The Guitar as a Human-Computer Interface

Marcelo Cicconet

Doctor of Sciences Dissertation

Instituto Nacional de
Matemática Pura e Aplicada

Rio de Janeiro, November 16, 2010

Abstract

In this work we explore the visual interface of the guitar. From the analysis point of view, the use of a video camera for human-computer interaction in the context of a user playing guitar is studied. From the point of view of synthesis, visual properties of the guitar fretboard are taken into account in the development of bi-dimensional interfaces for music performance, improvisation, and automatic composition.

The text is divided in two parts. In the first part, we discuss the use of visual information for the tasks of recognizing notes and chords. We developed a video-based method for chord recognition which is analogous to the state-of-the-art audio-based counterpart, relying on a Supervised Machine Learning algorithm applied to a visual chord descriptor. The visual descriptor consists of the rough position of the fingertips in the guitar fretboard, found by using special markers attached to the middle phalanges and fiducials attached to the guitar body. Experiments were conducted regarding classification accuracy comparisons among methods using audio, video and the combination of the two signals. Four different Data Fusion techniques were evaluated: feature fusion, sum rule, product rule and an approach in which the visual information is used as prior distribution, which resembles the way humans recognize chords being played by a guitarist. Results favor the use of visual information to improve the accuracy of audio-based methods, as well as for being applied without audio-signal help.

In the second part, we present a method for arranging the notes of certain musical scales (pentatonic, heptatonic, Blues Minor and Blues Major) on bi-dimensional interfaces by using a plane tessellation with especially designed musical scale tiles. These representations are motivated by the arrangement of notes in the guitar fretboard, preserving some musical effects possible on the real instrument, but simplifying the performance, improvisation and composition, due to consistence of the placement of notes along the plane. We also describe many applications of the idea, ranging from blues-improvisation on multi-touch screen interfaces to automatic composition on the bi-dimensional grid of notes.

Acknowledgments

I would like to thank Prof. Paulo Cezar Carvalho for guiding me through this research, and Prof. Luiz Velho for the invaluable technical and motivational support. I also thank Prof. Luiz Henrique Figueiredo, Prof. Giordano Cabral, Prof. Jean-Pierre Briot and Prof. Moacyr Silva, for serving on my defense committee and providing all-important comments; and Prof. Marcelo Gattass for the Computer Vision lessons at PUC-Rio.

For the invaluable help during the hard years at IMPA, thank you very much, my colleagues and collaborators: Ives, Julio, Emilio, Augusto, Daniel, Pietro, Fabio, Fernando, Guilherme, Marcelo, Tertuliano, Vanessa, Adriana, Ilana.

Thank you Clarisse, for receiving me in São Paulo for the Brazilian Computer Music Symposium in 2007. Thank you Thiago, for your help with the VISIGRAPP 2010.

I gratefully thank you Italo and Carolina for the friendship, and for the inestimable support in some difficult occasions. Thanks Rafaella, for the same reason.

I thank my parents, Mario and Ivanilde, for getting out of Vorá; and my sister, Franciele, for being a consistent friend during this years of nomad life.

Acknowledgments are also due to IMPA, for the excellent studying environment, and to CNPq, for the financial support.

To those who have not yet found somebody (or something) really special to whom dedicate their work, their time, their lives.

Just keep looking for.

Just keep working.

Contents

Chapter 0

Introduction

In the website of the Association of Computing Machinery [26], we find the following definition for *Human-Computer Interaction* (*HCI*):

> Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.

The times in which this text is being written are particularly interesting for HCI, since computer hardware and software are becoming capable of accepting many non-obvious ways of interaction, approaching a status where humans can communicate information to computers as if they were other humans.

An example is interaction using voice. When contacting a Call Center of certain companies, we are sometimes surprised with a recorded voice posing some questions and expecting voice answers, not a sequence of keyboard commands. Sound input is being also used for systems that recognize the played, whistled, or simply hummed song[1].

More recently it has become possible to (efficiently) use video as well, and some interesting applications have appeared. We can mention, for instance, a portable game console[2] which uses camera face tracking to create three dimensional mirages, and a console game extension[3] which allows playing some games without using controllers, only human gestures.

It is, however, a more conservative (yet novel) interaction paradigm which is currently in vogue, being recently adopted by mainstream-available devices: multi-touch interfaces.

The term multitouch is more commonly applied to touchscreen displays which can detect three or more touches, though it also designates devices able to detect

[1] midomi.com, musipedia.org.

[2] Nintendo DSi.

[3] Kinect for XBox 360.

multiple touches simultaneously but that do not display data.

This is precisely the case of the guitar.

In fact, right out of the box the guitar is only a device for producing sounds; after all it is just a musical instrument. However, once we connect it with a computer, in the sense that the machine is able to understand the language of the musical instrument, we are talking about a human-computer interface, i.e., a device for human-computer interaction.

That is what this work is about.

By seeing the guitar as a tool to interact with computers, we are in the realm of the definition at the beginning of this chapter. The quote mentions four aspects concerning computing systems for human use: design, evaluation, implementation and the study of surrounding phenomena. In this work we have studied the usage of audio, video, and both signals, for guitarist-computer interaction, with special emphasis in the problem of chord identification. We have also explored the related topic of how the guitar interface (more precisely, the fretboard), can guide the design of musical interfaces for traditional multi-touch screen devices.

This text is divided in two parts, Analysis and Synthesis. First, we will essentially talk about the methods used to make the computer understand the scene of a user playing guitar. Then the adaptation of some of the features of the guitar fretboard to other multi-touch interfaces for the purpose of producing (i.e., synthesizing) music will be discussed.

The guitar-computer connection is essentially performed using Machine Learning tools, some of which especifically designed for the problem at hand, modeled according to the way humans themselves perceive a guitarist playing. "*Machine Learning* is programming computers to optimize a performance criterion using example data or past experience" [3]. "One case where learning is necessary is (...) when humans are unable to explain their expertise" [to perform some task].

In fact, problems like chord recognition fit in this category. It can be argued that guitar chords are essentially a set of strings vibrating simultaneously, so that audio signal processing could be used to evaluate the frequencies present in the sound, and, looking up to a table, the chord which better explains the present frequencies could be declared as the one being played. However, many factors may have influence on this task: type of string (nylon or steel), tuning, quality of the instrument, noise, etc. These factors, in view of the high number of possible chords and the subtle differences between some of them, make it reasonable to use methods where the system is *taught* how each chord "looks" like.

Besides, in some situations, that is how humans perform chord recognition. It can not be explained how we perform the task, except perhaps by saying that we have passed many years listening to those chords, so that their patterns have been engraved in our brains through repetition. That is, the chords have been *learned*.

In the case of a particular musical instrument, the process of learning it involves not only the auditory sense. Given the interface of the instrument, each musical element has a visual correspondence associated with. For guitar chords, for instance, that is the visual shape of the hand, which, as a matter of fact, is an information effectively used in teaching/learning situations: any guitar student knows how a particular chord that he/she has learned looks like when performed by a person in front of him/her.

The central question of the first part of this work is to analyze how visual information can be used for guitarist-computer interaction, being the video signal used by itself or in combination with the audio signal. We will show that, indeed, using visual information is worthwhile.

When working with audio or video, separately, we have used standard Supervised Machine Learning methods. To deal with both signals, *Data Fusion* techniques were employed to combine classifiers. This is sometimes called *Ensemble Learning*. In this context, one of the approaches led to a method where Supervised and Unsupervised Machine Learning appear in the same algorithm.

So, to summarize, the first part of this work is about dealing with data. In our case, the data comes from a musical instrument, but the methods discussed could be applied as well in other problems where the goal is to assign the input data (which may come from two different signals) to some class.

On its turn, the second part of this work fits in the area of multi-touch interfaces for music performance, composition, and improvisation.

We will see that many of the performance effects which are possible in the guitar fretboard are realizable in multi-touch screen interfaces as well. Moreover, the properties of these electronic interfaces allow simplifying the playing experience, by presenting more consistent arrangement of musical notes, while keeping the idea of a stringed and fretted instrument. This conforms with the definition at the beginning, as one of the "surrounding phenomena" related to the paradigm of seeing the guitar as a human-computer interface.

Many applications featuring these arrangements of musical notes in multi-touch interfaces are presented. The diversity of applications shows the flexibility and usefulness of the idea of tessellating the plane using musical scale tiles, as it happens in fretted instruments, like the guitar.

The content of the following chapters can be summarized as follows:

Chapter 1. Introduces the guitar, presenting some fundamentals about the physics of vibrating strings.

Chapter 2. Presents techniques for guitarist-computer interaction using audio, video and both signals. The problems of pitch and chord recognition are discussed.

Chapter 3. Describes a representation of musical scales of 5 and 7 notes suited for multi-touch screen devices. It is inspired on the distribution of the

notes of the chromatic scale in the guitar fretboard.

CHAPTER 4. Reports some experiments on automatic music composition algorithms designed for bi-dimensional grids of notes.

CHAPTER 5. Contains final remarks, and points to future directions.

This text comprises both reviews of existing methods and contributions that we have made, mainly to the field of human-computer interaction. The contributions are:

- Method for guitar-chord recognition using video.
- Method for guitar-chord recognition using both audio and video.
- Analysis of necessary conditions to the Product Rule algorithm for Data Fusion to behave as a minimum weighted-sum-of-squared-distances method.
- Analysis of necessary conditions for the Concatenation and the Product Rule algorithms for Data Fusion being equivalent.
- Hierarchical algorithm for combining classifiers, inspired by the method humans themselves use to perform guitar-chord recognition.
- Method for representing musical scales in multi-touch devices.
- Some automatic music composition algorithms and their adaptation for bi-dimensional grids of notes.
- Method for visualizing music pieces, suited for the task of editing medias containing audio.

Additional material related to this work, particularly those which do not fit the text media, can be found at `http://www.impa.br/~cicconet/thesis/`.

Part I

Analysis

Chapter 1

The Interface

1.1 Overview

Let us get started by specifying the musical instrument which is the subject of our work.

According to [38], the [acoustic] guitar (Fig. 1.1) is "a stringed musical instrument with a fretted fingerboard, typically incurved sides, and six or twelve strings, played by plucking or strumming with the fingers or a plectrum". The same reference describes the electric guitar (Fig. 1.2), as being "a guitar with a built-in pickup or pickups that convert sound vibrations into electrical signals for amplification".

There are many types of acoustic guitars (like the folk, the twelve-string and the jazz guitars [1]) as well as of electric guitars (where, besides the shape of the body, the type and location of the pickups are key issues). The strings can be made of nylon or of steel, and many distinct tunings exist. It is also large the number of parts that make up the instrument, especially in the electric version. Fig. 1.3 labels some of the more important parts, in the sense we may refer to them in this text.

Physically speaking, the acoustic guitar is a system of coupled vibrators [19]: sound is produced by the strings and radiated by the guitar body. In electric guitars, the vibrations of the body are not of primary importance: string vibrations are captured by pickups and "radiated" by external amplifiers. Pickups can be electromagnetic (usually located as shown in Fig. 1.3) or piezoelectric (located in the bridge). Piezoelectric pickups are also common in acoustic guitars, eliminating the need of microphones, although microphones capture better the "acoustic" nature of the sound produced.

In this text, when mentioning the term *guitar* we mean a six-string musical instrument, acoustic or electric, with the default tuning: 82.41, 110.00, 146.83,

Fig. 1.1: Acoustic guitar.

Fig. 1.2: Electric guitar.

196.00, 246.94 and 329.63 Hz, from the top to the bottom string, according to the point of view depicted in Fig. 1.3 (where strings are omitted). The mentioned *fundamental frequencies* correspond to MIDI notes E_2, A_2, D_3, G_3, B_3 and E_4, respectively [57]. We also suppose that frets are separated in the fretboard according to the *equally-tempered scale*, a concept to be introduced in Section 1.3.

1.2 Strings

Guitars are stringed instruments. So, regardless of amplification issues, the understanding of the sound it produces is a subject of the theory of *vibrating strings*. The vibration of a string was mathematically modeled back in the 18th century [5]. Therefore, many references about it can be found in the mathematical literature, usually in Partial Differential Equations texts treating the One-Dimensional Wave Equation. Here we will just mention some related facts that we will make use of, later in this text. They were taken from [6], which we recommend as a nice self-contained discussion about the subject.

Let it be

ρ, the mass per unit length
l, the length, and
T, the tension,

of a string fixed at its two ends.

With $c = \sqrt{T/\rho}$, the transverse displacement $y(x,t)$ of a point x on the string satisfies, at time t, the *wave equation*:

$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y}{\partial t^2} \ . \tag{1.1}$$

The general solution of (1.1), with boundary conditions

$$y(0,t) = y(l,t) = 0, \text{ for all } t, \tag{1.2}$$

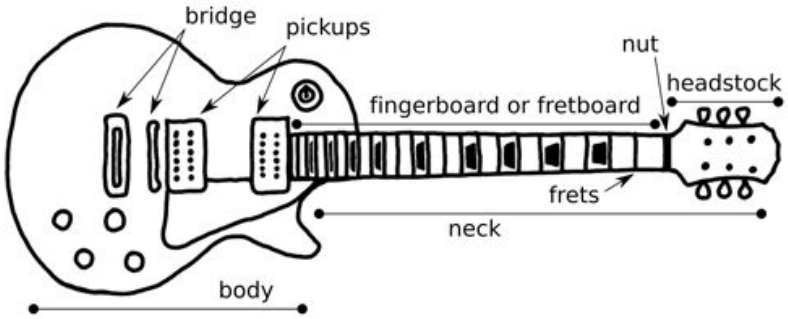


Fig. 1.3: Electric guitar with some parts labeled.

is given by

$$y = \sum_{n=1}^{\infty} \sin \frac{n\pi x}{l} \left(A_n \cos \frac{n\pi ct}{l} + B_n \sin \frac{n\pi ct}{l} \right) , \tag{1.3}$$

where A_n and B_n are constants.

Putting $r_n = \sqrt{A_n^2 + B_n^2}$ and $\theta_n = \tan^{-1}(B_n/A_n)$, the nth term in the sum (1.3) reduces to

$$r_n \sin \frac{n\pi x}{l} \cos \left(\frac{n\pi ct}{l} - \theta_n \right) , \tag{1.4}$$

which, with fixed x, is a periodic function of t, with frequency ν given by

$$\nu = \frac{n}{2l} \sqrt{\frac{T}{\rho}} . \tag{1.5}$$

Summarizing, the string vibrates according to a sum of an infinite number of sinusoidal functions of type (1.4), n ranging in the set of positive integers. Function (1.4), for a particular value of n, is said to be the nth *mode* of vibration of the string. For $n = 1$ we have what is called the *fundamental mode*. The frequency corresponding to the fundamental mode,

$$\nu = \frac{1}{2l} \sqrt{\frac{T}{\rho}} , \tag{1.6}$$

is referred to as the *fundamental frequency* of vibration. For $n > 1$ the corresponding frequencies are known as the *harmonics* or *overtones* of the fundamental.

Equation (1.6) is known as Mersenne's law. It was discovered experimentally by Mersenne in 1636. Here the important aspect of Mersenne's law is the relation between ν and l:

> the fundamental frequency of transverse vibration of a string is inversely proportional to the length of the string.

This fact is the main guiding rule for the arrangement of frets on the guitar fretboard, a subject discussed in the next section.

But before finishing this section we would like to mention that, in order to precisely describe the constants A_n and B_n in (1.3), some initial conditions must be established, besides the boundary conditions (1.2). Those conditions concern knowing

$$y(x, t_0) \text{ and } \frac{\partial y}{\partial t}(x, t_0), \text{ for all } x \in [0, l] \ , \tag{1.7}$$

where t_0 is the initial time. They reflect the exact manner in which the string is plucked.

1.3 Fretboard

The interval between two notes, where the fundamental frequency of the second is twice the fundamental frequency of the first, is called an *octave*, in music terminology [16].

In modern Western musical instruments, the octave is divided in 12 equal-sized *semitones* [36]. This means that there are 12 notes per octave, and that, being f_A and f_B the fundamental frequencies of two consecutive notes, with $f_A < f_B$, the ratio f_B/f_A is a constant, equal to $2^{1/12}$. Indeed, multiplying any reference frequency f_R by $2^{1/12}$ twelve times, we get a note with frequency $2f_R$, i.e., one octave above the note corresponding to the reference note. This is called *equal-temperament*, and the scale of notes so built is known as an *equally-tempered* scale.

Let d be the distance between the bridge and the nut of a guitar[1], and f_R the fundamental frequency of vibration of an open stretched string on that musical instrument. Let us enumerate the frets from the right to the left, according to the point of view of Fig. 1.3.

Then, the fundamental frequency of the note corresponding to the first fret should be $2^{1/12} f_R$ in order to make this note one semitone higher than the note of the open string. According to Mersenne's law (1.6), this means that the distance between the bridge and the first fret should be $2^{-1/12} d$. In general, the distance d_i between the bridge and the ith fret is given by

$$d_i = 2^{-i/12} d \ . \tag{1.8}$$

In particular, $d_{12} = d/2$, i.e., the twelfth fret is halfway from the nut to the bridge, which means the corresponding note is one octave above the note of the open string, as it was expected to be.

[1] d is usually some value around $65cm$ [19].

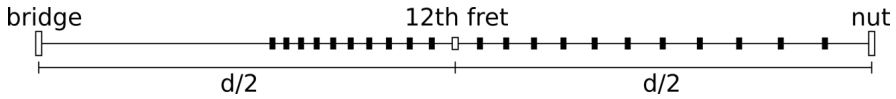


Fig. 1.4: Illustration of the distance between frets on the guitar fretboard.

Fig. 1.4 gives an idea of the separation between frets on the guitar fretboard.

The scale with twelve notes per octave that we have just built is known as the *chromatic scale* [16]. In Western music, it is very unusual for a music piece to be composed using this scale. Music writers normally prefer scales with about seven notes per octave. Those notes are in general a subset of the chromatic scale notes, so the musician just need to memorize where, in the fretboard, are the notes of that subset.

Let us now enumerate the strings of the guitar from the bottom to the top, according the the point of view of Fig. 1.3, and call them $S_1, ..., S_6$. Let f_i be the fundamental frequency of string S_i. The default tuning, mentioned in the end of Section 1.1, is such that, for $i = 6, 5, 4, 2$, we have

$$f_{i-1} = 2^{5/12} f_i \ , \tag{1.9}$$

that is, the note of string S_{i-1} is five semitones above the note of the string S_i. In music terminology, we say, in this case, that the note of S_{i-1} is the *fourth* of the note of S_i. With this tuning the guitar falls in the class of the string instruments *tuned in fourths*[2].

The arrangement of notes on the fretboard is determined by the tuning of the instrument and the location of the frets. An important aspect of string instruments (having more than one string) is the redundancy of notes. That is, the same pitch can be obtained from distinct pairs $(string, fret)$. So, the same sequence of notes could be played in several different ways. In particular, twelve consecutive notes of the chromatic scale can be arranged in different patterns on the fretboard. Those patterns are studied in the second part of this text.

We finish this section discussing the *coordinate system* of the fretboard.

So far the guitar was illustrated as being played by a right-handed person, and seen by a spectator in front of the player (Fig.'s 1.1, 1.2 and 1.3). This point of view is convenient for Computer Vision, where a video camera would play the role of spectator. But the player (more specifically, a right-handed player) sees

[2] What about S_3? That is, why does S_2 is not the fourth of S_3? Well, as of June 25, 2010, the article on Guitar Tunings from Wikipedia [21] states that "standard tuning has evolved to provide a good compromise between simple fingering for many chords and the ability to play common scales with minimal left hand movement".

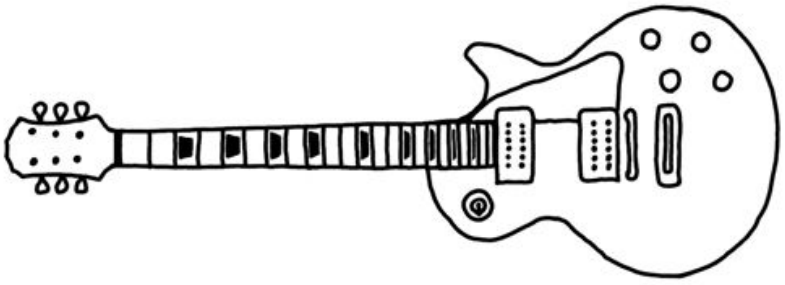


Fig. 1.5: Bi-dimensional representation of the guitar that more closely illustrates the way a right-handed player sees the musical instrument.

the instrument under another point of view. Fig. 1.5 depicts more closely how the guitar is seen by the user[3].

Therefore the point of view of Fig. 1.5 seems to be more appropriate to represent musical information to the player, although it is not the default choice, as can be seen comparing [2] and [12], two popular websites for guitar chords and scales. Another interesting feature of this perspective is the following: given a string, frequency goes up as frets are traversed from left to right; and, given a fret, frequency goes up as strings are traversed from bottom to top. This is in accordance with the canonical coordinate system of the plane.

In this text, the fretboard will be seen like in Fig. 1.3 when using Computer Vision methods to understand the performance of a player, and like in Fig. 1.5 when developing bi-dimensional interfaces for music performance, improvisation and automatic composition.

[3] It is a common practice, in Human-Computer Interaction (HCI) literature, refer to the "human" as the "user", since it is supposed that some computer system is being *used* by a human to perform some task. Since we are considering, in this work, the guitar as a device for HCI, we will sometimes write "user" to mean the person who is playing that musical instrument.

Chapter 2

Interfacing with the Guitar

2.1 Interfacing Using Audio

As mentioned in Chapter 1, in the guitar the vibrations of the strings are commonly captured by means of electromagnetic or piezoelectric pickups [19]. The pickups convert mechanical vibrations into electrical voltages, which are *continuous-time* signals, i.e., *analog signals*. For these signals to be processed by a computer, an *analog-to-digital* conversion must be performed, since computers can handle only *discrete-time* signals [52]. The process of converting a signal from continuous to discrete is called *discretization* [24].

Mathematically, an audio signal is represented by a function

$$f : U \subset \mathbb{R} \to \mathbb{R} \ . \tag{2.1}$$

The independent variable is commonly called t, for *time*, and $f(t)$ is usually some physical quantity (say, the output voltage of a guitar pickup).

Analog-to-digital converters perform two discretizations, one in the domain of f, called *sampling*, and the other in its codomain, called *quantization*. As an example, the sampling frequency used for the standard Compact Disk is 44100 Hz (samples per second) at 16 bits per channel [46]. This means that the *digital* representation of f consists of the values of $f(t)$ for 44100 equally spaced values of t for each interval of one second, and that $f(t)$ can assume only integer values in the range $[-2^{15}, 2^{15} - 1]$.

The choice of 44100 Hz as sample rate is justified by the Sampling Theorem, which states that to represent frequencies up to x Hz, for the purpose of later *reconstruction* of a signal (without loss of information), the sample rate must be of at least $2x$ samples per second [24]. It happens that humans can only listen to sounds with frequencies ranging from about 20 Hz to about 20 kHz [46].

Except otherwise stated, in this text we will work under the assumption that the audio sample rate is 44100 samples per second.

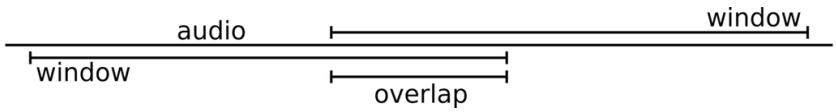


Fig. 2.1: Overlapping windows.

2.1.1 Audio Descriptors

Raw digital sound, i.e., the pure sample values provided by the analog-to-digital converter, carry no significative information *per se*. Each time a chunk of audio arrives, some processing has to be performed in order to obtain a vector representing some property of the sound, sometimes a property related to a physical quantity. Such a vector is called *audio descriptor* (or, sometimes, *audio feature*).

Normally the audio descriptors are obtained from segments (chunks) of constant size, called *windows*, which *overlap* each other, as shown in Fig. 2.1. The size of the window depends on the feature. For sounds sampled at 44100 Hz, in general it ranges from 512 to 4096 frames (samples), which means 11.6 to 92.9 mili-seconds. The overlap is also of variable size, a common choice being half the window size. The distance between the left bounds of two consecutive windows is called *hop size*. It determines the temporal resolution of the extracted feature.

Discrete Fourier Transform

A great number of audio features are based on the *frequency-domain* representation of the signal, i.e., on the coefficients of its Discrete Fourier Transform (DFT). Let $x = (x_0, \ldots, x_{N-1})$ be a discrete signal. The Discrete Fourier Transform of it, $\hat{x} = (\hat{x}_0, \ldots, \hat{x}_{N-1})$, is given by

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n / N} \ . \tag{2.2}$$

The DFT gives the complex amplitudes with which the frequencies from $zero$ to half the sample rate are present in the signal. For example, for a sample rate of 44100 frames per second and a window size of 1024 frames, because of the fact that the DFT of a signal is symmetric around the point of index $(N-1)/2$, the result is 512 values of amplitudes for frequencies equally spaced between $zero$ and 22050. So, increasing the window size does not increases the range of analyzed frequencies, but the resolution in which these frequencies are observed.

However, there is a problem in applying the DFT directly on an audio segment. In the theory behind formula 2.2 there is the assumption that the audio segment

Fig. 2.2: Hann window, audio segment and their pointwise product.

to be processed consists of an entire period of a periodic signal defined all over the line of real numbers ($\mathbb{R}$). So, if x_0 and x_{N-1} are "too different", the DFT will present non-null amplitudes for high frequencies.

To get around this problem one multiplies the segment x by a smooth non-negative function, whose integral is unitary and which usually is equal to $zero$ in the boundaries x_0 and x_{N-1}. Such a function also receives the name of *window*, and the process of pointwise multiplying an audio segment by this function is called *windowing*.

There are many kinds of windows in the literature. A frequently used one is the Hann window (Fig. 2.2), which is defined by

$$h_n = \frac{1}{2}\left(1 - \cos\frac{2\pi n}{N-1}\right) \ , \tag{2.3}$$

for $n = 0, ..., N-1$.

From Equation 2.2 we can see that the computational cost of the straightforward algorithm to compute the DFT is $O(N^2)$, where N is the length of the signal x. Fortunately there are more efficient algorithms, like that of *Cooley-Tuckey*, which uses the divide-and-conquer paradigm to reduce the cost to $O(N \log N)$. Such an algorithm is called Fast Fourier Transform (FFT).

Power Spectrum

The entry $\hat{x}_n$ of the DFT carries magnitude and phase information of the frequency corresponding to index n. However for most of the applications only the magnitude information, $|\hat{x}_n|$, is used. The graph

$$\{(n, |\hat{x}_n|^2) : n = 0, ..., N-1\} \tag{2.4}$$

is called the *power spectrum*, or *spectrogram* of the signal x.

Loudness

This descriptor corresponds to the subjective judgement of the intensity of a sound [31]. In [35] it is defined as the root mean square of the entries of the

power spectrum. Sometimes the value is represented in logarithmic (decibel) scale, to be more closely related with the calibration of the human auditory system and to deal with the wide dynamic range involved [35]. There are some other definitions used in the literature, like the average of the entries of the log-warped power spectrum [31]. Ultimately, the choice of the definition is a matter of engineering, i.e., of choosing the one that provides better results. We have used the logarithmically-scaled sum of the entries of the power spectrum.

Pitch

Pitch is a perceptual attribute whose corresponding physical concept is the *fundamental frequency* (Sec. 1.2), which allows ordering songs in a scale extending from low to high. (In a piano, for example, given two keys, the key to the left sounds lower.) It is defined as the frequency of a sine wave that is matched to the target sound, the match being decided by human listeners [35].

Pitch Class Profile

The Pitch Class Profile (PCP), also known as *chroma*, is a 12-dimensional vector representing the energy distribution of the 12 chromatic-scale notes (regardless of the octave) in the audio window under analysis. The definition that follows is adapted from that in [31].

First, the audio chunk is Hann-windowed, and the power spectrum computed. We define a vector $f = (f_1, ..., f_{120})$ such that

$$f_i = 440 \cdot 2^{\frac{i-46}{12}} \ . \tag{2.5}$$

The entries of f correspond to the frequencies of MIDI note numbers ranging from 24 ($f_1 \approx 32.7$) to 143 ($f_{120} \approx 31608.5$). Let now be $v = (v_1, ..., v_{120})$, where v_i corresponds to the sum of power spectrum entries corresponding to frequencies lying in the interval $(f_{i-\frac{1}{2}}, f_{i+\frac{1}{2}})$ around f_i, weighted[1] by some gaussian-like function centered in f_i. Then the PCP, $x = (x_1, ..., x_{12})$, is defined by setting

$$x_i = \sum_{j=1}^{10} v_{12(j-1)+i}. \tag{2.6}$$

Audio descriptors are a central issue in the area of *Music Information Retrieval* (MIR). In the context of guitarist-computer interaction, they are mainly used for

[1] The impact of the distance between the frequencies of the DFT bins and the frequencies of the MIDI notes in the PCP computation has been analyzed in [9].

pitch and chord recognition. We will talk about these subjects in the following subsections. But before, let us (literally) illustrate them by showing an application at which most of the mentioned audio descriptors are combined for the purpose of visualizing musical information.

The musical content of a piece can be essentially divided in two categories: the harmonic (or melodic) and the rhythmic (or percussive). Well, rigorously speaking it is impossible to perform such a division, since melody and rhythm are (sometimes strongly) correlated. However, (respectively) *chroma* and *loudness* can be used with great effect to represent those categories in some contexts, as for providing a representation of a music piece which helps visually segmenting it. That would be useful, for instance, in audio and video editing tasks.

Given a song (music piece), we start by computing its sequence of chroma and loudness feature vectors.

The sequence of loudness values is then normalized to fall in the range $[0, 1]$, and warped logarithmically according to the equation

$$x \mapsto \frac{\log(x + c) - \log c}{\log(1 + c) - \log c}\ , \tag{2.7}$$

where $c > 0$ is an offset parameter[2]. The logarithmic warp is important because the human auditory and visual systems are roughly logarithmically calibrated.

The elements of each chroma vector are normalized to the range $[0, 1]$, to avoid taking into account differences of loudness in different windows, and squared, to give more importance to the peak value, highlighting the melodic line.

We arrange the chroma vectors $c = (c_1, ..., c_{12})$ as columns side by side in a matrix, the bottom corresponding to the pitch of C. The entries of each vector are associated with a color (h, s, v) in the HSV color space, where the value c_i controls the hue component h as follows: supposing the hue range is $[0, 1]$, we make $h = \frac{2}{3}(1 - c_i)$, so the color ranges from blue ($h = \frac{2}{3}$) to red ($h = 0$), linearly, when c_i ranges from 0 to 1. We set $s = 1$ and $v = l_c$, where l_c is the loudness value corresponding the chroma vector c.

Each vector (column) is then warped vertically (in the sense of image warping), having the middle point as pivot. The warping is such that the height h_c of the vector ranges from α to 2α, where α is some positive constant not smaller then 12 pixels. More precisely, h_c is given by $h_c = (1 + l_c)\alpha$.

Fig. 2.3 shows the described representation at work. The waveform and the audio features used are presented as well.

In the example of Fig. 2.3, besides the many segmentation points presented, it is also possible to guess what portions of the picture corresponds to the chorus

[2] We have used $c = 0.1$.

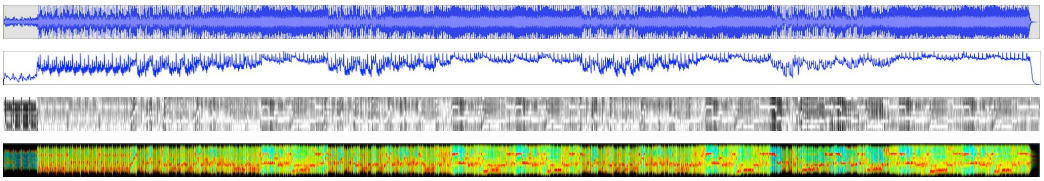


Fig. 2.3: Waveform (top), loudness, chroma, and the combination of loudness and chroma as described in the text, for the the song Sex on Fire, by Kings of Leon (RCA Records, 2008).

of the song. This kind of "visual segmentation" is, in general, more difficult to achieve when we use the traditional *waveform* representation.

Details of the audio-information visualization method just described can be found in [13].

2.1.2 Pitch Recognition

The concepts of musical note and fundamental frequency (also known as F_0) are closely related. When a digital device renders a sine wave with frequency, say, 440 Hz, the human ear perceives a musical note which happens to be called A4 (the central piano key of A). However, when we strike, at the piano, that key, or when the note of the fifth fret of the first string of the guitar is played, the power spectrum of the audio shows not only a peak in the frequency of 440 Hz, but also in frequencies corresponding to the multiples of 440. All these frequencies, and their corresponding amplitudes, are the elements which make a given musical instrument sound particular. In the case described, 440 Hz is the fundamental frequency of the note because it is the frequency such that their multiples better explain the spectral content of the signal [47].

There are many algorithms for F_0 recognition in the literature, but essentially two categories, depending on the domain of work: time or frequency. We will now describe some of them.

Cross-Correlation

Given an audio segment $x = (x_0, ..., x_{N-1})$, the *cross-correlation* c at the point k is a measure of how much the signal $(x_0, ..., x_{N-1-k})$ is similar to the signal $(x_k, ..., x_{N-1})$. Formally, $c_k = \sum_{n=0}^{N-1-k} x_n x_{n+k}$, for $k = 0, ..., N-1$.

So the straightforward algorithm for computing the cross-correlation is $O(N^2)$. However, using the Fast Fourier Transform algorithm and the *circular convolution*

(a) (b) (c) (d) (e)

Fig. 2.4: (a) Signal and its corresponding cross correlation (b), McLeod-normalization (c), difference function (d) and YIN-difference function (e).

theorem[3], it is possible to reduce that complexity to $O(N \log N)$.

The algorithm is simple: define $\bar{x} = (\bar{x}_0, ..., \bar{x}_{2N-1})$ so that $\bar{x}_k = x_k$ for $k = 0, ..., N-1$ and $\bar{x}_k = 0$ otherwise. Compute the DFT of $\bar{x}$, obtaining $\hat{\bar{x}}$. Then, compute the IDFT[4] of $(|\hat{\bar{x}}_0|^2, ..., |\hat{\bar{x}}_{2N-1}|^2)$. The real part of the kth entry of the result will be c_k. Details of this algorithm can be found in [53].

The maximum of the cross-correlation function is c_0 and, in the ideal case (i.e., for waves with no harmonics), the index of the second highest peak determines the *period* T of the wave, from which the fundamental frequency f can be obtained, using the relation $f = 1/T$. This method works well for some "real-world" sounds, like guitar-string sounds, for instance. But for more general use some kind of normalization must be performed, as in the next method.

McLeod's Method

The method of McLeod [39] is based on the cross-correlation function defined in the previous subsection, but includes some heuristics to circumvent the problems which appear when there is more energy in one of the harmonics of the signal instead of in the fundamental.

Let us consider, for example, the signal $x_n = \sin(2\pi \cdot 2n) + \sin(2\pi \cdot 4n) + 3\sin(2\pi \cdot 6n)$ in the interval $[0, 1]$ (Fig. 2.4(a)). The fundamental frequency is 2, but there is high energy in the frequency 6 (second harmonic). In this case, the second highest peak of the cross-correlation function (Fig. 2.4(b)) will not correspond the fundamental frequency.

The normalized cross-correlation function proposed in [39] is

$$c_k = \frac{2\sum_{n=0}^{N-1-k} x_n x_{n+k}}{\sum_{n=0}^{N-1-k} x_n^2 + x_{n+k}^2} \ , \tag{2.8}$$

[3] The convolution between two signals is equal to the Inverse Fourier Transform of the product of the Fourier Transform of them.

[4] Inverse Discrete Fourier Transform. The IDFT of a signal $(y_0, ..., y_{N-1})$ is given by $\check{y}_n = \frac{1}{N}\sum_{k=0}^{N-1} y_k e^{2\pi i k n/N}$.

which, in the mentioned case, would be more adequate for the procedure of taking the frequency corresponding to the second highest peak (Fig. 2.4(c)).

The McLeod algorithm is as follows. First, compute the normalized cross-correlation function (Equation 2.8). Then the *key maxima* should be found. They are the local maxima of some intervals, such intervals having left-boundary in a zero-crossing with positive slope and right-boundary in the subsequent zero-crossing with negative slope. The first of such maxima above certain threshold (given by a fraction of the highest peak) is taken to determine the fundamental frequency.

YIN Method

The idea of the YIN [11] method is similar to the previous, but instead of looking for a peak in a cross-correlation function, we look for a valley of a *difference function*.

Let us consider the following difference function:

$$d_k = \sum_{n=0}^{N-1} (x_n - x_{n+k})^2 \ , \tag{2.9}$$

for $k = 0, ..., N-1$. Their local minima correspond to indices k such that the window with the respective shift is more similar to the window without shift than those corresponding to the indices $k-1$ or $k+1$. Fig. 2.4(d) shows the difference function of the signal in Fig. 2.4(a).

The method described in [11] uses the following normalized version of d_k:

$$\bar{d_k} = 1_{[k=0]} + 1_{[k\neq 0]} \frac{d_k}{\frac{1}{k}\sum_{j=1}^{k} d_j} \ , \tag{2.10}$$

where $1_{[A]}$ is 1 resp. 0 if the sentence A is true resp. false. Fig. 2.4(e) shows an example, for the signal of Fig. 2.4(a).

In the YIN method the fundamental frequency will correspond to the value k such that $\bar{d_k}$ is a local minimum of the function (2.10) bellow a certain threshold (greater than zero).

HPS Method

Let us say that the fundamental frequency of a signal is 100 Hz, and that the audio has many harmonics, i.e., non-zero energy for frequencies 200 Hz, 300 Hz, 400 Hz e so on. In an ideal case the energy corresponding to other frequencies would be zero. However, the signal can have other strong *partials*, like, for instance, for the frequency of 90 Hz. But the probability of the energy in the

Fig. 2.5: Comb function, Hann window, and their convolution.

other integer multiples of 90 Hz being high is small. In this case, being $E(f)$ the energy corresponding to the frequency f, the product $\prod_{j=1}^{5} E(100j)$ should be greater than $\prod_{j=1}^{5} E(90j)$.

That is the idea of the HPS (Harmonic Product Spectrum) method [17]. Being R the number of factors to be considered (usually $R = 5$), and $\hat{x}_k$ the kth entry of the DFT of the signal x, for k between zero and the index corresponding to the frequency of $22050/R$ we compute $h(k) = \prod_{r=0}^{R-1} |\hat{x}_{(k+1)(r+1)-1}|$, and take as F_0 the frequency corresponding the the $\bar{k}$ such that $h(\bar{k}) = \max_k h(k)$.

The problem of this method is the resolution of the DFT. If F_0 is 80 Hz but the resolution of the DFT doesn't allow precisely evaluate the frequencies near this value and its integer multiples then the product $\prod_{j=1}^{5} E(80j)$ may not be the highest between all products computed by the HPS. We can deal with this problem by *zero-padding* the audio window, at the expense of increasing the computational cost of the algorithm.

Maximum Likelihood Method

In this algorithm (described in [17]) a database with the so called "ideal spectra" is created and, given the spectrum of the wave of which we want to know the fundamental frequency, we look in the database for the nearest spectrum (according to the Euclidian norm), and the corresponding F_0 is returned.

For a given F_0, an ideal spectrum (Fig. 2.5(right)) is built from a comb function (Fig. 2.5(left)), with peaks in the fundamental and the corresponding harmonics, convolved with a kernel like the Hann window, for instance (Fig. 2.5(center)).

Obviously the database should be large enough to cover all possible musical notes we want to test. In the case of the piano, for example, it has to contain an ideal spectrum for each key. This method works better for instruments which produce a discrete range of musical notes, like the flute and the piano. But for the guitar the method would have problems with bends and vibratos.

2.1.3 Chord Recognition

According to [10], most of the audio-based chord recognition methods are variations of an idea introduced in [23]: the PCP audio feature is used along with

some Supervised Machine Learning method. (Details of the Machine Learning methods we will use hereafter can be found in the Appendix B of this text.)

Here is a brief description of how it works. Let us suppose we want the system to recognize chords $c_1, ..., c_M$. In the training phase N Pitch Class Profile samples from each of the chords are captured, and in the classifying phase the incoming PCP is associated with one of the c_j according to some supervised Machine Learning method.

In the seminal work [23], the Nearest Neighbor method is used, and the machine is trained with "ideal" chroma vectors: those whose entries are 1s in the notes of the chord and 0s otherwise.

2.2 Interfacing Using Video

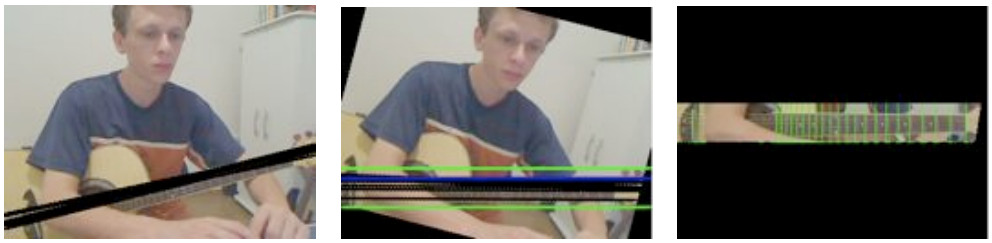
There are still few Computer Vision approaches in the area of guitarist-computer interaction. Here we will cite two recent works on the subject. For references regarding Computer Vision terms that will appear along the text, we recomend [20] and [7].

In [8] a camera is mounted on the guitar headstock in order to capture the first five frets. The Linear Hough Transform is used to detect strings and frets, and the Circular Hough Transform is used to locate the fingertips. The system has also a module for movement detection. The idea is to use the Hough transforms only when the hand is not moving. The purpose is to identify chords and notes sequences in real-time by detecting the fingertips positions in guitar fretboard coordinates. So the system does not use Machine Learning tools.

The work of [34] is more ambitious. They use stereo cameras and augmented reality fiducial markers to locate the guitar fingerboard in 3D, and colored markers (with different colors) attached to the fingertips to determinate their three-dimensional position relative to the fretboard. They apply a Bayesian classifier to determine the color probabilities of finger markers (to cope with changes in illumination) and a particle filter to track such markers in 3D space. Their system works in real-time.

In the beginning of this research, we have tried to capture the necessary information from the scene of a user playing guitar without using special artifacts on the guitar and on the hands of the guitarist.

We started by trying to segment the region of the strings, and locate the frets, using methods for edge detection (see Fig. 2.6). Roughly speaking, the pipeline is as follows. First, the linear Hough Transform was used to locate straight lines (Fig. 2.6(a)); lines with length above a certain threshold would be the strings. From the histogram of the slopes of the found lines, the image is rotated to make the direction of the strings horizontal, and a first crop of the region containing



(a) (b) (c)

Fig. 2.6: Locating the region of the strings, and the frets, using edge-detection techniques.

the strings can be performed (Fig. 2.6(b)). After this, the Sobel x-derivative filter is applied, highlighting the frets. Summing the Sobel image along columns leads to a curve where higher peaks are expected to correspond to frets. At this point, equal temperament properties (and the number of frets) can be used to estimate the location of the frets (Fig. 2.6(c)).

The problem with the described approach is that it is very time consuming and unstable. The Sobel image, for example, is very noisy, so that the along columns sum does not properly maps frets to peaks. This is the reason why we decided to use artifacts attached to the guitar and to the guitarist's fingers.

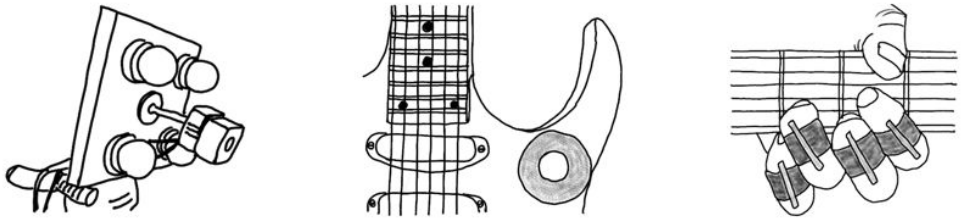
2.2.1 Pitch Recognition

Recognizing the pitch of a single note played in a guitar using video seems not to make sense, because pitch is an audio feature (see Subsection 2.1.1). However, if we know the tuning of the instrument and the pair $(fret, string)$ which is touched, then the pitch can be easily inferred.

The difficult part of this method is knowing if a finger positioned over a particular pair $(fret, string)$ is *effectively* touching the string. For this purpose, 3D information must be used, and the precision of the system must be very high. As mentioned, 3D information is captured in [34], but the authors remarked the problem of accuracy.

Besides, knowing that a string is in fact being pressed in a particular fret is a necessary but not sufficient condition for a video-based pitch recognition method to output the correct result: the string must be played, for there is no pitch without sound. So the system should also be able to see which string is played, which, again, requires high capture accuracy.

Using audio is a natural way of getting around these problems. In fact there have been some studies on the subject, which we will mention in Section 2.4.



(a) (b) (c)

Fig. 2.7: Capture hardware: (a) infra-red camera surrounded by four infrared light sources, (b) hollow disk made with retro-reflexive material, four of which are used to locate the plane containing the ROI, and (c) middle-phalange gloves with small rods coated so as to easily reflect light.

In what follows, we will describe a method which uses video for the problem guitar-chord identification.

2.2.2 Chord Recognition

It is natural to adapt the audio-based method for chord-recognition described in Section 2.1 to a video-based method. Essentially, we keep the Machine Learning part and replace the audio descriptor by a visual feature, which is the "visual shape" of the chord.

Let us define the Region of Interest (ROI) in the scene of a person playing guitar as being the region including the strings, from the nut to the bridge.

To simplify the capture process, avoiding the overhead of segmenting the ROI, we chose to work in the infrared-light range.

Fig. 2.7 shows the equipment that supports our method. We use a infrared camera to capture the scene, which is properly illuminated with infrared light. Special markers (fiducials) are attached to the guitar in order to easily locate the instrument, and for the fingers, reflexive gloves dress the middle phalanges.

The visual-feature extraction pipeline is illustrated in Fig. 2.8. The developed software takes advantage of some nice and robust algorithms implemented in OpenCV, an open-source Computer Vision library [7].

First, a threshold is applied to the input image, so that the only non-null pixels are those of the guitar and finger markers. Then, using the contour detection algorithm and contour data structure provided by OpenCV, guitar and finger markers can be separated. Note that guitar fiducials and finger markers are, respectively, contours with and without a hole. Once the positions of the four guitar fiducials are known in the image, by using their actual positions in

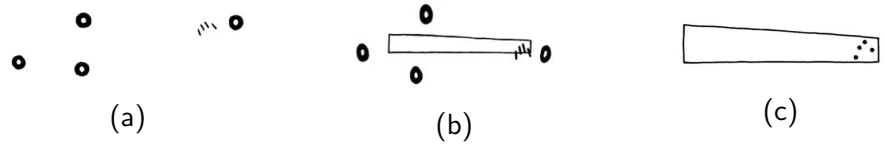


Fig. 2.8: Feature extraction pipeline: (a) a threshold is applied to take only guitar and finger markers, using a contour detection algorithm; (b) a projective transformation "immobilize" the guitar, regardless the movement caused by the musician; (c) the projective transformation is applied to the north-most extreme of finger rods in order to roughly locate the fingertips in guitar-fretboard coordinates.

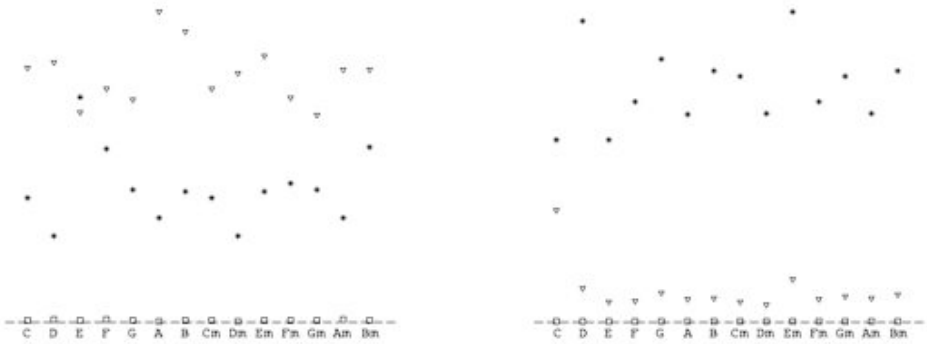
guitar fingerboard coordinates a projective transformation (homography) can be determined and applied in order to "immobilize" the guitar and easily extract the ROI. This homography is then applied to the north-most extreme of the finger rods, so we get the rough position of fingertips in guitar fretboard coordinates, since the distal phalanges are, in general, nearly perpendicular to the fingerboard.

However, in fact we do not use the precise coordinates of fingertips. Instead we apply a Supervised Machine Learning technique to train the machine with the guitar chords we want it to identify. The chord a musician plays is viewed by the system as an eight-dimensional vector composed by the coordinates (after projective transformation) of the four fingertips, from the little to the index finger. By analogy with the PCP, we call this eight-dimensional vector the Visual Pitch Class Profile (VPCP).

Summarizing, the proposed algorithm for guitar chord recognition has two phases. In the first (the training phase), the musician chooses the chords that must be identified and takes some samples from each one of them, where by sample we mean the eight-dimensional vector formed with the positions of the north-most extreme of the finger rods, i.e., the VPCP. In the second (the identification phase), the system receives the vector corresponding to the chord to be identified and classifies it using some Supervised Machine Learning method, like the K Nearest Neighbor, for instance.

2.3 Audio Versus Video

Before talking about quantitative comparisons, let us address some theoretical aspects. Video methods, even knowledge-based, are immune to wrong tuning of the instrument. Despite not being desirable to play a wrongly tuned instrument, this feature is good for beginners, that are not able to have a precisely regulated guitar. On the other hand, it can be argued that knowledge-based methods only



Audio Video

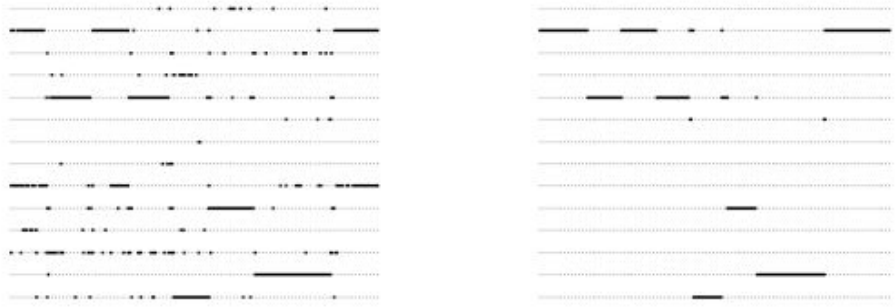
Fig. 2.9: Analysis of the audio and video sample clusters. A square (respectively, a triangle) represent the average (respectively, the maximum) distance between the class samples and the class mean vector. The asterisk represent the distance between the cluster mean vector and the nearest cluster mean vector. This shows that the clusters of video samples are better defined relatively to those from audio samples.

work properly when trained by the final user itself, since the shapes of some given chord are slightly different from person to person. This is a fact, but the knowledge-based techniques using audio data also have to face with this problem, since different instruments, with different strings, produce slightly different songs for the same chord shape.

Seeking quantitative comparisons, we took 100 samples from each one of the 14 major and minor chords in the keys of C, D, E, F, G, A, B, choosing just one shape per chord (in the guitar there are many realizations of the same chord). The video samples were taken by fixing a given chord and, while moving a little bit the guitar, waiting until 100 samples were saved. For the audio samples, for each chord we recorded nearly 10 seconds of a track consisting of strumming in some rhythm keeping fixed the chord. The audio data was then pre-processed in order to remove parts corresponding to strumming (where there is high noise). Then, at regular intervals of about 12 milliseconds an audio chunk of about 45 milliseconds was processed to get its Pitch Class Profile, as described in Section 2.1.

These audio and video samples tend to form clusters in $\mathbb{R}^{12}$ and $\mathbb{R}^{8}$, respectively. Fig. 2.9 provides some analysis of them. Note that in both cases the samples are placed very close to the mean of the respective cluster, but there are more outliers in the audio data.

Regarding classification performance, both methods behaved similarly in the tests we have conducted. The difference is that the audio-based algorithm is less



Audio Video

Fig. 2.10: The same chord sequence, played twice, is analyzed by the traditional audio-based algorithm (Section 2.1) and the video-based method described in Section 2.2. While the former needs some extra processing to cope with the noise caused by strumming, the video-based method is immune to that. However, both techniques have problems with chord transitions.

robust, partly because of the noise caused by strumming not being completely removed. Of course the video-based method is not prone to such kind of noise. This is illustrated in Fig. 2.10, where the same chord sequence (played twice) was performed and analyzed by the two methods, using 20 Nearest Neighbors for classification. Note how the video-based method is more stable. It can also be seen that both algorithms have problems with chord transitions.

2.4 Interfacing Using Audio and Video

Humans make extensive use of visual information to identify chords when someone else is playing, not by precisely detecting fingertips positions in the guitar fingerboard, but by roughly identifying the shapes of the hand and associating them with known chords. This fact is the main motivation of the guitar-chord recognition method described in Section 2.2. Of course in some cases it is very hard to distinguish chords visually, an example being the chords $Dmaj$ and $Dsus4$ (Fig. 2.11). But once we recognize the hand shape, its easy to separate the chords by hearing how they sound.

In this Section we investigate the use of visual information in cooperation with audio methods for guitar-chord recognition.

Six different algorithms are compared, ranging from the purely audio-based (Section 2.1) to the analogous video-based method (Section 2.2), passing through hybrid methods, in which we explore different *Data Fusion* techniques: feature

fusion (i.e., concatenation of audio and visual features), sum and product rules, where likelihoods computed from the signals are summed (respectively, multiplied) before maximization, and a Bayesian approach, where video information is used as *prior* (in Bayesian Theory terminology), this way resembling humans chord recognition strategy, as mentioned before.

Data Fusion is the main aspect of this section. Fusion of audio and video information is a recent approach in the subject of guitarist-computer interaction. We now cite two works where this strategy has been applied.

In [49], the video information helps solving the ambiguity regarding which string was actually fingered or plucked once the fundamental frequency of the played note is known, via audio. In real-time, the guitar is detected using edge methods, and a skin recognition technique is applied to roughly locate the position of the hand relatively to the fretboard.

The same idea is used in [44], but their system is not designed to work in real-time. In the first video frame, the Linear Hough Transform is applied to segment the guitar from the scene, and after the image is rotated so that the guitar neck becomes horizontal, edge methods are used to locate the fretboard. After that, tracking of the fretboard points in the video is done by means of the Kanade-Lucas-Tomasi (KLT) algorithm. The hand position is determined via skin color methods.

In what concerns audio and video cooperation, the mentioned methods are essentially based on heuristics. By putting the bimodal chord recognition problem in the realms of Data Fusion and Statistical Learning, we can make use of some mathematical tools those fields provide.

2.4.1 Data Fusion

Data Fusion consists, as the name suggests, of the combination of two of more sources of information in order to infer properties of the system under analysis. Such information can be, for instance, raw data from some sensor, or even data derived from sensory data [40]. That is why Data Fusion is sometimes called Sensor Fusion, or Multi-Sensor Data Fusion.

In our case there are two sensors, a video camera and an audio analog-to-digital interface, and we will be processing data derived from sensory data, namely the PCP and VPCP vectors.

In the following we describe some Data Fusion strategies that were investigated in the experiments we conducted.

Feature Fusion

This is the simplest Data Fusion approach. Given a PCP sample $X = (x_1, ..., x_{12})$ and a VPCP sample $Y = (y_1, ..., y_8)$, we define

$$Z = (z_1, ..., z_{20}) := (x_1, ..., x_{12}, y_1, ..., y_8) = (X, Y)$$

so training and classification is performed on the concatenated vector Z.

Although simple, there is a small catch. X and Y might be at different magnitude scales, causing situations like this: a shift from $X = x$ to $X = x + d$ in the PCP space could lead the audio-based Machine Learning algorithm to (correctly) change from class c_i to class c_j, but might not lead the fusion-based method to do the same if the shift in Y would not be large enough.

To cope with this issue, statistical normalization must be performed on the training sets before concatenation. Let $\{w^1, ..., w^P\}$ be a set of samples from some gaussian distribution, and μ (respectively, Σ) the estimated mean (respectively, covariance matrix). Being $\Sigma = VDV^\top$ the Spectral Decomposition of Σ, we define $T := D^{\frac{1}{2}}V^\top$, and the normalization as the mapping ν such that $\nu(w) = T \cdot (w - \mu)$. This way the mean (respectively, covariance matrix) of $\{\nu(w_1), ..., \nu(w_P)\}$ is 0 (respectively, I_P, the $P \times P$ identity matrix).

So, given sample sets $\{x^1, ..., x^P\}$ and $\{y^1, ..., y^P\}$, with corresponding normalization mappings ν_X and ν_Y, the Machine Learning algorithm is trained with the sample set $\{z^1, ..., z^P\}$, where $z^i = (\nu_X(x^i), \nu_Y(y^i))$. The same concatenation is performed before classification when fresh samples x and y arrive to be classified.

The drawback of feature concatenation is the well known *curse of dimensionality*: the larger the dimension of the training samples, the more complex the system, and the larger the number of samples necessary for the estimators to be accurate [3].

Sum and Product Rules

The previously defined fusion strategy is said to be *low level*, since data is combined before the analysis is applied. There are also the *high level* fusion methods, where classifiers are applied on data coming from different sources and their results are combined somehow. For example, we could set as the true answer the one from the classifier with the least uncertainty for the given input. Finally, in *middle level* methods the combiner does not use the final answer of different classifiers to make the final decision, but instead some intermediary by-product of them: likelihoods, for instance.

That is the case when we use the sum and product rules. Let's say $p(X = x|C = c_j)$ and $p(Y = y|C = c_j)$ are the likelihoods of the chord being

$C = c_j$ given that the PCP vector is $X = x$ and the VPCP is $Y = y$, respectively. We now define

$$p_{X,j}(x) := p(X = x|C = c_j) \ , \tag{2.11}$$
$$p_{Y,j}(y) := p(Y = y|C = c_j) \ . \tag{2.12}$$

The *sum rule* combiner states that the chord is $C = c_{\hat{j}}$ when

$$p_{X,\hat{j}}(x) + p_{Y,\hat{j}}(y) = \max_{j=1,...,M} p_{X,j}(x) + p_{Y,j}(y) \ , \tag{2.13}$$

and the *product rule* would say that if

$$p_{X,\hat{j}}(x) \cdot p_{Y,\hat{j}}(y) = \max_{j=1,...,M} p_{X,j}(x) \cdot p_{Y,j}(y) \ . \tag{2.14}$$

It should be mentioned that the product rule arises naturally when maximizing the *posterior probability*

$$P(C = c_j|X = x, Y = y) \tag{2.15}$$

over $j = 1, ..., M$ supposing conditional independence between X and Y given c_j (for all j) and assuming that the *priors* $P(C = c_j)$, for $j = 1, ..., M$, are equal[5]. Equation 2.15 reads as "the probability of the chord C to be c_j given that the PCP a and VPCP features are equal to x and y, respectively".

Bayesian Approach

Here is another middle level data fusion approach. It is based on the method humans sometimes use to figure out what is the chord being played.

As an example, let us imagine the following simple situation. A (relatively) musically-trained person, P_A, in front of a guitarist, P_B, is asked to decide what chord he is playing. Both know that the only possible chords are $Emaj$, Em, $Dmaj$ and $Dsus4$ (as shown in Fig. 2.11). If P_B chooses $Emaj$ or Em, by visual inspection P_A will know that it may be $Emaj$ of Em, but he will need to listen how the chord sound to decide which one it is. Analogous situation applies when P_B chooses $Dmaj$ or $Dsus4$.

[5] The question of conditional independency is not easily verifiable. Intuitively, given a chord c_j, a small shift in the position of the hand along the strings does not cause a change in the audio information (as long as the fret-border is not exceeded). We have inspected the estimated covariance of (X, Y) given a particular chord, and have seen that, for most of the analyzed chords, it is relatively small. Anyway, in general, uncorrelation does not imply independency. It would imply if we knew the data were normally distributed [4]. However, for high-dimensional spaces as in our case, this property is difficult to verify as well.

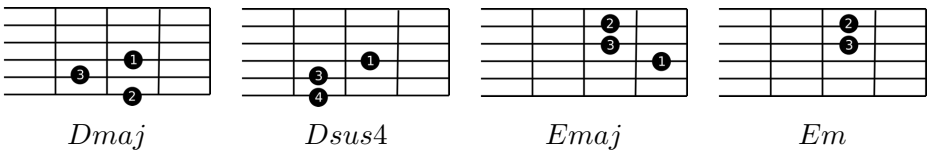


Fig. 2.11: The visual shapes of the hand for chords $Dmaj$ and $Dsus4$ are very similar, but their sounds are easily separated by the trained ear. The same occurs for the chords $Emaj$ and Em.

What is happening is that the visual signal is providing some *prior* information about chord classes. In the example, our visual system easily identify the E-cluster (containing $Emaj$ and Em) and the D-cluster (containing $Dmaj$ and $Dsus4$).

A question that arises is: given a set of chords to be recognized, how do we know what subsets will form visual clusters? The answer is that the system will find them via clustering, and since in this situation the purpose of using video is to enhance audio-based methods, even a small number of clusters would lead to accuracy improvements in the cooperative approach.

The details of the method are as follows.

Each time a pair of samples $X = x$ and $Y = y$ arrives, we will want to find the chord c_j that maximizes the *posterior probability* expressed by Equation 2.15.

According to Bayes rule, Equation 2.15 is equal to

$$\frac{p(X = x|C = c_j, Y = y)P(C = c_j|Y = y)}{\sum_{l=1}^{M} p(X = x|C = c_l, Y = y)P(C = c_l|Y = y)} , \tag{2.16}$$

where p represents a density function, and P a probability (more precisely a *prior* probability).

Therefore we have an optimization problem equivalent to

$$\max_{j=1,...,M} p(X = x|C = c_j, Y = y)P(C = c_j|Y = y) . \tag{2.17}$$

Let's suppose Y assumes discrete values (corresponding to the visual clusters we have discussed), say $y_1, ..., y_L$. The problem 2.17 would than reduce to

$$\max_{j=1,...,M} p(X = x|C = c_j, Y = y_k)P(C = c_j|Y = y_k) , \tag{2.18}$$

where[6]

$$p(X = x|C = c_j, Y = y_k) = p(X = x|C = c_j) , \tag{2.19}$$

[6] In Equation 2.19 we are supposing that the information $Y = y_k$ does not contribute with the knowledge about the distribution of $X|C = c_j$.

which can be modeled as a gaussian density function, with mean and covariance estimated at the training phase using the samples taken from chord c_j, evaluated at the point x. The value of $P(C = c_j|Y = y_k)$ can be estimated as the quotient between the number of training points from the chord c_j in the cluster y_k and the total of training points in the cluster y_k.

Summarizing, the algorithm is as follows:

- Training phase
  - Discretize Y using some clustering method (K Means, for instance).
  - Estimate the *priors* $P(C = c_j|Y = y_k)$.
  - Estimate the distribution of $X|C = c_j$.
- Classification phase
  - Find y_k.
  - Compute the likelihoods $p(X = x|C = c_j)$.
  - Maximize the product $likelihood \cdot prior$ over the set of possible chords.

We notice that, by discretizing Y, the method becomes hierarchical. In fact, once $Y = y_k$, chords that do not have representatives in the video-cluster y_k will not be considered (at least when we compute $P(C = c_j|Y = y_k)$ as mentioned).

2.4.2 Experimental Results

There are many scenarios of guitarist-computer interaction. In a game application, for instance, where the player is asked to follow some chord sequence displayed on the screen, and such a sequence is based on a music piece, the system could save computational effort by training only with the usually small set of chords that it would need to identify. Now, in an improvisation scenario, where the musician choses a chord sequence at random for the purpose of, say, control some automatic composition algorithm, it would be better to train the machine with the greatest possible set of chords.

We investigated these two scenarios in our experiments, by using as training set samples from a small (6) and a large (49 and 168) number of chords. Two sample sets were captured: one with samples from 49 chords (C, D, E, F, G, A and B with variations $maj, m, aug, dim, maj7, m7$ and 7) and the other with samples from 168 chords ($C, C\#, D, D\#, E, F, F\#, G, G\#, A, A\#$ and B with three realizations of each maj, m and 7 variations, two realizations of the $m7$ variation and one realization of each $maj7$, dim and aug variations).

We took 700 PCP samples and 500 VPCP samples from each one of the chords. The number of VPCP samples is smaller due to its smaller inter-cluster distance-to-centroid variation when compared to the PCP samples (see Fig. 2.9). For each chord, 200 samples were used as test set, and the remaining for training. In the case where the number of training samples must be the same (such as

Method	Accuracy			
	6 Chords (P1)	6 Chords (P2)	49 Chords	168 Chords
Audio	0.9958	0.9625	0.7699	0.7065
Video	1.0000	1.0000	0.9380	0.9927
Concatenation	1.0000	1.0000	0.9796	0.9932
Sum	0.9833	0.9983	0.8941	0.9528
Product	1.0000	1.0000	0.9781	0.9928
Bayes: 2 Clusters	0.9458	0.9817	–	–
Bayes: 3 Clusters	0.9939	0.9861	–	–
Bayes: 5 Clusters	–	–	0.8409	0.8062
Bayes: 10 Clusters	–	–	0.8656	0.8344
Bayes: 15 Clusters	–	–	0.8816	0.8472
Bayes: 20 Clusters	–	–	0.8988	0.8656

Table 2.1: Accuracies of methods when trained on 6, 49 and 168 chords. Results corresponding to Bayes rows are averages of results obtained from three independent executions of the algorithm. Chord progression P1: C, Dm, Em, F, G, Am. Chord progression P2: G, Am, Bm, C, D, Em. Samples for P1 and P2 were taken from the 49-chords sample set. The 49- and 168-chords sample sets were independently collected.

in the Feature Fusion method), the VPCP clusters size were augmented via *bootstrapping* [3].

Table 2.1 shows the obtained results. The accuracy is the quotient between the number of correct answers and the total number of test samples. Audio and Video methods correspond to the Maximum Likelihood classifier (see Appendix B). Concatenation correspond to the Maximum Likelihood classifier applied to the concatenated features, as previously described. Sum, Product and Bayes methods are as previously described as well.

Comparing Audio and Video accuracies we see that the video-based method is more precise. Among data fusion methods, Concatenation is better, but the accuracy of the product rule is nearly the same. Accuracy of the sum rule is smaller than Video's, so it seems not to be a good combiner. The Bayes combiners also have less accuracy than the Video method. Nevertheless, here we can see some positive facts: Bayes classifier accuracy increases as long as the number of clusters increase; and, in the case of training with a large number of chords (49, 168), even for small number of clusters (5, 10) Bayes accuracy is better than Audio's, indicating that, in this case, any rough knowledge about chord clusters is relevant and can be used to improve audio-based methods accuracy.

The fact that Concatenation is a good data fusion method is not surprising, because the classifier has access to all information provided by the audio and video features. More interesting is the performance of the product rule, where two different *experts* know only part of the total information available, and the final decision is taken observing the opinions of the experts, not the data itself.

We recall that the Product Rule arises under the hypothesis of conditional independency given the chord and supposing the same prior probability. Therefore, the performance of this data fusion method may be another evidence that audio and video information are in fact independent.

There is yet another explanation for the success of the Product Rule. Let us use the following likelihood functions:

$$p_{X,j}(x) = e^{-\frac{1}{2}(x-\mu_{X,j})^\top \Sigma_{X,j}^{-1}(x-\mu_{X,j})} \ , \tag{2.20}$$

$$p_{Y,j}(y) = e^{-\frac{1}{2}(y-\mu_{Y,j})^\top \Sigma_{Y,j}^{-1}(y-\mu_{Y,j})} \ , \tag{2.21}$$

which are gaussian multivariate distributions without the normalization factors $1/((2\pi)^6|\Sigma_{X,j}|^{1/2})$ and $1/((2\pi)^4|\Sigma_{Y,j}|^{1/2})$, respectively. To simplify, let us suppose also that $\Sigma_{X,j} = \sigma_{X,j} I_{12}$ and $\Sigma_{Y,j} = \sigma_{Y,j} I_8$. This way

$$p_{X,j}(x) = e^{-\frac{1}{2\sigma_{X,j}}\|x-\mu_{X,j}\|^2} \text{ and} \tag{2.22}$$

$$p_{Y,j}(y) = e^{-\frac{1}{2\sigma_{Y,j}}\|y-\mu_{Y,j}\|^2} \ . \tag{2.23}$$

Now the product rule

$$\max_{j=1,\ldots,M} p_{X,j}(x) \cdot p_{Y,j}(y) \tag{2.24}$$

becomes

$$\max_{j=1,\ldots,M} e^{-\left(\frac{1}{2\sigma_{X,j}}\|x-\mu_{X,j}\|^2 + \frac{1}{2\sigma_{Y,j}}\|y-\mu_{Y,j}\|^2\right)} \ , \tag{2.25}$$

which (applying $\log$ and multiplying by 2) is equivalent to

$$\min_{j=1,\ldots,M} \frac{1}{\sigma_{X,j}}\|x-\mu_{X,j}\|^2 + \frac{1}{\sigma_{Y,j}}\|y-\mu_{Y,j}\|^2 \ . \tag{2.26}$$

Equation 2.26 says that, given (x, y), the product rule tries to minimize the sum of the squared distances to the respective "centers" of audio and video classes, such distances being weighted by the inverse of the "spread" of the the classes. This is an intuitively reasonable strategy, indeed.

Now let us go back to gaussian likelihoods:

$$p_{X,j}(x) = \frac{1}{(2\pi)^6|\Sigma_{X,j}|^{1/2}} e^{-\frac{1}{2}(x-\mu_{X,j})^\top \Sigma_{X,j}^{-1}(x-\mu_{X,j})} \ , \tag{2.27}$$

$$p_{Y,j}(y) = \frac{1}{(2\pi)^4|\Sigma_{Y,j}|^{1/2}} e^{-\frac{1}{2}(y-\mu_{Y,j})^\top \Sigma_{Y,j}^{-1}(y-\mu_{Y,j})} \ . \tag{2.28}$$

Let us suppose also that, conditioned to the chord c_j, X and Y are uncorrelated, that is, being Σ_j the covariance of $(X, Y)|C = c_j$, we have

$$\Sigma_j = \begin{bmatrix} \Sigma_{X,j} & 0 \\ 0 & \Sigma_{Y,j} \end{bmatrix} \ . \tag{2.29}$$

Then

$$\Sigma_j^{-1} = \begin{bmatrix} \Sigma_{X,j}^{-1} & 0 \\ 0 & \Sigma_{Y,j}^{-1} \end{bmatrix} \ , \tag{2.30}$$

so, defining $\mu_j := (\mu_{X,j}, \mu_{Y,j})$, the expression

$$(x-\mu_{X,j})^\top \Sigma_{X,j}^{-1}(x-\mu_{X,j}) + (y-\mu_{Y,j})^\top \Sigma_{Y,j}^{-1}(y-\mu_{Y,j}) \tag{2.31}$$

reduces to

$$((x,y)-\mu_j)^\top \Sigma_j^{-1}((x,y)-\mu_j) \ . \tag{2.32}$$

And since

$$\frac{1}{(2\pi)^6|\Sigma_{X,j}|^{1/2}} \cdot \frac{1}{(2\pi)^4|\Sigma_{Y,j}|^{1/2}} = \frac{1}{(2\pi)^{10}|\Sigma_j|^{1/2}} \ , \tag{2.33}$$

we end up with

$$p_{X,j}(x) \cdot p_{Y,j}(y) = \frac{1}{(2\pi)^{10}|\Sigma_j|^{1/2}} e^{-\frac{1}{2}((x,y)-\mu_j)^\top \Sigma_j^{-1}((x,y)-\mu_j)} \ , \tag{2.34}$$

which is precisely the likelihood of the class being c_j given $(X = x, Y = y)$, supposing that the distribution of $(X, Y)|C = c_j$ is gaussian, with mean μ_j and covariance Σ_j.

This shows that, under certain conditions, Concatenation is equivalent to the product rule, and explains, along with the fact that in our case $X|C = c_j$ and $Y|C = c_j$ are almost uncorrelated, why the accuracies of those methods have being so close in the experiments.

We recall that the product rule arises when maximizing Equation 2.15 under the hypothesis of equivalent priors and conditional independence given a chord. We

have just seen that, supposing only uncorrelation (which is less then independency), the Product Rule appears as well. But in fact we have used gaussian likelihoods, i.e., we supposed the data was normally distributed. This is in accordance with the fact that normality and uncorrelation implies independency.

The main consequence of this discussion has to do with the *curse of dimensionality*. If we suspect that the conditional joint distribution of (X, Y) given any chord $C = c_j$ is well approximated by a normal distribution, and that $X|C = c_j$ and $Y|C = c_j$ are uncorrelated, than we should better use the product rule, because we do not have to deal with a feature vector with dimension larger the largest of the dimensions of the original descriptors. Besides, the product rule allows parallelization.

On the Sum Rule, we should mention that it can be regarded simply as a voting scheme, where votes consist of degrees of belief in the classes, given by the likelihood functions.

2.5 Applications and Further Development

We have seen that video information can be effectively used to improve audio-based methods for guitarist-computer interaction. However, hardware improvements are needed to make the system more user friendly. In fact, although the middle-phalange gloves are unobtrusive for the guitarist, not needing to use them would increase the naturalness of the interaction. So we plan to work on a visual chord descriptor not based on helper artifacts.

Also, there are other techniques that can be explored to capture the guitar: we could use infrared LEDs, instead of fiducials, for instance. Furthermore, the video signal could be replaced by some information about the pressure of the fingertips on the guitar fretboard.

In this direction, there is a very recent work [25] in which capacitive sensors are placed between frets, under the strings. The authors point that, in fact, "fingers do not perform a big pressure on the fingerboard, and even, do not necessarily touch the fingerboard (specially in high pitches)". So they opted for a sensor that measures the distance between the fretboard and the fingers. The paper shows good experimental results for detecting some left hand gestures, like vibratos, finger bars, basic arpeggios and sequences of single notes. They leave chord recognition as future work.

The same conference proceedings features a work [51, 22] that describes some techniques aiming to "augment" the guitar-playing experience, based essentially on the use of hexaphonic pickups for multi-pitch estimation, and stereo cameras for detecting fingertip positions as in [34]. The authors mention the use of Bayesian Networks for combining audio and video information, but the paper

does not present performance measures.

Yet in this regard, the company Fender, maker of guitars, is announcing to Mar 01, 2011, the release of Squier, a Stratocaster guitar which should be also a controller for the Rock Band 3 game. According to the website of the company, they make use of a position-sensing fingerboard.

Most of the previous/ongoing works that we have found give more attention to problems related to score-following or music transcription applications where the interest is in detecting sequences of single notes, perhaps using video or another not-audible signal. By concentrating on bimodal chord recognition, our work complements existing studies on guitarist-computer interaction. In fact, often the guitarist of a band is just playing chords, not improvising or performing a solo. Therefore, following/detecting chords is, from the point of view of an application, as important as following/detecting musical notes.

Regarding the problem of Data Fusion, we have used video information as *prior* in one of the methods. It would be interesting, from the theoretical point of view, to study the behavior of such method when audio, instead of video, is used as prior. Besides, some musicians may argue that, in the brain, audio information is processed before video information (when both are available).

Another task left for future work concerns testing the usability and robustness of the proposed methods with different users. This is especially important due to the fact that we are applying Supervised Machine Learning techniques. We decided not to perform such evaluation for this work because of the current video descriptor, which is based on helper artifacts on the hands of the user.

Part II

Synthesis

Chapter 3

Musical Scale Tiles

3.1 Introduction

This text is being written in a period particularly interesting in what concern human-computer interaction, a period characterized by the term *multi-touch*.

Mouse and keyboard have been the main interfaces between humans and computers for years, but now the technology, the people and, of course, the market, are finally ready for a, say, more direct use of fingers to interact with computer software. Consequently, multi-touch interfaces are becoming very popular in the realm of musician-computer interaction.

As usual, in the beginning developers try to emulate the old technology in the new one. That is why so many musical applications designed for multi-touch devices have interfaces that are simply pictures of piano keyboards [56], guitar fingerboards [29], etc. Of course there is the strong argument that it would be easier to play the computational device if its interface would appear like that of some real instrument. But this way the computational instrument cannot compete with the one it is trying to imitate, since the usability and sound of the latter is the best possible, by definition.

In this chapter we describe a bi-dimensional interface for music performance inspired on the guitar fretboard. It resembles the guitar interface in two ways. First, because of multi-touch device capabilities, some musical effects common on the real instrument (like bend, vibrato and polyphonic melody) are allowed. Second, we have applied some well known design principles [37] to make the interface simple and clean, what is especially useful for devices with small screens.

Those design principles led to a representation where the notes of particular scales are presented as a "matrix of points" on the screen. In this case, a multi-touch display is not essential, and the representation can be embodied in a "matrix of buttons" as well.

3.2 Previous Work

Perhaps the best known instruments displaying musical notes as matrices of points are the many types of accordions and concertinas. As a more recent example we can cite the AXiS-49 MIDI controller [41], a bi-dimensional interface whose keys are hexagonal, forming a honeycomb pattern.

Regarding touch-screen interfaces for music production, an extensive list of examples can be found in [32], where they are viewed as a subset of the more general class of *tangible* instruments. Multi-touch interfaces are also being used as controllers for sequencers, synthesizers and virtual instruments, as is the case of the Lemur, by JazzMutant [30]. Finally, we cannot forget to mention the "iPhone family" of multi-touch devices, by Apple Computer [28]. The App Store concept, pioneered by the company, allows developers from outside Apple to build applications for the device. Because of this, the number of musical applications using multi-touch interfaces has grown fast, and the adoption of multi-touch screens and of the App Store concept by competitors is making that phenomenon even more intense.

That said, it is difficult to claim that the ideas and concepts to be presented hereafter are totally novel. What we can say is that, as far as we can see, helped by the lens of modern web-search engines, we have found no bi-dimensional multi-touch interface for music performance like the one we are about to describe.

3.3 Tiles

Let us start by looking at Figs. 3.1(a) and 3.1(b), where notes with the same number have the same corresponding fundamental frequency. The representation of Fig. 3.1(b) appears naturally in instruments tuned in fourths. This means that the note immediately above the note 0 (i.e., note 5 in Fig. 3.1(b)) is its perfect fourth; that the note immediately above the number 5 (i.e., note number 10) is the perfect fourth of note number 5; and so on.

In Figs. 3.1 (c), (d) and (e), three examples of heptatonic scales, according to the representation of Fig. 3.1(b), are shown. Fig. 3.1(g) depicts a pentatonic scale. The idea is to hide the squares that are not filled, since they represent notes out of the scale, and re-arrange the remaining notes. This way we arrive at the representation shown in Fig.'s 3.1 (f) and (h), respectively, where this time the gray-filled note represents the scale root. The order is preserved, that is, from the tonic note (scale root), left to right and bottom to top.

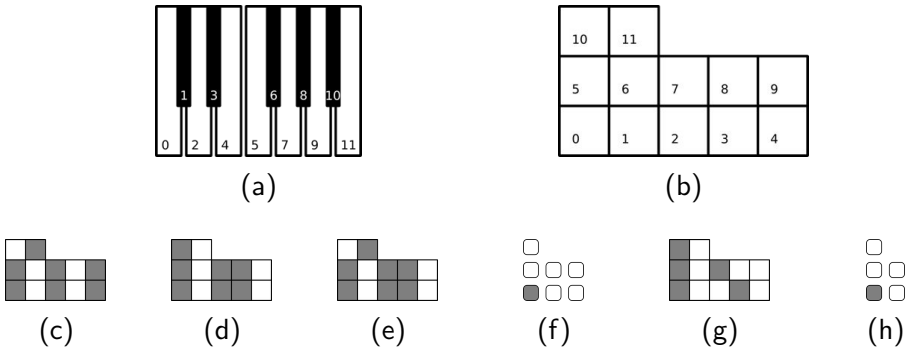


Fig. 3.1: Chromatic scale on the piano interface (a) and in instruments tuned in fourths (b). Diatonic major (c), natural minor (d), harmonic minor (e) and pentatonic minor (g) scales. Heptatonic (f) and pentatonic (h) scale tiles.

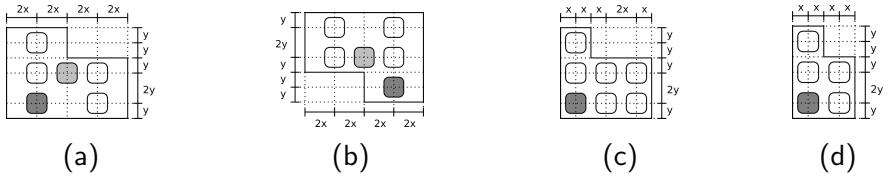


Fig. 3.2: Scale tiles for the Blues Minor (a), Blues Major (b), general heptatonic (c) and general pentatonic (d) scales. x and y represent positive measures, not necessarily equal.

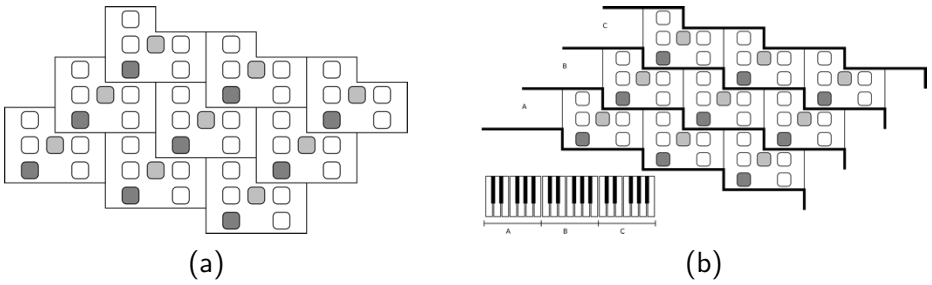
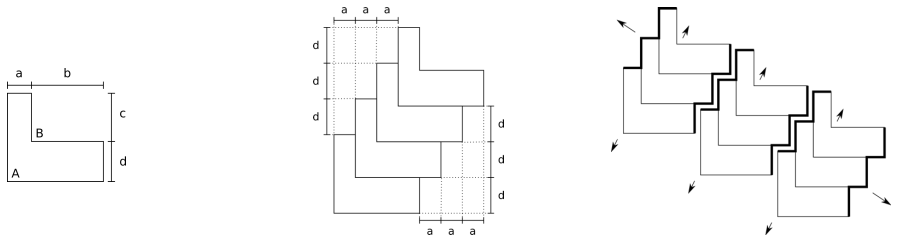


Fig. 3.3: (a) Tiling of the plane with the Blues Minor scale tile. The blue note (augmented fourth) has special highlight in this representation: columns having blue notes contain no other notes. (b) Octave relation in the Blues Minor scale tiling. Tiles in the same band (A, B, C, etc) are such that the fundamental frequencies associated with points having the same relative position in the corresponding tile are equal. Notes of tiles in the region B are one octave above the corresponding notes in the tiles of region A, and so on.



(a) (b) (c)

Fig. 3.4: All presented musical scale tiles have a common L-like shape (a), and the corresponding tessellation is such that corners A and B meet (b). By coupling side by side the bands shown in (b) the tessellation is completed (c).

3.4 Plane Tessellation

In view of tiling the plane with *musical scale tiles* like those shown in Fig.'s 3.1 (f) and (h) it is necessary to state precisely some geometrical measurements. Here, we will use as example the Blues Minor scale, the process for the other scales being similar. The corresponding tile is shown in Fig. 3.2(a). It is worth mentioning that the Blues Minor scale notes are: scale root, minor third, perfect fourth, augmented fourth, perfect fifth and minor seventh (see also Appendix A).

Given a tile, the next step is tiling the plane as shown in Fig. 3.3(a). Fig. 3.3(b) shows the octave relation in the tessellation. Again, it is similar to the one that appears naturally in instruments tuned in fourths. After building a tessellation, what remains is to select the area that actually will be used in the algorithm. For simplicity, such region will normally have a rectangular form.

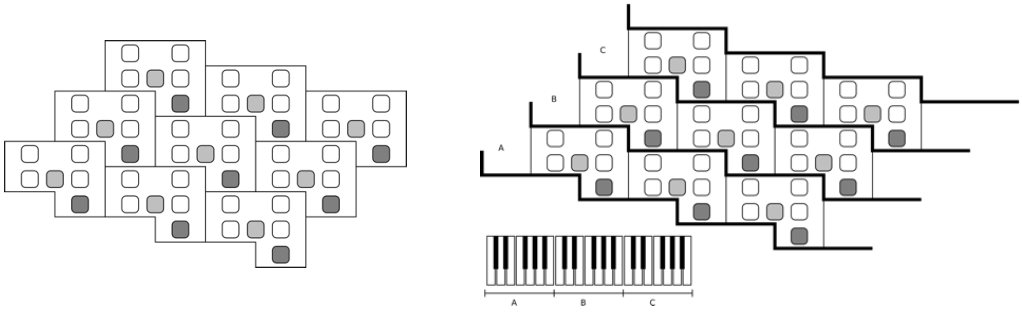
We have studied the shape of tiles for the Blues Major and Minor scales, as well as general heptatonic and pentatonic scales. We just described how to tessellate the plane using Blues Minor scale tiles. For the other mentioned scales, the procedure is analogous, tiles being the ones showed in Figs. 3.2 (b), (c) and (d).

Notice that all tiles have a common L-like shape, as shown in Fig. 3.4(a). The corresponding tessellation must satisfy the condition that corner A of some tile coincide with corner B of the adjacent tile (Fig. 3.4(b)). The tiling is completed by coupling side by side the bands shown in Fig. 3.4(b) (see Fig. 3.4(c)), what is possible due to the coincident metrical relations of adjacent boundaries (shown in Fig. 3.4(b)).

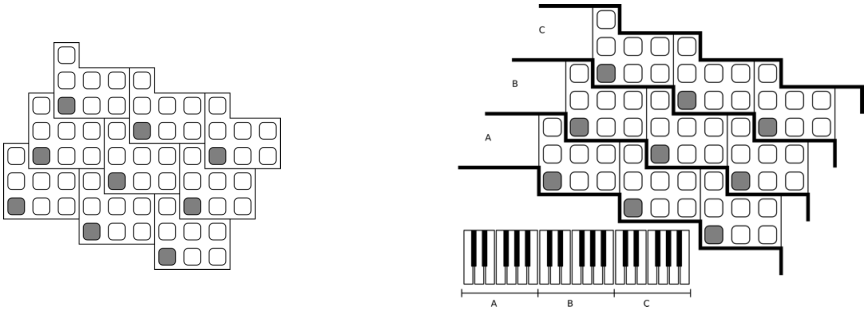
Fig. 3.5 illustrates the tessellation and octave distribution for the Blues Major, heptatonic and pentatonic scales, whose tiles are presented in Fig. 3.2.

The representation of musical scales presented in this chapter has been subject

Blues Major Scale



Heptatonic Scales



Pentatonic Scales

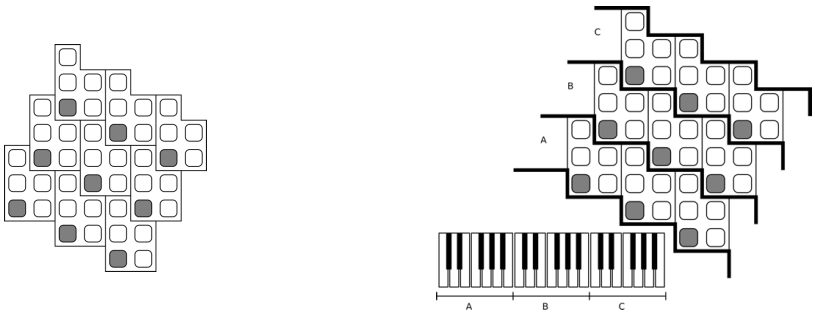


Fig. 3.5: Analog of Fig. 3.3 for the Blues Major, heptatonic and pentatonic scales.

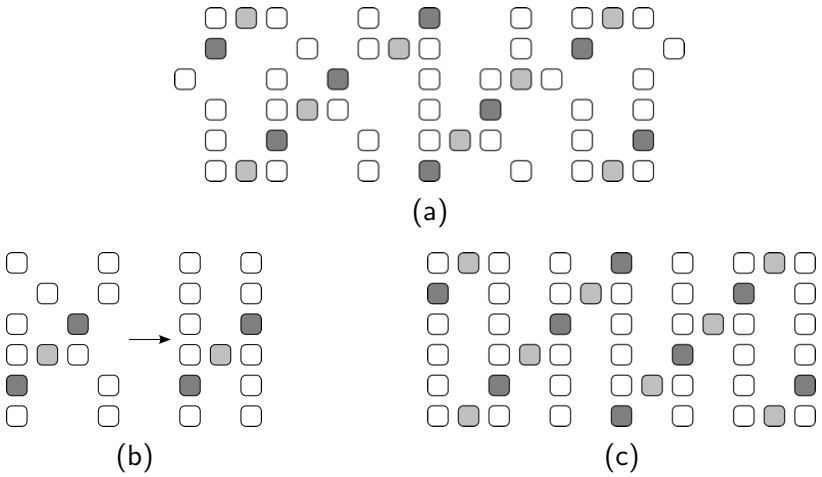


Fig. 3.6: (a) Patterns of the Blues scale on the guitar fretboard. After proper vertical alignment (b), a more visual-friendly configuration is obtained (c).

of a Brazilian patent application [14].

3.5 Implementations

Let us now relate some implementations of the representations of musical scales introduced in this chapter. We begin by describing the particular case where those ideas came up.

3.5.1 Blues-Music Improvisation on Multi-Touch Interfaces

As our first application, we decided to build a computational musical instrument for the purpose of improvising solos in the Blues style.

The Blues music genre originated from music played/sung by slaves in the USA in the 19th century. In its modern form there are two elements very frequently used: a 12-bar chord progression and the so called Blues scale[1]. Fig. 3.6(a) shows the notes of the Blues scale on part of the real guitar fingerboard with the default tuning (E, A, D, G, B, E, from the 6th to the 1st string). A very common exercise (one that guitar students learn early in a guitar course) is to improvise on the Blues scale listening to a 12-bar Blues base.

[1] In fact there are two kinds of Blues scales: the Major and the Minor. However, being the most commonly used, the Blues Minor scale is simply referred to as the Blues scale.

Regarding performance, the guitar is a very rich instrument, in the sense that musicians are allowed to apply many different effects, especially *bends*, *hammer-on*'s, *pull-off*'s and *slides* (widely used in Blues songs). So it would be interesting to preserve these possibilities in a computational musical instrument.

Most of the mentioned effects are allowed, indeed, when we use multi-touch interfaces. The bend effect, for example, can be obtained by touching the screen and sliding the finger up or down. Sliding up and down repeatedly produces vibratos. The distance between the touchdown point and the current finger point in a slide movement determines the amount of pitch shift. And, by definition, multi-touch interfaces allows the user to play many notes at the same time, something that actually occurs very often in music performance in general.

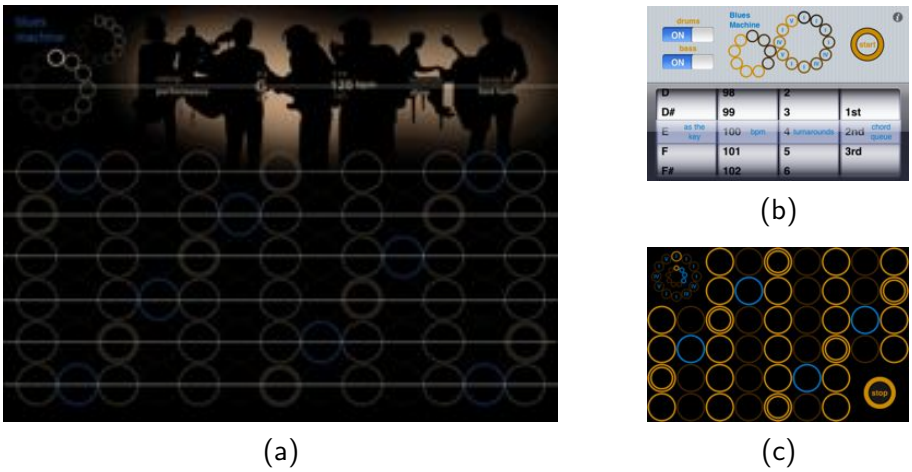
Back to the Blues scale, there are five different patterns to learn, that overlap each other to form the arrangement shown in Fig. 3.6(a). Besides learning the patterns, one important difficulty is that the guitarist can not turn the notes lying out of the Blues scale off, what could prevent the playing of all "wrong" notes. *Wrong* is in quotes because music is an art, and as such there is no rule that can not be violated. In many cases, however, a musician chooses some scale and tries to build phrases over it. The point is that memorizing (and, more importantly, embodying) a given scale is a process that takes a long time and considerable effort.

That is how we came up with the scale pattern shown in Fig. 3.6(c). That pattern was obtained by simply aligning vertically the notes of the scale at the guitar fingerboard's pattern, as shown in Fig. 3.6(b). As mentioned before, the arrangement of notes in Fig. 3.6(c) can be seen as a tiling of the plan with especially designed Blues scale tiles, as clarified in Fig. 3.3.

We implemented the above idea in two different multi-touch devices: one with a large 40×30 *cm* screen, and the other a smart-phone with a 480×320 *pixels* screen.

The hardware of the former is based on the Reactable project [50]. The Reactable has a tangible multi-touch screen interface for interaction. It can be used by many players at the same time, in the context of collaborative electronic music performance. Under the hood, i.e., underneath the screen, there is a projector and a video camera. A computer vision framework [33] tracks (by means of the video camera) finger positions and send them to the application software. The projector displays the interface and user-interaction feedback.

Fig. 3.7(a) shows the realization of our proposed interface on the Reactable. In our implementation of the hardware there are real strings mounted upon the screen. This extra component has a conceptual and practical function: to bring back the real guitar tangibility, so important in the playing experience. The bend effect, for example, is limited by the elastic properties of the strings. The form factor is also important here: a 40×30 *cm* screen size allows comfortable



(a) (b) (c)

Fig. 3.7: Interface for Blues improvisation on the multi-touch table (a) and on a smartphone display (c). For the smartphone version, a separate interface for setup is needed (b).

performance, in the sense that it's big enough for bend effects and small enough to allow the playing of two or three notes without global movement of the hands.

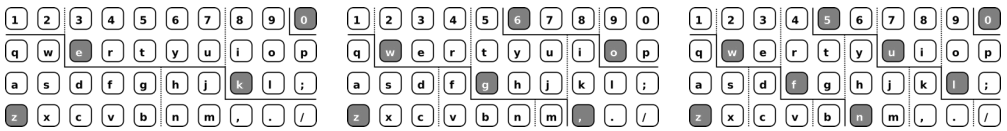
Both implementations have some kind of configurable accompaniment. The user can chose key, tempo, etc, which is helpful if he/she wants to play by him/her-self. The smart-phone version (Fig. 3.7(c)) has a separate screen for setup (Fig. 3.7(b)), while the other presents a unified interface.

3.5.2 Musical-Scale Mappings on the QWERTY Keyboard

Except for the Blues Major and Minor scales, the representation of musical scales described in Sections 3.3 and 3.4 can be easily adapted to computer keyboards. In fact, a mapping between keyboard keys and the notes of the chromatic scale, based on the distribution of such notes in fretted musical instruments tuned in fourths (as in Fig. 3.1(b)), has already appeared in [18].

A possible mapping obtained by tessellating the plane using those 12-notes tiles is depicted in Fig. 3.8(a). For heptatonic and pentatonic scales, possible tilings are presented in Fig.'s 3.8 (b) and (c), respectively. In Fig. 3.8 the Z key is being used as pivot, but, obviously, this is not mandatory. Any other key of the 4×10 grid could be used as well. Besides, the note and the octave corresponding to the pivot key are also variable.

One of the good features of the keyboard interface is its tangibility. In multi-touch flat interfaces, it is more difficult to "press a button" without looking at it.



(a) (b) (c)

Fig. 3.8: Chromatic-scale (a), heptatonic-scale (b) and pentatonic-scale (c) keyboards.

There are also some shared limitations (between keyboards and flat interfaces), like the fact most keyboards are unaware of key-down velocity, a feature that impairs performance expressiveness[2]. Furthermore, at least for the hardware we have used in the experiments, polyphony is not the same over the keyboard, i.e., there are, for instance, some combinations of three keys that can be played simultaneously, but others do not.

Other performance limitation concerns the absence of a pitch wheel, very common on MIDI keyboards. This issue could be circumvented by using the mouse or the trackpad. For a pitch shift of one or two semi-tons, up or down the chromatic scale, modifier keys (ctrl, alt, etc) could be applied. This would be especially useful to reach that particular note which is out of the chosen scale, but that the composer (or performer) does not renounce to make use of.

[2] As far as we know, the current consumer-available multi-touch screen devices have no pressure sensor. Pressure-aware multi-touch surfaces do exist [54], but they are not able to display data.

Chapter 4

Automatic Composition

In this chapter we will talk about some experimental implementations of automatic composition algorithms, in which we use the bi-dimensional representations of musical scales discussed in the previous chapter.

4.1 First Experiments

Some instruments have interfaces which, loosely speaking, can be said to be one-dimensional, in the sense that the buttons associated with musical notes are arranged on a straight line. From this point of view, the representation of musical scales introduced in Chapter 3 can be classified as bi-dimensional.

It is natural, therefore, to think about automatic composition algorithms that explore this property. We have done some experiments in this direction, and here we will describe the first one. It has two main aspects: a bi-dimensional domain of musical notes (as expected) and a Markovian Process engine, in which the random melodic sequence must observe certain (harmonic related) restrictions.

Markov models have been widely exploited for automatic music composition [45, 43, 42], but we haven't found works about their use on bi-dimensional grids of musical notes. The motivation for this usage relies on the fact that, in instruments like the guitar, the performer, when improvising, sometimes think about sequences of notes not in terms of their location in the score, but in terms of where they are located in the fretboard.

We now present the tested algorithm for generating melodic sequences, as well as the related probabilistic tools[1]. We have chosen the 12-bar Blues style, because of its simplicity, and the Minor pentatonic scale for the melody, since it is largely used in Blues songs. Fig. 4.1 shows the part of the Minor pentatonic scale

[1] The experiments described in this section were conducted in collaboration with T. Franco [15], who has contributed with the modeling of the random processes involved.

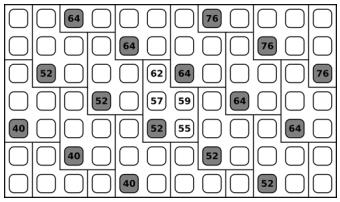


Fig. 4.1: Pentatonic scale tessellation.

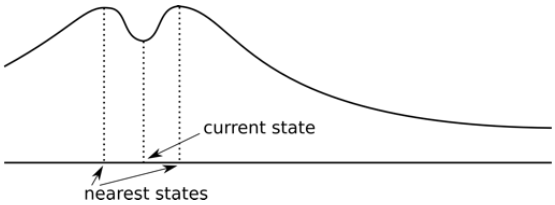


Fig. 4.2: Shape of the transition probabilities curve.

tessellation we have selected for the experiments. Numbers represent MIDI-note codes.

The finite Markov Chain state-space is defined as $E = H \times R \times M$. First, H is the space of possible chords, whose sequence (the chord progression) determines the music harmony. In our implementation, $H = \{I7, IV7, V7\}$ (where I, IV and V are the root, sub-dominant and dominant chords, respectively). Second, R is the space of rhythmic patterns. We have used five different states, corresponding to silence (rest), one whole, two halves, three thirds and four quarter notes. Lastly, M is the space of possible notes, namely, the scale. Here is where *bi-dimensional composition* appears, since M is a rectangular subset of a tessellation as described in Chapter 3.

The Markov Chain of harmony (X_i), and of rhythm, (Y_j), are independent. On the other hand, (Z_k), which gives the choice of notes, is dependent on (X_i) and (Y_j). The dependency on harmony, (X_i), is natural from the fact that the melody must follow harmony rules. On the other hand, the dependency on (Y_j), the sequence of time figures, comes from the fact the total number of notes in the sequence (Y_j) determines the length of the sequence (Z_k).

The sequences (X_i), (Y_j) and (Z_k) are sampled every time a new series of 12 bars is about to begin. So i ranges from 1 to 12, j from 1 to 48 (12 bars $\times$ four beats per bar), and the range of l, as we have mentioned, depends on the number of notes, which is determined by (Y_j).

The conditioning on specific events mimics the behavior of a musician, who, when improvising, pursues a target note in meaningful chords. That is what we call the *target-note* improvisation paradigm. In our implementation we have conditioned both the first and the last notes of the 12-bar series as being the scale root. Regarding chords we have conditioned the first, the fifth, the ninth and the twelfth as being $I7$ (the root chord), $IV7$ (sub-dominant), $V7$ (the dominant), and again $V7$, respectively. About the rhythmic-patterns sequence, no conditioning was imposed.

Now let A be the set of sequences which satisfy the just described restrictions.

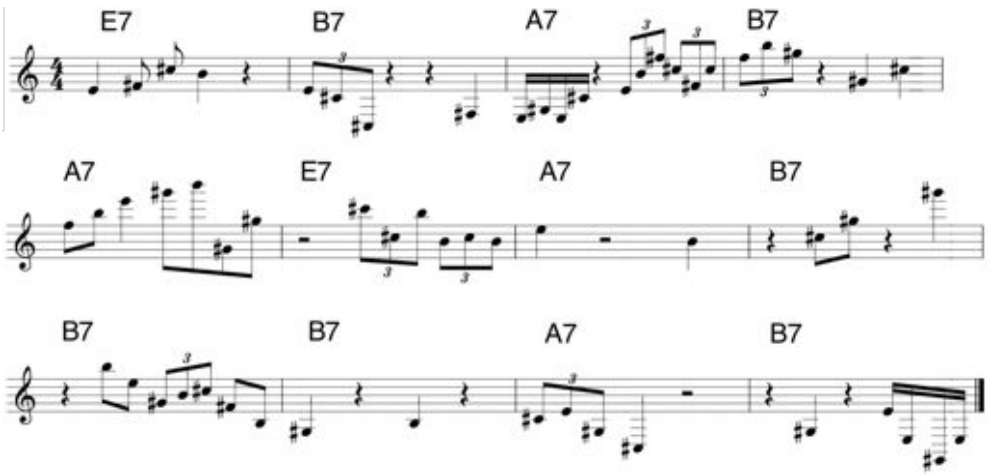


Fig. 4.3: A 12-bar sample from the automatic composition algorithm implemented.

The method to simulate the conditioning of the Markov Chain on A is the very well known *rejection method*, which consists simply in sampling the Markov Chain, and if the sample belongs to the set A, keeping it. If not, we resample until we get an allowed sample. Theoretically, the number of trials until an allowed sample is obtained can be arbitrarily large. For this reason, we limited the number of trials. If no allowed sample is found, the last one is chosen. Of course doing this we do not simulate exactly the conditioned Markov Chain defined above. Nevertheless, this way the algorithm imitates musician's errors, when the target note is not reached, something that can eventually happen.

Summarizing: each time a new 12-bar series will begin we sample three Markov Chains as described above until the mentioned conditions are satisfied or the maximum specified number of trials is reached, what comes first.

We have used the uniform distribution as initial distribution of all (X_i), (Y_j) and (Z_k) sequences. The transition probabilities for (X_i) was set as uniform, i.e., being at state I, the next state could be IV or V with equal probability, and so on. For (Y_j) we have chosen M-shaped functions centered in the current sample. This means that if at the current beat the chosen figure is "three thirds", in the next beat the probability of playing the same figure is small, the probability of playing two half-notes or four quarter notes is high, etc. Fig. 4.2 illustrates this situation. The case of the sequence (Z_k) is analogous, with the states being the *row* and the *column* of the points in the bi-dimensional representation of the scale. Actually, there are two independent Markov Chains controlling the sequence of notes, one for the row index and the other for the column index, the transition probabilities of them being shaped as shown in Fig. 4.2.

Fig. 4.3 shows the score corresponding to a 12-bar sample from our method. The algorithm outputs what resemble jazz-like improvisations. This behavior is explained by the fact that the number of restrictions is small, so there are many notes that, regarding the current chord being played, may seem dissonant for the unaccustomed ear.

However, the greater the number of restrictions, the more trials the algorithm has to perform to satisfy them. This could preclude the execution of the algorithm in real-time[2]. In the next section we describe an experiment at which we circumvented this issue by sampling shorter sequences of notes.

4.2 Musician-Computer Interaction using Video

In this experiment we mixed techniques from sections 2.2, 3.3 and 3.4. We have implemented an automatic composition algorithm, similar to the one introduced in the previous section, at which the information of the current chord being played is sometimes provided by the video-based guitar-chord recognition method, and, if desired, it is also possible to control the region of the tessellation to which the sampled sequence of notes has to converge, by observing the location of the hand relatively to the guitar fretboard.

In this case we chose the diatonic scale, in the key of G. Fig. 4.4(a) shows almost all the notes of such a scale between the first and the 19th fret of the guitar, and Fig. 4.4(b) shows the corresponding representation using heptatonic scale tiles, as described in Chapter 3.

As in the previous example, the sequence of rhythmic patterns is controlled by a Markovian Process. Every time a new beat is about to begin, the system decides if one whole, two halves, three thirds or four quarter notes should be played along it, or even if no note should be played at all. The number of notes will depend on the current value of the parameter describing the "level of intensity" for the music. The greater the intensity, the greater the probability of playing more musical notes in the next beat. Then the melodic line is built. This time the information of the current chord being played is relevant, because melody and harmony must combine: the algorithm may check if the first (and/or the last) note of the sequence sampled for the next beat is the same (regardless the

[2] In our implementation, we have seen that for two target notes an upper bound of one thousand trials is never reached, i.e., the algorithm always finds a satisfactory solution before the thousandth trial. But in some tests we have conducted, for more than 4 or 5 restrictions that upper bound is easily passed. We could in this case raise up the upper bound to, say, $10,000$. But in this case when the number of trials is high (near the upper bound) the time consumed is such that the algorithm cannot work in real time (for tempos around 120 beats per minute).

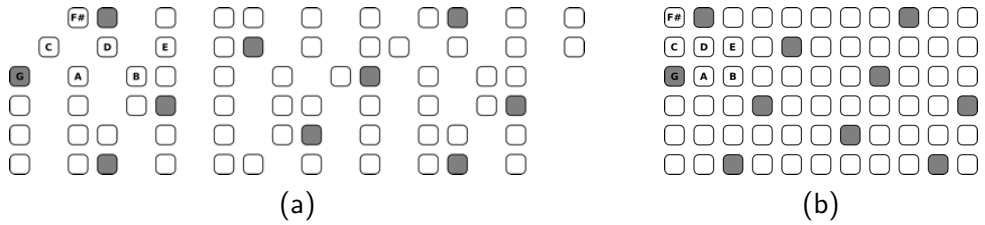


Fig. 4.4: Diatonic scale (key of G) as in the guitar fretboard (a) and as represented using heptatonic scale tiles (b).

octave) of the current chord's root note. Furthermore, it may be imposed that the last note of the sequence should fall in some region of the matrix of musical notes. That region, in its turn, can be controlled by the location of the guitarist left hand in the guitar fretboard.

As a proof of concept, we have composed a music piece in which the mentioned ideas are explored. It is organized in cycles, bars and beats: four beats per bar and four bars per cycle. We have used four musical instruments: guitar, string ensemble, drums and piano, of which just the former is a real instrument. Most of the time the string ensemble follows exactly the chord that is being captured by the computer vision system, but in some cycles it can also perform a chord sequence memorized in previous cycles. After the drum loop is triggered by a keyboard command, the pre-programmed loops will run for a certain number of cycles, up to the end of the piece. Every time a new beat is about to begin, the Markov-process based sequences are sampled and resampled until the melody conditions are satisfied or the maximum number of trials is reached. Eventually the system turns the *air guitar* module on, so the location of the hand (in guitar-fretboard coordinates) controls the region to where the sequence of notes has to converge.

A more detailed pseudo-score of the implemented/composed music piece is presented in Appendix C. An interesting aspect of this piece is that we explore the fact that the computer vision system can detect the chord even when just some of their notes are picked (i.e., the player is fingering the chord).

We found that the automatic composition algorithm just described outputs better melodic sequences when compared to the one in Section 4.1, because of the more restricted relation between melody and harmony. However, the use of a bi-dimensional Markovian Process, unaware of the order of notes in the scale, produces sequences that sound too random.

In the next Section another algorithm is presented, for yet another application. We kept the notion of "level of intensity", but a different random process for generating the sequence of notes is used.

4.3 Scale-Aware Random Walk on the Plane Tessellation

The automatic composition algorithm to be described was developed to produce the score for a synthesized dance motion[3], which was obtained through a graph of movement segments, captured using MoCap[4] technology.

There are five steps:

1. Some songs are composed.
2. A dancer performs listening to the songs, and the dance is captured.
3. The dance is segmented, and the segments are used to build a motion graph.
4. A new dance is synthesized by random-walking on the motion graph.
5. A new song, based on the walk on the graph, is composed.

This time we have chose to compose songs in the brazilian *samba* style, for which we need, essentially, just an instrument for each one of the percussion, harmony and melody categories. In samba, *cavaquinho*[5] and *tambourine*[6] are classically used for harmony and percussion, respectively. Regarding melody, there is the cavaquinho, the flute, the mandolin and other choices (like the human voice, for instance).

We have composed three music pieces to be performed by the dancer. Each music is organized in a 4-levels multi-resolution structure, consisting of *cycles*, *bars* (measures), *beats* and *clocks*: 16 bars per cycle, 2 beats per bar and 12 clocks per beat.

For each song we have varied the melodic instrument, using the flute, the cavaquinho and the clarinet. All songs were composed to have 3 cycles, the second (respectively, the first) being the more (respectively, the less) musically intense. We model intensity by putting this concept in direct proportion with the number of notes per bar which are played. The presence or not of a percussion instrument affects music intensity as well.

Drum loops (i.e., tambourine loops) were composed manually, as well as cavaquinho loops. We have chosen to use G, E7, Am, D7 as the chord progression. The melody is organized in phrases, each phrase extending a bar (measure). Notes for each phrase are not picked manually. Instead, we specify just the *number of notes* that must be played for each one of the (two) beats of the phrase.

[3] This work has been done in collaboration with A. Schulz and L. Velho [55].

[4] Motion Capture.

[5] We are using the brazilian-portuguese word. As far as we know, the best translation would be *ukulele*, but, despite having a very similar shape, ukulele's default tuning is G, C, E, A (from the lowest to the highest pitched string), while cavaquinho's is D, G, B, E.

[6] In portuguese, *pandeiro*.

Beats can be filled with 0, 1, 2, 3, 4 or 6 equally spaced notes, what explains why we are using 12 clocks as the resolution for the beat. This way, composing for a phrase means calling a functions saying something like:

```
fill beat 1 with m notes,
fill beat 2 with n notes.
```

In fact sometimes another slightly different method is called:

```
fill beat j with m notes, the first note being of type x.
```

The type of the note has to do with the current chord being played as in Section 4.2: the note must be the same (regardless the octave) of the current chord's root note.

This extra restriction is usually observed in the first beat of the measure, because, in our implementation, that is when the chord of the harmony is changing (there is a chord change every new bar).

In the implementation used in this work, for the

```
fill beat j with m notes
```

method the new note is simply one step above or bellow the current note[7] in the chosen musical scale (which, by the way, is the diatonic major), while for the method

```
fill beat j with m notes, the first note being of type x
```

the algorithm skips up or down the scale until a note which "sounds good" with the current chord is reached.

If the algorithm will search for a note up or down the scale is a matter of

- a random sample taken from a uniform distribution in the interval $[0, 1]$ being greater or smaller than $1/2$ and
- a note in the chosen direction being available.

Notes are confined in a 4×9 grid of the diatonic major scale tessellation, obtained as described in Chapter 3. Since in the grid the same musical note can be associated with more than one point, the algorithm decides which point to use by, again, tossing a fair coin.

After the songs are composed, the dancer performs listening to them. The captured motion of the dancer is segmented according to the length of a musical phrase, which is equal to the length of a measure (bar), this way making the built motion graph *measure-synchronous*. Once a new dance is obtained by random-walking on such a graph, the next step is to compose the music score for it.

Let us suppose that we have constructed a new dance consisting of 4 segments,

[7] What makes the random walk scale-aware.

```
(c1|s4), (c3|s12), (c3|s13), (c2|s10),
```

where `(cI|sJ)` means the segment `J` of clip (song) `I`. Now let us say that `(c1|s4)` was composed using the high-level command:

```
fill beat 1 with 4 notes, the first note being of type x
                fill beat 2 with 2 notes
```

and that the tambourine was omitted in that phrase. Then the first phrase of the score for the synthesized dance will be composed in the same way: the first beat will be filled with 4 notes, the first of which being of type `x`, the second beat will be filled with 2 notes, and the percussion will be omitted.

The chord progression can be set manually, or we can simply use the same sequence of chords for the pre-dance songs and for the synthesized dance.

4.4 Applications and Further Development

There are some interesting applications to be developed, and questions to be investigated, regarding the kind of interface we have presented in Chapter 3.

First, the project discussed in Subsection 3.5.1 can be extended for other scales. Also, it would be interesting allowing the musician to navigate between scales. It is usual, for example, when improvising Blues, to build phrases in both the minor (default) as well as the major Blues scale. So the interface could present the possibility of changing from one scale to another at performance time.

Besides, we have implemented some performance effects, like *bend* and *vibrato*, in the table version of the application, where a MIDI-synthesizer provided by the operating system was used. In smartphones, however, this task is more difficult, since they do not have built-in synthesizers. One option is to use smartphones just as multi-touch interfaces, sending commands to a more powerful computer, to perform the synthesis. But this would eliminate the "mobility" appeal of smartphones. So, implementing sound effects compatible with multi-touch interfaces for smartphones is an immediate task to be performed. We have mentioned *bend* and *vibrato*, but other effects brought from stringed instruments (the *slide*, for instance) also deserve attention.

Another idea is to build a game, where the player is invited to perform phases shown by the system, repeat them, or even improvise over, performing the so called call-and-response, a very common behavior in improvised music. The capabilities of such interfaces as tools for teaching/learning music theory, and as musical instruments to be used on stage, have to be more deeply tested as well.

Results from the automatic composition algorithm introduced in Section 4.3 were very interesting and encouraging. As future work in this subject is the

study of other scale-aware walks on bi-dimensional grids of notes, as well as the introduction of geometric restrictions, like the one in Section 4.2. In a multi-touch device, this would allow the user to improvise using high level commands like the number of taps in the interface per unit of time and the region in which they occur.

Playing the QWERTY keyboard with the keys-to-notes mapping described in Section 3.5.2 was also an interesting experiment. We are planning to work on an independent electronic musical instrument using the proposed interface. Such instrument would be a synthesizer (a simple wavetable synthesizer would suffice), where the piano-keyboard interface is replaced by the bi-dimensional grid of notes described in Chapter 3. We could then provide controls for sound effects (again, *bends* and *vibratos*, among others), as well as good polyphonic capabilities, which are not present in traditional computer keyboards.

Chapter 5

Conclusion

In a text from a research funding agency[1], published in June 9, 2010, one read:

> (...) research outcomes are expected to transform the human-computer interaction experience, so that the computer is no longer a distraction or worse yet an obstacle, but rather a device or environment that empowers the user at work, in school, at home and at play, and that facilitates natural and productive human-computing integration.

This quote is an evidence that the questions concerning the interaction between humans and machines are of central importance in the present times.

Along the work for this thesis, one of the main goals has been to simplify such interaction. We could, for instance, merely display a picture of the guitar fretboard in a multi-touch capable device for the purpose of simulating the use of the guitar. However, by eliminating the notes out of the chosen musical scale and re-arranging the remaining notes, we arrive at simple patterns, which, besides facilitating the playing experience, can be used for any scale with the same number of notes.

We have also seen that some problems related with the guitarist-computer interaction using audio can be circumvented by using video information. As when, for instance, the guitarist is playing a chord one string at a time, instead of playing all of the strings simultaneously.

However, in general, the simpler the interaction, the greater the complexity of the computational system involved. To capture the scene of a person playing guitar, for instance, we started by trying to segment the fretboard from a sequence of video frames without using helper artifacts in the instrument and hands. We

[1] U.S.A.'s National Science Foundation. Information and Intelligent Systems (IIS): Core Programs grant. Available at `http://www.nsf.gov/pubs/2010/nsf10571/nsf10571.htm`.

have had some success in this direction, using mainly algorithms for edge detection and bi-dimensional rigid transformations, but the computational cost increases fast with the number of necessary operations.

The good news about this fact is that there is a lot of work to be done in the field of Computer Vision, and the problem we have worked with can be the source and motivation for many developments in this area. In a not-so-far future, we expect computers to understand audio and video information in such a degree that entertainment, teaching, learning, communication and other human activities, can be performed at a near-optimum level, in the sense that machines can capture all the information that is needed for a particular task, and process it properly.

Up to that point, there will always be a trade-off between computational power and adequate mathematical tools. For now, we believe the bottleneck against efficiency is on the side of computational power, because data seems to be all the time more complex than computers can deal with efficiently. Perhaps the use of parallel processors can change this, but it is difficult to tell.

On the other hand, more and more we see that mathematical methods for data handling have to deal with wrong, imprecise, redundant and incomplete information. In addition, softwares have to be prepared to work properly under situations of imprecise and unknown input. For example, an automatic music composition algorithm which uses the information of the current chord being played should be able to produce good results even if the computer doesn't provide the correct information about the chord, or such information arrives latter than it should.

In what concern multi-touch (screen or keyboards) devices, there are still some hardware limitations to be solved before their playing capabilities can be compared to actual musical instruments. However, in this case the way seems not to be as long as in the Computer Vision case, because similar devices already exists (e.g., velocity sensitive keyboards or pressure sensitive pads).

We genuinely believe that this work has contributed to the development of (practical and theoretical) technologies in the mentioned directions.

From the mathematical point of view, we have (1) modeled a strategy of chord recognition which uses visual information as prior and the audio signal for fine-tuning, much like a human would perform to identify a chord; (2) analyzed the behavior of different Data Fusion techniques commonly found in the literature, always in light of data from real-world experiments; and (3) distilled patterns for representing musical scales in multi-touch interfaces through a tiling process, as it occurs in the guitar fretboard for the chromatic scale.

On the engineering side, we implemented tools for capturing audio and video descriptors, working on both the hardware and software levels. We also implemented computer programs to visualize such descriptors, especially in the case

of audio. Besides, we developed applications for music improvisation, performance and automatic composition on multi-touch interfaces. One of them, in fact, has been available worldwide, free of charge, for the iPhone platform.

Artistically speaking, we have shown an example of how a chord recognition method based on video can be used to control an automatic composition algorithm. Also, one of the automatic composition algorithms that we have developed has been successfully used in combination with a technique for producing a new performance from segments of motion-captured dance [55].

We are very enthusiastic about keep working toward providing better human-computer interaction experiences in music-related contexts. Our approach to science is that abstract concepts and ideas reach their maximum potential when they are "brought to life" as tools for students, workers, artists, etc. After all, the better the tools for learning and creative expression, the greater the possibilities for people to accomplish their goals as human beings.

Appendices

Appendix A

Music Theory

This Appendix contains some basic definitions from the field of Music Theory, which can help reading the main part of this text. We will also talk about those concepts having in mind the representations of musical scales described in Section 3.3. We have used [16] and [27] as main references.

A.1 Musical Scales

A *musical scale* is a series of notes in ascending or descending order. A scale begins in the *tonic note* and, in Western music, is usually a subset of the 12 notes of the *chromatic scale*.

Each note has an associated *fundamental frequency* (see Subsection 2.1.2). We say that the chromatic scale is *equally-tempered* when the ratio between the fundamental frequencies of two consecutive notes is $2^{1/12}$ or $2^{-1/12}$. The fundamental frequency increases when we go up to the scale. Two consecutive notes are one *semitone* apart, or, equivalently, they differ by an *interval* of one semitone. One *tone* is equivalent to two semitones.

The notes of the chromatic scale receive particular names according to the interval they form with the tonic: unison (zero semitones), minor second (1 semitone), major second (2), minor third (3), major third (4), perfect fourth (5), augmented fourth (6), perfect fifth (7), minor sixth (8), major sixth (9), minor seventh (10), major seventh (11) and perfect eight (12). The perfect eight is usually called *octave*.

In Chapter 3 we have talked about pentatonic, heptatonic and Blues scales. As far as we know, there are only two Blues scales: the Blues Major and the Blues Minor. On the other hand, a lot of pentatonic and heptatonic scales in use exist.

The Blues scales are defined, in terms of the intervals with respect to the tonic, as follows:

Blues Minor:	0	3	5	6	7	10	12
Blues Major:	0	2	3	4	7	9	12

The Blues scales are in fact extensions of the major and minor pentatonic scales[1]:

Pentatonic Minor:	0	3	5	7	10	12
Pentatonic Major:	0	2	4	7	9	12

Now a list of some popular heptatonic scales:

Diatonic Major:	0	2	4	5	7	9	11	12
Natural Minor:	0	2	3	5	7	8	10	12
Harmonic Minor:	0	2	3	5	7	8	11	12

A sequence of notes, played sequentially, defines the *melody* of a song. Three or more notes played simultaneously define a *chord*.

A.2 Chords

While melody can be seen as a sequence of notes, harmony is basically a sequence of chords, or *chord progression*. Some composers build the chord progression after the melody is finished. Others take the opposite direction. In any case, melody and harmony must combine, otherwise the music piece will not sound good. That is commonly obtained by taking melody and harmony from the same musical scale, at least when we are talking about the popular Diatonic Major, Natural Minor and Harmonic Minor scales (all of them heptatonics).

In these cases, the representation of musical scales described in Chapter 3 is not only useful for the melody. It can be used for the harmony as well.

The harmony for the mentioned scales is usually built using *triads* or *tetrads*, which are chords of three and four notes, respectively. For each note of the scale there is a corresponding chord (triad or tetrad), of which it will be the *root*.

The rule for building triads is simple: given a root, the 2nd note is the third, and the 3rd note, the fifth, up to the scale. The tetrad has one more note: the seventh note, from the root, up to the scale. Since there is redundancy of notes in the plane tessellation using heptatonic-scale tiles, many realizations of a particular chord are possible. Fig. A.1 shows two realizations of triads and two realizations of tetrads for each chord.

Chords are named depending on the root note and the intervals between the root and the remaining notes. For example, a triad with root C, 2nd note a minor

[1] The added note is called *blue note*.

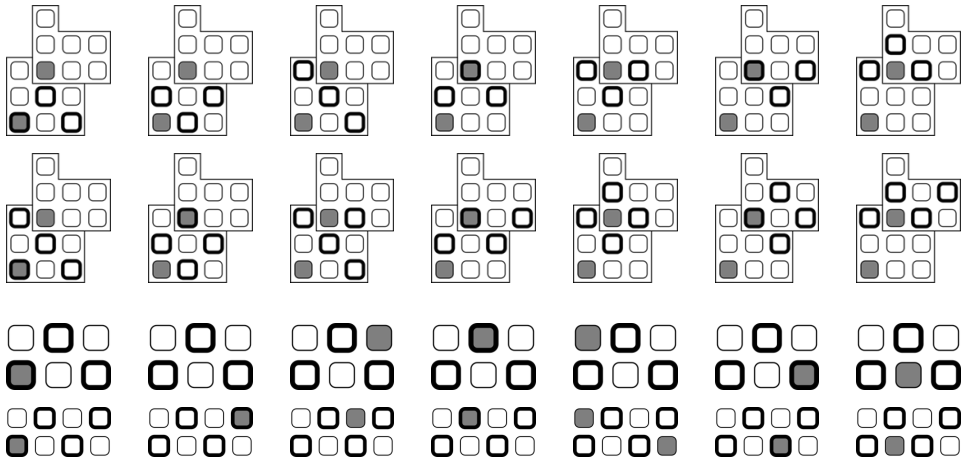


Fig. A.1: Triads and tetrads built from the notes of an heptatonic scale. At the top: realizations on two octave-consecutive tiles. At the bottom: realizations which preserve the pattern of the chord.

third away from the root, and 3rd note a perfect fifth away from the root, will be called C Minor, or Cm, for short.

A more detailed explanation on musical scales, chord progressions and chord names can be found in [27].

Appendix B

Machine Learning

Here we will briefly review some Machine Learning concepts we have used in Chapter 2. For more details, as well as for a general introduction on the subject, we recommend [3].

B.1 Maximum Likelihood Estimation

Suppose we want to implement a chord recognition algorithm using audio, i.e., the PCP vector $x = (x_1, ..., x_{12})$ as defined in Subsection 2.1.1.

Let C be the discrete probability distribution of chords, assuming values $c_1, ..., c_M$. In the training phase, we capture N samples from each of the chords. The classification method can be parametric or non-parametric, the latter being generally (much) more computationally expensive, so less suited for real-time applications.

Maximum Likelihood estimation is a parametric method which consists of computing the likelihoods of each chord given some sample and maximizing over the set of chords. Formally, being X the 12-dimensional random variable representing PCP vectors, and $p(X = x|C = c_j)$ the density function of such variable conditioned to the chord c_j evaluated at the point x, the algorithm return $c_{\hat{j}}$ as the recognized chord if

$$p(X = x|C = c_{\hat{j}}) = \max_{j=1,...,M} p(X = x|C = c_j)$$

The distribution of $X|C = c_j$ is usually assumed to be normal, so that $p(X = x|C = c_j)$ is a gaussian density function (of the variable x).

B.2 K-Means Clustering

In the Bayesian approach we described in Subsection 2.4.1, a clustering algorithm was necessary.

K Means is perhaps the simplest of them. It works as follows: (0) set initial K means; (1) associate each sample point to the nearest mean, this way defining clusters; (2) define the new means as the centroids of clusters; (3) iterate steps (1) and (2) up to convergence.

The question here, as usual, is about the value of K. One can run the method for many different values of K, analyzing the accuracy of the algorithm which uses clustering, and look for the "elbow" of the accuracy graph [3]. Or, in cases that the number of clusters is not of fundamental importance (what happens to be ours), choose some small value, for the sake of computational efficiency.

Appendix C

Pseudo-Score

Progress, cycle by cycle, of the music piece mentioned in Section 4.2:

1. The strings ensemble follows the chords played by the musician, as recognized by the computer vision system.
2. The drum loop number 1 is triggered by hitting the "Enter" button.
3. Musician starts fingering, keeping the shape of the chords, so the video-based chord recognition algorithm can work properly.
4. Musician starts strumming. Drum loop changes to level 2, a more intense level. Automatic composition algorithm starts at level of intensity 1.
5. Automatic composition algorithm goes to level of intensity 2.
6. Drum loop changes to number 3, the more intense level. Automatic composition algorithm goes to level of intensity 3, the greatest.
7. The number of restrictions to be satisfied by the sequence of notes increases. The musician should play the sequence of chords that will be repeated for the next two cycles.
8. Drum loop goes back to level 1. "Air Guitar" mode is turned on: the position of the hand indicates the region to which the improvised sequence of notes has to converge. Automatic composition algorithm goes back to level 2.
9. The parameters of the previous cycle are kept. The system remains in the "Air Guitar" mode to give it significant importance.
10. Control of the sequence of chords goes back to the computer vision system. Musician changes the guitar effect. "Air Guitar" mode is turned off. Automatic composition algorithm returns to level 3. Drum loop returns to level 3. Musician performs a sequence of chords different from the one of the previous cycle.
11. Parameters of the system are the same as in cycle 10. Musician performs

yet another sequence of chords.

12. Parameters of the system are the same as in cycle 11. Musician performs the first part of the riff of chords preparing the conclusion of the piece.
13. Parameters of the system are the same as in cycle 12. Musician performs the second part of the riff of chords preparing the conclusion of the piece.
14. Drum loop goes back to level 1. Musician positions the hand in the last chord of the piece and performs the last strum. This is the last cycle of the music piece.

Appendix D

Implementation

The experiments described in this text were performed using one of the two platforms:

- Macintosh MacMini, with a 1.66 GHz Intel Core Duo processor and 2GB of RAM memory, running Mac OS X version 10.6.
- Macintosh MacBook, with a 2 GHz Intel Core 2 Duo processor and 2GB of RAM memory, running Mac OS X version 10.6.

In Chapter 2, to capture audio and video we have used QuickTime, by means of the QTKit framework. Many video processing tasks were performed using the OpenCV computer vision library [7]. Almost all code has been written in Objective-C, using Xcode as de development environment. The system is able to work in real time, but for convenience we have taken the samples and worked on them in Matlab [48], to allow more flexibility in a prototyping stage. Audio sample rate is 44100 frames per second, the PCP descriptors being evaluated at intervals of 512 samples over a window of 1024 samples. Video sample rate is around 30 frames per second, and the concatenation is video synchronized, i.e., data fusion is performed every time a video frame is processed, using the last audio window available by that time.

Most of the experiments for Chapter 4 were implemented in Objective-C, using Xcode. The multi-touch table version of the application described in Subsection 3.5.1 was developed using Quartz Composer, and the smart-phone version was tested in an iPhone 3G device, running iPhone OS 3.0.

Appendix E

Publications

Most parts of this text have appeared elsewhere, and some by-products of the thesis work, which didn't fit in this report, have been published as well. Here is a list (in reverse chronological order) of works to which we have contributed during the Ph.D:

A. Schulz, M. Cicconet, L. Velho, B. Madeira, A. Zang and C. da Cruz. *CG Chorus Line*. 23th SIBGRAPI - Conference on Graphics, Patterns and Images: Video Festival. Gramado, 2010.

M. Cicconet and P. Carvalho. *Playing the QWERTY Keyboard*. 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques: Poster section. Los Angeles, 2010.

A. Schulz, M. Cicconet and L. Velho. *Motion Scoring*. 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques: Poster section. Los Angeles, 2010.

M. Cicconet, L. Velho, P. Carvalho and G. Cabral. *Guitar-Leading Band*. 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques: Poster section. Los Angeles, 2010. (Student Research Competition semifinalist.)

M. Cicconet, I. Paterman, L. Velho and P. Carvalho. *On Multi-Touch Interfaces for Music Improvisation: The Blues Machine Project*. Technical Report TR-2010-05. Visgraf/IMPA, 2010.

A. Schulz, M. Cicconet, B. Madeira, A. Zang and L. Velho. *Techniques for CG Music Video Production: the making of Dance to the Music / Play to the Motion*. Technical Report TR-2010-04. Visgraf/IMPA, 2010.

M. Cicconet, P. Carvalho, I. Paterman and L. Velho. *Método para Representar*

Escalas Musicais e Dispositivo Eletrônico Musical [Brazilian patent application]. 2010. Deposited at INPI.

M. Cicconet, P. Carvalho and L. Velho. *On Bimodal Guitar-Chord Recognition*. Proceedings of the International Computer Music Conference. New York, 2010.

M. Cicconet, T. Franco and P. Carvalho. *Plane Tessellation with Musical Scale Tiles and Bidimensional Automatic Composition*. Proceedings of the International Computer Music Conference. New York, 2010.

M. Cicconet and P. Carvalho. *The Song Picture: on Musical Information Visualization for Audio and Video Editing*. Proceedings of the International Conference on Information Visualization Theory and Applications. Angers, 2010.

M. Cicconet, M. Gattass, L. Velho and P. Carvalho. *Visual Pitch Class Profile: A Video-Based Method for Real-Time Guitar Chord Identification*. Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. Angers, 2010.

M. Cicconet, S. Krakowski, P. Carvalho and L. Velho. *Métodos Matemáticos e Computacionais em Música*. 32th Congresso Brasileiro de Matemática Aplicada e Computacional. Book and Mini-Course. Cuiabá, 2009.

M. Cicconet, I. Paterman, L. Velho and P. Carvalho. *The Blues Machine*. 22nd Brazilian Symposium on Computer Graphics and Image Processing: Video Festival. Rio de Janeiro, 2009.

M. Cicconet and P. Carvalho. *EigenSound: Sound Visualization for Edition Purposes*. 22nd Brazilian Symposium on Computer Graphics and Image Processing: Poster Section. Rio de Janeiro, 2009.

M. Cicconet, I. Paterman, L. Velho and P. Carvalho. *The Blues Machine*. 36th International Conference and Exhibition on Computer Graphics and Interactive Techniques. Poster and Talk. New Orleans, 2009.

M. Cicconet. *BluesMachine*. iPhone and iPod Touch software. Apple's App Store, 2009.

M. Cicconet, L. Velho and I. Paterman. *Relativistic Visualization*. 20th Brazilian Symposium on Computer Graphics and Image Processing: Video Festival. Belo Horizonte, 2007. (Best video award.)

Bibliography

[1] Bozhidar Abrahev. *The Illustrated Encyclopedia of Musical Instruments: From All Eras and Regions of the World*. Könemann, 2000.

[2] All-Guitar-Chords. *All-Guitar-Chords*. `http://www.all-guitar-chords.com/`, last checked Mar 03 2010.

[3] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2004.

[4] R. Ash. *Lectures on Statistics*. `http://www.math.uiuc.edu/~r-ash/Stat.html`, last checked Apr 18 2010.

[5] Dave Benson. *Music: a Mathematical Offering*. Cambridge University Press, 2006.

[6] D. R. Bland. *Vibrating Strings: An Introduction to the Wave Equation*. Routledge and Kegan Paul, London, 1960.

[7] Gary Bradski. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[8] A. Burns and M. Wanderley. Visual methods for the retrieval of guitarist fingering. In *International Conference on New Interfaces for Musical Expression*, 2006.

[9] Giordano Cabral. Inpact of distance in pitch class profile computation. In *Simpósio Brasileiro de Computação Musical*, Belo Horizonte, 2005.

[10] Giordano Cabral. *Harmonisation Automatique en Temps Reel*. PhD thesis, Université Pierre et Marie Curie, 2008.

[11] A. Cheveigné and H. Kawahara. Yin, a fundamental frequency estimator for speech and music. *Journal of the Acoustic Society of America*, 111(4), April 2001.

[12] Chordbook. *Chordbook.com: Interactive Chords, Scales, Tuner*. `http://www.chordbook.com/index.php`, last checked Mar 03 2010.

[13] M. Cicconet and P. Carvalho. The song picture: On musical information visualization for audio and video editing. In *International Conference on Information Visualization Theory and Applications*, Angers, France, 2010.

[14] M. Cicconet, P. Carvalho, I. Paterman, and L. Velho. Método para representar escalas musicais e dispositivo eletrônico musical. Brazilian Patent Application [Deposited at INPI], 2010.

[15] M. Cicconet, T. Franco, and P. Carvalho. Plane tessellation with musical scale tiles and bi-dimensional automatic composition. In *International Computer Music Conference*, New York and Stony Brook, USA, 2010.

[16] Richard Cole and Ed Schwartz. *Virginia Tech Multimedia Music Dictionary*. `http://www.music.vt.edu/musicdictionary/`, 2009.

[17] P. de la Quadra, A. Master, and C. Sapp. Efficient pitch detection techniques for interactive music. Technical report, Center for Computer Research in Music and Acoustics, Stanford University, 2001.

[18] R. Fiebrink, G. Wang, and P. Cook. Don't forget your laptop: Using native input capabilities for expressive music control. In *International Conference on New Interfaces for Musical Expression*, 2007.

[19] Neville H. Fletcher and Thomas D. Rossing. *The Physics of Musical Instruments*. Springer-Verlag, New York, second edition, 1998.

[20] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[21] Wikimedia Foundation. *Wikipedia: The Free Enciclopedia*. `http://www.wikipedia.org/`, 2010.

[22] C. Frisson, L. Reboursière, W. Chu, O. Lähdeoja, J. Mills, C. Picard, A. Shen, and T. Todoroff. Multimodal guitar: Performance toolbox and study workbrench. Technical report, Numediart Research Program, 2009.

[23] Takuya Fujishima. Real-time chord recognition of musical sound: A system using common lisp music. In *International Computer Music Conference*, 1999.

[24] Jonas Gomes and Luiz Velho. *Computação Gráfica: Imagem*. IMPA, 2002.

[25] E. Guaus, T. Ozasian, E. Palacios, and J. Arcos. A left hand gesture caption system for guitar based on capacitive sensors. In *International Conference on New Interfaces for Musical Expression*, Sydney, 2010.

[26] T. Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, and W. Verplank. *ACM SIGCHI Curricula for Human-Computer Interaction*. `http://old.sigchi.org/cdg/`, last checked Jul 05 2010.

[27] Michael Hewitt. *Music Theory for Computer Musicians*. Course Technology, Boston, Massachusetts, 2008.

[28] Apple Computer Inc. *Apple Computer Inc.* `http://www.apple.com/`, last checked Apr 30 2010.

[29] Curious Brain Inc. *TouchChords*. Apple's App Store, last checked April 29 2010.

[30] JazzMuttant. *JazzMuttant*. `http://www.jazzmutant.com/`, last checked Apr 30 2010.

[31] Tristan Jehan. *Creating Music by Listening*. PhD thesis, Massachusetts Institute of Technology, 2005.

[32] M. Kaltenbrunner. *Tangible Music*. `http://modin.yuri.at/tangibles/`, last checked Apr 30 2010.

[33] M. Kaltenbrunner and R. Bencina. reactivision: A computer vision framework for table based tangible interaction. In *First international conference on Tangible and embedded interaction*, Baton Rouge, 2007.

[34] Chutisant Kerdvibulvech and Hideo Saito. Vision-based guitarist fingering tracking using a bayesian classifier and particle filters. In *Advances in Image and Video Technology*, Lecture Notes in Computer Graphics. Springer, 2007.

[35] Anssi Klapuri and Manuel Davy (Editors). *Signal Processing Methods for Music Transcription*. Springer, New York, 2006.

[36] Gareth Loy. *Musimathics: The Mathematical Foundations of Music*, volume 1. The MIT Press, Cambridge, Massachusetts, 2006.

[37] John Maeda. *The Laws of Simplicity*. MIT Press, 2006.

[38] Erin McKean. *New Oxford American Dictionary*. Oxford University Press, second edition, 2005. As Dictionary, Mac OS X Software, by Apple Inc. 2009.

[39] P. McLeod and G.Wyvill. A smarter way to find pitch. In *International Computer Music Conference*, 2005.

[40] H. B. Mitchell. *Multi-Sensor Data Fusion: An Introduction*. Springer, Berlin Heidelberg, 2007.

[41] C-Thru Music. *The AXiS-49 Harmonic Table Music Interface*. `http://www.c-thru-music.com/cgi/`, last checked Apr 30 2010.

[42] Gerhard Nierhaus. *Algorithmic Composition: Paradigms of Automated Music Generation*. SpringerWienNewYork, 2009.

[43] F. Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, pages 333–341, Sep 2003.

[44] M. Paleari, B. Huet, A. Schutz, and D. Slock. A multimodal approach to music transcription. In *15th International Conference on Image Processing*, 2008.

[45] G. Papadopoulos and G. Wiggins. Ai methods for algorithmic composition: a survey, a critical view and future prospects. In *AISB Symposium on Musical Creativity*, pages 110–117, 1999.

[46] Tae Hong Park. *Introduction to Digital Signal Processing: Computer Musically Speaking*. World Scientific, 2010.

[47] G. Peeters. A large set of audio features for sound description (similarity and classification). Technical report, IRCAM: Analysis/Synthesis Team, 2004.

[48] R. Pratap. *Getting Started With Matlab 5: A Quick Introduction for Scientists and Engineers*. Oxford University Press, 1999.

[49] G. Quested, R. Boyle, and K. Ng. Polyphonic note tracking using multimodal retrieval of musical events. In *International Computer Music Conference*, 2008.

[50] Reactable. *Reactable*. `http://www.reactable.com/`, last checked Jun 22 2010.

[51] L. Reboursière, C. Frisson, O. Lähdeoja, J. Mills, C. Picard, and T. Todoroff. Multimodal guitar: A toolbox for augmented guitar performances. In *International Conference on New Interfaces for Musical Expression*, Sydney, 2010.

[52] Curtis Roads. *The Computer Music Tutorial*. The MIT Press, Cambridge, Massachusetts, 1996.

[53] Justin Romberg. *Circular Convolution and the DFT*. `http://cnx.org/content/m10786/2.8/?format=pdf`, 2006.

[54] Ilya Rosenberg and Ken Perlin. The unmousepad: an interpolating multi-touch force-sensing input pad. *ACM Transactions on Graphics*, 2009.

[55] A. Schulz, M. Cicconet, and L. Velho. Motion scoring. In *37th International Conference and Exhibition on Computer Graphics and Interactive Techniques*, 2010.

[56] Jumpei Wada. *MiniPiano*. Apple's App Store, last checked April 29 2010.

[57] Joe Wolfe. *Note Names, MIDI Numbers and Frequencies*. `http://www.phys.unsw.edu.au/jw/notes.html`, last checked Feb 28 2010.

Index