Imperial College London

Imperial College of Science, Technology and Medicine

Department of Computing

BEng Individual Project Report

Implementing an Automatic Guitar Music Transcription Web Application

Pavlos Kosmetatos

Supervisor:
Dr. Iain Phillips

2016-2017

Abstract

Transcribing a piece of music is a very time consuming process. Traditionally musicians did it manually on paper, whereas nowadays most have started to use specialised software that works by "dragging-and-dropping" each note onto a music sheet. The task of automating the process of transcribing a musical piece is called Automatic Music Transcription (AMT) and has seen continuous development in the previous 5 years, with a number of papers focusing on finding optimal algorithms for individual stages of the process (pitch detection, onset detection, duration detection, etc.). However, only one application that combines those exists, and it has been optimised specifically for the piano (AnthemScore). In this paper, I present the State of the Art in AMT and my implementation of a monophonic (only one note at a time) guitar music transcription using autocorrelation (pitch detection), spectral flux (onset detection) and a duration detection based on successive onsets. Furthermore, I implement an extension of this transcription system to polyphonic (multiple notes at at time) audio using non-negative matrix factorisation. The main system and its extension are embedded in a web application written in Python/Flask and integrated with the Flat.io API which allows the rendering and user-editing of the music sheet after the transcription. Finally, the performance of the two systems is evaluated using the standard metrics in the field of AMT over a manually annotated guitar data set, while the main, monophonic system is also compared to AnthemScore, an implementation of a successful State of the Art method, showing a better overall accuracy on the monophonic guitar data set.

Acknowledgements

I would like to thank:

Dr. Iain Phillips, who provided valuable advice throughout the duration of this project and was always available when additional feedback was needed.

My family, friends and girlfriend, without the help of whom this project would have been much more difficult. I would like, however, to especially thank **my father**, as he not only showed genuine interest in my project but also actively helped me by providing me with guitar recordings and indispensable advice. For providing the recordings, apart from my father I would like to thank **Giorgos Alexandros Vallis** who provided me with a large subset of the whole data set and substantial musical advice.

Contents

1 Introduction

The task of composing a piece of music is extremely time consuming and involves a very inefficient two-stage development process: creating the piece and then transcribing it (converting it to music notation). Traditionally, the transcription of the piece was done manually on paper. However, this was quite inefficient, as editing a small part of the piece would require rewriting it. Also, sharing an editable version of the piece online would be impossible. Nowadays, most composers perform this task using specialised software (e.g. MuseScore [1]) that provide a music sheet creating/editing interface. The music sheets are exported in MusicXML, an XML-like format, and can then be printed, re-edited, and shared. Even though this specialised software is very appropriate for the editing of those files, transcribing a whole new piece from scratch is, again, very time consuming as it essentially requires dragging and dropping notes one-by-one to their correct place on the sheet. This paper is concerned with the automatic conversion of an audio signal (a recording) to music notation, known as **automatic music transcription** (AMT).

AMT falls into the broader field of **Music Information Retrieval** (MIR), the interdisciplinary science dealing with the extraction and analysis of information from music. MIR is a growing area of research in the past decade, presenting some similarities with speech recognition. Other applications of MIR include music recommendation systems, automatic genre classification and song identification (Shazam, SoundHound).

Automatic music transcription (AMT) is a relatively old concept, with the first approaches being developed in the 1970's. However, as a research field, it remained quite stagnant until the early 2000s. AMT can be divided into two main processes: **detecting the characteristics of the piece** (notes, durations, tempo, etc.) and then **converting this information to music notation**. The second process is relatively easy if you have the information needed. Therefore, all the research in AMT has been focused around the first process: musically characterizing the piece. Many research papers have attempted to find optimal algorithms for the subtasks that comprise that process; however, only one (AnthemScore) recently published application has combined them to create a complete transcription system.

[1] https://www.musescore.org

The automatic transcription of a music recording can be divided into several subtasks. Pitch detection, onset detection, and duration detection are the most essential for the development of an automatic transcription system, each one of which presents their own challenges. If one note is played at a time (**monophonic audio**), signal processing methods combined with some basic musical knowledge can be used to tackle all of those tasks. Single pitch detection, for example, is nowadays considered a solved task, with relatively simple algorithms like autocorrelation and YIN being the most widely known solutions. When many notes are played at a time, however, the problem becomes much more complex since musical instruments produce a variety of composite, imperfect sounds that cannot be easily distinguished from one another. Furthermore, these sounds are very different depending on the instrument; the perfect algorithm for detecting trumpet notes will work poorly on piano music. Even if such a perfect algorithm was implemented, it would not be sufficient for songs that contain notes from multiple instruments at a time. Attempts to tackle these problems require concepts from several disciplines such as **machine learning** and **statistics**. Interestingly, even the most recent research in AMT has not provided a solution more efficient than a trained musicians' ear.

This paper is comprised of three main parts: Initially we introduce the common challenges and **state of the art** in the automatic music transcription field while emphasising on pitch detection algorithms (see Section 3). Next, an implementation of a **monophonic guitar music transcription system** is presented (see Section 4.2): for **pitch and onset detection**, we discuss my own implementations of standard algorithms, variations of those and also methods provided by third party libraries. For **duration detection**, we only explain our implementation of a two-stage normalisation duration detector. These are evaluated using a 35-song annotated data set comprised of both studio and home recordings (see Section 5. The findings show the implementation of the autocorrelation method to be equally accurate to the YIN algorithm provided by an external library, whereas the third party onset detection method outperforms the one implemented. The duration detection method implemented produces excellent results on the rhythmically simple data set used. Finally, an attempt to extend the existing system to polyphonic audio using non-negative matrix factorisation is presented (see Section 4.3) and evaluated using a data set of 8 recordings (see Section 5.2). The challenges and limitations encountered are discussed in the end of this paper, which also includes future work

that could be done to improve the system (see 6).

The two aforementioned systems are embedded in a **web application** written in Python using the Flask framework. As described above, this kind of application would be mainly of interest to music composers who could use it to easily produce music notation for their compositions. Therefore, instead of simply providing the resulting MusicXML file, we integrate with the Flat.io API to render the music sheet in the browser and allow the user to make edits where appropriate.

2 Background

The area of automatic music transcription contains concepts from areas that have not been covered in the BEng Computing course at Imperial College London. This section will be focusing on introducing the reader to some concepts needed to follow the rest of the paper. The first part, Subsection 2.1 will cover some basic background in music notation and theory. Of course, these are complex topics so only a few important points needed to understand the rest of the paper are going to be covered. This will be followed by a briefing of the physics of the guitar and a comparison between the notions of frequency and pitch, described in 2.2. The rest of this section will focus on the representation of sound as a signal (2.3), and different ways we can process this signal (2.4, 2.5). Signals and signal processing techniques are going to be covered superficially in this report; for more detail about signals and systems refer to [1], whereas for signal processing techniques specifically for music refer to [2], [3] and [4].

2.1 Music Notation

Music notation is the process of music representation through the use of written symbols, so that it can be reproduced by others. There are two main ways of notating guitar music: **traditional sheet music** (also referred to as score) and **tablature**, which is specific to guitar music notation. Although sheet music is the main way in which Western classical music has been notated for centuries, tablature has proved more popular in recent years amongst guitar players, especially amateur ones.

Before we go into comparing those two methods, we have to introduce the most important idea behind music notation: the **note**. A note is a written symbol used to indicate two main attributes of a tone: **duration** and **pitch**. The duration of a note is intuitive: it is the time for which the note is played. On a piano, for example, this is the same as the duration for which the piano key is held. A note's pitch can be understood as how "high" or "low" it sounds. When pressing a key on a piano or plucking a string on a guitar, the sound generated oscillates at a certain frequency. Pitch is very closely related to that frequency, but is also affected by the

perception of the human listener, as discussed later. We order pitches according to the frequencies they produce. For example, the six strings of the guitar correspond to the frequencies 82Hz, 110Hz, 146Hz, 196Hz, 246Hz, 330Hz.

Furthermore, based on the ratio between pitches' fundamental frequencies, we can derive the notion of **pitch classes**. For example, if the ratio of two pitches is a power of two, then they sound very similar to each other, and we classify them in the same pitch class. Of course, to represent pitches we first have to reduce them to a finite number of symbols. In Western music, which is going to be the focus of this project, an octave is divided into 12 pitch classes, where each class differs from the next and previous one by a **semitone**. Seven of the pitch classes are represented by the letters from A to G. The remaining five pitch classes are represented by a letter and an accidental (either ♭ or ♯) and correspond to an increase of a semitone (in the case of the ♯) or decrease of a semitone (in the case of the ♭). We refer to those symbols as sharp (♯) or flat (♭).

This is better understood by looking at Figure 1 [2, p. 4]. Every key on the piano corresponds to a different note, which belongs to a pitch class and an octave. Modern pianos usually cover seven and a half octaves and therefore 87-88 notes in total. As the same pitch classes repeat in every octave, we have to add the **octave number** to distinguish between notes in the same class (e.g. E3 and E4). The octave number is incremented in C notes.

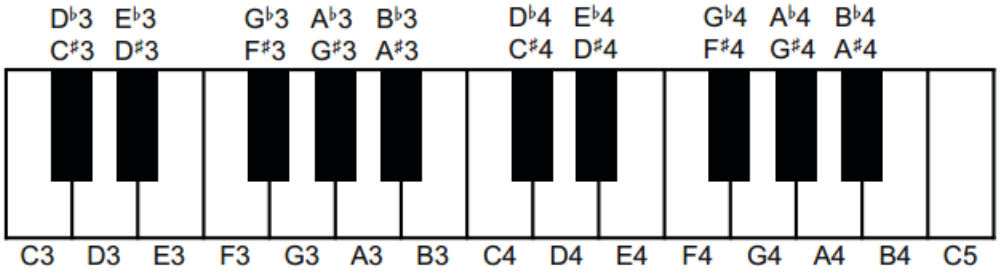


Figure 1: Pitches on a piano [2, p. 4].

Interestingly, the perceived distance in pitch between 220Hz and 440Hz is the same as the perceived distance between 440Hz and 880Hz. For that reason we can say that the human perception of pitch is **logarithmic**. The full relationship table between pitches and their fundamental frequencies is shown at Appendix A.

2.1.1 Score

Score notation is considerably harder to understand than tablature. as it provides much more information to the reader. Therefore we will dedicate this section to introduce some of its basic concepts. To show how pitch and rhythm are notated we first have to explain the structure on which everything is based: the **staff** (or stave). The staff is a set of five parallel, horizontal black lines. Notes are placed on different locations on the staff (either on the lines or the spaces between them) according to their pitch. The staff is divided into **measures** by vertical black lines.

2.1.1.1 Pitch

As explained in the previous section, pitch defines how "high" or "low" a note sounds. In traditional score notation, it is represented by placing notes higher or lower on the staff; the higher the location of the staff, the higher the pitch of the note. However, the height of the note is not enough to determine its pitch. One will also need to see the **clef**. The clef is a symbol in the beginning of the staff which assigns a reference note to each line. Figure 2 shows how the two most well-known clefs, the G (or **treble**) and the F (or **bass**) place pitches on different lines. When the pitch of a note does not "fit" inside a staff, it will be placed in "invisible" lines that extend above and below the staff (**ledger lines**). **Guitar music** specifically is always notated with the **treble clef** raised by a whole octave. Therefore C3 is instead of C4 the middle C we see in Figure 2.

2.1.1.2 Duration

Duration determines how "long" a note sounds for. Instead of measuring duration in seconds or minutes, we measure it with respect to a each other. This relation is shown in Figure 3.

A dot next to a note represents that the duration of that note will be multiplied by 1.5. For example, a half note with a dot next to it will not have a duration of 2 quarter notes but three quarter notes. Another set of symbols is used to indicate **rests** (times where no note is playing; pauses). As in notes, those symbols also have different representations according to their duration. The relationship between

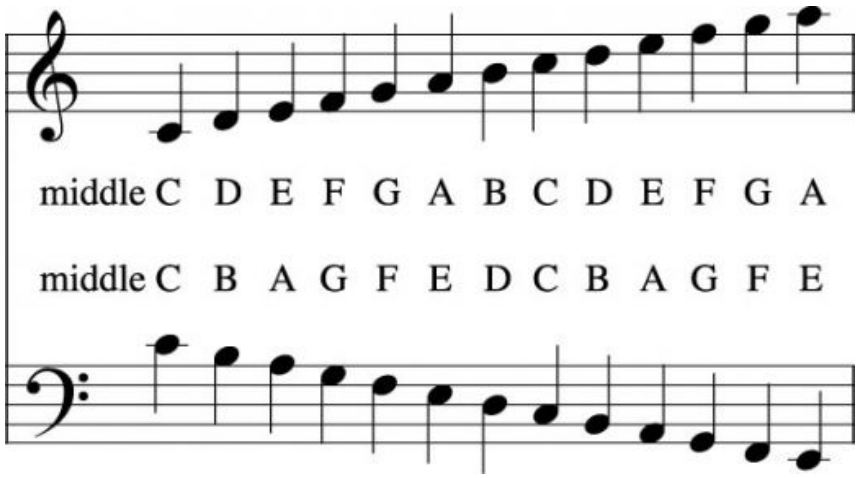


Figure 2: The G and F clefs. [5]

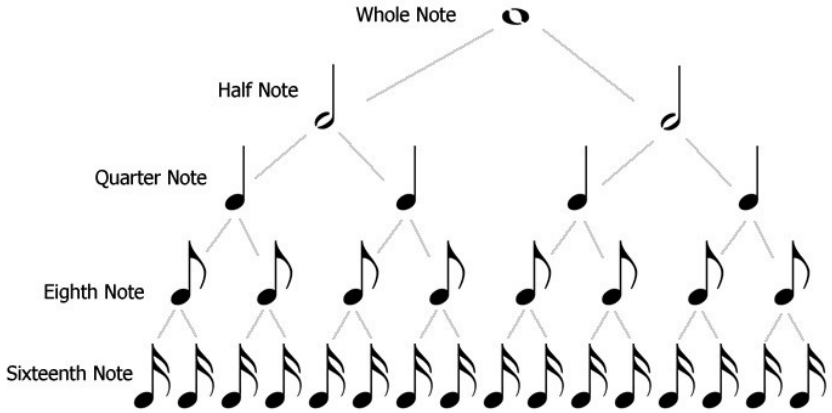


Figure 3: The duration relation between the notes [6].

durations in notes and in rests is shown in Figure 4. When two eighth notes are next to each they are usually combined to form **tuplets** (♫). For sixteenth notes, two lines instead of one connect the two notes.

2.1.1.3 Time and Key Signature

Two more conventions used in score notations are the **time** and **key signatures**. The time signature (or meter) of a piece specifies how many **beats** each measure contains and what is the note that corresponds to that beat. For example, a time signature of $\frac{3}{4}$ denotes that there are three beats in every measure, and that a beat

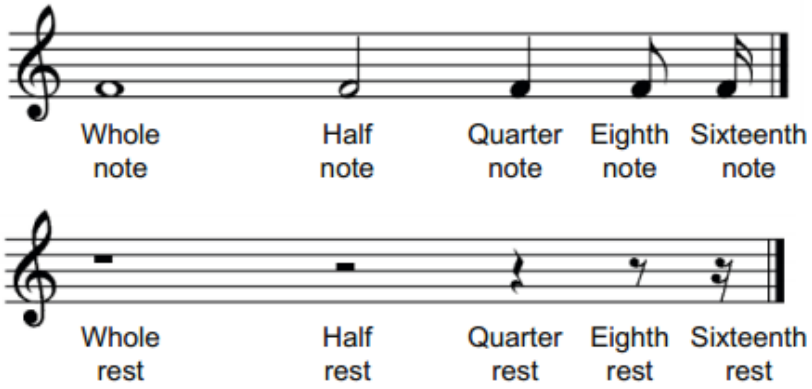


Figure 4: The relationship between rest and note durations [2, p. 7].

.

corresponds to a quarter. In Figure 7 we see a time signature of $\frac{4}{4}$. This can be validated by adding up the durations of the notes in the first measure: there are seven eighth notes and one eighth rest summing up to four beats.

The key signature of the piece is the set of sharps or flats that follow the clef. It shows which notes are going to be played in sharps or flats throughout the song. For example, 5 shows that every B, F and G note throughout the piece is going to be raised a semitone (played with a sharp). An exception of the key signature is denoted with a $\natural$ symbol next to a note.

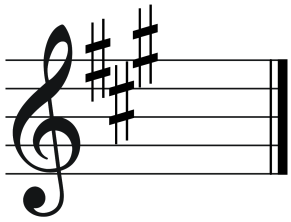


Figure 5: A key signature with sharps at B, F, G

.

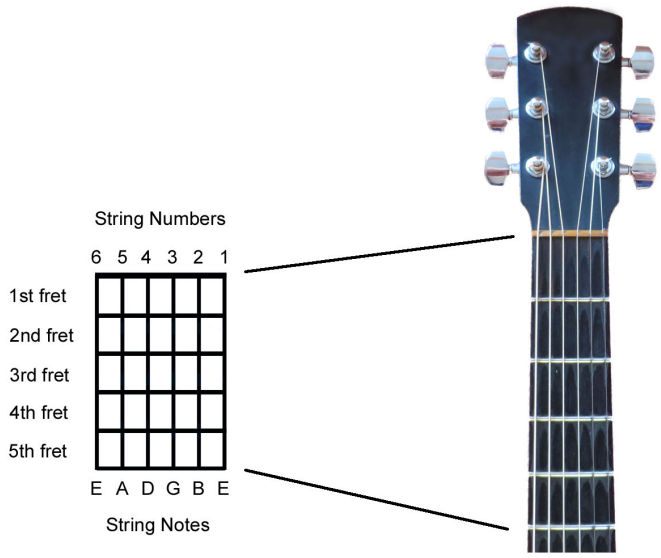


Figure 6: Relation between tablature and the guitar fretboard [7].

2.1.2 Tablature

Tablature is a notation system which represents the position of the notes when played on the guitar. The six lines correspond to the six strings of the guitar, and the number on top of those correspond to the fret the musician has to place their finger on before plucking the string. Figure 6 shows the relation between a guitar and tablature notation.

Now that some basic concepts in score notation have been covered, a comparison between score and tablature will be presented. In Figure 7, the differences between music sheet and tablature are visible. Music sheet is challenging to read and understand, and even experienced musicians need time to decipher it. The pitch of the sound played is perfectly described in both systems. However, if the musician has not heard the song before, he will struggle to reproduce it by only looking at tabs, as it lacks temporal information, like duration and timing for plucking the strings.

Tablature notation provides less information to the reader but it is much more intuitive and lightweight than the classic music sheet notation, making it easier for a student to understand. However, score notation is more suitable for the scope of this project whose main target group is composers. Both notations provide interesting and difficult challenges; tablature needs a way of assigning notes to locations on the

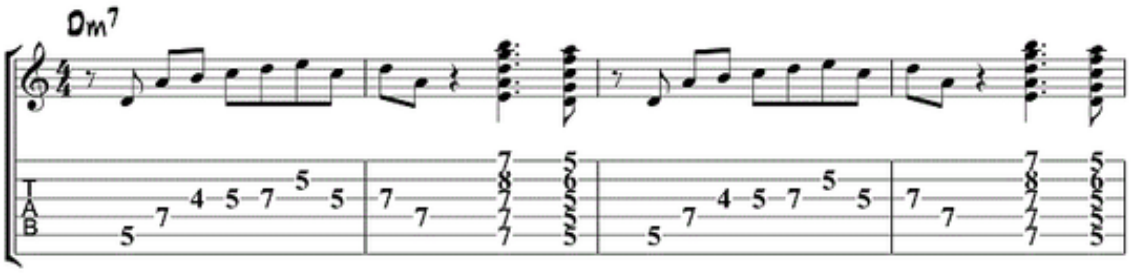


Figure 7: Music sheet notation in the top half, tablature in the bottom half [8].

fretboard (as the same note can be played in many different locations) and score notation will require heavier temporal analysis. This trade-off will be discussed in more detail in later sections.

2.2 Strings and waves

Before we go into the details of the task we are faced with, a short review of the physics perspective of the guitar must be covered. Most of the information included in this section can be found on any standard music physics textbook such as [9].

A guitar string is an elastic string fixed at both ends. The vibrations of such a string are called **standing waves** and they satisfy the relationships between wavelength-frequency and frequency-period that come from the definition of waves.

$$v = f\lambda$$

$$T = 1/f$$

Where v is the wave speed and f is the **frequency** of the wave (the number of oscillations per second) measured in Hz, λ is the **wavelength** and T the period (or inverse of the frequency). Frequency plays a big role in the **pitch** perceived by the listener (though is not the only factor). The **speed** of the wave on a string depends on the string tension and density. Waves travel faster on a tighter/lighter string and the frequency is therefore higher for a given wavelength. On the other hand, waves travel slower on a more loose/dense string and the frequency is therefore lower for a

given wavelength. The **wavelength** is directly related to the length of string, and as all the guitar strings have the same length, follows that all strings use the same range of wavelengths. That means that, to have a different pitch for every string, the strings must differ in tension or density, and for various reasons guitar producers have decided to produce them to differ in the latter.

When a string is plucked, a standing wave is generated by the composition of multiple waves moving in opposite directions along the string. This wave that mostly determines the pitch we hear is called the **fundamental**. The transcription of audio we will see in the following sections is based to a great extent on the estimation of this fundamental frequency.

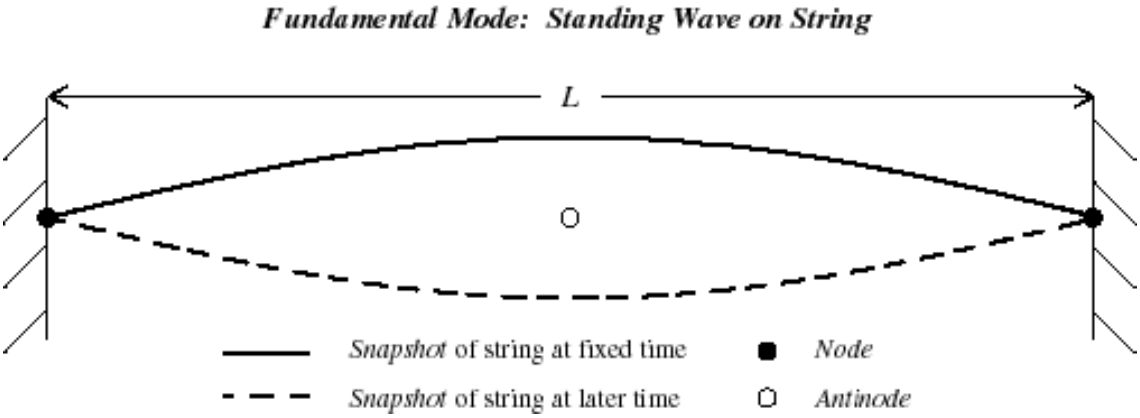


Figure 8: The fundamental and its node & antinode [10].

However, musical instruments do not only produce that fundamental wave. What makes the guitar sound unique compared to a, for example, lute or trumpet, is the variety of **overtones** it generates. The overtones are additional standing waves that are generated and have a larger frequency than the fundamental. The particular overtones that are natural multiples of the fundamental frequency are called **harmonics** (the fundamental frequency is, in this paper, **not** considered a harmonic; the first harmonic is twice the fundamental frequency). Those are the ones mostly generated when a string is plucked. For example, if a guitar player plucks the lowest string of his guitar, then 82Hz will be the fundamental frequency. There will be harmonics (whose amplitude decreases as they go higher in frequency) of 164Hz (2 * 82 Hz), 246 Hz (3 * 82 Hz), etc. When listening to a note, the harmonics are not distinguishable from the sound, but the combination of them is what defines the perceived pitch.

A concept we have not introduced so far is that of **inharmonicity**: the degree to which the frequencies of the overtones depart from whole multiples of the fundamental frequency. Even though some instruments (violin, cello, brass instruments) produce perfectly harmonic sounds, described by what we have stated so far, there are some that do not. Plucked string instruments like the guitar are considered **nearly harmonic**, meaning that they they produce harmonics close to integer multiples of the fundamental. This inharmonicity can be modeled by:

$$f_k = k f_0 \sqrt{1 + \lambda(k^2 - 1)}$$

where k is the index of the harmonic and λ is the inharmonicity factor [9].

Inharmonicity is one example of something that makes the guitar sound very different than, let's say, a trumpet. However, what makes it sound different from a piano, a lute or a traditional Greek "bouzouki" is **timbre** [2]. Timbre is quite intuitive to grasp but hard to define. The differences in timbre, however, between the guitar and other instruments, are huge. This can be seen by looking at the **envelope** of the tone: the outline of its extremes in amplitude. Specifically, the part of the envelope where the difference is greatest is the **attack phase** of the tone (or attack **transient**) where many non-periodic sounds are produced. Interestingly, after the attack transient fades out and only the harmonic content remains, the envelope of most instruments are very similar.

Furthermore, a characteristic of most instruments and something that poses a challenge when trying to detect pitch in audio recordings is that, sometimes, a fundamental frequency can be less strong in amplitude than some harmonic (or even completely missing), while the perceived pitch stays the same. Therefore, after directly relating the pitch to its fundamental frequency in Section 2.1, we now see why they are not the same concept. A pitch is a **combination** of vibrations in different frequencies, that may be related to the fundamental frequency at a big extend, but is also affected by **human perception**.

2.3 Sound as a signal

A signal is defined as a function that "conveys information about the behaviour or attributes of some phenomenon" [11]. As we described above, guitar sound is produced by the vibrating guitar strings. These vibrations cause oscillations of air molecules in the surrounding region. The alternating pressure travels through the air and reaches the listener's ear that in turn processes it and sends the appropriate nerve impulses to the brain. Therefore, sound fits perfectly in the definition of a signal. The sound signal associates a pressure value to every value of time (at a particular point in space). This can be graphically represented in a **pressure-time plot** (the **waveform** of the signal) (see Figure 9). Instead of representing raw pressure on the y-axis, we often use it to represent the deviation from the average air pressure (**amplitude**) instead.

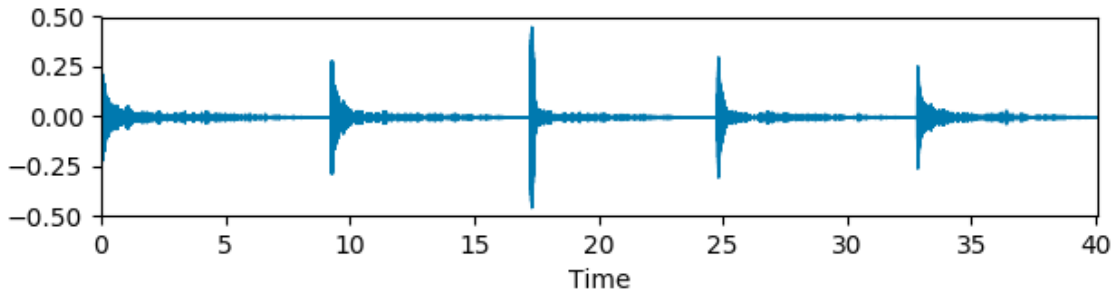


Figure 9: The waveform a simple guitar melody.

In Figure 9, the melody G3, G#3, A3, A#3, B3 played on the guitar is shown. We can immediately detect the five locations where there are large oscillations in air pressure. Those are the times when the guitar strings were plucked. Specifically, the times are at about 0, 9, 17, 25 and 33 seconds.

We can think of an audio signal as a representation of the sound just like sheet music and tablature, described in the previous section. Actually, the signal representation is the only representation that contains all information of a piece of music needed to reproduce it. As we can see in Figure 9, the audio representation does not explicitly give the characteristics of the piece; it is a "black box". The area of processing a signal and extracting information from it is called **signal processing**.

A signal $x(t)$ is said to be periodic if for some positive constant T_0

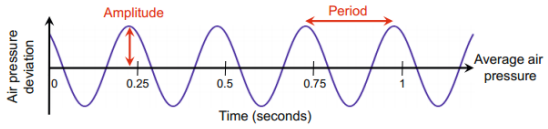


Figure 10: An example sinusoid [2, p. 21].

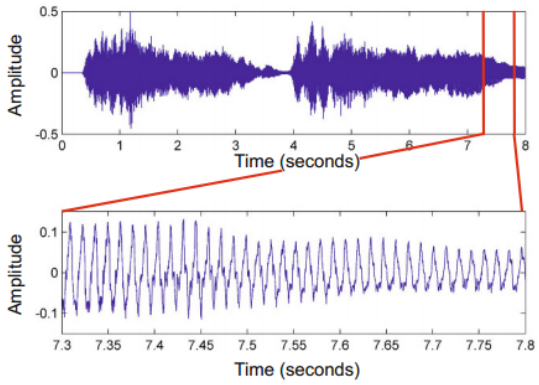


Figure 11: A waveform of a musical piece and an enlargement of the section between 7.3 and 7.8 seconds [2, p. 20].

$$x(t) = x(t + T_0) \quad \text{for all } t$$

This can be understood intuitively by looking at the waveform of the signal in Figure 10: the period is the distance between peaks or the time required to complete a cycle. The fundamental frequency of that signal is the reciprocal of the period $f_0 = 1/T_0$

This is not to be confused with the distance (in time) between peaks in Figure 9. Each of these peaks have their own period (and fundamental frequency) which is not visible without magnification. Figure 11 shows the enlargement of a particular section of a signal from where the periodicity can be seen [2, p. 20].

The waveform shown in Figure 10 is a specific type of periodic waveform which is of great importance in signal processing: a **sinusoid**. Sinusoids are elementary signals that can be described by the general equation $x(t) = sin(\omega t + \phi)$ where ω is the frequency and ϕ is the phase (which shows how much "lag" a sinusoid has compared to another starting at time zero with a value zero).

We are now ready to connect the concepts explained in Section 2.1 and 2.2. We have already defined the **pitch** in relation to its fundamental frequency. However, in Section 2.2, we said that the guitar and any other instrument, also produces other waves called harmonics. Each of these harmonics can be represented by a **sinusoid**. Therefore, we say a pitch is a **superposition** of these sinusoids.

Analog and Digital Signals

So far, we have described audio signals as continuous functions. However, in the digital domain, to process this signal, we have to **discretise** it in both amplitude and time. Discretising amplitude is referred to as **quantization** and discretising time is referred to as **sampling**. Both of those tasks are done by an **analog-to-digital converter** (ADC): quantization replaces each real value in the amplitude-axis with an approximation from a finite set of values and sampling is done by only considering signal values at specific times. The **sampling period** T_S is the distance in seconds between sampling points. The sampling rate is its reciprocal: $f_S = \frac{1}{T_S}$.

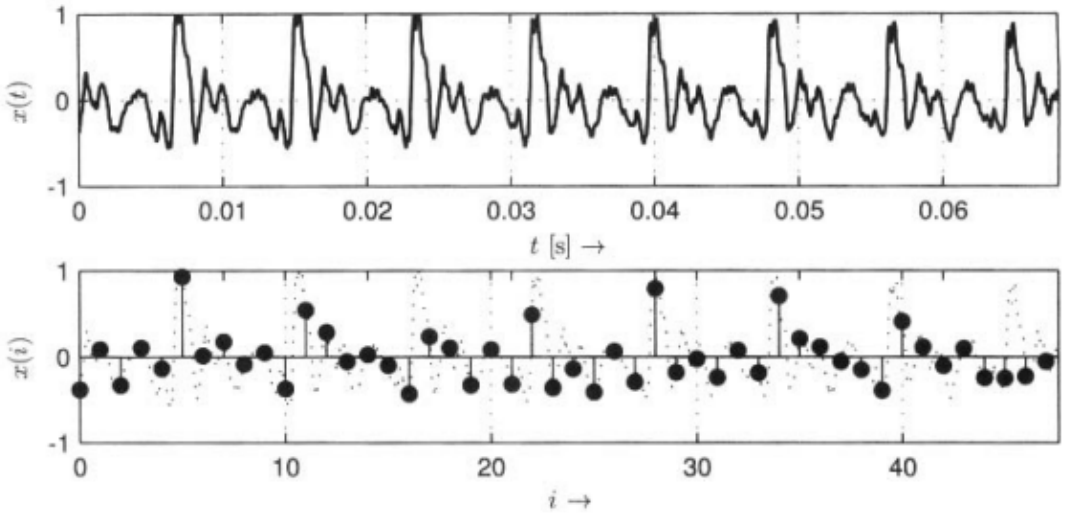
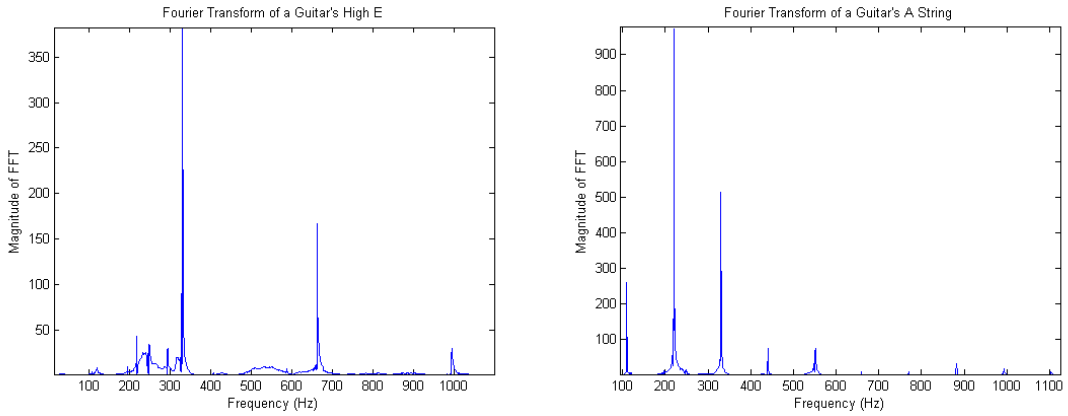


Figure 12: The analog signal (top) and its sampled version at a sampling rate of $f_S = 700Hz$ [4, p. 10].

2.4 The Fourier Transform

As stated above, sounds produced by a musical instrument do not only contain a single frequency. Therefore, a sinusoid with frequency 440Hz does not sound like the A4 (440Hz) note played on the guitar. The reason for that is the addition of **harmonics** (or partials) in the sound. In reality, the A4 note of the guitar will contain the sinusoid of frequency 440Hz but also the sinusoid of frequency 880Hz, 1320Hz, etc.



(a) E4 guitar note.

(b) A2 guitar note.

Figure 13: Two different guitar spectra [12].

The goal of the **Fourier transform** is to separate the elementary sinusoids that comprise the final sound. The output, usually called the **frequency spectrum**, shows how much each such component contributes to the overall mixture. In Figure 13a, we can see that the biggest peak in the spectrum is near the 320Hz mark. However, that is not enough to conclude that 320Hz is the fundamental frequency of the pitch.

As we have stated before, in some cases, a harmonic can be stronger in terms of magnitude than the fundamental frequency. We also know that a harmonic has to be a natural multiple of the fundamental. In Figure 13a, it is clear that the biggest peak is also the peak with the lowest frequency. Thus, we can conclude that the note played indeed has a fundamental of 320Hz. The second harmonic of that note is also clearly visible at about 650Hz.

In contrast, in Figure 13b we see a case where the strongest peak does not correspond to the fundamental frequency. Here, the strongest peak is clearly the one at about 220Hz, but the note played to generate this spectrum was an A2 note (110Hz). Therefore this is a good example of the first harmonic being stronger in magnitude than the fundamental frequency. When implementing a pitch detector later, we have to find ways to distinguish between a strong harmonic and a fundamental frequency.

During this report we are often going to refer to the **Fast Fourier Transform** (FFT) which is the algorithm that computes the Fourier transform of a digital

signal.

2.5 The Short-Time Fourier Transform (STFT)

A spectrum corresponds to a short snapshot in time and therefore does not provide any information about the time evolution of different frequencies. For that reason, short-time spectra can be linearly combined to produce a **spectrogram**. The most common format of that is with the horizontal axis representing time, the vertical axis representing frequency and a third dimension (usually colour) representing amplitude (loudness).

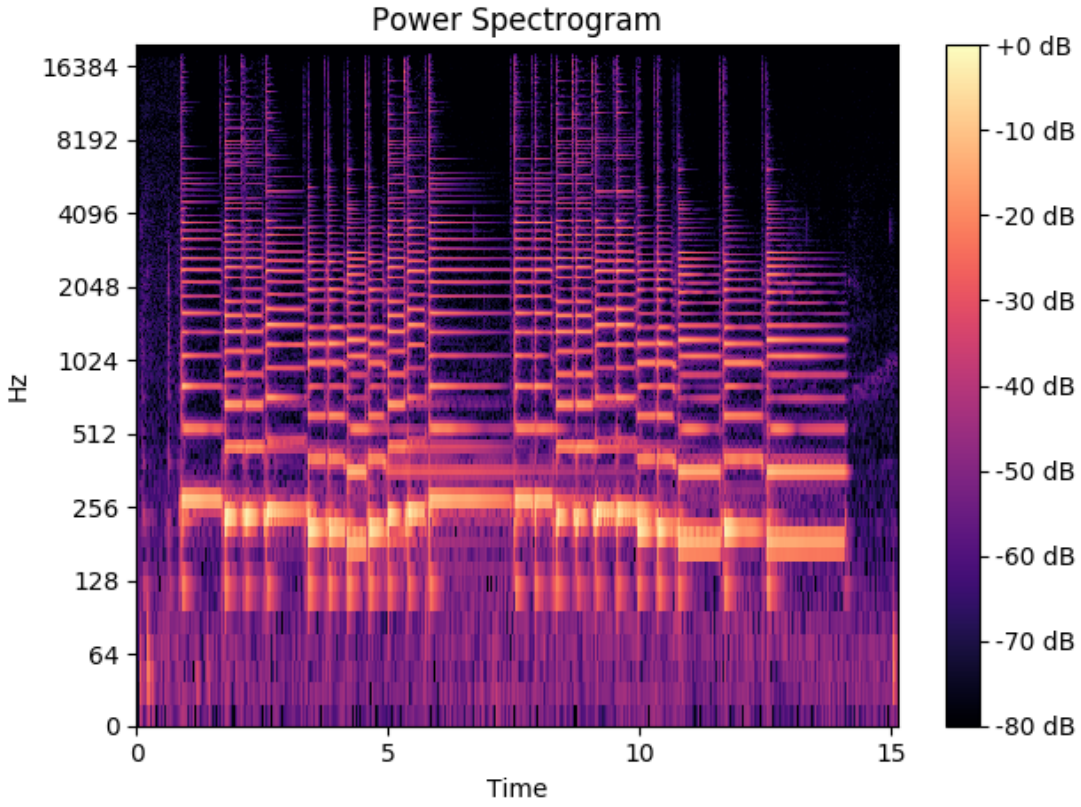


Figure 14: The spectrogram of guitar melody.

In Figure 14 we see the spectrogram of the "Soft Kitty, Warm Kitty" melody played on the guitar. Looking at the brightest places in each time frame will usually (as sometimes the harmonic could be higher in magnitude than the fundamental,

therefore brighter in the STFT) give us the notes played. The value of the STFT is immediately seen: on a spectrum we would not have the opportunity to see what frequencies were strongest at different points in time.

We can notice from the spectrogram that apart from the fundamental and the harmonics of the notes played, there is also brightness in other, random places in the spectrogram. This brightness is what we call **noise** and it usually gathers in low frequencies. In the next chapters, when analysing the spectrogram, we will see how we can use a **filter** to discard this noise and only keep the valuable information.

3 State of the Art

Automatic transcription of music is defined as the process of analysing an acoustic music signal and writing down the music parameters of the sounds that occur in it, either in the form of traditional score or tablature. In simpler words, we can consider automatic transcription of music as a task of converting a piece of music to notation that musicians can understand and reproduce. A complete music transcription system must be able to detect when each note is played, the pitch of that note, and for how long it is played. These three domains are called respectively onset detection, pitch detection and offset or duration detection. There are also other, more specialised research areas, such as tempo detection (Cemgil, A. & Kappen, B [13]), chord detection (Lee, K. & Slaney, M [14]), fingering assignment (Barbancho et al. [15]) etc. For the scope of this project, and therefore for this State of the Art review, the focus is going to be on pitch detection.

Single pitch detection (pitch detection in **monophonic music**) is considered a partially solved task, with several reliable solutions, the most popular being the **YIN algorithm** [16] explained in Section 3.1.2. One application where monophonic pitch detection has been successfully implemented is guitar tuners - tools that determine the pitch generated and advise the guitar player on how "far" it is from the correct pitch. However, one must keep in mind that even though guitar tuners detect pitch very efficiently, they are not designed to find pitch in signals where pitch is constantly changing, like a song where the guitar plays a melody. That task would make a system considerably more challenging to implement. On the other hand, pitch detection in **polyphonic** music (also referred to as multi-pitch detection or multi-F0 detection) has been the subject of increasing research interest since about 1980 [17], but with no -widely accepted as optimal- solution, as none of the proposed solutions perform even nearly as well as a trained musician's ear.

3.1 Single Pitch detection

As stated before, monophonic audio is audio comprised of only one sound at a time. Single pitch detection methods in literature ([18], [19]) are generally divided into two main areas: **time domain** methods and **frequency domain** methods.

Most of those methods are regarded as a prerequisite and get used in multiple pitch detection methods. We will go through the most widely known ones: the **zero-crossings** method, **autocorrelation**, the **HPS**, the **FFT based pitch detection** and some of their variants.

3.1.1 Common challenges

The following challenges are very common amongst all types of monophonic pitch detectors. During the implementation of my web application, I encountered a version of all of them. Therefore, solutions to these common challenges are going to be described in later sections.

Weak fundamentals: We have already stated that a fundamental frequency can be weaker than a harmonic frequency. That is a common issue in pitch detectors, that could erroneously pick a multiple of the actual pitch.

Attack transients: One important issue faced by pitch detectors is that of attack transients. At the moment a guitar produces a note, there is a short duration when the fundamental frequency is obscured by high amplitude noise and inharmonic partials. An efficient pitch detector must ignore those attack transients.

Noise: What is crucial for a pitch detector is to "be accurate, but not too accurate" [18]. External noise always make the task of detecting pitch harder. The pitch detector has to be able to distinguish noise from sounds it must analyse.

Low guitar pitches and high-numbered frets: The inharmonicity produced by the guitar gets increased when the pitch is either lower (for example on the 6th string), or we fingered a high-numbered fret. This inharmonicity can be an issue when trying to detect the pitch of a sound.

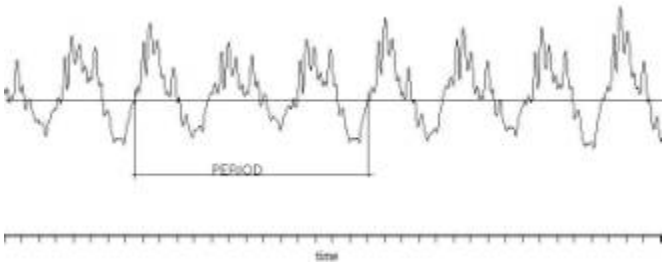


Figure 15: The zero-crossings method [20].

3.1.2 Time-Domain Methods

The methods described here look at the input signal as fluctuating amplitude in the time domain.

Zero-Crossings:

Zero-Crossings is the simplest technique to detect the fundamental frequency of a signal in the time domain. As every other technique we will come across, there is a trade-off between simplicity and precision. The zero crossings technique finds the period of a signal by counting the number of times the signal crosses the time axis and is followed by a rising slope. It is a very intuitive and inexpensive method, but it is very inaccurate when dealing with non-perfect signals, meaning signals that contain noise or harmonic signals where the partials are stronger than the fundamental, as those signals will contain multiple time axis crossings.

Autocorrelation:

Correlation functions are used to measure the "similarity" between two signals, by comparing them on a point-by-point basis and returning themselves a signal. A good way to think of the correlation function is the following: if the two signals "meet" at a certain point, the correlation function is 1 at that point. If not, then the function is 0 at that point. The **autocorrelation** method compares a signal with a version of itself delayed by some intervals. The point of comparing a signal with delayed versions of itself is to find patterns that repeat - and from there to calculate the **period** of the signal that will lead us to the fundamental frequency.

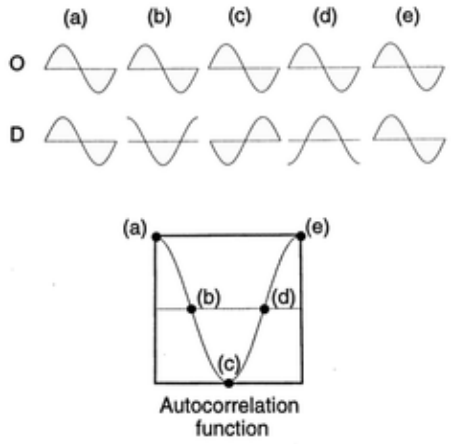


Figure 16: The autocorrelation function is itself a sinusoid wave [18].

Pitch detectors implemented with the autocorrelation function store a snapshot of the input signal in a buffer. As the signal comes in, the PD tries to match the stored snapshot with the incoming waveform. The autocorrelation function is defined by:

$$r'_t(\tau) = \sum_{j=t+1}^{t+W} x_j x_{j+\tau}$$

Pitch detectors using the autocorrelation function usually search for repeated peaks in the resulting waveform. From their distance (usually in measured in milliseconds) the period and thus the fundamental frequency are easily derived.

This pitch detection method is most efficient at mid to low frequencies. Thus it has been popular in speech recognition applications where the pitch range is limited. However, in musical applications, where the pitch range is broader, direct calculation of the autocorrelation is computationally heavy and not perfectly efficient.

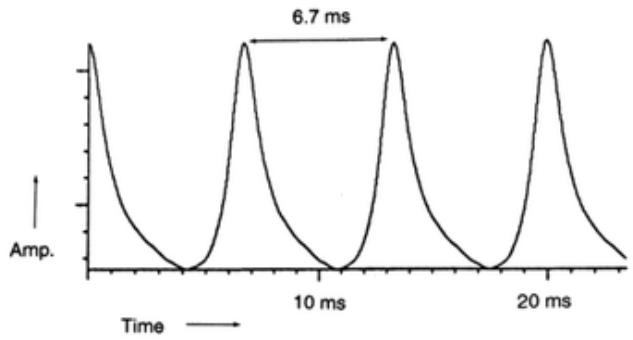


Figure 17: Autocorrelation of a signal with four additional harmonics to the fundamental. The period is 6.7ms and therefore the calculated fundamental is 149 Hz (close to D3) [18].

The YIN Algorithm:

The YIN algorithm [16] proposed by Alain de Cheveigné and Hideki Kawahara is based on the autocorrelation method explained above. It proposes five modifications to the traditional autocorrelation function and results in a decrease from 10% to 0.5% gross error on the data set used.

1. **Difference function**: Instead of searching for the peaks of the autocorrelation function, we search for dips of the squared difference function:

$$d_t(\tau) = \sum_{j=1}^{W}(x_j - x_{j+\tau})^2)$$

2. **Cumulative mean normalised difference function**: Dividing each value of the old function by its average over short-lag values. This avoids the selection of dips near zero lag.

3. **Absolute threshold**: Instead of picking the lag τ of the lowest dip, an absolute threshold is set and the first τ that gives a value less than that threshold is selected. This minimizes the issue of octave errors.

4. **Parabolic interpolation**: Fit parabola over each dip and its neighbours to get more precise estimation of the real minimum.

5. **Best local estimate**: Heuristic to choose the best local estimate around each point.

As this method is relatively simple compared to the polyphonic audio transcription, it will be implemented as the first transcription milestone of the project.

3.1.3 Frequency-Domain Methods

Frequency-Domain methods operate on the spectrum of the signal. Therefore, their first step is to use the Fourier transform to dissect the signal into the elementary sinusoids that it consists of. Instead of looking at the periodicity of the signal, like the time-domain methods, they look at patterns in the frequency peaks: the strongest peak (in amplitude) has a serious chance of being the fundamental. The remaining high peaks usually are the harmonics of the note. Frequency-domain methods take that information into account and determine the pitch of the sound.

An issue with the frequency domain methods is that the FFT does not tell us exactly the frequency of each sinusoidal. As explained in Section 2.5, each sinusoidal will be assigned to a **bin**, which will usually span a length of 10-100Hz. Therefore, we immediately encounter an precision issue with frequency-domain methods.

Spectrum Peak Picking:

The most intuitive and simple method in the frequency-domain is to pick the strongest peak from the spectrum. However, this solution proves quite inaccurate. As explained in Section 3.1.1, the fundamental frequency of a note does not have to be the strongest frequency. In some cases (not in the guitar) we even see the fundamental missing from the spectrum.

The Harmonic Product Spectrum:

The Harmonic Product Spectrum is a much more precise and efficient version of the brute-force Spectrum Peak Picking, as it takes into account the harmonic content in a frame. The HPS, proposed by Noll (1969) [21], finds the frequencies of the harmonic peaks of the spectrum and then computes the greatest common divisor of the

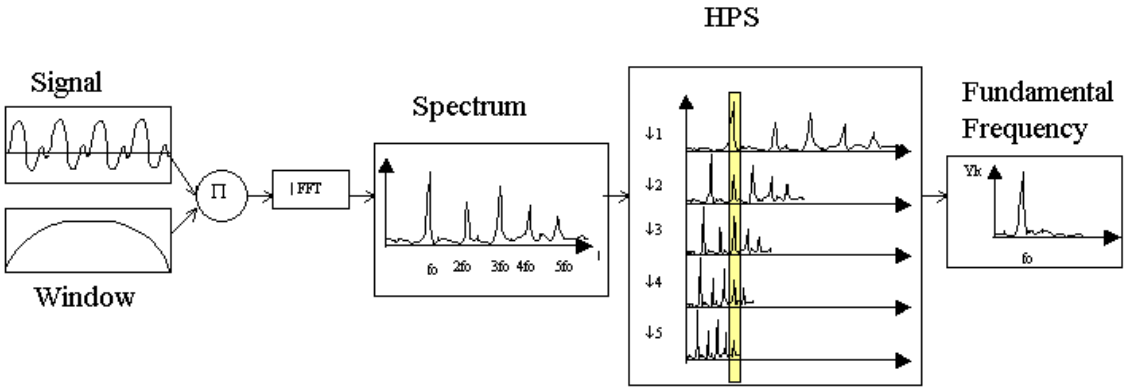


Figure 18: The HPS method [20].

those peaks. This is intuitively understood by looking at Figure 18. The following equation measures, for a spectrum X, the maximum coincidence of harmonics ωr. Noll defines R as the number of harmonics to be considered and ω_i a frequency in the range of all possible fundamentals.

$$Y(\omega) = \prod_{r=1}^{R} |X(\omega r)|$$

To get our fundamental frequency, we simply extract the maximum value from our resulting array $Y(\omega)$.

3.2 Multiple Pitch Detection

When faced with the task of real music transcription, the aforementioned algorithms are insufficient. Figure 19 shows the inability of the autocorrelation function to detect the correct period when given a polyphonic signal [22, p. 19]. In general, every method mentioned so far is suitable for the task of recognising a single note played, but a song will always have multiple notes at once, and often many instruments playing at the same time. Therefore, a new task of finding a reliable, **multiple-pitch** detection algorithm arises. We define a **multi-pitch detection** algorithm one that, given a time frame, can detect all pitches in that frame. The task of finding the onset and offset times of every note is often separated, and the combination of all those tasks is called **note tracking**.

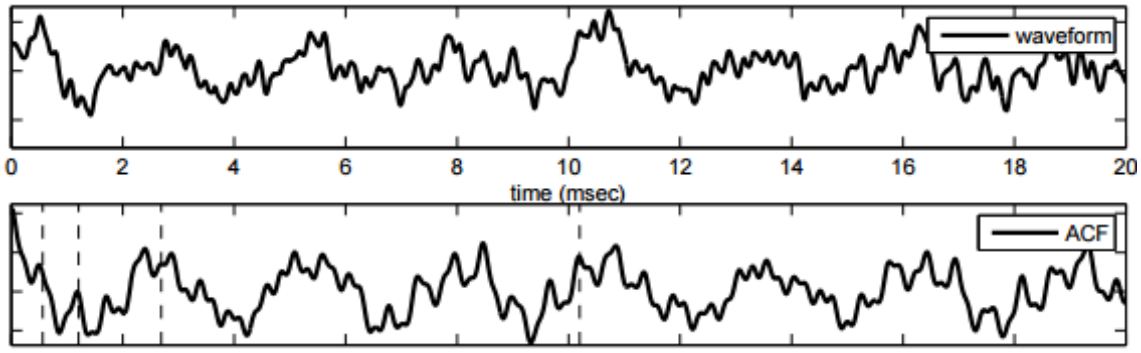


Figure 19: A polyphonic signal with four sources (up) and its autocorrelation function (down). The correct periods are marked by the vertical lines [22, p.19]

The **Music Information Retrieval Evaluation Exchange** (MIREX) is an annual event where state-of-the-art algorithms concerning Music Information Retrieval are evaluated and compared. Methods in particular are evaluated separately on multi-pitch detection and note tracking.

Even though research in Automatic Music Transcription has progressed significantly in the past decade, applications that can do this task sufficiently well only came out very recently. Lunaverus [23], published in 2016, is the most complete project done so far, but without the ability to separate sounds from multiple instruments. It has also been less successful in instruments other than the piano.

3.2.1 Common challenges

We have already discussed the common challenges in single pitch detection which also exist in multiple pitch detection. However, just by seeing the spectrograms of a monophonic and a polyphonic signal, one can directly see there is a difference in the complexity of the problem. Specifically, there are two main complex issues in the polyphonic version which we are going to discuss in this subsection.

3.2.1.1 Overlapping harmonics

An important issue one needs to tackle when implementing a multiple-pitch detector is **overlapping harmonics**. As we have said in previous sections, a pitch is

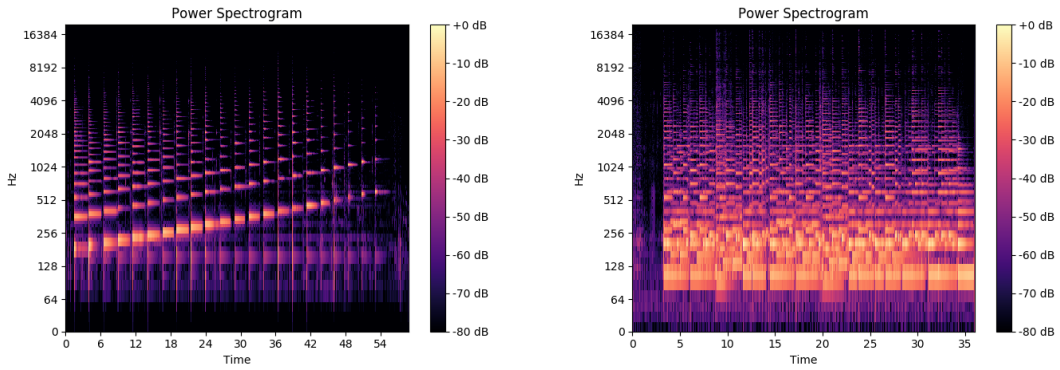


Figure 20: A monophonic (left) and a polyphonic (right) spectrogram.

not only its fundamental f but also a series of harmonic sounds κf where k is a natural number. In polyphonic signals however, two notes could have some identical harmonics. What makes this issue even worse is that, often, in musical pieces, notes that are combined together are harmonically related; meaning that they form particular ratios which results in some of their harmonics to overlap. A worst-case scenario is that in which we are analysing a combination of two notes that differ by an octave; such as an E2 and an E3 note. Then, all of E3 harmonics will also be E2's harmonics. Therefore we could wrongly assume that the only note that is sounding is E2.

3.2.1.2 Diversity in musical instruments

Musical instruments differ greatly in the sound produced. That makes modelling the characteristics of the expected sound much harder. We have already mentioned **inharmonicity** that occurs in several string instruments. Another big difference lies in the amplitude of the harmonics compared to the fundamental. For example, the piano often has weak fundamentals. This issue is something we will not be looking in much detail as this project is only about guitar audio pitch detection.

3.2.2 Methods

This section will primarily focus on covering the State of the Art in multi-pitch detection methods, without paying too much attention on note tracking. As the

proposed methods are many and vary greatly in concepts involved, we will classify them and discuss only some classes that could be implemented later.

Yeh (2008) [22], classifies multi-pitch estimation (detection) methods as **iterative** or **joint**. Iterative approaches take a frame and apply pitch estimation iteratively. At every iteration we detect the most dominant pitch and **extract** it (cancel it) from the signal. We continue until no other pitch can be found. On the other hand, joint detection approaches evaluate possible combinations of pitches as a whole, without any cancellation. Both classes of methods present their own challenges and very different techniques have been developed to tackle them.

Iterative Estimation Methods

Iterative estimation methods rely on two main steps; firstly, detect the most dominant pitch at each time frame. This is usually done similarly to monophonic approaches (see Section 3.1). Secondly, we remove the pitch from that frame, to be able to extract the next most dominant pitch. This is the core of iterative multi-pitch detection methods. According to Yeh, there are two main techniques that have been developed to "cancel" a predominant pitch from a signal.

Direct cancellation completely removes **all** harmonics of the found pitch from the signal. Of course, this is a quite naive approach. As we have discussed earlier, a major difficulty in multi-pitch detection is that of overlapping harmonics, which is not taken care of in direct cancellation. That is why direct cancellation methods have not been developed recently, and will not be discussed further in this report.

The second cancellation technique in iterative detection methods is **cancellation by spectral models**. Klapuri (2003, 2006) [24][25] has proposed a number of estimation algorithms based on using spectral features to apply iterative cancellation. Specifically, in [24], he proposes new methods in both stages of the iteration. In the estimation stage, he utilizes the harmonic relationships between frequency components without assuming ideal harmonicity. An overview of his method is seen in Figure 21. In [25], he evaluates F0 candidates by representing their strength as a weighted sum of the amplitudes of their harmonic partials.

Iterative estimation methods rely greatly on the accuracy of the F0 estimation method. That is because if an error happens at that stage, then a wrong pitch

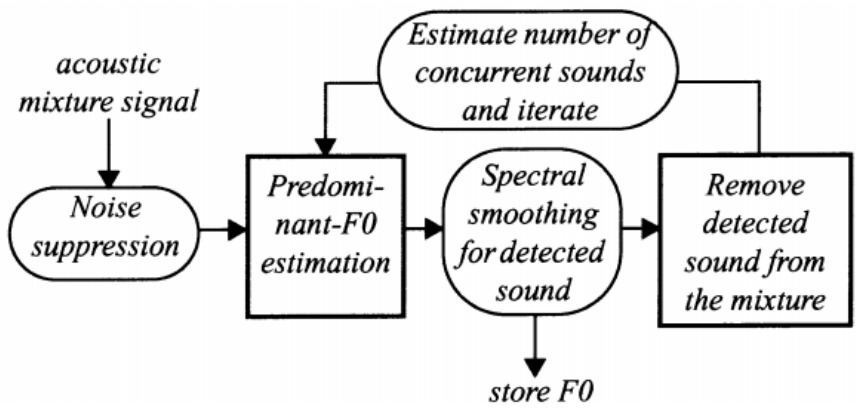


Figure 21: The overview of Klapuri (2003) [24].

will be extracted from the signal, "corrupting" the next iterations.

Joint Estimation Methods

Joint estimation methods do not act iteratively as those described above. On the contrary, they look at each frame as a whole, and evaluate possible combinations of multiple pitch hypotheses. This solves the biggest issue of the iterative methods which is additive error in each iteration. Of course, as many hypotheses have to be evaluated, these methods are usually more computationally expensive.

However, by looking at recent submissions in MIREX, one can quickly understand that this is not a very good classifier anymore. Since 2007-2008, research has been focused on joint estimation methods. Table 1 shows recent MIREX submissions that have scored well on the MIREX data set. All of them are joint estimation methods.

As joint estimation methods have been developed significantly from the time of Yeh (2008), we will instead look at how Benetos et al. (2013) [27] divided these methods in **feature-based methods** (which also contain some iterative methods we will not cover here), **statistical model-based** methods and **spectrogram factorisation-based** methods. We also add one more category of methods which gathered a lot of attention recently, **classification-based methods**.

	Author(s)	PR	RE
2016	M. Marolt	0.77	0.58
2015	E. Benetos, T. Weyde	0.75	0.52
2014	A. Elowsson, A. Friberg	0.86	0.77
2013	E. Benetos, T. Weyde	0.77	0.73
2012	K. Dressler	0.85	0.67
2011	C. Yeh, Axel Roebel	0.73	0.84

Table 1: Top MIREX submissions since 2011, based on precision & recall [26]

3.2.2.1 Joint feature-based methods

Feature-based methods are methods that use audio features to derive multiple pitches. The iterative methods described above were all feature-based methods. However, there are also a number of joint feature-based methods. A very well-known one is the method by Yeh (2008) [22], who defined a pitch candidate set consisting of harmonicity, mean bandwidth, spectral centroid and "synchronicity" and then based his joint estimation method on a pitch candidate set **score function**. This method was the top method in terms of accuracy in the MIREX 2009 multi pitch detection and note tracking tasks. Another method employing signal processing features in a joint estimation fashion is the work by Dressler (2012), which was also a top method in MIREX 2012 (shown in Table 1).

3.2.2.2 Spectrogram factorisation-based methods

Spectrogram factorisation-based methods have been very popular recently. Those are based on non-negative matrix factorisation (NMF), a mathematical technique that seeks to **factorise** an input matrix **X** into two matrices **W** and **H**, given that all three matrices have no negative elements. Of course, this fits the description of audio spectrograms we have seen so far. Applying a magnitude spectrogram to NMF was first proposed by Smaragdis (2003) [28]. In a monophonic context, Smaragdis realized that when given the spectrogram of a musical piece, the resulting matrices provide a lot of information. Specifically, the rows of **H** correspond to the temporal activity of the notes, whereas the columns of **W** correspond to their frequency spectra. An example of a simple musical piece and the result matrices are shown in

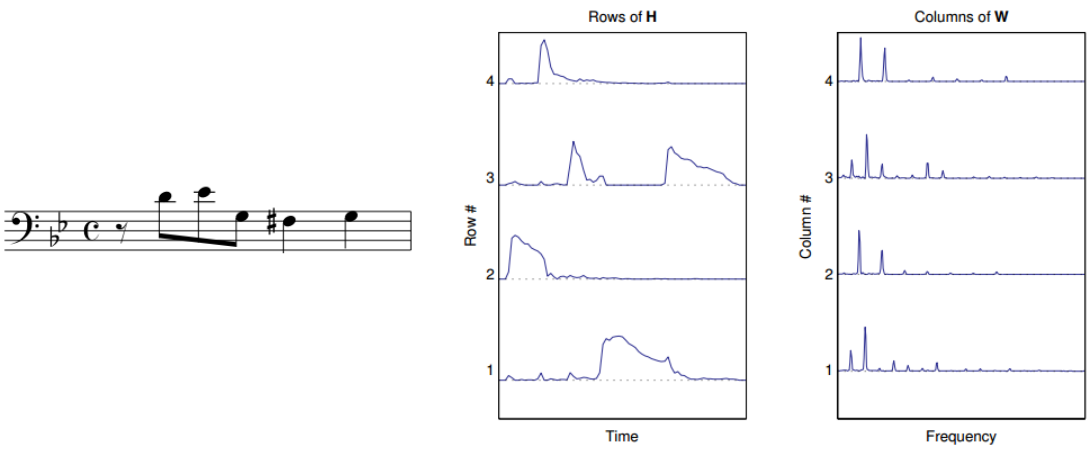


Figure 22: The musical piece and the result of its spectrogram's NMF [28].

Figure 22. From the result matrices, we can estimate the pitch based on methods we have discussed in previous sections.

The result of the NMF is an approximation of the initial matrix $\mathbf{X}$. Specifically, the two non-negative matrices $\mathbf{W} \in \mathbb{R}^{\geq 0, M \times R}$ and $\mathbf{H} \in \mathbb{R}^{\geq 0, R \times N}$ have to be chosen to minimize the **error of reconstruction** computed by a cost function. Their size depends on a parameter R. In Figure 22, the R parameter was equal to 4, which was convenient as it was equal to the number of unique notes, and therefore each note could be represented with one row of $\mathbf{H}$ and column of $\mathbf{W}$.

If the chosen R was less than the number of notes, then the analysis will be incomplete; as some rows/columns will combine note events. Therefore we do not have a choice but to choose a large R. If the resulting R is greater than the notes in the piece, any extra components of the resulting matrices will be clearly distinguishable as seen in Figure 23.

Proceeding with polyphonic examples, Smaragdis noticed that one new issue appeared. Figure 24 shows an example of a musical piece that contains a polyphonic component ($G3$ and $B\flat 4$ played at the same time) and their NMF (with $R = 4$). The new issue is immediately visible; instead of 7 note components, we have 6 note components and one non-note component (column 3). The reason for this is that NMF extracts **events**, not unique notes. Neither $G3$ nor $B\flat 4$ had appeared before in the musical piece. If they had, then they would have been considered isolated notes, and their combination would have been correctly detected as a combination

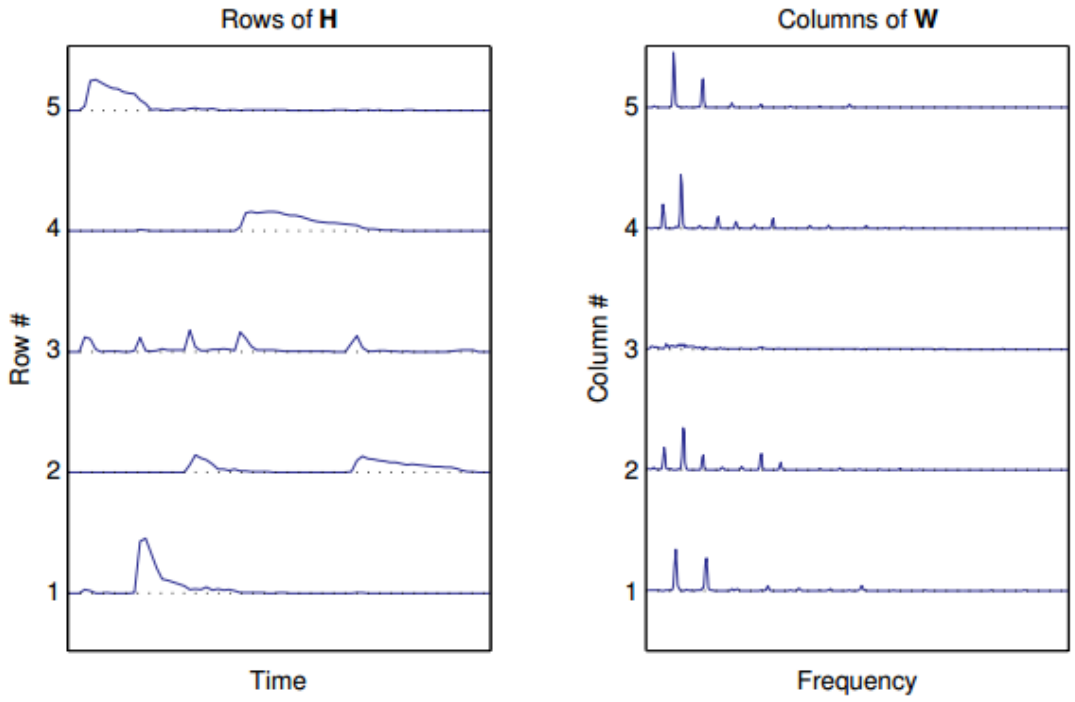


Figure 23: NMF resulting matrices when R is greater than notes. Row/Column 3 is the non-note component [28].

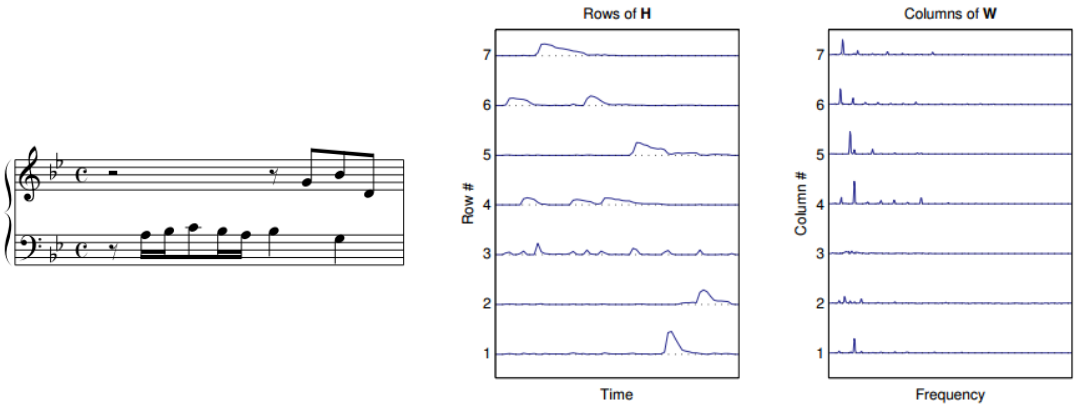


Figure 24: A polyphonic musical piece and the result of its spectrogram's NMF [28].

of two components. In general, NMF needs enough data so that all the notes that are played in combinations, are also exposed individually.

Since the initial proposal of NMF for multi-pitch detection, several variations have been developed. Duan & Benetos (2015) [29] classify these variations as **fixed template** and **adaptive template** methods.

Proposals based on a **fixed basis of pitch templates** (the components of the **W** matrix) are particularly useful when working with specific instruments, where you can explicitly define the set of pitches you expect. This helps as it firstly avoids mistakes such as two notes being considered one component (as described above). It also makes the procedure less computationally expensive. Otsauka & Kitahara have done exactly that for the guitar, while Dessein et al. (2010) [30] have done it for the piano. The latter was also submitted to MIREX 2010, where it got an accuracy score of 0.457 for the Multi-FO estimation task. Both of these methods are **real-time** which means that the transcription is done live as the music is playing. Furthermore, these methods have focused entirely on pitch detection combined with a very basic onset detection function working with thresholds on the time domain. When working with multiple instruments, there have also been several publications that have proposed successful methods with fixed pitch templates. Grindlay & Ellis (2011) [31] have proposed a general probabilistic model combined with NMF to transcribe music from multiple instruments without having any prior knowledge on them, except from how many instruments there are. They do that by explicitly modelling each instrument, so that each pitch is classified according to each

source and then analysed further. A major issue with the above approaches is that fixed templates could be very different to the input audio, since many parameters such as noise, the musician's playing style, the quality of the instrument, etc. can considerably affect the sound.

Adaptive template methods provide a solution for this issue, as the note dictionary is created from the audio itself, so the components closely match the spectral characteristics of the notes played. Vincent at al. (2007) [32] implemented two such variations using the NMF in combination with detecting pitch from the **W** components and detecting onsets in the **H** matrix to form a basic transcription system. Both variations were submitted in MIREX 2007 and got accuracy scores of 0.466 and 0.543 for the Multi-FO estimation task. More recently, Benetos et al. (2014) [33] proposed a method firstly using a note dictionary like the fixed template methods, which then gets adapted to the input audio. After that adaptation step, the final transcription step happens. This way the fixed template issue is solved but the advantage of having prior information on the notes played is preserved.

There are many more variations of NMF methods. An overview is given by Duan & Benetos in an Automatic Music Transcription Tutorial at ISMIR (International Society of Music Information Retrieval) 2015 [29].

3.2.2.3 Statistical model-based methods

Statistical model-based methods try to model the problem of multiple-pitch detection using statistics and probability. Several proposals view spectra as probabilistic distributions, whereas others like Goto (2004) [34] model the task as a maximum a posteriori problem:

$$\hat{C}_{MAP} = \arg\max_{C \in \mathcal{C}} P(C|x)$$

Where x is an observed frame, C is a set of fundamental frequencies and $\mathcal{C}$ is the set of all possible fundamental frequency combinations. There are many varieties of statistical approaches that have been taken to solve the multiple-pitch detection problem, but they are not going to be covered in detail in this report, as I am very unfamiliar with most of the concepts they are based on and other method categories

have been more successful in recent MIREX evaluations. Benetos et al. [27] gives a much more detailed overview of these approaches.

3.2.2.4 Classification-based methods

What is interesting is that the latest work in Automatic Music Transcription has been based on concepts not stated by the aforementioned (Yeh, Benetos). Specifically, the majority of recent work was **classification-based** using machine learning techniques. The basic concept in those is to see multiple-pitch detection as **multi-label classification**, with each pitch being a class.

Interestingly, one of the first pieces of work using a classification-based method, M. Marolt (2004) [35] was also the method with the largest accuracy in the 2016 MIREX evaluation. The method is implemented specifically for the piano and uses a set of 76 neural networks, one for each note. Networks for the lowest octave, A0-A♭1 have not been implemented because of poor results. These pitch detection networks are combined with an onset detector and a length and loudness estimator to form a full **piano music transcription** system, called SONIC.

Apart from neural networks, there have been many other machine learning techniques used for multiple pitch detection. Poliner & Ellis (2007) [36] proposed a polyphonic piano transcription system using support vector machines, while Nam et al. (2011) based their approach on deep belief networks. A similarity of those approaches are that they have been trained on piano specifically. This is because of the very limited data set in other instruments, including the guitar.

4 Implementation & Design

4.1 High-Level Overview

The main stages of the web application pipeline are:

1. Pre-Processing
2. Onset Detection
3. Duration Detection
4. Single/Multi-Pitch Detection
5. Converting to MusicXML
6. Rendering MusicXML

During the implementation process I have used methods from three libraries implementing signal processing methods for music, Librosa [37], Madmom [38] and Essentia [39]. Other standard Python libraries that also used were SciPy and NumPy.

User Journey

A user that visits the web application will firstly be asked to record or upload a guitar audio recording of their choosing. While the audio is processed by the server, the user will be asked if they wish to input any pre-defined settings for the resulting composition (e.g. the time signature of the piece). After the audio is processed, the user will be shown the rendered MusicXML file that resulted. They will be able to make changes if some of the results are wrong. Finally, after making the desired changes, the user will be able to download or share the MusicXML file.

4.2 A Complete Monophonic AMT System

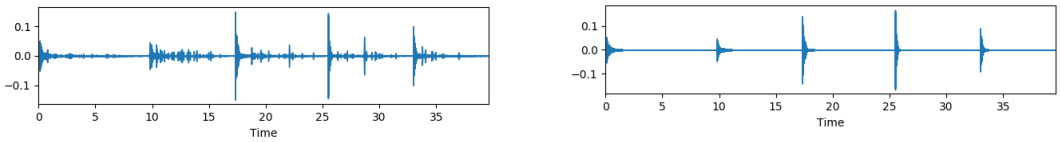
The core of this project is the implementation of a complete monophonic guitar audio transcription system, which is then integrated in a web application. As stated before, a transcription system should at least have a single pitch detection method

combined with an onset detection method. These two tasks are thus the main priorities. After these are completed, the implementation of a duration detection method and the extension of the whole system to polyphonic audio are the next priorities. Finally I experimented with chord, key, meter and tempo detection and these will be discussed in the end of the section. This section does not follow the implementation in a chronological order, but instead the order of the application pipeline. Therefore, the subsections described here follow more-or-less the order in which the audio signal gets analysed and processed.

4.2.1 Pre-Processing

The need for pre-processing the signal was clear. In many cases, a low frequency noise with high amplitude, such as a string touching the fretboard or the guitar touching the microphone would cause the program to detect very low pitches at points during the recordings. That created the need to apply a filter to the signal, so that these low frequencies could be removed. Such a filter is a very common one in signal processing and is called a **high-pass filter**. As the minimum fundamental frequency a guitar can produce is 82Hz (the E2 note), the high-pass filter was designed such that every frequency below 75Hz was discarded. 82Hz was not chosen instead as often the pitch detection methods would detect frequencies near but not exactly 82Hz.

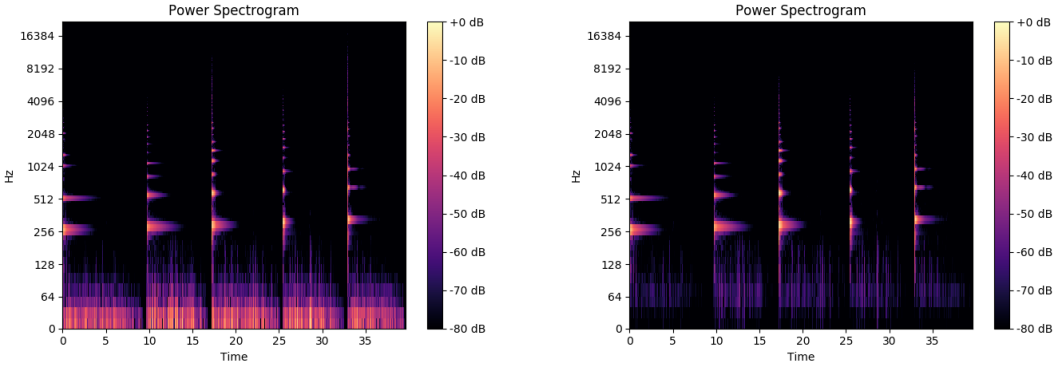
Furthermore, as the maximum fundamental frequency that the guitar can produce is at about 1600Hz, cutting frequencies above a certain value was also considered. Specifically, values in the range between 1600Hz and 5-6kHz were useful as they could include harmonics of the higher guitar notes, but frequencies above that threshold were not needed. Therefore, a **bandpass** filter (a combination of a high-pass filter and a low-pass filter) that discarded frequencies below 75Hz and above 6000Hz was also attempted and compared with the sole high-pass filter. An example of the bandpass filter applied on a guitar recording is shown in Figures 25 and 26. The evaluation of the three filtering techniques (the third being not filtering the signal at all) are shown in Table 2. How the scores in the table were computed is described in 5 whereas the pitch detection methods are described in 4.2.3. After seeing the evaluation of the three different techniques, the conclusion was that the band-pass filter did not provide any big advantage over the high-pass filter. Considering that



(a) Before band-pass filter.

(b) After band-pass filter.

Figure 25: Waveform of unfiltered (a) or filtered (b) guitar melody.



(a) Before band-pass filter.

(b) After band-pass filter.

Figure 26: Spectrogram of unfiltered (a) or filtered (b) guitar melody.

the band-pass filter also cut high frequencies that could possibly prove useful in other stages of transcription, the high-pass filter was chosen.

	Autocorrelation	**Minimum-Peak STQIFFT**	**YIN**
Band-pass filter	98.3%	88.5%	98.2%
High-pass filter	98.3%	72.5%	98.2%
No filter	97.5%	81.6%	97.4%

Table 2: A comparison of filtering/pitch detection combinations.

4.2.2 Onset Detection

In previous sections we have described pitch detection as a task of finding pitch from a signal that only contains one note. Therefore, to work with most pitch detection methods described (except peak picking from an STFT), we first had to detect the **onset times** of the signal.

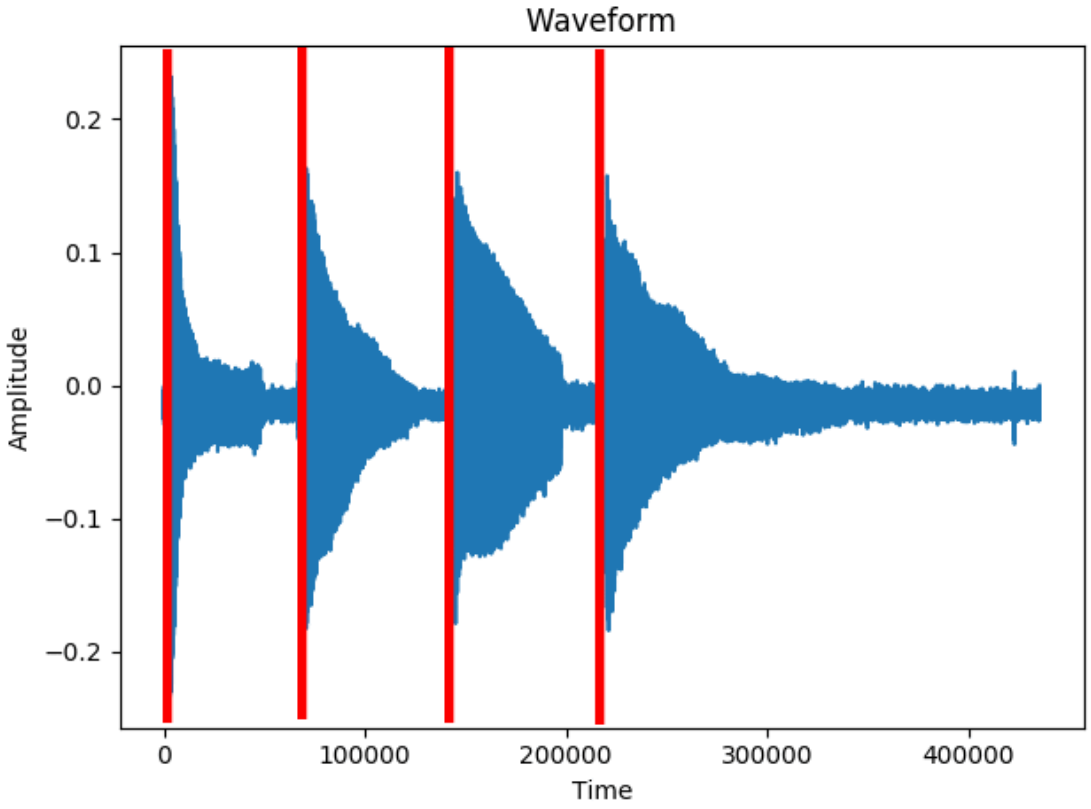


Figure 27: The onsets of a simple guitar melody.

The task of detecting onset times is intuitive, yet incredibly important. Specifically, given an audio signal, our task is to find the times t_i when the notes are played by the musician. The resulting list of length i will be used by the pitch detection algorithm to determine the pitch at those times. In Figure 27, for example, the four big peaks in amplitude (the four onset times) are immediately visible. Specifically, the final list should be (approximately) `t = [1, 80000, 130000, 210000]`. To convert those measurements in seconds, we can simply divide with the sampling rate (in this case, 44100), to get approximately 0, 1.8, 3 and 4.8.

Initially, for this task I used the Librosa built-in method `onset_detect` which picks peaks in the onset strength envelope. Its parameters have been chosen by "large-scale hyper-parameter optimisation" over a large data set. When tested on monophonic guitar samples recorded in a studio environment, there were no issues. However, when tested against audio samples recorded in normal conditions with amateur

microphones, the results were very poor.

Noise in non-studio recorded audio

When recording guitar audio with a standard, mediocre quality microphone, the signal was filled with an external noise. That caused the onset detection algorithm to detect many onset times, even if the real onset times were one or two.

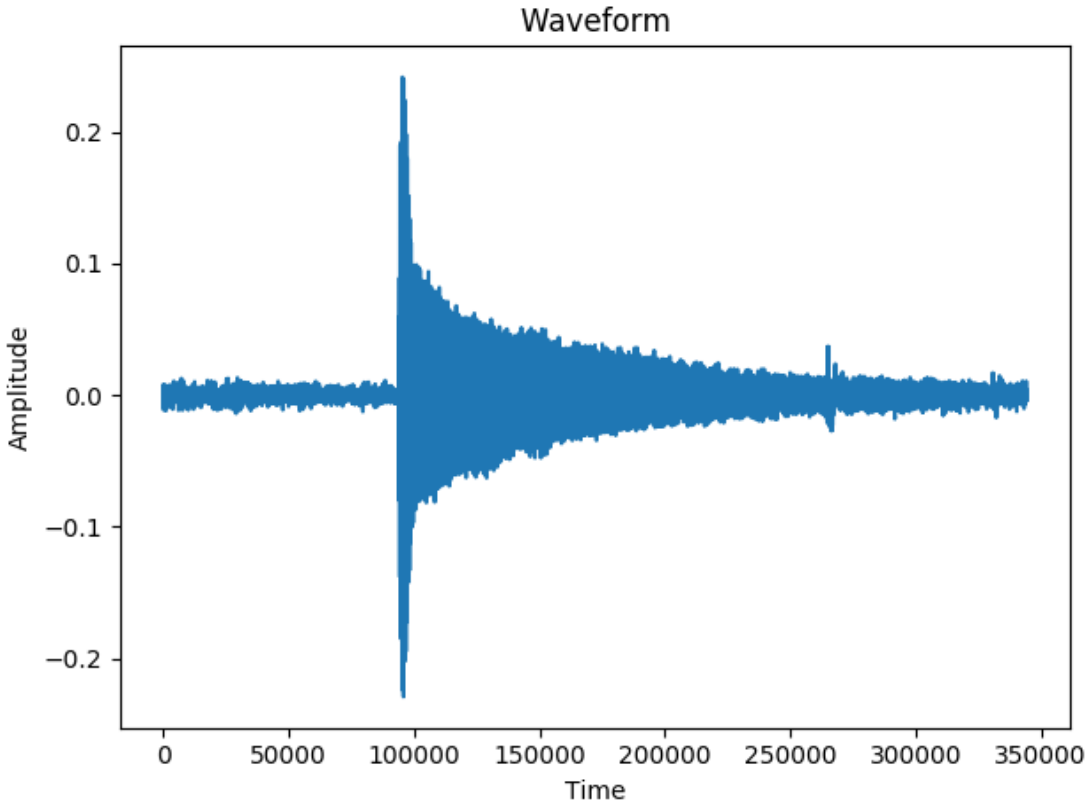


Figure 28: My recording of an E2 note.

For example, in Figure 28 I have recorded, with my mediocre quality microphone, an E2 note played with my guitar. The correct onset time is just before sample 100,000. However, the noise is clearly seen as the amplitude before the onset is never exactly 0. In contrary, there is a consistent pattern of noise peaks before and after the onset.

As a result, the `onset_detect` produced many errors when encountered with this

type of recording, wrongly considering noise peaks as onset times. For the audio shown in Figure 28, the output was a list of **7** onset times, instead of single onset time. Pitch detection later applied on this list produced, of course, many wrong and confusing results.

I implemented three simple heuristics that minimized this issue.

1. Applied an onset amplitude threshold value, `ampl_thresh`. For every onset time t_i we compare the amplitude of the strongest pitch p_i at that time to `ampl_thresh`. Onsets at times where $p_i <$ `ampl_thresh` get ignored and are not included in the result.

2. When an onset time t_1 is followed by an onset time t_2, the strongest pitch at t_1 and t_2 are approximately the same and the amplitude of that pitch at time t_1 is greater than that amplitude at t_2, then t_2 is ignored and not included in the result.

3. If for two consecutive onset times t_1, t_2 we have $t_2 - t_1 <$ `min_diff` (where `min_diff` is a positive value near 0), then t_2 is ignored and not included in the result.

To find appropriate parameter values for `ampl_thresh` and `min_diff` , several tests were done. In those tests, there was always a **trade-off** between false positives and false negatives, as setting the values too low would cause some false onset times to be included in the result, whereas setting them too high would cause real onset times to not be included. I decided that, for the scope of this application, resulting in less false positives was more important, and focused my testing on relatively low values.

Firstly, the amplitude threshold value `ampl_thresh` was chosen to be relative to the rest of the signal. Specifically, I tried to define the threshold with respect to the rest of the onset times' **mean amplitude**. That, however, caused many errors when the recordings were in a studio environment and all the onset times were correct. Finally, I ended up defining the threshold as an **absolute value**, and the one that picked up the best results was the value of 5dB, as I could not find a recording where a real onset time has an amplitude of less than that value. On the other hand, `min_diff` was defined to be 0.1 seconds. This was enough to cover eighth notes in a tempo of 220 which was deemed sufficient as tempo faster than this is

very rare.

When applying these heuristics to the E2 recording, I got a correct result of one onset time which was later successfully detected as an E2 note by my pitch detection algorithm. Moreover, my optimized version minimized the issue when more than one onset time is present, such as the one shown in Figure 27, since, compared to the 57 onset times detected by the original `onset_detect` algorithm, my version reduced that number to 7.

Because of the minimization of this issue, onset detection was not a priority. However, I later found Madmom [38], a library containing implementations of several State of the Art algorithms. Specifically, they provided Superflux [40], ComplexFlux [41], an RNN-based onset detector proposed in [42], and traditional methods such as Spectral Diff [43] and Spectral Flux. At this point, I also had a more complete data set to test my onset detector on, enriched by more melodies recorded with a good quality microphone. When testing those new methods, I had to optimise six main parameters:

- `threshold`
- `smooth`
- `pre_max`
- `post_max`
- `pre_avg`
- `post_avg`

The threshold has the traditional meaning (amplitude) and the smooth parameter determines the level of smoothness that the input function will experience. The `pre_max`, `post_max`, `pre_avg` and `post_avg` parameters gave information on future and past moving averages/maximums.

Table 4 (in Section 5) shows the comparison between the different methods when tested on the optimal parameters. All methods (except the "Librosa without heuristics") include step 3 of the heuristics explained above (removing the onsets that are close to each other) which improved their score by 0.2-0.4%. In conclusion, I chose the Spectral Flux function, as not only it produced better results on my data set, but it was also significantly faster than the rest of the methods. Roughly, it works

by measuring the difference between consecutive time frames and detecting onsets in increases in spectral content.

4.2.3 Single Pitch Detection

The task of pitch detection has, as explained in previous sections, several approaches. Librosa provides a method `piptrack`, which was the first method used for pitch detection. Autocorrelation was also implemented and the YIN algorithm was taken from the Essentia library [39]. After comparison between those three methods, the autocorrelation method was preferred.

Thresholded parabolically-interpolated STFT

As discussed in Section 3.1.3, the simplest method to find the pitch of a given signal is to first compute its Fourier transform and then pick the largest peak in the frequency domain. Librosa implemented an optimised version of this simple method, called the quadratically interpolated FFT (QIFFT) [44]. QIFFT provides several steps that make it more precise when searching for the strongest frequency in a frequency spectrum. Firstly, it **zero-pads** the signal (extends it with zeros), then computes the FFT and finally interpolates the peaks in the spectrum to estimate the largest one. QIFFT efficiently solves the problem of precision that we addressed in Section 3.1.3, but not the one of weak fundamental frequencies.

Of course, the QIFFT, similarly to the standard FFT does not take into account time. When trying to apply it on a signal where pitch is variating, it is not sufficient. Librosa `piptrack` method implemented a Short-Time Fourier Transform variation (**STQIFFT**), using the QIFFT instead of the FFT. The output of the function was simply two `ndarrays`, `pitches` and `magnitudes`, both with `shape=(d, t)]`, where `d` is the number of FFT bins and `t` is the duration of the signal in frames. To find the largest peak at time `t`, one needed to go through the array `magnitudes[:, t]`, find the bin in which the maximum magnitude is located, and then find the instantaneous pitch at that particular bin with `pitches[bin, t]`. With a functional onset detector, firstly finding the onset times of the signal and then finding the largest peaks at those times make up a simple pitch detector.

However, when tested on guitar samples, it simply was not enough, as the problem of fundamental frequencies not being the strongest peaks was very common.

Minimum Peak STQIFFT

To yield better results, I tried to implement a variation of Librosa's `piptrack` such that weaker fundamental frequencies are not ignored. Specifically, at each onset time, instead of extracting the peak with the largest magnitude and considering it the fundamental frequency of the note, I extracted a number of peaks. For each note, therefore I did not have one frequency which I would consider as the pitch but a list of `candidate_pitches`. From those pitches, the one with the lowest frequency was considered the pitch. This version of the QIFFT proved much more precise, as it fixed many errors that the method had when tested on the complete data set.

Furthermore, looking for the minimum peak at each onset time t sometimes resulted in frequencies that were part of the attack transient of the guitar. Therefore, I tweaked my method so that the lookup for the minimum strongest peak is done at time $t + offset$. After testing, the offset value of 25 proved to be the most appropriate.

Segmenting the signal

As the rest of the methods that were implemented to detect pitch work in the time-domain, some sort of additional pre-processing had to be implemented so that the signal is segmented first, and then each segment is fed onto these methods. Therefore, given a list of onset times, my task was to find the correct indexes where I would slice the signal.

Initially, I attempted to slice the signal at points of local minima right before the onset times. Librosa provided a handy method that backtracks the signal after finding the onset times and returns the indices of the local minima. These were used to slice the signal: for each onset time t_i, we find the local minima min_{t_i} that precedes t_i and we use that as the start of the slice i. At the end of slice i, we simply use $min_{t_{i+1}} - 1$. This is depicted in Figure 29 where the beginnings/ends of the slices are shown in red.

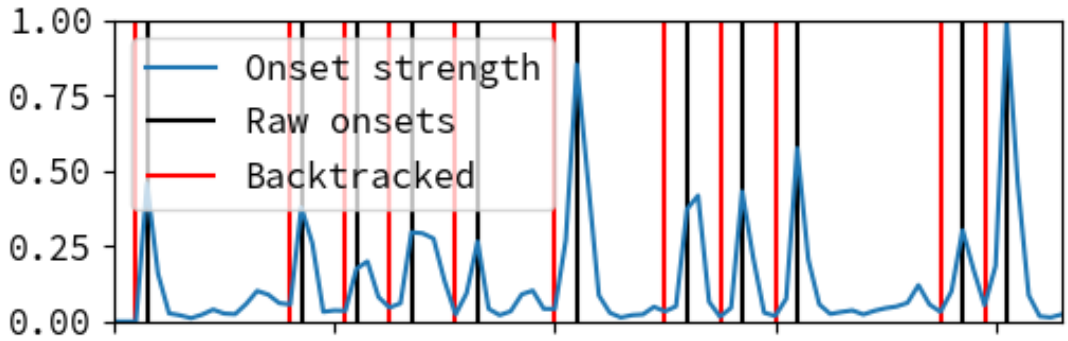
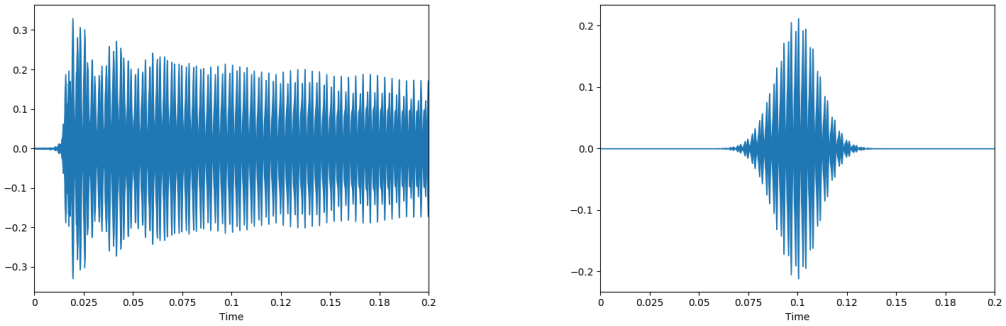


Figure 29: The backtracked onset strength envelope [37].



(a) Non-windowed slice.

(b) Windowed slice.

Figure 30: The effect of a window function in the time domain.

This worked but produced some errors as the attack transients at note onsets were fully included to each slice, resulting to some estimation errors in the pitch detection algorithm. As a result a new method had to be implemented that would cancel out these transients. Instead of each slice being defined as $[min_{t_i}, min_{t_{i+1}})$, we defined it as $[t_i + \texttt{start\_offset}, t_i + \texttt{end\_offset}]$. With a high `start_offset`, all attack transients after the note onset were correctly ignored, but some harmonics of the sound were dissolved as well, as their duration was short. A high `end_offset` allows for better precision but creates problems when consecutive onsets are close to each other. Finally, the existence of a `start_offset` is deemed unnecessary as **windowing** was more accurate. Specifically, each slice is multiplied by a window function so that it gets smoothly zeroed-out outside an interval. The can be seen in Figure 30. After applying the window function, the results were improved by more than 10% for both methods implemented.

Time-Domain Methods

The autocorrelation method was implemented as described in 3.1.2, and applied segments of the signal corresponding to the piece's onset times, as described above.

To choose which peak in the autocorrelation function corresponds to the correct period, the `peakutils` library is used to get a list of peaks in the function above a certain threshold. Instead of choosing a fixed threshold value, it is initially set as a large value, and gets decreased in consecutive iterations. In each iteration, we try to find peaks in the autocorrelation function above the current threshold while discarding those which corresponded to a fundamental frequency out of the guitar's reach (less than E2 and more than E6). If we find multiple period peaks, then we consider the smaller as the correct one.

As stated before, an attempt to limit the attack transients of each note was the segmenting of the signal with a starting offset. However, this proved unnecessary when windowing the slice. Specifically, when using autocorrelation with the optimal `start_offset` and `end_offset` values without windowing, the accuracy score was **83.2%**. After applying windowing, the optimal parameters resulted in a score of **98.1%**. Table 3 shows some of the different parameters used for comparison.

`start_offset`	`end_offset`	**Score**
0	0.1	96.2%
0	0.2	98.1%
0	0.25	97.6%
0.01	0.1	94.6%
0.01	0.2	97%
0.05	0.1	94.8%
0.05	0.2	97.6%
0.1	0.15	95.6%

Table 3: Comparison of different offset parameters for autocorrelation.

The YIN algorithm described in 3.1.2 was taken from the Essentia library and was applied to a segmented, windowed signal similarly to autocorrelation. Interestingly, the YIN method did not improve the results as expected. Using the optimal `start_offset` and `end_offset` parameters, the score achieved was 98.2% an almost

identical score my implementation of the autocorrelation function.

4.2.4 Duration Detection

The task of duration detection is not very different in monophonic and polyphonic cases. In both cases, the task involves two main steps: firstly, assigning a duration, in seconds (or samples), to each note played. Secondly, converting that duration to the sheet music terms, described in 2.1. For the scope of this project where duration detection is not the primary focus, I assumed a precise onset detection method and defined a note's duration to be the difference between its onset time and the next onset time.

Therefore, from the list of onset times o I got from my onset detector, I firstly extracted a list of durations d where the note with duration $d[i]$ has an onset time $o[i]$. The last note played, which has no "next note" to get an onset from, is given a duration of the maximum duration of the song, usually a whole note. Then, I had to find a way to convert those durations - which were up until that point measured in frames - to the standard sheet music terms.

Initially, I developed a simple way using only the tempo of the song:

1. Convert all the durations from frames to seconds.
2. Derive the quarter duration in seconds given the tempo of the song. For example given a tempo of $quarter = 120$, we know that the quarter will last 0.5 seconds ($60 seconds/120$).
3. Divide all the durations by the quarter duration. For example, for a quarter duration of 0.5 seconds, $[0.486, 0.99, 0.22]$ becomes $[0.972, 1.98, 0.44]$.
4. Round every duration to the nearest "music" duration. For example, $[0.972, 1.98, 0.44]$ becomes $[1, 2, 0.5]$.
5. Convert to music sheet terms. This is done by the Music21 library [45].

However, this proved to be very prone to human error. Duration detection in general proved to be a tricky task, as a slight inaccuracy by the musician will result in a wrong duration. As an example, Figure 31 shows the score for a song recorded for the purpose of testing the application.

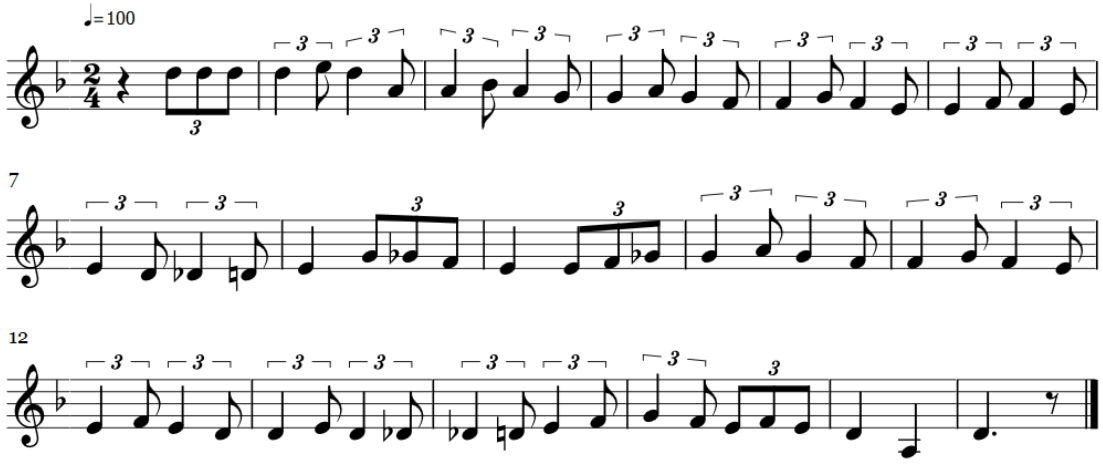


Figure 31: "Fragkosiriani" in score notation.

It was recorded without a metronome, and therefore I was expecting slight inaccuracies. Indeed, the durations of the first measure (the first four notes) in seconds (before step 3) was: $[0.18575964, 0.18575964, 0.2554195, 0.39473923]$. We already come across an issue: the third note is not of an equal duration as the first two. After step 3, the list becomes: $[0.31, 0.31, 0.42, 0.66]$, which makes the first two notes round down to 0.25 and get converted to eighth notes whereas the third note rounds up to 0.5 and gets converted to a half note. The resulting score using this method is shown in Figure 32.

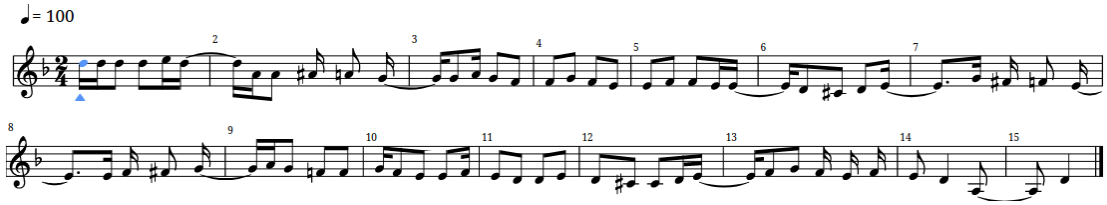


Figure 32: "Fragkosiriani" in score notation with the initial duration detection method.

Therefore, a method less prone to such inaccuracies had to be implemented. The new method's main concept was **normalizing** the durations with respect to others. The initial implementation was simple:

1. Find the minimum element of the duration list (we don't have to convert to

seconds). That element is considered our reference quarter duration.

2. **Normalize** the list: Divide every element of the list with the reference quarter and round it to the nearest integer.
3. We now have a list of the form $[1.0, 4.0, ..., 2.0, 2.0]$. Convert that to music sheet terms.

This was a basic implementation that, in most cases, produced the correct results. Again, however, the precision was dependent on the musician. If the recording contained one note that was mistakenly very smaller than the other ones, the method produced very wrong results. This was fixed by running two rounds of normalization:

1. Steps 1-3 as above.
2. In the normalized list generated, get the indices of all the notes that were considered having a duration of 1.0.
3. Get the mean of the durations from the above indices. That will be our new reference duration, `min_avg`.
4. Normalize the initial list, but this time with respect to `min_avg`.
5. Get the ratio of the quarter length derived from the tempo to `min_avg`. That tells us what is the `min_avg` compared to a quarter; is it a eighth, a quarter, etc.
6. Divide every duration in the final normalized list with `ratio`. Before this, the minimum element in the normalized list was 1.0, making the minimum duration a quarter. This converts all the relative durations to actual durations derived from the tempo.
7. Round every duration to the nearest "music" duration. For example, $[0.972, 1.98, 0.44]$ becomes $[1, 2, 0.5]$.
8. Convert to music sheet terms.

This fixed the issue we had before. We first normalize the list using the minimum duration, for example 22 (in frames), and then find that the mean minimum duration (`min_avg`) is somewhere around 25. This is used for the second "round" of normalization. In general, this yielded very good results on all the melodies in my

data set. The updated duration detection method on "Fragkosiriani" is shown in Figure 33 to be almost identical to the ground truth one.

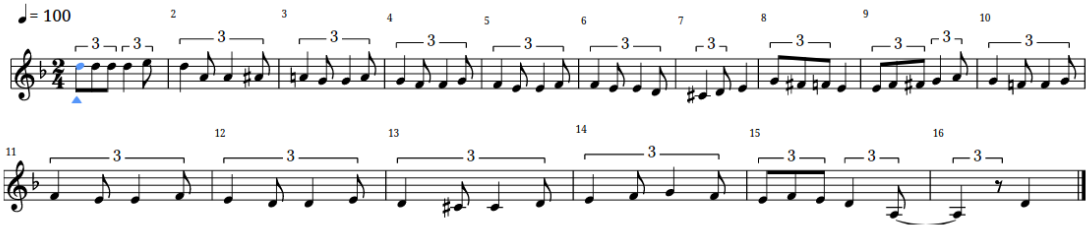


Figure 33: "Fragkosiriani" in score notation with the second duration detection method.

Let us consider some disadvantages and other challenges in this approach.

Firstly, a guitar song could contain pauses. By defining a note's duration as the difference between its onset to the next onset, we do not take into consideration any long pauses that could be in between. Specifically, we end up with a note having a very large duration instead of its real duration and a following pause. A possible solution to this would be by tracking the pitch of the notes. Specifically, when looking at a note, we can consider that it has stopped playing when the magnitude at its peak reaches 0 or peaks suddenly (which means that a new note with the same pitch was played). This could also be reduced by applying a threshold to a note's duration. For example, we can assume there is no way a guitar note can last more than 4 seconds. Therefore, if we find a note with a duration more than 4 seconds, we can "cut" its duration to 4 and consider the rest a pause.

Furthermore, determining durations is still hugely dependent on the musician's (and our onset detection algorithm) precision. One note played too early or too late could cause many error to the durations of all the other notes, as a note's duration in sheet music is relative to a reference note, and not absolute. An example of this is shown in 33, where still the last measure contains a mistake. If the musician uses this web application to record their audio, then providing a metronome would reduce this issue.

Of course there are other issues we haven't tried to solve at all. For example, trying to detect duration in some music styles like Jazz would affect our results significantly, as sometimes delays and speed-ups exist on purpose.

4.2.5 Converting to MusicXML

After detecting the onset times and the corresponding pitches, a conversion to MusicXML had to be made. To do that the `Music21` [45] module was used. This task was fairly simple, as after pitch and duration detection I had what I needed to pass to the `Music21` module. A simple stream of `Note` objects was created, with each `note` having its own `pitch` and `duration`. A simple call to the `write` function created a MusicXML file and stored it into the server.

4.2.6 Rendering MusicXML

Rendering MusicXML was done with the use of the `Flat.io` API, which provided an interactive music sheet renderer with a combined guitar audio player, metronome and the option to print and export as PDF.

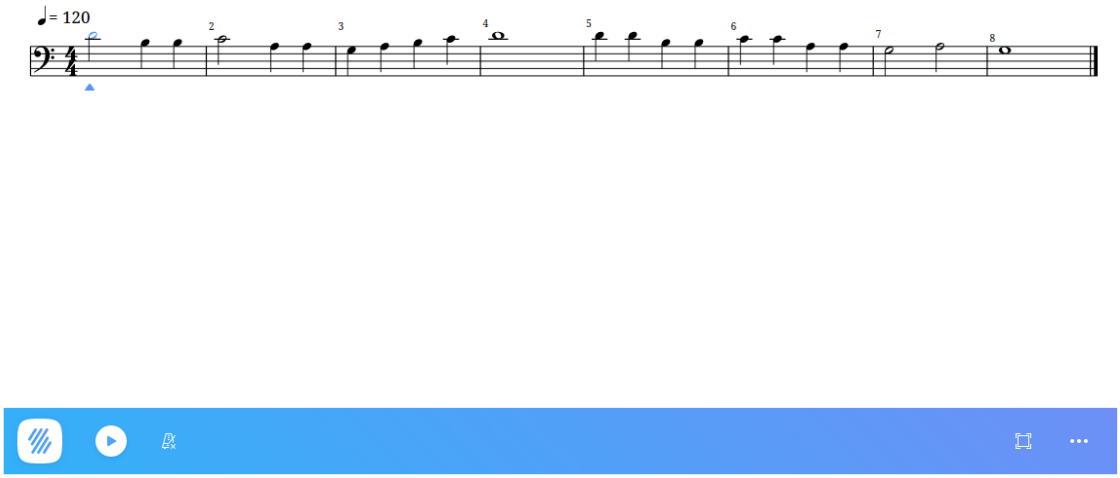


Figure 34: The Flat.io `Embed` container.

The button on the bottom left corner also allows users to direct themselves to the Flat.io website, so that they can edit their sheet music and download it or share it from there.

4.2.7 Tempo, Meter and Key

Tempo, meter and key detection were outside of the scope of this project, and methods for these tasks are taken from the Essentia library. The results for tempo detection are almost almost perfect with a rare mistake of halving/doubling the estimated tempo. To minimize that, a boundary for minimum and maximum tempo was set, so that the error is sometimes avoided. After some research and a discussion with a professional musician I decided that 60 and 220 were appropriate boundaries for tempo. On the other hand, key signature and time signature detection were quite erroneous. As these details of the piece are not time consuming for the musician to provide, it was deemed more appropriate that they are part of the user input before the processing of the song.

4.3 Extension to Polyphonic Audio

As we have stated before, there is no large data set with guitar audio. Therefore, the implementation of a classification-based method for multiple pitch detection was not possible. Furthermore, the time frame to work on the multiple pitch detection algorithm was small; therefore the method implemented had to be relatively simple.

The method attempted was a non-negative matrix factorisation method similar to the ones described in 3. Initially, the method was tested on monophonic samples. The magnitude spectrogram was provided by Librosa, and I had to find a method to decompose that into two matrices **W** and **H**. For this task, the most standard method is the one provided by the `scikit.learn` library, but I decided to go with the Librosa built in method `decompose` which was a bit easier to use, as I did not require any special set up.

To become familiar with this, the initial approach was simplified, where the number of components R was provided to the method. When given the monophonic sample of "Soft Kitty" (Figure 35) the number of components provided was 5, as this is the number of unique notes in the piece.

The resulting **W** and **H** matrices are shown in Figures 36 and 37.

One quickly understands it is not clearly visible to which pitch each component belongs. However, each component is a frequency spectrum and therefore we can apply

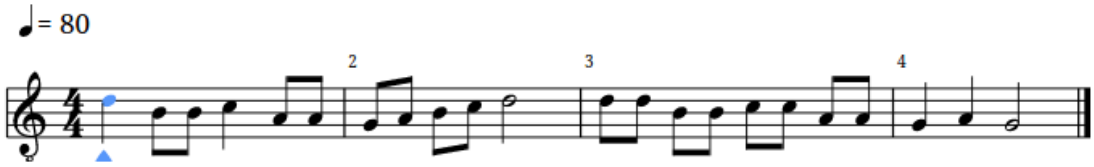


Figure 35: The "Soft Kitty, Warm Kitty" score.

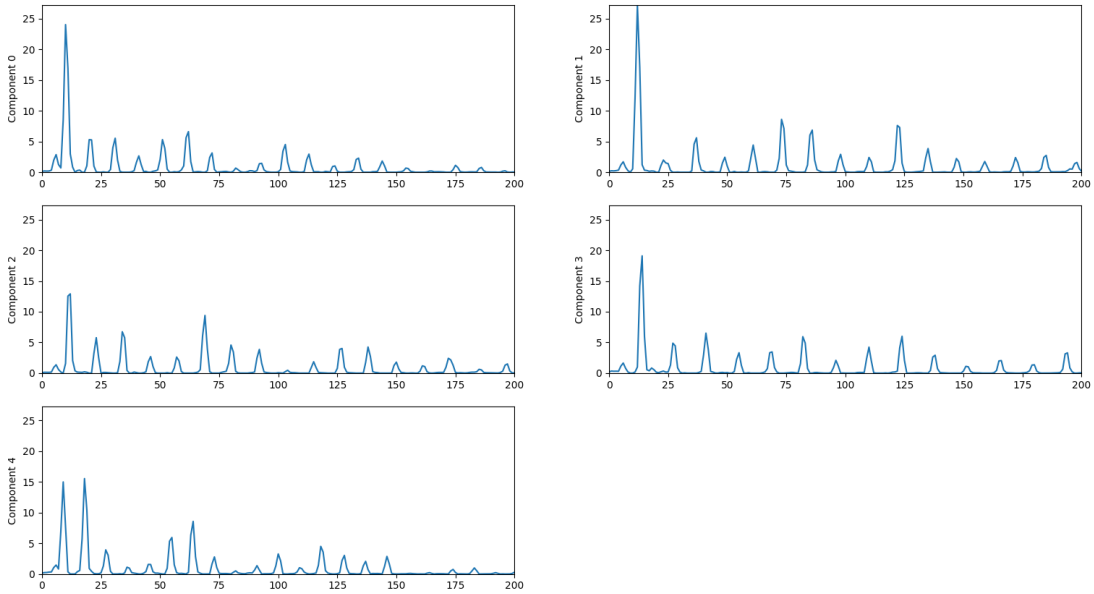


Figure 36: The pitch templates of the piece (the x-axis represents frequency bins, not Hz).

any pitch detection technique we want to that spectrum. Essentia's `YinPitchFFT` (which converts the signal from the frequency-domain to the time-domain and then applies YIN), for example, immediately gives us the pitch for each component:

- Component 0: 221Hz
- Component 1: 264Hz
- Component 2: 248Hz
- Component 3: 296Hz
- Component 4: 196Hz

Indeed, these frequencies correspond to A3, C4, B3, D4, G3 respectively. By looking

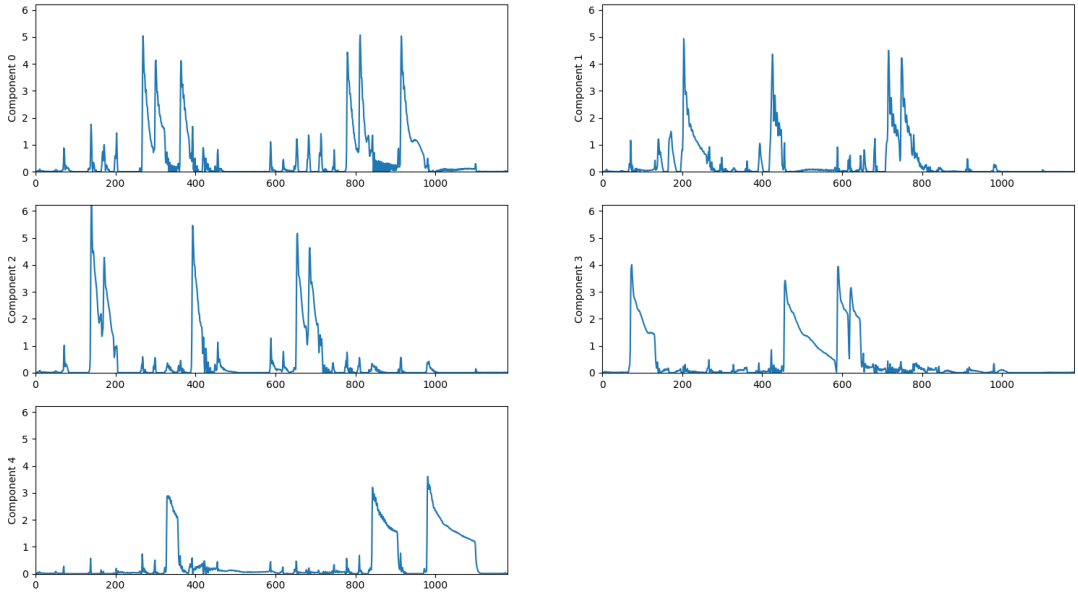


Figure 37: The temporal activations of the piece with 5 components.

at the H matrix (Figure 37 we see that component 3 (D4) was played first, then component 2 twice (B3), and so on.

These results are pleasing. However, when we do not provide the number of components before hand, then the decomposition is not so accurate. Specifically for "Soft Kitty, Warm Kitty", the decomposition sets R to 1025, instead of 5. This makes the whole procedure very inefficient, as a single NMF takes about 10 minutes.

A more sensible approach is to set the number of components as the number of notes that can be played using a guitar (49). The resulting **W** matrix can be seen in Figure 38. Unfortunately, the decomposition has been quite unsuccessful. The number of components is too large, and certain harmonics have been considered standalone features. There are many more than 5 components that could be considered notes, and therefore this is not a good approach.

After these attempts, a similar approach to Dessein et al. (2010) [30] was followed. This paper proposes a **fixed pitch template matrix** which is built using pre-recorded note samples. Of course, the paper is proposing a **real-time** system, in contrast to ours which is off-line. Also, the system described in the paper is based on the piano. These two differences are quite substantial. The off-line aspect of our

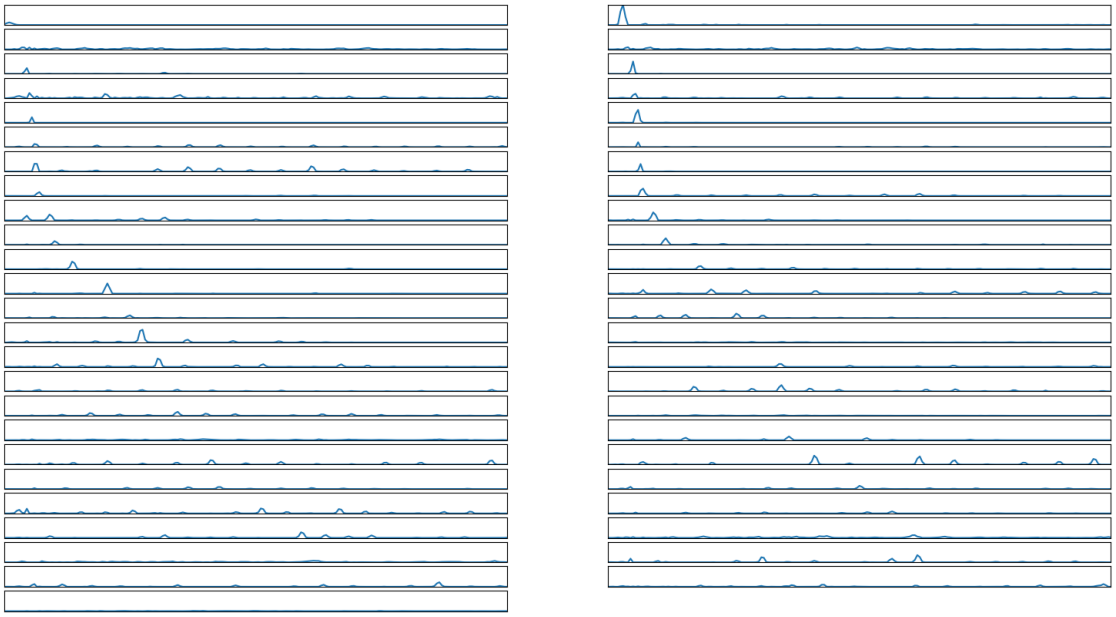


Figure 38: The temporal activations of the piece with 49 components.

system makes the task easier; however, the guitar presents one significant challenge when compared to the piano and that is the possibility to play the same note in different ways.

Therefore, to perform the NMF on "Soft Kitty, Warm Kitty" without providing the number of components, the first task was to create a pitch template matrix. To do that, each unique note played in the piece in its correct fret/string combination (D4 on the B string, B3 open B string, etc.) was fed into an NMF with rank $R = 1$. The five decompositions resulted in five w vectors. For example, the NMF of the D4 note resulted in the vector shown in Figure 40. By saving that vector to a file using the `numpy.savetxt` function, and doing the same for each note played in the song, we end up with five files. The H matrices are discarded, and the vectors in the files are then stacked up in columns using `numpy.loadtxt` to load the array from the text file and `numpy.column_stack` to stack the vectors in columns and form the matrix. Plotting the newly created **W** matrix looks exactly like the W matrix in 36, which is expected. Furthermore, applying the `YinPitchFFT` to each component shows that the pitches remain correct.

We now re-do the NMF of the original piece "Soft Kitty, Warm Kitty" while also providing the newly created **W** matrix. To do so, we have to use the NMF

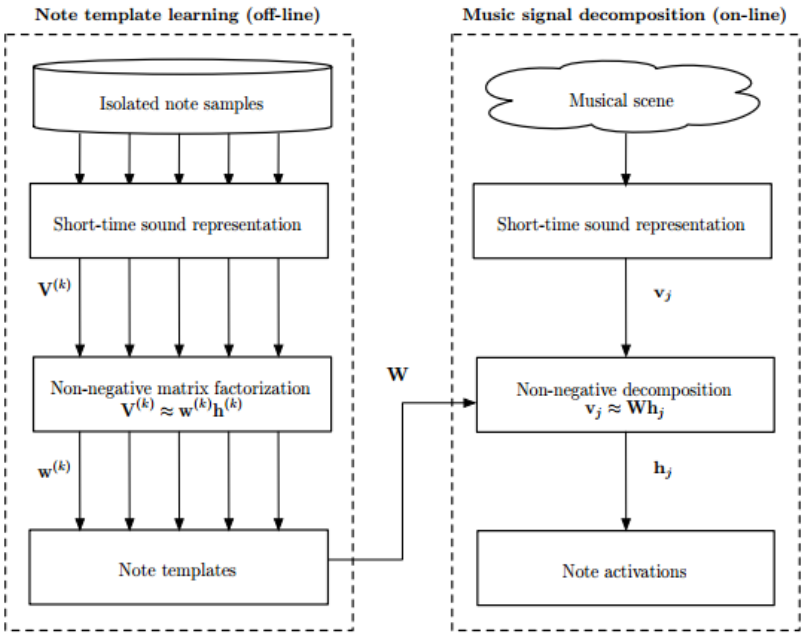


Figure 39: The general architecture of Dessein et al. (2010) [30].

method `scikit.learn`, as Librosa does not expect a fixed **W** matrix. Actually `scikit.learn` does not expect a fixed **W** matrix either, but it does expect a fixed **H** matrix. Therefore, we transpose **W** and provide it as fixed **H** matrix. We also need to transpose the input spectrogram and the resulting activations matrix. After doing that, we see that the results are identical to Figure 37. Therefore, our approach has been sound. However, we have a long way to go to detect multiple pitches. Firstly, we have to find a good way to detect onsets in the activation matrix. Unfortunately, we cannot entirely base our solution to the onset detection methods described in 4.2.2. Also, we have to see what happens when the pre-computed **W** matrix contains more than the notes in the audio. Finally, we will have to inspect the NMF's behaviour when we provide polyphonic samples.

To detect onsets using NMF, what we have implemented in 4.2.2 is not sufficient. If we take an onset time given from the already implemented method, we have to search through the activation matrices and find to which components that onset time belongs. That becomes quite complex when working with multiple pitches. Instead, we use the `peakutils` package to pick peaks in the temporal activation matrices. We use a low threshold to avoid having many false nega-

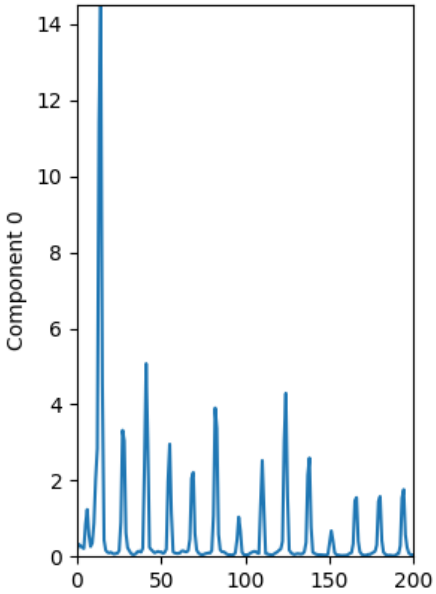


Figure 40: The w vector of the D4 note.

tives and get a list of possible peaks for each component. Then, these are cross-checked using the onset detection method described in 4.2.2. For example, for the first component in 37, the `peakutils.indexes` function returns `[269, 301, 364, 781, 812, 915]`. Our previous detection method applied on the whole piece returns `[70, 137, 163, 201, 266, 298, 327, 361, 391, 416, 454, 586, 618, 652, 676, 713, 745, 777, 809, 841, 912, 978]`. It is clear that these produced by picking peaks in the temporal activation matrices are valid are correspond to `[266, 298, 361, 777, 809, 912]`, and the others are discarded. Because of the slight imprecision, an onset was considered correct if the distance between itself and an onset produced by the method in 4.2.2 was less than 15 frames.

Subsequently, we insert more pitch templates in the $\mathbf{W}$ to see the temporal activations of the new components. Specifically the notes added were the E2, F2, F♯2, G2 and G♯2. The W matrix of now rank $R = 10$ contain those and the ones in the "Soft Kitty, Warm Kitty" piece. The resulting temporal activation matrix is shown in Figure 41 and is what we expected; the components that are played in the piece are temporally active while the others do not have any peaks. The decomposition has also successfully distinguished between the G3 note, actually played in the piece, and the G2 note (octave below), which is not. However, the peak detection based

onset detector detects peak in every component, and that is because the threshold applied in `peakutils.indexes` is **normalized**, meaning that it is relative to the values of the rest of the function. Therefore any very small peak in a function which is mostly zero is considered an onset. That can be fixed by also applying an absolute threshold, which is quite risky and may cause issues in the future.

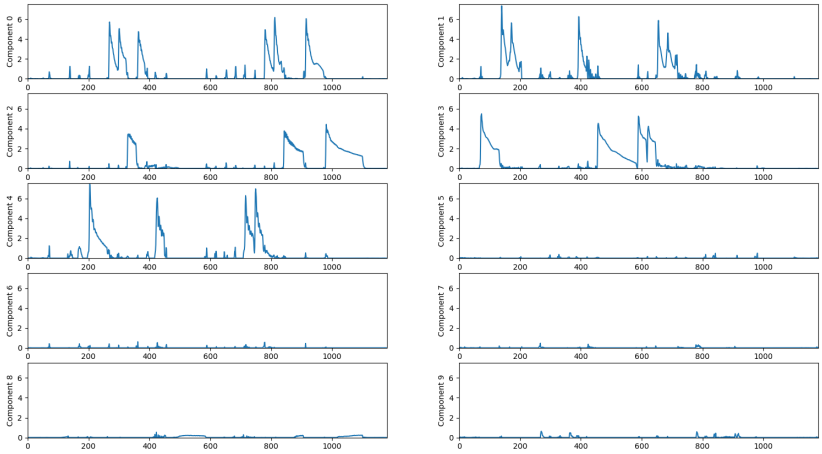


Figure 41: The temporal activation matrix after NMF with five extra components.

Finally, we add pitch templates for all notes in the guitar in a similar fashion. By performing NMF with the whole matrix on "Soft Kitty, Warm Kitty", we get the same results. However, when attempting to decompose a spectrogram of **polyphonic** audio, we encounter our first two issues: firstly, the attack transient of each note corrupts the temporal activations of other pitches. For example, shown in Figure 42 is the lower components of a polyphonic song containing E2 and A2 notes. Unfortunately, the NMF decomposition is imprecise and considers F2, G♯2 and A♯2 and B2 more temporally active than A2. This is believed to be because of two reasons: first, the samples used to generate the pitch templates are not representative of all guitar samples. Secondly, the initial attack transients in the notes produce non-harmonic frequencies that are hard to distinguish.

To solve the first issue, two solutions are proposed. Firstly, the pitch templates were now constructed with computer-generated "pure" guitar sounds. This way, the sound used to generate the pitch template contains only what "defines" that pitch: the fundamental frequency and harmonics. However, this resulted in worse scores, specifically a fall from 48.1% to 43.3% in overall accuracy on my polyphonic

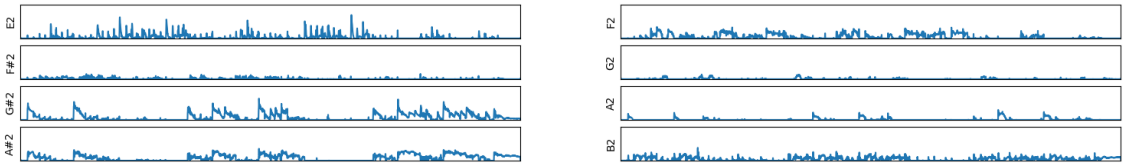


Figure 42: The temporal activation matrix given a polyphonic sample.

data set. The issue with the imprecision because of attack transients is solved, but now we have many mistakes towards harmonics (for example a C3 note detected as C4). As a result, we reduce both issues by taking a middle solution, and constructing the pitch templates from a concatenated (in series) real and digital tone. This way, the overall accuracy score was increased to 52.5%. Secondly, before decomposing the spectrogram of the audio we want to analyse, we apply a 1-dimensional median filter to each column (spectrum) to smooth to avoid sudden high peaks of amplitude, which is a common attack transient characteristic. This improves the results by 3% to get a final overall accuracy score of 55.5%.

5 Evaluation

While implementing the transcription system described above, valid metrics were needed to evaluate the designed modules. Similar state of the art systems like Alcabasa et al. (2012) [46] are being tested on custom data sets, as there is no standard guitar music data set. Similarly, we create our own data set for the evaluation of the two systems. Initially, 18 samples provided by the University of Iowa Electronic Music Studios were used to evaluate onset detection and pitch detection. These monophonic samples, recorded in a studio environment with a classical guitar contained notes played without any rhythm. Later, when there were more requirements from the transcription system such as multiple pitch detection and duration detection, simple melodies and whole, more complex songs recorded (in a home environment) from family and friends were used. The total number of recordings used for the evaluation of the project is 50 (35 monophonic and 15 polyphonic). We perform the evaluation of the two transcription systems (monophonic and polyphonic) separately, as they differ greatly in requirements and tests applied.

5.1 Monophonic Transcription

In this subsection we evaluate the three main modules of the monophonic transcription system, namely onset detection, single pitch detection and duration detection. We discuss the results and then compare them to the two published similar applications, ScoreCloud[2] and AnthemScore[3].

Onset Detection

Initially, we begin with onset detection evaluation as it is a prerequisite for the pitch and duration detection methods. The data set used for the onset detection method was the whole set of recordings (47) which contained both monophonic and polyphonic samples. The evaluation followed the standard conventions of onset detection evaluation with precision, recall and f-measure computed for every recording, where an onset is considered correct if there is a ground truth onset within 100ms around

[2]http://www.scorecloud.com/
[3]https://www.lunaverus.com/

the predicted time. The ground truth annotations were produced by running the onset detection method with a very low threshold (so that there is high recall) and then refining the results manually. Table 4 shows the comparison between the onset detection methods.

	Precision	**Recall**	**F-measure**
SuperFlux	99.3%	95.3%	96.6%
ComplexFlux	98.3%	95.6%	96.4%
RNN	87.6%	93.6%	89.5%
SpectralDiff	99.8%	96%	97.4%
SpectralFlux	99.5%	97.9%	98.4%
Librosa	72.5%	100%	81.3%
Librosa with heuristics	88.8%	86.3%	85.5%

Table 4: Comparison of different onset detection methods.

The data set is comprised of three main types of recording: 18 studio recordings, 9 good quality home recordings and 8 poor quality home recordings. Therefore, it is interesting to see how the best method overall, SpectralFlux, performs in these individual subsets. Table 5 shows the results, which are very interesting. We can see that there is a symmetrical relationship between the recording types. Studio recordings have a high precision and low recall, whereas bad quality home recordings have the opposite. Good quality home recordings stand somewhere in the middle. This is normal considering that the parameters chosen for an onset to be detected have been chosen to work for all types of recordings. When setting, for example, an amplitude threshold value, we had to take into account that bad quality recordings will have noise. Setting a high amplitude because of that will affect the results in recordings where there is no noise. The opposite also applies. Therefore, this table pictures the trade-off encountered when setting those parameters.

	Precision	**Recall**	**F-measure**
Studio	100%	96.5%	98%
Good quality home	99.4%	98%	98.6%
Bad quality home	98%	100%	99%

Table 5: Performance of SpectralFlux depending on recording type.

Single Pitch Detection

The evaluation of the single pitch detection method is done in a simpler way, by considering a pitch correctly detected if the conversion to a note results in the ground truth note. For example, for a ground truth $F\sharp 4$, 380Hz is considered correct, whereas 350Hz is considered wrong. I preferred this method from comparing the raw frequencies themselves (measuring difference in Hz as the error) as it followed the logarithmic scale in which the pitch is perceived. Ground truth onset times are provided to the evaluation method, which then calls the implemented `get_pitches`. Table 6 shows the comparison of the three evaluated methods, STQIFFT (see 4.2.3), Minimum Peak STQIFFT (see 4.2.3), Autocorrelation and YIN (see 4.2.3). The mean accuracy of an individual piece is computed by diving the correctly detected pitches over all the detected pitches. The average value of those scores is considered the **overall accuracy score** of the pitch detection method.

	Accuracy
STQIFFT	73.2%
Minimum Peak STQIFFT	88.5%
Autocorrelation	98.3%
YIN	98.2%

Table 6: Comparison of different single pitch detection methods.

Interestingly, my autocorrelation implementation produced results identical to the YIN algorithm (provided by Essentia) which was developed as an improvement of autocorrelation and is considered one of the optimal solutions. The windowing of each slice in the time domain definitely had a very important role in this, as the attack transients were eliminated while leaving enough time to detect the period. Table 7 shows the three main stages applied to the autocorrelation function and how they improved the scores. Unfortunately, while my variation of the STQIFFT was a big improvement over the simple QIFFT provided by Librosa, the precision in the detection of the frequencies was a big issue.

	Error
1. Basic autocorrelation function	26%
2. Iterative decrease of threshold	21%
3. Discarding the out-of-reach notes	19%
4. Windowing slices	1.6%

Table 7: Each stage's contribution in the minimization of the error.

Duration Detection

The duration detection algorithm described in Section 4.2.4 was very precise when applied on my data set. Of course, this data set was only comprised of the recordings that included rhythm (9 recordings). The testing of the method was done similarly to the pitch detection method, where we provide the filename and the ground truth onset frames to the testing method which calls `get_durations`. We then compare the ground truth durations with the output of the method, and compute an average score for the song. The average of these average scores over the data set was considered the overall accuracy score.

The evaluation of the method resulted in the exceptional overall accuracy score of **96.7%**. Furthermore, we notice that in most songs where a metronome has been used for the recording that scores are 100%, whereas in the recordings where it has not, the scores lie in the 70%-80% range. That is of course, expected, as the attempts described in Section 4.2.4 could not entirely minimize this issue. Also, we have not provided the ground truth **tempo** for these recordings. Therefore, we notice that the tempo detection method provided by Essentia is very reliable which overrides my initial decision of asking the user for the tempo before the processing.

Comparison with State of the Art

When comparing our transcription system with the state of the art, I had to consider that the data sets used for guitar-specific state of the art transcription systems are very different. Therefore, the results described above cannot be directly compared to the papers that use the instrument-agnostic MIREX data set. However, we can compare them with AnthemScore, the only automatic transcription application

I found after my research. AnthemScore is an implementation of the Lunaverus algorithm [23] embedded in a desktop application for all operating systems. It supports the transcription of audio recordings to MusicXML which can then be edited with other software. Lunaverus was submitted in the latest MIREX (2016) and achieved an accuracy of 44%. It is optimised for the piano but works well with all instruments, both for monophonic and polyphonic audio.

AnthemScore

AnthemScore provides a feature where audio files can be taken as input to generate MusicXML files, similarly to our application. Therefore, a fair comparison can be done in terms of the note and duration precision of the resulting MusicXML file. As AnthemScore is an implementation of Lunaverus, a successful and recent MIREX submission, we can understand where this project lies in accuracy.

As there was no way to automate this comparison between the two MusicXML files, the 9 audio files in my data set that contain **rhythm** were given as input to AnthemScore. The resulting music sheets are shown in Appendix B. It is clear by looking at the resulting music sheets that our monophonic transcription system has performed better on the data given. The AnthemScore output makes common polyphony estimations, octave errors (and other pitch errors), several wrong onsets and durations and also does not produce tuplets (two consecutive eights or sixteenths that "group" together), which makes reading the score hard. Of course, we also have to mention that AnthemScore also supports other instrument input and polyphony, therefore producing better results at this specific task does not mean that our system in general is better.

5.2 Polyphonic Transcription

The polyphonic transcription algorithm we used is non-negative matrix factorisation, which can be used to for both multiple pitch detection and onset detection. However, as stated in Section 4.3, the onset detection algorithm of NMF has been integrated with the onset detection method implemented in 4.2.2. Therefore the polyphonic transcription method only returns a list of lists of notes. The first list, which can contain more than one element, is the notes played at the first onset, and so on.

Having said that, we evaluate the polyphonic transcription system as a multi-pitch detection algorithm (similarly to single pitch detection), by providing the ground truth onset frames and comparing the ground truth notes to the detect notes. This has several flaws that should be considered and fixed in the near future: firstly, the **number** of notes detected is not considered. If the ground truth notes at an onset time are `[E2, F2]` and the method detects `[F2]`, there will be no penalty for only detecting one note. This has been implemented using `difflib`'s `SequenceMatcher` which provides a method to get a "measure of the sequences' similarity as a float in the range `[0, 1]`." The improvement of this evaluation so that it considers all factors is a big priority for future work.

Table 8 shows the results with the existing evaluation method when using real, digital and combined samples for the construction of pitch templates, **without** the use of the filter. The filter improved the result of the method with **combined** samples by 3% to get a final 55.5%.

	Accuracy Score
Real	48.1%
Digital	43.3%
Combined	52.5%

Table 8: Comparison of different samples used to construct the pitch templates.

The data set used to evaluate it is limited to recordings with two notes played at a time. When encountered with more complex audio, it produces worse results. However, as this was implemented as an extension to the main part of the project, the results are definitely not considered bad. There are several issues that, due to the limited amount of time, were not studied appropriately. For example, an issue in polyphonic transcription which was ignored for the implementation of this extension was the fact that durations cannot just be assumed to be the difference between successive onsets. Often, in polyphonic piece a low note sounds at the same time that a melody is played. A future optimisation of this system would definitely need a better evaluation method and further research in the field of NMF so that these issues can be tackled.

6 Conclusion and Future Work

This paper introduced the reader to the field of Automatic Music Transcription and the State of the Art methods to perform it. In Section 4, the implementation of an **Automatic Monophonic Guitar Music Transcription** was showcased and discussed, whereas in Section 4.3, the extension to polyphonic audio was presented. In Section 5, the two AMT systems were evaluated using different annotated data sets comprised of 18 studio recordings, 18 good quality home recordings and 8 bad quality home recordings.

The main contribution of this project is the implementation of an accurate automatic music transcription web application, which allows composers to automatically convert their guitar melodies to score notation. The AMT system implemented is not only the sole open-source[4] AMT application developed but also the only **web-based** one.

The autocorrelation-based pitch detection method implemented produces an accuracy score of 98.3%, slightly better than that of the YIN method provided by a third party library. The onset detection function provided by Madmom [38] is also very accurate with a resulting f-measure of 98.4%. These methods are combined with a reliable duration detection method to form a **precise AMT system** which successfully transcribes an average guitar melody.

The evaluation of the monophonic AMT system as a whole shows very successful results, outperforming AnthemScore, the only complete AMT application that has been published in the task of **monophonic guitar audio transcription**. The extension of the system to work with polyphonic audio, based on non-negative matrix factorisation, encountered many interesting challenges and limitations. Its evaluation was promising, getting an accuracy score of 55.5% on the limited and simple data set it was applied on.

The development of the guitar audio transcription system and especially its extension to polyphonic audio should not be limited to this project. The project set a strong base where a great amount of future work could be applied, especially in the fields of multiple pitch detection.

[4] https://www.github.com/pk1914/guitarmat

6.1 Future Work

If more time was available, there would be several main issues to focus on. Firstly, in the monophonic audio transcription system, the data set has been comprised of relatively simple melodies. Many guitar pieces include characteristics that have not been discussed, such as very quick solos or vibrato. The consideration of quick solos, for example, would have a big effect on our onset detection method. Therefore, an enrichment of the data set with more annotated pieces, both of good and bad recording quality, would be a top priority.

Furthermore, there are several aspects of a score that have not been covered. We rely on a third party library for **tempo detection**, as it produced very promising results, but have not discussed **time signature detection** and **key signature detection**. These are tasks very different to the ones we have presented and pose other challenges. Of course, the first two differ from, for example, pitch detection, as they can be defined by the musician with one click. That is why they have not been a top priority for this project. However, a complete AMT system could have these features; AnthemScore, for example, does provide a time signature and key signature detection. Therefore, a discussion on these should be done in case this project is continued further.

One of the most important aspects of a complete AMT system is, of course, polyphonic audio transcription. A promising proof-of-concept non-negative-factorisation method based on Smaragdis et al. (2003) [28] has been implemented, however big optimisation steps are required for it to be usable. Firstly, a fair evaluation method should be implemented, as the one used for the evaluation of this system does not take into account the number of notes detected in an onset frame, or the more complex durations that can occur in polyphonic audio. Secondly, the addition of more annotated pieces in the data set is required, as the current number of 8 pieces is certainly not enough to test a complex polyphonic system. Of course, these evaluation tools do not have any use if the actual multiple pitch detection and onset detection methods themselves are not developed further. There are many improvements that could be implemented on the current system, including construction of NMF templates with more than two samples (to take into account more playing styles and guitar types) per pitch and heuristics to cancel the bias against low notes which are currently detected as less "temporally active" in general than the higher notes.

7 References

[1] B. Lathi, *Linear Systems and Signals*, ser. Oxford series in electrical and computer engineering. Oxford University Press, 2005. [Online]. Available: https://books.google.co.uk/books?id=7resQgAACAAJ

[2] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer International Publishing, 2015. [Online]. Available: https://books.google.co.uk/books?id=HCI_CgAAQBAJ

[3] A. Klapuri and M. Davy, *Signal Processing Methods for Music Transcription*. Springer US, 2007. [Online]. Available: https://books.google.co.uk/books?id=AF30yR41GIAC

[4] A. Lerch, *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*. Wiley, 2012. [Online]. Available: https://books.google.co.uk/books?id=YSPT1LJqTbIC

[5] D. Carter, "How to read music and rhythm in 10 minutes for beginners," 2017, [Accessed June 02, 2017]. [Online]. Available: https://hubpages.com/entertainment/Learn-to-Read-Music-in-Ten-Minutes

[6] "How to read music durations, measures and time signatures," 2017, [Accessed June 02, 2017]. [Online]. Available: http://musicweheart.com/how-to-read-music-note-durations-measures-and-time-signatures/

[7] lbartman, "Guitar fretboard - tablature relation," [Accessed February 10, 2017]. [Online]. Available: http://lbartman.com/worksheet/how-do-u-read-guitar-tabs.php

[8] O. J. Guitar, "Music notation," [Accessed February 10, 2017]. [Online]. Available: http://www.jazzguitar.be/miles_davis_licks.html

[9] N. Fletcher and T. Rossing, *The Physics of Musical Instruments*. Springer New York, 2008. [Online]. Available: https://books.google.co.uk/books?id=9CRSRYQlRLkC

[10] I. U. Bloomington, "Standing waves on a string," 1999, [Accessed February 10, 2017]. [Online]. Available: http://hep.physics.indiana.edu/ rickv/Standing_Waves_on_String.html

[11] R. Priemer, *Introductory Signal Processing*, ser. Advanced series in electrical and computer engineering. World Scientific, 1991. [Online]. Available: https://books.google.co.uk/books?id=QBT7nP7zTLgC

[12] archive.cnx.org, "The frequency domain," 2017, [Accessed June 02, 2017]. [Online]. Available: http://archive.cnx.org/contents/9d10dbd4-b95d-45dd-8e70-4c68b868c681@6/another-way-of-viewing-the-world

[13] A. T. Cemgil and B. Kappen, "Monte carlo methods for tempo tracking and rhythm quantization," *Journal of Artificial Intelligence Research*, vol. 18, no. 1, pp. 45–81, 2003.

[14] K. Lee and M. Slaney, "Acoustic chord transcription and key extraction from audio using key-dependent HMMs trained on synthesized audio," *IEEE Transactions on audio, speech, and language processing*, vol. 16, no. 2, pp. 291–301, 2008.

[15] A. M. Barbancho, A. Klapuri, L. J. Tardón, and I. Barbancho, "Automatic transcription of guitar chords and fingering from audio," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 3, pp. 915–921, 2012.

[16] A. De Cheveigné and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, 2002.

[17] A. Klapuri and A. Eronen, "Automatic transcription of music," in *Proceedings of the Stockholm Music Acoustics Conference*, 1998, pp. 6–9.

[18] C. Roads, *The Computer Music Tutorial*. MIT Press, 1996. [Online]. Available: https://books.google.co.uk/books?id=nZ-TetwzVcIC

[19] E. Gómez, A. Klapuri, and B. Meudic, "Melody description and extraction in the context of music content processing," *Journal of New Music Research*, vol. 32, no. 1, pp. 23–40, 2003. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1076/jnmr.32.1.23.16799

[20] P. De La Cuadra, A. S. Master, and C. Sapp, "Pitch detection methods review," accessed: 2017-06-30. [Online]. Available: https://ccrma.stanford.edu/ pdelac/154/m154paper.htm

[21] A. M. Noll, "Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate," in *Proceedings of the symposium on computer processing communications*, vol. 779, 1969.

[22] Y. Chunghsin, "Multiple fundamental frequency estimation of polyphonic recordings," Ph.D. dissertation, Université Lille 1, 2008.

[23] D. Troxel, "Music transcription with a convolutional neural network," 2016.

[24] A. P. Klapuri, "Multiple fundamental frequency estimation based on harmonicity and spectral smoothness," *IEEE Transactions on Speech and Audio Processing*, vol. 11, no. 6, pp. 804–816, 2003.

[25] A. Klapuri, "Multiple fundamental frequency estimation by summing harmonic amplitudes," in *ISMIR*, 2006, pp. 216–221.

[26] "Music Information Retrieval Evaluation Exchange (MIREX)," 2017, [Accessed June 03, 2017]. [Online]. Available: http://www.music-ir.org/mirex/wiki/

[27] E. Benetos, S. Dixon, D. Giannoulis, H. Kirchhoff, and A. Klapuri, "Automatic music transcription: challenges and future directions," *Journal of Intelligent Information Systems*, vol. 41, no. 3, pp. 407–434, 2013.

[28] P. Smaragdis and J. C. Brown, "Non-negative matrix factorization for polyphonic music transcription," in *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.* IEEE, 2003, pp. 177–180.

[29] Z. Duan and E. Benetos, "Automatic Music Transcription Tutorial - ISMIR 2015," 2015.

[30] A. Dessein, A. Cont, and G. Lemaitre, "Real-time polyphonic music transcription with non-negative matrix factorization and beta-divergence," in *ISMIR-11th International Society for Music Information Retrieval Conference*, 2010, pp. 489–494.

[31] G. Grindlay and D. P. Ellis, "Transcribing multi-instrument polyphonic music with hierarchical eigeninstruments," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 6, pp. 1159–1169, 2011.

[32] E. Vincent, N. Bertin, and R. Badeau, "Two nonnegative matrix factorization methods for polyphonic pitch transcription," in *2007 Music Information Retrieval Evaluation eXchange (MIREX)*, 2007.

[33] E. Benetos, R. Badeau, T. Weyde, G. Richard *et al.*, "Template adaptation for improving automatic music transcription," 2014.

[34] M. Goto, "PreFEst: A predominant-F0 estimation method for polyphonic musical audio signals," *Proceedings of the 2nd Music Information Retrieval Evaluation eXchange*, 2005.

[35] M. Marolt, "A connectionist approach to automatic transcription of polyphonic piano music," *IEEE Transactions on Multimedia*, vol. 6, no. 3, pp. 439–449, June 2004.

[36] G. E. Poliner and D. P. Ellis, "A discriminative model for polyphonic piano transcription," *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, pp. 154–154, 2007.

[37] B. McFee, M. McVicar, O. Nieto, S. Balke, C. Thome, D. Liang, E. Battenberg, J. Moore, R. Bittner, R. Yamamoto, and et al., "Librosa," Feb 2017.

[38] S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer, "madmom: a new Python Audio and Music Signal Processing Library," in *Proceedings of the 24th ACM International Conference on Multimedia*, Amsterdam, The Netherlands, 10 2016, pp. 1174–1178.

[39] D. Bogdanov, N. Wack, E. Gómez, S. Gulati, P. Herrera, O. Mayor, G. Roma, J. Salamon, J. Zapata, and X. Serra, "ESSENTIA: an open-source library for sound and music analysis," in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM '13. New York, NY, USA: ACM, 2013, pp. 855–858. [Online]. Available: http://doi.acm.org/10.1145/2502081.2502229

[40] S. Böck and G. Widmer, "Maximum filter vibrato suppression for onset detection," in *Proc. of the 16th Int. Conf. on Digital Audio Effects (DAFx). Maynooth, Ireland (Sept 2013)*, 2013.

[41] ——, "Local group delay based vibrato and tremolo suppression for onset detection."

[42] S. Böck, A. Arzt, F. Krebs, and M. Schedl, "Online real-time onset detection with recurrent neural networks," in *Proceedings of the 15th International Conference on Digital Audio Effects (DAFx-12), York, UK*, 2012.

[43] C. Duxbury, M. Sandler, and M. Davies, "A hybrid approach to musical note onset detection," in *Proc. Digital Audio Effects Conf.(DAFX,'02)*, 2002, pp. 33–38.

[44] J. O. Smith, *Spectral audio signal processing*. W3K, 2011.

[45] M. S. Cuthbert and C. Ariza, "music21: A toolkit for computer-aided musicology and symbolic music data," 2010.

[46] L. Alcabasa and N. Marcos, "Automatic guitar music transcription," in *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, Nov 2012, pp. 197–202.

Appendices

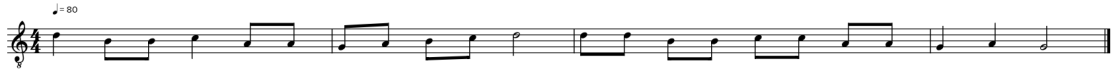
A Pitch - Frequency Table

	C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B
0	16.35	17.32	18.35	19.45	20.6	21.83	23.12	24.5	25.96	27.5	29.14	30.87
1	32.7	34.65	36.71	38.89	41.2	43.65	46.25	49	51.91	55	58.27	61.74
2	65.41	69.3	73.42	77.78	82.41	87.31	92.5	98	103.8	110	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185	196	207.7	220	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370	392	415.3	440	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740	784	830.6	880	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

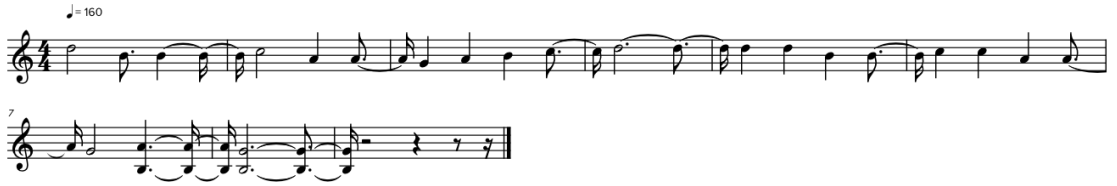
Table 9: Relation between frequencies and pitches.

B Music Sheets

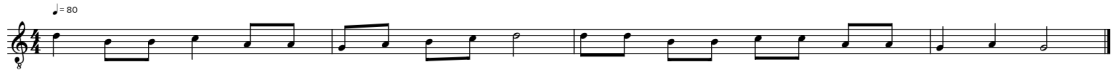
This section shows, for each song in the 9-song data set described in 5, the output music sheet from the application implemented for this project (top), AnthemScore (middle) and the ground truth music sheet (bottom).



(a) Identical to ground truth.

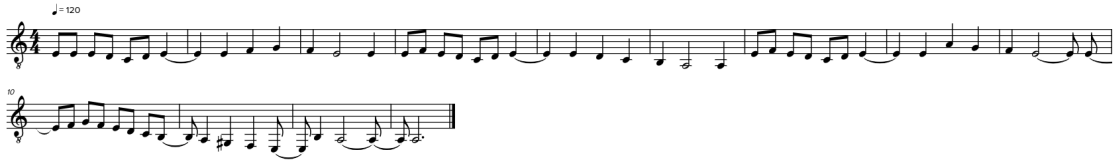


(b) Several notes wrongly estimated as polyphonic. Also, multiple wrong durations.

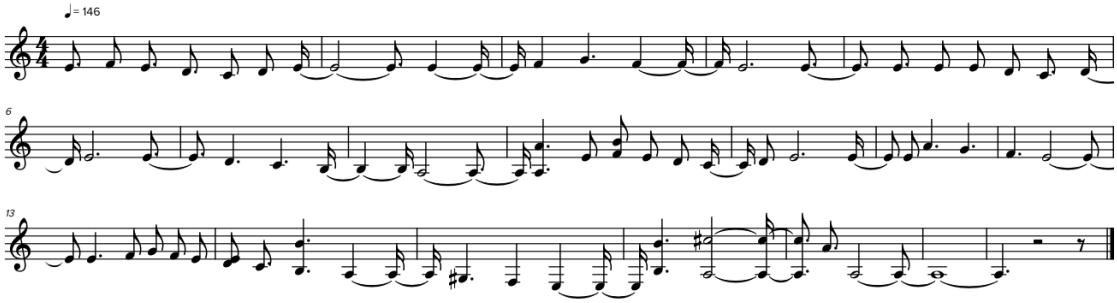


(c) Ground truth score.

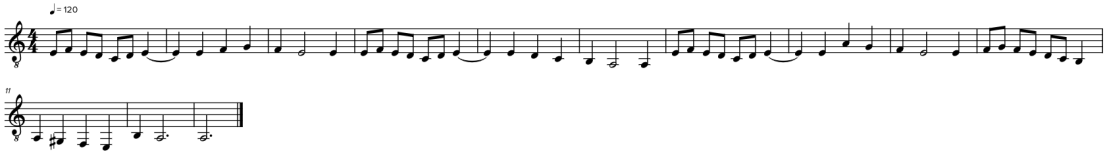
Figure 43: "Soft Kitty, Warm Kitty"



(a) Only second note on score wrong.

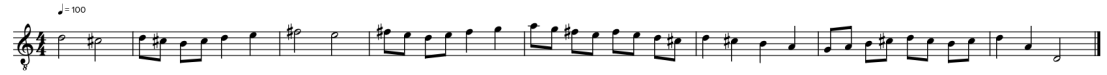


(b) Several notes wrongly estimated as polyphonic. Also, multiple wrong durations and onsets.

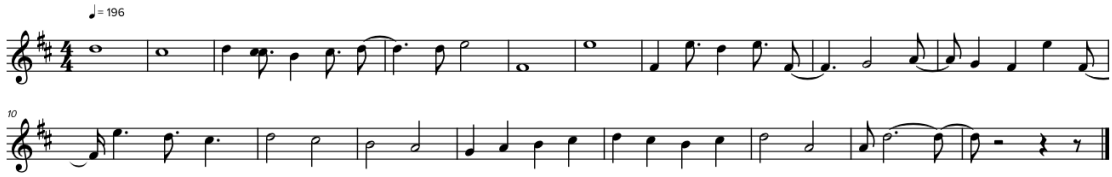


(c) Ground truth score.

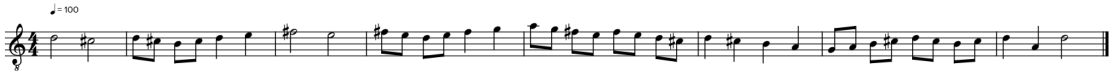
Figure 44: "Improvisation 1"



(a) Octave error in last note.

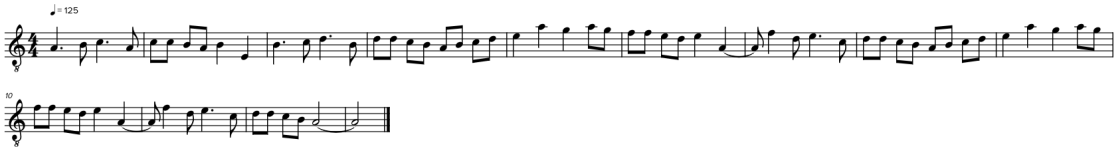


(b) Several notes wrongly estimated as polyphonic and some pitch errors throughout the piece.

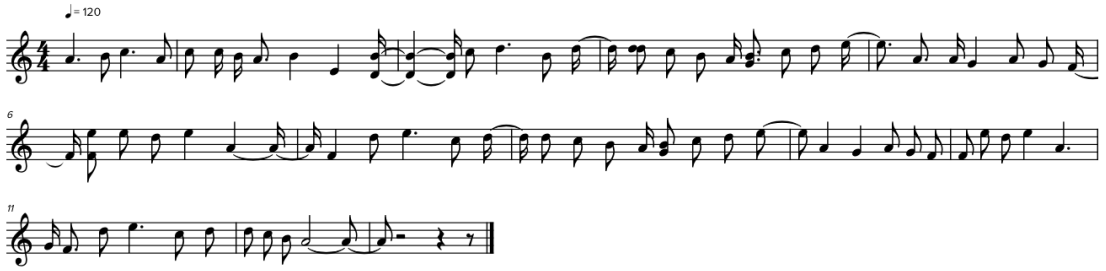


(c) Ground truth score.

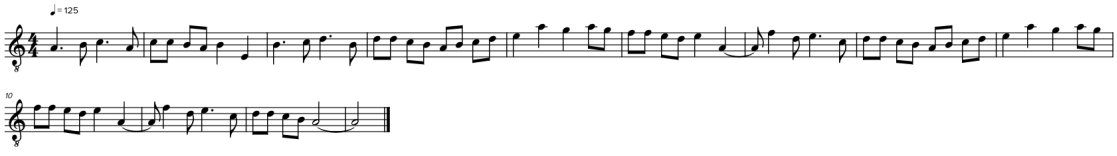
Figure 45: "Improvisation 2"



(a) Identical to ground truth.

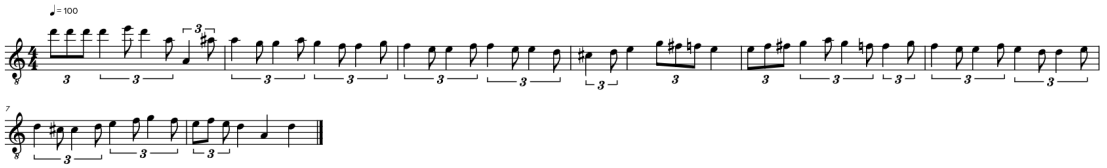


(b) Several notes wrongly estimated as polyphonic. Some wrong onsets.

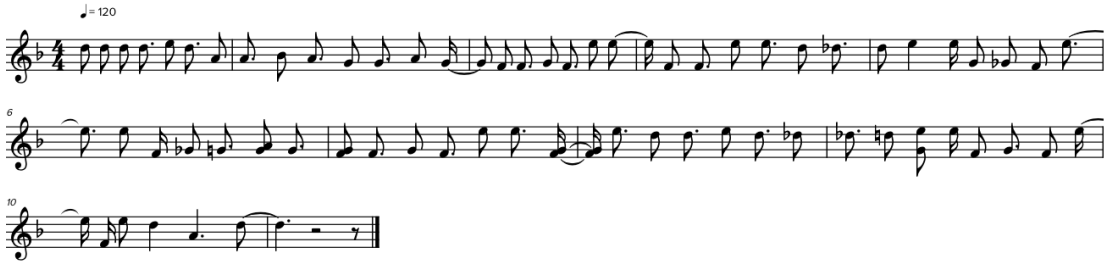


(c) Ground truth score.

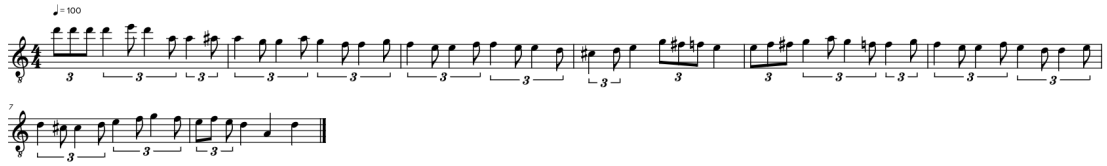
Figure 46: "To EAM"



(a) One octave error (8th note).

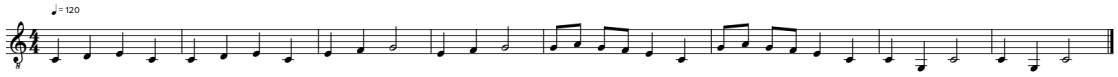


(b) Multiple wrong notes.

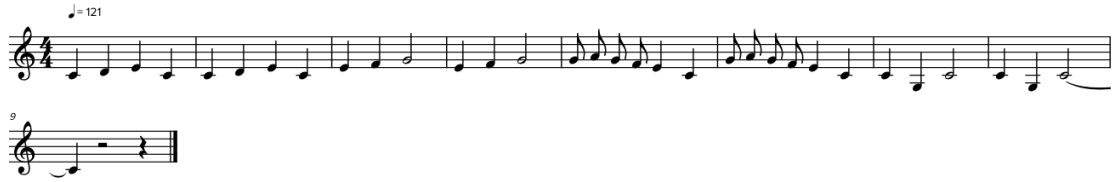


(c) Ground truth score.

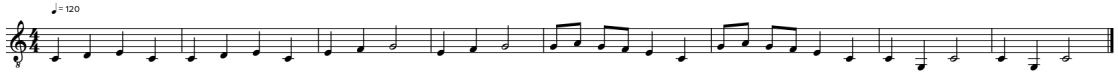
Figure 47: "Fragkosiriani"



(a) Identical to ground truth.

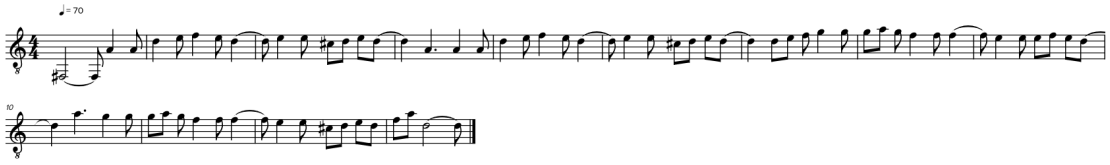


(b) Identical to ground truth.

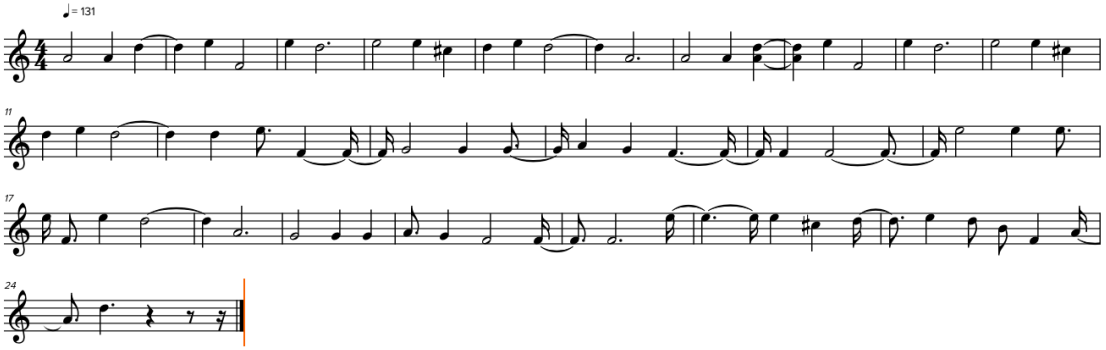


(c) Ground truth score.

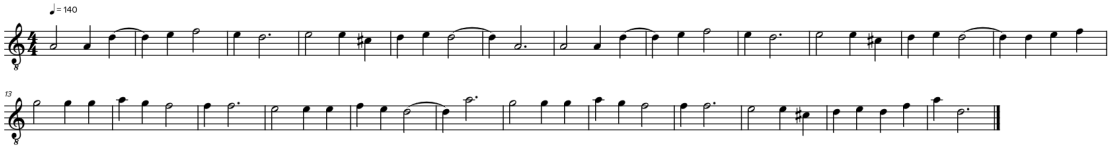
Figure 48: "Frere Jacques"



(a) Wrong onset in the beginning. Otherwise identical to ground truth.

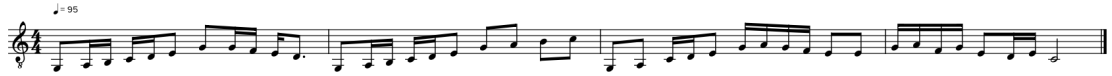


(b) Multiple note errors.

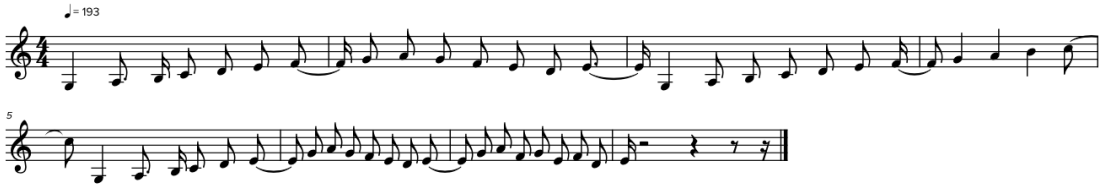


(c) Ground truth score.

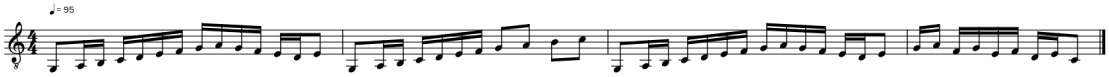
Figure 49: "Asta ta malakia sou"



(a) Some duration errors throughout the piece.

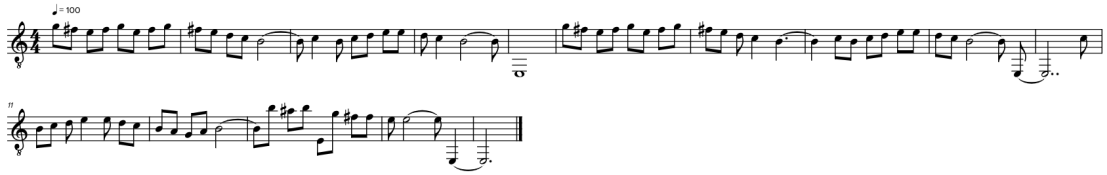


(b) Some duration errors throughout the piece.

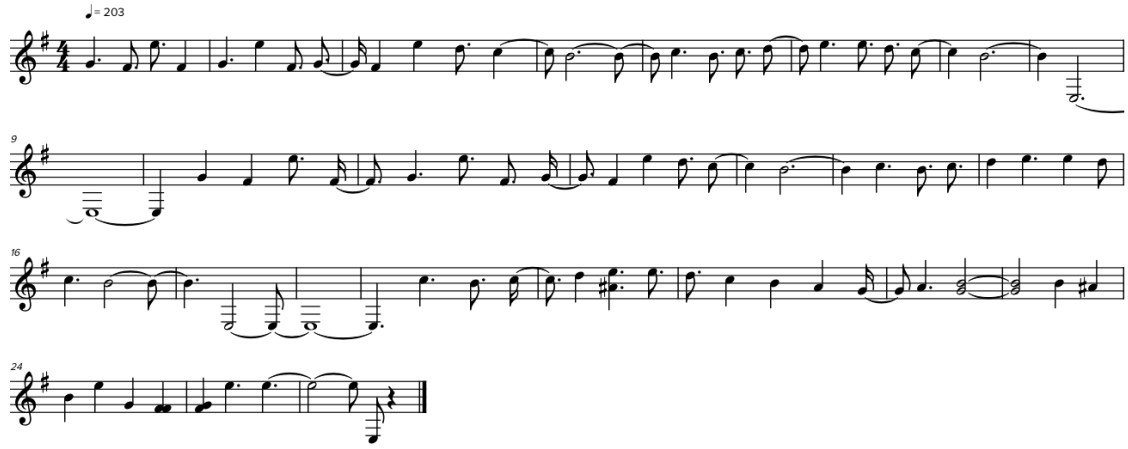


(c) Ground truth score.

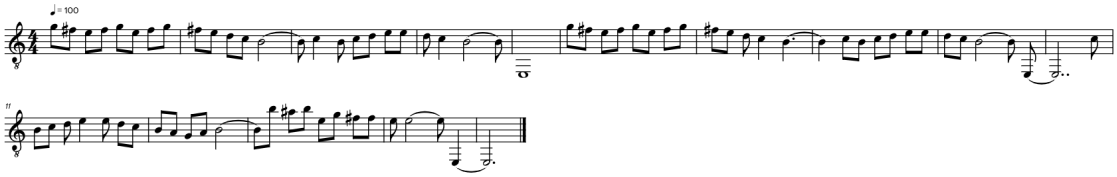
Figure 50: Greek traditional folk song



(a) Octave error near the end of the song.



(b) Several notes wrongly estimated as polyphonic and some wrong notes.



(c) Ground truth score.

Figure 51: "Proino Tsigaro"