PCM TO MIDI TRANSPOSITION

Luís Gustavo Pereira Marques Martins

Licenciado em Engenharia Electrotécnica e Computadores
pela
Faculdade de Engenharia da Universidade do Porto

Dissertação submetida para satisfação parcial dos requisitos do grau de mestre em
Engenharia Electrotécnica e de Computadores
Área de especialização de Telecomunicações

Dissertação realizada sob a supervisão do Professor Doutor Aníbal João de Sousa Ferreira,
do Departamento de Engenharia Electrotécnica e de Computadores
da Faculdade de Engenharia da Universidade do Porto

Porto, Setembro de 2001
ABSTRACT

The extraction of semantic features from audio signals is a research area that is of the most interest for the automatic classification, indexation and retrieval of audio material. Furthermore, real-time interactive systems that use sound as an input may also benefit from the development of such technologies in order to achieve a better interaction with "real-world" events.

This dissertation addresses the analysis of acoustical music signals, and in particular, its transcription to musical notation. The development and implementation of an automatic transcription system are presented and the results obtained are evaluated and discussed.
RESUMO

A extracção de características semânticas a partir de sinais áudio é uma área de investigação que se tem revelado ser do maior interesse para a classificação, indexação e pesquisa automáticas de conteúdos áudio. Por outro lado, sistemas interactivos em tempo-real, que utilizem o som como entrada, podem também beneficiar do desenvolvimento de tais tecnologias de forma a atingir uma melhor interacção com os eventos do “mundo real”.

Esta dissertação aborda a análise de sons musicais e, em particular, a sua transcrição para a notação musical. É apresentado o desenvolvimento e implementação de um sistema de transcrição automática de música e os resultados obtidos são avaliados e discutidos.
PRÉCIS

L'extraction des caractéristiques sémantiques résultantes des signaux d'audio c'est un domaine de recherche que s'est rendu un des plus intéressants pour la classification, l'indexation et les quêtes automatiques de contenus d'audio. En outre, des systèmes interactifs en temps-réel qui utilisent du son comme entrée peuvent aussi être bénéficiés par le développement de telles technologies de façon qu'elles atteignent une meilleure interaction avec les événements du "monde réel".

Cette dissertation adresse l'analyse des sons musicaux et surtout sa transcription pour la notation musicale. On présente le développement et l'implémentation d'un système de transcription automatique de musique. Finalement, les résultats obtenus sont évalués et discutés.
The idea of having machines that analyse and process the reality in the same abstract way humans do, has challenged and inspired many research projects since the beginning of the electronic era. The development of systems that could respond to stimulus and react in a similar way a human would do, are still considered the holy grail for many research areas. These perceptual machines could be of extreme interest for developing truly interactive applications, where human users could query the systems using free-text or free-spoken questions or by using examples (i.e. “Get me the song that has the following guitar line” or “Get me a photo similar to this one”) or for implementing supervising systems that would be able to judge and react in the presence of specific events (i.e. surveillance systems that would able to distinguish the sound of a breaking glass and inspect if a window has been violated).

Until today, vision and hearing have been the two human senses most studied, perhaps based in the idea that humans acquire 80% of their knowledge through the sense of vision and 11% through the sense of hearing [Fluckiger95]. Several psychological studies have been conducted in order to understand how humans acquire and process visual and audio information, and the results have just now started being used in computer systems. These perceptual systems try to extract features from audio and video signals that are representatives of the cues the human brain uses to interpret such stimuli.

This thesis addresses the automatic transcription of polyphonic music, where the ultimate objective is to convert a digitally recorded piece of music into a symbolic representation. This requires the extraction of notes, their pitches, timings and dynamics.

To study and review these problems, it was necessary to use results from other science branches such as Psychoacoustics (that tries to understand the mechanism of sound perception), Computational Auditory Scene Analysis (that has a somewhat wider scope, covering the interpretation of the numerous distinct events that result from the analysis of a physical environment’s acoustical information) and Digital Signal Processing (that deals with the digital representation and processing of signals, and the use of computers to analyse, modify and extract information from signals).

The resulting symbolic representation of the musical signal is a higher order description that gets closer to the way humans understand it, allowing higher levels of analysis, manipulation, storage, comparison and retrieval. In this work, MIDI (Musical Instrument Digital Interface) is the formal syntax used for the description of the high order symbolic representation of musical signals, since it is a well-established industry standard, specifically created for this purpose.

---

1 Although only 20% of what is seen is kept, against 30% of what is heard. Thought, humans keep 50% from what is simultaneously seen and heard [Fluckiger95].
ACKNOWLEDGEMENTS

This work has been developed in the *Telecommunications Unit – Audio Group of INESC Porto*, to whom I am especially grateful for giving me the opportunity to collaborate with projects as interesting as the ones I have been involved with.

I obviously owe a lot of gratitude to Prof. Antón Ferreira, who, besides leading my academic career for the last few years, invited me to join *INESC Porto’s Audio Group*, giving me the chance to be immersed in such an overwhelming environment.

A great word of thanks also goes to my friends and colleagues at the *INESC Porto Audio Group*, Miguel Falcão and Gabriel Fernandes, for their constructive comments and suggestions to my work, and for the accumulated experience acquired by working with them on *INESC Porto Audio Group’s* projects.

I would also like to thank all my friends that, even without knowing it, contributed with their happiness and friendship during those relaxation moments spent in their company, so important to keep the motivation and determination I needed during the most difficult moments that naturally make part of such a project.

A special honour to the memory of an irreplaceable and close friend that destiny took away from this life too soon, but whose memory will always be a strong inspiration of beauty, honesty and happiness: I dedicate this Work to you, Sara.

Finally a huge thank you goes to my parents and grandparents for their never-ending support and encouragement, mainly in the most difficult moments. This Thesis is also dedicated to them!

READER NOTES

The font used in normal text is Times New Roman. *Courier New* is used for computer code.

Several acronyms are used across the text. In order not to clutter the text with long definitions, the reader is asked to refer to the acronym list on the beginning of this text.

Bibliographic references are listed at the end of the document. Whenever referred to on the text, a tag between parentheses (composed by the author’s last name and year of publication) will identify them.
TABLE OF CONTENTS

Abstract ...................................................................................................................... 1
Resumo........................................................................................................................ 2
Précis .......................................................................................................................... 3
Preface ........................................................................................................................ 4
Acknowledgements ..................................................................................................... 5
Reader Notes ............................................................................................................... 5
Table of Contents ...................................................................................................... 6
List of Figures ............................................................................................................ 8
List of Tables ............................................................................................................ 9
Acronyms .................................................................................................................. 10
1. Introduction ......................................................................................................... 11
  1.1. Characterization of the Problem ............................................................... 11
  1.2. An Historical Perspective ........................................................................... 11
  1.3. Related Areas of Research ........................................................................ 13
  1.4. Applications .................................................................................................. 13
  1.5. Objectives of this Work ............................................................................... 14
2. Characterization of Musical Signals .................................................................... 16
  2.1. The Relevance of the Note as a Musical Symbol ....................................... 16
  2.2. Properties and Features of Acoustical Musical Signals ....................... 17
      2.2.1. Pitch ........................................................................................................ 18
      2.2.2. Loudness ................................................................................................ 19
      2.2.3. Tempo and Rhythm .............................................................................. 19
      2.2.4. Timbre .................................................................................................... 20
3. Mid-level Representations of Musical Signals .................................................. 21
4. Selected Approach and System Design .............................................................. 24
  4.1. System Overview ......................................................................................... 24
  4.2. Frequency Analysis ...................................................................................... 26
      4.2.1. Analysis Front-end Overview ............................................................... 26
      4.2.2. The ODFT ............................................................................................ 27
      4.2.3. The Analysis / synthesis Window ....................................................... 28
  4.3. Harmonic Analysis ....................................................................................... 30
      4.3.1. Overview ............................................................................................... 30
      4.3.2. Identification of Spectral Peaks ........................................................... 31
      4.3.3. Interpolation of Peak Frequency ....................................................... 32
      4.3.4. Detection of Harmonic Structures ................................................... 35
  4.4. Tracking of Harmonic Structures ............................................................... 39
      4.4.1. Overview ............................................................................................... 39
      4.4.2. Frequency Continuation Algorithm .................................................. 41
      4.4.3. The Detection Delay ............................................................................ 45
  4.5. Post-Processing ............................................................................................. 46
      4.5.1. Transient Detector ............................................................................... 46
      4.5.2. Trajectory On-Set Time Adjust ......................................................... 48
      4.5.3. Clustering and Pruning of Trajectories ............................................. 49
      4.5.4. Transcription Output ........................................................................... 58
5. Tests and Transcription Results of the System .................................................. 59
  5.1. Transcription of Monophonic Musical Signals .......................................... 59
LIST OF FIGURES

Figure 4.1 - System Overview ................................................................. 24
Figure 4.2 - Simulation and Development Environment ......................................... 25
Figure 4.3 - Frequency analysis front-end .................................................... 26
Figure 4.4 - Normalized frequency response of the analysis window ....................... 29
Figure 4.5 - Harmonic Analysis overview ................................................... 30
Figure 4.6 - Peak detection .................................................................. 32
Figure 4.7 - Frequency responses of the first four channels of the ODFT filter bank .... 33
Figure 4.8 - Relation between the magnitudes of the ODFT channels $\ell - 1$, $\ell$ and $\ell + 1$ when the input signal is a sinusoid whose frequency is given by $\frac{2\pi}{N}(\ell + \Delta\ell)$ ..................................................... 34
Figure 4.9 - Harmonic Structure detection .................................................... 39
Figure 4.10 - Detected Harmonic Structures ................................................. 40
Figure 4.11 - Trajectory structure .............................................................. 41
Figure 4.12 - Trajectory list .................................................................... 41
Figure 4.13 - Frequency Continuation Algorithm ............................................ 43
Figure 4.14 - Detection of the trajectories defined by the harmonic structures of Figure 4.10 ......................................................... 45
Figure 4.15 - Transient Detector release parameter ........................................... 48
Figure 4.16 - Trajectory on-set time adjust .................................................... 49
Figure 4.17 - Definition of Time Clusters ...................................................... 50
Figure 4.18 - Definition of Harmonic Clusters ............................................... 51
Figure 4.19 - Frequencies of octave related musical notes .................................. 52
Figure 4.20 - Trajectory Pruning criteria ....................................................... 54
Figure 4.21 - Harmonic Cluster filtering result ............................................... 56
Figure 4.22 - Time Cluster filtering output ..................................................... 57
Figure 4.23 - Trajectory Pruning output ........................................................ 58
Figure 5.1 - MIDI score used to generate the monophonic musical signal ............ 60
Figure 5.2 - Mid-level representation of the monophonic musical signal ............... 60
Figure 5.3 - MIDI score of the monophonic musical signal transcription ............... 60
Figure 5.4 - MIDI score used to generate a polyphonic musical signal ............... 62
Figure 5.5 - Mid-level representation of the polyphonic musical signal of Figure 5.4 ................................. 62
Figure 5.6 - MIDI score of the transcription of the polyphonic signal of Figure 5.4 ...... 62
Figure 5.7 - MIDI score used to generate a polyphonic multitimbric musical signal .... 63
Figure 5.8 - Mid-level representation of the multitimbric polyphonic musical signal of Figure 5.7 .......................................................... 63
Figure 5.9 - MIDI score of the transcription of the multitimbric polyphonic signal of Figure 5.7 .......................................................... 63
Figure 5.10 - MIDI score used to generate an ambiguous polyphonic musical signal .... 65
Figure 5.11 - Mid-level representation of the ambiguous polyphonic musical signal .... 66
Figure 5.12 - MIDI score of the transcription of the ambiguous polyphonic signal of Figure 5.10 .......................................................... 66
Figure 5.13 - Original music score ................................................................ 67
Figure 5.14 - Music score of the transcription of the musical piece presented in Figure 5.13 .......................................................... 67
Figure 6.1 - Frequency relations between musical notes of equal temperament scales ........ 74
LIST OF TABLES

Table 6.1 - MIDI Note numbers and their corresponding frequencies ........................................ 75
Table 6.2 - MIDI File Chunk ..................................................................................................... 77
Table 6.3 - MIDI File structure ................................................................................................. 77
Table 6.4 - MThd Header Chunk ............................................................................................... 78
Table 6.5 - MIDI File Division ................................................................................................. 79
Table 6.6 - MTrk Header Chunk ............................................................................................... 80
Table 6.7 - MIDI Channel voice messages ................................................................................ 84
Table 6.8 - MIDI Channel mode messages .............................................................................. 86
# ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA</td>
<td>Auditory Scene Analysis</td>
</tr>
<tr>
<td>CASA</td>
<td>Computational Auditory Scene Analysis</td>
</tr>
<tr>
<td>CQT</td>
<td>Constant Q Transform</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>MDCT</td>
<td>Modified Discrete Cosine Transform</td>
</tr>
<tr>
<td>MIDI</td>
<td>Musical Instrument Digital Interface</td>
</tr>
<tr>
<td>ODFT</td>
<td>Odd Discrete Fourier Transform</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse Code Modulation</td>
</tr>
<tr>
<td>SMF</td>
<td>Standard MIDI File</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1. CHARACTERIZATION OF THE PROBLEM

Transcription of Music can be defined as the act of listening to a piece of music and of writing down musical notation for the notes that constitute the piece. This implies the extraction of specific features out of a musical acoustic signal, resulting in a symbolic representation that comprises notes, their pitches, timings, dynamics and may include the identification of the instruments used.

Humans without musical education have usually much more difficulties to transcribe polyphonic music\(^2\) than musically skilled persons. Experience in musical style, instrument sounds and knowledge of music theory can give listeners the ability to resolve even the most complex and rich polyphonies with regard to the variety of instrumental sounds and musical styles, leaving the performance of automatic transcription systems clearly behind.

Automatic Transcription of Music has interested musicians and computer scientists for over twenty-five years.

Monophonic transcription solutions have matured over the years with many well-understood algorithms, including time-domain techniques based on zero-crossing and autocorrelation, as well as frequency-domain techniques, based on the discrete Fourier transform and the cepstrum [Brown&Zhang91, Brown92, Brown&Puckette93]. These algorithms proved to be reliable, commercially applicable and most of them can operate in real-time, having been adopted massively in the area of speech processing.

Polyphonic music transcription has enjoyed much less relative success, maybe because of the increased complexity of the signals in question. Furthermore, music recognition research, when compared with the speech coding and recognition areas, has not yet attracted the desirable attention to its commercial potentialities.

1.2. AN HISTORICAL PERSPECTIVE

The first attempts towards automatic polyphonic music transcription date back to the 1970s, when Moorer [Moorer75, 77] and Piszczalski and Galler [Piszczalski&Galler77] developed the first transcription systems, and coined contemporaneously the term automatic music transcription.

The work of Piszczalski and Galler [Piszczalski&Galler77] focused only on single instrument analysis, and only on instruments with a relatively strong fundamental frequency. Their system operated on an FFT front-end, and tried to measure the fundamental directly from the spectrogram.

\(^2\) During this text, the term polyphonic music refers to musical pieces with multiple simultaneously sounding notes, while monophonic music refers to musical pieces where only one note is played at each time instant.
Moorer’s system [Moorer77] was limited to musical pieces with a maximum of two instruments with distinct timbres and note ranges, and imposed restrictions to the allowable simultaneous musical intervals. Furthermore, it was unable to detect octaves or any other intervals in which the fundamental frequency of the higher note corresponded to the frequency of one of the overtones of the lower note. Moorer’s work was carried on by a group of researchers at Stanford in the beginning of the 1980s [Chafe82, 85, 86] and further developments were made latter on by Maher [ Maher89, 90]. Though, polyphony was still restricted to two voices and the range of fundamental frequencies for each voice was limited to non-overlapping intervals.

Systems capable of transcribing more than two simultaneously musical voices were demonstrated only in the last few years.

In 1993, Hawley presented a system that assumedly could transcribe polyphonic piano performances based on a differential spectrum analysis [Hawley93]. He used a short-time spectral analysis and spectral comb filtering to extract note spectra, and looked for note on-sets in the high-frequency power and with bilinear time-domain filtering. The system was reported to be fairly successful, although with very limited scope (only getting good results with piano sounds), showing that transcription systems can be successful by narrowing the range of input signals they consider, and by relying on specific characteristics of those signals, which may not be present in a more general class of signals [Martin96].

Kashino et al. and Martin presented systems based on blackboard frameworks, integrating front-ends based on sinusoidal analysis with musical knowledge [Kashino95, Martin96a]. Though, these systems rely on instrument models for detecting octaves. Martin also presented a modified system that uses an autocorrelation-based front-end, that tries to make bottom-up detection of octaves possible [Martin96b].

On his MSc thesis, Scheirer extended the idea of incorporating musical knowledge into a polyphonic transcription system [Scheirer95]. By using the score of the music as a guide, he demonstrated reasonably accurate transcription of overlapping four- and six-voice piano music. The system used both frequency-domain and time-domain methods to track partials and detect on-sets.

Enríquez developed his PhD thesis work on the theme of automatic transcription of polyphonic music [Enríquez98], presenting a system assumedly independent of the type and combinations of acoustical instruments, and based on low-cost computational techniques possible to implement as real-time applications. His strategy used a sinusoidal model to extract sets of partials that were expected to describe the polyphonic signal.

Goto presented a system capable of extracting melody and bass lines from real-world recordings [Goto2000]. His method tries to estimate the fundamental frequency of the most predominant harmonic structure in sound mixtures containing simultaneous sounds of various instruments.

Klapuri et al. propose an algorithm based on an iterative estimation and separation procedure, being able to resolve at least a couple of most prominent pitches up to ten sound polyphonies [Klapuri98, Klapuri2000a, Klapuri2000b].
1.3. RELATED AREAS OF RESEARCH

There are several other fields of science related to the problem of Automatic Transcription of Music.

Research on Psychoacoustics tries to explain the relationship between acoustic sound signals, the psychology of the human hearing system and the perception of sound. Psychoacoustical thinking dates back to the ancient Greeks, when Pythagoras recognized that strings whose lengths are related as the ratio of small integers sound good when plucked together.

Already on the 20th century, Stevens, Moore and Fletcher have contributed to the evolution of modern psychoacoustics, allowing a sophisticated understanding of the psychophysics of audition [Stevens57, Moore89, Fletcher40]. With their contributions, robust and well-tested models have been developed, mainly focusing on basic perceptual features (like pitch and loudness) of simple stimuli, and on the way in which a simple sound masks another, depending on the time-frequency relationship between the two sounds.

Recently, a research focus has developed to study the perceptual grouping and segregation of sounds under the broad heading of Auditory Scene Analysis (ASA). ASA refers to the analysis of the acoustic information produced in a physical environment followed by the interpretation of the numerous distinct events in it.

Albert Bregman’s three-decade research work on ASA has been compiled on a book [Bregman90] that is widely referenced in the branches of computer science that are related to auditory perception.

The use of computer systems in ASA gave birth to CASA: Computational Auditory Scene Analysis. The work of Daniel Ellis [Ellis92, 96] is one of the most up to date researches on CASA. His latest studies comprise prediction-driven processing that use predictions of an internal world model and higher-level knowledge. He evaluated a computational model, and obtained a good agreement between the events detected by the model and by human listeners.

Also closely related to the automatic transcription of music are the processes of identification of the musical instruments playing in a musical piece [Serra89, Martin98a, 98b] and the automatic extraction of tempo and beat of a musical performance [Scheirer98]. Besides the obvious interest in knowing the instruments playing in a musical piece as well as its tempo and time signature, these results can also help to resolve ambiguities in the transcription process, mimicking the way the human brain combines information from several sources of knowledge in order to achieve the most accurate perceptual image possible.

1.4. APPLICATIONS

The applications for music perception systems are numerous, but still limited by the low reliability of the results presented by current solutions.

Nevertheless, it is easy to point some fields of application for current and future music perception systems, such as:
- **Music transcription systems.** These systems are mainly of interest to music composers and musicians, who could use it to efficiently analyse compositions they only have in the form of acoustic recordings. The resulting symbolic representation allows flexible and selective musical analysis, editing and mixing, otherwise difficult or impossible to perform.

- **Synthetic Performance Systems.** With the ability to hear music, systems for enabling musical collaboration between humans and machines could become more expressive and capable.

- **Algorithmic Composition.** Current computer-composition systems still need to have their outputs evaluated by humans so that they can be adjusted in order to create meaningful and interesting musical pieces. The use of human perception models may enable machines to evaluate and critique their own compositions, turning it possible to attain a more truly automated composition process.

- **Visual Music Displays.** Real-time computer graphic routines can generate compelling multimedia experiences when synchronized with music. By enabling real-time music listening as a component of such a system, the display can be made to work interactively with any music input (i.e. musical improvisations).

- **Access to musical databases.** Using robust automated music perception solutions, systems could be built to augment Internet-based music recommendation services with the ability to make musical judgments. Music databases could also benefit from the automatic music transcription and recognition as a way to improve indexation, classification and retrieval of information. Users of such databases could query the system using examples or higher order descriptions of the music pieces they were looking for.

- **Structured Audio Encoding.** The theory of musical listening could shed light on structural regularities in music that could be used in structured audio coding. This is not only in the case of music transcription for generating note lists, but in more subtle ways such as identifying perceptually equivalent sounds for coding. As loudness and masking models led to the development of low-bit rate perceptual coders [Ferreira98], so can psychoacoustic music processing pave roads to new techniques for audio coding and compression [Ferreira2001a].

- **Automatic Teaching Systems.** A new generation of assisted music teaching tools could include the capability of automatic music recognition, allowing the system to hear and evaluate the performance of a musical instrument student (even at distance). The system would be able to analyse the notes played, their pitch precision, their timings, dynamics and even the expressiveness of the notes (i.e. staccatos, sforzandos, vibratos, etc), and then criticise and make suggestions to the student’s performance.

### 1.5. Objectives of this Work

This thesis addresses the problem of automatic transcription of polyphonic music, and presents the development and implementation of a **PCM to MIDI transposition system.**

The proposed system analyses digitally recorded musical signals (in PCM format) and attempts to extract features relevant to the determination of the notes played, their pitches, timings and dynamics. These parameters are then represented as MIDI information, a standard way of
representing, transmitting, storing and playing musical information using digital musical instruments [see Appendix B].

It is important to remark that only harmonic sounds will be considered by the presented transcription system, leaving out of the classification process sounds like the resulting from drums or percussion instruments.

The identification of instruments is not covered in the current version of the system, mainly due to time limitations and because this is a problem that can be, on its own right, a subject for a dissertation or research work [Serra89, Martin98a, Martin98b]. Nevertheless, the proposed signal analysis front-end already implements functions that may be used to include automatic instrument identification/separation capabilities in future versions of the system.

Similarly, in the current version of the system no concerns were taken in the determination of musical tempo and beat, paying attention only to the accurate extraction of notes, their on-set times and durations.

The following chapters will address in detail the problems posed by the automatic music transcription and will discuss the solutions that allowed the development of an automatic music transcription system.

Chapter 2 characterizes the specificities of the audio signals that result from the acoustical interpretation of musical pieces, with the objective of evidencing the features that should be addressed more incisively by the processing algorithms of a music transcription system.

Chapter 3 presents some possible mid-level representations of musical signals, and discusses their suitability for an accurate and efficient analysis of musical signals.

In chapter 4, the approach and the implementation solutions used in the music transcription system that led to this work will be addressed and discussed in detail.

In chapter 5, the results of some tests and simulations of the transcription system proposed in this thesis are presented and evaluated.

Chapter 6 goes over the main challenges and results of our work, and tries to foresee the evolution of the music transcription system presented in this thesis.

\[1\] In the following chapters a formal definition of harmonic sounds will be presented. Meanwhile, it is possible to set a coarse definition that may serve the purposes of this section’s context: harmonic sounds are those sounds where pitch is easily perceived.
2. CHARACTERIZATION OF MUSICAL SIGNALS

"Music has something to do with Sound."

"Music is the organization of aural space and sounds at some level of consciousness."

"Music is the structured organization of tones into melodic, harmonic, and rhythmic patterns."

The definition of music has been subject of enthusiastic and long discussions among musicians, music theorists and specialists. However, today it is still difficult to find a definition that collects a representative and sustaining support from the music expert’s community.

Overlooking the philosophical and stylistic quests for the widest descriptive and correct definition of music, each of the above definitions, as well as many others published in several specialised texts, can be a valid choice depending on the scope of its use.

For the purpose of the discussion of this thesis, the concern is not so much in the music definition itself, but more in its acoustical instantiation. More precisely, this work concerns the definition of a basic symbolic representation and its extraction out of the acoustical evidence of the musical event.

2.1. THE RELEVANCE OF THE NOTE AS A MUSICAL SYMBOL

The aforementioned symbolic representation of music calls for the discussion of what symbols should make part of it. Traditionally, Western musical notation has defined the note as its fundamental representational symbol, referring it to the sound produced by a musical instrument. This note definition usually includes features such as pitch, loudness, rhythm (here representative of a note’s time related information such as its duration, onset time and offset time, among others) and, in some cases, even timbre. This notation is usually known as a musical score, and its ultimate purpose is to allow the acoustical reproduction of the original musical idea, whose faithfulness depends ultimately on the score accuracy.

However, the use of the note as the fundamental representation symbol of music perception is far from being consensual. Experimental evidences indicate that, instead of notes, humans extract auditory cues that are then grouped into percepts, and that most stages of music perception have nothing to do with notes for most listeners [Scheier2000, p.67-69].

Ellis states that the brain or any other source separator will fail to analyse a mixed sound back into precise waveforms that were added together [Ellis92]. Instead, it is his opinion that the brain does form some idea of what the individual sources would sound like alone, proving meaningful to try and reconstruct a signal out of mixture which exhibits all the qualities

4 Considering a virtuoso musical execution of the score in question.

5 Percepts can be defined as the brain’s images of the distinct acoustical sources present in a sound. They are only impressions or reduced representations that do not contain all the details of the acoustical source to which they relate [Ellis92].
attached to the brain’s partial impression. But the rules to be applied depend totally upon the kinds of stimuli in question, and in the case of music, the organization of the human auditory mechanism is “tickled” by the fact of several instruments making sounds with aligned onsets (i.e. in time) and common harmonics (i.e. in tune), violating many of main assumptions for distinguishing sources [Ellis92, p.16].

The perception of multiple physical sound sources as a single “auditory object” (the so-called tonal fusion [Scheirer2000, p.30]) seems to play a large role in the perception of music. Bregman has written about this hypothesis, defining the term chimera to describe a tonally fused object:

“Natural hearing tries to avoid chimeric percepts, but music often tries to create them. It may want the listener to accept the simultaneous roll of the drum, clash of the cymbal, and brief pulse of noise from the woodwinds as a single coherent event with its own striking properties. The sound is chimeric in the sense that it does not belong to any single environmental object.” [Bregman90]

In other words, the human auditory system tries naturally to segregate a sound mixture on the physical sources, but orchestration is often called to oppose these tendencies and force it to create a single chimeric sound, which would be irreducible in perceptually smaller units [Bregman90].

As a result of the above discussion, it becomes clear that the note is maybe not a representative symbol for music perception. But in the problem of music transcription, the intention is to produce a symbolic representation that could be used by musicians to reproduce an acoustic signal using musical instrument sounds, and not to understand the mental processes in human music perception.

Klapuri suggests that human perceptions models as such are not the most appropriate for the problem of music transcription [Klapuri98]. In fact, an average listener is not able to transcribe even the mixtures of only a few musical sounds, even if the individual sounds had been clearly introduced before the transcription problem. Music transcription does not seem to be a natural born perceptual capability that humans are able to use unconsciously, being more a specific and absorbing task of separating chimeric sounds. This suffices to agree with Klapuri when he suggests that specific methods are needed to approach the transcription problem, instead of the ones proposed by the general computational auditory scene analysis area.

As a result, this thesis accepts the note as the basic musical element to be analysed in a musical piece and investigates means to efficiently extract the notes’ main features (i.e. pitch, loudness and rhythm) from acoustical recordings of musical pieces.

2.2. PROPERTIES AND FEATURES OF ACOUSTICAL MUSICAL SIGNALS

In this section it is important to define the physical and perceptual properties of musical sounds.

According to Scheirer [Scheirer2000], physical properties of sounds are those that can be measured directly using scientific instruments while the perceptual properties are those that a
human listener associates with the sound. For some perceived attributes, it is possible to easily correlate them with physical properties, being the later the responsible for that particular percept.

It is important to keep in mind the differences between the physical and perceptual properties of musical sounds. It is feasible to define several levels of music interpretation: a music-theoretical level (normally related to the symbolic representation of music – the score), a physical manifestation level (related to the acoustical expression of music) and finally a perceptual level (where the human mind perceives the acoustical signal as a musical instance).

In its attempt to extract a musical score out of the acoustic manifestation of a musical piece, this work mainly worries about the first two levels of music interpretation (although in the reverse level order since this is a transcription problem and not a music execution one), relegating to a second plan the concerns about perceptual issues.

In the following sections the main perceptual properties of a musical sound of interest for our research are presented, as well as the current theories of their physical correlates. In addition, it will be discussed this work’s approach to each of the presented features, taking into account its specificities and objectives.

2.2.1. Pitch

Pitch is the perceptual correlate of the frequency of a single tone, and it is the feature that enables an ordering of sounds on a scale extending from lowest to highest.

When a note on a certain instrument sounds at a certain pitch, say frequency $f$, all this really means is that sound is (roughly) periodic with that frequency. The definition of harmonic sound can use the help of the theory of Fourier, that shows that such a harmonic sound can be decomposed as a sum of sine waves (single tones) with various phases, and whose frequencies are integer multiples of the frequency $f$. This frequency $f$ is known as fundamental frequency of the harmonic sound. Given their contrasting aperiodic characteristic, non-harmonic sounds (such as drum sounds) are more difficult to explain using this model.

The component with frequency $mv$ is called the $m$th harmonic, or the $(m - 1)$st overtone. For example, if $m = 3$, the third harmonic is obtained, or the equivalently, the second overtone is obtained.

Another term frequently used is partial. The $m$th partial of a sound is the $m$th frequency component, counted from the bottom. In a sound where only the odd harmonics are present, the first partial is the fundamental frequency, or first harmonic, and the second partial is the third harmonic.

Since it is not possible to measure the pitch directly as we can do with frequency, it is only possible to model the pitch by creating mappings from measurable properties derived from experimental data [Scheirer2000].

Some authors explore extensively the perceptual relation between the pitch and the fundamental frequencies of musical sounds, exploring the concepts of tonal consonance, tonal
fusion and the influence of loudness and timbre in its perceived pitch [Moore89, Scheirer2000, Benson2001].

Benson also discusses Western music scales and temperaments and, in an effort to understand the rules of their creation and adoption, tries to relate them to the sound’s harmonic structure and perceptual phenomena.

Considering the techniques of transcription of harmonic musical sounds proposed in this thesis, the terms pitch and fundamental frequency will be used as synonyms, assuming that this concept models reasonably well the correlation between the note’s fundamental frequency and its perceived pitch. Furthermore, the tuning of the musical notes will follow the Equal Temperament scale (see Appendix A).

2.2.2. LOUDNESS

Loudness is the perceived attribute of a sound that corresponds to the physical measure of sound intensity. It allows the sound to be positioned on a scale spanning from “soft” to “loud”. The conceptual relation between loudness and intensity is very similar to that between pitch and frequency.

In the context of musical notes, this feature sets the basis of the dynamics of a musical phrase (a longer-term time evolution of the loudness of the succeeding musical notes).

In this work, the determination of the musical loudness of notes will be based on the measurement of its power. As with the pitch, this seems to be an adequate approximation to be used in a music transcription system, since it models reasonably well the correlation between the expressiveness information written in a musical score and the correspondent acoustical perception that results from the playing of the musical piece on a musical instrument.

2.2.3. TEMPO AND RHYTHM

The tempo of a sound can be defined as the perceptual sense that the sound is recurrent in time at regular intervals, where the interval length is between about 250ms. and 2s. [Scheirer2000].

Like pitch and loudness, tempo is a perceptual attribute that cannot be measured directly from a sound, but contrary to them, it is not presently well understood what the physical correlation of tempo is.

Nevertheless, this thesis does not concern the automatic extraction of tempo or beat out of musical pieces. The present work only concerns the determination of rhythmic features (such on-set times, off-set times and durations) of the musical notes. For that purpose, a simple process of finding the times when distinct sounds start and stop playing in an acoustic musical signal will be used.
2.2.4. Timbre

Timbre of a sound can be defined as the quality or set of qualities that allows a listener to identify the physical source of the sound (i.e. the musical instrument) [Scheirer2000]. Until now, it proved difficult to correlate a physical property to timbre.

Timbre can be seen as depending mainly on the relative magnitudes of the harmonics [Moore89], and during the last years some models of timbre have been proposed [Serra89, Martin98a, Martin98b].

A musical transcription system could make use of timbre identification as a supplementary mean of source separation and parameterisation. However, in the current version of our transcription system, there are no concerns about timbre extraction. Yet, the implemented digital signal processing front-end already provides some basic tools for the harmonic characterization of sounds. This feature may be exploited in future versions of the system.
3. **Mid-level Representations of Musical Signals**

As discussed in Section 2.2, music can be interpreted at different levels: from a music-theoretical level (normally related to the symbolic representation of music – the score) to a perceptual level (where the human mind perceives the acoustical signal as a musical instance) and passing through a physical manifestation level (related to the acoustical expression of music).

In this text, the waveform of an acoustical musical signal will be considered to be a "low-level" representation, while the musical notation (notes) will be regarded as a "high-level" representation of musical signals. Because the musical notation symbols cannot be directly derived from the acoustic signal, a mid-level representation must be defined.

The definition of such a mid-level representation must take into consideration the limitations and strengths that it imposes on the evaluation of the “meaning” of a music signal during the process of reducing the number of “objects” in it. The chosen representation should readily answer the questions asked by the higher levels of processing, using the most efficient computational method possible.

In the following paragraphs some of the most important mid-level representations used for musical signals are presented.

Ohm and Helmholtz, who observed that the ear is like a Fourier analyser that divides the sound into spectral components, started the earlier discussions about the signal processing in the ear. Their conclusions encouraged the extensive use of spectral representations of sound in audio applications. These representations have been subject of several studies [Oppenheim & Schafer75] and the development of the Fast Fourier Transforms (FFT) [Cooley & Tukey65] as a computationally efficient way of calculating the DFT was the responsible for the popularity of the Fourier-based analysis and synthesis in many scientific and technical applications [Piszczalski & Galler77, Serra89].

However, there are some differences between the spectral analysis performed by the human auditory system and the standard Fourier analysis methods. The most important difference is that the auditory system obtains a log-scale like spectrum whereas traditional Fourier analysis computes the spectrum with a linear scale. Another difference is that the standard spectrum analysis implementations do not consider other characteristic of the auditory system, such as the masking phenomena in time and frequency domains, neither the differences in the perception of loudness in relation with the frequency. These differences should be taken into consideration so that they can be corrected, or at least taken into account when interpreting the results of the analysis [Serra89].

In any case, this Fourier-based representation is a very efficient solution to the analysis of audio signals, specifically in the case of the transcription of acoustic musical signals. In the particular case of musical signals, perceptual considerations should only be taken into account whenever they bring improvements to the musical results and do not excessively complicate the technique [Serra89].
The transcription system presented in this work uses a slightly modified DFT known as Odd Discrete Fourier Transform (ODFT) [Ferreira98]. Its definition and use will be discussed in the next chapter, where the basic signal analysis framework of the transcription system is presented.

In order to overcome the DFT perceptual shortcomings, some authors proposed the Constant Q Transform as the base for the signal analysis front-end of a music perception system. This transform is explained in [Brown88] and an efficient algorithm for calculating it is given in [Brown&Puckette92].

Basically, the exponential frequency sampling of the constant Q filter bank mimics the human perception of pitch distance – an octave sounds like the “same” pitch distance over a wide range of frequencies.

When compared to the FFT, the CQT is a better approximation to the standard cochlea models. However, the FFT uniform spacing of the frequency axis results a much better calculation efficiency. Furthermore, each of the FFT’s filtered sub-band has the same bandwidth (and this not the case in the CQT) and can thus be sampled at the same rate, leading to a simpler architecture.

Both FFT and CQT fail to fulfil one of the criterions for a mid-level representation: they neither reduce the number of “objects” in the representation nor increase the meaning of each “object”.

By transforming a signal into a time-frequency distribution, and then analysing the resulting spectral peaks and arrangements [Ferreira2001b] it is possible to arrive at a representation that tracks sinusoids through time. This may be called an additive synthesis model of musical sounds [Serra89], and several techniques can be used to extract the sinusoids [Serra89, Ferreira2001b].

At this level, it is possible to develop grouping heuristics that try to regroup the extracted sinusoids and arrangements as sources (i.e. notes played by musical instruments). A method to do so will be presented in the next chapter.

This representation is particularly successful in making a discrete and more compact representation of the signal. The sound resulting from the resynthesis of these sinusoidal tracks possesses a high degree of perceptual fidelity to the original, in spite of being a poor approximation in mean-squared error sense [Serra97].

Meddis and Hewitt proposed another model for compact psychoacoustic phenomena representation [Meddis&Hewitt91], which is related to the correlogram described in [Slaney&Lyon93]. A correlogram is calculated by applying a short-time autocorrelation to the outputs of a constant Q frequency filter bank (typically 20-40 frequency bands), resulting in a three-dimensional volume, where the dimensions are time, frequency and lag time. In practice, a correlogram means searching for periodicities from the outputs of a filter bank.

Ellis presents in his dissertation [Ellis96] a signal-processing algorithm that is a variant of the Meddis and Hewitt model. Ellis computes a “log-lag” correlogram, where the axes of the correlogram volume are frequency channel frequency, lag on a logarithmic scale, and time.
Using this model, common periodicities found at different frequency bands can be associated to obtain a more discrete representation, which Ellis named as *weft*. The *weft* allows simultaneous representation of pitch and spectral shape for multiple harmonic sounds in complex sound scene. This fulfills another one of the criterions for a good *mid-level* representation, since it tries to organize sounds to their independent sources. A straightforward discretization of a *correlogram* is sometimes known as *summary autocorrelation* or *periodogram*.

Apart from being quite complex, *periodograms* and *wefts* are motivated by the fact that they explain a wide range of psychoacoustic phenomena and oddities, and have proved to be practically efficient in CASA studies and applications.

Martin presented a *music transcription system* that uses the *correlogram* as its *mid-level* representation [Martin96]. However, Klapuri performed some simulations using Slaney's implementation of the auditory filter bank and *correlogram* [Slaney&Lyon93], and concluded that the *correlogram* and *wefts* suit CASA well in general, but they are not very appropriate to the *transcription of polyphonic music* [Klapuri98, p.13].

Several others *mid-level* representations have been proposed and used in applications and studies about the perception of audio signals. For further references, Enríquez presents a comprehensive listing of the most important methods used in the monophonic and polyphonic detection of musical notes [Enríquez98].
4. **SELECTED APPROACH AND SYSTEM DESIGN**

A *music transcription system* may be composed of several components that must contribute to the ultimate objective of achieving a meaningful higher-level representation of the original musical signal. The design and implementation of each component should pay attention to the needs of the subsequent processing blocks, and should satisfy them in the most efficient way possible.

Furthermore, the specificities of the signals involved should be taken into consideration at each processing step. For example, it is important to keep in mind the characterization of music signals (addressed in chapter 2), and the different ways to represent them (see chapter 3), when choosing the most appropriate solutions for each processing stage.

In the following sections, and after a brief system overview, the solutions and implementation details used in each one of the processing stages of the *polyphonic music transcription system* presented in this thesis will be discussed and analysed.

4.1. **SYSTEM OVERVIEW**

The following diagram depicts the basic building blocks of the *transcription system* presented in this thesis:

![Diagram](image)

**Figure 4.1- System Overview**

The PCM encoded acoustic music signal is transformed to the frequency domain using an *Overlap-Add* scheme and an ODFT transform.

The spectral representation of the music signal is then processed by the *harmonic analysis* stage, which identifies quasi-stationary sinusoidal components in each frame of the signal and then tries to find an *harmonic structure* linking them.

If detected, these *harmonic structures* are then tracked over time in order to establish a preliminary set of trajectories that, although being closely correlated to the *musical notes*, still contain an excessive number of "false" notes. Nevertheless, using these preliminary trajectories
it is already possible to generate an acceptable MIDI output, mainly in the case of monophonic musical sounds.

Until here, all the processing is being made in a causal scheme, making it feasible to operate in real-time (i.e. on-line processing). The next stages (called post-processing stages) break this scheme, since they run a second iteration of processing over the entire signal. This implementation was chosen for the sake of robustness and quality of results, but further work can be developed in future versions of the system, so that it may be possible to achieve similar results using a causal approach.

In the post-processing stages, a time-domain transient detector marks the non-stationarities in the PCM music signal. This information is used in the trajectory on-set time adjust block where the on-set times of the trajectories coming from the on-line processing stages will be fine-tuned.

After that, the resulting adjusted trajectories will be grouped in time and harmonic clusters. This segregation procedure is used to calculate the probability of a trajectory being a "real" note, which is used to purge the "false" notes and to generate a set of confirmed trajectories (i.e. detected notes).

Finally, these detected notes are represented as MIDI information, which can then be used to generate a musical score or to play a music synthesizer.

A simulation and development environment where the above mentioned processing blocks are implemented was developed using MATLAB®, and the following figure shows its overall appearance.

![Figure 4.2 - Simulation and Development Environment](image-url)
4.2. FRIQUENCY ANALYSIS

A time-frequency mapping operation is the responsible for delivering a convenient spectral representation of the audio signal, necessary to derive important information such as spectral power distribution and tonality behaviour.

The frequency analysis framework used in this transcription system is based around a 50% overlap analysis scheme using a time window \( h(n) \) of length \( N \) and an \( N \)-point ODFT direct transformation.

Although the current version of the system described in this text only concerns the analysis of signals, it may be of interest to future versions to extend it to a resynthesis module, as a way to evaluate and validate the analysis performed in the acoustic signal. In fact, using an appropriately chosen 50% overlap analysis / synthesis window and an ODFT inverse transformation, it is possible to achieve perfect reconstruction (apart from a system delay) in the absence of spectral modification [Ferreira & Vieira 95].

4.2.1. ANALYSIS FRONT-END OVERVIEW

The frequency analysis scheme implemented in this work is depicted in Figure 4.3. The numbers in the circles indicate the order in which data is moved and processed, starting in step 1 and going till step 9. Next, the process is restarted all over again with the processing of the next PCM data block, and only stops when the end of the input PCM file is reached.

The names of the buffers depicted in Figure 4.3 are placed next to their graphic representations. Some of the operations are performed in-place, and are distinguished because they use the same buffers as its input and output. For the sake of graphical clarity, multiple graphic instances of a same buffer are used to represent in-place operations.

![Figure 4.3 – Frequency analysis front-end](image-url)
The size of the transform and the length of the transform window is \( N \) and, consequently, the overlap-add data shift of 50% of the transform window length is \( N/2 \). This is made clear in steps 1, 2 and 8. At each iteration, the \( N \) sized input buffer (InData) receives a new block of \( N/2 \) samples from the input signal and concatenates it with the previous \( N/2 \) input signal samples already stored in the first half of the InData buffer (step 8 of the previous iteration). Since the InData buffer is used by in-place operations, this new block of \( N/2 \) samples has to be saved into a temporary buffer (InSaveData) (step 2) so that at the end of the current iteration it can be moved to the beginning of the input buffer (InData), maintaining the continuity of the analysis process (step 8).

This results in an \( N \)-sized vector whose first half is filled with \( N/2 \) samples gathered from the input signal in the previous iteration and whose second half has the consecutive and just acquired \( N/2 \) input signal samples.

After being multiplied by the analysis window (step 3), the input vector is transformed to the frequency domain using an ODFT. Here it becomes evident how the ODFT is conveniently calculated using an \( N \)-point complex FFT. For that purpose, a pre-processing stage is needed, where the windowed time-domain signal is firstly multiplied by a complex phasor. This sub-optimal ODFT implementation is addressed in section 4.2.2.

The ODFT returns a complex vector with length \( N \). As will be shown in section 4.2.2, only the first \( N/2 \) points of the ODFT output are unique. Consequently, the remaining ones are redundant and are discarded.

The power of the complex ODFT spectrum is calculated (steps 6 and 7). The use of the spectrum power instead of the spectrum magnitude makes no difference to the harmonic analysis routines but has the advantage of simplifying its calculation (a square root calculation is saved).

4.2.2. THE ODFT

The ODFT is a reformulation of a framework successfully used in the area of high-quality audio coding [Ferreira98, Ferreira2001a]: the MDCT filter bank.

This reformulated framework allowed the development of accurate estimation methods for the determination of the frequency, the phase and the magnitude of stationary sounds [Ferreira2001b], turning it into a suitable front-end for the harmonic analysis of music signals (see section 4.3).

The ODFT can be simply seen as the DFT with the discrete frequency bins shifted right by \( \pi/N \), where \( N \) is the size of the transform (and also of the time analysis window), and can be defined as:

\[ x(n, m) = x(n + m N/2), \]

\[ X_d(k, m), \]

in order to make clear that the analysis / synthesis is made on a segment basis and in order to explicit the 50% overlap between successive segments. Since the concern of this sub-section is only with the analysis of an arbitrary segment, the index will be omitted for simplicity.

---

\[ m \] To be accurate, a block (or segment) index \(-\infty \leq m \leq +\infty \) should be included when representing the signal in time, \( x(n, m) = x(n + m N/2) \), or in frequency, \( X_d(k, m) \), in order to make clear that the analysis / synthesis is made on a segment basis and in order to explicit the 50% overlap between successive segments. Since the concern of this sub-section is only with the analysis of an arbitrary segment, the index will be omitted for simplicity.
\[ X_\theta(k) = \sum_{n=0}^{N-1} h(n)x(n)e^{-j\frac{2\pi}{N} \left( k + 1 \right) n} \]  

(4.1)

It should be noted that for \( x(n) \) and \( h(n) \) real, \( X_\theta(k) = X_\theta^*(N-1-k) \), where \( * \) denotes complex conjugation. This implies a subtle advantage of the ODFT relative to the DFT: when the input signal is real, an \( N \)-length DFT exhibits \( N/2+1 \) unique spectral coefficients (although two of them are real) while an \( N \)-length ODFT exhibits only \( N/2 \), which is important in signal modification and synthesis [Ferreira98].

The size of the ODFT (as well as the size of the analysis window, discussed next) is determinant for the spectral resolution necessary to the detection of the lowest expected pitch and for a sufficient spectral selectivity to allow the resolution of two very close tones. An \( N \)-point ODFT transform has a spectral resolution of \( f_s/N \) Hz, where \( f_s \) is the sampling frequency. It is thus clear that an increase of spectral resolution is obtained at the cost of a decreasing time resolution. This calls for a trade-off between time and spectral resolution that should have in mind the specificities of the musical signal under analysis (such as its tempo, shorter notes, range of pitches, etc.).

Experience revealed that transform and window sizes in the order of 1024 or 2048 points leading to frame sizes in the order of 11~22ms (assuming a 50% overlap-add scheme and a sampling frequency of 44100 Hz), represent a good compromise between time and frequency resolution.

The subject of spectral resolution will be further discussed in section 4.3 when the accurate estimation of the frequency of spectral peaks will be addressed.

### 4.2.3. THE ANALYSIS / SYNTHESIS WINDOW

The first processing step in the presented scheme is the windowing of the audio waveform. The choice of the analysis window is important since it determines the trade-off of time versus frequency resolution, which affects the smoothness of the spectrum and the detectability of different sinusoidal components [Serra89].

Following its audio coding heritage [Ferreira98], this system uses an analysis window that derives from the Hanning window and is known as "sine window".

A typical \( N+1 \) point Hanning window is symmetrical with both first and last coefficients equal to zero:

\[ h_{\text{ann}}(n) = \frac{1}{2} \left[ 1 - \cos \frac{2\pi}{N} n \right], \quad 0 \leq n \leq N \]  

(4.2)

A symmetrical \( N \)-point Hanning window with all coefficients different from zero is given by:

\[ h_{\text{ann}}^{\text{mod}}(n) = \frac{1}{2} \left[ 1 - \cos \frac{2\pi}{N} \left( n + \frac{1}{2} \right) \right] = h^2(n), \quad 0 \leq n \leq N - 1 \]  

(4.3)
This result can be seen as the component-to-component product between two identical and symmetrical windows expressed by a simple trigonometric function:

\[
h(n) = \sin\frac{\pi}{N}\left(\frac{N + \frac{1}{2}}{2}\right), \quad 0 \leq n \leq N - 1
\]  (4.4)

The popularity of this window in audio coding is primarily due to the fact that it satisfies the perfect reconstruction requirement, and because of its spectral selectivity. Its analytical tractability also eases the spectral analysis (and modification), which is important for the harmonic analysis of musical signals (see sub-section 4.3).

The frequency response of \(h(n)\) is easily obtained as follows:

\[
|H(\omega)| = \cos\frac{N}{2}\omega \left\{ \sin\frac{1}{2}\left(\frac{\pi}{N} - \omega\right) + \sin\frac{1}{2}\left(\frac{\pi}{N} + \omega\right) \right\}^2
\]  (4.5)

\(H(\omega)\) has zeros at \(\omega = \frac{\pi}{N} + k\frac{2\pi}{N}\), with \(k\) integer, and has two poles at \(\omega = \frac{\pi}{N}\) and \(\omega = -\frac{\pi}{N}\). Clearly, these poles are cancelled out by the two zeros at the same frequencies. The normalized magnitude of \(H(\omega)\), represented here as \(|\widetilde{H}(\omega)|\), can be obtained as:

\[
|\widetilde{H}(\omega)| = \cos\frac{N}{2}\omega \left\{ \sin\frac{1}{2}\left(\frac{\pi}{N} - \omega\right) + \sin\frac{1}{2}\left(\frac{\pi}{N} + \omega\right) \right\} \sin\frac{\pi}{2N}
\]  (4.6)

The normalized frequency response of this analysis window is represented in Figure 4.4:

![Figure 4.4 - Normalized frequency response of the analysis window](image)

\[^7\] Reprinted with permission [Ferreira2001b].
This illustration reveals that $|\hat{H}(\omega)|$ is low-pass, that the bandwidth of the main lobe (i.e. passband) is $6\pi/N$, and that the envelope of the stop-band is monotonously decreasing and exhibits zeros at $\omega = \pm\left(\frac{2\pi}{N} + k\frac{2\pi}{N}\right)$, $k = 1, 2, 3, \ldots$.

4.3. Harmonic Analysis

As discussed in Chapter 3, the analysis procedure discussed in the previous section does not provide, by itself, a flexible sound representation, since it is still very difficult to identify "objects" that easily correlate to musical notes.

The purpose of the harmonic analysis is to accurately extract, from the spectral representation of a musical signal, parametric information that could lead to an easier and more robust way of detecting the presence of musical notes.

4.3.1. Overview

A block diagram that depicts the sequence of events necessary to the harmonic analysis of a musical signal is presented below:

![Figure 4.5 - Harmonic Analysis overview](image)

The harmonic analysis system begins with the detection of the most prominent spectral peaks out of the power spectra of the sound, which was calculated by the frequency analysis front-end (as presented in Section 4.2), followed by the correct parameterisation of those peaks.

For that parameterisation, it is necessary to resolve the peaks as well as possible, and the estimation of their frequencies, phases and magnitudes should be accurate. This assumes a time to frequency transformation with enough spectral resolution to allow the detection of the lowest expected pitch and sufficient spectral selectivity to allow the aforementioned discrimination between two very close tones.

After the determination of spectral peaks, the system must search for harmonic relations between them in an attempt to determine harmonic structures (defined in a following section) and their fundamental frequencies. This analysis is based in a sinusoidal model of musical sounds, as discussed in Section 2.2.

Consequently, higher-order entities, which have a tight correlation to eventual musical notes, are derived from the spectral information of a musical signal. The next step is to study their time evolution in order to form harmonic trajectories (addressed in Section 4.4).

---

8 Only the accurate estimation of the frequency of stationary sinusoids will be addressed, since the phase and magnitude are not critical features to the transcription solution presented here.
4.3.2. Identification of Spectral Peaks

A peak is defined as a local maximum in the power spectrum \( |X_o(k)|^2 \). If \( k_B \) is a bin number in the spectrum, then its value is a local maximum when

\[
|X(k_B - 1)|^2 \leq |X(k_B)|^2 \geq |X(k_B + 1)|^2
\]  

(4.7)

However not all peaks are equally prominent in the spectrum and it is important to have control over its selection. This is done by measuring the height of the peaks in relation to the neighbouring valleys, where the neighbouring valleys are the closest local minima on both sides of the peak.

If the detected valleys for \( X(k_B) \) are \( X(k_B^-) \) and \( X(k_B^+) \), left and right respectively, then a condition for a spectral peak could be:

\[
|X(k_B^-)|^2 < \text{thres} |X(k_B)|^2 > |X(k_B^+)|^2, \quad 0 < \text{thres} \leq 1
\]  

(4.8)

where \( \text{thres} \) represents a fraction of the magnitude\(^9\) of the peak being evaluated.

Serra proposes a method for the determination of the peak height and its subsequent selection, and discusses the effects of the frequency and magnitude in the perceptual prominence of a peak [Serra89]. However, in the system presented here, the effects of the peak’s frequency on its selection are not taken into consideration.

On the other hand, the magnitude is used to select the peaks, whose level in a frame is above a definable “noise-floor” threshold.

Serra defines an additional threshold (that he calls “general-dB-range”) that is used to compare the peak magnitude in relation to the overall sound (against a frame comparison) [Serra89], in an attempt to mimic the human auditory system. Though, since the harmonic analysis used in this system does not use interframe information, only intraframe comparisons are performed.

As a result, a spectral peak in a frame is selected if its weight in relation to both the neighbouring valleys is greater than a defined level and if its absolute power is over a “noise-floor” threshold level. The condition (4.8) can then be redefined as:

\[
\begin{cases} 
|X(k_B^-)|^2 < \text{thres} |X(k_B)|^2 > |X(k_B^+)|^2, & 0 < \text{thres} \leq 1 \\
|X(k_B)|^2 > \text{nf}
\end{cases}
\]  

(4.9)

where \( \text{thres} \) represents a fraction of the power of the peak being evaluated and \( \text{nf} \) represents the “noise floor” threshold.

\(^9\) As mentioned in previous sections, the harmonic analysis described in this text is done using the power of the ODFT spectrum and not its magnitude spectrum. The term magnitude, when related to a spectral peak, is used to describe the height level of that spectral peak, and should not be confused with the ODFT magnitude spectra.
The next figure shows the result of the peak detection algorithm applied to a frame of a musical signal, where the X-axis represents the discrete frequency scale and the Y-axis represents the ODFT spectral power. Each detected peak is surrounded by thin vertical lines.

![Peak detection](image)

Figure 4.6 - Peak detection

4.3.3. Interpolation of Peak Frequency

Due to the nature of the spectra returned by the ODFT, each spectral peak (i.e. a spectral bin that is a local maximum) is accurate only to within half a sample. A bin (sample in the frequency spectrum) represents a frequency interval of \( f_s / N \) Hz, where \( N \) is the ODFT size, and \( f_s \) is the sampling frequency.

One way to increase this resolution is by zero padding the signal in the time domain, increasing the number of ODFT bins per Hz and thus increasing the accuracy of the simple peak detection. However, to obtain a frequency accuracy on the level of 0.1% of the distance from the top of the sinc function\(^{10}\) to its first zero crossing (in the case of a rectangular window), the zero-padding factor required would be 1000 [Serra89]. This demands very large ODFT sizes, resulting in a very inefficient solution.

A more efficient spectral interpolation method that avoids the zero-padding procedure and is capable of superresolution (i.e. the accuracy of the frequency estimate is finer than the discrete frequencies corresponding to the spectral bins of the complex transform) was developed by Ferreira [Ferreira2001b]. For the sake of clarity and convenience of the presentation, the pertinent sections of this paper addressing frequency estimation will be repeated below\(^{11}\).

Given a discrete sinusoid of the form:

\[
x(n) = A \sin \left[ \frac{2\pi}{N} \left( \ell + \Delta \ell \right) n + \phi \right].
\]  

(4.10)

\(^{10}\) A sinc function is defined as \( \sin(x)/x \).

\(^{11}\) With permission from the author.
where $A$ is the magnitude, $\ell$ and $\Delta \ell$ are respectively the integer part and the fractional part of a DFT-type frequency bin scale, and $\phi$ is the initial phase, the intention is to accurately estimate the frequency of the tone, after it has been windowed by a real function $h(n)$ of size $N$, and transformed to a complex frequency domain using an $N$-point ODFT.

Considering the ODFT and the "sine" analysis window used in our system (see section 4.2), the frequency response of each channel of the filter bank can be obtained by modulating $H(\omega)$ (the frequency response of the "sine" window - see Figure 4.4) to the discrete center frequencies $\omega = (k + \frac{1}{2})\frac{2\pi}{N}$, with $k = 0, 1, ..., N - 1$. An illustration of this modulation is presented in Figure 4.7, where a sinusoid, whose frequency is $\omega = 4\pi / N$, is also represented.

![Figure 4.7 - Frequency responses of the first four channels of the ODFT filter bank](image)

It can be seen that as the ODFT channel separation is $2\pi / N$, the zeros of all modulated functions will occur at frequencies that are multiple integer of $2\pi / N$. As a consequence, a sinusoid whose frequency is $\omega = \frac{2\pi}{N}(\ell + \Delta \ell)$, with $\ell$ integer, will be seen by the frequency responses of two ODFT channels whose indexes are $\ell - 1$ and $\ell$.

A sinusoid whose frequency is not just discrete (on the bin frequency scale) but generally given by $\omega = \frac{2\pi}{N}(\ell + \Delta \ell)$ with $1 \leq \ell \leq \frac{N}{2} - 1$ and $0.0 \leq \Delta \ell < 1.0$, will therefore be represented by at least two subbands below the Nyquist frequency. In fact, four possibilities relating the magnitudes of subband channels with indexes $\ell - 1$, $\ell$ and $\ell + 1$ may occur that are of interest for the purpose of accurate frequency estimation. These possibilities are illustrated in Figure 4.8 and concern two particular values for $\Delta \ell$ and two particular ranges for $\Delta \ell$.

---

12 Reprinted with permission [Ferreira2001b].
\[ \Delta \ell = 0.0 \quad 0.0 < \Delta \ell < 0.5 \]

\[ \ell - 1 \ell \ell + 1 \quad k \]

\[ \Delta \ell = 0.5 \quad 0.5 < \Delta \ell < 1.0 \]

\[ \ell - 1 \ell \ell + 1 \quad k \]

Figure 4.8 - Relation between the magnitudes of the ODFT channels \( \ell - 1, \ell \) and \( \ell + 1 \) when the input signal is a sinusoid whose frequency is given by \( \frac{2\pi}{N} (\ell + \Delta \ell) \).\(^{13}\)

It can be concluded from this figure that except for \( \Delta \ell = 0.0 \), the magnitude of subband \( \ell \) will be a local maximum, which means the value of \( \ell \) can be directly and easily extracted from the ODFT spectrum. On the other hand, it can also be concluded that the relative magnitudes of subbands \( \ell - 1 \) and \( \ell + 1 \) can be used to estimate the fractional frequency \( \Delta \ell \).

Assuming stationary conditions\(^{14}\), a sinusoidal signal will be projected in different subbands as a function of two parameters:

1. the exact frequency distance between the frequency of the sinusoid and the center frequency of each ODFT subband: \( \frac{2\pi}{N} (\ell + \Delta \ell) - \frac{2\pi}{N} (k + \frac{1}{2}) \), and

2. the shape of the frequency response of the time analysis window \( |H(\omega)| \).

The signal magnitude at each subband of the ODFT filter bank is therefore expressed as a function of \( |H(\frac{2\pi}{N} (\ell + \Delta \ell - k - \frac{1}{2}))| \). Given that the magnitude of the signal in subband \( k = \ell \) is a local maximum, \( \Delta \ell \) may be determined by computing the ratio of the magnitudes of the signal in subbands \( k = \ell - 1 \) and \( k = \ell + 1 \) i.e., by evaluating:

\[
\frac{|X_\ell(\ell - 1)|}{|X_\ell(\ell + 1)|} = \frac{H(\frac{2\pi}{N} (\Delta \ell + \frac{1}{2}))}{H(\frac{2\pi}{N} (\Delta \ell - \frac{1}{2}))} \quad (4.11)
\]

and extracting the unknown \( \Delta \ell \). In order to circumvent the analytical complexity of \( |H(\omega)| \), and given that (4.11) depends only on the shape of the main lobe of \( |H(\omega)| \), we will instead

\(^{13}\) Reprinted with permission [Ferreira2001b].

\(^{14}\) In the case of musical signals, and for frames with 20-50ms, this stationary condition may be considered as being true.
consider a simple function for a model approximating the main lobe of $|\mathcal{H}(\omega)|$ (depicted in Figure 4.4) and allowing tractable results [Ferreira2001b]:

$$|\mathcal{H}(\omega)| \approx \left[ \cos \frac{N}{6}(\omega) \right]^G, \quad |\omega| < \frac{3\pi}{N} \tag{4.12}$$

where $G$ is a real constant. Using this simplified model, expression (4.11) simplifies to:

$$G \sqrt{\frac{|X_0(\ell-1)|}{|X_0(\ell+1)|}} \cdot \frac{1}{2} = \frac{\sqrt{3}}{2} \cot \frac{\pi \Delta \ell}{3} \tag{4.13}$$

from which it is possible to derive:

$$\Delta \ell = \frac{3}{\pi} \arctan \frac{\sqrt{3}}{1 + 2 \left[ \frac{|X_0(\ell-1)|}{|X_0(\ell+1)|} \right]^G} \tag{4.14}$$

The constant $G$ in this expression has been adjusted to 27.4 / 20.0 in order to minimize the maximum absolute error of the estimation [Ferreira2001b].

The maximum absolute error of this estimation is less than 1% of the bin width and is essentially independent of $N$, of the frequency bin of the ODFT (i.e. of $\ell$), of the magnitude $A$ and of the initial phase $\phi$ [Ferreira2001b]. This result compares favourably to other techniques that do not take into consideration the specificities of the analysis window (e.g. the “quadratic fit” [Brown96] whose associated error has been reported to be as high as 5.7% of the bin width).

### 4.3.4. Detection of Harmonic Structures

In this sub-section, we recall the definition of harmonic sound discussed in section 2.2.1: an harmonic sound is defined as a sound that can be decomposed as a sum of sine waves (single tones), whose frequencies are integer multiples of a fundamental frequency that can be seen as good approximation to the perceived pitch of the sound.

Those single tones that constitute a harmonic sound correspond to the spectral peaks detected by the algorithms presented in the previous sections, and therefore can be arranged in groups around a common fundamental frequency. These groups of peaks (i.e. partials or harmonics) will be here referred to as harmonic structures. Latter in this text, the requirements for the detection of a harmonic structure will be discussed.

Many of the classic transcription systems presented in section 1.2 and chapter 3 track the evolution over time of the peaks detected in each signal frame. The system described here tries to follow not the peaks but the harmonic structures that result from those peaks. It is clear that there are much less harmonic structures than peaks and this results in a lower computational
load and in a lower system delay, both being equally important for real-time operation of the system.

On the other hand, the tracking of the harmonic structures is much more robust than the tracking of single peaks. Even if the detection of a partial fails, the harmonic structure may still be detected.

Additionally, our system does not assume any previous knowledge of the spectral models of the instruments that play in a given musical piece. This allows a more generic and robust detection of the notes since it is solely based in the accurate analysis of the harmonic structures presented by a given sound.

Furthermore, the pitch of a harmonic structure is computed using the frequency of all itspartials, making it increasingly accurate with the increasing number of partials detected within the structure. This tends to create a smooth evolution among frames of a harmonic structure’s pitch and therefore eases the task of the subsequent harmonic structure tracking block (see section 4.4).

The method described here is flexible enough to identify several pitches, suiting the problem of polyphonic music transcription.

Following the extraction of the spectral peaks in a frame, the system starts with the lowest frequency peak and scans all the remaining peaks to see if there is an integer relation between their frequencies (interpolated as described in section 4.3.3). In other words, it tries to discover harmonics of the peak under analysis. This search has a tolerance of one spectral bin for the position of each one of the harmonics, since this is the maximum deviation that leads to an accurate matching of each individual partial of the harmonic structure [Ferreira2001a].

Each time the search algorithm finds a matching partial, the fundamental frequency of the correspondent harmonic structure (represented as pitchd in the below expression) is fine-tuned by minimizing the following error function [Ferreira96]:

\[
error = \sum_{i=1}^{\text{peaks}} (\text{pos}[i] - i \times \text{pitchd})^2
\]

(4.15)

where peaks represents the number of peaks considered (i.e. the number of peaks found until now as being part of the harmonic structure in analysis) and pos[i] represents a vector with the frequency positions (on the bin scale) of the tonal components of the current harmonic structure.

The solution is given by [Ferreira96]:

\[
\text{pitchd} = \frac{\sum_{i=1}^{\text{peaks}} i \times \text{pos}[i]}{\sum_{i=1}^{\text{peaks}} i^2}
\]

(4.16)
This proves the aforementioned advantage of having the calculation of the pitch of a harmonic structure based on the frequencies of all its partials, making it increasingly precise with the increasing number of partials detected within the structure.

This harmonic search algorithm admits missing harmonics in the harmonic structure under analysis, but in the current version, it only allows one recovery from an undetected harmonic situation. This means that, for example, if in the current harmonic structure were already detected the first, the second and fourth harmonics (therefore missing the third harmonic) the system previously used its single possibility to recover from a missing harmonic situation when it detected the fourth harmonic. If the fifth harmonic is not detected, the system will assume that the harmonic structure detection has reached its end, and start the harmonic analysis for the following spectral peak. The number of missing harmonics allowed in each missing harmonic situation is user-definable so that it can suit a wide range of music signals. If this maximum number of missing harmonics is reached, the recovery process is interrupted and the harmonic structure detection stops for the current spectral peak. The result of this procedure can be clearly seen in Figure 4.9, where the algorithm resumed the harmonic analysis after a missing harmonic situation that started when the detection of the 21st harmonic peak failed.

It is also possible to define a maximum number of partials per harmonic structure, which may be used to avoid exhaustive partial searches, and therefore achieve a faster searching procedure.

The system restarts the harmonic detection process with the next lowest frequency peak, whenever the maximum number of partials per harmonic structure is reached in the current iteration, or when the maximum number of missing harmonics or recoveries is attained.

Each detected harmonic structure has yet to comply with another user-definable requirement before being validated: it has to have a minimum number of partials or otherwise it will be discarded. This is useful to discard spurious harmonic sets that have a small number of partials, and therefore may have a low probability of being a good representative of a note.

For each of the detected and validated harmonic structures, the algorithm returns its fundamental frequency (as an interpolated and accurate value), the order of the first missing harmonic and the total number of missing harmonics detected. In the current version of the system, these parameters are only used for graphical display purposes, but the development of a resynthesis module would make extensive use of this data.

The algorithm also returns a power value for each harmonic structure, calculated as the summation of the power values of all its partials. The purpose of this power value is simply to give a notion of the harmonic structure power, and will be used for estimating the power of each trajectory (in the harmonic structure tracking block – see section 4.4) and in the trajectory segmentation and segregation process (in the trajectory clustering and pruning block - see section 4.5.3). The calculation of the MIDI velocity values (as described in Appendix B) will also make use of these values.

Future versions of the system may use the possibility offered by the algorithm to additionally return the power values of each of the partials of the harmonic structures. The accurate estimation of the amplitude of the spectral peaks is also described in [Ferreira2001b] and could
be easily implemented. This could be used to identify the musical instruments that played each one of the notes in the music file as well as used as a supplementary source of information for the resolution of ambiguities in the note detection process.

Although robust and efficient, this algorithm does not live without its own shortcomings: its current implementation does not admit a "missing fundamental" harmonic structure that although not very common, may appear in some musical instruments sounds (e.g. oboe). In fact, this family of musical instruments (i.e. woods, such as clarinets, oboes, bassoons) possesses a peculiar harmonic structure (i.e. timbre) since they are characterized by having missing or strongly attenuated even-partials. This results in several missing harmonic situations (at all the even-harmonics) that the current algorithm does not handle more than once, discarding all the eventually existing partials that would follow the second attempt of recovery.

Other problem is the ambiguous detection of harmonic structures, whose fundamental frequencies are related by an integer number (e.g. $f_0$, $2f_0$, $3f_0$, ...), that results in the detection of more harmonic structures than the number of actually played musical notes. Since the algorithm scans all the detected spectral peaks successively, a peak already selected as a $n^{th}$ harmonic of a previously detected harmonic structure has a high probability of being detected as the first harmonic of a new harmonic structure, whose partials would in fact be the $kn^{th}$ harmonics (with $k=1, 2, 3, ...$) of the former harmonic structure.

Closely related to the above problem, is the difficulty, at this stage, of clearly identifying "real" notes whose pitches are related by octave intervals. In the algorithm presented here, these notes separated by octave intervals are in most of the cases detected as harmonic structures whose fundamental frequencies are related by a multiplicative factor that is a power of two (for more details about these frequency relations see section 4.5.3.2), but it becomes difficult to distinguish them from the harmonic structures that are the result of the ambiguous detection problem, addressed in the previous paragraph.

Although some solutions to minimize these problems could already be implemented at this stage, it was chosen to try to solve them during the subsequent processing blocks, where additional information acquired from the signal (mainly related to the time evolution of the harmonic structures) will provide new insights for its solution. Consequently, no harmonic structure filtering, based on fundamental frequencies values, will be performed at this stage, allowing that all the harmonic structures detected by the algorithm may be analysed by the subsequent processing blocks.

In the following figure is possible to see one of the harmonic structures detected from the peaks in the signal frame depicted in Figure 4.6. Each detected peak is surrounded by thin vertical lines.
4.4. TRACKING OF HARMONIC STRUCTURES

The harmonic analysis processing block addressed in the previous section returns the harmonic structures detected in a given frame, and they are characterized by their fundamental frequency, harmonic structure and power value.

The next step is to study the evolution of those harmonic structures over time and to try to define the correspondent trajectories. These trajectories are entities that will already share many of the properties of a musical note: they will have start and stop times, and consequently a duration, and it will be possible to identify their fundamental frequencies as well as their intensity.

Although these trajectories may still be affected by a high error rate (i.e. many of them are still related to “false” notes), they could already allow a coarse real-time MIDI output that for the particular case of monophonic music. In section 4.5, further post-processing algorithms will be used to filter the trajectories and select the ones that may lead to a more adequate representation of a polyphonic musical piece.

4.4.1. OVERVIEW

It has been observed that by aligning the successive frame results of the harmonic analysis block as a discrete time-frequency representation, it becomes possible to obtain a revealing image of the evolution of the fundamental frequencies and energies of the detected harmonic structures.

Figure 4.10 depicts such a representation, where the Y-axis represents frequency (that for display purposes uses a lower frequency resolution than the one estimated by the frequency interpolation algorithm presented previously) and the X-axis represents time (in frames). Although this representation is somewhat similar to that of a spectrogram, it is important to note that what is represented here are the fundamental frequencies of harmonic structures and
not the frequencies of partials or sinusoidal components. This results in a much cleaner and higher-level time-frequency representation since the sinusoidal components that did not integrate valid harmonic structures (consequently having a very little probability of being representatives of musical notes) were already discarded by the previous harmonic analysis algorithm. This eases the task of tracking harmonic trajectories and reflects the fact that a harmonic structure is a much more robust and musically meaningful entity than a simple partial or sinusoidal tone (as discussed in section 4.3.4).

\[ \text{Figure 4.10 - Detected Harmonic Structures} \]

The fundamental frequencies of the harmonic structures detected at each frame appear vertically aligned in the frequency axis, and the power level of each harmonic structure is indicated by a greyscale colormap (darker tones represent higher power levels).

Performing a longer-term analysis of several frames, it is easy to identify the existence of horizontal “lines” in the above representation. These “lines” result from the detection, in succeeding frames, of harmonic structures with close fundamental frequencies. Consequently they may be the best indicators of musical notes, whose pitches vary little and smoothly from frame to frame.

The length of the “lines” depends on the musical note duration and on the time resolution of the system. A theoretical value for the minimum duration of a musical note, considering a 1/32th note and a tempo of 120bpm, is 31.25 ms. This corresponds to almost three frames of duration when using a time resolution of 11.6 ms (that results from the 50% overlap-add buffer of 512 samples and a sampling frequency of 44100Hz, as described in section 4.2.2) and shows that a played musical note is normally represented by a spectral “line” with a few frames of duration. Additionally, it also evidences that spurious very low duration detections of structures have a high probability of being the result of noise and consequently may be discarded.
4.4.2. Frequency Continuation Algorithm

The main objective of the frequency continuation algorithm is to organize the detected harmonic structures in time-oriented trajectories.

A trajectory is basically defined by a start frame value, a stop frame value (and from these two values a duration value can be inferred), a fundamental frequency vector and a power vector (see Figure 4.11).

<table>
<thead>
<tr>
<th>&gt; TRAJECTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>START FRAME</td>
</tr>
<tr>
<td>STOP FRAME</td>
</tr>
<tr>
<td>[F0(1), F0(2),..., F0(DUR)]</td>
</tr>
<tr>
<td>[P(1), P(2),..., P(DUR+INTERPOL)]</td>
</tr>
<tr>
<td>[INTERP(1),...,INTERP(INTERPOL)]</td>
</tr>
</tbody>
</table>

\[ \text{DUR} = \text{STOPFRAME} - \text{STARTFRAME} \times 1 \]
\[ \text{F0} = \text{FUNDAMENTAL FREQUENCY VECTOR} \]
\[ \text{P} = \text{POWER VECTOR} \]
\[ \text{INTERPOL} = \text{NR. OF INTERPOLATED GAPS} \]

Figure 4.11 - Trajectory structure

These fundamental frequency and a power vectors store the frequency and power values for each one of the frames that constitute a trajectory, allowing to track their evolution throughout the entire duration of the trajectory.

Additionally, an interpolation vector includes the indication about the frames where frequency information gaps were linearly interpolated (explained further below).

Using this structure definition, two trajectory lists are defined in the current system:

- \emph{candidate trajectories list}: trajectories whose start frame is known by the system but whose stop frame and remaining values are still under analysis.

- \emph{validated trajectories list}: trajectories that were completely analysed by the system and whose start and stop frames, as well as all the remaining values, are already determined.

These trajectory groups can be seen as a vector of structures and a graphic representation is depicted in Figure 4.12.

<table>
<thead>
<tr>
<th>TRAJECTORY LIST ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
</tr>
<tr>
<td>START FRAME</td>
</tr>
<tr>
<td>STOP FRAME</td>
</tr>
<tr>
<td>[FREQUENCY]</td>
</tr>
<tr>
<td>[POWER]</td>
</tr>
<tr>
<td>[INTERPOLATIONS]</td>
</tr>
<tr>
<td>(2)</td>
</tr>
<tr>
<td>START FRAME</td>
</tr>
<tr>
<td>STOP FRAME</td>
</tr>
<tr>
<td>[FREQUENCY]</td>
</tr>
<tr>
<td>[POWER]</td>
</tr>
<tr>
<td>[INTERPOLATIONS]</td>
</tr>
</tbody>
</table>

Figure 4.12 - Trajectory list
The frequency continuation algorithm will try to follow the frequency continuities (the aforementioned “lines”) among consecutive frames using a causal scheme. In other words, at each frame, the system will analyse the harmonic structures detected in it and, looking to a historic registry of the structures found in previous frames (i.e. the candidate trajectories list), it will try to follow the frequency lines produced by them (see Figure 4.13).

In order to initiate, follow, end and validate trajectories, the system needs to define the basic requirements for a trajectory, represented by the following parameters:

- **Minimum-note-duration**: this parameter controls the minimum duration (in frames) admitted for a musical note. Once a candidate trajectory is ended, its duration is evaluated and if it is equal or superior to the value set for this parameter, the trajectory is validated and becomes a validated trajectory. All the candidate trajectories with durations inferior to this value are considered too short for being a good representative of a musical note, and consequently are discarded. This parameter is user-definable.

- **Minimum-pause-duration**: a musical pause can be seen as a musical note that sounds like silence. The musical durations of pauses are defined in music theory in the same way notes’ durations are, and therefore the minimum duration of a musical pause and of a musical note are, in theory, exactly the same. However, in practice some differences may exist, and this parameter is used to achieve a more flexible detection of musical pauses. Additionally, this parameter also specifies the maximum number of frames a candidate trajectory may be inactive and, if it is set to 1, a candidate trajectory will be ended as soon as it fails to find a continuation harmonic structure, and consequently no interpolation operation will be performed. This parameter is user-definable.

- **Maximum-frequency-deviation**: candidate trajectories advance through the sound frames selecting harmonic structures. This parameter controls the maximum allowable frequency deviation from a fundamental frequency of a harmonic structure to the frequency of the candidate trajectory that it is selected by. Recall that this system uses the equal temperament scale (see Appendix A) and, due to its logarithmic nature, the deviation interval is not linear or symmetrical. The default value defined for this parameter is \( \frac{1}{2} \) semitone, but can be user-defined.

In order to illustrate the operation of the algorithm, we assume that, when arriving at frame \( n \), the fundamental frequencies of the existing candidate trajectories are \( f_1, f_2, f_3, \ldots, f_p \), where \( p \) is the number of trajectories currently in the candidate list (i.e. \( p = \text{candidate list size} \)), as depicted in Figure 4.13. In fact, each of these \( f_1, f_2, f_3, \ldots, f_p \) values are the last value of the FUNDAMENTAL FREQUENCY vector of each of the corresponding candidate trajectories. This field corresponds to the fundamental frequencies of the harmonic structure followed by each one of the \( p \) candidate trajectories in the previous frame (frame \( n-1 \)). In other words, regarding the frequency continuation algorithm here in discussion, a trajectory takes as its own fundamental frequency the one of the last harmonic structure pursued. This approach allows the detection of glissandos or vibratos, often found in played musical notes. However, their correct treatment falls out of the scope of this thesis.
2. Inactive trajectories (i.e. trajectories that failed to find a continuing harmonic structure in a number of previous frames inferior to the defined minimum-pause-duration value) are re-activated. Their STOPFRAME field is updated with the current frame number and the gaps produced by the inactivity period in the [FREQUENCY] vector are linearly interpolated between the last values stored and the new value added by the detected harmonic structure. The gaps in the [POWER] vector are not interpolated since this will become important to the process of trajectory clustering and pruning that will be described in section 4.5.3.3. Equally important to the trajectory clustering and pruning process will be the information stored in the [INTERPOLATIONS] vector, because the numbers of the frames whose gaps were interpolated are stored in there.

The candidate trajectories that did not find a harmonic structure in the current frame are submitted to the next evaluation process:

1. If the candidate trajectory has been inactive for a period superior to the defined minimum-pause-duration value (defined earlier), it is declared to have reached its end. If its duration is smaller than the defined minimum-note-duration parameter (also defined earlier), the trajectory is discarded and consequently removed from the candidate trajectory list. Otherwise, if the trajectory's duration is superior or equal to the defined minimum-note-duration, it is considered to be a valid trajectory that reached its end and consequently is moved to the validated trajectory list.

2. On the other hand, if the candidate trajectory has been inactive for a period inferior or equal to the defined minimum-pause-duration value, it is kept in the candidate trajectory list in the hope that in the following frames a harmonic structure may give it continuation.

The harmonic structures that were not incorporated by any trajectory are used to start new candidate trajectories, and are added to the candidate trajectory list. The process described above will evaluate their validity in the course of the following frames.

Figure 4.14 illustrates the result of the frequency continuation algorithm when applied to the detected harmonic structures depicted in Figure 4.10. The detected trajectories are presented as black line-connected dots.
4.4.3. The Detection Delay

As is, our frequency continuation algorithm presents a trajectory validation delay that can be defined as: \( \text{delay} = \text{trajectory duration} + \text{minimum pause duration} \). A trajectory is only validated after it has been inactive for a minimum pause duration number of frames and thus this delay is variable and dependent of the trajectory duration and always greater than the sum of the minimum note duration and minimum pause duration values.

This problem poses difficulties for the real-time detection of notes (e.g. in a system that outputs the detected notes as real-time MIDI information), since the detection delay is not constant and can grow too high.

A solution would be to divide the candidate trajectory list in two distinct lists: a sounding trajectory list and a silent trajectory list. All the trajectories would start their lives in the silent trajectory list and would be moved to the sounding trajectory list when their durations exceed the defined minimum-note-duration value. As a result, since the system would already know that these sounding trajectories will become valid trajectories (since they already have the required minimum duration; the system only does not know how much long they will last), all the trajectories in the sounding trajectory list could already send information about their start to the system’s output (e.g. send the correspondent Note-On MIDI message to the system’s MIDI output). This eliminates the dependency on the duration of the trajectories and as a consequence, the detection delay becomes constant and only dependent on the minimum pause duration value, turning its use on a real-time system feasible. The information about the ending of each sounding trajectory would only be sent to the system’s output (e.g. Note-Off MIDI message) when the trajectory becomes validated and is moved to the validated trajectory list.
4.5. POST-PROCESSING

The purpose of the post-processing blocks is to further filter, select and fine-tune the trajectories that resulted from the on-line processing, discussed during the previous sections, in an attempt to identify the best candidates to represent the musical notes played in the musical piece under analysis.

As already explained in section 4.1, where a system overview was presented, this processing stage runs in a non-causal scheme, since it performs a second iteration of processing over the entire signal.

Three processing blocks constitute the post-processing stage (recall Figure 4.1- System Overview):

1. a time-domain Transient Detector that will be the responsible for the detection of non-stationarities in the musical audio signal, which can be important indicators of the onset times of the musical notes,

2. a Trajectory On-set Time Adjust block that will use the Transient Detector output in order to fine-tune the on-set times of the preliminary trajectories \(i.e\). the trajectories that result from the previous on-line processing stages) and to split trajectories erroneously “glued” by the harmonic tracking algorithm (described in section 4.4),

3. and a Trajectory Clustering and Pruning block, where the adjusted trajectories \(i.e\). the trajectories returned by the Trajectory On-set Time Adjust block) are grouped in time clusters and harmonic clusters in order to calculate the probability of a trajectory being a “real” note and consequently purge the “false” notes.

The implementation details of each one of these blocks will be addressed in the following sections.

4.5.1. TRANSIENT DETECTOR

The transient detector block will try to detect non-stationarities in the time-domain representation of musical signals, in an attempt to determine the most probable spots for the onset time of musical notes.

Consider an audio sample sequence \(y(p)\) with \(n\) samples \((p=1, \ldots, n)\). The first operation performed to that audio sequence is a high-pass filtering, with the following transfer function:

\[
B(z) = -0.24z^5 + 0.73z^4 - 1.27z^3 + 1.57z^2 - 1.74z + 1.00
= 1 - 1.74z^{-1} + 1.57z^{-2} - 1.27z^{-3} + 0.73z^{-4} - 0.24z^{-5}.
\]  \hspace{1cm} (4.18)

Assuming that the 5 samples preceding the current sequence are also known, the result of the filtering operation is given by:
\[ f(p) = \sum_{i=0}^{5} y(p-i) b(i), \quad p = 1, \ldots, n \]  

(4.19)

where \( b(i) \) is the impulse response of \( B(z) \).

Making use of the overlap-add scheme already implemented in the system (see section 4.2.1), the musical audio signal is analysed in the time-domain and in segments (i.e. frames) with a size of \( N/2 \) samples (where \( N \) is the size of the ODFT and of the analysis window).

The activity on each audio frame is estimated using the following energy measurement [Ferreira98]:

\[ E(s) = \sum_{k=1}^{N/2} \left| f(k + \frac{N}{2} s) \right|, \quad s = 1, \ldots, ns \]  

(4.20)

where \( ns \) represents the number of segments in the audio signal and that can be determined as \( ns = \left\lfloor \frac{n}{N/2} \right\rfloor \), where the notation \( \lfloor \cdot \rfloor \) denotes the minimum integer.

A segment is considered as non-stationary if the following condition is verified:

\[ \log_{10}\left( \frac{E(s)+1}{E(s-1)+1} \right) > \text{transsthres} \]  

(4.21)

where \( \text{transsthres} \) is a threshold value above which a transient will be detected. In the current version of our system this parameter is user-defined so that it can be adjusted to avoid an over- or under-detection of transients, depending on the specificities of audio material under analysis.

Additionally, a \textit{transient detector release} parameter is also defined and its purpose it to avoid consecutive or too close transient detections that could lead to confusion and bad decisions in the \textit{trajectory on-set time adjust} block (discussed in section 4.5.2). Non-stationary frames separated by a number of frames inferior to the value defined by the \textit{transient detector release} value are represented by the one with the maximum value for the expression (4.21). This can be seen in Figure 4.15, where two outputs of the \textit{transient detector} are depicted. The upper output was generated without using the \textit{transient detector release} parameter, while the lower one uses a the \textit{transient detector release}=2.
Other option that could be used for detection of the on-set times of the musical notes would be the use of the power time evolution, recorded in the power vector included in each trajectory structure, as presented in section 4.4.2 and in Figure 4.11. This information, combined with the transient detector output, could provide additional insights about a note’s on-set time, but this processing is not implemented in the current version of our system.

4.5.2. TRAJECTORY ON-SET TIME ADJUST

In this processing stage, the non-stationarities detected in the musical signal by the transient detector (described previously) can be used to accurately adjust the start frame of the preliminary trajectories and to split trajectories that have a high probability of being two distinct musical notes, but that were previously merged due to the interpolation mechanism of the frequency continuation algorithm (see 4.4.2).

As depicted in Figure 4.16, the trajectories that have a starting frame inside a transient tolerance interval (this interval can be user-defined) have their on-set times adjusted to the frame where the non-stationarity was detected, and this may imply discarding trajectory frames (as is the case of the middle frequency trajectory depicted in Figure 4.16) or adding new frames to the trajectory (these new added frames will have the same frequency and power values as the original starting frame, as illustrated in the lowest frequency trajectory in Figure 4.16).
Furthermore, trajectories whose interpolated frames (i.e. frames that are the result of the interpolation process applied to trajectory gaps, as discussed in section 4.4.2) fall inside transient tolerance intervals of nearby transients have a high probability of being the result of two discrete musical notes that were connected incorrectly into a single trajectory. Consequently these trajectories will be split into two new trajectories at the transient frame.

However, before performing the trajectory split, the system must first analyze if the resulting two trajectories will have a duration that meets the minimum-note-duration requirement (defined in section 4.4.2). If this condition is not verified, the split is not performed (this can be seen in the highest frequency trajectory depicted in Figure 4.16, where a minimum-note-duration value greater than two is assumed).

On the other hand, if a trajectory meets the specified requirements for a split, it will be divided but the consecutive interpolated frames that intersect a transient tolerance interval and that are previous to the frame of the transient in question are discarded. This avoids coarse errors in the determination of the durations of the first split notes, as exemplified in the lowest frequency trajectory in Figure 4.16.

The adjusted trajectories that result from this processing block are now ready for a segregation and selection procedure that will be described in the next section.

4.5.3. CLUSTERING AND PRUNING OF TRAJECTORIES

The parameters that define a trajectory (i.e. the duration, frequency, power and the number of interpolations, as depicted in Figure 4.11 - Trajectory structure) are of utmost importance to the selection of the trajectories with the best probabilities of being good representatives of musical notes.

Additionally, the position of a trajectory relatively to the others is also an important factor, since the analysis of the harmonic and time relations between adjacent and simultaneous trajectories allows a more efficient filtering of the trajectories that may fail to represent a “true” note.
Taking that into consideration, the first step performed by the system at this stage is to organize the adjusted trajectories returned by the trajectory on-set time adjust block into time and harmonic clusters.

4.5.3.1. Time Clusters

The division of the trajectories in time clusters allows to divide a musical signal into sections (i.e. time clusters), where the trajectories that constitute each one of them are evaluated relatively to the section’s specificities and properties (e.g. some sections may be louder while others may be softer, or have a more active rhythmic pattern that generates shorter duration notes than a more calm section). As it will be discussed during the following sections, this allows the system to adapt its evaluation process to the dynamically changing characteristics of a musical signal.

Figure 4.17 shows graphically an example of the result of the grouping in three time clusters of trajectories detected in a given musical signal.

![Figure 4.17 - Definition of Time Clusters](image)

Time clusters can be created by grouping all the trajectories that occur simultaneously with a longer duration trajectory. In other words, time clusters are formed by trajectories whose START FRAMES and STOP FRAMES values are higher and smaller, respectively, than the corresponding values of a longer duration trajectory (that can be seen as a time cluster representative).

The addition of a tolerance interval for the start and end frames of a trajectory allows a more flexible and meaningful construction of time clusters, since it is not always possible to find time clusters whose trajectories comply perfectly with the above mentioned requirements of start and end frames. Figure 4.17 illustrates a case where the second lowest frequency trajectory of time cluster 1 ends two frames after the end of the longest trajectory in the cluster, but is still included in that time cluster (assuming a tolerance value of at least two frames).

This time clusters creation solution may present excessively long time clusters if trajectories with very long durations occur. Although this may not be very common in traditional western music, the occurrence of such situations would compromise the main objective of the grouping
of trajectories in time clusters, since it would be difficult to group inside that same long cluster, trajectories whose durations and power fall in the same order of magnitude. This would imply that the subsequent analysis would return less robust trajectory selections and consequently would increase the error rate of the transcription process.

4.5.3.2. Harmonic Clusters

After the organization of all the trajectories in time clusters, each one of them is further divided into harmonic clusters. Each time cluster can have one or more harmonic clusters. This can be seen in Figure 4.18 where the first time cluster, depicted in Figure 4.17, is divided into four distinct harmonic clusters.

![Diagram of Harmonic Clusters](image)

**Figure 4.18 - Definition of Harmonic Clusters**

In order to understand the reason for the creation of such harmonic clusters it is important to recall from section 4.3.4 that one of the shortcomings of the harmonic structure detector was the ambiguous detection of trajectories with "frequencies" related by an integer number (e.g. \(f_0, 2f_0, 3f_0, \ldots\)). This results in the detection of more trajectories than the number of actually played musical notes.

As a result, the trajectory segregation process described in the following section will attempt to select the best representative trajectories in each harmonic cluster, assuming that the remaining trajectories of the same harmonic cluster are "ghost" trajectories that result from the ambiguity problem mentioned in the previous paragraph.

As a result, a harmonic cluster will be defined as a sub-set of trajectories that belong to the same time cluster and whose "frequencies" are separated by octave intervals. This definition assumes that there is a "frequency" value associated with each trajectory, but the definition of a
trajectory structure given in section 4.4.2 and depicted in Figure 4.11 does not include such a parameter.

However, each trajectory has a frequency vector where the values of the fundamental frequencies of the harmonic structures detected at each frame of the trajectory are stored (see Figure 4.11). This record of the evolution over time of the frequency of a trajectory can be used in several ways, allowing the study and analysis over time of the pitch evolution of an eventual note.

As a result, and considering that at this stage the system is trying to extract notes from the set of adjusted trajectories returned by the previous processing blocks, it is already possible to define a “frequency”, or more adequately termed, a “pitch” value for each of the trajectories here under analysis. In the current implementation of this system, this “pitch” value is calculated as the average value of all the elements of the frequency vector of a trajectory. Other metrics could be used so that vibratos or glissandos could be treated appropriately in order to avoid errors in the frequency determination of the detected notes.

Furthermore, considering that the system uses the equal temperament scale (described in Appendix A) and consequently only using a finite number of discrete frequencies for the pitch of each note (i.e. in order to be in accordance to the equal temperament the pitch of a note must correspond to one of the frequencies defined by this temperament, not being possible to have pitches “between” these discrete values), the “frequency” value of each trajectory here under analysis is previously “quantized” to the nearest equal temperament note.

Additionally, and since MIDI also uses this equal temperament organization of notes, the quantization of the “frequency” of the trajectories allows, at this stage, the easy determination of the correspondent MIDI NOTE NUMBER for each trajectory (see Appendix A and Table 6.1). As it will be explained next, this has the added advantage of facilitating the search of octave related trajectories.

Two notes are separated by one or more octaves if their fundamental frequencies are related by a multiplicative factor that is a power of two. For example, a middle A musical note (i.e. A3) has a fundamental frequency of f0=440 Hz, the A4 note one octave above has a fundamental frequency of 880 Hz = 2×f0, the A5 note two octaves above A3 has a frequency of 1780 Hz = 4×f0 and the three octaves above A6 note has a frequency of 3520 Hz = 8×f0. The following figure shows this relation graphically:

```
<table>
<thead>
<tr>
<th>A3</th>
<th>A4</th>
<th>≅E5</th>
<th>A5</th>
<th>≅C#6</th>
<th>≅E6</th>
<th>≅G6</th>
<th>A6</th>
</tr>
</thead>
<tbody>
<tr>
<td>440Hz</td>
<td>880Hz</td>
<td>1320Hz</td>
<td>1760Hz</td>
<td>2200Hz</td>
<td>2640Hz</td>
<td>3080Hz</td>
<td>3520Hz</td>
</tr>
<tr>
<td>f0</td>
<td>2×f0</td>
<td>3×f0</td>
<td>4×f0</td>
<td>5×f0</td>
<td>6×f0</td>
<td>7×f0</td>
<td>8×f0</td>
</tr>
</tbody>
</table>
```

**Figure 4.19 - Frequencies of octave related musical notes**

An easier way of identifying octave notes is considering not their frequencies but their correspondent MIDI NOTE NUMBERS. The difference between the MIDI NOTE NUMBERS of octave notes is 12×n, where n is an integer number, and results from the definition of an equal
temperament scale (see Appendix A). This way, the system deals with discrete and quantized frequency values, avoiding the complex definition of frequency intervals and deviation ranges in order to determine \textit{octave} relations between the frequencies of the trajectories.

Trajectories with frequencies whose relations are other than \textit{octave} intervals are not considered as making part of a same \textit{harmonic cluster} since they may represent, by themselves, totally unrelated musical notes, whose existences must be analysed on their own. This is clearly depicted in Figure 4.19 where the correspondent musical notes of the $3f0$, $5f0$, $6f0$ and $7f0$ frequencies are not \textit{octaves} of an A note, being very close to distinct musical notes (e.g. E5, C#6, E6 and G6), and consequently may integrate their own \textit{harmonic clusters}. Taking as example Figure 4.18, it can be seen that four \textit{harmonic clusters} were defined by the system. Note how the trajectories, with correspondent musical notes E4 and D6, although having a “frequency” that has an integer relation with the “frequencies” of trajectories A3 and G4, fail to be an \textit{octave note} of them, and therefore are included in different \textit{harmonic clusters}.

\subsection*{4.5.3.3. Criteria for the Trajectory Pruning}

After the grouping of all the trajectories in \textit{time} and \textit{harmonic clusters}, it is now possible to apply segregation criteria inside each cluster in order to select the trajectories that proved to have a sufficient probability of being a good representative of a “real” note.

The \textit{trajectory pruning} algorithm used for the selection and filtering of trajectories inside both the \textit{time} and \textit{harmonic clusters} will be the same, since the evaluation process uses user-definable parameters that can be configured to meet the specificities imposed by each one of them. A detailed characterization of the requirements for the analysis inside each type of cluster will be discussed further down in this text.

The \textit{trajectory pruning} block is based on a \textit{relative} evaluation process of the trajectories. The process starts inside a \textit{harmonic cluster} with the objective of choosing the best representative trajectories for that cluster, and the process is repeated inside each one of the \textit{harmonic clusters} that make part of a \textit{time cluster}.

Subsequently, the selected trajectories that result from each of the \textit{harmonic clusters} (and as it will be described next, each \textit{harmonic cluster} selects at least one trajectory as a good note representative) are going to be compared with each other inside the corresponding \textit{time cluster}, where the weaker \textit{harmonic clusters} may be discarded (by discarding their representative trajectories).

The pruning criteria used in the current implementation of our system is based on the duration and on the value of the power of each trajectory. This power value is the sum of the values of the elements of the power vector that makes part of a trajectory structure divided by the duration of the trajectory (i.e. the number of frames the trajectory lasts). These values can be calculated using the trajectory fields defined in section 4.4.2 and depicted in Figure 4.11.

Other factors could have been used as criteria for the trajectory filtering process, such as:

\begin{itemize}
  \item the number of interpolations that each trajectory suffered, since this could be a good indicator of the “presence strength” of the trajectory in the signal,
\end{itemize}
- the distance of the start frame of each trajectory from the frame where transients were detected, given that experience shows that "ghost" trajectories ambiguously detected (as explained in the previous section) tend to start latter than the trajectories that may best describe a "real" note,

- different metrics and statistical treatments that could be applied to the frequency and power vectors of a trajectory structure, in a way to best describe the validity of a trajectory as a "real" played note representative.

As stated above, the current version of the transcription system does not take into consideration the number of interpolations a trajectory suffered when selecting the best note representatives, but this interpolations number is somewhat implicit in the power value attributed to each trajectory. The fact that the power vector of each trajectory does not have interpolated values (as described in section 4.4.2) means that an increasing number of interpolated frames in a trajectory results in a lower power value, when compared to trajectory with a similar duration and power per frame value.

The duration and power of a trajectory can be seen as a pair of features that may be used to compare a trajectory with all the others that also integrate the same cluster.

In our system, the relative evaluation of the trajectories that belong to a given cluster is implemented using a two-dimensional space representation of the trajectories features (i.e. duration and power), as depicted in Figure 4.20.

![Figure 4.20 - Trajectory Pruning criteria](image)

Each trajectory is represented in the above axis by a point whose coordinates are \((\text{duration}, \text{power})\), which correspond to its duration and power values.
In order to select the best trajectories, the system determines the \textit{Euclidian distance}\textsuperscript{15} of each trajectory point to the axis origin, and selects the point with the higher distance. This "most distant" point corresponds to the trajectory with the best compromise between its power and duration values, consequently considered to be the best candidate to a good representative of a "true" note, among the remaining trajectories plotted in the axis.

As is, the above solution still has a significant shortcoming: it assumes that only one trajectory will be selected in each \textit{time} or \textit{harmonic cluster}. This could be a good strategy for monophonic musical signals, but becomes an inappropriate approach for the transcription of polyphonic musical pieces, since no simultaneously playing notes would be detected.

Given that, it was necessary to define an \textit{inclusion range} around the best choice trajectory so that trajectories whose \textit{(duration, power)} coordinates fall inside the specified range may as well be selected as good representatives of a \textit{note}. This range is represented in Figure 4.20 as a circular section, whose thickness corresponds to the user-definable range of the \textit{inclusion interval}.

As stated earlier, the algorithm used for the selection of trajectories in \textit{time} and \textit{harmonic clusters} is the same, differing only in the adjustment of some parameters that may allow the process to meet more accordingly the requirements of each situation. Next, the specificities that may be necessary to observe when evaluating \textit{harmonic clusters} or \textit{time clusters} will be addressed.

\textbf{4.5.3.4. Trajectory Pruning}

When analysing trajectories inside a \textit{harmonic cluster} the main objective is to filter all the trajectories that are in fact "ghost" \textit{octave} replicas of the "true" note that was played in the musical signal. However, the system must also be able to detect the occurrence of two \textit{octave notes} in a musical piece, since this is a very common situation in traditional western music harmonies.

Experimental observation of the characteristics of the trajectories grouped in the \textit{harmonic clusters} reveals that the "ghost" \textit{octave} trajectories have a lower power value than the "real" \textit{note} trajectory. This is in conformity with the explanation about the ambiguous detection of \textit{harmonic structures}, which ultimately results in the ambiguous detection of trajectories, as discussed in section 4.3.4. As explained there, these "ghost" \textit{octave} trajectories result from a sub-set of the partials of the "true" \textit{note} that the system is trying to extract. Consequently, since these "ghost" trajectories are formed by \textit{harmonic structures}, whose partials are likely to be in a fewer number than those of the "real" note, the overall power value of the trajectory will also be smaller.

If two "real" notes, whose pitches are an integer number of \textit{octaves} apart, were played simultaneously in the musical piece under analysis, the corresponding trajectories of both \textit{notes} would have a power value that is the result of the sum of the power values of complete sets of partials detected for each "true" \textit{note}. This may result in two trajectories with similar power

\textsuperscript{15} \textit{euclidian distance} = $\sqrt{\text{duration}^2 + \text{power}^2}$
values, whose prominence would perhaps be easy to detect and select among the “ghost” trajectories of a harmonic cluster.

In a similar way, the durations of the trajectories inside a harmonic cluster may also give important clues about their relevance. Experiments showed that “ghost” octave trajectories usually start latter and end earlier than the respective “true” note trajectory, consequently having a smaller duration (this can also be justified as an effect of the same above mentioned ambiguous detection of harmonic structures). However, in longer time clusters, this may become ambiguous since several rhythmic elements may be present in the musical signal segment that consequently would result in “true” notes with very distinct durations.

Using the presented solution, at least one trajectory will be selected in each harmonic cluster, but more than one trajectory can be selected if the system detects a high probability of existence of “true” octave notes played simultaneously. The remaining trajectories are discarded.

The following figure depicts an ideal and exemplificative result of the above-described algorithm when applied to the harmonic clusters presented in Figure 4.18, and assuming that the musical segment under analysis contained the “true” A3 and G4 musical notes. It can be seen that the “false” notes E4 and D6 were not yet discarded since they constitute by themselves harmonic clusters that, as stated earlier, must always have at least one representative trajectory. The subsequent pruning process of the trajectories in the time cluster will have the mission of evaluating validity of the trajectories as “real” notes.

![Figure 4.21 - Harmonic Cluster filtering result](image)

After all the harmonic clusters inside a time cluster have been filtered, the resulting trajectories will now be compared with each other inside the time cluster. The objective is now to filter the trajectories that, relatively to the set of trajectories under analysis in this cluster, demonstrate to
have a low probability of being a good representative of a real note. This is done using, once again, the values of the duration and power of each trajectory as depicted in Figure 4.20.

Considering that in most of the cases a time cluster represents a short segment of the musical signal, there is a high probability that all the trajectories in it are sounding more or less simultaneously. Given that, the trajectories with the higher values of power may be the best candidates to “real” notes. This results in trajectories that represent “weak” harmonic clusters being discarded in favour of more prominent ones.

Taking as an example Figure 4.21, ideally the system should reject the E4, and D6 trajectories, since we are considering that that music segment only contains two “true” notes: A3 and G4. However, in practice the system does not know which are the “true” notes that were played in a given musical piece, and consequently should analyse all the possibilities. If a note was really played in a given segment, its power and duration values should, ideally, be among the most prominent ones, and consequently would be selected by the system.

The following figure shows a possible output of the described trajectory filtering process applied to the time cluster depicted in Figure 4.21.

![Figure 4.22 - Time Cluster filtering output](image)

In order to meet the specific requirements imposed by the analysis of trajectories in harmonic clusters and time clusters, and to improve the flexibility in the analysis of different musical audio signals, the current version of our system allows the parameterisation, separately for the harmonic cluster filtering operation and for the time cluster filtering operation, of the weight given to the duration and power values of a trajectory. This is achieved by affecting the calculation of the Euclidian distance of a trajectory point in the following manner:

\[
\text{distance} = \sqrt{\text{duration}^2 \cdot \text{weight} + \text{power}^2 \cdot (1 - \text{weight})}
\]  (4.22)
where $0 \leq \text{weight} \leq 1$ and represents a user-definable parameter for the distribution of the weight given to the duration and power of a trajectory in the selection process. If, for example, the user makes $\text{weight}=1$, only the durations of the trajectories will be considered, resulting in the selection of the longest ones, whatever are their power values.

### 4.5.4. Transcription Output

The trajectory and pruning process described previously gives birth to a filtered list of trajectories whose meaning is expected to be very close to the musical transcription of the musical piece under analysis.

Figure 4.23 presents a possible output of the trajectory clustering and pruning block, assuming as input the trajectories depicted in Figure 4.17.

![Figure 4.23 – Trajectory Pruning output](image)

This filtered trajectory list can now be converted to a MIDI representation. MIDI uses note-on and note-off messages to describe the start and end of a note, and each one of these messages includes information the note to be played (i.e. the note pitch), the time when the note starts and when it ends, as well as information about its loudness (which is represented by the MIDI parameter named velocity). All this information can be derived from the information included in each trajectory's structure (recall Figure 4.11):

- The start and end-time of a MIDI note can be derived from the STARTFRAME and ENDFRAME fields of the corresponding trajectory;
- The MIDI NOTE NUMBER of each MIDI note is calculated using the frequency information recorded in the FUNDAMENTAL FREQUENCY VECTOR of the respective trajectory, as previously described in section 4.5.3.2 and detailed in Appendix A;
- The MIDI velocity information of each MIDI note (i.e. the loudness of each note) can be derived from the POWER VECTOR of the trajectory in question.

Detailed information about the creation a MIDI file is presented in Appendix B.
5. TESTS AND TRANSCRIPTION RESULTS OF THE SYSTEM

In this chapter, the PCM to MIDI Transcription System described in the previous chapters will be submitted to a set of simple tests in an attempt to demonstrate its capabilities and limitations.

Several musical audio files were used in the testing process, with the concern of having monophonic and polyphonic musical signals as well as sounds with multiple or only a single timbre. These music files do not pretend to be musically relevant or interesting, and should be considered as a limited set of examples that try to show, in a simple manner, the system response.

The test audio signals were generated from a MIDI description of the music, which was then played by a wave-table MIDI synthesizer. This way, it was possible to compare the original musical score (a MIDI file) with the transcribed notes (also represented as a MIDI file).

The following sections present and discuss the various tests performed.

5.1. TRANSCRIPTION OF MONOPHONIC MUSICAL SIGNALS

The monophonic audio test signal used is an octave of a chromatic scale\textsuperscript{16}, whose root is the C3 note. The timbre used is that of an acoustic piano, and all notes were played with the same MIDI VELOCITY (see Appendix B for more details about MIDI parameters).

The MIDI description of the original musical score played by the MIDI synthesizer is depicted in Figure 5.1. This type of musical score representation is usually known as piano-roll (very common in most of the commercially available software MIDI sequencers). The Y-axis refers to the musical notes (i.e. musical pitches) and is represented as a piano keyboard for an easier note reference, while its X-axis refers to time and uses musical measures and beats\textsuperscript{17} as units.

After rendering the corresponding audio signal from the score depicted in Figure 5.1 and using a MIDI synthesizer playing an acoustic piano sound, the system analyses the signal and generated a mid-level representation, illustrated in Figure 5.2. In this figure it is possible to see the output result of the transient detector (explained in section 4.5.1) above the spectrogram-like harmonic structure time tracking plot (described in section 4.4). Over-imposed on the harmonic structure plot are the indications of the places where transients were detected (represented as vertical dashed lines) and the trajectories of the detected notes (as black line-connected dots), being possible to observe the correlation between the transients and the on-set times of the detected notes.

Figure 5.3 depicts the MIDI score of the notes detected by the transcription system.

\textsuperscript{16} A chromatic scale is composed of twelve notes, and where each note distances one semitone from the previous and subsequent notes.

\textsuperscript{17} A measure is a unit of time in Western music, and is also known as bar. It represents a regular grouping of beats, as indicated in notation by the time signature (which represents the basic rhythm of a piece of music).
When compared to the original score presented in Figure 5.1, the transcription result only presents a significant difference: the higher C4 note is detected with a longer duration than the one defined in the original score. This may be the result of the piano note decay, since it is also visible in the mid-level representation of the signal, depicted in Figure 5.2.

Figure 5.1 - MIDI score used to generate the monophonic musical signal

Figure 5.2 - Mid-level representation of the monophonic musical signal

Figure 5.3 - MIDI score of the monophonic musical signal transcription
In general, our transcription system presents good results for monophonic musical signals, since it is fairly easy to select the best candidate notes that best describe a “true” note at each time instant.

5.2. TRANSCRIPTION OF POLYPHONIC MUSIC SIGNALS

The task of transcribing polyphonic music is a much more demanding task than the analysis of monophonic signals. In this case, the system must be able to select the best trajectories (i.e. candidate notes) that could be the best “true” notes representative.

As described in the previous chapters, our system does not use spectral models of the instruments that play in a given musical piece, and consequently its capability of detecting notes independently of the instrument playing will be tested with audio files where three distinct instruments are playing simultaneously.

The first polyphonic musical signal uses a flute sound (from the wave-table MIDI synthesiser) that plays the MIDI score depicted in Figure 5.4. This score presents a C major triad (i.e. whose notes are C, E and G) arpeggio followed by a combination of simultaneous notes, ending with the complete C major chord.

Figure 5.5 presents the mid-level representation of the signal and Figure 5.6 shows the transcription score returned by the system.

Comparing the original MIDI score (see Figure 5.4) and the resultant transcription (Figure 5.6) it is possible to see that the system was able to identify all the pitch of all notes played in the audio signal. However, the two G4 notes that appear between beat 3 and 4 in the original score were erroneously detected as only one G4 note, whose duration is nearly the double of the “true” notes duration. This may be the result of the notes being played to close to each other (i.e. legato playing), and looking to the mid-level representation of Figure 5.5, it becomes clear that there were no “gaps” between these two notes, which made the system detect them as only one note. Furthermore, the transient detector, although having detected a transient between these two notes (which is a valuable evidence of a new note just starting), had no opportunity to “disconnect” them, since no interpolated points (i.e. gaps) were detected in the respective harmonic structure trajectory.

Additionally, the system slightly overextends the durations of the last notes detected (the ones that form the C major chord) maybe because of the decay of the flute sound, and presents some minor timing drifts, when compared to the original score.

Further tweaks and fine-tuning of the system detection parameters (defined and discussed in the previous chapters) could perhaps lead to the solution of the above described detection errors, showing that the manual adjustment of such values can be considered a limitation of the system. This should be considered in future versions of the system, since it would be desirable to have the transcription parameters automatically adjusted by the system, in accordance with the music signal specificities.
The next test uses exactly the same MIDI score as the one presented in Figure 5.4 (repeated in Figure 5.7, for convenience), but now uses a different instrument for each one of the voices: a cello for the $C$, a clarinet for the $E$ and a flute for the $G$. The purpose is to analyse the differences in the detection when using multiple timbres in a same musical piece.
Figure 5.7 - MIDI score used to generate a polyphonic multitimbral musical signal

Figure 5.8 – Mid-level representation of the multitimbral polyphonic musical signal of Figure 5.7

Figure 5.9 - MIDI score of the transcription of the multitimbral polyphonic signal of Figure 5.7

Observing the transcription result depicted in Figure 5.9 and the mid-level representation of the multitimbral sound presented in Figure 5.8, it becomes clear that the transcription task encountered much more difficulties. The combination of the multiple timbres generated a more scattered harmonic structure plot and consequently detected much less robust trajectories. This resulted in greater timing drifts in the on-set times and durations of the detected notes as well as more erroneously connected notes (e.g. the C4 note between beat 2 and 4). Additionally, the
use of different timbres may also be the cause for these timing drifts (mainly noticeable in the on-set times of some notes), since each timbre is characterized by distinct attack times.

Anyhow, the system was still able to detect the presences of the pitch of all notes at each time instant. Further adjustments to the harmonic analysis parameters (described in section 4.3) and to the transient detector parameters could perhaps lead to better timings and note detection.

The following test was meant to test the capability of the system to detect octave notes and ambiguous pitches (as discussed in sections 4.3.4, 4.5.3.2 and 4.5.3.3).

Figure 5.10 depicts the original score used in the test. The sound used here is that of a flute (from the wave-table MIDI synthesiser), since it provides a harmonic structure with several partials. Given that the aim of this test is to evaluate the already demanding capability of detecting ambiguous pitches, it was opted to ease the harmonic structure detector block task, by providing sounds with strong partials.

The score creates three ambiguous situations: two octave notes sounding simultaneously (the A2 and the A3 notes) followed by two ambiguous notes (the A2 and the E5 note whose fundamental frequencies are related by a factor of 6) and finally three simultaneously octave and ambiguous notes (A2 and A3, which are octave related, A2 and E5 which have an ambiguous pitch relation, and finally A3 and E5 which also have an ambiguous pitch relation – as shown in Figure 4.19). Subsequently, the A2, A3 and E5 notes are played alone so that it becomes easier to identify them in the mid-level representation illustrated in Figure 5.11.

Figure 5.12 depicts the result of the detection where it is possible to see that the first simultaneously sounding A2 and A3 notes (between beats 1 and 2) were identified correctly, although being separated by an octave interval.

The subsequent ambiguous A2 and E5 notes (between beats 2 and 3) were also identified correctly, but an A3 note was detected erroneously. Figure 5.11 shows that the harmonic structures that correspond to the A3 pitch, between beat 2 and 3, have power values that are lower than the harmonic structures that correspond to the previous A3 note (between beats 1 and 2). These harmonic structures are not the result of the detection of a “true” A3 note being in reality a consequence of the presence of the A2 note. Consequently the trajectory that corresponds to these harmonic structures should have been discarded.

However, and since the original MIDI score has a “true” A3 note between beats 3 and 4, the A3 “ghost” is selected because the system detects it as a trajectory starting in beat 2 and ending at beat 4 (see Figure 5.11). This happens because the frequency continuation algorithm (described in section 4.4.2) connected the “ghost” A3 trajectory, which occurs between beats 2 and 3, with the “real” A3 related trajectory that occurs between beats 3 and 4.

Since there is no gap between them, the transient detector, although signalling a transient occurrence near beat 3, could not split the long trajectory into two distinct ones. As a consequence, the “real” A3 note, which should have occurred between beats 3 and 4, saw its start anticipated to the beat 2 due to the connection with the prior “ghost” A3 trajectory. This longer note presents a duration and power value that allowed it to be selected by the system.
Between beats 3 and 4, the system detects correctly the three pitches that are sounding simultaneously: A2, A3 and E5. However, and as explained in the previous paragraph, the start and duration of the A3 note was wrongly determined.

An additional E4 note was also wrongly detected between beats 3 and 4, and it may be the result of the presence of the “true” A2 note.

The detection of the isolated A2, A3 and E5 that occur subsequently posed no difficulties to the system.

These results show that our system has some capabilities of resolving simultaneously sounding notes whose pitches are octave or ambiguously related. However, the results also put in evidence the fact that the detection process is intimately dependent of the correct determination of the timings of each trajectory, since they largely interfere with the time and harmonic cluster grouping of the trajectories, which are ultimately the main entities where the trajectory selection and pruning is performed.

Figure 5.10 – MIDI score used to generate an ambiguous polyphonic musical signal
Figure 5.11 - Mid-level representation of the ambiguous polyphonic musical signal

Figure 5.12 - MIDI score of the transcription of the ambiguous polyphonic signal of Figure 5.10
As a final example, the musical score presented in Figure 5.13 will be used to test the system capability of transcribing more “real world” musical pieces. The score was played by a musician on a digital piano, and the result recorded as a PCM audio file.

The musical score of the transcription result is presented in Figure 5.14.

![Figure 5.13 - Original music score](image)

![Figure 5.14 - Music score of the transcription of the musical piece presented in Figure 5.13](image)

Without going into a detailed analysis of the transcription results (since this already is a reasonably complex musical piece), it is possible to see that, although there are some note detection errors (false notes detected, missing notes, and wrongly detected pitches), the majority of the note detections are correct.

In what concerns the notes durations and on-set times determination, the system behaved reasonably well considering that some of the timing drifts and differences could also be consequence of the musician interpretation of the original score.

However, the error rate of the detection process of the current version of our system is still very high for applications where the accuracy and robustness of the transcription results are mandatory (see section 1.4). Future enhancements to the system may provide better detection capabilities and more accurate results.
6. CONCLUSION

In this dissertation we have attempted to present a reasonably thorough description of the work that led to the current version of the PCM to MIDI Transposition system, preceded by a synopsis of some of the most important state-of-the-art music transcription systems.

This chapter points out the lessons learned and the major difficulties encountered during the development of the PCM to MIDI Transposition system. The main strengths as well as the most important limitations that still exist in the current version of the system will be recalled and some enhancements that may be implemented in the future will be proposed.

6.1. STRENGTHS, LIMITATIONS AND FUTURE WORK

The development of the PCM to MIDI Transposition system started from a very specific set of requirements, drawn from the research on the extraction of features from audio signals. Being an area where few previously work was published, and even fewer successful system implementations are known to be available commercially or in prototype form, the challenge to implement a working and demonstrable transcription system was considerable.

However, the development of algorithmic solutions for the robust and efficient analysis of audio signals allowed the inception of such a project with an inspired hope that some promising results would be achieved.

As a result, this first version of the transcription system was designed using a bottom-up approach that allowed keeping for now the focus centred on the algorithmic solutions that make the foundations of such a system. Consequently, this allowed developing solutions that do not assume previous knowledge about the signals under analysis, turning them as generic as possible, in what regards musical styles and the type of instruments played in each musical piece.

Future versions of the system may include a top-down approach, which by including higher-level knowledge about the type of signals under analysis, may allow to find solutions to processing problems that otherwise are difficult to solve.

Concerns with the evolution of the current system to a real-time version were already taken during the development of the current non-real-time version. This implied the use of efficient methods for the frequency analysis blocks and for the time analysis of spectral features of a given sound, in a way to anticipate solutions to problems that will be posed in the development of a real-time solution.

The decision of keeping, in the current system version, some of the main processing stages in a non-causal scheme (i.e. the post-processing blocks) is justified by the necessity of testing and evaluating, in a more controlled and supervised way, the algorithmic solutions employed. The experience and knowledge acquired from the study and implementation of these solutions gives new and broader insights to the development of real-time versions of the transcription system.
Although efficient and accurate, the frequency analysis algorithm, which is based on a FFT scheme, still suffers some frequency resolution problems, having difficulty to detect low frequency spectral components when using moderate transform sizes (e.g. 1024–2048 points). New frequency analysis algorithms, such as multirate filter bank designs, may have to be developed and implemented in future versions of the transcription system, in order to have a more uniform and accurate analysis over the relevant frequency range of audio musical signals.

Related areas of application can also be the subject of future work on the system, such as the identification of the instruments playing in a musical piece, or the automatic transcription of the music played by percussive instruments (e.g. drums). The extraction of the tempo and time signature are also related to the process of music transcription, and it would be an interesting and challenging expansion of the system’s capabilities. The development of a resynthesis module within the framework of the analysis / synthesis scheme currently used, in future versions of the system, would allow a more insightful verification of the extracted features and consequently a more focused development of feature analysis and processing solutions.
REFERENCES


[Kashino95] Kunio Kashino, Kazuhiro Nakadai, Tomoyoshi Kinoshita and Hidehiko Tanaka. “Application of bayesian probability network to music scene analysis”. In IJCAI95 Workshop on Computational Auditory Scene Analysis, Montreal, Quebec, August 1995.


APPENDIX A - EQUAL TEMPERAMENT SCALES AND MIDI NOTE NUMBERS

A.1 - EQUAL TEMPERAMENT SCALES

The equal temperament is the most pervasive contemporary tuning system, in which the octave is divided into twelve equal intervals, called semitones. Each semitone is mathematically tuned to the twelfth root of 2, as depicted, for the G#3, A3 and A#3 notes, in Figure 6.1.

Tuning and temperament are similar concepts, but they are not equivalent. Temperament means an adjustment in tuning to get rid of inaccuracy in the intervals between notes. Tuning can relate to one note, but temperament refers to the entire tuning of a scale.

![Semitone Diagram](image)

Figure 6.1 – Frequency relations between musical notes of equal temperament scales

A.2 - MIDI NOTE QUANTIZATION

MIDI defines 128 possible musical notes and, assuming the use of equal temperament and that middle A has a frequency of 440 Hz, their corresponding frequencies are presented in Table 6.1. That table can be generated using a computer routine like the one presented next (using C language code):

```c
int i;
int A=440;
float MIDINOTE#[128];

for i=0 to 127
{
    MIDINOTE#[i]=(A/32)*(2^((i-9)/12));
}
```

In order to generate a MIDI output, the pitch of a detected note has to be quantized to the MIDI NOTE with the closest frequency. In Figure 6.1 is depicted a smaller frequency interval with an exemplificative width of ½ semitone. This is the interval used by our system for the quantization of the pitches of notes into MIDI NOTE NUMBERS (see section 4.5.3.2) into the closest MIDI NOTE NUMBER frequency, as presented in Table 6.1.

Additionally, this same interval is also used in the frequency continuation algorithm, discussed in section 4.4.2.
<table>
<thead>
<tr>
<th>MIDI NOTE#</th>
<th>FREQ (Hz)</th>
<th>MIDI NOTE#</th>
<th>FREQ (Hz)</th>
<th>MIDI NOTE#</th>
<th>FREQ (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>8.175798916</td>
<td>48</td>
<td>130.8127827</td>
<td>96</td>
<td>2093.004522</td>
</tr>
<tr>
<td>C#</td>
<td>8.661957218</td>
<td>49</td>
<td>136.5913155</td>
<td>97</td>
<td>2217.461048</td>
</tr>
<tr>
<td>D</td>
<td>9.177023997</td>
<td>50</td>
<td>146.832384</td>
<td>98</td>
<td>2349.318143</td>
</tr>
<tr>
<td>D#</td>
<td>9.722718241</td>
<td>51</td>
<td>155.5634919</td>
<td>99</td>
<td>2489.015871</td>
</tr>
<tr>
<td>E</td>
<td>10.30066115</td>
<td>52</td>
<td>164.8173785</td>
<td>100</td>
<td>2637.024555</td>
</tr>
<tr>
<td>F</td>
<td>10.91338223</td>
<td>53</td>
<td>174.6141157</td>
<td>101</td>
<td>2793.825851</td>
</tr>
<tr>
<td>F#</td>
<td>11.56232571</td>
<td>54</td>
<td>184.9972114</td>
<td>102</td>
<td>2959.955382</td>
</tr>
<tr>
<td>G</td>
<td>12.24985737</td>
<td>55</td>
<td>195.997718</td>
<td>103</td>
<td>3135.963488</td>
</tr>
<tr>
<td>G#</td>
<td>12.9782718</td>
<td>56</td>
<td>207.6523488</td>
<td>104</td>
<td>3322.437581</td>
</tr>
<tr>
<td>A</td>
<td>13.75</td>
<td>57</td>
<td>220</td>
<td>105</td>
<td>3520</td>
</tr>
<tr>
<td>A#</td>
<td>14.58761755</td>
<td>58</td>
<td>233.0818808</td>
<td>106</td>
<td>3729.310932</td>
</tr>
<tr>
<td>B</td>
<td>15.43383516</td>
<td>59</td>
<td>246.9416506</td>
<td>107</td>
<td>3951.066411</td>
</tr>
<tr>
<td>C</td>
<td>16.35159783</td>
<td>60</td>
<td>261.6255863</td>
<td>108</td>
<td>4186.009045</td>
</tr>
<tr>
<td>C#</td>
<td>17.32391444</td>
<td>61</td>
<td>277.182631</td>
<td>109</td>
<td>4434.922096</td>
</tr>
<tr>
<td>D</td>
<td>18.35404799</td>
<td>62</td>
<td>293.664767</td>
<td>110</td>
<td>4698.636287</td>
</tr>
<tr>
<td>D#</td>
<td>19.44543648</td>
<td>63</td>
<td>311.1269837</td>
<td>111</td>
<td>4978.03174</td>
</tr>
<tr>
<td>E</td>
<td>20.60172231</td>
<td>64</td>
<td>329.6275569</td>
<td>112</td>
<td>5274.049011</td>
</tr>
<tr>
<td>F</td>
<td>21.82676446</td>
<td>65</td>
<td>349.2283214</td>
<td>113</td>
<td>5587.651703</td>
</tr>
<tr>
<td>F#</td>
<td>23.12465142</td>
<td>66</td>
<td>369.9944227</td>
<td>114</td>
<td>5919.910763</td>
</tr>
<tr>
<td>G</td>
<td>24.49971475</td>
<td>67</td>
<td>391.995436</td>
<td>115</td>
<td>6271.929676</td>
</tr>
<tr>
<td>G#</td>
<td>25.9565436</td>
<td>68</td>
<td>415.3046976</td>
<td>116</td>
<td>6644.875161</td>
</tr>
<tr>
<td>A</td>
<td>27.5</td>
<td>69</td>
<td>440</td>
<td>117</td>
<td>7040</td>
</tr>
<tr>
<td>A#</td>
<td>29.13523509</td>
<td>70</td>
<td>466.1637615</td>
<td>118</td>
<td>7458.620184</td>
</tr>
<tr>
<td>B</td>
<td>30.86707633</td>
<td>71</td>
<td>493.8830313</td>
<td>119</td>
<td>7902.13282</td>
</tr>
<tr>
<td>C</td>
<td>32.70319566</td>
<td>72</td>
<td>523.2511306</td>
<td>120</td>
<td>8372.01809</td>
</tr>
<tr>
<td>C#</td>
<td>34.64782887</td>
<td>73</td>
<td>554.365262</td>
<td>121</td>
<td>8869.644191</td>
</tr>
<tr>
<td>D</td>
<td>36.70809599</td>
<td>74</td>
<td>587.3295358</td>
<td>122</td>
<td>9397.272573</td>
</tr>
<tr>
<td>D#</td>
<td>38.89087297</td>
<td>75</td>
<td>622.2539674</td>
<td>123</td>
<td>9956.063479</td>
</tr>
<tr>
<td>E</td>
<td>41.20344461</td>
<td>76</td>
<td>659.2551134</td>
<td>124</td>
<td>10548.08182</td>
</tr>
<tr>
<td>F</td>
<td>43.65352893</td>
<td>77</td>
<td>698.4564629</td>
<td>125</td>
<td>11175.30341</td>
</tr>
<tr>
<td>F#</td>
<td>46.24930284</td>
<td>78</td>
<td>739.9888454</td>
<td>126</td>
<td>11839.82153</td>
</tr>
<tr>
<td>G</td>
<td>48.99942925</td>
<td>79</td>
<td>783.990872</td>
<td>127</td>
<td>12543.85395</td>
</tr>
<tr>
<td>G#</td>
<td>51.9130872</td>
<td>80</td>
<td>830.6093952</td>
<td>128</td>
<td>13254.85959</td>
</tr>
<tr>
<td>A</td>
<td>55</td>
<td>81</td>
<td>880</td>
<td>129</td>
<td>13964.80256</td>
</tr>
<tr>
<td>A#</td>
<td>58.27047019</td>
<td>82</td>
<td>932.327523</td>
<td>130</td>
<td>14674.85904</td>
</tr>
<tr>
<td>B</td>
<td>61.73541266</td>
<td>83</td>
<td>987.7666025</td>
<td>131</td>
<td>15394.80256</td>
</tr>
<tr>
<td>C</td>
<td>65.40639133</td>
<td>84</td>
<td>1046.502261</td>
<td>132</td>
<td>16114.85904</td>
</tr>
<tr>
<td>C#</td>
<td>69.29565774</td>
<td>85</td>
<td>1108.730524</td>
<td></td>
<td>16834.80256</td>
</tr>
<tr>
<td>D</td>
<td>73.41619198</td>
<td>86</td>
<td>1174.659072</td>
<td></td>
<td>17554.80256</td>
</tr>
<tr>
<td>D#</td>
<td>77.78174593</td>
<td>87</td>
<td>1244.507935</td>
<td></td>
<td>18274.80256</td>
</tr>
<tr>
<td>E</td>
<td>82.40688923</td>
<td>88</td>
<td>1318.510228</td>
<td></td>
<td>18994.80256</td>
</tr>
<tr>
<td>F</td>
<td>87.30705786</td>
<td>89</td>
<td>1396.912926</td>
<td></td>
<td>19714.80256</td>
</tr>
<tr>
<td>F#</td>
<td>92.49860568</td>
<td>90</td>
<td>1479.977691</td>
<td></td>
<td>20434.80256</td>
</tr>
<tr>
<td>G</td>
<td>97.998859</td>
<td>91</td>
<td>1567.981744</td>
<td></td>
<td>21154.80256</td>
</tr>
<tr>
<td>G#</td>
<td>103.8261744</td>
<td>92</td>
<td>1661.21879</td>
<td></td>
<td>21874.80256</td>
</tr>
<tr>
<td>A</td>
<td>110</td>
<td>93</td>
<td>1760</td>
<td></td>
<td>22594.80256</td>
</tr>
<tr>
<td>A#</td>
<td>116.5409404</td>
<td>94</td>
<td>1864.655046</td>
<td></td>
<td>23314.80256</td>
</tr>
<tr>
<td>B</td>
<td>123.4708253</td>
<td>95</td>
<td>1975.533205</td>
<td></td>
<td>24034.80256</td>
</tr>
</tbody>
</table>

Table 6.1 - MIDI Note numbers and their corresponding frequencies
APPENDIX B - STANDARD MIDI FILE FORMAT

This appendix presents detailed information about the standard MIDI file format, and it is based on information gathered on the following websites:

- www.midi.org
- www.borg.com/~jglatt/tech/midifile.htm
- crystal.apana.org.au/ghansper/midi_introduction/midi_file_format.html

B.1 - INTRODUCTION

The Standard MIDI File (SMF) is a file format used to store MIDI data (plus some other kinds of data typically needed by a sequencer\(^\text{18}\)).

This format stores the standard MIDI messages (i.e. status bytes with appropriate data bytes) plus a time-stamp for each message (i.e. a series of bytes that represent how many clock pulses to wait before "playing" the event). The format also allows saving information about tempo, time and key signatures, the names of tracks and patterns, and other information typically needed by a sequencer. One SMF can store information for numerous patterns and tracks so that any sequencer can preserve these structures when loading the file.

NOTE: A track usually is analogous to one musical part, such as a Trumpet part. A pattern would be analogous to all of the musical parts (i.e. Trumpet, Drums, Piano, etc) for one song.

The format was designed to be generic so that all sequencers can read the most important data. Think of a MIDI file as a musical version of an ASCII text file (except that the MIDI file contains binary data), and the various sequencer programs as text editors all capable of reading that file. But, unlike ASCII, MIDI file format saves data in chunks (i.e. groups of bytes preceded by an ID and size), which can be parsed, loaded, skipped, etc. Therefore, SMF format is flexible enough for a particular sequencer to store its own proprietary, "extra" data in such a way that another sequencer won't be confused when loading the file and can safely ignore this extra stuff that it doesn't need. For example, maybe a sequencer wants to save a "flag byte" that indicates whether the user has turned on an audible metronome click. The sequencer can save this flag byte in such a way that another sequencer can skip this byte without having to understand what that byte is for. In the future, the SMF format can also be extended to include new "official" chunks that all sequencer programs may elect to load and use. This can be done without making old data files neither obsolete, nor making old sequencers no longer able to load the new files. So, the format is designed to be extensible in a backwardly compatible way.

Other MIDI software than just sequencers may use SMF files. Since SMF files can store any and all types of MIDI messages, including System Exclusive messages, they may be used to store/load data by all kinds of MIDI software, such as a Patch Editor that wants to save some System Exclusive messages it received from a MIDI module (the "timestamp" for each

\(^{18}\) A sequencer is a special type of computer program that allows to record and subsequently to play musical information as sequences of MIDI information.
message may be irrelevant to such a Patch Editor, but it's easily ignored for programs that don't really need it).

In conclusion, any software that saves or loads MIDI data should use SMF format for its data files.

B.2 - DEFINITION OF A CHUNK

Data is always saved within a chunk and several chunks may exist inside a MIDI file.

Each chunk can be a different size (and likely will be), which is related to the number of (8-bit) bytes that are contained in the chunk.

The data bytes in a chunk are typically related in some way. For example, all of the bytes in one chunk may be for one particular sequencer track. The bytes for another sequencer track may be put in a different chunk. So, a chunk is simply a group of related bytes.

Each chunk must begin with a 4 character (i.e. 4 ascii bytes) ID, which tells what "type" of chunk this is. The next 4 bytes must form a 32-bit length (i.e. size) of the chunk, and are followed by length bytes of data, as depicted below:

<table>
<thead>
<tr>
<th>type</th>
<th>length</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes</td>
<td>4 bytes</td>
<td>\textit{length} bytes</td>
</tr>
</tbody>
</table>

Table 6.2 - MIDI File Chunk

All chunks must begin with these two fields (i.e. 8 bytes), which are referred to as the chunk header.

NOTE: The Length does not include the 8 byte chunk header. It simply tells you how many bytes of data are in the chunk following this header.

There are two types of chunks:

- \textit{Header Chunks}: which have a chunk type of \texttt{MThd}
- \textit{Track Chunks}: which have a chunk type of \texttt{MTrk}

Next is presented a schematic representations of a MIDI file:

\[
\begin{array}{|c|c|c|}
\hline
\text{type} & \text{length} & \text{Data} \\
\hline
\text{MThd} & 6 & \text{<format> <tracks> <division>} \\
\hline
\text{MTrk} & \text{<length>} & \text{<delta_time> <event>} ... \\
\hline
\text{MTrk} & \text{<length>} & \text{<delta_time> <event>} ... \\
\hline
\end{array}
\]

Table 6.3 - MIDI File structure
And here's an example chunk header (with bytes expressed in hex) if examined with a hex editor:

\[ 4D \ 54 \ 68 \ 64 \ 00 \ 00 \ 00 \ 06 \]

It can be seen that the first 4 bytes make up the ascii ID of MThd (i.e. the first four bytes are the ascii values for 'M', 'T', 'h', and 'd'). The next 4 bytes tell us that there should be 6 more data bytes in the chunk (and after that we should find the next chunk header or the end of the file).

NOTE: The 4 bytes that make up the Length are stored in (Motorola) "Big Endian" byte order, not (Intel) "Little Endian" reverse byte order. (i.e. the 06 is the fourth byte instead of the first of the four).

In fact, all MIDI files begin with the above MThd header (and that's how a MIDI file is identified).

**B.3 - MThd Chunk**

The MThd header has an ID of MThd, and a length of 6 bytes, as depicted below:

<table>
<thead>
<tr>
<th>Chunk Type</th>
<th>length</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bytes (ascii)</td>
<td>4 bytes (32-bit binary)</td>
<td>16-bit 16-bit 16-bit</td>
</tr>
<tr>
<td>MThd</td>
<td>&lt;length&gt;</td>
<td>&lt;format&gt; &lt;tracks&gt; &lt;division&gt;</td>
</tr>
</tbody>
</table>

Table 6.4 - MThd Header Chunk

The first two data bytes tell the Format. There are actually 3 different types (i.e. formats) of MIDI files. A type of 0 means that the file contains one single track containing MIDI data on possibly all 16 MIDI channels. If your sequencer sorts/stores all of its MIDI data in one single block of memory with the data in the order that it's "played", then it should read/write this type.

A type of 1 means that the file contains one or more simultaneous (i.e. all start from an assumed time of 0) tracks, perhaps each on a single MIDI channel. Together, all of these tracks are considered one sequence or pattern. If your sequencer separates its MIDI data (i.e. tracks) into different blocks of memory but plays them back simultaneously (i.e. as one "pattern"), it will read/write this type. A type of 2 means that the file contains one or more sequentially independent single-track patterns. If your sequencer separates its MIDI data into different blocks of memory, but plays only one block at a time (i.e. each block is considered a different "excerpt" or "song"), then it will read/write this type.

The next 2 bytes tell how many tracks are stored in the file, NumTracks. Of course, for format type 0, this is always 1. For the other 2 types, there can be numerous tracks.

The last two bytes indicate how many Pulses (i.e. clocks) Per Quarter Note (abbreviated as PPQN) resolution the time-stamps are based upon. This parameter is known as Division, as presented below:
Table 6.5 – MIDI File Division

<table>
<thead>
<tr>
<th>Bit:</th>
<th>15</th>
<th>14</th>
<th>...</th>
<th>8</th>
<th>7</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;division&gt;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ticks per quarter note</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>frames/second ticks/frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **bit 15 = 0:**
  - **bits 0-14**
    - number of delta-time units in each a quarter-note.

- **bit 15 = 1:**
  - **bits 0-7**
    - number of delta-time units per SMPTE frame
  - **bits 8-14**
    - form a negative number, representing the number of SMPTE frames per second.
    - 
      - \(-24 = 24\) frames per second
      - \(-25 = 25\) frames per second
      - \(-29 = 30\) frames per second, drop frame
      - \(-30 = 30\) frames per second, non-drop frame

If the first byte of Division is negative, then this represents the division of a second that the time-stamps are based upon. The first byte will be \(-24\), \(-25\), \(-29\), or \(-30\), corresponding to the 4 SMPTE standards representing frames per second. The second byte (a positive number) is the resolution within a frame (i.e. subframe). Typical values may be 4 (MIDI Time Code), 8, 10, 80 (SMPTE bit resolution), or 100.

You can specify millisecond-based timing by the data bytes of -25 and 40 subframes.

**NOTE:** The 2 bytes that make up the Division are stored in (Motorola) "Big Endian" byte order, not (Intel) "Little Endian" reverse byte order. The same is true for the NumTracks and Format.

And here's an example of a complete MThd chunk (with header) if you examined it in a hex editor:

```
4D 54 68 64 MThd ID
00 00 00 06 Length of the MThd chunk is always 6.
00 01 The Format type is 1.
00 02 There are 2 MTrk chunks in this file.
E7 28 Each increment of delta-time represents a millisecond.
```

**B.4 - MTrk Chunk**

After the MThd chunk, you should find an MTrk chunk, as this is the only other currently defined chunk. (If you find some other chunk ID, it must be proprietary to some other program, so skip it by ignoring the following data bytes indicated by the chunk's Length).

An MTrk chunk contains all of the MIDI data (with timing bytes), plus optional non-MIDI data for one track. Obviously, it should be possible to encounter as many MTrk chunks in the file as the MThd chunk's NumTracks field indicated.
As presented below, the MTrk header begins with the ID of MTrk, followed by the Length (i.e. number of data bytes for this track). The Length will likely be different for each track. (After all, a track containing the violin part for a Bach concerto will likely contain more data than a track containing a simple 2 bar drumbeat).

<table>
<thead>
<tr>
<th>Track Chunk</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>length</td>
<td>data</td>
</tr>
<tr>
<td>4 bytes (ascii)</td>
<td>4 bytes (32-bit binary)</td>
<td>&lt;-- length bytes --&gt; (binary data)</td>
</tr>
</tbody>
</table>
| MTrk         | <length> | <delta_time> <event> ...

Table 6.6 - MTrk Header Chunk

B.5 - VARIABLE LENGTH QUANTITIES – EVENT’S TIME

A sequencer track contains a series of events. For example, the first event in the track may be to sound a middle C note. The second event may be to sound the E above middle C. These two events may both happen at the same time. The third event may be to release the middle C note. This event may happen a few musical beats after the first two events (i.e. the middle C note is held down for a few musical beats). Each event has a "time" when it must occur, and the events are arranged within a "chunk" of memory in the order that they occur.

In a MIDI file, an event's "time" precedes the data bytes that make up that event itself. In other words, the bytes that make up the event's time-stamp come first. A given event's time-stamp is referenced from the previous event. For example, if the first event occurs 4 clock after the start of play, then its "delta-time" is 04. If the next event occurs simultaneously with that first event, its time is 00. So, a delta-time is the duration (in clocks) between an event and the preceding event.

NOTE: Since all tracks start with an assumed time of 0, the first event's delta-time is referenced from 0.

A delta-time is stored as a series of bytes which is called a variable length quantity. Only the first 7 bits of each byte is significant (right-justified; sort of like an ASCII byte). So, if we have a 32-bit delta-time, it is necessary to unpack it into a series of 7-bit bytes (i.e. as if we were going to transmit it over MIDI in a SYSEX message). Of course, we will have a variable number of bytes depending upon our delta-time. To indicate which is the last byte of the series, we leave bit #7 clear. In all of the preceding bytes, we set bit #7. So, if a delta-time is between 0-127, it can be represented as one byte. The largest delta-time allowed is 0xFFFFFE, which translates to 4 bytes variable length. Here are examples of delta-times as 32-bit values, and the variable length quantities that they translate to:

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>VARIABLE QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00</td>
</tr>
<tr>
<td>00000040</td>
<td>40</td>
</tr>
<tr>
<td>0000007F</td>
<td>7F</td>
</tr>
<tr>
<td>00000080</td>
<td>81 00</td>
</tr>
<tr>
<td>00002000</td>
<td>C0 00</td>
</tr>
<tr>
<td>00003FFF</td>
<td>FF 7F</td>
</tr>
</tbody>
</table>
Here are some C routines to read and write variable length quantities such as delta-times. With `WriteVarLen()`, we pass a 32-bit value (i.e. unsigned long) and it spits out the correct series of bytes to a file. `ReadVarLen()` reads a series of bytes from a file until it reaches the last byte of a variable length quantity, and returns a 32-bit value.

```c
void WriteVarLen(register unsigned long value)
{
    register unsigned long buffer;
    buffer = value & 0x7F;

    while ( (value >>= 7) )
    {
        buffer <<= 8;
        buffer |= ((value & 0x7F) | 0x80);
    }

    while (TRUE)
    {
        putc(buffer,outfile);
        if (buffer & 0x80)
            buffer >>= 8;
        else
            break;
    }
}

unsigned long ReadVarLen()
{
    register unsigned long value;
    register unsigned char c;

    if ( (value = getc(infile)) & 0x80 )
    {
        value &= 0x7F;
        do
        {
            value = (value << 7) + ((c = getc(infile)) & 0x7F);
        } while (c & 0x80);
    }

    return(value);
}
```

NOTE: The concept of variable length quantities (i.e. breaking up a large value into a series of bytes) is used with other fields in a MIDI file besides delta-times, as it will be described later.

Next is presented a pseudo-code explanation of the above routines. In pseudo-code, `ReadVarLen()` is:

1. Initialise the variable that will hold the value. Set it to 0. We'll call this variable 'result'.
2. Read the next byte of the Variable Length quantity from the MIDI file.

3. Shift all of the bits in 'result' 7 places to the left.  (i.e. Multiply 'result' by 128).

4. Logically OR 'result' with the byte that was read in, but first mask off bit #7 of the byte. (i.e. AND the byte with hexadecimal 7F before you OR with 'result'. But make sure you save the original value of the byte for the test in the next step).

5. Test if bit #7 of the byte is set. (i.e. Is the byte AND hexadecimal 80 equal to hexadecimal 80)? If so, loop back to step #2. Otherwise, you're done, and 'result' now has the appropriate value.

In pseudo code, WriteVarLen() could be:

1. Assume that you have a variable named 'result' which contains the value to write out as a Variable Length Quantity.

2. Declare an array that can contain 4 numbers. We'll call this variable 'array'. Initialise a variable named 'count' to 0.

3. Is 'result' less than 128? If so, skip to step #8.

4. Take the value 'result' AND with hexadecimal 7F, and OR with hexadecimal 80, and store it in 'count' element of 'array'. (i.e. The first time through the loop, this gets stored in the first element of 'array'). NOTE: Don't alter the value of 'result' itself.

5. Increment 'count' by 1.

6. Shift all bits in 'result' 7 places to the right. (This can be done by dividing by 128).

7. Loop back to step #3.

8. Take the value 'result' AND with hexadecimal 7F, and store it in 'count' element of 'array'.


10. Write out the values stored in 'array'. Start with the last element stored above, and finish with the first element stored. (i.e. Write them out in reverse order so that the first element of 'array' gets written to the MIDI file last). NOTE: The variable 'count' tells you how many total bytes to write. It also can be used as an index into the array (if you subtract one from it, and keep writing out bytes until it is -1).

B.6 - EVENTS

An MTrk can contain MIDI events and non-MIDI events (i.e. events that contain data such as tempo settings, track names, etc).

The first (1 to 4) byte(s) in an MTrk will be the first event's delta-time as a variable length quantity. The next data byte is actually the first byte of that event itself. This will be referred to as the event's Status. For MIDI events, this will be the actual MIDI Status byte (or the first MIDI data byte if running status). For example, if the byte is hex 90, then this event is a Note-On upon MIDI channel 0. If for example, the byte is hex 23, we would have to recall the
previous event's status (i.e. MIDI running status). Obviously, the first MIDI event in the MTrk must have a status byte. After the MIDI status byte appear the message data bytes (that can be 1 or 2 bytes, depending upon the status - some MIDI messages only have 1 subsequent data byte). After that we will find the next event's delta time (as a variable quantity), and so on.

### B.6.1 - MIDI EVENTS

A MIDI event is any MIDI Channel message. This includes:

- Channel Voice Messages
- Channel Mode messages

These messages are detailed in the following tables:

#### MIDI Channel Voice Messages

(All MIDI status byte and data byte values are in hexadecimal)

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8n</td>
<td>kk vv</td>
<td>Note off</td>
<td>Normally sent when a key (on a synthesizer) is released (see note 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0–F</td>
<td>MIDI Channel</td>
</tr>
<tr>
<td>kk</td>
<td>00–7F</td>
<td>Key which was released</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This must correspond to a previous note-on message for correct operation</td>
</tr>
<tr>
<td>vv</td>
<td>00–7F</td>
<td>Velocity with which key was released</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Devices which are not velocity sensitive should send vv=40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The interpretation of this message is up to the receiving MIDI device</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9n</td>
<td>kk vv</td>
<td>Note on</td>
<td>Normally sent when a key (on a synthesizers) is pressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A corresponding note-off message must be sent for each and every note-on message</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0–F</td>
<td>MIDI Channel</td>
</tr>
<tr>
<td>kk</td>
<td>00–7F</td>
<td>Key which was pressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each value is a 'half-step' above or below the adjacent values</td>
</tr>
<tr>
<td>vv</td>
<td>00–7F</td>
<td>Velocity with which key was pressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Devices which are not velocity sensitive should send vv=40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>By definition, a note-on message with vv=0 is equivalent to the message: &quot;note-off vv=40&quot; (see note 2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A n</td>
<td>kk vv</td>
<td>Polyphonic Key Pressure</td>
<td>Also known as Aftertouch. This message is sent when there is a change in the pressure being applied to a key (i.e. on a per-key basis).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Byte</td>
<td>Data Bytes</td>
<td>Message</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>n</strong> 0–F</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>kk</strong> 00–7F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>ww</strong> 00–7F</td>
</tr>
<tr>
<td><strong>Be</strong></td>
<td><strong>cc</strong></td>
<td><strong>nm</strong></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><strong>Value</strong></td>
<td><strong>n</strong> 0–F</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td><strong>cc</strong> 00–77</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td><strong>nn</strong> 00–7F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Cc</strong> <strong>pp</strong></td>
<td><strong>Program Change</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used to change the instrument (or sound) to be played when a note-on</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message is received. This is usually not retro-active, and only applies to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>subsequent note-on messages. This message may have a completely different</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interpretation depending on the type of device. For example, it could</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>change the current rhythm on a drum-machine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> 0–F</td>
<td><strong>MIDI Channel</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Value</strong> 00–7F</td>
<td>New Program number. 00=1st program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bn</strong></td>
<td><strong>ww</strong></td>
<td>Channel Key Pressure</td>
<td>Also known as Aftertouch. This message is sent when there is a change in</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td></td>
<td>the overall pressure being applied to the keyboard (ie for the channel</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><strong>Value</strong></td>
<td></td>
<td>overall, and not on a per-key basis).</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><strong>Value</strong></td>
<td><strong>n</strong> 0–F</td>
<td><strong>MIDI Channel</strong></td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td><strong>Value</strong></td>
<td><strong>ww</strong> 00–7F</td>
<td>Channel Pressure Value. 00=min, 7F=max</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>En</strong></td>
<td><strong>lsb</strong></td>
<td><strong>msb</strong></td>
<td><strong>Pitch Bend</strong>. Sent when a change is made in a pitch-bender lever.</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td><strong>n</strong> 0–F</td>
<td><strong>MIDI Channel</strong></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td><strong>lsb</strong> 00–7F</td>
<td>Least significant byte. 00=min, 7F=max</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td><strong>Range</strong></td>
<td><strong>msb</strong> 00–7F</td>
<td>Most Significant Byte. 00=min, 7F=max</td>
</tr>
</tbody>
</table>

Table 6.7 - MIDI Channel voice messages

**Note 1**
Just because a device has received a note-off message does not automatically imply that the note should cease abruptly. Some sounds, such as organ and trumpet sounds will do so. Others, such as piano and guitar sounds, will decay (fade-out) instead, albeit more quickly after the note-off message is received.

**Note 2**
Sending note-on with \( w=0 \) improves the effectiveness of Running Status. Hence this message is preferred over the regular note-off message for devices, which do not detect release velocity.
MIDI Channel Mode Messages

For all MIDI Channel Mode Messages, the message channel 'n' must be the Basic Channel of the MIDI device, or the message will be ignored.

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>00</td>
<td>All Sound Off</td>
<td>Turn off all sound, including envelopes of notes still sounding, and reverb-effects (if applicable).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0 - F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>00</td>
<td>Reset All Controllers</td>
<td>Reset all controllers to their 'default' positions, including all continuous and switch controllers, pitch-bend, and aftertouch effects. Each controller should be returned to a suitable initial condition for that controller. For example, pitch-bend should be returned to it 'centre' position. This message must be ignored if Omni is On (Modes 1 and 2).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0 - F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7A</td>
<td>xx</td>
<td>Local Control</td>
<td>Disconnect (or reconnect) the keyboard and the sound generator in a MIDI synthesizer. The keyboard should continue to send messages via the MIDI-out port, and the sound-generation circuitry should continue to respond to messages received via the MIDI-in port, regardless of this switch.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0 - F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7B</td>
<td>00</td>
<td>All Notes Off</td>
<td>Turn off all notes which for which a note-on MIDI message has been received. (see note 1) This only applies to notes turned on via MIDI, and not to notes turned on via pressing keys on a local keyboard. This message must be ignored if Omni is On (Modes 1 and 2). In Mode 4 (as well as Mode 3), this message must only affect the MIDI Channel on which it is received. If a hold-pedal is 'on' (controller 0x40), then this message should not be acted on until the hold-pedal is released.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0 - F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7C</td>
<td>00</td>
<td>Omni Mode Off</td>
<td>The receiver should respond only to Channel Voice messages, which are received on its Basic Channel. (see note 2) This puts the receiving MIDI device into Channel Mode 3 or 4, depending on the current state of the Mono/Poly switch. (see note 3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>0 - F</td>
</tr>
<tr>
<td>Status Byte</td>
<td>Data Bytes</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>7D</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7E</td>
<td>m</td>
<td>Mono Mode On</td>
<td>Puts the receiver into monophonic mode. (see note 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This puts the receiving MIDI device into Channel Mode 2 or 4, depending on the state of the Omni switch. (see note 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>While Omni is on, the m=1 is used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If n+m=1 &gt; Ch.16 there is no wrap-around to Ch.1. Only channels n...16 are used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status Byte</th>
<th>Data Bytes</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7F</td>
<td>00</td>
<td>Poly Mode On</td>
<td>Puts the receiver into polyphonic mode. (see note 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This puts the receiving MIDI device into Channel Mode 1 or 3, depending on the state of the Omni switch. (see note 3)</td>
</tr>
</tbody>
</table>

**Table 6.8 - MIDI Channel mode messages**

**Note 1**
There is no requirement that a MIDI device support the All-Notes-Off message. A MIDI transmitter must still send individual note-off messages for each note, which has been turned on, regardless of the fact that it has sent an All-Notes-Off message.

**Note 2**
There is an implicit 'All-Notes-Off' function associated with a change of the Channel Mode. That is, when changing Channel Mode, the receiver should turn off all notes turned on via MIDI messages while in the old mode.
This excludes notes turned on via the 'local' keyboard (if applicable).

Although it is not stated explicitly in the specification, I think this:

- includes notes on all channels n...n+m-1 while in Mode 4.
- applies even if the Omni was or is 'On' (unlike the All-Notes-Off message).

**Note 3**
There is no requirement on a MIDI device to support all 4 possible Channel Modes. If a Mode Change message is ignored (due to a particular mode being absent), there is no requirement to perform the All-Notes-Off function.

B.6.2 - SYSEX EVENTS

SYSEX (system exclusive) events (status = F0) are a special case because a SYSEX event can be any length. After the F0 status (which is always stored -- no running status here), you'll find yet another series of variable length bytes. Combine them with ReadVarLen() and you'll come up with a 32-bit value that tells you how many more bytes follow which make up this SYSEX event. This length doesn't include the F0 status.

For example, consider the following SYSEX MIDI message:

```
F0 7F 7F 04 01 7F 7F F7
```

This would be stored in a MIDI file as the following series of bytes (minus the delta-time bytes which would precede it):

```
F0 07 7F 7F 04 01 7F 7F F7
```

The 07, above, is the variable length quantity (which happens to fit in just one byte for this example). It indicates that there are seven subsequent bytes that comprise this SYSEX message.

Really oddball MIDI units send a system exclusive message as a series of small "packets" (with a time delay in-between transmission of each packet). The first packet begins with an F0, but it doesn't end with an F7. The subsequent packets don't start with an F0 nor end with F7. The last packet doesn't start with an F0, but does end with the F7. So, between the first packet's opening F0 and the last packet's closing F7, there's one SYSEX message there. Of course, since a delay is needed in-between each packet, you need to store each packet as a separate event with its own time in the MTrk. Also, you need some way of knowing which events shouldn't begin with an F0 (i.e. all of them except the first packet). So, the MIDI file redefines a MIDI status of F7 (normally used as an end mark for SYSEX packets) as a way to indicate an event that doesn't begin with F0. If such an event follows an F0 event, then it's assumed that the F7 event is the second "packet" of a series. In this context, it's referred to as a SYSEX CONTINUATION event. Just like the F0 type of event, it has a variable length followed by data bytes. On the other hand, the F7 event could be used to store MIDI REALTIME or MIDI COMMON messages. In this case, after the variable length bytes, you should expect to find a MIDI Status byte of F1, F2, F3, F6, F8, FA, FB, FC, or FE (it can be seen that we wouldn't find any such bytes inside of a SYSEX CONTINUATION event). When used in this manner, the F7 event is referred to as an ESCAPED event.

B.6.3 - NON-MIDI EVENTS

A status of FF is reserved to indicate a special non-MIDI event (note that FF is used in MIDI to mean "reset", so it wouldn't be all that useful to store in a data file. Therefore, the MIDI file
arbitrarily redefines the use of this status). After the FF status byte is another byte that tells what Type of non-MIDI event it is. It is similar to a second status byte. Then, subsequent to this byte there is another byte (s -- a variable length quantity again) that tells how many more data bytes follow in this event (i.e. its Length). This Length doesn't include the FF, Type byte, or the Length byte. These special, non-MIDI events are called Meta-Events, and most are optional unless otherwise noted. The section "Meta-Events" presented further down in this text lists the currently defined Meta-Events. It can be seen that unless otherwise mentioned, more than one of these events can be placed in an MTrk (even the same Meta-Event) at any delta-time. (Just like all MIDI events, Meta-Events have a delta-time from the previous event regardless of what type of event that may be. So, it is possible to freely intermix MIDI and Meta events).

B.7 - Meta Events in an MTrk

B.7.1 - Sequence Number

FF 00 02 ss ss

or...

FF 00 00

This optional event must occur at the beginning of an MTrk (i.e. before any non-zero delta-times and before any MIDI events). It specifies the sequence number. The two data bytes ss ss, represent the number which corresponds to the MIDI Cue message. In a format 2 MIDI file, this number identifies each "pattern" (i.e. Mtrk) so that a "song" sequence can use the MIDI Cue message to refer to patterns.

If the ss ss numbers are omitted (i.e. the second form shown above), then the MTrk's location in the file is used. (i.e. The first MTrk chunk is sequence number 0. The second MTrk is sequence number 1. Etc).

In format 0 or 1, which contains only one "pattern" (even though format 1 contains several MTrks), this event is placed in only the first MTrk. So, a group of format 0 or 1 files with different sequence numbers can comprise a "song collection".

There can be only one of these events per MTrk chunk in a Format 2. There can be only one of these events in a Format 0 or 1, and it must be in the first MTrk.

B.7.2 - Text

FF 01 len text

This event represents any amount of text (amount of bytes = len) that can be used for any purpose. It's best to put this event at the beginning of an MTrk. Although this text could be used for any purpose, there are other text-based Meta-Events for such things as orchestration, lyrics, track name, etc. This event is primarily used to add "comments" to a MIDI file that a program would be expected to ignore when loading that file.
Note that len could be a series of bytes since it is expressed as a variable length quantity.

**B.7.3 - COPYRIGHT**

FF 02 len text

This event can be used to include a copyright message. It's best to put this event at the beginning of an MTrk.

Note that len could be a series of bytes since it is expressed as a variable length quantity.

**B.7.4 - SEQUENCE/Track NAME**

FF 03 len text

This event can be used to include the name of the sequence or track. It's best to put this event at the beginning of an MTrk.

Note that len could be a series of bytes since it is expressed as a variable length quantity.

**B.7.5 - INSTRUMENT**

FF 04 len text

This event can be used to identify the name of the instrument (i.e. MIDI module) being used to play the track. This may be different than the Sequence/Track Name. For example, maybe the name of the sequence (i.e. Mtrk) is "Butterfly", but since the track is played upon, for example, a Roland S-770, it is also possible to include an Instrument Name of "Roland S-770".

It's best to put one (or more) of this event at the beginning of an MTrk to provide the user with identification of what instrument(s) is playing the track. Usually, the instruments (i.e. patches, tones, banks, etc) are set up on the audio devices via MIDI Program Change and MIDI Bank Select Controller events within the MTrk. So, this event exists merely to provide the user with visual feedback of what instruments are used for a track.

Note that len could be a series of bytes since it is expressed as a variable length quantity.

**B.7.6 - LYRIC**

FF 05 len text

This event can be used for including a song lyric which occurs on a given beat. A single Lyric MetaEvent should contain only one syllable.

Note that len could be a series of bytes since it is expressed as a variable length quantity.

**B.7.7 - MARKER**

FF 06 len text
This event defines the text for a marker which occurs on a given beat. Marker events might be used to denote a loop start and loop end (i.e. where the sequence loops back to a previous event).

Note that `len` could be a series of bytes since it is expressed as a variable length quantity.

**B.7.8 - CUE POINT**

```
FF 07 len text
```

This event defines the text for a cue point which occurs on a given beat. A Cue Point might be used to denote where a WAVE (i.e. sampled sound) file starts playing, for example, where the text would be the WAVE's filename.

Note that `len` could be a series of bytes since it is expressed as a variable length quantity.

**B.7.9 - DEVICE (PORT) NAME**

```
FF 08 len text
```

This event defines the name of the MIDI device (port) where the track is routed. This replaces the "MIDI Port" Meta-Event that some sequencers formally used to route MIDI tracks to various MIDI ports (in order to support more than 16 MIDI channels).

For example, assume that we have a MIDI interface that has 4 MIDI output ports. They are listed as "MIDI Out 1", "MIDI Out 2", "MIDI Out 3", and "MIDI Out 4". If we wished a particular MTrk to use "MIDI Out 1" then we would put a Port Name Meta-event at the beginning of the MTrk, with "MIDI Out 1" as the text.

All MIDI events that occur in the MTrk, after a given Port Name event, will be routed to that port.

In a format 0 MIDI file, it would be permissible to have numerous Port Name events intermixed with MIDI events, so that the one MTrk could address numerous ports. But that would likely make the MIDI file much larger than it need be. The Port Name event is useful primarily in format 1 MIDI files, where each MTrk gets routed to one particular port.

Note that `len` could be a series of bytes since it is expressed as a variable length quantity.

**B.7.10 - PROGRAM NAME**

```
FF 09 len text
```

This event defines the name of the program (i.e. patch) used to play the MTrk. This may be different than the Sequence/Track Name. For example, it maybe the name of a sequence (i.e. Mtrk), but since the track is played upon an electric piano patch, it may also be a Program Name, like "ELECTRIC PIANO".
Usually, the instruments (i.e. patches, tones, banks, etc) are set up on the audio devices via MIDI Program Change and MIDI Bank Select Controller events within the MTrk. So, this event exists merely to provide the user with visual feedback of what particular patch is used for a track. But it can also give a hint to intelligent software if patch remapping needs to be done. For example, if the MIDI file was created on a non-General MIDI instrument, then the MIDI Program Change event will likely contain the wrong value when played on a General MIDI instrument. Intelligent software can use the Program Name event to look up the correct value for the MIDI Program Change event.

Note that one could be a series of bytes since it is expressed as a variable length quantity.

B.7.11 - END OF TRACK

FF 2F 00

This event is not optional. It must be the last event in every MTrk. It’s used as a definitive marking of the end of an MTrk. Only 1 per MTrk.

B.7.12 - TEMPO

FF 51 03 tt tt tt

This event indicates a tempo change. The 3 data bytes of tt tt tt are the tempo in microseconds per quarter note. In other words, the microsecond tempo value tells you how long each one of your sequencer’s "quarter notes" should be. For example, if you have the 3 bytes of 07 A1 20, then each quarter note should be 0x07A120 (or 500,000) microseconds long.

Thus, the MIDI file format expresses tempo as "the amount of time (i.e. microseconds) per quarter note”.

NOTE: If there are no tempo events in a MIDI file, then the tempo is assumed to be 120 BPM.

In a format 0 file, the tempo changes are scattered throughout the one MTrk. In format 1, the very first MTrk should consist of only the tempo (and time signature) events so that it could be read by some device capable of generating a "tempo map". It is best not to place MIDI events in this MTrk. In format 2, each MTrk should begin with at least one initial tempo (and time signature) event.

B.7.13 - SMPTE OFFSET

FF 54 05 hr mn se fr ff

This event designates the SMPTE start time (hours, minutes, seconds, frames, subframes) of the MTrk. It should be at the start of the MTrk. The hour should not be encoded with the SMPTE format as it is in MIDI Time Code. In a format 1 file, the SMPTE OFFSET must be stored with the tempo map (i.e. the first MTrk), and has no meaning in any other MTrk. The ff field contains fractional frames in 100ths of a frame, even in SMPTE based MTrks which
specify a different frame subdivision for delta-times (i.e. different from the subframe setting in the MTmd).

B.7.14 - Time Signature

FF 58 04 nn dd cc bb

Time signature is expressed as 4 numbers. nn and dd represent the "numerator" and "denominator" of the signature as notated on sheet music. The denominator is a negative power of 2: 2 = quarter note, 3 = eighth, etc.

The cc expresses the number of MIDI clocks in a metronome click.

The bb parameter expresses the number of notated 32nd notes in a MIDI quarter note (24 MIDI clocks). This event allows a program to relate what MIDI thinks of as a quarter, to something entirely different.

For example, 6/8 time with a metronome click every 3 eighth notes and 24 clocks per quarter note would be the following event:

FF 58 04 06 03 18 08

NOTE: If there are no time signature events in a MIDI file, then the time signature is assumed to be 4/4.

In a format 0 file, the time signatures changes are scattered throughout the one MTrk. In format 1, the very first MTrk should consist of only the time signature (and tempo) events so that it could be read by some device capable of generating a "tempo map". It is best not to place MIDI events in this MTrk. In format 2, each MTrk should begin with at least one initial time signature (and tempo) event.

B.7.15 - Key Signature

FF 59 02 sf mi

sf = -7 for 7 flats, -1 for 1 flat, etc, 0 for key of c, 1 for 1 sharp, etc.

mi = 0 for major, 1 for minor

B.7.16 - Proprietary Event

FF 7F len data

This can be used by a program to store proprietary data. The first byte(s) should be a unique ID of some sort so that a program can identity whether the event belongs to it, or to some other program. A 4 character (i.e. ascii) ID is recommended for such.

Note that len could be a series of bytes since it is expressed as a variable length quantity.
B.8 - TEMPO AND TIME BASE

The MIDI file format's Tempo Meta-Event expresses tempo as "the amount of time (i.e. microseconds) per quarter note". For example, if a Tempo Meta-Event contains the 3 bytes of 07 A1 20, then each quarter note should be 0x07A120 (or 500,000) microseconds long.

B.8.1 - BPM

Normally, musicians express tempo as "the amount of quarter notes in every minute (i.e. time period)". This is the opposite of the way that the MIDI file format expresses it.

When musicians refer to a "beat" in terms of tempo, they are referring to a quarter note (i.e. a quarter note is always one beat when talking about tempo, regardless of the time signature. It can become a bit confusing to non-musicians that the time signature's "beat" may not be the same thing as the tempo's "beat" - it won't be unless the time signature's beat also happens to be a quarter note. But that's the traditional definition of BPM tempo). To a musician, tempo is therefore always "how many quarter notes happen during every minute". Musicians refer to this measurement as BPM (i.e. Beats Per Minute). So a tempo of 100 BPM means that a musician must be able to play 100 steady quarter notes, one right after the other, in one minute. That's how "fast" the "musical tempo" is at 100 BPM. It's very important that the concept of how a musician expresses "musical tempo" (i.e. BPM) is understood, in order to properly present tempo settings to a musician, and yet be able to relate it to how the MIDI file format expresses tempo.

To convert the Tempo Meta-Event's tempo (i.e. the 3 bytes that specify the amount of microseconds per quarter note) to BPM:

\[ \text{BPM} = \frac{60,000,000}{(tt \ tt \ tt)} \]

For example, a tempo of 120 BPM = 07 A1 20 microseconds per quarter note.

The MIDI file format uses "time per quarter note" instead of "quarter notes per time" to specify its tempo because it's easier to specify more precise tempos with the former. With BPM, sometimes you have to deal with fractional tempos (for example, 100.3 BPM) if you want to allow a finer resolution to the tempo. Using microseconds to express tempo offers plenty of resolution.

Also, SMPTE is a time-based protocol (i.e. it's based upon seconds, minutes, and hours, rather than a musical tempo). Therefore it's easier to relate the MIDI file's tempo to SMPTE timing if you express it as microseconds. Many musical devices now use SMPTE to sync their playback.

B.8.2 - PPQN CLOCK

A sequencer typically uses some internal hardware timer counting off steady time (i.e. microseconds perhaps) to generate a software "PPQN clock" that counts off the timebase (Division) "ticks". In this way, the time upon which an event occurs can be expressed to the musician in terms of a musical bar:beat:PPQN-tick rather than how many microseconds from
the start of the playback. Remember that musicians always think in terms of a beat, not the passage of seconds, minutes, etc.

As mentioned, the microsecond tempo value tells you how long each one of your sequencer's "quarter notes" should be. From here, you can figure out how long each one of your sequencer's PPQN clocks should be by dividing that microsecond value by your MIDI file's Division. For example, if your MIDI file's Division is 96 PPQN, then that means that each of your sequencer's PPQN clock ticks at the above tempo should be 500,000 / 96 (or 5,208.3) microseconds long (i.e. there should be 5,208.3 microseconds in-between each PPQN clock tick in order to yield a tempo of 120 BPM at 96 PPQN. And there should always be 96 of these clock ticks in each quarter note, 48 ticks in each eighth note, 24 ticks in each sixteenth, etc).

It can be seen that it is possible to have any timebase at any tempo. For example, it is possible to have a 96 PPQN file playing at 100 BPM just as having a 192 PPQN file playing at 100 BPM. It is also possible to have a 96 PPQN file playing at either 100 BPM or 120 BPM. Timebase and tempo are two entirely separate quantities. Of course, they both are needed when the hardware timer is set up (i.e. when defining how many microseconds are in each PPQN tick). And of course, at slower tempos, the PPQN clock tick is going to be longer than at faster tempos.

B.8.3 - MIDI Clock

MIDI clock bytes are sent over MIDI, in order to sync the playback of 2 devices (i.e. one device is generating MIDI clocks at its current tempo which it internally counts off, and the other device is syncing its playback to the receipt of these bytes). Unlike with SMPTE frames, MIDI clock bytes are sent at a rate related to the musical tempo.

Since there are 24 MIDI Clocks in every quarter note, the length of a MIDI Clock (i.e. time in-between each MIDI Clock message) is the microsecond tempo divided by 24. In the above example, that would be 500,000/24, or 20,833.3 microseconds in every MIDI Clock. Alternately, it is possible to relate this to the timebase in use (i.e. PPQN clock). If we have 96 PPQN, then that means that a MIDI Clock byte must occur every 96 / 24 (i.e. 4) PPQN clocks.

B.8.4 - SMPTE

SMPTE counts off the passage of time in terms of seconds, minutes, and hours (i.e. the way that non-musicians count time). It also breaks down the seconds into smaller units called "frames". The movie industry created SMPTE, and they adopted 4 different frame rates. It is possible to divide a second into 24, 25, 29, or 30 frames. Later on, musical devices needed even finer resolution, and so each frame was broken down into "subframes".

Consequently, SMPTE is not directly related to musical tempo. SMPTE time doesn't vary with "musical tempo".

Many devices use SMPTE to sync their playback. If it is needed to synchronize with such a device, then it may be necessary to deal with SMPTE timing. Probably, it will still be necessary to maintain some sort of PPQN clock, based upon the passing SMPTE subframes, so that the user can adjust the tempo of the playback in terms of BPM, and consider the time of each event.
in terms of bar:beat:tick. But since SMPTE doesn't directly relate to musical tempo, it is necessary to interpolate (i.e. calculate) the PPQN clocks from the passing of subframes/frames/seconds/minutes/hours (just as we previously calculated the PPQN clock from a hardware timer counting off microseconds).

Take the easy example of 25 Frames and 40 SubFrames. As previously mentioned in the discussion of Division, this is analogous to millisecond based timing because you have 1,000 SMPTE subframes per second. (We have 25 frames per second. Each second is divided up into 40 subframes, and you therefore have 25 * 40 subframes per second. And remember that 1,000 milliseconds are also in every second). Every millisecond therefore means that another subframe has passed (and vice versa). Every time we count off 40 subframes, a SMPTE frame has passed (and vice versa).

Assume that we desire 96 PPQN and a tempo of 500,000 microseconds. Considering that with 25-40 Frame-SubFrame SMPTE timing 1 millisecond = 1 subframe (and remember that 1 millisecond = 1,000 microseconds), there should be 500,000 / 1,000 (i.e. 500) subframes per quarter note. Since we have 96 PPQN in every quarter note, then every PPQN ends up being 500 / 96 subframes long, or 5,208.3 milliseconds (i.e. there's how we end up with that 5,208.3 microseconds PPQN clock tick just as we did above in discussing PPQN clock). And since 1 millisecond = 1 subframe, every PPQN clock tick also equals 5,208.3 subframes at the above tempo and timebase.

B.8.5 - FORMULAS

\[
\begin{align*}
\text{BPM} &= 60,000,000 / \text{MicroTempo} \\
\text{MicrosPerPPQN} &= \text{MicroTempo} / \text{TimeBase} \\
\text{MicrosPerMIDIClock} &= \text{MicroTempo} / 24 \\
\text{PPQNPerMIDIClock} &= \text{TimeBase} / 24 \\
\text{MicrosPerSubFrame} &= 1000000 \times \text{Frames} \times \text{SubFrames} \\
\text{SubFramesPerQuarterNote} &= \text{MicroTempo} / (\text{Frames} \times \text{SubFrames}) \\
\text{SubFramesPerPPQN} &= \text{SubFramesPerQuarterNote} / \text{TimeBase} \\
\text{MicrosPerPPQN} &= \text{SubFramesPerPPQN} \times \text{Frames} \times \text{SubFrames}
\end{align*}
\]

B.9 - OBSOLETE META-EVENTS

The following Meta-Events are considered obsolete and should not be used. Use the Device (Port) Name Meta-Event instead of the MIDI Port Meta-Event.

B.9.1 - MIDI CHANNEL

\[
\text{FF 20 01 cc}
\]
This optional event which normally occurs at the beginning of an MTrk (i.e. before any non-zero delta-times and before any MetaEvents except Sequence Number) specifies to which MIDI Channel any subsequent MetaEvent or System Exclusive events are associated. The data byte cc, is the MIDI channel, where 0 would be the first channel.

The MIDI spec does not give a MIDI channel to System Exclusive events. Nor do MetaEvents have an imbedded channel. When creating a Format 0 MIDI file, all of the System Exclusive and MetaEvents go into one track, so it is hard to associate these events with respective MIDI Voice messages. (i.e. for example, if we wanted to name the musical part on MIDI channel 1 "Flute Solo", and the part on MIDI Channel 2 "Trumpet Solo", we would need to use 2 Track Name MetaEvents. Since both events would be in the one track of a Format 0 file, in order to distinguish which track name was associated with which MIDI channel, we would place a MIDI Channel MetaEvent with a channel number of 0 before the "Flute Solo" Track Name MetaEvent, and then place another MIDI Channel MetaEvent with a channel number of 1 before the "Trumpet Solo" Track Name MetaEvent.

It is acceptable to have more than one MIDI channel event in a given track, if that track needs to associate various events with various channels.

**B.9.2 - MIDI PORT**

FF 21 01 pp

This optional event which normally occurs at the beginning of an MTrk (i.e. before any non-zero delta-times and before any MIDI events) specifies out of which MIDI Port (i.e. buss) the MIDI events in the MTrk go. The data byte pp, is the port number, where 0 would be the first MIDI buss in the system.

The MIDI spec has a limit of 16 MIDI channels per MIDI input/output (i.e. port, buss, jack, or whatever terminology you use to describe the hardware for a single MIDI input/output). The MIDI channel number for a given event is encoded into the lowest 4 bits of the event's Status byte. Therefore, the channel number is always 0 to 15. Many MIDI interfaces have multiple MIDI input/output busses in order to work around limitations in the MIDI bandwidth (i.e. allow the MIDI data to be sent/received more efficiently to/from several external modules), and to give the musician more than 16 MIDI Channels. Also, some sequencers support more than one MIDI interface used for simultaneous input/output. Unfortunately, there is no way to encode more than 16 MIDI channels into a MIDI status byte, so a method was needed to identify events that would be output on, for example, channel 1 of the second MIDI port versus channel 1 of the first MIDI port. This MetaEvent allows a sequencer to identify which MTrk events get sent out of which MIDI port. The MIDI events following a MIDI Port MetaEvent get sent out that specified port.

It is acceptable to have more than one Port event in a given track, if that track needs to output to another port at some point in the track.