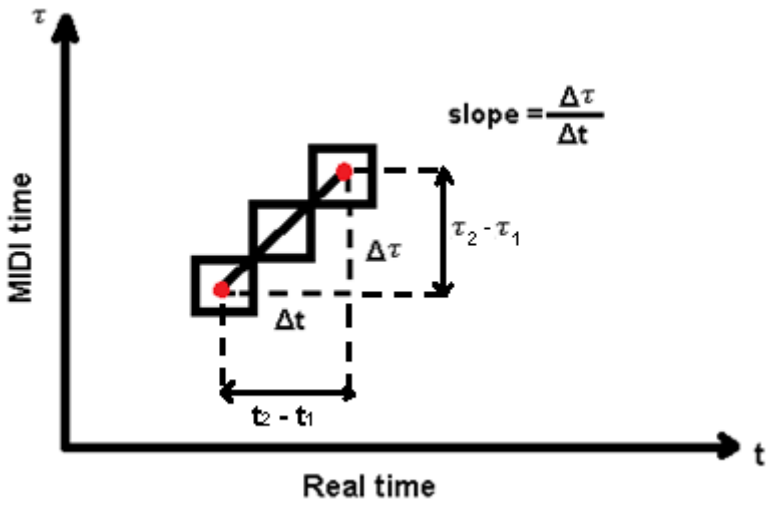


Figure 4.14 – Definition of slope.

The definition of tempo that we have used is the following:

$$tempo = \frac{\tau - v\_tau(Pos\_\tau, Pos\_t)}{t - v\_t(Pos\_\tau, Pos\_t)} \quad (22)$$

where $\boldsymbol{\tau}$ and **t** are the coordinates in MIDI time and real time of the cell that the system is analyzing, respectively. $\boldsymbol{Pos\_\tau}$ and **Pos_t** are the coordinates in MIDI time and real time of the new cell that the system has computed as optimal, respectively. The term **v_tau(Pos_$\boldsymbol{\tau}$, Pos_t)** represents the MIDI time $\boldsymbol{\tau}$ that is stored in the MIDI time count matrix in coordinates **($\boldsymbol{Pos\_\tau}$, Pos_t)**. The term **v_t(Pos_$\boldsymbol{\tau}$, Pos_t)** represents the value of **t** that is stored in the real time count matrix in coordinates **($\boldsymbol{Pos_\tau}$,Pos_t)**.

The definition of average tempo (**μ**) and standard deviation (**σ**) corresponds to the following equations:

$$\mu = v\_mean(Pos\_\tau, Pos\_t) + \frac{tempo - v\_mean(Pos\_\tau, Pos\_t)}{n} \quad (23)$$

where $\boldsymbol{v\_mean(Pos\_\tau, Pos\_t)}$ represents the average tempo value that is stored in the average tempo matrix in coordinates **($\boldsymbol{Pos\_\tau}$, Pos_t)**, tempo is the value that has been computed previously and **n** is the anchor point value.

$$\sigma = \sqrt{\frac{(v\_Stds(Pos\_\tau, Pos\_t)^2 * (n-1) + (tempo - v\_mean(Pos\_\tau, Pos\_t)) * (tempo - \mu)}{n}} \quad (24)$$

where $\boldsymbol{v\_Stds(Pos\_\tau, Pos\_t)}$ represents the standard deviation value that is stored in the standard deviation tempo matrix in positions **($\boldsymbol{Pos\_\tau}$,Pos_t)**, $\boldsymbol{v\_mean(Pos\_\tau, Pos\_t)}$ represents the average tempo value that is stored in the average tempo matrix in positions **($\boldsymbol{Pos\_\tau}$,Pos_t)**, tempo is the value that has been computed previously and **n** is the anchor point value.

It is important to note that the tempo, mean and standard deviation values are only updated when the step between cells implies a change of state and also a new **anchor point**. That is why the definition of our cost function is regulated by these updated values.

Then, it is very useful to explain the **"anchor" point concept**. It is true that local restrictions that are carried out in the DTW technique help to get the estimated path on a regular basis, but there are some places in the score where the algorithm is not sure where to go. Also, there are other places where the certainty of each state which is being played is too high. This certainty occurs in states with notes that rarely appear in the score. If we calculate the cross-correlation of each state, it gives us lower results than computing it for a state with notes that appears more often in the score. These places in the score are taken as "anchor" points. Before starting the alignment process, a correlation between each state and the rest is calculated and the total similarity between

each state and the others is computed. For this reason, the concept of "anchor" point is created to help the alignment process decide on one path or another when the certainty threshold is too low.

In *Figure 4.15*, we can see the optimal path (red line) associated to the cost matrix M. This path is obtained once we have finished the DTW process.

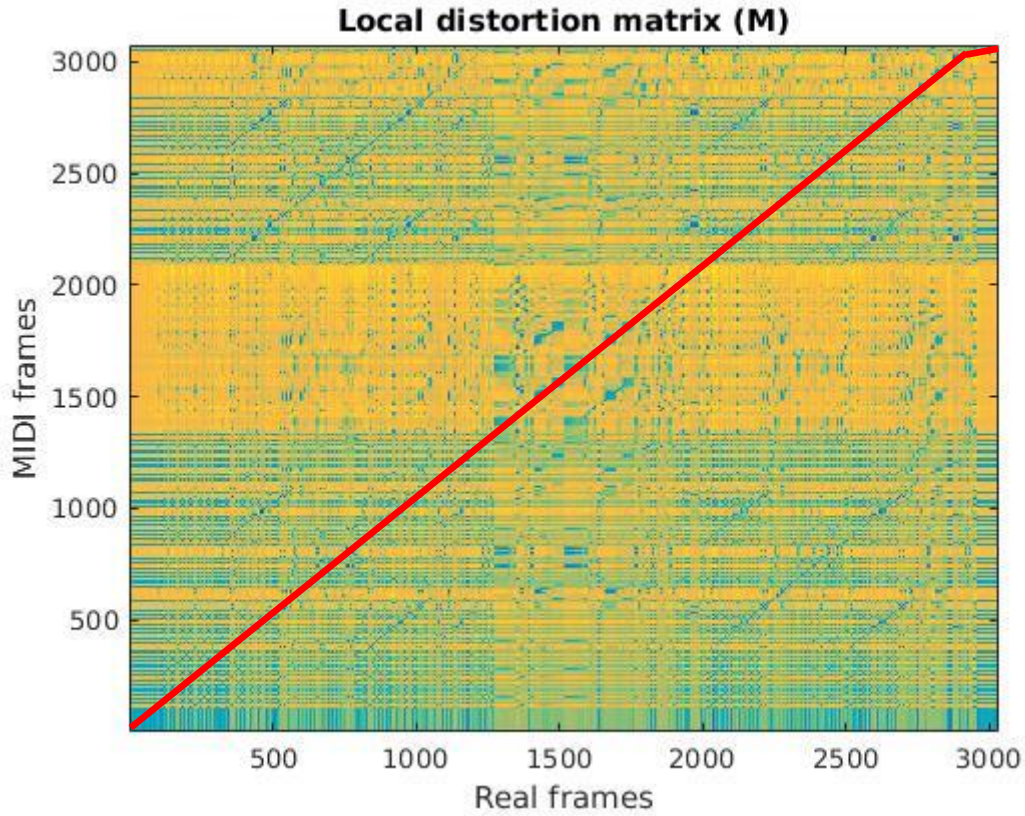


Figure 4.15 – Alignment path (red) on the distortion matrix.

After showing the parameters that are computed, I am going to define the matrix in which each of those values is stored for each MIDI and real frame. I have to define some matrices that are useful to explain this algorithm and that serve to store the information of the path. The matrices are as follows and next to the definition of each matrix we can visualize an example extracted from the alignment process of the musical piece "**03_Dance_fl_cl"** from the *URMP* database:

- **Real time step matrix** (*v_step_t*): stores the real-time step value of each cell, that is, it contains the optimal step decided in *equation (16)* for the real-time axis, from 1 to α_t.

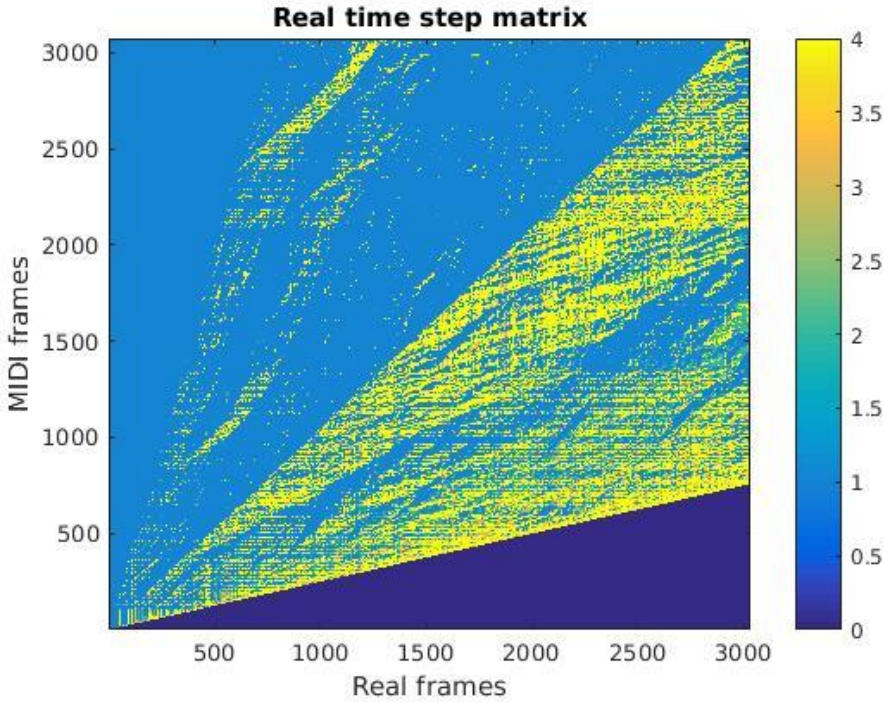


Figure 4.16 – Real time step matrix (v_step_t) of the example. The value range of the array is from 0 to 4. 4 is the maximum step value in real time that the system supports.

- **MIDI time step matrix** (*v_step_*τ): stores the MIDI time step value of each cell, that is, it contains the optimal step decided in *equation (16)* for the MIDI time axis, from 1 to α_τ.

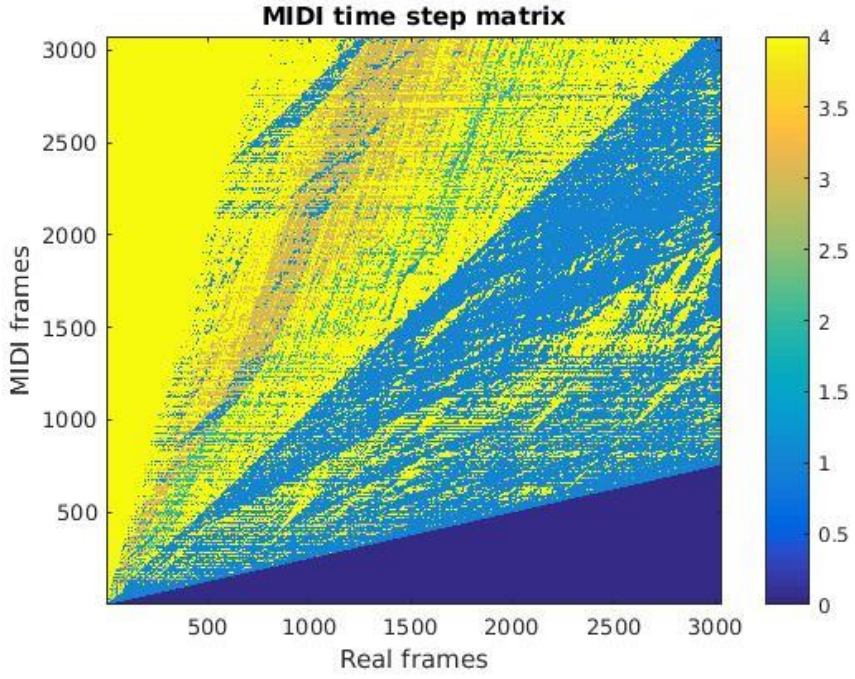


Figure 4.17 – MIDI time step matrix (v_step_*τ) of the example. The value range of the array is from 0 to 4. 4 is the maximum step value in MIDI time that the system supports.*

- **Real time count matrix** (*v_t*): stores the real frame of the optimal step obtained in *equation (16)* for the real-time axis, keeping in mind that the value is updated when there is a state change and that state change implies a new anchor point.

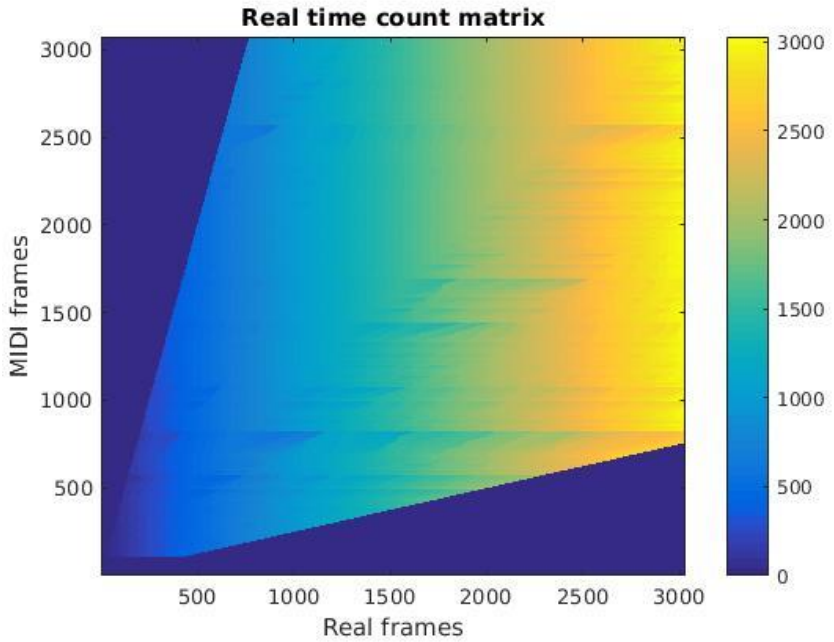


Figure 4.18 – Real time count matrix (v_step_t) of the example. The value range of the array is from 0 to 3000 frames.

- **MIDI time count matrix** (*v_τ*): stores the real frame of the optimal step obtained in *equation (16)* for the MIDI time axis, keeping in mind that the value is updated when there is a state change and that state change implies a new anchor point.

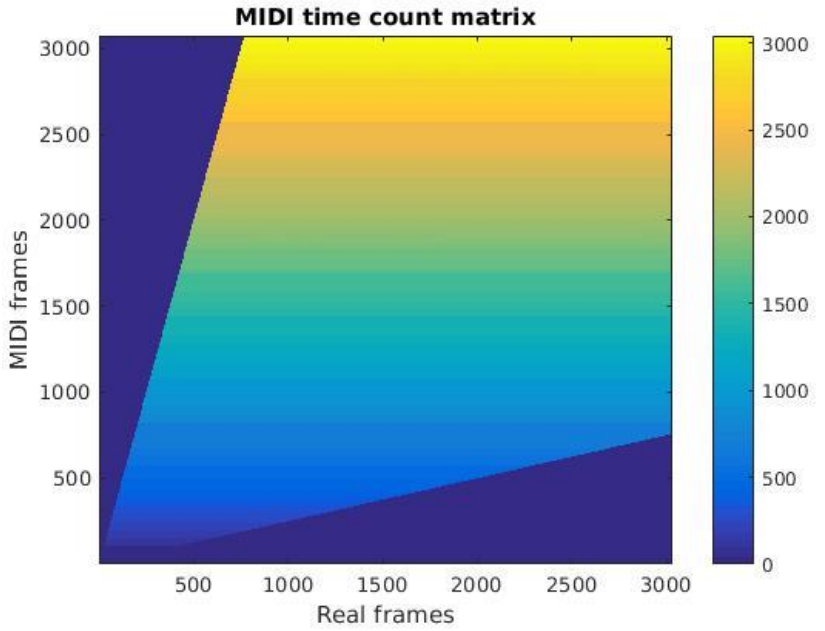


Figure 4.19 – MIDI time count matrix (v_step_τ) of the example. The value range of the array is from 0 to 3000 frames.

- **Anchor point matrix** (*v_numPA*): stores the anchor point number of each cell.

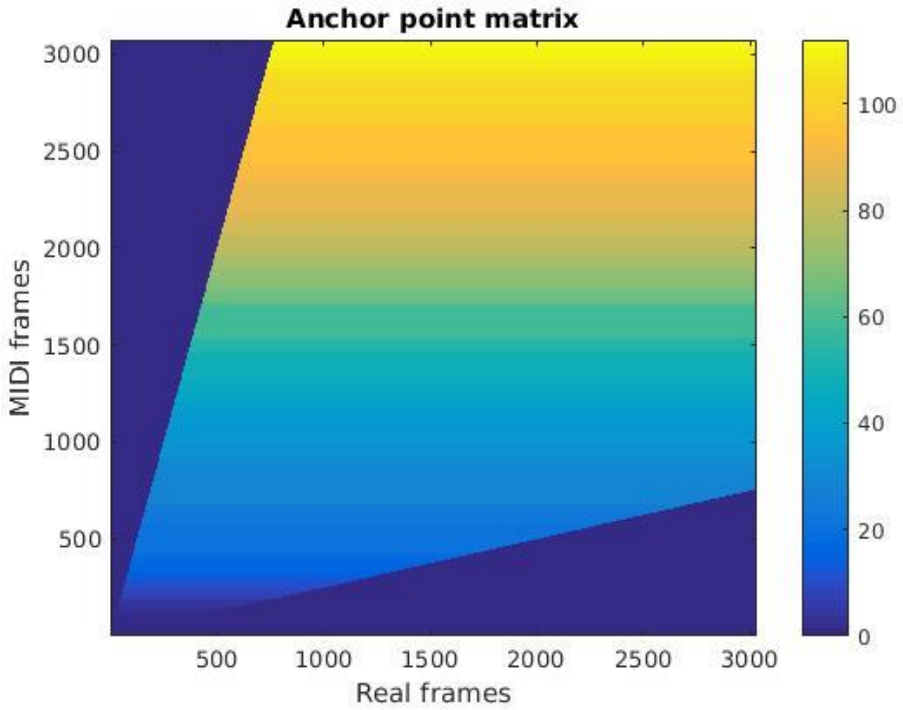


Figure 4.20 – Anchor point matrix (v_numPA) of the example. The value range of the array is from 0 to 120 more or less.

- **Average tempo matrix** (*v_mean*): contains the tempo average of each cell computed in *equation (22)*.

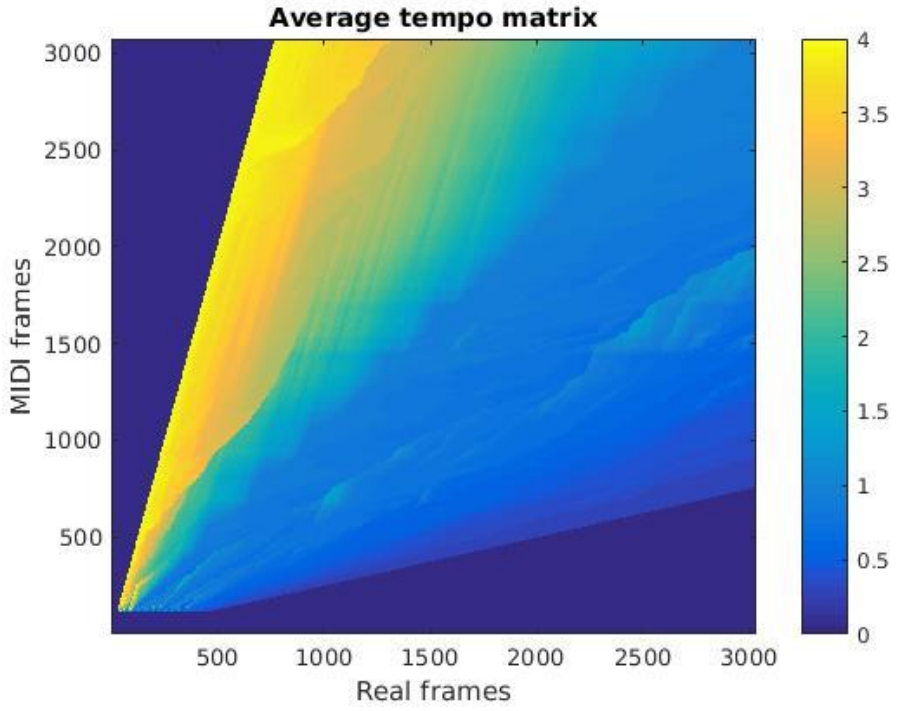


Figure 4.21 – Average tempo matrix (v_mean) of the example. The value range of the array is from 0 to 4.

- **Standard deviation tempo matrix** (*v_Stds*): Contains the standard deviation of each cell computed in *equation (23)*.

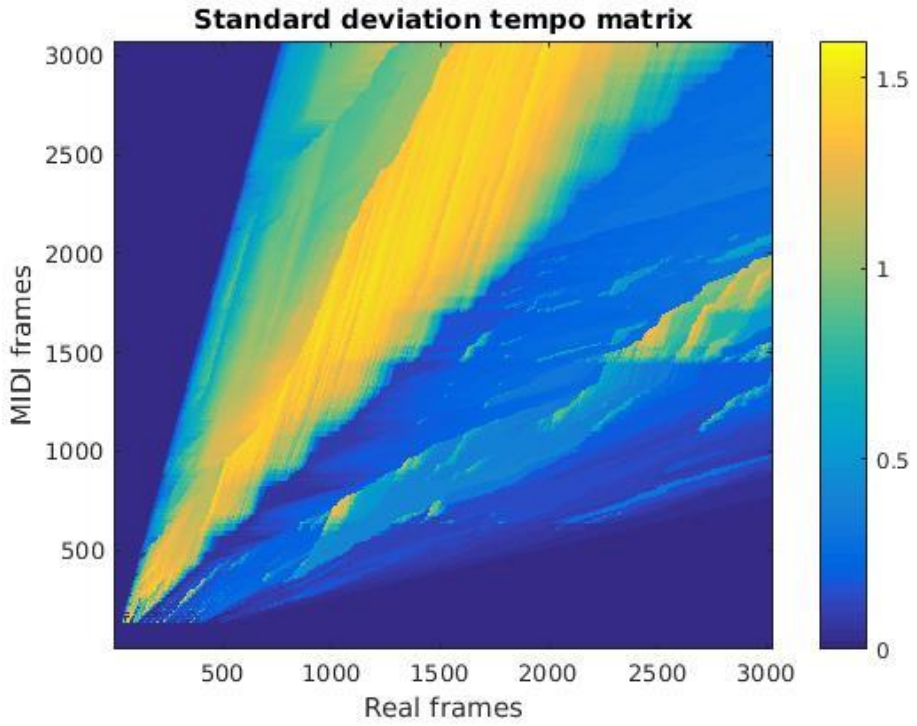


Figure 4.22 – Standard deviation tempo matrix (v_step_ τ) of the example. The value range of the array is from 0 to 1.6.

Numerical example

Next, I am going to show a numerical example of cell choice and matrix update in which I explain more clearly how the system works:

Let's suppose that we are analyzing the cell in the real frame **t = 200** and MIDI frame $\boldsymbol{\tau}$ **= 50**. So, taking into account the steps that our system allows between the cells in each matrix, our scenario would be the following:

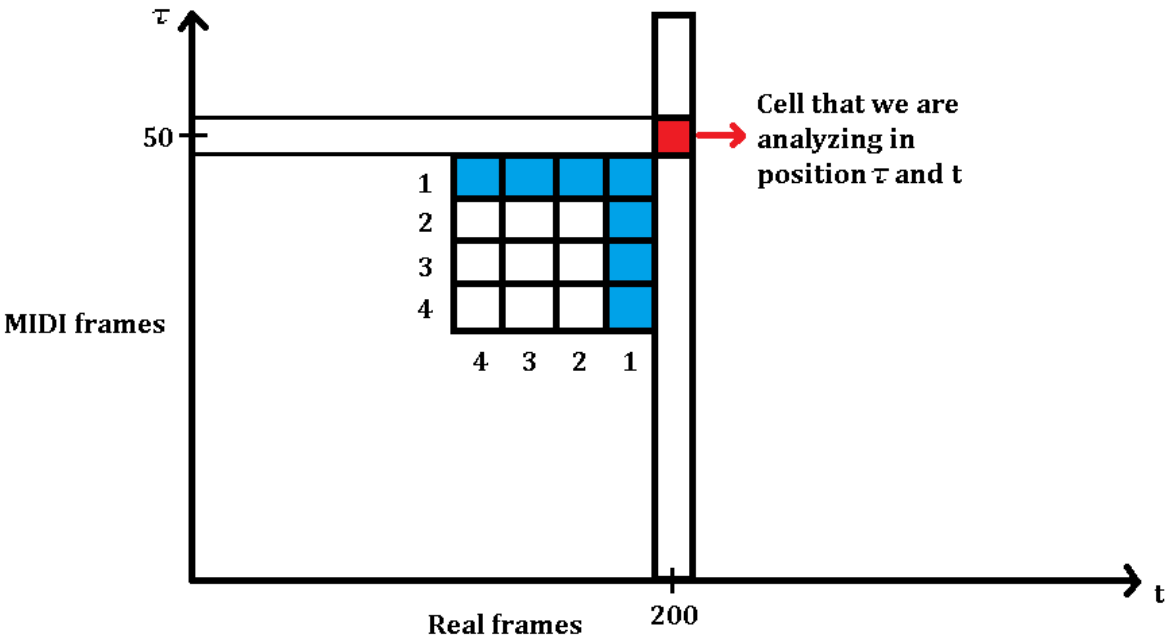


Figure 4.23 – Scenario of the example. We can see the representation of the matrix in which the system has to compute the optimal path. The blue cells represent the possible cells by which the path can pass, taking into account the steps that the system supports, and the red cell is the one that the system is analyzing.

First, our system has to estimate the cell with the minimum cost by computing the distortion associated with each possible cell and the cell with the lowest distortion value is chosen. This is accomplished with *equations (16)* and *(17)*.

As an example, let's assume that the chosen cell has turned out to have the following steps: **step_t = 1** and **step_ τ = 2**. Therefore, we compute the information associated with these steps:

- **Coordinates of the cell of each matrix**:

  $Pos\_t = t - step\_t = 200 - 1 = 199$

  $Pos\_\tau = \tau - step\_\tau = 50 - 2 = 48$

  With these values, we can say that the optimal cell through which the path comes is in the coordinates **(τ, t) = (48,199)**

- **Partial tempo or instantaneous speed**:

  $$Vi = \frac{step\_\tau}{step\_t} = \frac{2}{1}$$

- **Probability**:

  If Vi > v_mean(Pos_ τ,Pos_t) → Let's assume that in this case: **v_mean(Pos_ τ,Pos_t) = 1.9** and **v_Stds(Pos_ τ,Pos_t) = 0.3**

  $p = 2 * CDFGAUSS(Vi, \text{v\_mean}(\text{Pos\_}\tau, \text{Pos\_t}), 1) = 0.342$

- **Cost (λ = 3)**:

$$cost = 1 + \lambda \frac{\text{v\_Stds(Pos\_}\tau\text{, Pos\_t)}}{\text{v\_mean(Pos\_}\tau\text{, Pos\_t)}} \frac{1}{10} + \lambda (1 - p)$$

$$= 1 + 3\frac{0.3}{2.1}\frac{1}{10} + 3(1 - 0.342) = 3.017$$

- **Distortion**:

$d = M(\tau, t) * cost + D(\text{Pos\_}\tau, \text{Pos\_t}) = 0.1 * 3.017 + 0.3 = 0.6017$

Now, we have to store the following data:

d = 0.6017; Pos_ τ = 48; Pos_t = 199; step_ τ = 2; step_t = 1

Then, we have to compute the other parameters and update them in each matrix. The parameters are:

- **Tempo, average tempo and standard deviation**:

If the new cell implies a new state and the new state is also an anchor point (I am going to suppose that).

n = v_numPA(Pos_ τ, Pos_t) + 1

$$tempo = \frac{\tau - v\_tau(Pos\_\tau, Pos\_t)}{t - v\_t(Pos\_\tau, Pos\_t)} = \frac{50 - 48}{200 - 199} = 2$$

$$mean = v\_mean(Pos\_\tau, Pos\_t) + \frac{tempo - v\_mean(Pos\_\tau, Pos\_t)}{n}$$

$$= 2.1 + \frac{2 - 2.1}{6} = 2.0833$$

$$Std$$

$$= \sqrt{\frac{(v\_Stds(Pos\_\tau, Pos\_t)^2 * (n - 1) + (tempo - v\_mean(Pos\_\tau, Pos\_t)) * (tempo - mean)}{n}}$$

$$= \sqrt{\frac{(0.3^2 * (6 - 1) + (2 - 2.1) * (2 - 2.0833)}{6}} = 0.27$$

Finally, we store the information in each matrix:

D(τ,t) = D(50,200) = d = 0.6017	v_t(τ,t) = v_t(50,200) = t = 199
v_step_ τ(τ,t) = v_step_ τ(50,200) = step_ τ = 2	v_n(τ,t) = v_n(50,200) = n = 6
v_step_t(τ,t) = v_step_t(50,200) = step_t = 1	v_mean(τ,t) = v_mean(50,200) = mean = 2.0833
v_ τ(τ,t) = v_ τ(50,200) = τ = 48	v_Stds(τ,t) = v_Stds(50,200) = Std = 0.27

In the following figures we can see this example graphically. In each of them, the red cell represents the one that is analyzing the system and the blue cell with the yellow border represents the one that the system has chosen with the minimum cost.

v_step_τ:

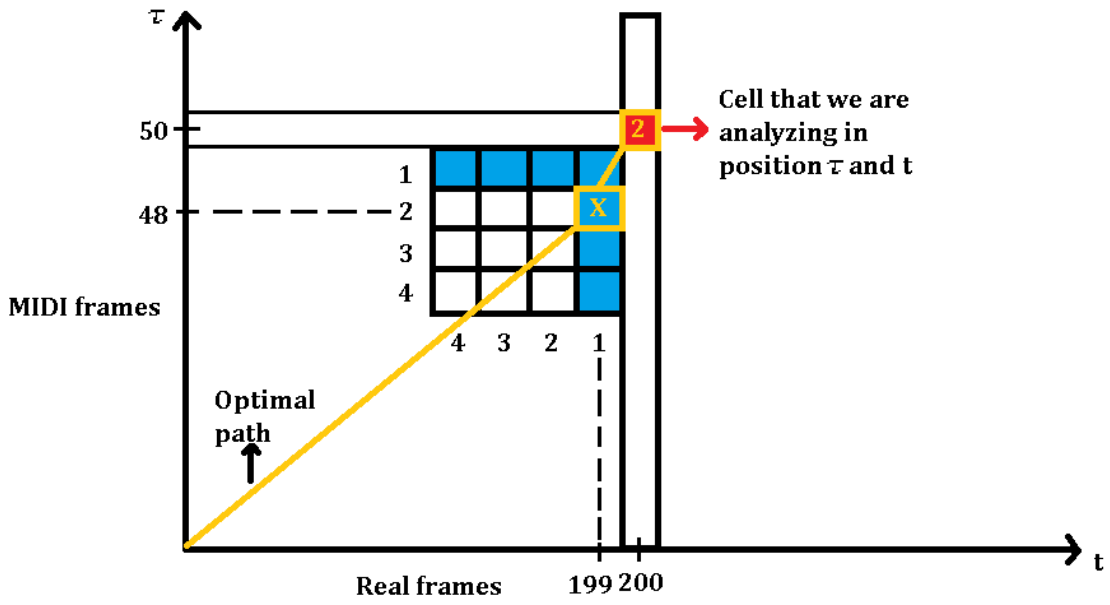


Figure 4.24 – MIDI time step matrix of the example proposed.

v_step_t:

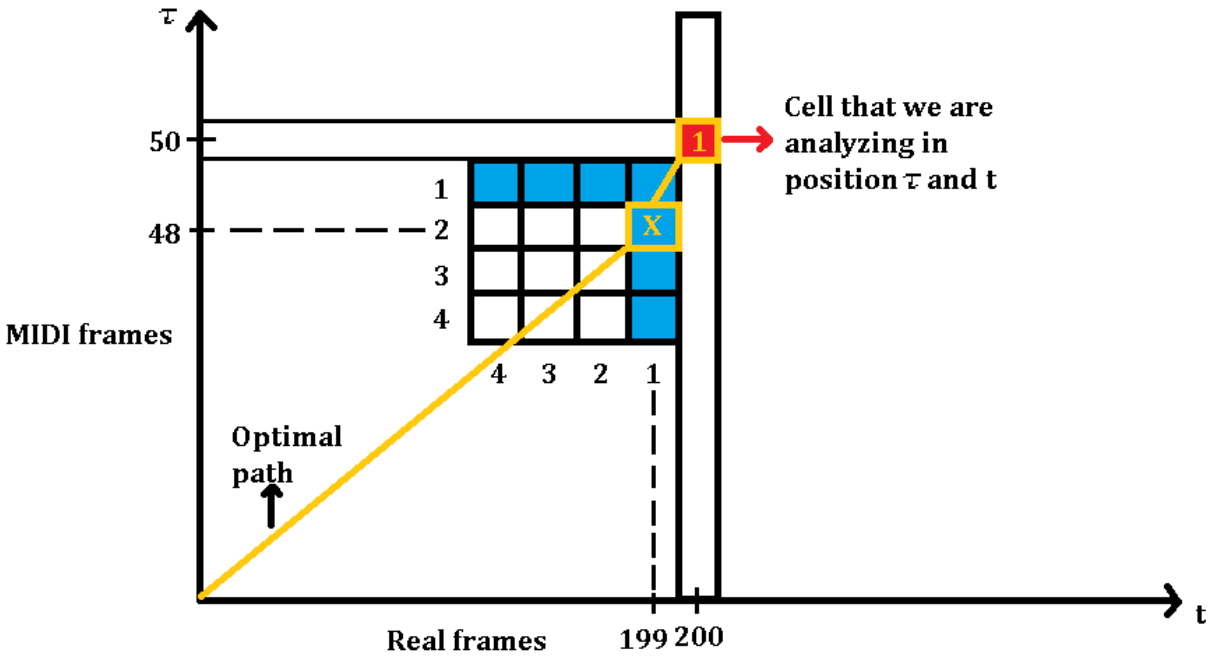


Figure 4.25 – Real time step matrix of the example proposed.

v_ τ:

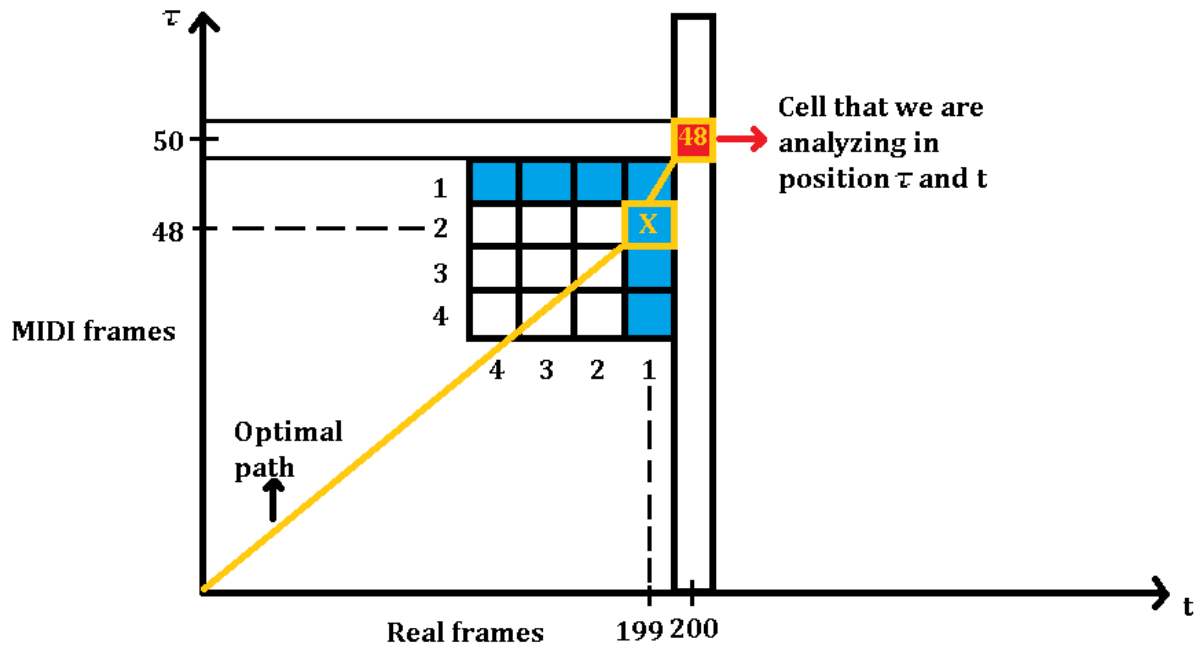


Figure 4.26 – MIDI time count matrix of the example proposed.

v_t:

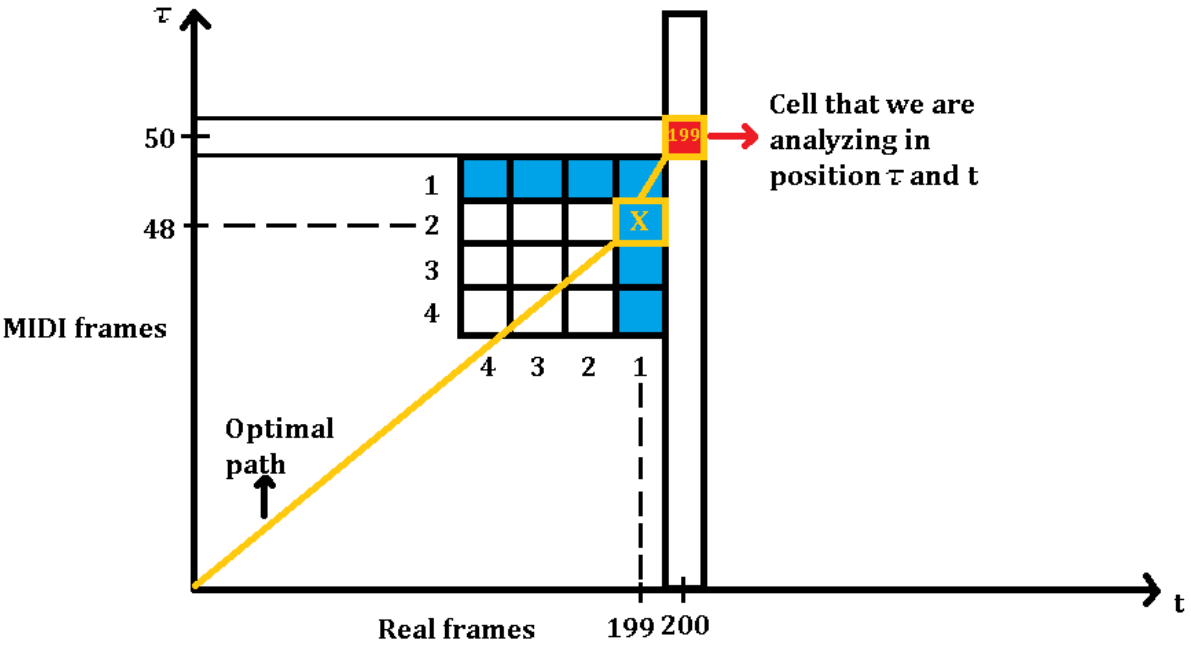


Figure 4.27 – Real time count matrix of the example proposed.

v_numPA:

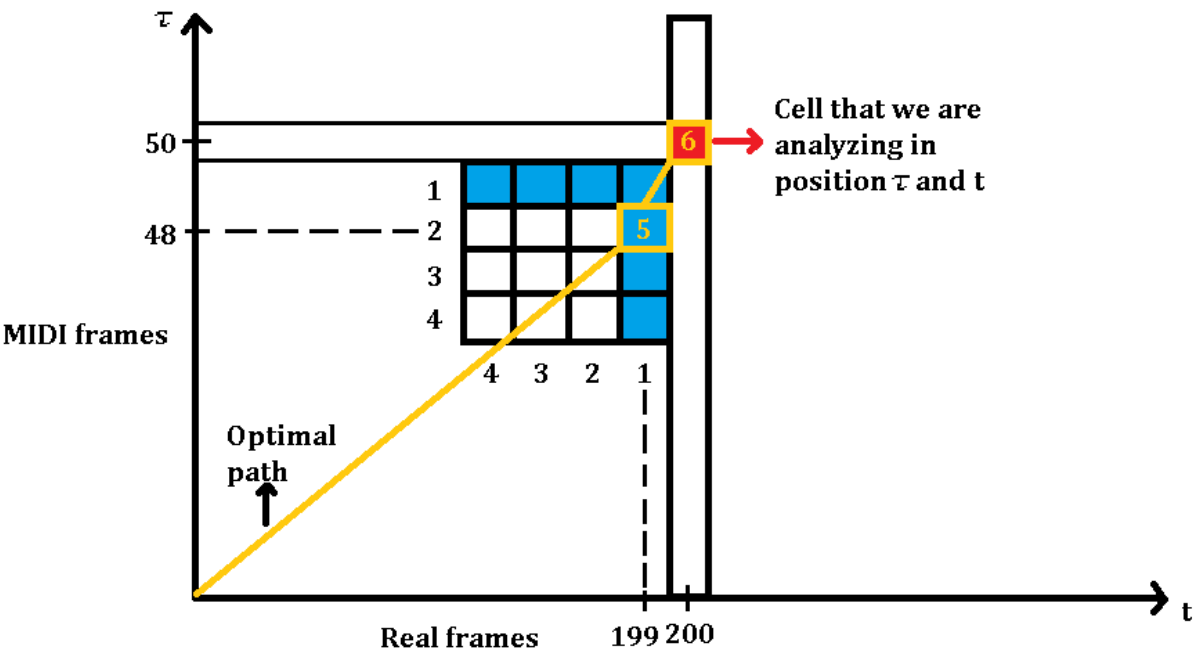


Figure 4.28 – Anchor point matrix of the example proposed.

v_mean:

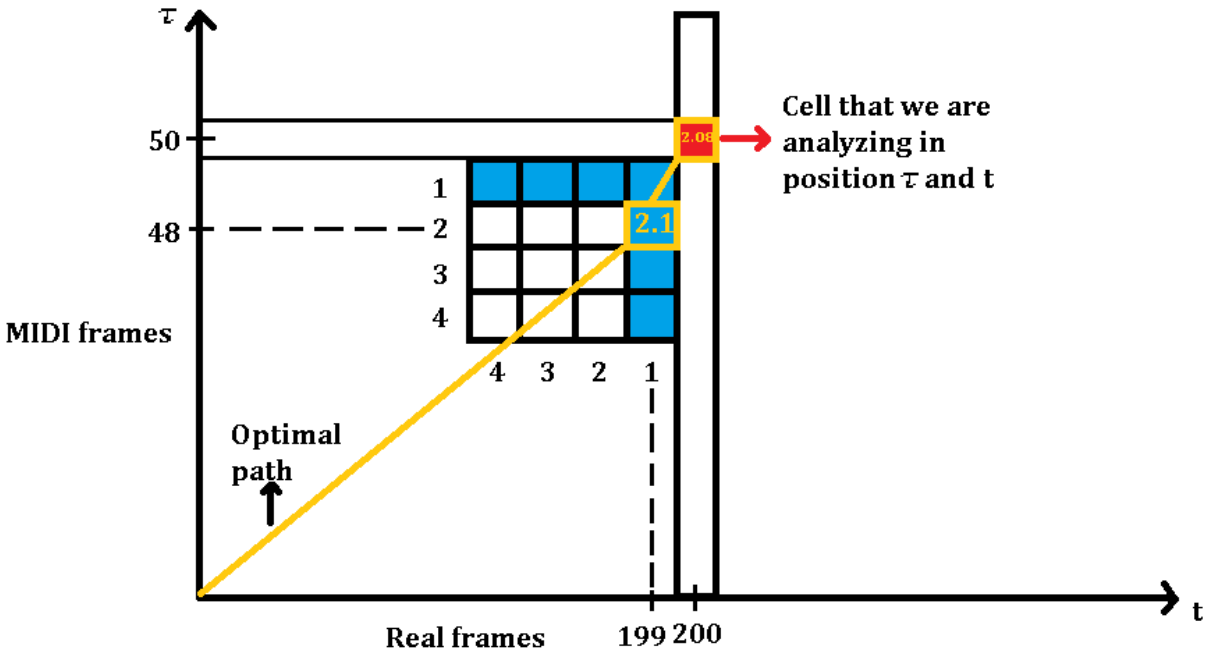


Figure 4.29 – Average tempo matrix of the example proposed.

v_Stds:

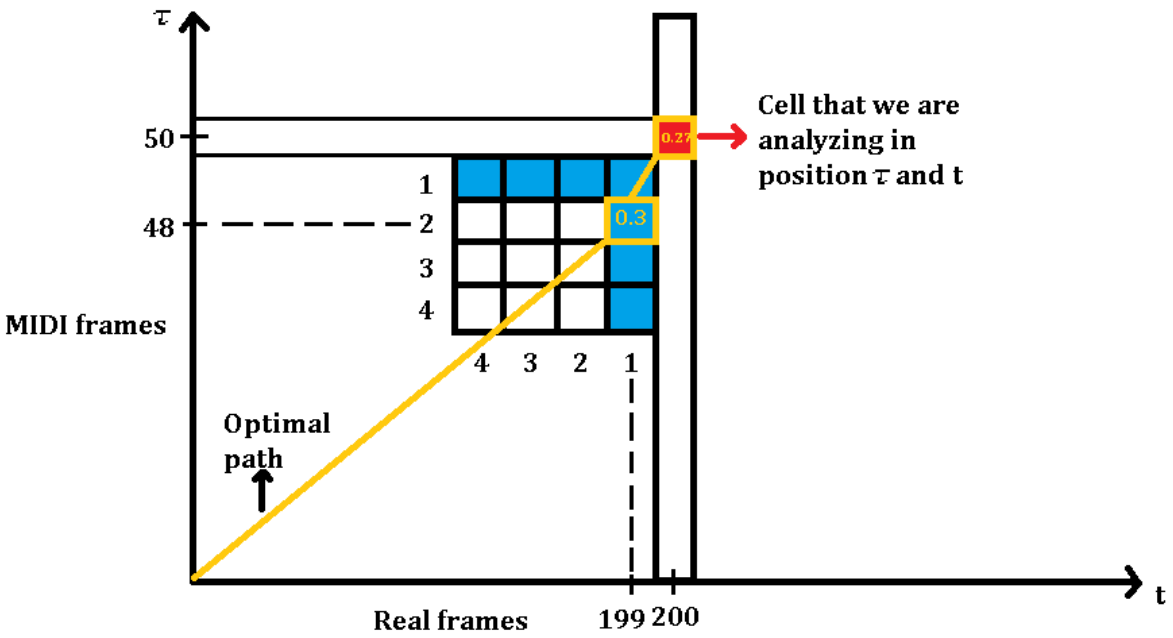


Figure 4.30 – Standard deviation tempo matrix of the example proposed.

DTW online/offline

This algorithm has been developed in two approaches: **online** and **offline**, depending on whether the piece of music is being processed at the same time as it is being performed, or is being processed after the performance audio has been collected. The two approaches are based on the general idea of *DTW* regulated by a constraint constant that I have explained before, but there is a fundamental difference.

From a descriptive point of view, the differences between the two approaches are:

- **Online** *DTW* method allows the data to be processed in real time, that is, the alignment from audio to score is performed while the musician is performing. In addition, it cannot obtain information about the future of the audio, since the system cannot know what the musician is going to play if he has not yet played it. Then, this approach computes all the necessary matrices and parameters as it processes audio from the musician's performance that reaches the system.

- In **offline** *DTW* technique, all the performance is accessible through the alignment process, that is, it allows "looking at the future" while establishing the correspondence, they are in charge of aligning a score with previously recorded audio, using the necessary process time. Then, this approach computes all the

matrices and parameters necessary to later be able to reconstruct the optimal path from the end of the performance to the beginning because we have all the real-time audio to be processed.

From a mathematical point of view, the two approaches have a clear distinction:

- In the **online** approach, the system makes the decision in real time about what is the minimum cost in each real frame t, that is, it calculates the optimal MIDI frame in each real frame t. We can define it mathematically as ***pos_min = min (D (τ, t))***, where pos_min is a vector that contains the optimal MIDI frame (the optimal path) at each frame t, and τ and t are the MIDI and real frames.

- In the **offline** approach, the system computes the total distortion matrix until the end and in the final real frame determines the optimal MIDI frame. Afterwards, a traceback is made going back through the matrix and taking into account the optimal jumps that the system has been computing. We can define it mathematically as ***pos_min = pos_min(t + step_t) - step_*** τ, where pos_min is a vector that contains the optimal MIDI frame (the optimal path) at each frame t, t is the real frame, step_t is the step in real frame and step_ τ is the step in MIDI frame.

The main problem with the offline method is that in order to implement it you have to store all the past of the matrix and the necessary memory is triggered.

For this reason, we have implemented an offline DTW algorithm **by blocks** of a specific set of frames, so that the memory needed to process the signals does not increase considerably. As it is the offline method we have the complete performance audio to be processed, therefore we break the signal into blocks of N frames. The first block of N frames arrives at the system to process it and we apply a *DTW* forward. Then the second block arrives and we apply the *DTW* forward and then we apply a *DTW* backward to these two blocks and so on until we process all the frames.

We are going to see a small **example** to explain it better and we can visualize it in *Figure 4.31*. We have an audio composed of 3072 frames and we break it down into blocks of 1024 frames. The first block of 1024 frames reaches the system and we apply a *DTW* forward. Then, the second block of 1024 frames arrives at the system and we apply the *DTW* forward again and we apply a *DTW* backward to the two blocks to improve the alignment. Now, a new frame block arrives and we apply a *DTW* forward to that block and a *DTW* backward to the set of the second and third blocks. Finally, we have estimated the

audio-score alignment using *DTW* forward and backward techniques and processing the signal in frame blocks [25].

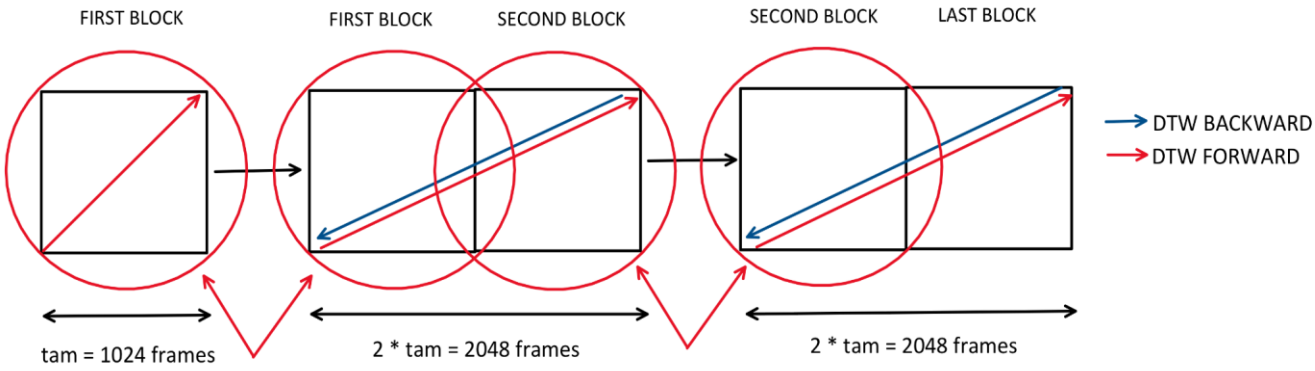


Figure 4.31 – Example of DTW technique by blocks.

Processing the audio signal using blocks is very useful because it saves resources of the device that is processing the signal and is more efficient. After having explained the alignment method we can explain the next stage.

4.1.5 Musical statistics stage

In this section I am going to show and explain the block of extraction of musical statistics from the optimal path that I have developed in this work. In this application, we extract musical statistics such as beat times, tempos and note duration from the score and audio of the musician's performance. These statistics are represented in plots for informing the musician about his/her progress.

Inputs:

- Optimal path: *pos_min*

Outputs:

- Musical statistics: *beat time, tempo and duration*

Musical statistics stage summary
- Computation of musical statistics based on the optimal path.

Explanation of the algorithm

As we have reviewed in the *section 2.2*, beat time, tempo and note duration are very important features for a musician because it indicates how a certain musical composition has played.

At the output of the alignment stage we obtain the optimal path, therefore we use it to compute the beat time. The **procedure** is as follows:

- We convert the optimal path from frames to states because the score information is given by states and we estimate the onset instant of time that corresponds to each state.

- Then, we create an alignment table where we show the start time of each measure (in seconds), the start and end of each note, the MIDI note and the beat number. To have measure data, it must be well defined in the MIDI file.

- We compute the MIDI beats, interpolate any missing beat value and finally estimate the real start of each beat in seconds.

Taking this procedure into account, we start from having the optimal computed path (pos_min), that is, we know the MIDI frame that corresponds to each real frame. Then we transform the frames into states, that is, the system searches for the pos_min values that correspond to each score state. Next, knowing the real frame of each state, the system computes where each pos_min begins.

Now, knowing where each frame of pos_min begins and using information from the MIDI file, we build two tables that are very important to the process of obtaining the start time of each beat. The first table, called nmat, contains information extracted directly from the MIDI file, as we can see in *Table 4.5*. This table gives us information about the measure of each MIDI note, the duration of each beat, the channel or instrument, the pitch or number of the note in MIDI scale, the playing velocity, the onset time of each beat and the duration.

Measure	**Beat Duration**	**Channel**	**Pitch**	**Velocity**	**Onset Time (s)**	**Duration (s)**

Table 4.5 – Table with MIDI information.

The second table, named table, contains the alignment information (see *Table 4.6*). This table gives us information about the estimation of the start of each note, the position of the sequence or measure, the onset and offset frame, the start of the MIDI note in seconds, the MIDI note and the channel or instrument.

Estimation note start	Sequence position	Onset Frame	Offset Frame	MIDI start (s)	MIDI note	Channel

Table 4.6 – Table with alignment information.

Once we have the above information, we build a new alignment table with backtrack formatting (see Table 4.7). This table gives us information about onset and offset time in milliseconds, start time of the MIDI note, channel or instrument and measure.

Onset Time (ms)	Offset Time (ms)	MIDI start	Channel	Measure

Table 4.7 – Table with new alignment information.

Then, with the information in the previous table, we estimate the start time of each measure in seconds. Next, we interpolate the missing values to complete all the beats. Finally, we build the output matrix that is composed by the information that appears in *Table 4.8*. To do this, we store the beats in the same position that corresponds to them and we interpolate the intermediate values. Also, there is the possibility that there are MIDI notes that occur at the same time, therefore we also interpolate the repeating values.

Beat Number	MIDI start	Beat Time (s)

Table 4.8 – Table with beat information.

When we have managed to estimate the start of each beat in seconds, we can compute the duration of each beat by applying the time difference between beats and we obtain the duration of each beat in seconds. This is because not all beats have note beginnings, that is, not all notes are given in beat beginnings. When a musician performs a certain piece of music, he does not do it in an ideal way like a synthesizer, it could be that there are notes that do not start at the same time that the beat marks, that is, they can be played a little earlier or a little later.

Then, having estimated the duration of each beat in seconds, we can compute the tempo in beats per minute applying the following relationship:

$$tempo(bpm) = \frac{60}{beat\ duration} \tag{25}$$

At this point we have already estimated the beat time and duration in seconds and the tempo in bpm, but when we visualize the data we are going to do it in each measure of the score. Therefore, from the score information we extract the type of measure of the

musical piece and with it we know how many beats make up each measure. Knowing the number of beats per measure and the total number of beats of the score we are able to estimate the beat time, tempo and duration in each measure.

I have to say that in order to estimate the time signature of the input musical piece I have used the "**get_time_signatures**" function from "**midi_lib**" Toolbox of MatLab. From the MIDI file it returns the numerator and denominator of the time signature of the score and thus we know the reference musical figure and the number of beats that make up each measure [35].

The function that we have used to know the time signature computes the number of ticks / quarter note and then converts it to beats. In this way, the MIDI file provides us with information about the numerator and denominator of the time signature, which is very useful to determine the musical statistics in each measure, since we know that not all measures are made up of the same number of beats in different scores.

In the **example** in *Figure 4.32*, I have a 4/4 measure, I know that there are 4 beats in each measure. If in the musical piece we have 20 beats, then we have 5 measures, therefore, I get the statistical value for each of those 5 measures.

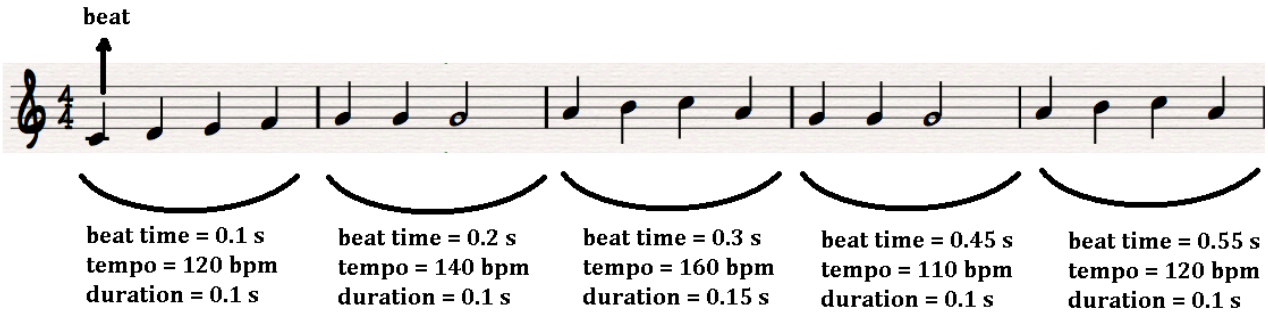


Figure 4.32 – Example about information of musical statistics

When we have already estimated the musical statistics we can explain the last stage of our system, the stage in which the performance of the user can be compared with another musician.

4.1.6 Comparison stage

Having explained the previous stages (**training**, **alignment** and **extraction of statistics**) and knowing that we have all the necessary data to be able to compare two interpretations, I am going to show this stage.

The **main objective** of this stage is to evaluate the comparison between two musical performances: the first is the entry and from which we extract certain statistics and the other is the one that we have in our database with all the necessary information previously loaded.

Inputs:

- Musical statistics of the input performance: *beat time, tempo and duration*

Outputs:

- Plots: linear(*beat time, tempo and duration*), *triangle-shaped and tree-shaped*

Comparison stage summary
- Extract musical statistics from the input performance and the musician with whom we want to compare. - Generate plots that compare the data from the two performances.

4.1.6.1 CHARM Mazurka Project

Before beginning to describe the behavior of this stage, we have to explain certain concepts that are very useful for later explanations. Specifically, I'm going to explain what our triangle and tree plots are based on.

These types of plots are based on the CHARM Mazurka Project and are part of a project created by the **Center for the History and Analysis of Recorded Music** (*CHARM*) about Chopin's Mazurkas [35]. The purpose of this project, driven by **Craig Sapp** (project main researcher) and **Nicholas Cook** (project manager), was to investigate the potential of computational approaches for characterization of performances. Chopin's Mazurkas were chosen as a varied repertory that poses interesting performance practice issues and had more than 3000 performances, although only a proportion was analyzed within the project.

Most of the work was based on the extraction of tempo and dynamic data, which were analyzed mathematically, to investigate musical features. For this reason, we found it interesting to use the plots of this project to show our data since, in fact, what this project did was to compare data (tempos) between the different performances.

Mazurka discography contains performances for different Mazurkas: *Op. 6, Op. 7, Op. 17, Op. 24, Op. 30, Op. 33, Op. 41, Op. 50, Op. 56, Op. 59, Op. 63, Op. 67 and Op. 68*. Some of these Mazurkas can also be viewed as a spreadsheet which contains manually annotated data for each performance and for different musicians such as:

- **absbeat**: numbering of the beats of the score.
- **measure**: measure number of each beat.
- **beat**: numbering of the beats within each measure.
- **time**: time (in seconds) of each beat.
- **duration**: duration (in seconds) of each beat.
- **tempo**: tempo (bpm) of each beat.
- **temptime**: the same as time.

absbeat	measure	beat	time	duration	tempo	temptime
0	1	1	0,621	0,327	183,572	0,62113378
1	1	2	0,948	0,310	193,548	0,94798185
2	1	3	1,258	0,310	193,548	1,25798185
3	2	1	1,568	0,280	214,286	1,56798185
4	2	2	1,848	0,300	200,000	1,84798185
5	2	3	2,148	0,310	193,548	2,14798185
6	3	1	2,458	0,310	193,548	2,45798185
7	3	2	2,768	0,320	187,500	2,76798185
8	3	3	3,088	0,380	157,895	3,08798185
9	4	1	3,468	0,340	176,471	3,46798185
10	4	2	3,808	0,440	136,364	3,80798185
11	4	3	4,248	0,630	95,238	4,24798185
12	5	1	4,878	0,470	127,660	4,87798185
13	5	2	5,348	0,229	261,825	5,34798185
14	5	3	5,577	0,311	193,026	5,57714285

Figure 4.33 – Spreadsheet example with annotated data of musical piece Mazurka 24-2 played by Biret.

4.1.6.2 Graphic representations used

And now, after having introduced a little on which this project is based on, we are going to explain the different plots.

First of all, we have to talk about the correlation concept. Correlation is a technique that measures the similarity between two sequences of numbers [34]. In a basic way,

correlation is the sum of the multiplication between the numbers that correspond to the pair of sequences as shown in the following expression:

$$\sum_i x_i y_i \tag{26}$$

where **x** and **y** are the two different sequences and the subscript **i** represents each value of the sequences.

It is important to normalize the result of the correlation so that they are within the **range from -1.0 to +1.0** using the expression corresponding to **Pearson's product moment correlation**:

$$\frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{27}$$

where **x** and **y** are the two different sequences, the subscript **i** represents each value of the sequences and $\bar{x}$ and $\bar{y}$ are the mean of each sequence.

In *Figure 4.34* we can see two examples taken from the Mazurka project website to explain how the correlation concept works. In the first case, the resulting correlation is 1 because the pair of values is the same, while in the second case the result is -1 since the values are completely opposite.

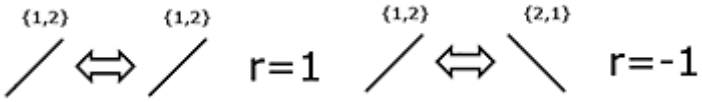


Figure 4.34 – Correlation example (I). Source [35]

We must emphasize that the Pearson correlation measures the gross contour of the pair of sequences and, although, the slope of the line varies, the correlation value remains as we see in the following examples:

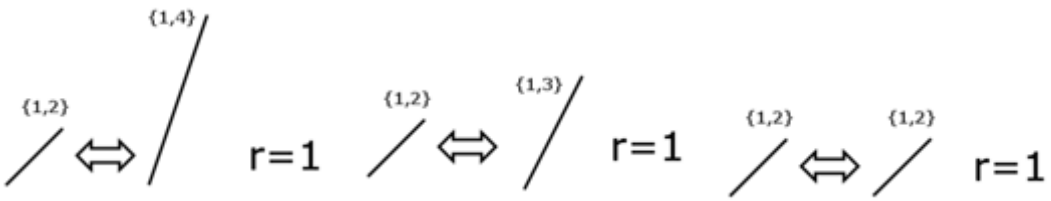


Figure 4.35 – Correlation example (II). Source [35]

The following figure demonstrates the correlation values for several sequences of length 3.

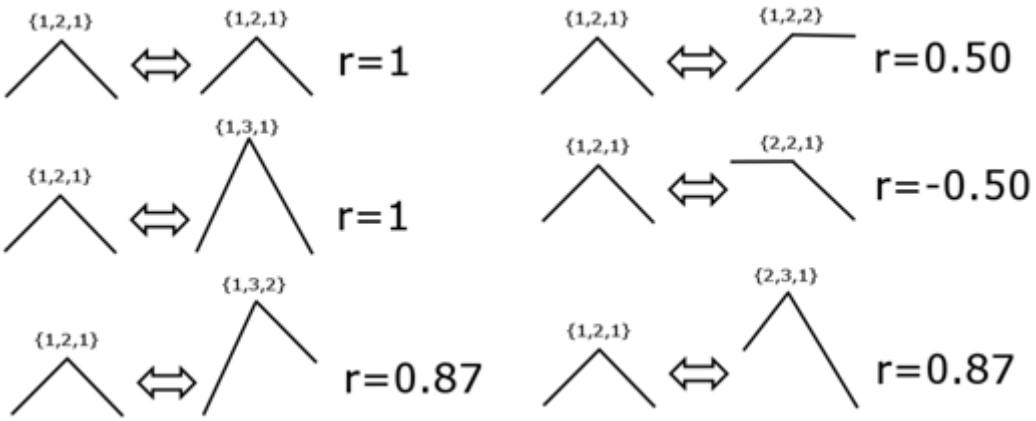


Figure 4.36 – Correlation example (III). Source [35]

Now we begin to explain the different plots that we have implemented in our application.

Hierarchical Correlation Plots

The main problem with the correlation is that it can only describe the general similarity of the two sequences when both are considered in their entirety. In order to make the correlation in its entirety, we have to chop up the two sequences into smaller pieces. For **example**, if both sequences have six numbers {A, B, C, D, E, F}, then {A, B, C}, {D}, {C, D, E, F} and {A, B, C, D, E} are subsequences which can be correlated with the entire sequence.

The best way to examine the internal similarity of two sequences is to divide them into all possible subsequences. The following Figure shows the 2-D plot that can display all those sub-sequence correlations simultaneously.

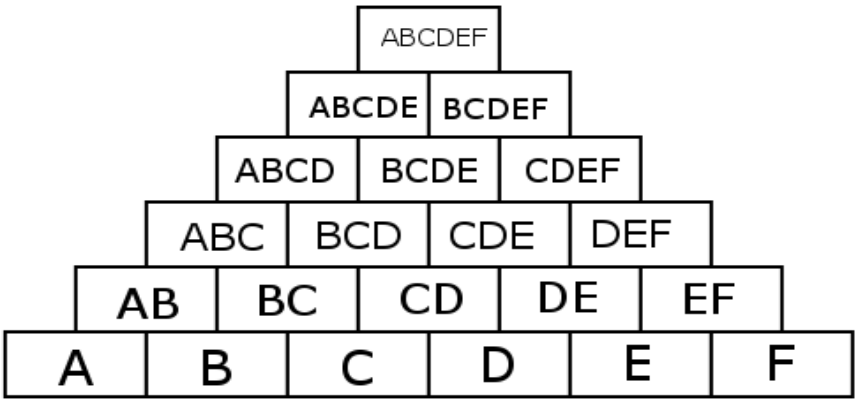


Figure 4.37 – Diagram about how the signal is decomposed to represent it as a triangle. Source [35]

If we transfer this concept to a musical signal, we have two sequences with the tempo of each performance to compare, then the upper part of the triangle corresponds to the complete signal and as we get closer to the base of the triangle the algorithm estimates the correlation between smaller parts of the signal. In this way, the peak of the triangle corresponds to the correlation of the entire signal and the base of the triangle represents the correlation between the smallest possible pieces of the performance.

During the rest of this subsection, we explain the different types of triangles and trees and we show an **example** of each of them corresponding to a signal extracted from the *2nd movement of Mazurka Op. 24* from the **Mazurka Project database** whose performer has been **Biret**.

Temposcape plots

This plot shows a comparison of the beat-by-beat performance tempos.

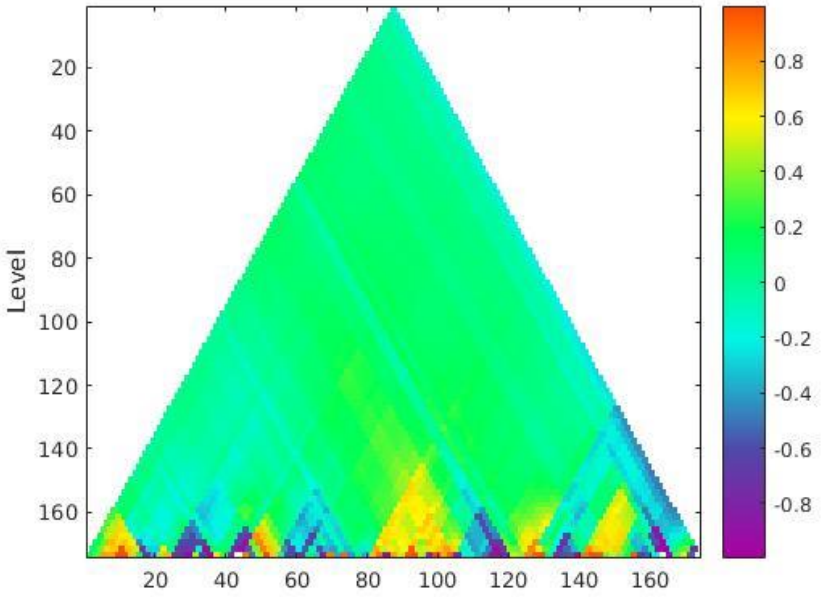


Figure 4.38 – Example of Temposcape plot. The resulting correlation is very close to 0 throughout the entire musical piece.

Polycorrelation plots

This plot allows us to extend the method of plotting the correlation between two performances to comparisons between multiple performances. So this type of plot shows the similarity to one or another interpreter in each area of the performance.

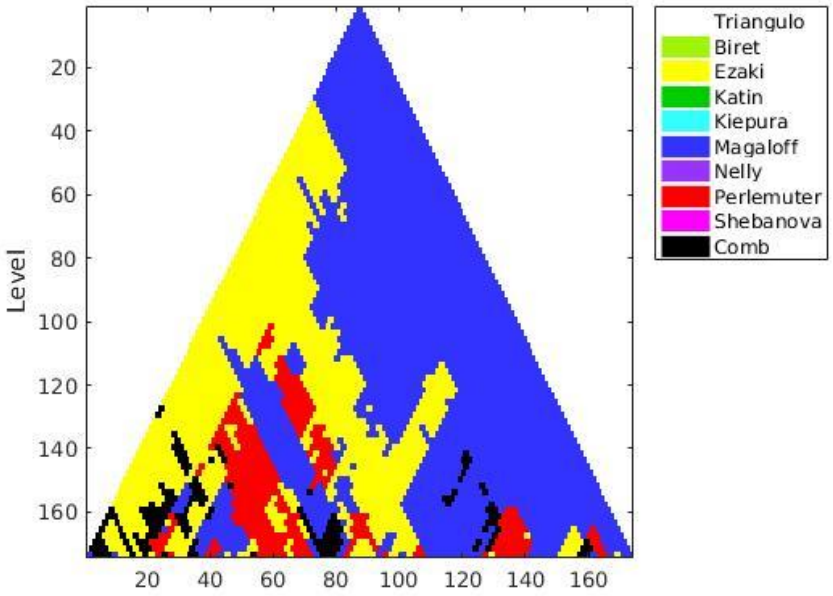


Figure 4.39 – Example of Polycorrelation plot. This plot shows us that this Biret performance looks like Magaloff and Ezaki in most of the time.

Hierarchical Average Plots

This plot is similar to the previous one, but in this case, it shows the average of the values of a single sequence.

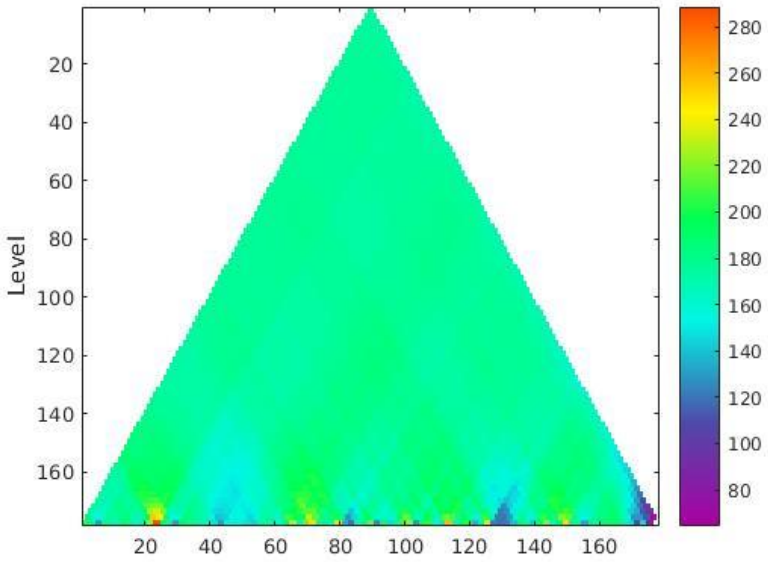


Figure 4.40 – Example of Hierarchical Average plot. We can see in this plot that the performer had a constant tempo of around 180 bpm.

Average Difference Plots

These types of plots show how fast or slow the performance of a musical composition is, that is, how fast or slow a sequence of tempo values is relative to each other. If the first performance is faster than the second performance in a given time of the piece, the plot is colored red. And, if the first performance is slower than the second performance in a given time of the piece, the plot is colored blue.

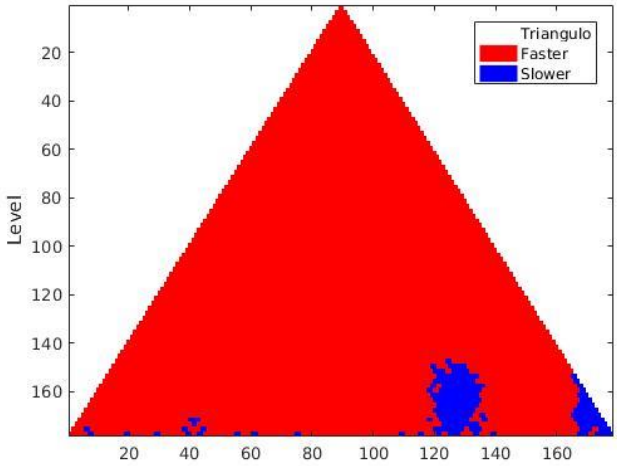


Figure 4.41 – Example of Average Difference plot. This plot informs us that Biret has been faster than another musician with whom he has been compared.

Contoured Average Difference Plots

The next type of plot compares the differences in tempo in a more refined way, that is, it gives us a better quantitative estimate of the differences in tempo between the two performances, coloring the plot according to a percent change.

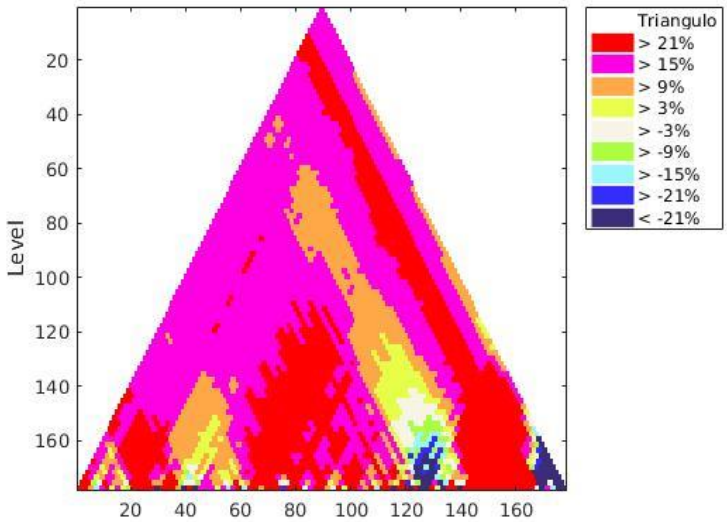


Figure 4.42 – Example of Contoured Average Difference plot. This plot informs us about the correlation in %.

Tree plots

In tree-shaped plots there are two approaches: the first, which shows the correlation of a musician with respect to all the others in the database, and the second, which shows the correlation of all the musicians to each other. The philosophy of this type of plot is the same as in the case of triangles, that is, to calculate the similarity between two sequences of numbers. The only difference is in the way of representing these correlations.

In *Figure 4.41* we can observe the tempo similarity of 1 performer against all in the database. For example, For example, the tempo similarity between Biret and Ezaki is more than 0.7 out of 1, so both performances are very similar. On the other hand, the tempo similarity between Biret and Katin is 0.4 out of 1, so these two performances are more or less alike.

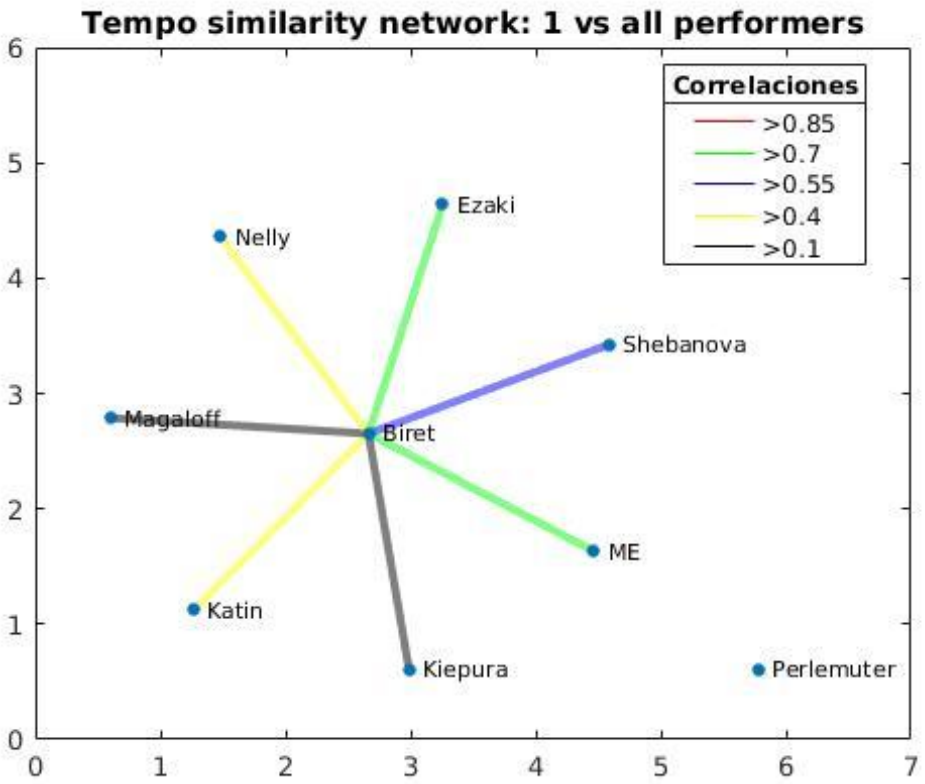


Figure 4.43 – Example of tree plot: 1 vs. all performers.

In *Figure 4.43* we can see the correlation between all the performers of a specific piece of music. For example, the tempo similarity between Biret and ME (corresponds to the user musician) is more than 0.75 out of 1, so both performances are very similar. On the other hand, the tempo similarity between Katin and Nelly is 0.1 out of 1, so these two performances are very different.

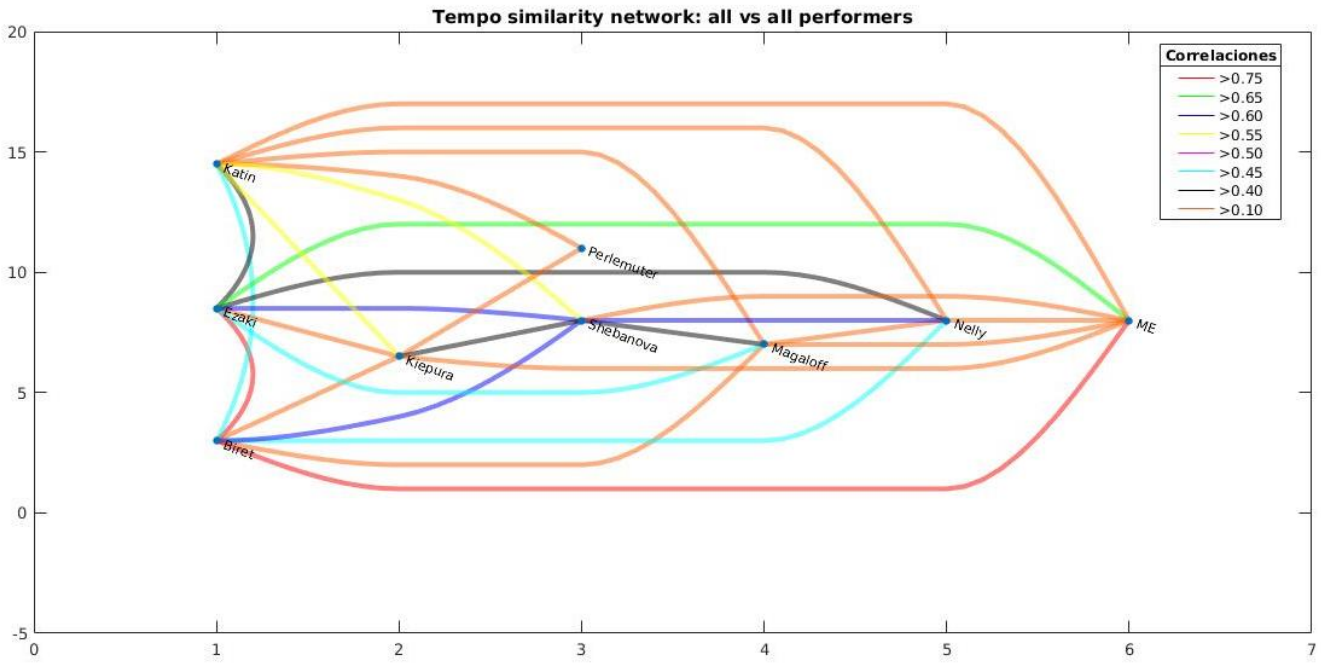


Figure 4.44 – Example of tree plot: all vs. all performers.

And, up to here comes the description of all the types of plots that have been used in this work. Now, I can explain the implemented comparison algorithm.

4.1.6.3 Algorithm used and sampling factor

After having described the Mazurka Project and its concepts, on which our algorithm is based, I can explain how this stage works.

Once the system has computed the musical statistics of the musician's performance, it extracts these calculated data in each measure. Then, the system obtains the statistics of the musician's performance to be compared and which have been computed in advance. That is, in our comparison database we have the necessary information stored so that the user can choose the musician to be compared to. To compare both interpretations, the system estimates these last comparative data in each measure.

Next, we can estimate the triangle plots that we have previously described, computing the correlation between the pair of input sequences.

And finally, we compute the similarities between sequences and estimate the tree branches to show the comparison between performances, but at this time, in a tree shape.

At this time, we already have the necessary information to show the comparison plots in the interface of our application.

It is important to indicate that in order to generate the triangle and tree plots we have had to apply a sampling factor to the comparative sequences. The main reason we had to do this was because the correlation error and tempo data were too high. In *Tables 4.1* and 4*.2*, we can observe the correlation error and the tempo data for three different Mazurkas without applying the sampling factor.

	Mazurka 24-2	**Mazurka 30-2**	**Mazurka 68-3**
Tempo error (bpm)	85.9	214.9	200

Table 4.1 – Tempo error in bpm for three different Mazurkas without applying the sampling factor.

	Mazurka 24-2	**Mazurka 30-2**	**Mazurka 68-3**
Correlation error	0.31	0.3	0.22

Table 4.2 – Correlation error for three different Mazurkas without applying the sampling factor.

Due to the high error values, we developed a sampler to apply to the comparison sequences. As we can see in *Tables 4.3* and *4.4*, the errors decrease considerably applying a sampling factor 4.

	Mazurka 24-2	**Mazurka 30-2**	**Mazurka 68-3**
Tempo error (bpm)	15.4	55.2	23.6

Table 4.3 – Tempo error in bpm for three different Mazurkas applying a sampling factor 4.

	Mazurka 24-2	**Mazurka 30-2**	**Mazurka 68-3**
Correlation error	0.09	0.25	0.14

Table 4.4 – Correlation error for three different Mazurkas applying a sampling factor 4.

It should be noted that the data that I show in the previous tables are mean values. The sampling factor that we apply to the sequences is **4** because it is the value with which the correlation error is stable at a minimum error value. As we have shown, it is important to use the sampler to minimize possible system errors.

The procedure that the sampler follows is simple because it was based on recalculating the tempos every X beats (in our case every 4 beats). This is done because the aligner accuracy is not very good at the beat level and therefore we obtain data in each measure. To achieve this, the algorithm performs the following steps:

- As input the algorithm has the vector with the beat times and the sampling factor, which in our case is always 4.
- Then, the system samples the input vector every 4 beats.
- Later, the algorithm estimates the difference between each value of the sampled vector, that is, it computes the duration in seconds every 4 beats (duration vector).
- Finally, it converts the sampled beat times to tempo data (bpm) according to the following definition:

$$tempo\ (bpm) = \frac{60\ \cdot sampling\ factor}{duration\ vector} \tag{28}$$

To help understand this concept, we can see the following example:

- We have the beat times vector in seconds [1.65 2.32 2.93 3.48 4.01 4.56 4.85 5.38 5.92 6.57 7.26 8.05] and the sampling factor is 4
- We sample the vector with the beat times every four values: [1.65 4.01 5.92]
- We compute the duration: [2.36 1.91]
- And now, we compute the tempo (in bpm) from the duration: [101.6949 125.6545]

4.2 Application

4.2.1 Introduction to the application

The last part of the work consisted of developing an application in which all the developed blocks are integrated. The user will be able to visualize the **musical statistics plots** in each measure of his performance and, in addition, he will be able to **compare** these results with another famous or professional performer that we have in our database. It is an application, whose interface has been developed using the development tools of a **Graphical User Interface** (*GUI*) in MatLab. The development environment that we have used is typical of MatLab and is called **GUIDE** [36]. Therefore, it is an application that can be executed on a computer and that contains parts developed in **C** and other parts developed in MatLab. In this chapter, my work has consisted of developing the graphical user interface and integrating all the algorithms developed and explained in the previous section.

4.2.2 Tools used

Hardware:

The development of the application has been carried out on a computer with an Intel Core i5 Quad-Core processor at 3.3 GHz, 6 MB Intel Smart Cache, 7.7 GB of integrated LPDDR3 memory and Ubuntu version 18.04.3 LTS.

The application tests have been carried out on a computer with an Intel Core i5 Dual-Core processor at 1.8 GHz, 3 MB Intel Smart Cache, 3.7 GB of integrated LPDDR3 memory and Ubuntu version 18.04.3 LTS.

Software:

MatLab R2016b has been used to test implementation of the application algorithms, which were later implemented in the C programming language.

Sublime Text 3 is a text editor that has allowed us to program the alignment stages that have been developed in the C programming language.

MuseScore 2.0 has been used to check and adapt some MIDI files.

FluidSynth is the software that has been used in MatLab tests to synthesize the audio signal in WAV format from the score in MIDI format.

Programming languages:

MatLab: for programming the alignment, musical statistics extraction and comparison modules.

C: for programming the alignment algorithms.

4.2.3 Application behavior

As I have previously indicated, the application aims to obtain the musical statistics that have resulted from the performance of a musician and, in addition, it has the possibility of comparing itself with other performers. Moreover, as we know, our database is made up of **two datasets**: **URMP** and **Mazurka Project**.

First of all, we have to take into account that certain information must be stored in our database. This information is: training files in .mat format, training files in .dat format and files with the information to compare for each performer in .mat format. We have thought about using MatLab's own .mat format because it is the fastest way to load data in this environment. The .mat files are used in the algorithms developed in MatLab and contain all the necessary parameters to generate the data. However, the .dat training files are used in the alignment method developed in C to generate the optimal path.

Consequently, in order to generate the information that is stored in the database, we have developed an approach that includes training, alignment and extraction of musical statistics. This approach can be seen in the diagram in *Figure 4.43*. First, we insert the MIDI file of each musical piece into the training algorithm. This generates the necessary information and stores it in the database, in the training folder. Then, the training block sends the .dat file to the alignment algorithm, which estimate the optimal path. And finally, the musical statistics are extracted to store them in the database, in the comparison folder.

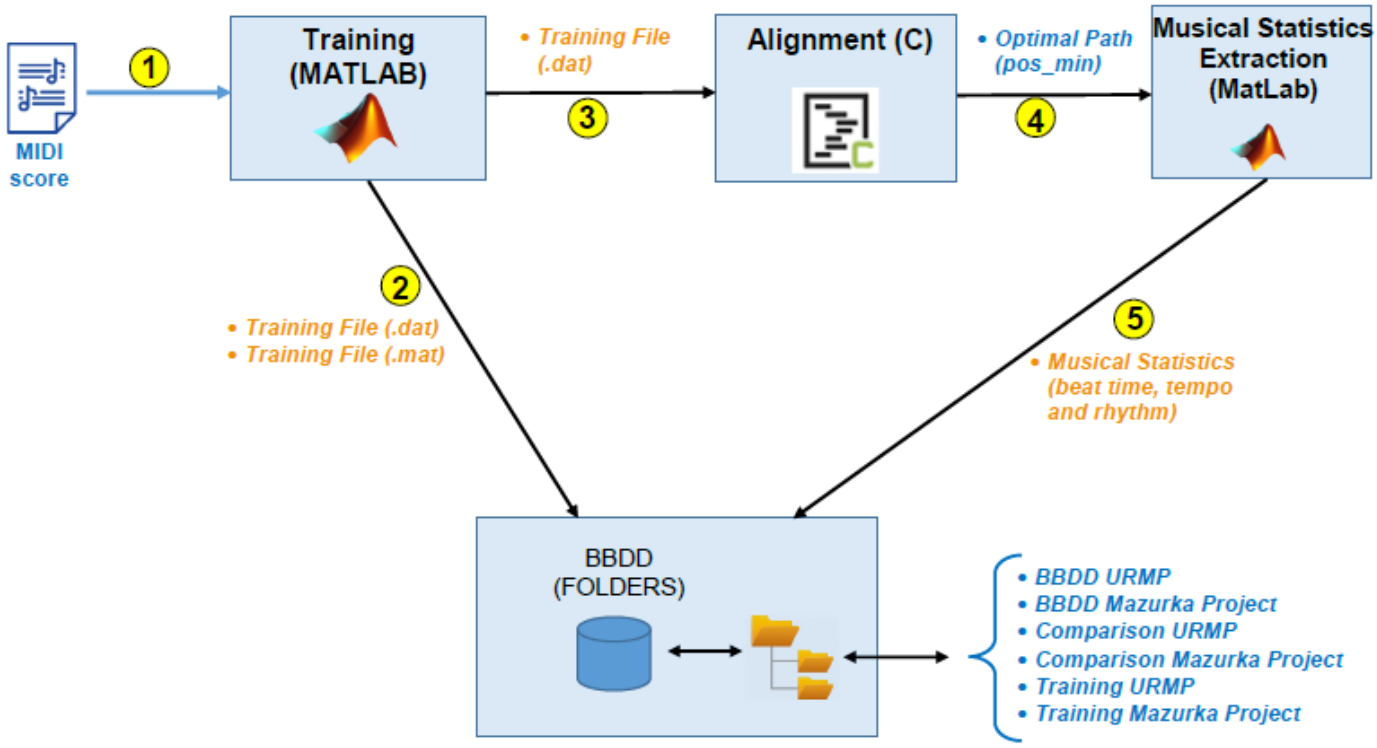


Figure 4.45 – Diagram with the relationship between the modules of pre-processing data.

According to this scheme, we can divide the preprocessing stage into 5 steps:

1) **MIDI score import**: we have to insert the MIDI file into the training stage and we extract the training files.

2) **Storage of training files in database**: we store the information into the database.

3) **Communication with the alignment process**: the training stage sends the training file to the alignment method in order to generate the optimal path that is passed to the next module.

4) and 5) **Musical statistics extraction stage**: this routine takes the optimal path and estimates the beat time, tempo, and rhythm of each measure and stores them into the database.

The user has to select the audio file of his performance and choose the score (training file) from the database that he has performed. Once the musician has chosen the audio and the score and has pressed the START button, the application start the alignment process with the file in .dat format associated with the score that contains the training files. Then, the alignment algorithm generates a text file with the optimal path that is passed to the musical statistics extraction method as input. This last method estimates

beat time, tempo and rhythm of each measure of the score and sends it to the interface to display it on a plot.

Then, as a separate block that can be used or not, the user indicate to the interface the file associated with the performer with which he wants to compare with. This information is extracted (musical statistics) that is available in our database and then the comparison algorithm will be put into operation. In this step, the method compares all the musical statistics and estimates the comparison triangles and trees by computing the correlation or similarity of the information of both performances. Finally, the system displays these comparison plots on the interface.

The following is the general scheme of the application in which it can be seen the relationships between the signal processing modules and the graphical user interface:

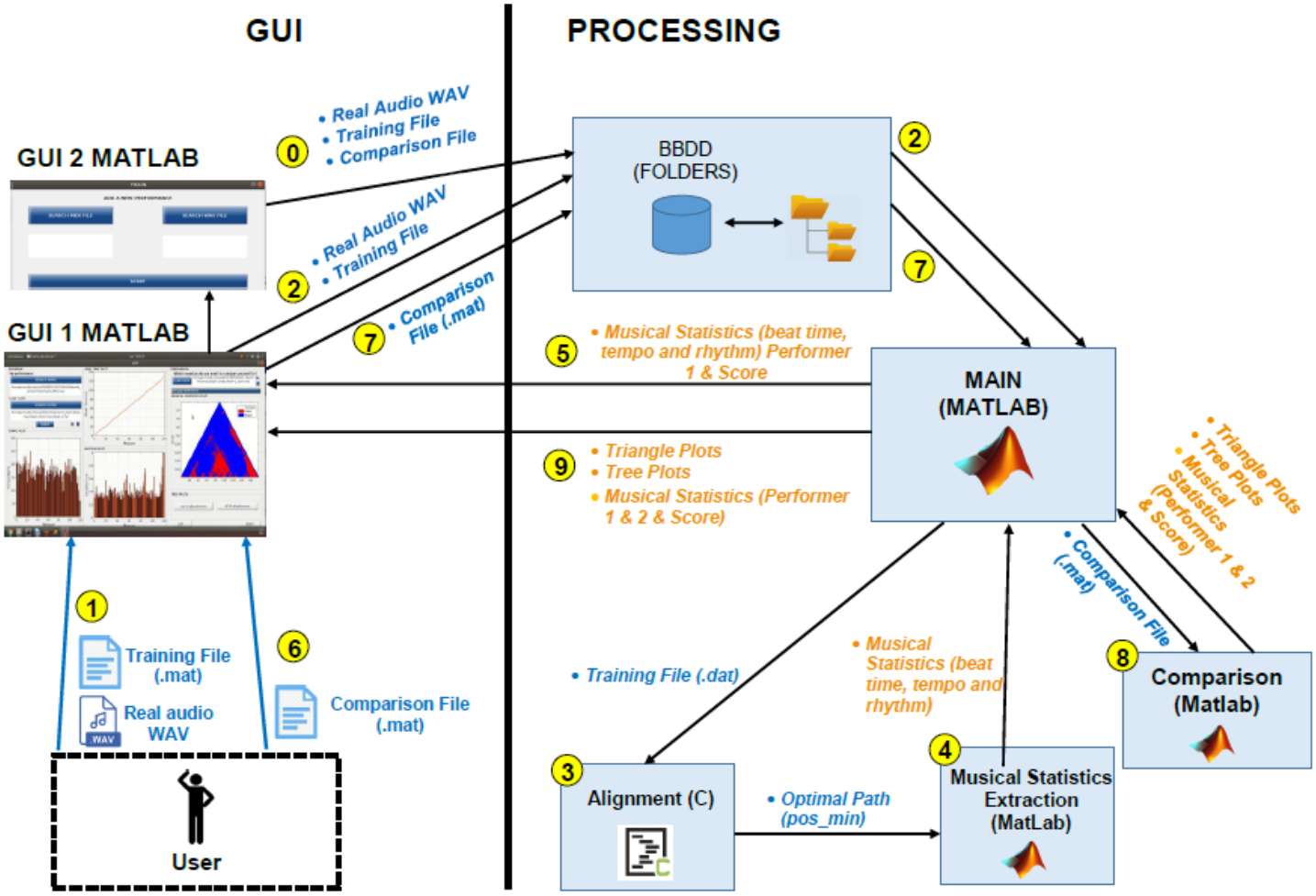


Figure 4.46- Diagram with the relationships between the processing stages and the graphical interface.

According to this scheme, we can divide the application execution process into 9 steps:

0) **Communication between GUI2 and database (Optional)**: if the user wants to add a new real audio recording to the database in order to compare with it, he has to select the MIDI file and the audio recording. Then, the system stores the wav, training and comparison files into the database.

1) **File import**: the user accesses the application interface through the MatLab GUI. The user chooses the audio of their performance and the score.

2) **Communication with the database**: the interface indicates the files to the database and passes them to the main routine.

3) **Alignment stage**: this routine in C gets the .dat file as input and generates the optimal path that is passed to the next module.

4) **Musical statistics extraction stage**: This routine takes the optimal path and estimates the beat time, tempo, and rhythm of each measure and sends them to the main process routine and comparison block.

5) **Communication with the interface**: the main routine sends the musical statistics to the interface to display them in plots.

6) **File import**: the user chooses the information of the performer with which he wants to compare.

7) **Communication with the database**: the interface indicates the file to the database and sends it to the main routine again.

8) **Comparison stage**: this stage extracts the information that the user has indicated, estimates the comparison of the musical statistics together with the triangle and tree plots and sends them to the main routine.

9) **Communication with the interface**: the main routine sends the comparison information to the interface in order to display it in plots.

Our interface is made up of two GUIs: *GUI1* and *GUI2*. In *Figure 4.45* we can see the relationship between the user and the two graphical interfaces. Our application has a main interface (*GUI1*), where we can visualize the different musical statistics. The second interface (*GUI2*) solves the problem of generating training data for different recordings of

the musician. This *GUI2* allows the user to generate the training data with the new real audio recording, so the user choose the MIDI file and the wav file and the system generates the necessary training, audio and comparison files and save them in their respective folders of the database, taking into account if the audio corresponds to **URMP** or **Mazurka Project dataset**. In this way, the user musician has the possibility to compare himself or another musician that is not in our initial database and will be able to build his own database with his own recordings or recordings of another musician. Moreover, he will be able to observe the improvement of the performance of a certain musical piece as many times as he wishes.

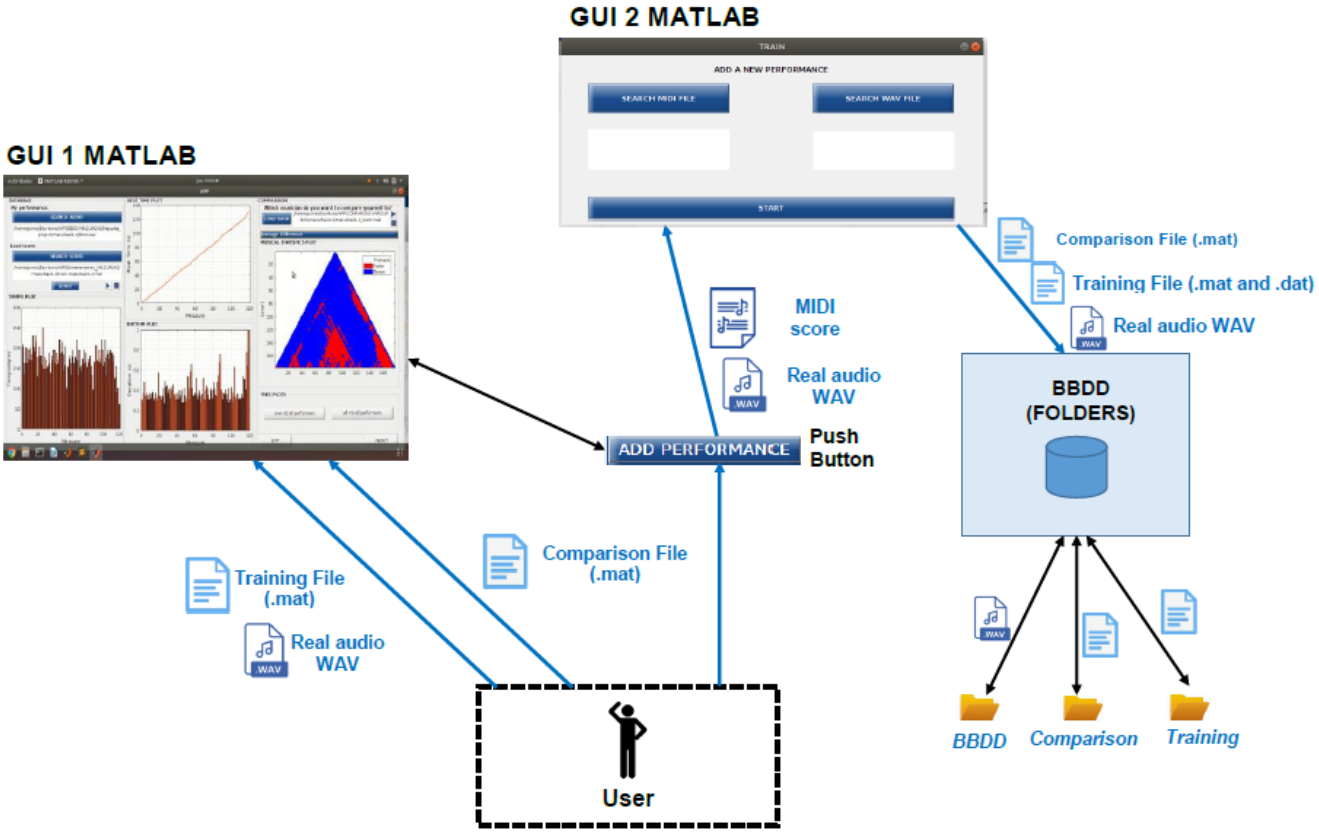


Figure 4.47 – Relationship between the two interfaces and the user musician.

The operation between the two interfaces is very simple: in *GUI1* there is a push button *("ADD PERFORMANCE")* and pressing it display *GUI2*. Being in the second interface, we simply have to select the **MIDI file** and the **wav file** with the new recording. The system generates the files and save them into the database.

4.2.4 *Database of the Application*

As we have explained on other occasions, our database is made up of the URMP and Mazurka Project datasets. Also, the user musician can grow this database by adding his own recordings of the musical pieces that make up the two databases used as reference. Our database is composed by 6 folders.

The folder structure that our database follows is:

- **BBDD-MAZURKAS**: contains MIDI and wav files (audio recording) of the Chopin's Mazurkas.
- **BBDD-URMP**: contains MIDI and wav files (audio recording) of each musical piece of URMP dataset.
- **Entrenamiento-MAZURKAS**: contains the training files (.mat and .dat format) with the training data of each musical piece from Mazurka Project dataset.
- **Entrenamiento-URMP**: contains the training files (.mat and .dat format) with the training data of each musical piece from URMP dataset.
- **COMPARISON-MAZURKAS**: contains the training files (.mat format) with the training data of each musical piece for each performer from Mazurka Project dataset.
- **COMPARISON-URMP**: contains the comparison files (.mat format) with the comparison data of each musical piece for each performer from URMP dataset.

5 Results

In this section of the document I am going to show the different results that we have obtained during the tests of the system. There is a great variety of databases with a multitude of musical signals with which you can work on projects related to the audio and music processing. In this work, databases with real sounds have been used, that is, live sound recordings. It should be said that, as we already know, we have decided to implement *DTW* offline in our application, although we have also developed the online method. However, the results we have obtained mostly focus on both online and offline approaches.

After testing some signals, we have found that the system is robust against vibrato in sound, noisy environments, silences at the beginning or at the end and other aspects that we discuss throughout this chapter. If we focus on the scores and recordings used to assess the system, we have used **two databases**: **Mazurka Project** with Chopin's Mazurkas and **URMP**.

5.1 Results with Mazurka Project Database

This section presents the results of the evaluation of the score alignment where we can see a comparison between annotated data obtained from the spreadsheets and automatic data obtained with the score alignment for the Mazurka Project dataset.

As we have previously mentioned, this dataset was developed by **CHARM (Center for the History and Analysis of Recorded Music)** to study the behavior of the Chopin's Mazurkas in the alignment field. In *Table 5.1* we can see the information of the different musical pieces of the database.

Musical piece	Performer	Duration (seconds)
Mazurka Op. 17 – 4th movement	Biret	285
	Horowitz	256
	Rubinstein	293
Mazurka Op. 24 – 2nd movement	Biret	129
	Ezaki	118
	Katin	128
	Kiepura	137
	Magaloff	146

	Nelly Ben Or	136
	Perlemuter	164
	Shebanova	121
Mazurka Op. 30 – 2nd movement	Biret	90
	Brailowsky	69
	Francois	70
	Michelangeli	94
	Shebanova	80
Mazurka Op. 33 – 2nd movement	Paderewsky	120
Mazurka Op. 63 – 3rd movement	Ashkenazy	145
	Duchoud	126
	Horowitz	118
	Julio García	122
	Magaloff	126
	Morozova	141
	Perlemuter	110
	Rubinstein	150
Mazurka Op. 68 – 3rd movement	Chiu	96
	Indjic	103
	Luisada	104
	Rubinstein	82

Table 5.1 – Musical pieces of Mazurka Project database.

We have two types of comparisons: one compares the tempo data and the other compares the correlations between the tempo data. In these results, we have analyzed the following Mazurkas: *Op. 24-2, Op. 30-2 and Op.68-3*. The following Figures show the similarity in terms of tempo and correlation between annotated and automatic data.

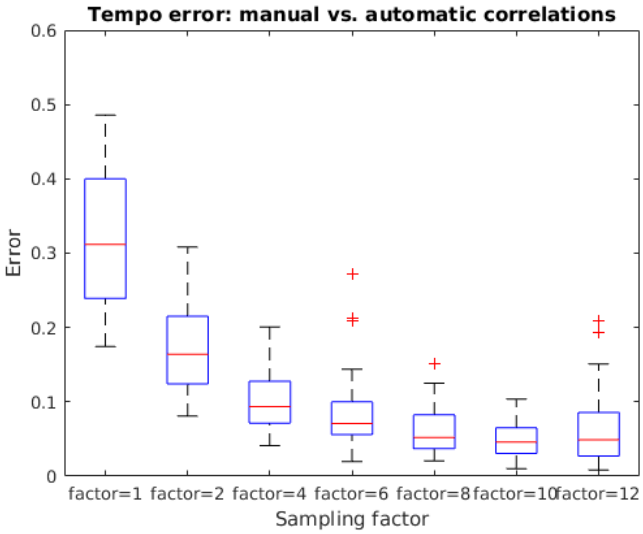


Figure 5.1 – Tempo correlation error manual vs. automatic correlations of Mazurka Op 24-2.

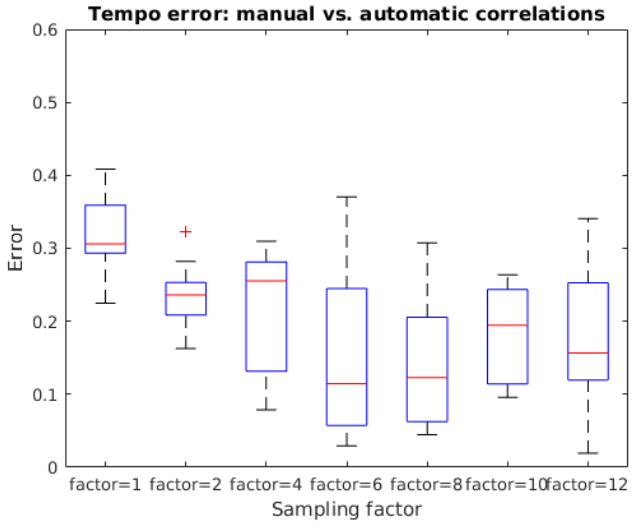


Figure 5.2 – Tempo correlation error manual vs. automatic correlations of Mazurka Op 30-2.

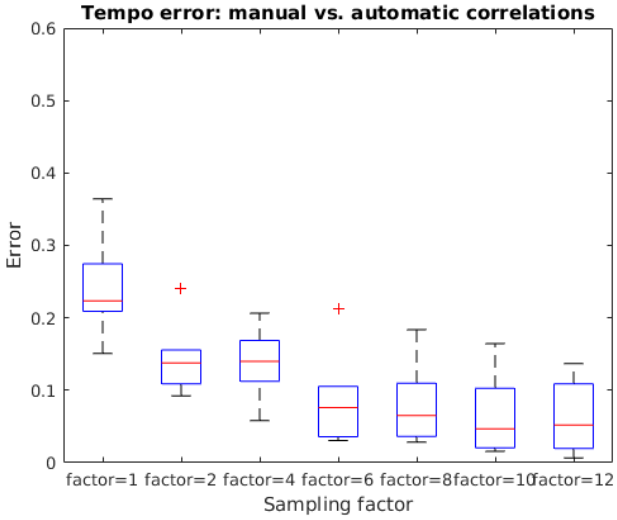


Figure 5.3 – Tempo correlation error manual vs. automatic correlations of Mazurka Op 68-3.

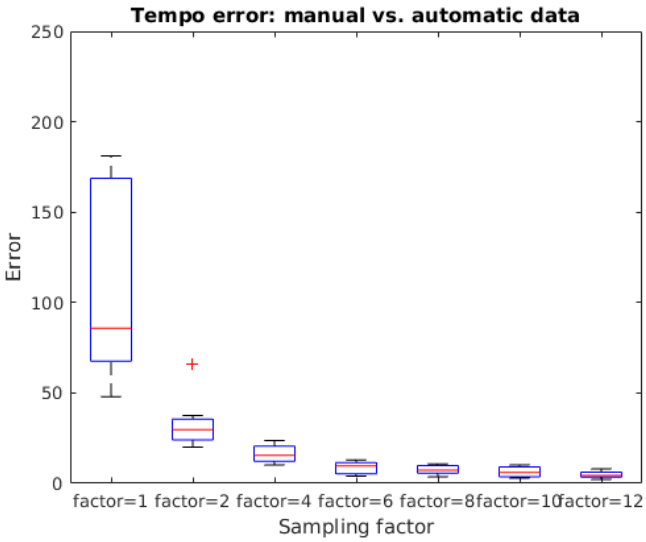


Figure 5.4 – Tempo error manual vs. automatic data of Mazurka Op 24-2.

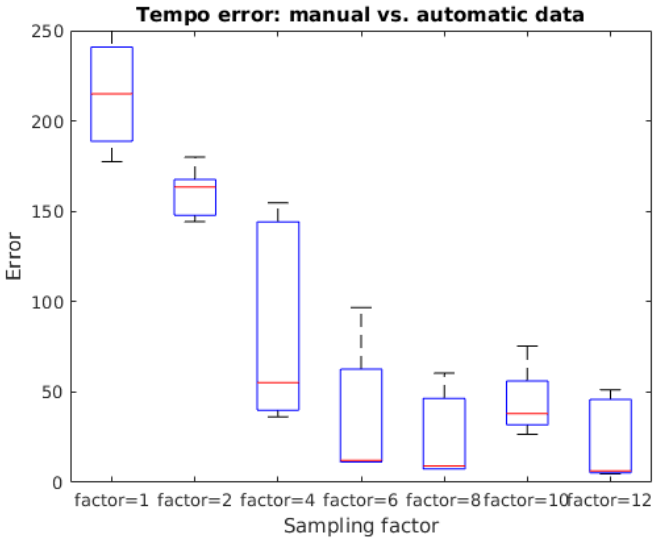


Figure 5.5 – Tempo error manual vs. automatic data of Mazurka Op 30-2.

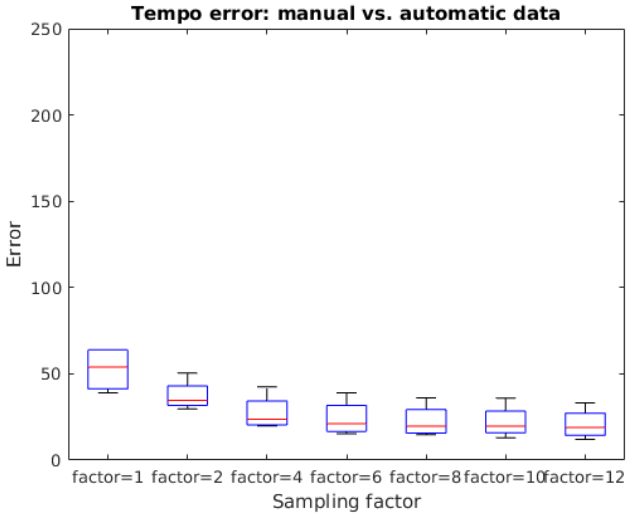


Figure 5.6 – Tempo error manual vs. automatic correlations of Mazurka Op 68-3.

We can see that the error in terms of tempo and correlation becomes stable for a **sampling factor 4**, which is the sampling factor that we have used to represent the comparison plots shown in *section 4.1.6.2*.

In *Tables 5.2* and *5.3*, we can see the average error in terms of tempo and correlation of the mazurkas analyzed.

Musical piece	**Sampling factor**						
	1	**2**	**4**	**6**	**8**	**10**	**12**
Mazurka 24-2	85.9	29.6	15.4	9.7	7.3	5.9	4.45
Mazurka 30-2	214.9	163.3	55.2	12.23	9.17	38.1	6.3
Mazurka 68-3	200	34.7	23.6	21.1	19.6	19.7	18.9

Table 5.2 – Tempo error in bpm: manual vs. automatic data.

Musical piece	**Sampling factor**						
	1	**2**	**4**	**6**	**8**	**10**	**12**
Mazurka 24-2	0.31	0.16	0.09	0.07	0.05	0.04	0.04
Mazurka 30-2	0.3	0.24	0.25	0.11	0.12	0.19	0.19
Mazurka 68-3	0.22	0.13	0.14	0.07	0.06	0.04	0.05

Table 5.3 – Correlation error: manual vs. automatic data.

As the previous *Figures and Tables* show, the tempo error in terms of correlation and data of tempo is low. This means that the similarity between the annotated and automatic data is very close, our system is working.

As mentioned in *section 5.6.1* of this work, this database provides us with the annotated data about beat time, duration and tempo of each Mazurka. According to this and apart from analyzing the work of our system, we have verified that in some Mazurkas the annotated data does not correspond to the interpretation of the musician.

The best way to analyze if the annotated data is correct or not is to convert beat times in seconds to path in frames. In this way, we can verify if the path obtained from the annotated data corresponds to the approximate path of the interpretation.

Mathematically, we can convert data from seconds to frames and from frames to seconds. We achieve this by knowing the sampling frequency of the performance (in our case it is always 44100 Hz) and the hop size in samples (in our case one frame is always made up of 1411 samples). Therefore, this conversion can be defined by the following equation:

$$pos\_min = beat\_time(s) \cdot \frac{fs\ (Hz)}{hop\_size\ (samples)} = beat\ time(s) \cdot \frac{44100}{1411} \qquad (29)$$

where beat_time corresponds to the beat times of the performance, fs is the sampling rate of the audio file and hop_size corresponds to the hop size in samples of each frame.

In this way, we can obtain the path of the annotated data, that is, we can estimate the MIDI frame corresponding to each real frame.

In *Figure 5.7*, we can see an example in which the alignment of the annotated data is not correct, since the path goes through all the MIDI frames of the score but does not arrive until the last real frame of the performance, it ends earlier. However, if we analyze the automatic alignment we can see that the path achieved is the correct one.

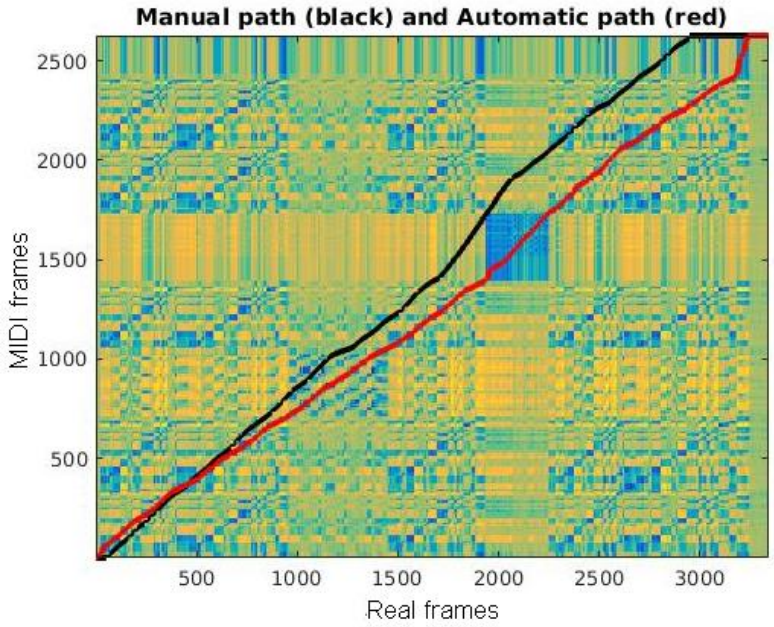


Figure 5.7 – Path of annotated data vs. path of automatic data. The musical piece corresponds to the Mazurka 68-3 performed by Chiu.

5.2 Results with URMP Database

This section presents the results of the evaluation of the score alignment where we can see a comparison with other reference methods for the **URMP** dataset created by **Li et al**. [41].

This database is part of the **University of Rochester** project supported by the **National Science Foundation** and titled **"BIGDATA: F: Audio-Visual Scene Understanding"** [36]. The dataset is comprised of a series of simple multi-instrument music pieces assembled from separately recorded performances of individual tracks. The database includes 44 classical chamber music pieces distributed among different groups of musicians (11 duets, 12 trios, 14 quartets and 7 quintets) and played by 14 different common instruments in orchestra. For each piece, the dataset provide us the musical score in MIDI format, the high-quality individual instrument audio recordings and the assembled pieces [38]. Although URMP is not an orchestra dataset, it has been used to evaluate the behavior of our proposal in ensembles with a reduced number of instruments.

In *Table 5.4* we can see the information of the different musical pieces of the database [37]:

No.	Musical Piece	Instrumentations	Duration (seconds)
01	Jupiter	Violin Cello	63
02	Sonata	Violin Violin	46
03	Dance of the Sugar Plum Fairy	Flute Clarinet	97
04	Allegro for Musical Clock	Flute Flute	157
05	The Entertainer	Trumpet Trumpet	87
06	The Entertainer	Saxophone Saxophone	88
07	Air on the G string	Trumpet Trombone	270
08	Spring from the Four Seasons	Flute Violin	35
09	Jesus Bleibet Meine Freude	Trumpet Violin	199
10	March from Occasional Oratorio	Trumpet Saxophone	103
11	Ave Maria	Oboe Cello	104
12	Spring from the Four Seasons	Violin Violin Cello	131
13	Hark the herald angels sing	Violin Violin Viola	47
14	Waltz from sleeping Beauty	Flute Flute Clarinet	93
15	Theme from Surprise Symphony	Trumpet Trumpet Trombone	53
16	Theme from Surprise Symphony	Trumpet Trumpet Saxophone	53

17	Nocturne	Violin Flute Clarinet	96
18	Nocturne	Violin Flute Trumpet	96
19	Pavane	Clarinet Violin Cello	133
20	Pavane	Trumpet Violin Cello	133
21	La Rejouissance	Clarinet Trombone Tuba	76
22	La Rejouissance	Saxophone Trombone Tuba	76
23	La Rejouissance	Clarinet Saxophone Tuba	76
24	Pirates of the Aegean	Violin Violin Viola Cello	50
25	Pirates of the Aegean	Violin Violin Viola Saxophone	50
26	In the Hall of the Mountain King	Violin Violin Viola Cello	85
27	In the Hall of the Mountain King	Violin Violin Viola Saxophone	85
28	The Art of the Fugue	Flute Oboe Clarinet Bassoon	175
29	The Art of the Fugue	Flute Flute Oboe Clarinet	172
30	The Art of the Fugue	Flute Flute Oboe Saxophone	172
31	Slavonic Dance	Trumpet Trumpet Horn Trombone	82
32	The Art of the Fugue	Violin Violin Viola Cello	174
33	Fur Elise	Trumpet Trumpet Horn Trombone	48
34	The Art of the Fugue	Trumpet Trumpet Horn Trombone	174
35	Rondeau from Abdelazer	Violin Violin Viola Double Bass	128
36	Rondeau from Abdelazer	Violin Violin Viola Cello	128
37	Rondeau from Abdelazer	Flute Violin Viola Clarinet	128
38	Jerusalem	Violin Violin Viola Cello Double Bass	119
39	Jerusalem	Violin Violin Viola Saxophone Double Bass	119
40	Miserere Mei Deus	Flute Flute Oboe Clarinet Bassoon	40

41	Miserere Mei Deus	Flute Flute Oboe Saxophone Bassoon	40
42	Arioso	Trumpet Trumpet Horn Trombone Tuba	255
43	Chorale	Trumpet Trumpet Horn Trombone Tuba	53
44	String Quintet K515	Violin Violin Viola Viola Double Bass	225

Table 5.4 – Musical pieces of URMP database

In order to analyze the performance of our alignment methods (Carrillo's offline and online), we have carried out a comparison with six reference methods: **Carabias' offline** [22], **Carabias' online** [22], **Munoz's offline** [26], **Munoz's online** [26], **Ellis' offline** [24] and **Soundprism** [25]. *Figure 5.8* shows the alignment results in terms of accuracy values of the analyzed methods as a function of the onset deviation threshold. The threshold value varies between 50 and 2000 ms. As can be observed, all the offline approaches reach similar results, although our offline approach obtains a higher precision value than the others in the threshold range 50 to 150 ms. Note that the number of instruments in this ensembles are limited (2 to 5 instruments). Regarding the online techniques, our online method obtains an accuracy value greater than the Carabias' online, Munoz's online and Soundprism approaches. As we have mentioned on other occasions, our method is based on the Carabias' approach, the main difference is based on using a regularization value to obtain a variable cost function. For this reason, our approach improves compared to the others, although the online approach is where this difference is most appreciated.

According to *Figure 5.8*, we can see that our methods require a tolerance window of at least 1 s to achieve the optimal alignment. Note that we show the values for **Carrillo's approaches** with the optimal lambda parameter for 100 ms.

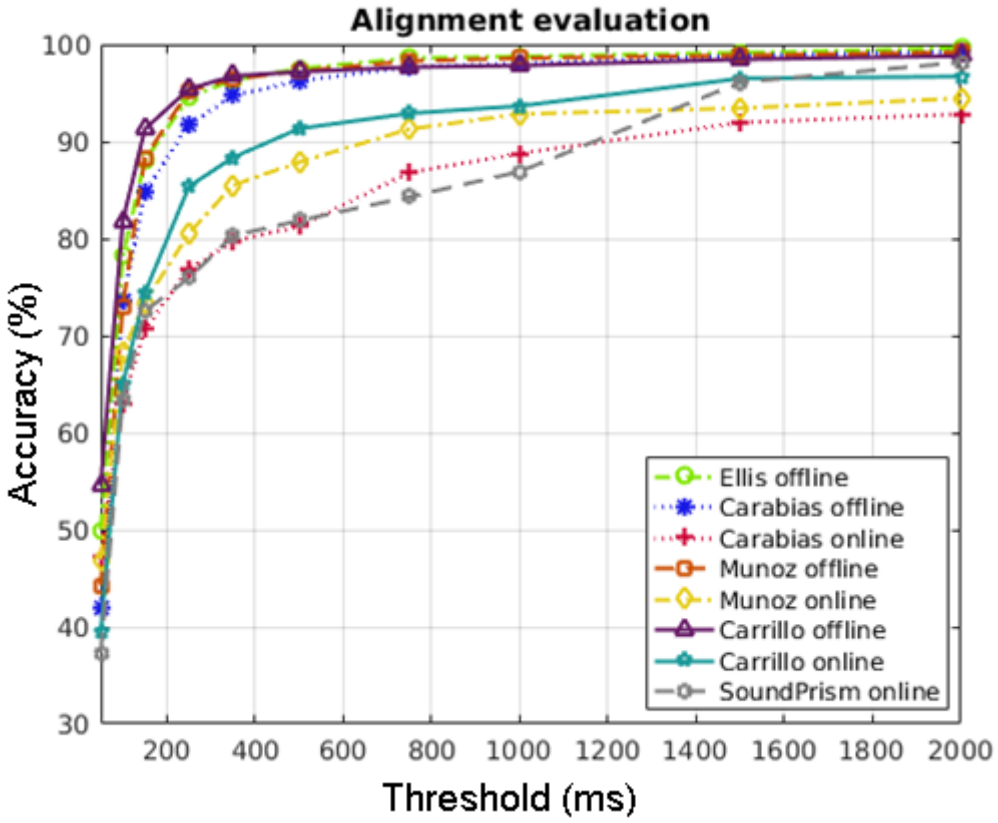


Figure 5.8 – Alignment evaluation in terms of accuracy values in function of the tolerance window over the URMP dataset.

In *Table 5.5* we can see the accuracy value in each value of tolerance window for all the methods that we have compared.

Threshold (Tolerance window)	Methods							
	Carrillo offline	Ellis offline	Munoz offline	Carabias offline	Carabias online	Munoz online	Carrillo online	Soundprism online
50	54.5%	49.9%	44.2%	41.9%	47.1%	46.8%	39.4%	37.3%
100	81.7%	78.2%	73.1%	73.5%	62.9%	68.3%	65.1%	63.5%
150	91.2%	88%	88.3%	84.7%	70.6%	73.4%	74.4%	72.5%
250	95.4%	94.6%	95.2%	91.8%	76.9%	80.5%	85.3%	75.9%
350	96.7%	96.3%	96.4%	94.7%	79.7%	85.5%	88.4%	80.3%
500	97.2%	97.5%	97.2%	96.2%	81.3%	87.9%	91.3%	81.9%
750	97.6%	98.6%	98.3%	97.6%	86.8%	91.3%	92.9%	84.3%
1000	97.8%	98.7%	98.6%	98.1%	88.8%	92.8%	93.6%	86.9%
1500	98.5%	99.1%	98.9%	98.9%	91.9%	93.4%	96.5%	96.1%
2000	98.6%	99.6%	99.2%	99.3%	92.8%	94.4%	96.7%	98.2%

Table 5.5 – Alignment evaluation in terms of accuracy values in function of the tolerance window over the URMP dataset.

As indicated before, we have to use the optimal value for the regularization constant (**λ**) in order to obtain the best possible alignment, but we have to take into account the two different approaches (online and offline). If we used our online method for automatic accompaniment, we are interested in the best possible result for a tolerance window error of 100 ms (**λ = 2.4**) because our system has to have a good and efficient response in the shortest possible time when using it as a real-time application. On the other hand, in the offline approach, we are interested in having a small error for small thresholds, also between 50 and 100 ms, so we take into account that the best accuracy value should be in an error of the tolerance window of 100 ms (**λ = 3**). We can see the **optimal regularization value (λ)** for each approach in *Figures 5.9* and *5.10*, where we show the accuracy value against the different lambda values for a threshold of 100 ms.

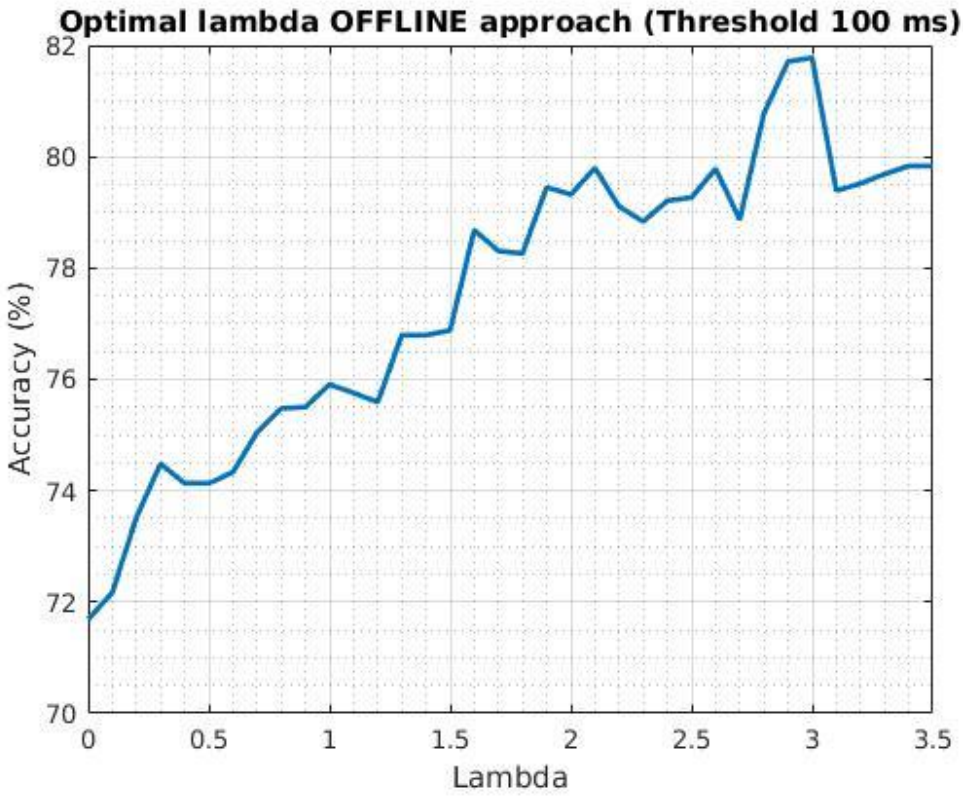


Figure 5.9 – Plot of optimal λ in OFFLINE approach (threshold 100 ms). The plot represents the accuracy value (%) for each λ value.

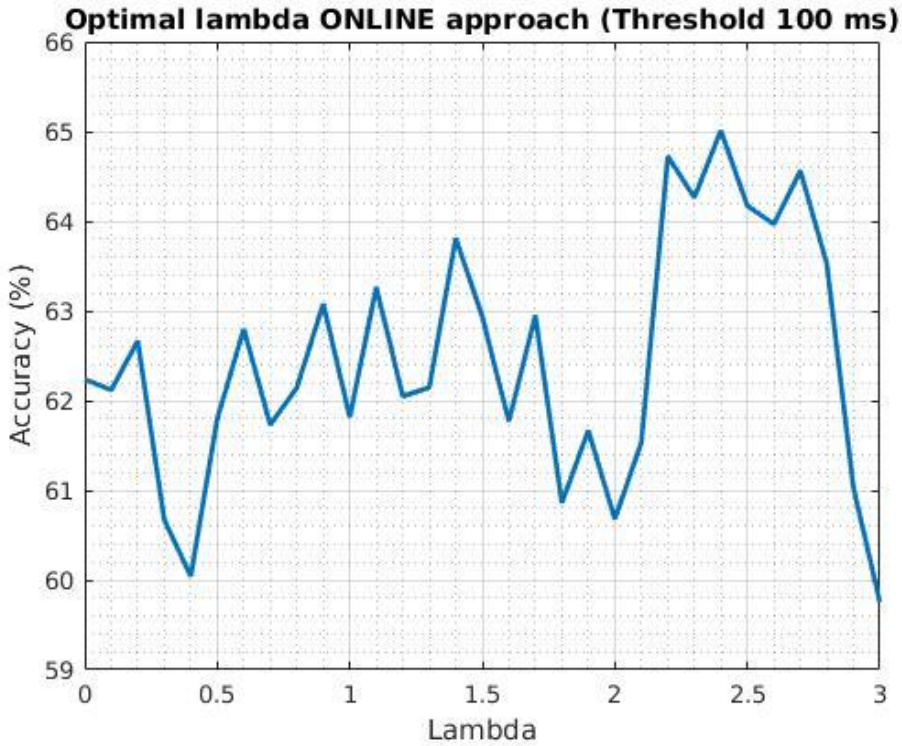
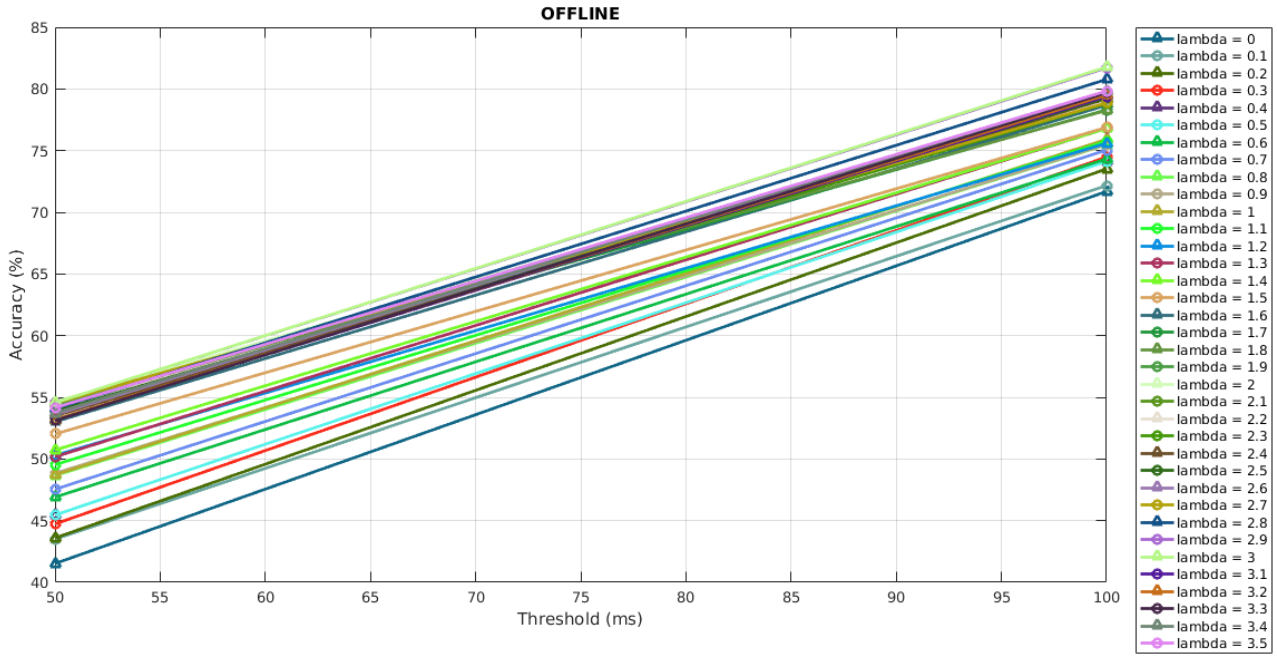
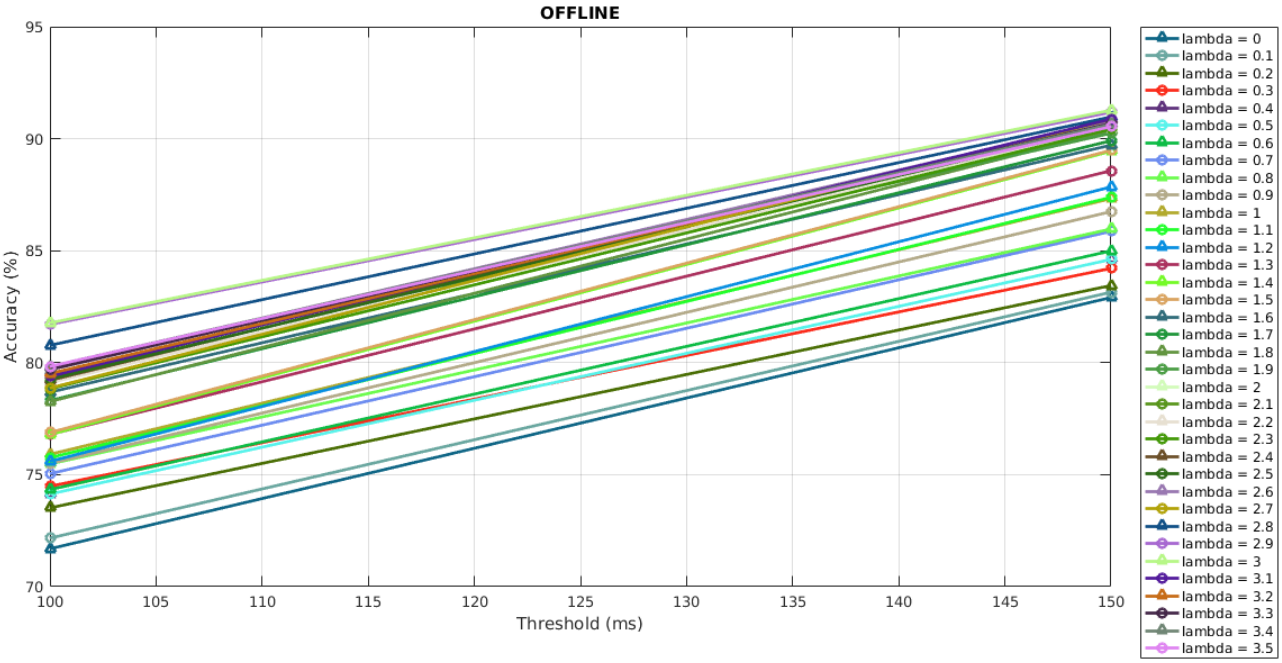


Figure 5.10 – Plot of optimal λ in ONLINE approach (threshold 100 ms). The plot represents the accuracy value (%) for each λ value.

To achieve the **optimal λ value** for different tolerance windows, we have swept the λ value from 0 to 3.5 in offline approach and from 0 to 3 in online approach. We have done this with the intention of estimating the accuracy for each regularization and threshold value, as we can see in the following *Figures*. The **λ sweep** for **online** and **offline** approach is different because when we reached the optimal value in the offline approach the accuracy still did not decrease and we had to estimate it for more values. We can verify that the chosen lambda value does not decrease too backwards (50 ms) or forwards (beyond 100 ms).


OFFLINE
Accuracy (%)
Threshold (ms)
85
80
75
70
65
60
55
50
45
40
50
55
60
65
70
75
80
85
90
95
100
lambda = 0
lambda = 0.1
lambda = 0.2
lambda = 0.3
lambda = 0.4
lambda = 0.5
lambda = 0.6
lambda = 0.7
lambda = 0.8
lambda = 0.9
lambda = 1
lambda = 1.1
lambda = 1.2
lambda = 1.3
lambda = 1.4
lambda = 1.5
lambda = 1.6
lambda = 1.7
lambda = 1.8
lambda = 1.9
lambda = 2
lambda = 2.1
lambda = 2.2
lambda = 2.3
lambda = 2.4
lambda = 2.5
lambda = 2.6
lambda = 2.7
lambda = 2.8
lambda = 2.9
lambda = 3
lambda = 3.1
lambda = 3.2
lambda = 3.3
lambda = 3.4
lambda = 3.5


OFFLINE
Accuracy (%)
Threshold (ms)
95
90
85
80
75
70
100
105
110
115
120
125
130
135
140
145
150
lambda = 0
lambda = 0.1
lambda = 0.2
lambda = 0.3
lambda = 0.4
lambda = 0.5
lambda = 0.6
lambda = 0.7
lambda = 0.8
lambda = 0.9
lambda = 1
lambda = 1.1
lambda = 1.2
lambda = 1.3
lambda = 1.4
lambda = 1.5
lambda = 1.6
lambda = 1.7
lambda = 1.8
lambda = 1.9
lambda = 2
lambda = 2.1
lambda = 2.2
lambda = 2.3
lambda = 2.4
lambda = 2.5
lambda = 2.6
lambda = 2.7
lambda = 2.8
lambda = 2.9
lambda = 3
lambda = 3.1
lambda = 3.2
lambda = 3.3
lambda = 3.4
lambda = 3.5

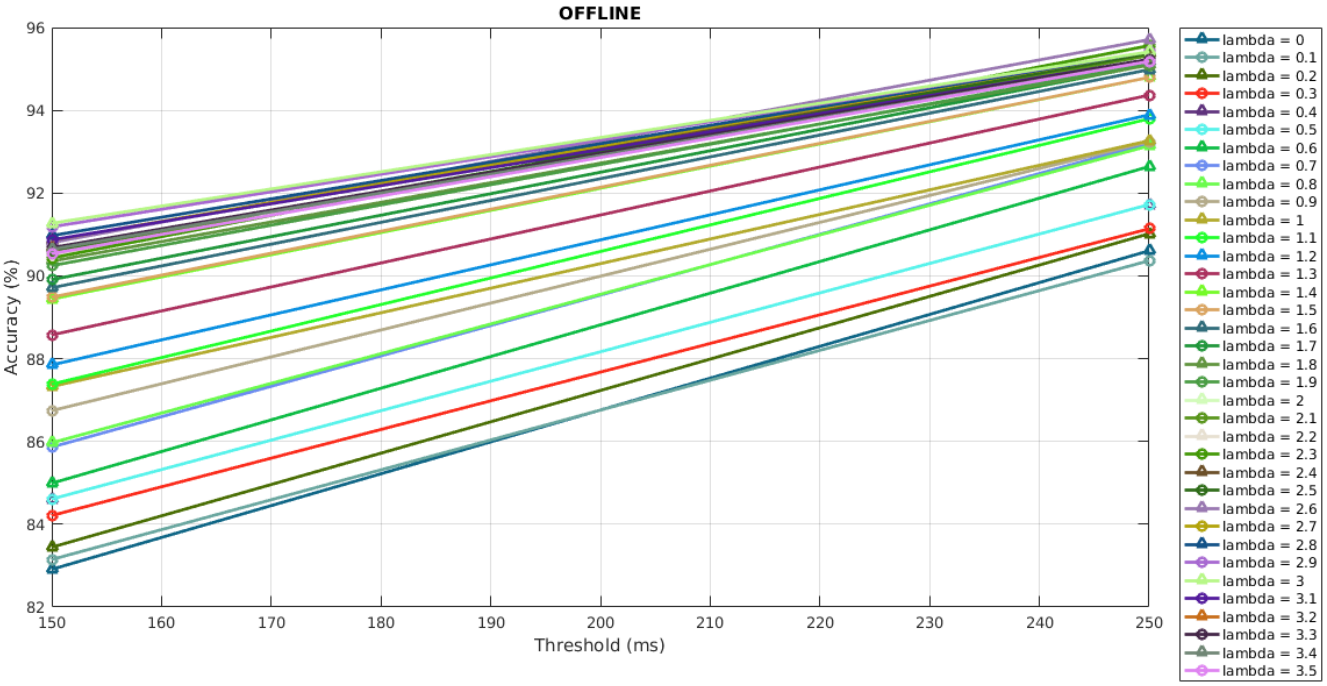
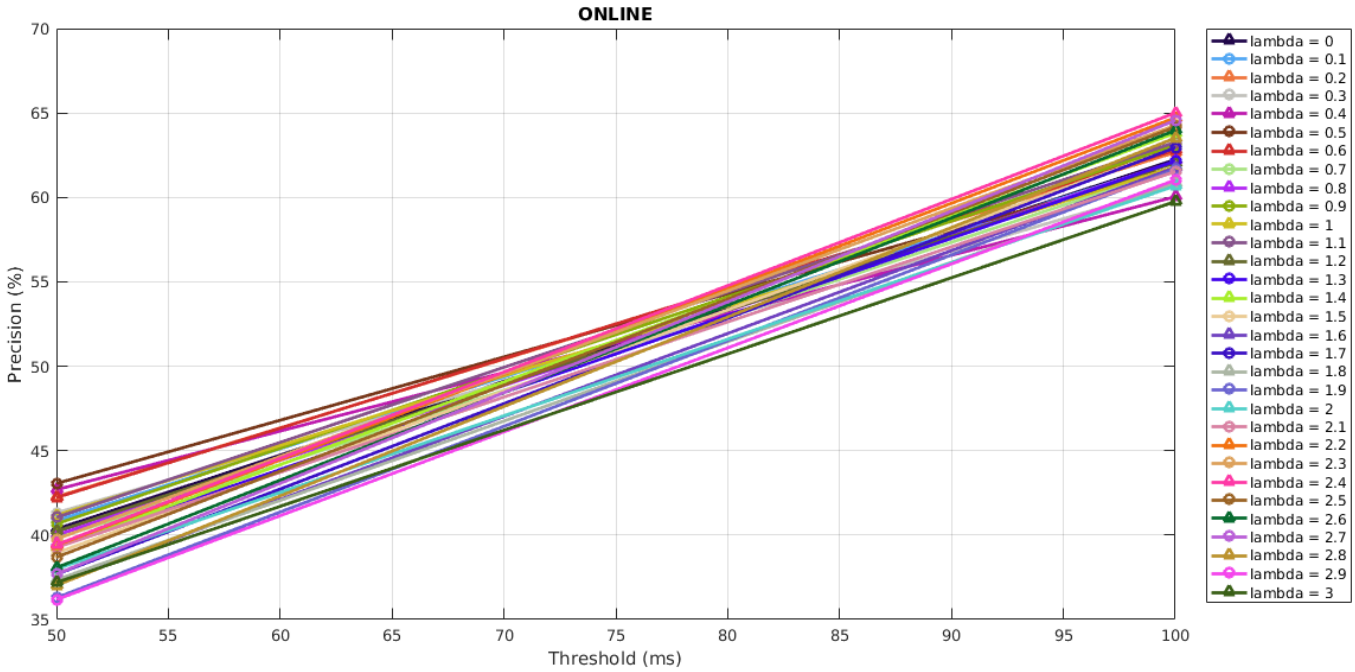


Figure 5.11 – Lambda value sweep in OFFLINE version from 0 to 3.5. For each lambda value we can see the accuracy value for each threshold value (ms).



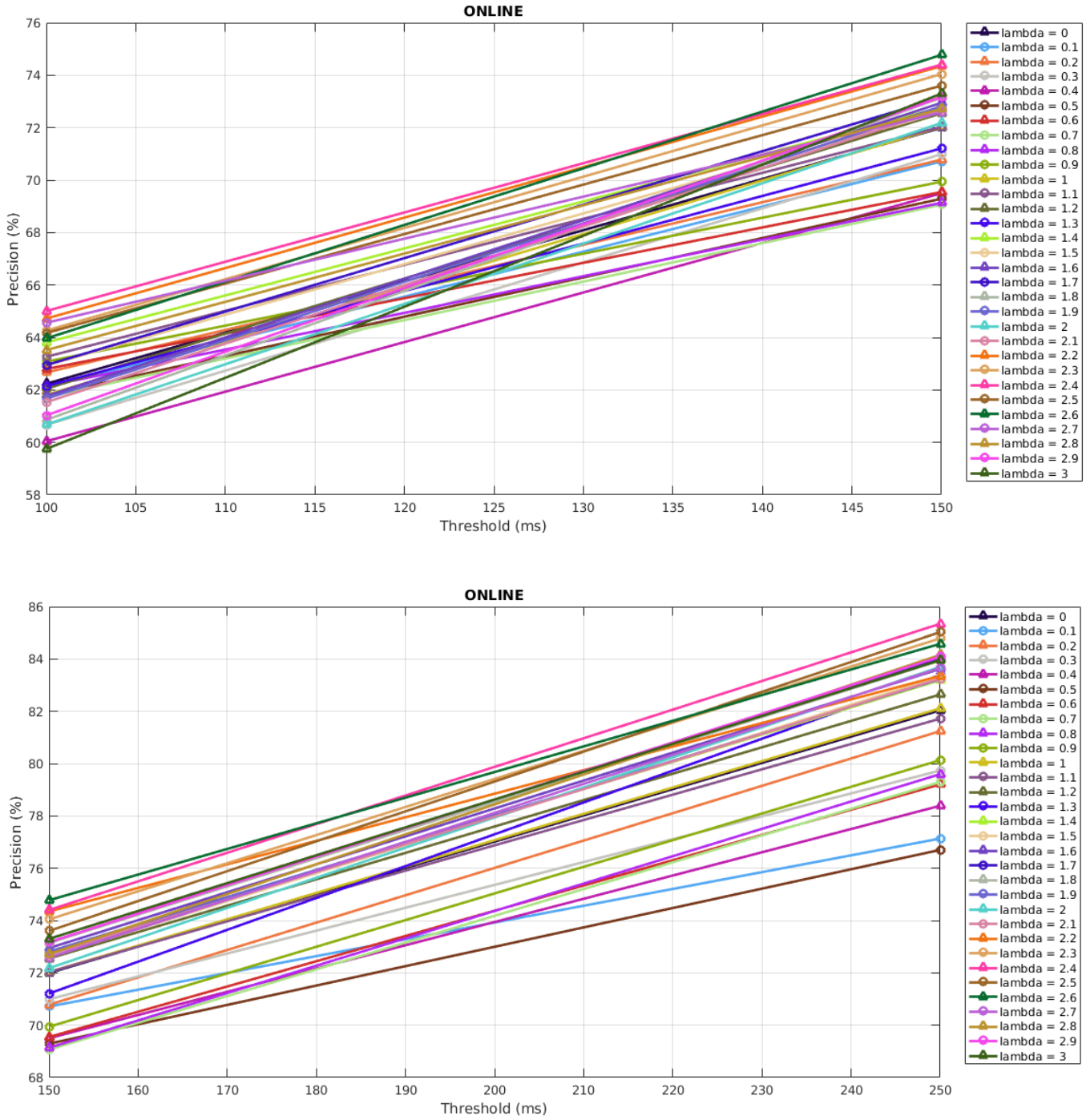


Figure 5.12 – Lambda value sweep in ONLINE version from 0 to 3. For each lambda value we can see the accuracy value for each threshold value (ms).

As we can see in the previous *Figures*, the online and offline methods proposed in this work give better results than the methods with which we are comparing. To be honest, the online approach is the one that gives better results for the optimal regularization value (**λ = 2.4**), although the offline approach gives very good results and better than its competing algorithms for a relatively small tolerance window and for the optimal regularization value (**λ = 3**).

6 Conclusions

In this Master's Thesis, we have proposed an application with which the user can improve their skills with their instruments in a simple, interactive and educational way. We have been able to verify that having the score and the instrument, we are able to extract the musical statistics related to the performance and compare ourselves with other performances.

We have evaluated the proposed alignment algorithm with audio signals belonging to two databases: **Mazurka Project** and **URMP**. With the first, we have evaluated the difference in tempo and correlation data between annotated and automatic data. The results we have obtained with this dataset have been good and with a sampling factor of 4 the minimum error that can be achieved with the system is stabilized. Using the URMP database, we have evaluated our proposed algorithm and compared it with the algorithms in the bibliography. We have observed that our algorithm performs better than the others with a tolerance window of around 100 ms. To be honest, we have performed better on the online approach than the offline approach, but the two approaches still work better than their respective state-of-the-art methods.

Leaving aside the field of audio signal processing and music, throughout the project we have effectively integrated concepts that belong to different areas related to Telecommunications Engineering, such as sound physics, digital signal processing audio, the analysis of signals in the temporal and spectral domains, the programming of an application or its implementation in a graphical interface; and also other fields of knowledge such as music theory.

In general, we can say that the proposed objectives have been achieved, since we have managed to implement a functional application that is made up of different modules and that meets the requirements that were sought in the proposal of this work. Although the application works satisfactorily, you can always keep improving to achieve better results and achieve a more complete application in every way. Considering the above, possible future lines could be to add more features to the application or integrate the system in a more complex graphical programming environment that can be used on different devices such as mobile or tablet. In addition, the number of scores the application allows could be increased by adding more databases to the application database.

7 Appendix

7.1 User manual

As we already know, an interface has been developed for the execution of systems for the extraction of musical statistics and comparison between musicians. For this, the MatLab GUIDE tool has been used, the analysis of musical signals can be made available to any user and, above all, we make an interactive application available to the user in which they can visualize their progress in the study of musical pieces from the database.

First of all, in order to run the application, we have to open MatLab and then we have to select in the workspace the folder where the application called APP is located. Next, we type APP in the MatLab command window and the application interface start.

Below we have the user manual to follow in order to understand the behavior of the designed interface. First, we can see the working environment in *Figure 7.1* with each of the parts that make it up.

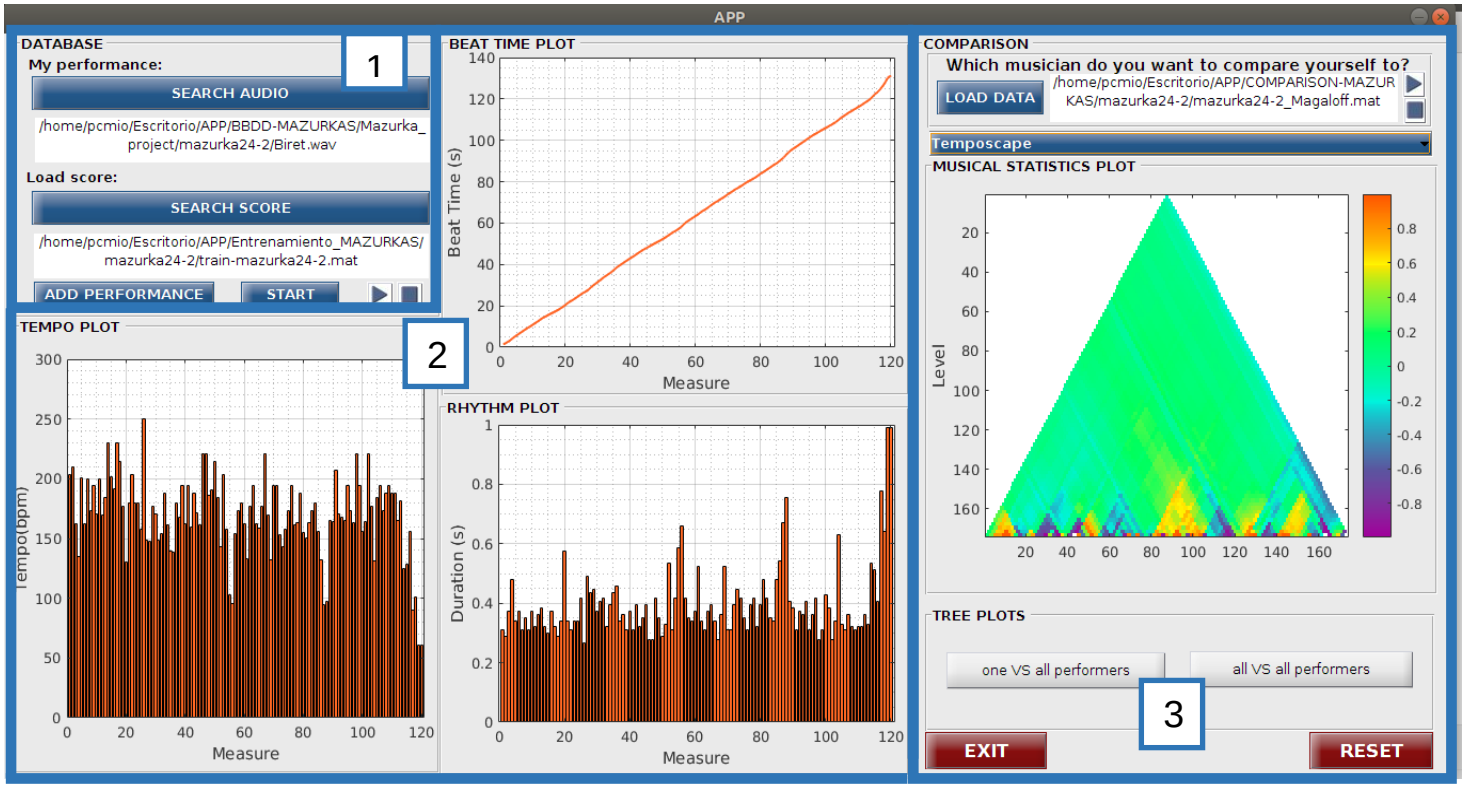


Figure 7.1 – User interface.

As we can see in the previous *Figure*, the working environment is very intuitive and is made up of 3 main sections (1 to 3); it is also made up of the interface described in the *Figure* and another that we will explain later. The behavior of the interface is detailed below, explaining each of the sections separately.

First, we detail the operation of the different buttons that make up the application:

Figure 7.2 – Search audio button.

This button allows us to select the **audio file** with which we want to work, and to which we will apply alignment and extraction of musical statistics.

Figure 7.3 – Search score button.

This button allows us to select the **score** with which we want to work, and which corresponds to the audio file. By pressing this button we will choose the .mat file in the database that contains all the training information of the score.

Figure 7.4 – Start button.

This button allows us to run the application.

Figure 7.5 – Add performance button.

This button allows us to add new recordings to the database to be able to compare ourselves with them whenever we want. Let's imagine that a music student has to study a Chopin Mazurka, since they can record as many times as they want and add their recordings to the database to see their improvements. By pressing this button the second interface will appear.

Figure 7.6 – Play button.

This button allows us to start playing the audio corresponding to section 1 and 4.

Figure 7.7 – Stop button.

This button allows us to stop playing the audio corresponding to section 1 and 4.

Figure 7.8 – Load data button.

This button allows us to load the data of the musician with whom the user wants to compare.

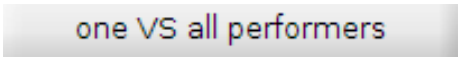


Figure 7.9 – One VS. all performers plot button.

This button shows us the plot with the tempo similarity in tree-shape one VS all performers.

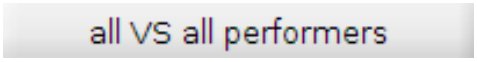


Figure 7.10 – All VS. all performers plot button.

This button shows us the plot with the tempo similarity in tree-shape all VS all performers.

Figure 7.11 – Exit button.

This button allows us to close the interface directly.

Figure 7.12 – Reset button.

This button allows us to reset all the parameters and plots of the interface.

Once the functionality of each of the buttons present in the user interface has been detailed, we explain each of the sections that make up the interface.

- **Section 1: Database**

This section allows us to select the recording of the performance of the musical piece and its score. We can also listen to the recording we have selected and start the process.

It is important to indicate that the input audio signal has to be a mono signal with a sampling rate of 44100 Hz.

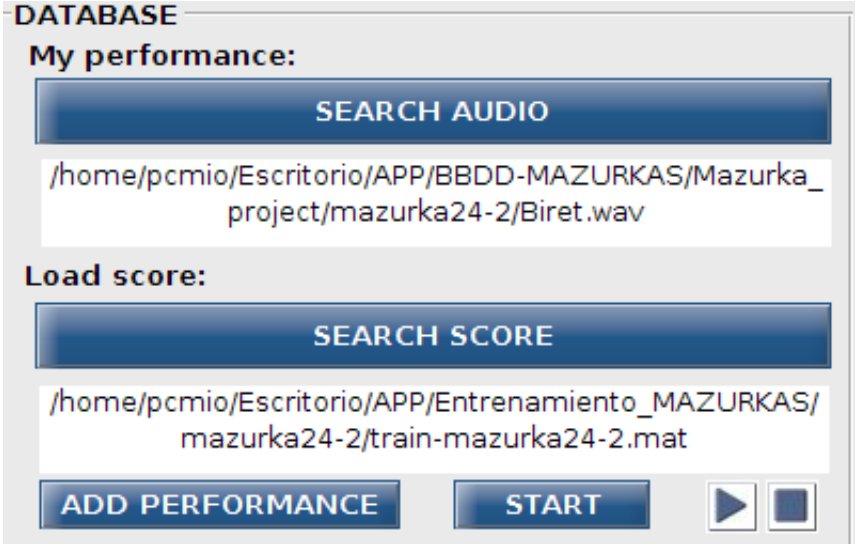


Figure 7.13 – Section 1 corresponding to the database.

This section has other functionality and consists of adding a new recording to the application database, making it a dynamic and interactive database with the user. When we press the "Add performance" button, another interface is displayed as we can see in *Figure 7.14*.

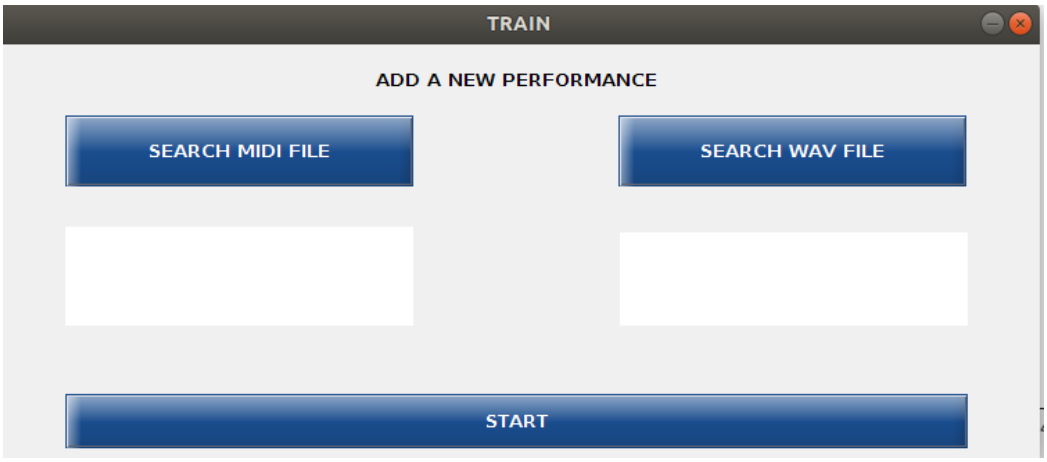


Figure 7.14 – Add performance interface.

This interface allows to the user to add a new recording to the database. For this, the user must select the MIDI file and the wav file with the performance and then start the process. This process starts the training method and the extraction of musical statistics for comparison. All these training, audio, score and comparison data is stored in the database in their corresponding folders explained in section 6.4 of this report.

- **Section 2: Musical statistics plot**

In this section, the system show the user the plots related to beat time, tempo and duration in each measure.

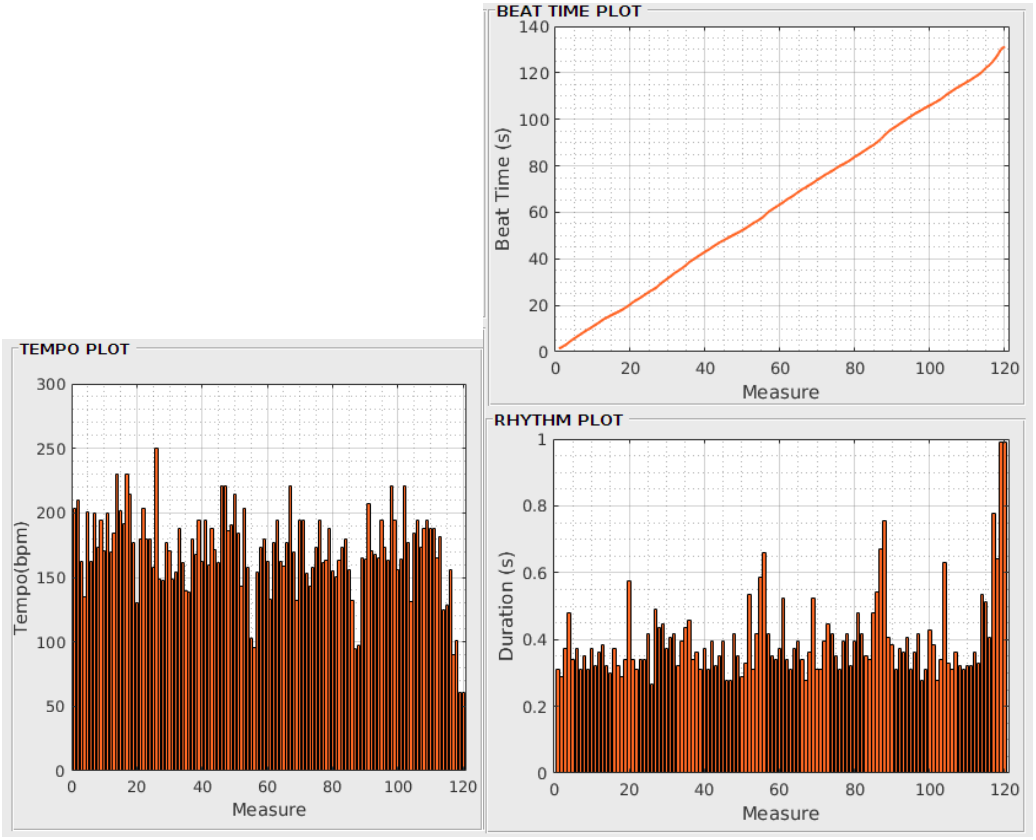


Figure 7.15 – Section 2 corresponding to the musical statistics plot.

- **Section 3: Comparison**

This section allows the user to load the comparison data of another musician and view all the types of plot available in the application and explained in *section 5.6.2*. The plots that we want to visualize are selected with a drop-down menu and to view the tree-shape plot the user presses the buttons and it appear in an adjacent figure.

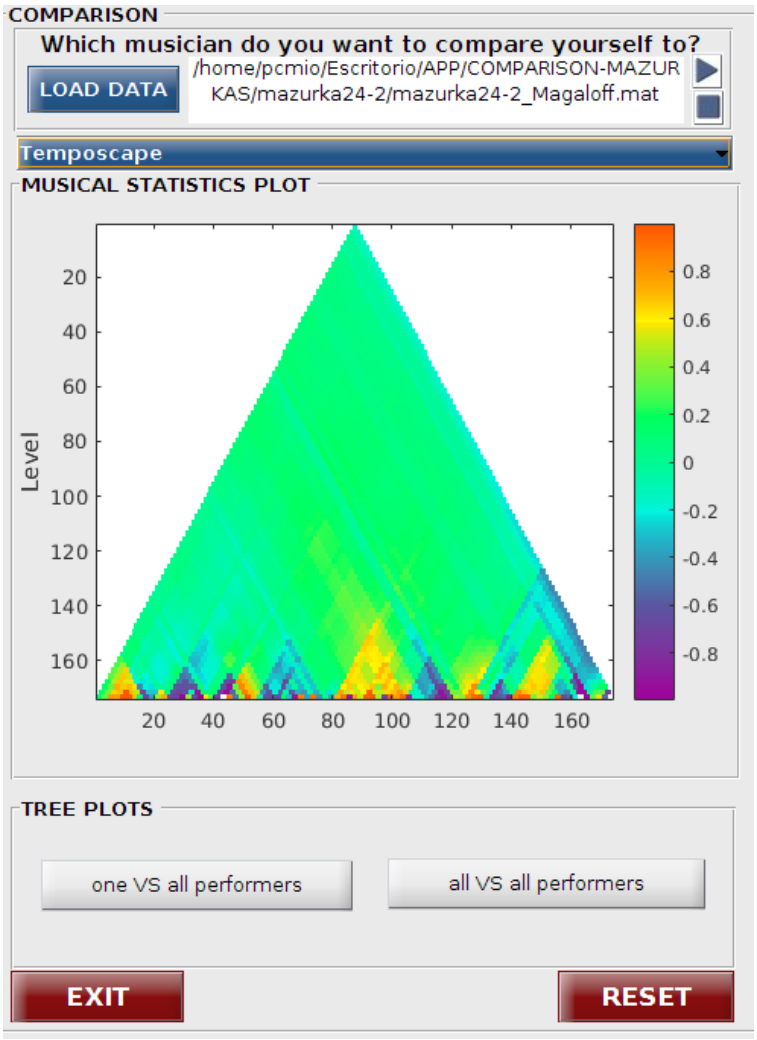


Figure 7.16 – Section 3 corresponding to comparison.

8 BIBLIOGRAPHY

[1] *Acoustics: An Introduction to Its Physical Principles and Applications*. Edition: -. Author: Allan D. Pierce. Editorial: Acoustical Society of Amer.

[2] *Blackwell handbook of sensation and perception*. Edition: E. Bruce Goldstein. Author: B. Goldstein. Editorial :Oxford, Blackweel Publishing. 8th Edition.

[3] R Paiva. Melody Detection in Polyphonic Audio. PhD thesis, Computer Science Department, Science and Technology, University of Coimbra (Portugal), 2006.

[4] F.J. Rodríguez Serrano. *Tesis Doctoral: Separación de Fuentes Sonoras en Señales Musicales*. Universidad de Jaén. Julio 2014.

[5] Oxford University Press Dictionaries (electronic resource): https://es.oxforddictionaries.com/

[6] Sergi Jordà Puig. *Audio digital y MIDI*. Guías Monográficas Anaya Multimedia, Madrid 1997.

[7] J.J. Carabias Orti. PhD *Thesis: Musical Instrument Models Estimation for Polyphonic Music Transcription.* Universidad de Jaén. December 2011.

[8] Rodriguez-Serrano, F. J., Carabias-Orti, J. J., & Vera-Candeas, P. (2016). *Tempo Driven Audio-to-Score Alignment Using Spectral Decomposition and Online Dynamic Time Warping.* ACM Transactions on Intelligent Systems and Technology, Vol. 8, No. 2, Article 22.

[9] IRCAM - Music Representation Team, «Score Following History in Video,» 12 1 2016. [Online]. Available: http://repmus.ircam.fr/antescofo/videos#scorefollowing-history-in-video.

[10] Dannenberg, R. B. (1985). *An On-Line Algorithm for Real-Time Accompaniment*.

[11] Vercoe, B. (1984). *The synthetic performer*.

[12] Puckette, M. (1990). *Explode, a user interface fopr sequencing and score following*. In ICMC '90 Proceedings.

[13] *A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Lawrence R. Rabiner, FELLOW, IEEE.

[14] Paper: "*Score-Performance Matching using HMMs*". Authors: Pedro Cano, Alex Loscos, Jordi Bonada. Audiovisual Institute, Pompeu Fabra University.

[15] *Real-Time Audio-to-Score Alignment of Music Performances Containing Errors and Arbitrary Repeats and Skips*. Tomohiko Nakamura, Eita Nakamura and Shigeki Sagayama. IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING, VOL. XX, NO. YY, 2015

[16] Paper: "*Automatic segmentation of acoustic musical signals using hidden Markov models*". IEEE.

[17] Orio, N. and Déchelle, F. (2001). *Score following using Spectral Analysis and Hidden Markov Models*.

[18] Dixon, S. (2005). *Live tracking of musical perfomances using on-line time warping*. In Conference on Digital Audio Effects.

[19] Paper: "*Polyphonic Audio Matching and Alignment for Music Retrieval*". Ning Hu, Roger B. Dannenberg and George Tzanetakis.

[20] Paper: "*Polyphonic Audio Matching and Alignment for Music Retrieval*". Ning Hu, Roger B. Dannenberg and George Tzanetakis.

[21] Paper: "*REAL-TIME AUDIO TO SCORE ALIGNMENT USING LOCALLY-CONSTRAINED DYNAMIC TIME WARPING OF CHROMAGRAMS*". Kosuke Suzuki, Yushi Ueda, Stanisław A. Raczynski, Nobutaka Ono, and Shigeki Sagayama.

[22] Paper: Carabias, J. J., Rodriguez, F. J., Vera, P., Caba, P., Ca, F. J., and Ruiz, N. (2012). *A real-time nmf-based score follower*. MIREX 2012.

[23] Francisco Jose Rodriguez-Serrano, Julio Jose Carabias-Orti, Pedro Vera-Candeas, and Damian Martinez-Munoz. 2016. *Tempo driven audio-to-score alignment using spectral decomposition and online dynamic time warping*. ACM Trans. Intell. Syst. Technol. 8, 2, Article 22 (October 2016), 20 pages. DOI: http://dx.doi.org/10.1145/2926717

[24] Ground-Truth Transcriptions of Real Music from Force-Aligned MIDI Syntheses. Robert J. Turetsky and Daniel P.W. Ellis LabROSA, Dept. of Electrical Engineering, Columbia University, New York NY 10027 USA

[25] *Soundprism: An Online System for Score-Informed Source Separation of Music Audio*. Zhiyao Duan and Bryan Pardo. IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, VOL. 5, NO. 6, OCTOBER 2011

[26] Munoz-Montoro et al. *Online/offline score informed music signal decomposition: application to minus one*. EURASIP Journal on Audio, Speech, and Music Processing (2019) 2019:23 https://doi.org/10.1186/s13636-019-0168-6

[27] Duan, Z. and Pardo, B. (2011). *Soundprism : An Online System for Score informed Source Separation of Music Audio*. IEEE Journal Of Selected Topics In Signal Processing.

[28] Dannenberg, R. B. (1985). *An On-Line Algorithm for Real-Time Accompaniment*. In Proceedings of the 1984 International Computer Music Conference.

[29] Cont, A. (2007). *Antescofo: anticipatory synchornization and control of interactive parameters in computer music*. Technical report.

[30] Woodruff, J., Pardo, B., and Dannenberg, R. (2006). *Remixing Stereo Music with Score-Informed Source Separation*. In Proceedings of the 7th international Conference on Music Information Retrieval.

[31] Fritsch, J. (2013). *Score informed audio source separation using constrained non negative matrix factorization and score synthesis*. In ICASSP 2013, Vancouver.

[32] Hu, N., Dannenberg, R. B., and Tzanetakis, G. (2003). *Polyphonic Audio Matching and Alignment for Music Retrieval*. IEEE Workshops on Applications of Signal Processing to Audio and Acoustics.

[33] Zhiyao Duan, B. Pardo, Speech Changshui Zhang. Audio, and Language Processing IEEE Transactions on. *Multiple Fundamental Frequency Estimation by Modeling Spectral Peaks and Non-Peak Regions*. Audio, Speech, and Language Processing, IEEE Transactions on, November 2010.

[34] Oriol Romaní Picas. Master *Thesis: A novel audio-to-score alignment method using velocity-driven DTW.* Department of Information and Communication Technologies Universitat Pompeu Fabra, Barcelona. 2014.

[35] MIDI Toolbox MatLab [online]: https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/projects2/past-projects/coe/materials/miditoolbox/index

[36] GUIDE and GUI MatLab Documentation [online]: https://es.mathworks.com/discovery/matlab-gui.html

[37] Information on the types of plot of the Mazurka Project, CHARM documentation [online]: http://www.mazurka.org.uk/ana/timescape/

[38] Information about CHARM [online]: https://www.charm.rhul.ac.uk/content/projects/chopin.html

[39] Information about URMP (Project developed by Rochester University) [online]: http://www2.ece.rochester.edu/projects/air/projects/URMP.html

[40] *University of Rochester Multi-modal Music Performance (URMP) Data*set Last Updated: February 2018. http://www2.ece.rochester.edu/projects/air/projects/URMP/URMP_doc.pdf

[41] B. Li, X. Liu, K. Dinesh, Z. Duan, G. Sharma, Creating a multitrack classical music performance dataset for multimodal music analysis: challenges, insights and applications. IEEE Trans. Multimed.

[42] RWC Music Database information. https://staff.aist.go.jp/m.goto/RWC-MDB/

[43] Score Following with Dynamic Time Warping - An Automatic Page-Turner. 2007. PhD thesis.

[44] Arshia Cont. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. International Computer Music Conference (ICMC), Aug 2008, Belfast, Ireland. pp.33-40.