Automatic Annotation of Musical Audio for Interactive Applications

Paul M. Brossier
piem@altern.org

Centre for Digital Music
Queen Mary, University of London

under the Direction of Dr. Mark Plumbley and Prof. Mark Sandler
submitted in partial fulfilment for the diploma of Doctor of Philosophy

electronic version: `http://aubio.piem.org/phdthesis`

August 2006

Abstract

As machines become more and more portable, and part of our everyday life, it becomes apparent that developing interactive and ubiquitous systems is an important aspect of new music applications created by the research community. We are interested in developing a robust layer for the automatic annotation of audio signals, to be used in various applications, from music search engines to interactive installations, and in various contexts, from embedded devices to audio content servers. We propose adaptations of existing signal processing techniques to a real time context. Amongst these annotation techniques, we concentrate on low and mid-level tasks such as onset detection, pitch tracking, tempo extraction and note modelling. We present a framework to extract these annotations and evaluate the performances of different algorithms.

The first task is to detect onsets and offsets in audio streams within short latencies. The segmentation of audio streams into temporal objects enables various manipulation and analysis of metrical structure. Evaluation of different algorithms and their adaptation to real time are described. We then tackle the problem of fundamental frequency estimation, again trying to reduce both the delay and the computational cost. Different algorithms are implemented for real time and experimented on monophonic recordings and complex signals. Spectral analysis can be used to label the temporal segments; the estimation of higher level descriptions is approached. Techniques for modelling of note objects and localisation of beats are implemented and discussed.

Applications of our framework include live and interactive music installations, and more generally tools for the composers and sound engineers. Speed optimisations may bring a significant improvement to various automated tasks, such as automatic classification and recommendation systems. We describe the design of our software solution, for our research purposes and in view of its integration within other systems.

Résumé

Alors que les machines deviennent de plus en plus portables et partie intégrante de notre quotidien, il apparait clairement que le dévelopement de systèmes interactifs et omniprésents est un aspect important des nouvelles applications créées par la communauté scientifique. Nous nous intéressons à la construction d'une couche robuste pour l'annotation automatique de signaux audio, utilisable dans des applications variées, des moteurs de recherche de musique aux installations interactives, et dans des contextes divers, processeurs embarqués ou serveurs de contenu audio. Nous proposons d'adapter des techniques existantes de traitement du signal à un contexte temps-réel. Parmi ces techniques d'annotation, nous nous concentrons sur des taches de bas et moyen niveaux telles que la détection d'attaque, le suivi de hauteur, l'extraction du tempo et le modelage de notes. Nous présentons un environnement logiciel pour extraire ces annotations et évaluer les performances de différents algorithmes.

La première tâche sera de détecter les débuts et fin d'évenements sonores dans les flux audio avec une faible latence. La segmentation des flux audio en objets temporels favorise les manipulations et analyse de la structure métrique. L'évaluation de plusieurs algorithmes et leur adaptation pour le temps réel est décrite. Nous addressons ensuite le problème de l'estimation de la fréquence fondamentale, à nouveau en essayant de réduire le délai et le coût de calcul. Plusieurs algorithmes sont déployés pour le temps réel et testés sur des signaux monophoniques et des enregistrements complexes. L'analyse spectral peut-être utilisée pour annoter les segments temporels; l'estimation de descriptions plus haut-niveau est approchée. Des techniques pour modeler des notes et localiser le tempo sont approchées.

Les applications de cet environnement comprennent les installations musicales interactives et plus généralement des outils pour le compositeur et l'ingénieur du son. L'optimisation des vitesses de calcul peu apporter un bénéfice important a plusieurs tâches automatisées, telles la classification automatique et les systèmes de recommandation. Nous décrivons la conception de notre solution logicielle, pour nos besoins de recherche et en vue de son intégration au sein d'autres systèmes.

A la musique qui fait battre mon cœur.

Acknowledgements

The author is supported by a Studentship from the Department of Electronic Engineering at Queen Mary College, University of London. This research has been funded by the EU-FP6-IST-507142 project SIMAC (Semantic Interaction with Music Audio Contents) and by EPSRC grants GR/R54620 and GR/S75802/01.

Several people must be acknowledged for their continuous support of an invaluable help: Dr. Mark Plumbley and Pr. Mark Sandler, Dr. Juan Pablo Bello, Chris Duxbury, Samer Abdallah and Thomas Blumensath, Matthew Davies, Xavier Amatrian, Guenter Geiger, Philippe Depalle, François Déchelle, Peter Hanappe, Perfecto Herrera, Mercedes Pacho, my mother, my father and my three syblins. We should also stress that this research was collaborative and would not have been possible without the precious contributions and feedback from the research community.

The material used in this study included sounds from the Real World Computing (RWC) [Goto et al., 2003] database and the Music Information Retrieval Exchange [MIREX, 2004a, 2005a] database. MIDI files borrowed from the Mutopia Project [Mutopia project, 2000] were synthesised using Timidity [Toivonen and Izumo, 1999]. The database for the evaluation of tempo tracking was kindly provided by Stephen Hainsworth and Nick Collins. Algorithms were evaluated using our C implementation and external libraries for Fourier transform computations [Frigo, 1997, Frigo and Johnson, 2005], audio resampling and sound files access [de Castro Lopo, 2006a,b]. Data visualisation and plots were made with Gnuplot [Williams and Kelley, 1986], graphs edited using XFig and Inkscape. The content of this document was formatted using LaTeX.

Contents

Introduction

Developing robust methods for the annotation of music signal is required by new applications of computer music. We are interested in studying different techniques to analyse music signals in a real time fashion and with minimal delays. We propose modifications of existing techniques for their implementation in real-time, and we evaluate these techniques. The aim of our research is to investigate solutions to derive simple symbolic notations from complex audio signals within very short delays. We focus on rapid signal processing and Music Information Retrieval techniques to extract four musically relevant audio descriptors: onset times, pitch, beats and notes. An implementation is proposed as a collection of C routines, and methodologies for the evaluation and optimisation of the different algorithm are described. The framework we used to evaluate automatically extracted features against hand-annotated results is proposed as a collection of Python scripts. The results of our experiments measuring the robustness of each algorithm are discussed.

An introduction is presented in Chapter 1, where some of the characteristics of the human auditory system are recalled, and different approaches to the analysis of digital music signals are reviewed. Chapter 2 explains the task of segmenting music signals at the boundaries consecutive sounds, and gives a review of several methods to obtain onset times, the beginning of sound events. Several methods are evaluated and modified to allow the extraction of onset times within short latencies. These methods are implemented and evaluated against a database of manually annotated audio signals. The estimation of the fundamental frequency of music signals is addressed in Chapter 3, where we give a definition of the pitch, the perceptual attribute associated with frequency, before describing several methods designed to extract the fundamental frequency. These methods are evaluated on different types of music signals, and their computational cost are compared. Chapter 4 gives an overview of several methods to extract the tempo from musical audio, and a causal approach to beat tracking is described in details. Results obtained with this method on a corpus of manually annotated music signals are compared to the results achieved by other approaches. In Chapter 5 we review different approaches to the transcription of music signals in notes, and we evaluate the performance of different methods to model these symbolic notations within a short delay. Several software environments for the manipulation of musical signals are reviewed in Chapter 6, and we describe the approach we have followed to implement our software solution, the *aubio* library. Several examples of integrations of aubio with other softwares are described. Chapter 7 gives an outline of the main findings described in this document and pointers to further research directions.

Chapter 1

Background

New hardware and software enable new forms of interaction with sound. Both composers and listeners can experiment with new relations to sound objects and music. The use of symbolic notations in music composition and production environments has been growing over the past decades. Meanwhile, several research areas of the music community are driven towards the extraction of semantic meaning from a musical stream. However, little has been done to link the extraction of this semantic information to its applications in composition and recording.

Here we focus on the applications of music information retrieval techniques in the context of real time environments, such as digital audio workstations and live computer music installations. Live effects or audio editing environments imply strong constraints on the choice and implementation of algorithms. The calculation time should be kept minimal, and the algorithm would preferably be causal or have the smallest possible delay.

In Section 1.1 of this introductory part, we review some of the main characteristics of the human auditory perception. These characteristics influence the way we hear, listen to and create music, and are therefore important to consider in the design of listening systems. Section 1.2 gives an overview of some major concepts and techniques developed within the music research community, along with some examples of interactive applications developed around these techniques. Our research objectives are described in Section 1.3, where the organisation of the following chapters in this document is detailed.

1.1 Perception and psychoacoustics

The human auditory system is composed of three main parts: the outer ear, which collects and focus sound waves up to the timpani; the middle ear, where three tiny bones, the *ossicles*, amplify the vibrations of the ear drum and transmit them to the vestibulum; the inner ear, where a specific organ, the *cochlea*, contains specific nerve cells for the analysis of audio stimulus. These cells are organised along the *basilar membrane*, which is found inside the coiled, tapered conduit of the cochlea, and fire patterns down the auditory nerve, further up into the brain.

The human ear is an extremely precise analysis engine, capable of distinguishing very small variations in intensity, able to differentiate very slight changes in frequency, and to separate events within a very short time lag. In order to analyse audio signals in a musically meaningful way, understanding some of the human listening mechanisms is important. These mechanisms of human hearing are indeed complex, and to some extent, music is tailored for the ears of the human listener [Roads, 1996]. For modern text-books on *psychoacoustics*, the study of the subjective human perception of sound, see [Bregman, 1990, Deutsch, 1982, McAdams, 1987, Zwicker and Fastl, 1990]. An overview of some of the of major investigations on auditory perception and psychoacoustics was given in [Roads, 1996, Chapter 7].

1.1.1 Perception of intensity

The physical intensity of an audio signal is defined by the energy carried by the acoustic wave. Sound intensity is measured in terms of *sound pressure level* (SPL) on a logarithmic scale and normalised to the atmospheric pressure P_0:

$$SPL = 20 \log_{10}(P/P_0). \tag{1.1}$$

The perceptual attribute corresponding to the intensity is the *loudness*, and its relation to measured intensity is not trivial. The human listener is capable of differentiating small changes in intensity, but the perception of loudness also depends on the spectrum of the signal, its duration, the presence of background noise and other physical properties of the signal. A useful measure of loudness, the *phon*, was defined in [Fletcher and Munson, 1933]. By definition, at a frequency of 1000 Hz, one phon is equal to the SPL value in decibels. Throughout the rest of the spectrum, the loudness in phon corresponds to the actual loudness perceived by the listener rather than the intensity of the signal. The curves shown in Figure 1.1 are the contour of constant loudness across the frequency range and for different

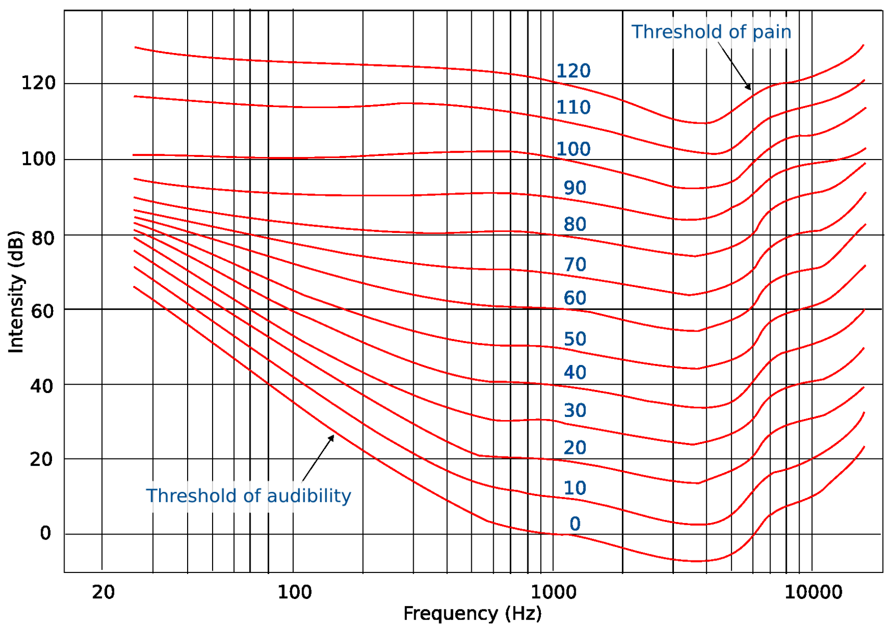


Figure 1.1: Fletcher-Munson Equal Loudness Contours, showing the perceived loudness as a function of the frequency and intensity of the stimulus. After [Fletcher and Munson, 1933, Ropshkow, 2005]

intensities. At an intensity of 110 dB (SPL), frequencies of 100 Hz, 1 kHz and 10 kHz are roughly perceived at the same loudness. However, at 40 dB, a frequency of 100 Hz would be just audible, and a frequency of 10 kHz would require a 10 dB boost to be perceived at the same loudness than a 1 kHz sound. The dependency between frequency and perceived loudness are important to design a system for the extraction of perceptual features from music signal. In the next chapters, we will use perceptually motivated filters to model these relations.

1.1.2 Perception of temporal features

Perceptions of frequency and temporal features are strongly related, and often cannot be separated into two distinct processes. There is however strong evidence that various types of temporal analysis occur within the inner ear and further in the brain of the listener [McAdams, 1987, Zwicker and Fastl, 1990, Bregman, 1990]. Amongst these mechanisms is a *period detector*: the nerve cells of the inner ear fire periodic patterns at the same rate as the waveform. When the period is short, the cells do not have enough time to recover and fire again within this period. In this case the cells fire patterns at a multiple of the period. Frequencies that can be detected in this way are up to about 4 kHz. Another mechanism allows us to detect amplitude modulations at frequencies between 75 and 500 Hz.

Another type of temporal encoding operated by the human ear allows for the analysis of sonic events: some of the nerve cells are triggered at *onset* and *offset* times, where a sound starts and finishes [Whitfield, 1983]. The attack of the note, where the sound starts rising, triggers these nerve cells, while the sustained part of the sound, where the note is held, does not. A detailed analysis of this phenomenon was given in a study of the *perceptual attack time* [Gordon, 1984]. The study included perceptual tests in which listeners were asked, while listening to repetitive sound patterns, to press a button at each perceived attack. Gordon observed that perceptual attack times were difficult to measure accurately and were perceived only slightly differently across listeners, given the imprecision of the measurements. The tests showed that perceptual attack times of tones could be perceived significantly later than the physical onset of the sound in the music signal, up to a few tens of milliseconds, depending on the instrument played and the way it is played. Gordon [1984] observed that the perceptual attack time of several sounds was dependent on the *timbre* of the instrument, this quality of a sound which enables us to distinguish one instrument from another [Grey, 1975]. In Chapter 2, we will look at ways to detect physical onset and offset times for different timbres.

As auditory nerve cells need to rest after firing, several phenomena may occur within the inner ear. Depending on the nature of the sources, two or more events will be merged into one sensation. In some cases, events will need to be separated by only a few millisecond to be perceived as two distinct events, while some other sounds will be merged if they occur within 50 ms, and sometimes even longer. These effects, known as the *psychoacoustic masking* effects, are complex, and depend not only of the loudness of both sources, masker and maskee, but also on their frequency and timbre [Zwicker and Fastl, 1990]. The different masking effects can be divided in three kinds [Bregman, 1990]. *Pre-masking* occurs when a masked event is followed immediately by a louder event. *Post-masking* instead occurs when a loud event is followed by a quiet noise. In both case, the quiet event will not be perceived – i.e. it will be masked. The third kind of masking effect is *simultaneous masking*, also referred to as *frequency masking*, as it is strongly dependent on the spectrum of both the masker and the maskee. Under certain circumstances, a quiet event, occurring while the masker event is being played, will not be heard. A representation

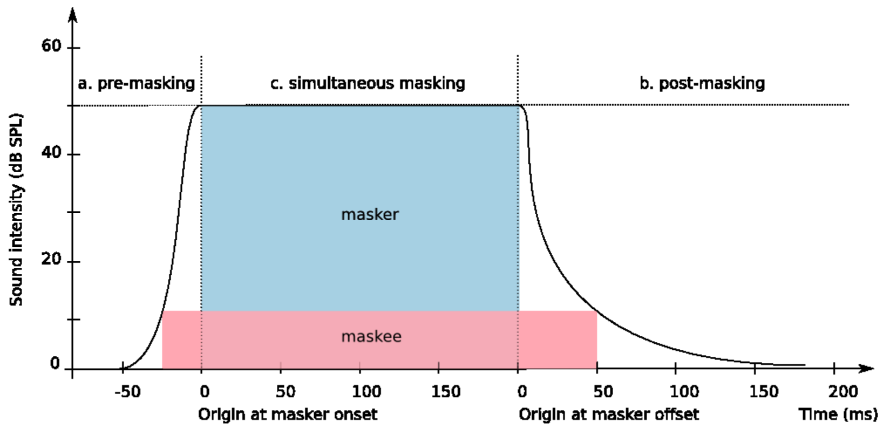


Figure 1.2: Schematic representation of the three types of psychoacoustic maskings. A masker event (blue) may mask another one, the maskee (red), in three ways: a. pre-masking: the maskee is followed by a louder event; b. post-masking: the maskee is preceded by a louder event. c. frequency masking: the maskee is masked by a louder event. After experimental results of Zwicker and Fastl [1990].

of the three main types of maskings is shown in Figure 1.2, with typical delays of about 50 ms for pre-masking and about 150 ms for post-masking. Establishing simple rules to model masking effects is not trivial, but realistic models are now used as the foundation of modern lossy coders, such as the well-known MPEG-1 Layer 3 (MP3) [Brandenburg and Bosi, 1997, Brandenburg, 1999] or more recently, Ogg Vorbis [Xiph.org, 2005]. These perceptual "speed limits" will be considered in Chapter 2 when designing a system for the extraction of temporal features such as the attack time of a sound.

1.1.3 Perception of frequency

The human ear is capable of distinguishing frequencies ranging from 20 Hz to 20 kHz, as well as small variations in frequency. Different frequencies are perceived at different regions of the basilar membrane of the human cochlea, and the distance from the middle ear to the region in the basilar membrane directly depends on the period of the audio waveform. These regions are referred to as the *critical bands* of the human auditory system [Scharf, 1970], and play an important role in the perception of harmony. Roeder and Hamel [1975] observed different perceptual effects when playing two pure tones of different frequencies. These effects are

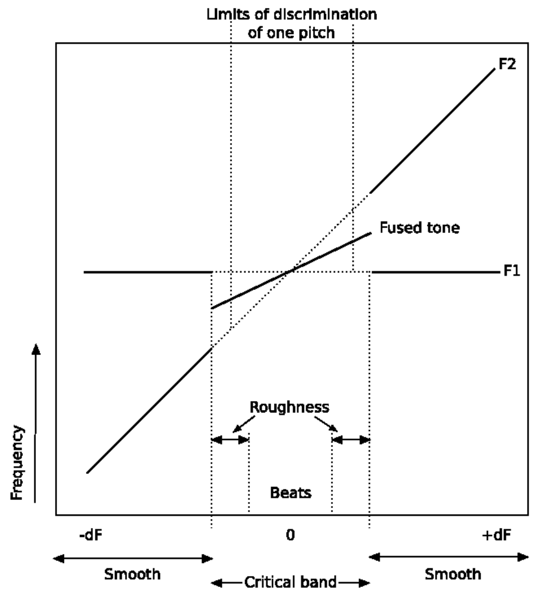


Figure 1.3: Schematic representation of the perception of two sine tones at frequencies F_1 and F_2 and played simultaneously. As the frequency difference $dF = F_1 - F_2$ is reduced, the perception of both tones changes from two distinct frequencies to a single fused tone. After [Roeder and Hamel, 1975]

diagrammed in Figure 1.3, where two tones of frequency $F1$ and $F2$ are played simultaneously. The widths of the critical bands change across the spectrum, with wider bands in the high frequencies (the scale of Figure 1.3 is arbitrary). When two sine tones have very close frequencies, the ear perceives them as a single fused tone corresponding to a frequency between those of the two tones. When the difference between the frequencies of both tones is small, the fused tone is perceived with a sensation of *beating* similar to an amplitude modulation. When the frequencies of both sine tones moves further apart, a sensation of *dissonance* is perceived – the term *roughness* is often preferred in the psychoacoustic literature. Finally when the frequencies of the two tones are found in different critical bands, both tones be identified as two distinct sources.

Frequency discrimination of the human auditory system is not limited by the width of the critical bands. The perception of *pitch*, the perceptual attribute of the frequency, is not directly related to the frequency of the signal. Indeed temporal encoding also plays a role in the way we perceive frequencies. The sensation of pitch is limited to a shorter range, and the perception of octave differences are for instance limited to a 60 Hz – 5 kHz range [Schubert, 1979]. When many frequencies are played together, the auditory system integrates the information from all the critical bands to decide the pitch of the source. The harmonic relationship between the partials of the source produce this sensation, but some inharmonic timbres, including noise, can also be perceived with a clear sensation of pitch. These perceptual cues participate in forming the sensation of timbre.

1.1.4 Cognition

The influence of neural activity and the acquired experience of the listener on the listening process is complex and not very well known. A trained human ear is able to analyse a complex signal into its different sources, identify each of these sources, and mentally follow them individually. The music itself is often tailored for the listener.

One important process occurring within the brain is the integration of neural signals coming from both ears, which permits the localisation of the source in space [Zwicker and Fastl, 1990]. Another example is that of the presence of echo in an auditory scene, which will be perceived as the reverberation of the main source from the walls of the room, rather than as a stream of events occurring within short delays. While the listener will not always be able to distinguish each of the reverberations, the sensation of echo will be clearly perceived. Another mechanisms of the human auditory system enables us to separate two different sources of repeated events with different rhythms into two or more distinct auditory streams [Bregman, 1990].

A trained ear will have the ability to mentally segregate the signals coming from different simultaneous sources. Specific abilities are developed by musicians, which enable them for instance to play their instrument while listening to another one. Finally, the cultural knowledge of the listener will also influence his perception of music.

1.2 Automatic annotation

Analog and digital systems have brought new means to study and understand auditory perception, speech and music signals. Computer systems have opened the way to digital media storage and faster computations. Complex systems for analysis and synthesis of audio signals and new composition tools have been designed.

The historical background of research in computer music helps in understanding the concepts of audio objects and symbolic representations.

1.2.1 Audio objects and semantic descriptors

In [Schaeffer, 1966], the approaches of both the composer and the listener to make and listen to music are discussed. Schaeffer [1966] develops his view of sound in his experiments on *musique concrète*, produced by editing together fragments of natural and synthetic sounds, and approaches the notion of *musical object*, sounds perceived as concrete entities and combined together when composing music. The properties of audio objects, their interaction with the external world and with other sound sources determine the way we perceive them. The notion of *auditory scene analysis* was introduced in [Bregman, 1990], where the perception of an acoustic scene is compared to the vision of an image. The different objects are first decomposed by the determination of their contours, and further identified by their details. Audio objects have different shapes in time and frequency. Real world sounds are known or new objects; synthetic sounds can mimic real world objects, and create new ones.

Analysis and synthesis of music signals have seen a growing interest in recent decades. As audio tapes opened ways to new musical applications and became popular, the mid-1960s brought new approaches and strategies to analyse and model speech and music signals. In [Risset and Matthews, 1969, Risset, 1969], frequency analysis of trumpet tones was performed using a computer. The system could sample values for amplitude and frequency of multiple sinusoidal components. The variations described by these measured values were approximated by linear segments, which in turn could be used to control the frequency and amplitude of synthesised sinusoids. By manipulating a small number of parameters describing the line segments, new sounds could be synthesised using this system, and identified by the listener as resembling to that of a trumpet. Similar strategies were used to study the nature of the timbres from different music instruments [Moorer and Grey, 1977b,a, 1978] and evaluate the perceptual relevance of synthesised tones [Grey and Moorer, 1977].

Analog vocoders were widely used for speech modellings in the 1960s, but the development of the digital vocoder [Portnoff, 1976] was a milestone towards high quality digital audio processing. Several major improvements were brought around the phase vocoder, shown to be useful for analysis and synthesis of music signals in [Moorer, 1978], including the efficient modelling of voice signals using sinusoidal representations [McAulay and Quatieri, 1986] and the decomposition of the signal in terms of deterministic and stochastic components [Serra, 1989, Serra and Smith, 1990]. Even though other time-frequency representations and wavelet transformations are being explored, sinusoidal representations have played an important role in the speech and music communities. With the increase of available processing power and the optimisation of fast Fourier transform implementations, techniques to synthesise and control several hundred sinusoids are now available on a standard desktop computer [Freed et al., 1993] allowing the analysis, generation and modification of complex synthetic timbres.

Beyond the analysis of the spectral structure of sounds and timbres, the research community has been interested in designing systems able to transcribe an acoustic wave into *notes*, which define the boundaries of audio objects. Early work on music transcription included a system to analyse a monophonic recorder or a symphonic flute [Piszczalski and Galler, 1977] and a system to follow duets, i.e. two-voice compositions [Moorer, 1977]. Segmentation of the audio signal into notes was done by the detection of important changes, in terms of signal amplitude or fundamental frequency frequency. In both systems, the user had to provide the smallest duration as a parameter, and the boundaries of extracted notes were defined at multiples of this smallest possible duration. The frequency of the flute partials were searched for in the frequency domain, and the fundamental frequency selected as the partial with the most energy. Noticing that the fundamental frequency was sometimes wrongly selected, Piszczalski and Galler [1977] used a stronger weight for the partials at low frequencies.

Further improvements to transcription systems were brought by the use of separate techniques to segment the objects at their boundary [Foster et al., 1982], and better modelling of the note accents [Chafe et al., 1982]. The use of musicological rules [Lerdahl and Jackendoff, 1983] has also been popular to infer the relations between these objects. Extracting symbolic notations from acoustic signals consist of drawing a series of rules to describe group of notes and infer metrical structure of a musical piece. Specific strategies were deployed to tackle this task. Algorithms for the recognition and grouping of spectral and temporal patterns have been developed [Mont-Reynaud and Goldstein, 1985]. An artificial intelligence technique, the blackboard approach was described in [Chafe et al., 1985], using event detection and metrical structure informations along the frequencies of the partials to infer hypothesis. Another approach, the clustering of partials into a timbre classification, was implemented in [Kashino and Tanaka, 1993] for source separation and tone modelling, based on features such as harmonic mistuning and attack time.

While computer music applications were adopted amongst musicians and composers, a variety of complex synthesis and composition algorithms have been designed and used by performers and composers [Roads, 1996]. The MIDI format [MIDI Association, 1983], using symbolic data to describe the note attributes, has become a standard control protocol and is widely used in recording studio and computer systems [Loy, 1985]. Existing and new control interfaces can be built to control dedicated synthesis modules in real time.

Several coding systems have been designed based on time frequency representations, using psychoacoustic models [Brandenburg, 1999, Xiph.org, 2005], or harmonic components plus noise decompositions [Purnhagen and Meine, 2000]. Modern specifications such as MPEG4 include standards such as *Structured Audio* [Scheirer, 1998a, Scheirer and Vercoe, 1999], inherit from Csound, a pioneering programming language for sound synthesis [Boulanger, 1998], and include the MIDI standard. New systems are being designed for the transmission of musical signals and symbols [Amatrian and Herrera, 2002].

The extraction of features from audio signals is used for a varied range of applications. Several research areas about analysis and synthesis of musical sounds aims at reducing the amount of data to process by extracting semantic informations from them to gain better control and create interactive systems.

1.2.2 Application fields

Advanced synthesis techniques for instrument modelling have been described, and the control over the parameters of the models has open the way to new creative applications. Realistic modelling of plucked strings [Karjalainen et al., 1993] and stringed instruments [Smith, 1993] have been proposed.

Symbolic notation of music permitted the elaboration of automatic accompaniment systems based on symbolic representations [Dannenberg, 1985, Vercoe, 1985, Dannenberg and Mont-Reynaud, 1987], sampling synthesis has allowed the elaboration of more complex generative systems [Pachet, 2002]. Symbolic music representation have also allowed the automation of symbolic pattern recognition [Huron, 1995]. Similarly score alignment and score following systems began with symbolic data and are now mixing signal processing and musicological rules to [Raphael, 2001b, Orio and Déchelle, 2001].

The parameters of an audio effects can be controlled automatically in an adaptive fashion [Verfaille, 2004]. Feature extraction of different signal characteristics start being used for such audio effects, so that the sound source can be used as a control over the effect algorithm.

Onset and pitch annotation were shown to improve significantly sound processing algorithms such as time scaling [Ravelli et al., 2005], by preserving the perceptual characteristics of the attacks. New interfaces are being created for the realignment of drum loops by which a sound object can be displaced within a recorded texture [Aucouturier and Pachet, 2005]. Fine grained modification of existing melodies requires the annotation of both pitches and onset times [Gómez et al., 2003b].

Recent sampling synthesis systems make use of database of sound segments, annotated with pitch and texture attributes [Casey, 1994]. File format specifically dedicated to these banks of sounds permit the creation, storage, exchange and modification of these samples. Sampling synthesis can be used in various creative ways to create new sound instruments based on automatically segmented audio [Aucouturier et al., 2004].

Similarly, automatic classification of sound segments is employed in creative applications like *micro-montage* to create new sound textures [Caires, 2004]. The classifier clusters short time slices into texture categories, that can be concatenated and controlled by an amplitude envelope. Music mosaicing [Zils and Pachet, 2001, Casey, 2005] is another new application that picks from an annotated corpus of audio data to mix and reuse existing samples. Other sampling synthesis based on audio segmentation include [Jehan, 2004, Collins, 2004].

Music recommendation and genre classification systems have to take a small number of decisions over a large corpus of music recordings, and thus process large amounts of data in a limited time. Real time identification of music such has recently becomes usable as results can be obtained within several tens of seconds [Wang, 2003]. Some applications such as *query by music* specifically require the extraction of semantic data [Pampalk et al., 2005, Tzanetakis, 2002]. Complex systems are often based on a classification technique to cluster the elements of the database. The classifiers most often include the extraction of signal features on time segments. The speed is one of the most prominent requirement for search engines.

1.2.3 Processing digital data

Only a few decades ago, most recording formats were analog. Although audio tapes and vinyl records are still in use and being produced, important parts of our music archives are now accessible on digital media, such as the popular compact disc. When designing systems dealing with audio data, a number of pitfalls, specifically encountered in processing digital signals, should be avoided. The precision of the data itself and the time required to access and process this data are our specific concerns.

Accessing large amounts of data requires some time, processing power and memory. Reducing these computation times will allow us to build responsive systems that can take decisions within short time lags. Despite their large size, storage of digital audio content is not our main concern: modern storage can access these media in real time and faster. The issue is to deal with substantial amounts of data in a way that is fast and efficient enough for a specific application. The speed and efficiency required to achieve acceptable results will depend on the application.

Obviously, the characteristics of digital signals affect the difficulty of processing them. Whether the recording is sampled at 8 or 192 kHz, onto 8 or 64 bits, the dynamics and spectral content of digital signals are limited by these specifications and can only approach the resolution of the recorded acoustic vibrations. Quantisation noise, harmonic distortion, floating-point computation errors and other artefacts of digital systems need to be carefully considered in the design of a music annotation systems [Roads, 1996]. Available time and resolution limit the precision of analysis algorithms, and various strategies must be deployed to preserve both physical and perceptual attributes of music signal.

Another problem resides in the difficulty of constructing large databases of annotated sounds. Gathering this data is often challenging, as large collections are often held under copyright law and only accessible to recording and label companies. Fortunately, recent years have seen the development of initiatives in this direction and collaborations between different research teams, such as the Real World Computing (RWC) music database [Goto, 2004] or the Music Information Retrieval Exchange [MIREX, 2005a]. A strong movement in favour of Copyright Free multimedia contents has also grown bigger in the past years. Large numbers of audio samples, song extracts, or even multi-track master recordings are now available under Creative Commons licenses or similar free licenses [Freesound, 2005, Mutopia project, 2000]. This forms important new material for the establishment of research databases and the reproduction of results.

1.3 Summary

The characteristics of the human ear are complex and influence strongly our musical activities, from composition to listening. The definition of semantic objects is useful for many applications. From a signal processing point of view, the definition of a semantic concept is difficult for music signals, because the level of abstraction required to englobe a majority of these concepts is on a much higher level than that of the signal characteristics. Moreover, processing digital signals requires the development of specific strategies to avoid various pitfalls and artefacts inherent to digital systems. Annotating musical audio signals precisely consists in reducing the size of raw audio signals to a few semantically and musically meaningful statements.

Modern applications of these sound processing techniques are now emerging and evolve toward semantic descriptions of music contents. Much research has been devoted to automate the annotation of musical signals into audio objects and musical semantics. The formalisation of this annotation task includes the design of the algorithm, the implementation of the system and the evaluation of its characteristics.

Chapter 2

Temporal segmentation

Temporal segmentation of an audio stream into shorter elements is a fundamental step in the transformation of sounds into semantic objects. Much research has been devoted to this operation, and in the last two decades, different algorithms have been developed to automatically separate music signals at the boundaries of audio objects: where the note starts – the *onset* – and finishes – the *offset* [Moelants and Rampazzo, 1997, Klapuri, 1999b]. The extraction of onset times is useful in sound processing applications for accurate modelling of sound attacks [Masri, 1996, Jaillet and Rodet, 2001], helps transcription systems in localising the beginning of notes [Bello, 2003, Klapuri, 2004], and can be used in sound software editors to break sound files in logical parts [Smith, 1996]. Onset detection methods have been used for music classification [Gouyon and Dixon, 2004] and characterisation of rhythmic patterns [Dixon et al., 2004]. Several systems for tempo tracking make use of detected onsets to infer the location of beats [Scheirer, 1998b, Davies and Plumbley, 2004]. A system capable of detecting these onset times as they occur, just like the human listener does, enables new interactions between acoustic and synthetic instruments [Puckette et al., 1998]. The establishment of robust methods for the real time detection of onsets is thus an important task for the elaboration of music installations and interactive systems.

The difficulty of constructing a singld detection method that can label all relevant observations is explained in the first section of this chapter. A number of approaches for the detection of onsets in musical audio are described in a second part, from temporal techniques to filter-bank and statistical methods. These approaches can generally be separated in two tasks: the construction of a detection function to characterise the changes in the signal, and the peak-picking of this function, to extract perceptually relevant onset times [Bello et al., 2005]. We will see that with real time requirements, the peak-picking process, where the selection of relevant changes occurs, needs to be specifically addressed. Indeed we aim at minimising the delay and achieve temporal precision, two constraints required to approach the responsiveness of the human ear. A method for the low-latency peak-picking of onset times is proposed and the system is implemented as a collection of C routines.

Because perception of onsets is a subjective process of the human auditory system, the evaluation of onset detection methods is a complex task. A framework to compare extracted onset times to hand-labelled annotations is described and tested over different detection methods. Localisation and precision of the extracted onset times are evaluated against manual annotations, and the computational costs of the different methods in our implementation are estimated.

2.1 Labelling musical changes

Moelants and Rampazzo [1997] describe a *perceptual onset* in a musical signal as the "perceived beginning of a discrete event, determined by a noticeable increase in intensity, or by a sudden change in pitch or in timbre." The term onset detection refers here to the detection of the beginnings of discrete events in acoustic signals [Klapuri, 1999b]. Two examples of sounds presenting perceptual onsets are shown in Figure 2.1. The drum sound on the left of Figure 2.1 is produced by a snare drum and starts after about 10 ms, as can be seen in the waveform with a sudden amplitude increase, and in the spectrogram with an increase of energy in all the bands of the spectrum. Percussive sounds such as drums or struck strings will often form sharp attacks, presenting a sudden increase of energy in their waveform and a broadband burst in their spectrum. We will refer to these broadband energy bursts as *percussive onsets*. A second onset, less loud than the first one, can be perceived in the percussive sound of Figure 2.1, after about 270 ms. Although less apparent than for the first event, this second event also presents a broadband increase of energy.

Other instrument timbres, such as voice or string instruments, present smooth transitions from one note to the other, and characterising these changes is subtle. The waveform of the viola recording in Figure 2.1 shows the transition between two notes with different pitches. This viola sound is perceived as a clear change from one pitch to another, with no noticeable change in loudness or timbre. These non-

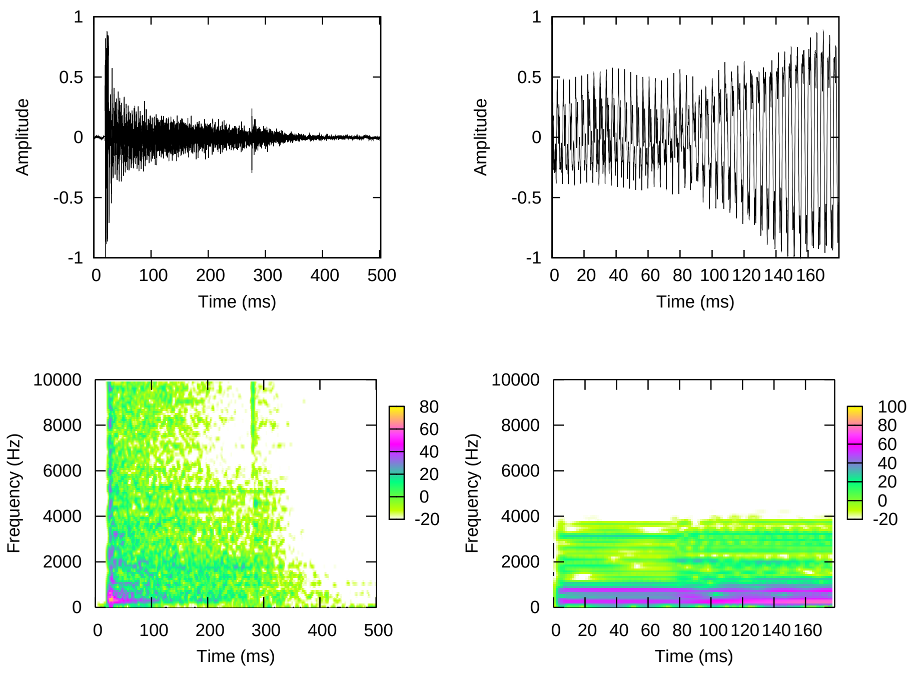


Figure 2.1: Examples of sounds creating the perception of onsets. Left column: snare drum hit at 10 ms and rimshot at 270 ms. Right column: two consecutive notes played legato on a viola; transition after about 80 ms. The waveform and spectrogram of each sound are plotted in top and bottom raw.

percussive onsets will be referred to as *tonal onsets*. The characterisation of onsets in polyphonic audio recordings is not trivial, since they can be defined by changes in loudness, pitch and timbre. The energy of the signal may be modulated by the presence of *tremolo*, and the frequency may be modulated with *vibrato*. These gradual changes are perceived as variations in amplitude or frequency, but not as discrete events.

With polyphonic signals, when different sound sources play simultaneously, the notion of attack time becomes less precise, as the attacks of simultaneous sound sources mix together. Observing sound events to define their temporal boundaries is a complex task because their nature changes not only from sound to sound – burst of energy across the spectrum for percussive sounds, or large variation of the harmonic content for tonal or voiced sounds – but also when different sounds occur together.

Gordon [1984] showed that perceived attack time was dependent on both timbre and loudness, and could be delayed from several tens of milliseconds by the actual note onset in the waveform. Recent psychoacoustic studies have shown that the perception of attack time is dependent of frequency [Moore, 1997]. The context in which a sound takes place will also change the way we perceive its attack. Fusion of simultaneous events may occur according to loudness and timbre, and two sound events played within 20 to 50 ms are usually perceived as synchronous [Bregman, 1990]. With repetitive temporal patterns separated by less than 80 ms, a sensation of streaming is perceived: the consecutive events are merged together; events separated by more than 100 ms can usually be identified by a human listener, who is then able to count several consecutive events [London, 2002]. Experimental measurements have confirmed that a time of 90 to 100 ms also corresponds to the limits of virtuosity for the production of notes [Repp, 1993, Friberg and Sundström, 2002] and for very short events to be perceived as distinct notes [Sundberg et al., 2003]. The minimum interval between two consecutive onsets is thus dependent on the context in which these onsets occur. As we are interested in detecting musically and perceptually relevant onset times, manual annotations are required to obtain these relevant onset times. A database of manually annotated sounds will be used in Section 2.5 to evaluate the performance of several onset detection methods.

Slicing an audio recording is a task known to recording engineers and computer musicians, for example when they select segments of recordings for sampling synthesis [Roads, 1996]. Compositional methods have been developed around sound samples and useful representations of music signal have been constructed using onsets and offsets sequences [Smith, 1996]. This slicing operation may require a higher temporal precision than that achieved by the listeners of the experiments of Gordon [1984]. For instance, sampling synthesis techniques use zero-slicing for the selection of the attack time [Roads, 1996]. Slicing was originally performed by hand, initially on dedicated tape machines [Schaeffer, 1966], later using a computerised waveform display and time frequency representations to help the selection of precise locations [Leveau et al., 2004]. For applications such as audio collage and resampling, the sample will be sliced preferentially at the beginning of a note, with the attacks of each object correctly preserved so that slices are perceptually relevant when played in isolation. The drum sound plotted in Figure 2.1 was intentionally sliced 10 ms earlier than its optimal slice point to display the sharp attack. When consecutive events overlap in time, attempts to minimise "leakage" from the previous segment into the current attack are also considered when determining the best slicing location [Roads, 1996].

We have seen that perceived attack time varies against timbre, frequency and loudness, and that two consecutive events are generally perceived as distinct when their attack time is separated by more than 50 ms. This suggests that describing musical changes using discrete time events implies observing the temporal features of the audio signal at a lower sampling rate than that of the audio signal. However, applications such as sampling synthesis may require a precision down to the sample to produce perceptually relevant slices.

2.2 Perceptual models for temporal segmentation

A first step in the extraction of discrete onset times is the evaluation of the amount of variation found in the signal. For a given time frame, a measure based on the characteristics of the audio signal is computed. Consecutive observations of this measure are gathered to form an *onset detection function* [Klapuri, 1999b, Bello et al., 2005]. The task of this onset detection function is to provide a mid-level representation: a function at a lower sampling rate than the low-level of the audio signal, reflecting the temporal features of the recording in order to obtain the high-level onset locations. These onset detection functions should present sharp peaks at onset times and no peaks during sustained notes and background noise. In a second step, peaks in this function will be selected to extract relevant onset times. The functions can be built using one of three methods: directly on the waveform in the temporal domain, in the spectral domain using several frequency bands or a phase vocoder, or using machine learning techniques on different features of the signal.

Before the construction of a detection function, some preparation can be performed to accentuate or attenuate various aspects of signal. These *pre-processing* steps depend on the requirements of the system, and may include the normalisation of the energy to minimise loudness changes across a collection, as well as algorithms to remove clicks and reduce the level of noise in the recordings.

As percussive sounds present important bursts of energy at the beginning of each event, an intuitive attempt to detect percussive events is to measure the energy of the signal to detect these bursts. Schloss [1985] used the energy contour of the waveform to find the attacks of percussive sounds, with an energy envelope follower be written as follows:

$$D_H[n] = \sum_{m=-N/2}^{N/2} w[m]x[n+m]^2, \tag{2.1}$$

where $w[m]$ is a smoothing window to evaluate the average energy over the window of width N. This approach can be successful at detecting sharp attacks of percussive sounds, which present abrupt energy variations, but fails at detecting several timbre and frequency changes, as we will see in Section 2.5.

To reflect changes in the spectral structure of the signal, a number of detection functions have been proposed based on a time-frequency representation, a view of the signal represented over both time and frequency. Time-frequency reperesentations can be obtained using either several frequency bands [Scheirer, 1998b, Klapuri, 1999b, Puckette et al., 1998], either short-time Fourier transforms [Masri, 1996, Foote and Uchihashi, 2001, Bello et al., 2003, Hainsworth and Macleod, 2003, Duxbury et al., 2003]. Multi band and Fourier transform methods rely on the notion of *transients* – transitional zones of short duration characterised by the non-stationarity of the signal spectral content – to model musical changes. These approaches were recently reviewed in a tutorial article [Bello et al., 2005]. To describe temporal features at different frequencies, Scheirer [1998b] used six frequency bands to analyse transients across different frequency regions, obtaining a function preserving the temporal features of the sound by combining the results of each band. A real-time implementation of a multi-band onset detector was described in [Puckette et al., 1998], where the logarithmic distance between two consecutive frames was measured in eleven bands.

Using 21 bands, Klapuri [1999b] constructed a detection function by summing over the bands using psychoacoustically motivated energy weightings. He also noted that the logarithmic derivative of the energy produces sharper peaks, closer to the attack time, linking his observation to that of Moore [1997], who suggests that the smallest perceivable variation in intensity is proportional to the intensity of the signal: the auditory system perceives relative intensity changes, rather than absolute intensity levels. Klapuri's results showed that robust detection can be achieved on percussive onset and polyphonic recordings, but failed on some tonal onsets of a symphony orchestra, and could produce false detection on strong amplitude modulations. Several methods based on spectral frames of the signal have since been proposed, either to address specifically tonal onsets [Bello et al., 2003] or to handle various timbres [Duxbury et al., 2003]. These approaches have been shown to be successful on a variety of signals [Bello et al., 2005]. They are suitable for real time implementation since time-frequency representations can be computed efficiently using Fourier transforms. Several of these methods are reviewed in the next section.

Other signal models have been proposed for the reduction of musical audio to a description of its rhythmic content, including machine learning techniques such as Markov chains [Punskaya et al., 2002] and Support Vector Machines (SVM) [Kapanci and Pfeffer, 2004]. Markov chains use probability models to estimate the likelihood of a transition from one *state* of the chain to another. They are useful for music signals as they can model model both continuity and abrupt changes [Rabiner, 1989]. The method described in [Punskaya et al., 2002] directly uses the amplitudes of the samples as the state of the model, and was shown to be efficient at detecting abrupt changes in synthetic and speech signals. Because this model works at the sample level, it can be used for denoising applications and click removal. Markov chains can predict future states given a set of past observations, and this method could also be applied to the restoration of old recordings, to fill gaps found in damaged records. However the system does not scale for an efficient detection of onset times, as several thousands of iterations are needed to obtain the position of changes.

A successful approach specifically designed for the detection of perceptual onsets was described in [Abdallah and Plumbley, 2003], where the signal is represented using Independent Component Analysis (ICA). From the set of Gaussian distributions describing a current frame of the signal, a function to measure the *surprisingness* of this frame is built as the likelihood of this frame to occur given a series of past events. This technique allows for the modelling of the probability of different sound objects and the training of these probabilities to model specific sound events. For the segmentation of audio with tonal events presenting soft transitions, Kapanci and Pfeffer [2004] adopt a different approach: rather than searching for precise change points, they evaluate whether two frames separated by a certain temporal distance could be produced by the same sound event. Each time frame is described by a vector of signal features: amplitude, fundamental frequency and relative weights of the first three harmonics. An SVM is used to identify groups of frames corresponding to the same sonic event. After training, the system was shown to be able to correctly segment a corpus of solo singing recordings. However the system is computationally intensive, since the detection of each onset depends on the analysis of past and future frames, and hence not easily applicable to real time implementations. The SVM has also been employed for the recognition of repetitive shapes in percussive signals and differentiate different drum instruments [Tindale et al., 2004]. Systems capable of learning specific patterns of the attack open the way to promising applications, as they could be used for the recognition of different timbres. However, real time implementation of these methods is difficult as these algorithms are complex and often computationally intensive.

2.3 Phase-vocoder onset detection functions

A phase vocoder is used to obtain a time-frequency representation of the signal. The phase vocoder and its use for musical signals have been described in details in the literature [Portnoff, 1976, Moorer, 1978, Dolson, 2001, de Götzen et al., 2000]. The notations we use throughout this document are as follows: for a signal x at time n, we define $\mathbf{X}[n]$ as its Short Time Fourier Transform (STFT). $X_k[n]$, the value of the complex spectral component in the k^{th} bin at n, can be expressed in its polar form as $|X_k[n]|e^{j\phi_k[n]}$ where $|X_k[n]|$ is the bin's spectral magnitude, and $\phi_k[n]$ its phase. Typical window size used for each the phase vocoder is 1024 or 512 samples, with an overlap rate of 50% or 75%, so that the window slides of 512 or 256 samples between each analysis frame. At 44100 Hz, a hop size of 512 samples give a temporal quantisation of 5.6 ms, which is a reasonable resolution to distinguish onsets separated by a few tens of milliseconds.

High Frequency Content

To favour the selection of wide-band burst of energy over other energy changes such as amplitude modulation, a stronger weight can be given to the high frequency components of the spectrum. Masri [1996] proposed a *High Frequency Content* (HFC) function, constructed by summing the linearly-weighted values of the spectral magnitudes:

$$D_H[n] = \sum_{k=1}^{N} k|X_k[n]|^2 \tag{2.2}$$

where $X_k[n]$ is the k^th bin of the STFT taken at time n. This operation emphasises energy changes occuring in the higher part of the spectrum, especially the burst-like broadband noise, usually associated with percussive onsets. However, the function is less successful at identifying non-percussive onsets – legato phrases, bowed strings, flute – which do not present such wide-band bursts.

Spectral difference

Harmonic components sliding from one fundamental frequency to the other may be missed by the energy and HFC detection functions, for instance when only small energy changes are observed. Other methods attempt to compensate for the shortcomings of the HFC by also measuring the changes on the harmonic content of the signal. One of such methods, known as the *spectral difference* [Foote and Uchihashi, 2001], calculates a detection function based on the difference between the spectral magnitudes of two successive STFT frames:

$$D_s[n] = \sum_{k=0}^{N} \left| |X_k[n]|^2 - |X_k[n-1]|^2 \right|. \tag{2.3}$$

This function attempts to quantify the amount of change found from one frame to another, rather than frame-by-frame measurements implemented by both the energy and HFC functions.

Phase deviation

Alternatively, a different approach [Bello et al., 2003] consists in building a function that measures the temporal instability of the phase. Tonal onsets will be identified by important phase variations. The energy burst found in percussive onsets also present such phase variations.

A steady state signal is expected to have the phase constantly turning around the unit circle. The phase delay, its angular speed, can thus be assumed to be constant, and its acceleration null. Phase changes can thus be detected looking at the phase acceleration. The function can be constructed by quantifying the *phase deviation* in each bin as:

$$\hat{\phi}_k[n] = \text{princarg}\left(\frac{\partial^2 \phi_k[n]}{\partial n^2}\right), \tag{2.4}$$

where princarg maps the phase to the $[-\pi, \pi]$ range. A useful onset detection function is generated as:

$$D_\phi[n] = \sum_{k=0}^{N} |\hat{\phi}_k[n]|. \tag{2.5}$$

A drawback of this function is that important phase changes may also occur at places not related to a musical change: noisy components of the signal will usually present an unstable phase. Although this may not affect tonal events with strong harmonic components, large variations may occur as as the signal becomes more percussive and noisy.

Complex-domain distance

In order to quantify both percussive and tonal onsets, the spectral difference and phase based approaches can be combined in the complex domain [Duxbury et al., 2003] to generate a prediction for the current spectral frame, $\hat{X}_k[n] = |X_k[n]|e^{j\hat{\phi}_k[n]}$, where $\hat{\phi}_k$ is the phase deviation function defined in Eq. 2.4. Then by measuring the complex-domain distance between target and observed STFT we obtain:

$$D_C[n] = \sum_{k=0}^{N} \left\| \hat{X}_k[n] - X_k[n] \right\|^2 . \tag{2.6}$$

This measure, similar to a Euclidean distance but in the complex domain, evaluates the distance between the current frame and the frame predicted from the previous one assuming both the phase shifting and the amplitude are constant.

Kullback-Liebler distance

Further alternative measures can be used to evaluate the distance between two consecutive spectral vectors. As we are looking at highlighting increase of energy, while ignoring decreases, the Kullback-Liebler distance can be used to highlight the large variations and inhibit small ones:

$$D_{\mathsf{kl}}[n] = \sum_{k=0}^{N} |X_k[n]| \log \frac{|X_k[n]|}{|X_k[n-1]|}. \tag{2.7}$$

This function accentuates positive amplitude changes: large peaks will be raised when the signal goes from silence to an event, as the denominator will be much smaller than the numerator. A variation of this function is proposed in [Hainsworth and Macleod, 2003], which removes the $|X_k[n]|$ weighting, accentuating the amplitude changes in the function:

$$D_{\mathsf{mkl}}[n] = \sum_{k=0}^{N} \log \frac{|X_k[n]|}{|X_k[n-1]|}. \tag{2.8}$$

To prevent the function from reaching negative values, which would increase the complexity of the peak-picking, and to ensure the function is defined even when a

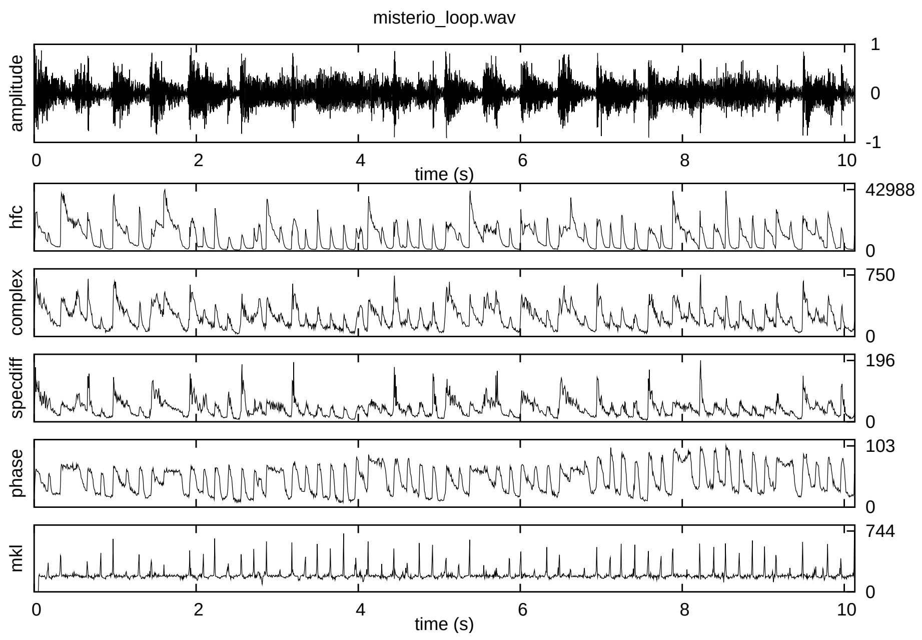


Figure 2.2: Examples of onset detection function profiles: HFC (hfc), Complex domain (complex), spectral difference (specdiff), Phase (phase), Modified Kullback-Liebler (mkl). Sound sample: Misterio, Azymuth

series of small values is encountered, we can further modify the function as follow:

$$D'_{\mathsf{kl}}[n] = \sum_{k=0}^{N} \log\left(1 + \frac{|X_k[n]|}{|X_k[n-1]| + \epsilon}\right), \tag{2.9}$$

where ϵ is a small constant, typically $\epsilon = 10^{-6}$. This constant is designed to avoid large variations when very low energy levels are encountered, and thus prevents large peaks in the detection function $D'_{\mathsf{kl}}[n]$ at offset times.

Examples of onset detection function profiles

In Figure 2.2 and Figure 2.3, examples of onset detection profiles obtained for two polyphonic recordings are shown. See Appendix B for availability of the recordings. The first example (Figure 2.2) is an excerpt of a Brazilian song by Azymuth, containing a brass ensemble and drums. The rhythmic structure of the excerpt appears clearly defined in the profile of each functions, with peaks sharper or less

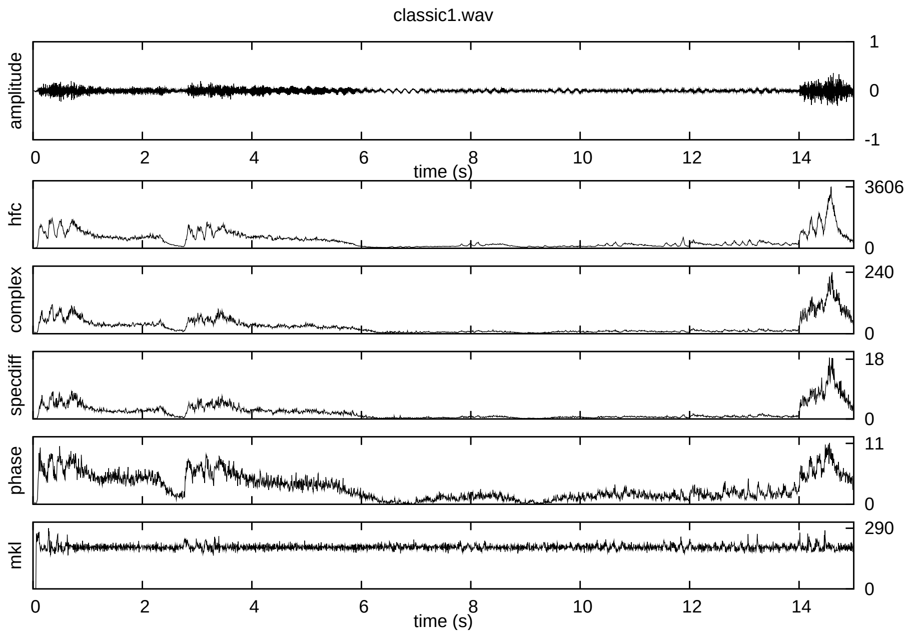


Figure 2.3: Examples of onset detection function profiles: HFC (hfc), Complex domain (complex), spectral difference (specdiff), Phase (phase), Modified Kullback-Liebler (mkl). Sound sample: First measures of the 5^{th} Symphony, Beethoven

sharp depending on the function. The brass notes tend to create some small variations in the spectral difference, the phase based approach and to a lesser extent, the complex-domain method. These variations create spurious peaks that the peak selection should carefully avoid in favour of the selection of the main peaks. The modified Kullback Liebler function, defined in Eq. 2.9, creates sharp spikes at percussive onsets; in this example, the Kullback Liebler function will give correct results for all onsets in the file.

The second example in Figure 2.3 shows the first measures of Beethoven 5^{th} Symphony. The violins start playing 8 notes *forte* from 0 to 6 s then continue their phrase *piano*, from 6 to 14 s, before the timpanist starts playing, from 14 s to the end of the file. The profile of the HFC allows the larger peaks to be clearly distinguished. However, peaks on notes with low energy have a very small magnitude. These magnitude differences tend to make the thresholding and peak picking operations difficult. The Kullback Liebler approach appears not as successful in detecting tonal onsets with weak transient components. The profile of the phase-based detection function is the only one containing all the peaks corresponding to the actual note onsets, and despite the presence of noise, this functionwill give best results after the selection of the relevant maxima. The differences between the profiles obtained for the two recordings and the presence of large amplitude changes within each example illustrate the difficulty of determining the best algorithm to select all relevant peaks in the onset detection functions.

2.4 Temporal peak picking of note onsets

The final selection of the onset locations consists in identifying local maxima in the detection functions that correspond to perceptual onsets. Depending on the signal content, peaks present in the detection function will be sharper or less sharp and may be masked by noise, either due to actual noise in the music signals or to other aspects in the signal, such as vibrato and amplitude modulation. Intuitively, the characterisation of onset times in the detection function is reduced to a *peak-picking* operation: the selection of local maxima above a given threshold value. Effective temporal peak-picking methods are required for the robust identification of onset times in the detection function. Rather than selecting local maxima, Puckette et al. [1998] proposed to select onset times when abrupt amplitude increases occur in the detection function, as implemented in the `bonk~` object for PureData (Section 6.1.2).

This implementation was informally tested – see Appendix B for examples of results – and found to detect accurately percussive onsets within short delays. However, several lower energy onsets are discarded on polyphonic recordings, and most tonal events are missed. Detecting these increases is efficient on sharp attacks, but fails on long attacks where the growth of the detection function is too slow. Alternative approaches for the selection of onset times have been proposed, with for instance the use of machine learning techniques to identify some characteristic shapes in the detection function is described in [Abdallah and Plumbley, 2003, Tindale et al., 2004]. Because of their complexity and their high computational cost, these approaches are difficult to implement in real time. Off-line implementations of the peak-picking process have been shown to perform a robust selection of the peaks on a variety of detection functions [Bello et al., 2005]. We review here some of these approaches to the peak-picking of onset detection function, and investigate their implementation in a real time context.

2.4.1 Post-processing

Some preparation can be done to limit the number of spurious peaks in the detection functions before searching for local maxima. Typical post-processing operations applied on the detection functions include low-pass filtering, DC-removal, and normalisation [Bello et al., 2005]. Low-pass filtering of the detection function aims at reducing the noisiness of the signal and minimise the spurious detections. The filter can be implemented efficiently and causally using a FIR filter:

$$\tilde{D}[n] = D[n] + \sum_{m=1}^{M} a_m D[n-m]. \tag{2.10}$$

This operation reduces the number of spurious peaks in the function with a minimal additional cost. Low-pass filtering is therefore well adapted for a real-time implementation. To avoid the delay implied by the low-pass filter, a window of the detection function around the current frame is filtered in both directions, simulating a zero-phase delay.

The DC-removal and normalisation stages bring the function into a fixed range, typically between 0 and 1. These steps ensure the function has a given profile regardless of the amplitude and nature of the sound, thus improving the success of the thresholding operation across a collection of samples. Off-line, the normalisation and DC-removal processes use information from a large time segment both before and after the current frame, allowing the use of fixed parameters for thresholding. In real-time, we can approximate this by using a long sliding window, which would increase significantly the delay of the system. Therefore, DC-removal and normalisation are not suitable to be implemented using very short delays, and not adapted for real time operations.

2.4.2 Dynamic thresholding

To obtain sequences of onsets, peaks in the post-processed detection function corresponding to actual onset times should be identified, yet avoiding spurious peaks. Important amplitude variations can be observed in the detection functions, depending on the content of the signal, and in particular the loudness, as can be seen in Figure 2.3 when the timpani enters after 14 s. To compensate for pronounced amplitude changes in the function profile, *dynamic thresholding* is used: for each observation in the detection function, a threshold is computed based on a small number of past and future observations; the amplitude of the current observation is then compared to this threshold. Methods to construct a dynamic threshold include frame histogramming [Hainsworth and Macleod, 2003], in which the most likely amplitude of the detection function is determined by studying the population of observations around the current time. The moving median was shown to be a successful to reduce noise and limit the number of spurious peaks [Rabiner et al., 1975]. This approach was successfully applied on onset detection functions, smoothing out small peaks while sharpening peaks of larger amplitude [Bello et al., 2005]. Median filtering is also computationally efficient, since the median can be simply obtained by sorting an array – which costs significantly less than constructing a histogram. The dynamic threshold is computed using the value of the median over a short buffer around the current sample:

$$\delta_t[n] = \lambda \cdot \text{median}(D[n-a], \cdots D[n], \cdots D[n+b]) + \delta, \tag{2.11}$$

where the section $D[n-a], \cdots D[n], \cdots D[n+b]$ contains a spectral frames before n and b after n. The scaling factor λ and the fine-tuning threshold δ are predefined parameters. Onsets are then selected at local maxima of $D[n] - \delta_t[n]$. The buffers used for this operation typically include about $a + b = 8$ frames taken before and after the current detection sample – less than 100 ms for a 44100 Hz sound and hop size of 512 samples.

2.4.3 Real-time peak-picking

To achieve a robust selection of relevant maxima within a short decision delay, we propose a modified approach that constructs a dynamic threshold based on a short window around the current location. The dynamic threshold $\delta_t[n]$ is designed to allow for the detection of peaks in normalised functions without DC-components. To compensate the absence of DC-removal and normalisation, an alternative thresholding operation is chosen. In this implementation, the dynamic thresholding favours both the median and the mean of a section of the detection function, centered around the candidate frame:

$$\begin{aligned} \tilde{\delta}_t[n] &= \lambda \cdot \text{median}(D[n-a], \cdots D[n], \cdots D[n+b]) \\ &+ \alpha \cdot \text{mean}(D[n-a], \cdots D[n], \cdots D[n+b]) \\ &+ \delta, \end{aligned} \tag{2.12}$$

where α is a positive weighting factor. The moving median filtering is used in a similar way as in the off-line implementation, except shorter buffer are used. The value of b in Eq. 2.12 is minimised in order to reduce the delay of the dynamic thresholding step. The introduction of the mean value attempts to replicate the effects of the normalisation and DC-removal processes, without the use of a long window, by using a dynamic value for the fine-tuning threshold. This step allows for the peak-picking process to cope with large dynamics changes found in music signals. Experimental results [Brossier et al., 2004b] have confirmed that, for small values of a and b, the modified threshold is robust to dynamic changes in the signal; the detection functions were peak-picked using a moving window of size $a = 5$ and $b = 1$ in Eq. 2.12.

This modified dynamic thresholding can be seen as a simple way to model the post-maskings and frequency maskings seen in Section 1.1.2: a peak is selected in the function if its amplitude is found above the average amplitude of past observations. If large amplitudes are observed in several consecutive frames, only the first peak will be selected. Here we make the assumption that the system can determine whether or not the current frame is an onset, depending only on a short frame window in the past, and regardless future events in the audio signal.

After the onset detection function has been post-processed and a dynamic threshold has been computed, the peak-picking process is reduced to the selection of local maxima above the threshold. The detection of a local maximum implies the comparison of at least three consecutive observations, which requires the knowledge of one observation after the peak. Onset times are thus defined as any local maximum in the peak-picking detection function:

$$\hat{D}[n] = D[n] - \hat{\delta}_t[n], \tag{2.13}$$

with $D[n]$ one of the functions defined in Section 2.3 and $\delta[n]$ defined in Eq. 2.12. To reduce the delay of the peak selection, yet minimising the impact on the detection of smooth onsets, we select all positive peaks defined by three consecutive spectral frames and found above the dynamic threshold.

2.4.4 Silence Gate and pre-masking

Informal listening tests have shown that a high number of false detections were found on a vinyl recording, higher than on a CD recording of the same piece, where the level of background noise is less prominent [Brossier et al., 2004b]. Amplitude variations in areas of low energy may not be perceived as onsets, yet observed as peaks in the detection functions. To reject spurious detections in areas of low energy, a simple envelope detector is built by measuring the mean energy of a window of the signal. The envelope detector acts as a silence gate, which prevents several spurious detections in background and quantisation noise, where onsets are more likely to be produced by background noise. Moreover, a measurement of the signal loudness is useful to detect offset times: a frame with a mean energy below a given threshold following a frame with a mean energy above this threshold indicates an offset. The threshold of the silence gate should be chosen to avoid spurious detections, not only between the songs but also during the short silence periods within some songs. By using the silence gate to discard onsets detected in low-energy regions, significant improvements on the detection accuracy could be achieved – to be later discussed in Section 2.5.3.

Because dynamic thresholding uses almost only past information, we have no means to detect when a peak in the detection function will be shortly followed by another larger peak. The system is thus prone to cause doubled detection. Using a minimum inter-onset interval, we can ensure that two consecutive onsets will not be detected within less than this interval. The parameter for the minimum inter-onset interval controls the shortest time lag after which a new onset will be detected. Obviously, imposing a minimum inter-onset interval reduces the number of false positives, triggered for instance by amplitude or frequency modulation. However, the minimum inter onset interval value should be short enough to identify rapid successions of onsets. We have measured experimentally that using a time lag 20 ms to 30 ms was long enough to avoid several false positives, without affecting the overall precision.

2.4.5 System overview

For efficiency and flexibility, we have chosen to implement on the different detection methods defined in Section 2.3 and based on the phase-vocoder. Several onset detection functions have been implemented and integrated as a library of C routines, described in Section 6.3. In the rest this chapter, we will concentrate on the evaluation of these detection function with two aims: maximise the robustness of the peak picking and minimise the delay of the system.

Figure 2.4 gives an overview of the process we use for the extraction of onset times. The audio signal is first reduced to an onset detection function at a lower sampling rate. We then perform temporal peak-picking on the detection function to obtain a sequence of onset times. This sequence is combined with the output of a silence detector to produce the onset/offset pairs that define the boundaries of

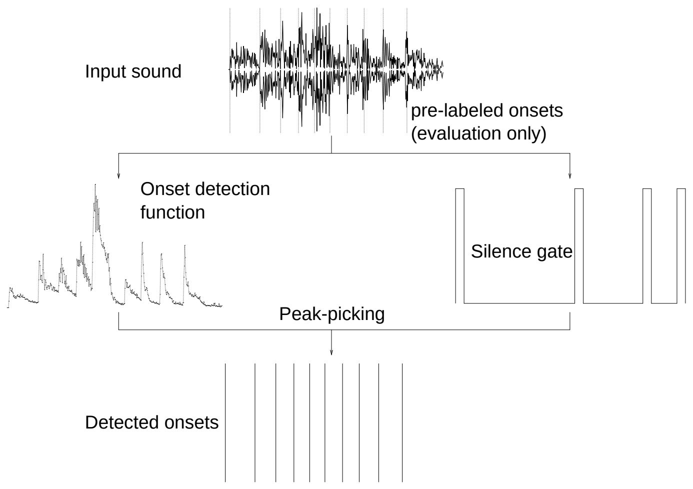


Figure 2.4: Overview of the segmentation process: the audio signal is reduced to an onset detection function at a lower sampling rate, and peaks are searched for in this function; a silence gate is used in parallel to prevent spurious detections in areas of low energy.

note objects.

Outlines of the post-processing and thresholding steps are shown in Figure 2.5 for both online and off-line implementation. In both case, low-pass filtering and moving median are used to remove noise and jitter and follow amplitude variations. In the off-line peak picking process, DC-removal and normalisation were used to obtain uniform detection function profiles across a collection of sound samples. Online, the moving mean aims at replacing these two steps.

After the processing of the phase vocoder and the onset detection function, the detected onset time is delayed of a few frames passed the actual attack time in the signal. The theoretical delay is of $(3 + b) \cdot$ hopsize/samplerate, where three frames are required to detect a peak, b for the dynamic thresholding step. For a sampling rate of 44100 Hz and a hop size of 256 samples, using $b = 1$ to compute the dynamic thresholding, the expected system delay is of 23.2 ms and can be further reduced by using shorter hop sizes.

Such a delay is acceptable for a perceptual attack time, and the onsets extracted in real time can be used to trigger audio or visual events without perceptible delay. For audio editing, cut-and-paste operations and other use of the annotated slices, the onset location must be more precise, down to the sample as much as possible.

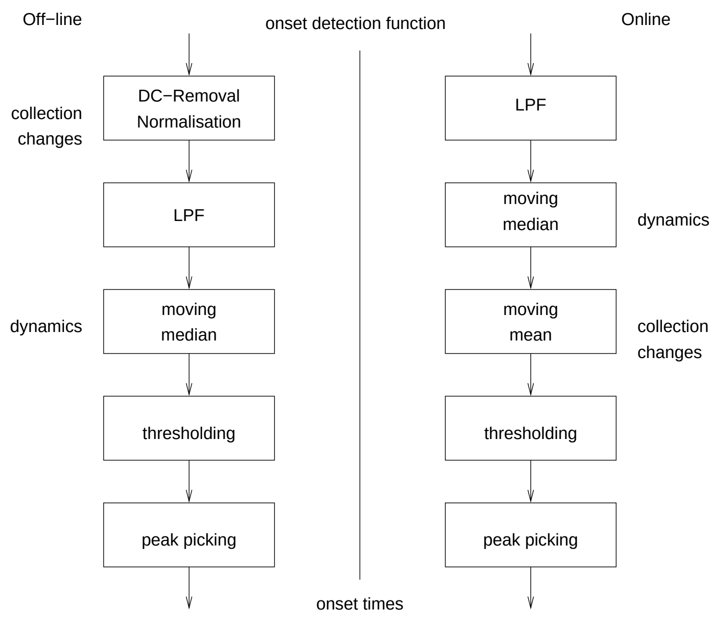


Figure 2.5: Comparison of off-line and online peak picking methods. Off-line, DC-removal and normalisation are used to cope with loudness variations across the database; the dynamic threshold modified for real time operation (Eq. 2.12) uses a moving mean to cope with loudness changes.

The different onset detection functions tend to peak at the maximum change in the attack and the peak is further delayed by the post-processing step. Appropriate removal of the system delay is required for a precise localisation of onset times. To reduce clicking and phase jumps artefacts that would be obtained by concatenating and looping the individual slices, the selection of a zero crossing point in the waveform is also preferable. From this local minima, ensuring that the attack of the next slice is preserved, we look for the closest zero crossing to select the best slicing location.

2.5 Evaluation

Evaluation of an onset extraction method a complex task and requires careful implementation and interpretation of the results. The aim here is to evaluate the ability of an algorithm to retrieve the onsets location as perceived by the human ear, which requires the gathering of hand-labelled sound files. The difference between manual annotations and automatically extracted onset times can then be evaluated by comparison and statistical analysis. With different instrument timbres and rhythmic patterns, the variety of the database is important to evaluate the behaviours of various onset detection methods against different types of signals.

2.5.1 Manual annotations

The gathering of an annotated database is a long and difficult task. This process typically consists in asking several listeners to manually label onset locations in a collection of sound files, using a standard sound editor (Section 6.1.2) or a tool designed for this purpose [Leveau et al., 2004]. The underlying idea is that each sound file of the evaluation database should be hand-labelled by different listeners. This *cross validation* process allows to minimise potential mistakes due to the annotation process. To reflect the perceptive results obtained for all listeners in the evaluation metrics, the evaluation of extracted onsets should be done against each manual annotation.

2.5.2 Precision and localisation

In order to quantify the success of the onset algorithms, the hand-labelled onset times are compared to the extracted times. A tolerance window is chosen in order to cope with the imprecision of the manual annotation process. A window of 50 ms on each side of the hand-labelled onsets is commonly used, a little larger than the average temporal discrimination lag to allow for imprecision in the annotations [Leveau et al., 2004, Bello et al., 2005]. Onsets found in this tolerance window will be considered as correct detections, whereas any other onsets will be considered as false alarms. Errors types for time labels can be separated into two categories: False Positives (FP) are false alarms, False Negatives (FN) are missed detections. Figure 2.6 represents the different cases that are to be considered by the evaluation algorithm. Points a and b in Figure 2.6 illustrate correctly detected onsets that were found within the tolerance window. Both detected onsets will be counted as correct detections (True Positives). Points c and d in Figure 2.6 gives and example of wrong detections. In both cases, one missed detection (True Negative) and one False Positive will be counted.

A refinement of this classification is to consider separately doubled detection, which helps in understanding some of the artefacts of detection functions and to highlight the various pitfalls encountered in the onset peak picking process. Point e

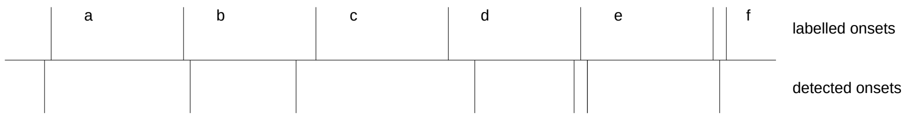


Figure 2.6: Different cases of good and bad detections. The horizontal axis is the time-line. The upper part represents the hand-labelled onsets, the lower part the detected onsets: a. correct detection, early but in the tolerance window; b. correct detection, late but in the tolerance window; c. early onset; d. late onset; e. double detection, both fall in the detection window; f. merged detection, two labelled onsets are within the tolerance window, only one detection occurred.

in Figure 2.6 represents a doubled detection. In this case, we will count one correct detection and one doubled detection. Point f in Figure 2.6 gives an example of merged detection, where we will count one missed detection and one correct detection. While double detections could be pruned by forcing the minimum distance between two detections to be equal or larger than the tolerance window, such a constraint would also reduce the number of correct detections found within the same tolerance window.

Each extracted onset must fall in one of these categories, so that after the list comparison has been done, the following assertion must be verified:

$$O_{\text{orig}} - O_{\text{FN}} - O_{\text{FNm}} = O_{\text{exp}} - O_{\text{FP}} - O_{\text{FPd}} = O_{\text{TP}}, \tag{2.14}$$

where O_{orig} and O_{exp} are respectively the number of original hand-labelled onsets and the number of automatically extracted onsets, while O_{FNm} O_{FPd} are the number of merged and doubled detections. The list comparison can be implemented using a matrix of size $O_{\text{orig}} \times O_{\text{exp}}$ containing all possible distances from each hand-labelled onset to each extracted onset. However, the cost of computing this matrix is $O(NM)$, with $N = O_{\text{orig}}$ and $M = O_{\text{exp}}$, and can become rather high for long lists, for instance when several false alarms are found. The comparison of both lists can be implemented with a cost $O(N + M)$ by using two loops to scan through the lists, which saves both memory space and computation time.

The next step is to measure the ratio of the different categories to ensure that onsets are correctly detected and spurious detections are limited. Correct detection and false alarm rates are defined as follow:

$$\text{GD} = (O_{\text{orig}} - O_{\text{FN}} - O_{\text{FNm}})/O_{\text{orig}} \tag{2.15}$$

$$\text{FP} = (O_{\text{FP}} + O_{\text{FPd}})/O_{\text{orig}}. \tag{2.16}$$

A perfect score would be GD $= 1$ and FP $= 0$. Another way to quantify the success of the detection is to evaluate the *precision* and *recall* [Crochemore and Rytter, 1994]:

$$P = O_{\text{TP}}/O_{\text{exp}} \tag{2.17}$$

$$R = O_{\text{TP}}/O_{\text{orig}}, \tag{2.18}$$

The rate of good detection GD is identical to the recall R. The definition of precision is less strict than $1 - FP$, as the number of correct detection is compared to the total number of extracted onsets O_{exp}, rather than to the number of labels annotated by hand O_{orig}. In order to maximise the precision P and the recall R, the weighted harmonic mean of precision and recall was proposed by van Rijsbergen [1979]. This measure, referred to as F_1*-measure* in the information retrieval literature [Yang and Liu, 1999], is computed with:

$$F = \frac{2 \cdot P \cdot R}{P + R}, \tag{2.19}$$

which is proportional to the surface occupied in the precision/recall plane. The general formula for the harmonic mean is $F_N = (1 + N^2) \cdot P \cdot R/(N^2 \cdot P + R)$. Other useful F-measures include $F_{0.5}$, which doubles the weight of the precision, and F_2, where the recall weights twice as much as the precision. We use here the F_1-measure, and will refer to it as the F-measure. Although the F-measure is less indicative than the FP and GD rates, looking for its maximum value is useful to optimise the parameters of the system.

To evaluate the localisation of the onset and align the extracted onset locations to the annotated onsets, we can measure the time lapses found between the hand-labelled onset and the automatically extracted times, within the tolerance window. This is especially important in our case to evaluate the delay of the system. The average time lag, its standard deviation and the distribution of time differences we obtain provide important informations on the localisation of the extracted onsets.

2.5.3 Experimental results

With the given evaluation system, we can start improving the detection algorithms. To evaluate the robustness of our implementation, various experiments have been implemented to measure the performance of each function against different category of signals, the speed of the algorithm and their computational cost, and the influ-

Category	Files	Annotations	Labelled onset
solo drums	30	90	2931
solo bars and bells	4	12	324
solo brass	2	6	213
solo plucked strings	9	27	431
solo singing voice	5	15	229
solo sustained strings	6	18	710
solo winds	4	12	266
poly pitched	10	30	859
complex	15	75	3563
Total	85	289	9526

Table 2.1: Details of the sound sample database database used for the evaluation of onset detection algorithms [MIREX, 2005b]. Each file is annotated by 3 or 5 different listeners. The total number of annotation is shown in the right column.

ence of the online peak picking method to compare with the off line peak picking algorithm.

Off-line implementations of the detection functions have proven to give good results on a variety of CD recordings, including percussive, purely harmonic signals and complex mixtures – pop and jazz recordings [Bello et al., 2005]. We now want to evaluate the effect of our modified peak picking algorithm on the overall performance results, as well as the precision of each detection function on the different categories of signal.

Evaluation database

The database we used was gathered for the Audio Onset Extraction contest of the 2005 Music Information Retrieval Evaluation eXchange [MIREX, 2005a] and consists of 85 sound samples recorded at 44100 Hz on a single channel. Each file was annotated by at least three members of the Centre for Digital Music at Queen Mary University of London, with polyphonic recordings being annotated by five listeners. The collection of files contains a large variety of instruments, music styles and sound mixtures. For a total duration of about 15 minutes, a total of 9526 onsets have been hand-labelled by 15 different listeners. The files are sorted along in various categories, as described in Table 2.1: struck bars and bells, solo drums, solo brass (e.g. saxophone), singing voice, sustained strings (e.g. violin, cello), plucked strings (e.g. sitar, harpsichord), polyphonic pitched instruments (e.g. vibraphone, piano) and complex mixtures (e.g. pop, symphonic orchestra).

A smaller database was used to compare the results of an off-line implementation [Bello et al., 2005] against our real time peak picking implementation. This database consists of 1065 onsets and is divided in four broad categories: pitched non percussive (e.g. bowed strings), pitched percussive (e.g. piano), non-pitched percussive (e.g. drums) and complex mixtures.

Overview of onset detection functions

The proportion of good detections against false positives obtained on the smaller database is shown in Figure 2.7. Note that the complex-domain, phase-based and spectral difference approaches produce functions smoother than the HFC, as they operate on information from more than one frame. It can be seen that, in contrast to the off-line peak-picking implementation, the HFC outperforms the complex-domain onset detection. This is due to the effect using short lengths of n_m has on smooth detection functions.

By design, the HFC and MKL functions are well-suited for the detection of percussive onsets, and methods that take the phase in account, such as the complex-domain and spectral difference approaches are best suited for the detection of tonal, non-percussive onsets. By using the multiplication of the HFC and the complex domain functions, the overall results are significantly improved. This combination consistently returns the best results for the whole set, increasing the overall reliability of the segmentation, and supporting the prevailing view that the different detection functions complement each other. This result is not surprising if we consider that both functions gave the best overall results, and is further confirmed when looking at the localisation of each function.

In Figure 2.8, the different categories of errors are detailed, showing the effect of the threshold on the overall score: using a very low threshold, the number of correct detection is maximised, but several false positives are generated. When increasing the threshold, both false positives and correct detections rates decrease. The number of merged detection remains unchanged as it is dependent on the time resolution of the algorithm. Instead, the rate of doubled detections decreases when the threshold increase, which confirms the behaviour of the moving mean to be able of discarding smaller peaks in favour of the larger ones.

Figure 2.9 show the F-measure values obtained with the larger database. The curves tend to an optimum value for $\alpha = 0.4$. When α is smaller than 0.4, more correct detections may be selected, but several false alarms occur. Using a higher threshold value, from 0.4 to 0.9, the system avoids more false positives, but some

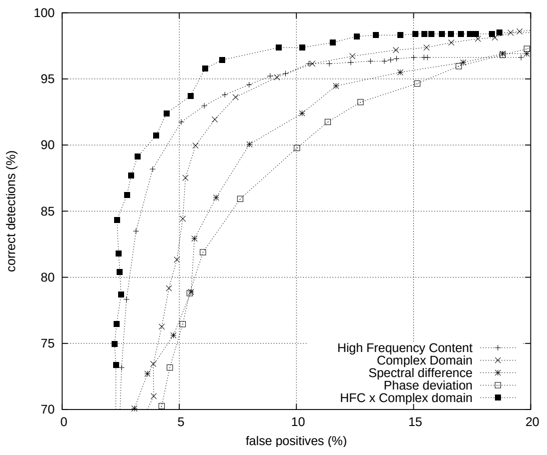


Figure 2.7: Correct detections against false positives using a silence gate and the real time peak picking method. Results obtained on the small database using values between 0.1 and 1.2 for α in Eq. 2.12 with different functions: high frequency content (Eq. 2.2), complex domain (Eq. 2.6), spectral difference (Eq. 2.3), phase deviation (Eq. 2.5) and multiplication of high frequency content and complex domain.

correct detections may be missed. Unsurprisingly, the temporal approach based on energy gave the worst results. Most of the functions, including complex domain, spectral difference, phase based and Kullback-Liebler, showed comparable results on the overall database.

The MKL function shows a very different behaviour from the others. This can be explained by the fact the MKL presents an important DC-component which as can be seen in the profiles of Figure 2.2 and Figure 2.3. The moving mean in Eq. 2.12 tends towards the DC-component, whereas for the other function the mean tends towards 0. Unlike the other functions, the F-measure for MKL decreases quickly for values of α above 0.2. However the function performs correctly for smaller values of the peak picking threshold. This behaviour can be avoided by using an appropriate value for the constant δ in Eq. 2.12 and was kept here to explain the behaviour of the peak picking. The multiplication of both MKL and HFC functions does not show the same behaviour, since the HFC function tends towards 0 when in regions of low energy.

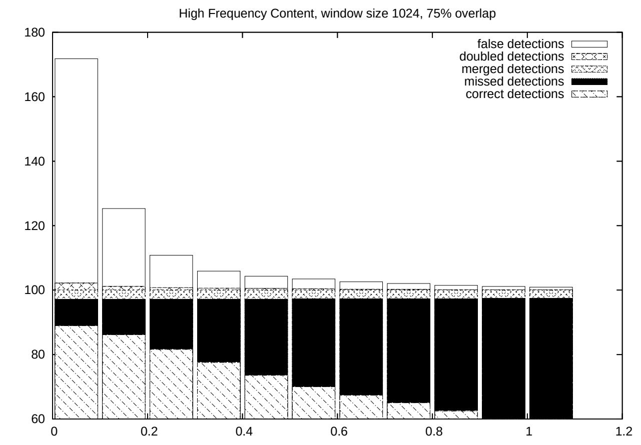


Figure 2.8: Categorisation of the errors: cumulative percentage of error types obtained with the High Frequency Content function (Eq. 2.2) for values of the threshold parameter from $\alpha = 0.1$ to $\alpha = 1$ in Eq. 2.12.

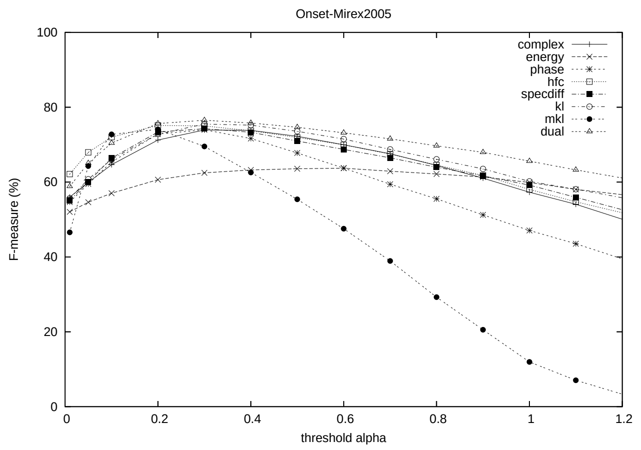


Figure 2.9: Effect of the threshold on the F-measure: results obtained on the MIREX 2005 database for different onset detection functions: complex domain (complex, Eq. 2.6), energy (Eq. 2.1), phase based (phase, Eq. 2.5), High Frequency Content (hfc, Eq. 2.2), spectral difference (specdiff, Eq. 2.3), Kullback Liebler (kl, Eq. 2.7), modified Kullback-Liebler (mkl, Eq. 2.9).

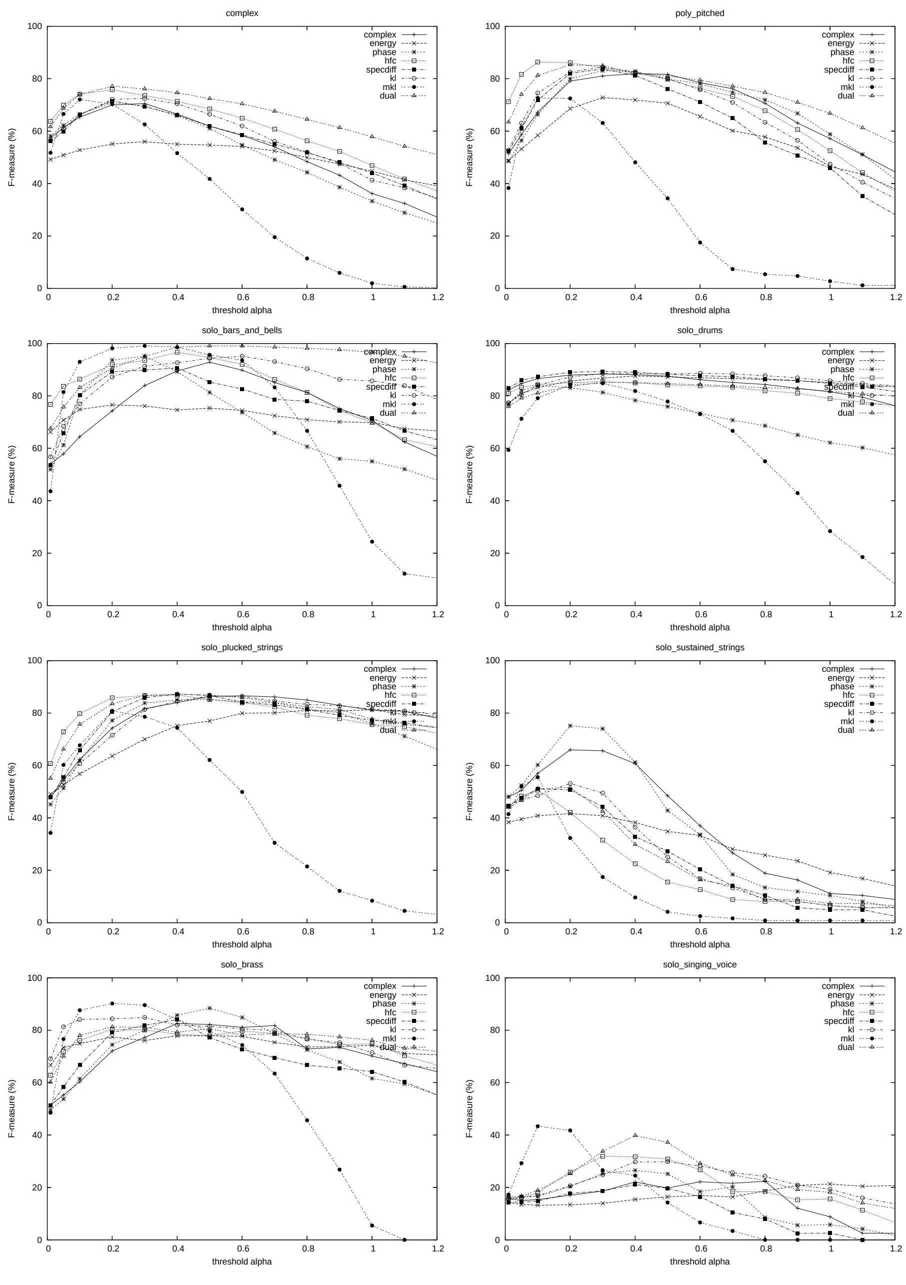


Figure 2.10: F measure against peak-picking threshold for different sound categories: a. complex, b. poly pitched, c. solo bars and bells, d. solo drums e. solo plucked string, f. solo sustained strings, g. solo brass, h. solo singing voice

Robustness across sound collections

In Figure 2.10, the results obtained on the large database are detailed for each sound category. Percussive sound samples with strong transients such as drums and struck bars gave best results. The best score was obtained for the complex domain function using a threshold $\alpha = 0.2$ in Eq. 2.12. 81.3% of the 2931 labelled percussive onsets were correctly detected with 2.86% of false positives. The detailed results also show that only a few samples of the collection account for a major contribution in the drop of the result. Most of the algorithms were unable to achieve usable results on singing voice (Figure 2.10 h). For string instruments, phase based and complex domain function outstands the results with F-measures of 0.75 and 0.67 respectively, which confirms that combining information from more than one frame provides more robustness on instruments with low transients.

An interesting behaviour can be observed for the dual function, the multiplication of HFC and MKL functions: the number of correct detections remains almost constant for values of α greater than 0.4. Using the dual function with a high threshold can thus maximise the number of correct detections while minimising the number of false positives.

Real time trade offs

For applications in a real time system, we wish to identify how fast we can detect onsets. The localisation curves in Figure 2.11 show the histogram of the time delays to hand-labelled onsets obtained for correct detections, for different window and hop sizes. The distribution are centred around values of 23 ms for hop sizes of 256 samples, which is consistent with the theoretical delay of the phase vocoder and the peak picking algorithm – 4 frames of 5.6 ms. For all the phase vocoder detection functions, the width of the distribution indicates the localisation is strongly limited by the window size and the delay of the system is only dependent on the hop size. An interesting behaviour is observed for the functions using a logarithm: the detection tends to occur earlier one frame earlier than the other functions.

Figure 2.12 shows the localisation histogram obtained when using a correction of four times the hop size. The delay correction successfully brings the distribution across 0, ensuring that most onsets are labelled earlier than the hand-labelled position. Again, for logarithm based functions the delay is closer to 3 frames, confirming that the functions peak one frame earlier than the other.

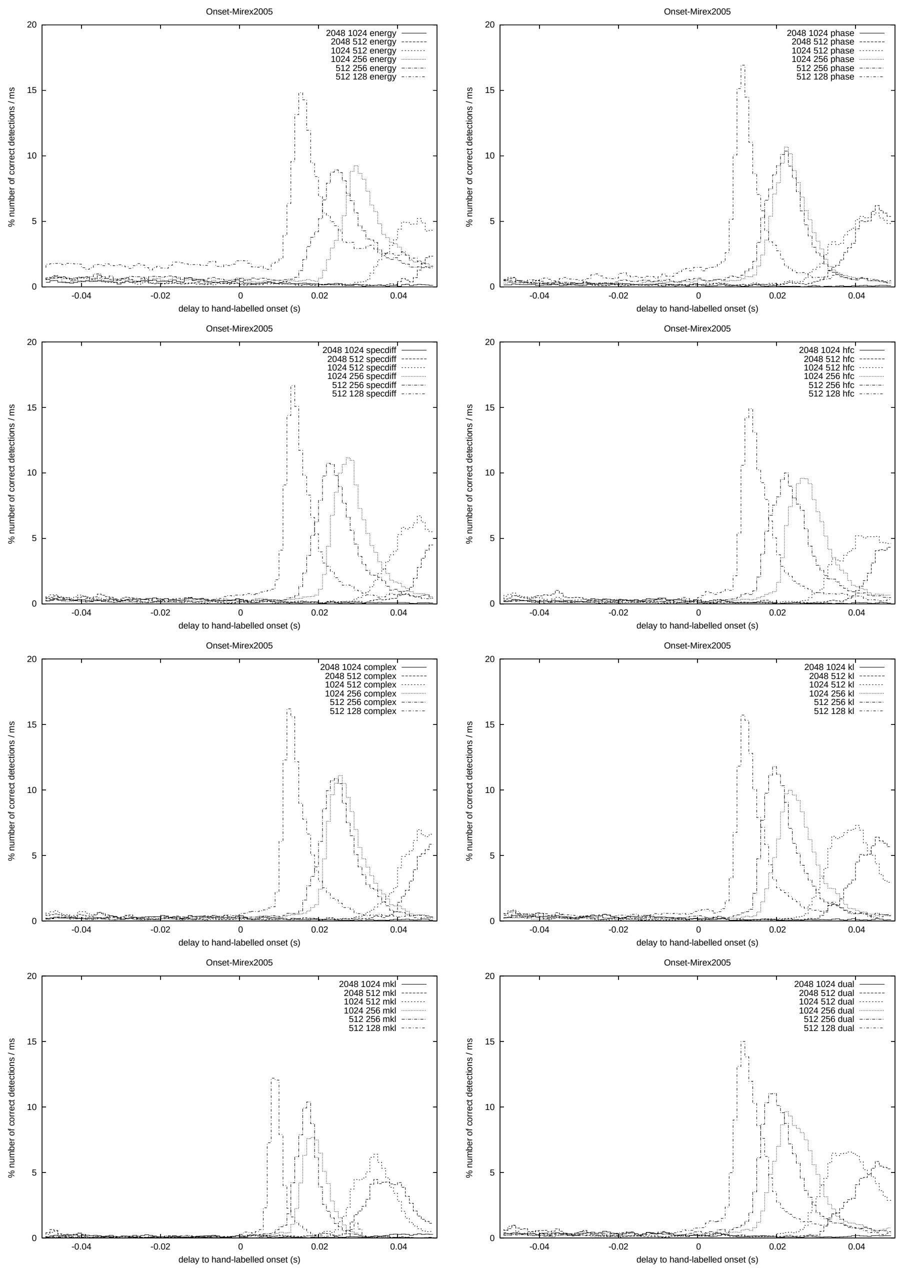


Figure 2.11: Localisation of the detections for different window and hop sizes: 2048/1024; 2048/512; 1024/512; 1024/256; 512/256; 512/128. Histogram of correct onset detections found for normalised over total number of annotated onsets: a. energy, b. phase, c. specdiff, d. hfc, e. complex, f. kl, g. mkl, h. dual

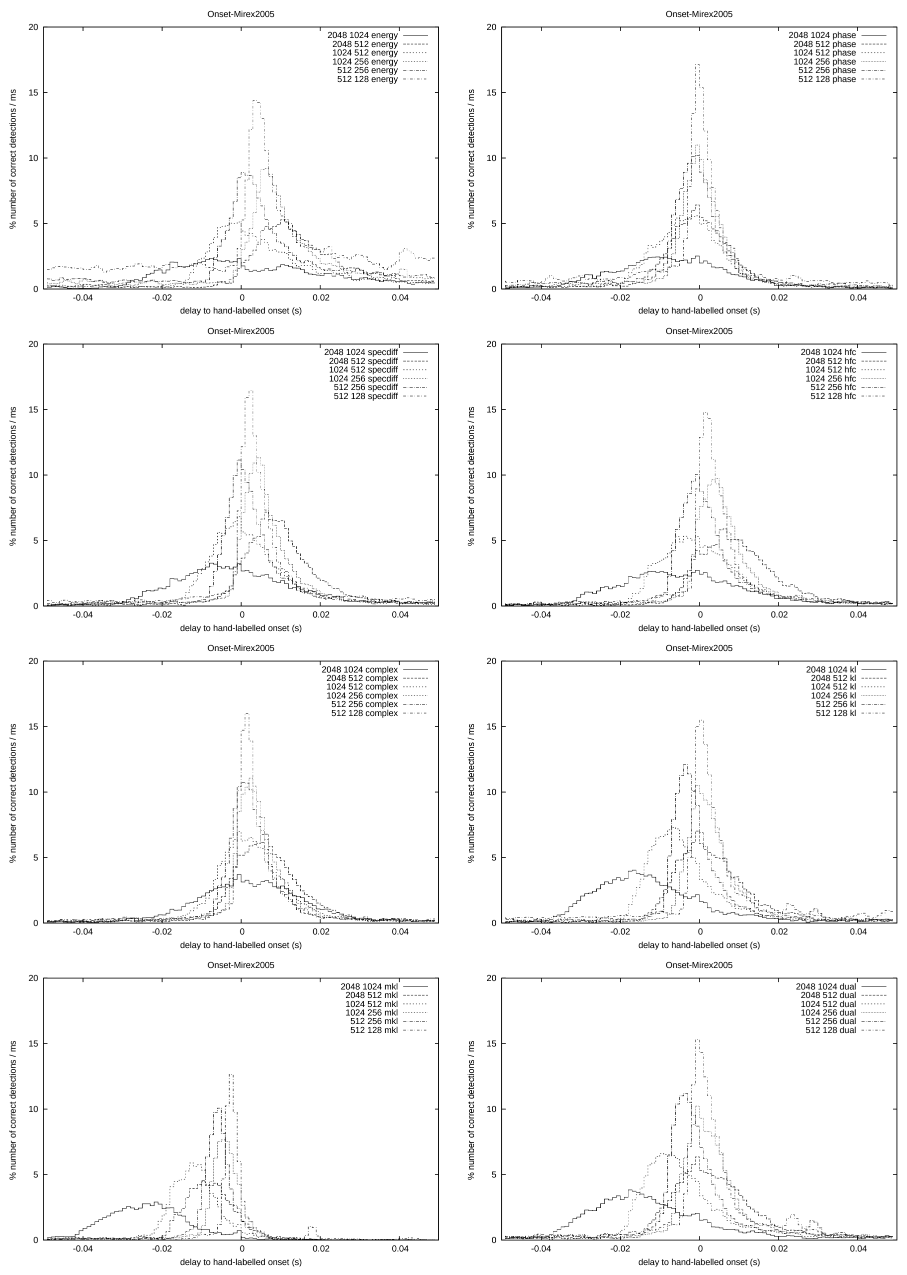


Figure 2.12: Localisation of the detections for different window and hop sizes: 2048/1024; 2048/512; 1024/512; 1024/256; 512/256; 512/128. Fixed delay of 4xhopsize. Histogram of correct onset detections found for normalised over total number of annotated onsets: a. energy, b. phase, c. specdiff, d. hfc, e. complex, f. kl, g. mkl, h. dual

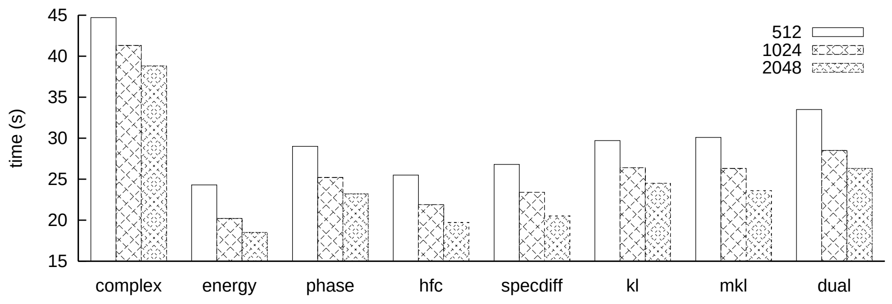


Figure 2.13: Computation times in seconds for different onset algorithms on the Mirex 2005 database (Table 2.1, approx. 23 min) at window sizes 512, 1024 and 2048, with 50% overlap. Tests were run on an Apple iBook G4 1.0 GHz running Debian GNU/Linux Etch.

Computational costs

Figure 2.13 shows a comparison of the time required to compute each of the detection functions. The benchmarks were run on an Apple iBook G4 1.0 GHz running Debian GNU/Linux Etch. Further results against other implementation have been provided by the MIREX contest and are recalled in Table 2.2 – these times were measured on different machines. Best results were obtained using Bayesian network (Lacoste & Eck 2) to select the best function adapted for each signal and inferring the onset locations using a tempo tracker. Amongst all the methods based on the phase vocoder, our implementation (Brossier, P.) placed first, slightly after Klapuri's method using multiple frequency bands (Ricard, J.). Our implementation was the second fastest, with a total runtime of 50 seconds. This time, significantly slower than the 12 seconds obtained by the fastest algorithm (Collins, N.), can be explained by the fact that we used a Python script to call our C functions, rather than a C program, so that the Python runtime environment had to be loaded for every sound file analysed. These loading times can be avoided by running the entire benchmark within a single Python script (see also Chapter 6 for details).

Effect of the silence gate

The silence gate proved to reduce the overall number of false positives by about 2% in all functions, while having a minimal effect on the percentage of correct detections [Brossier et al., 2004b]. When reducing the silence threshold to allow the selection of onsets with low energy, the overall performance increased. Observing the details of the results, two effects are perceived. On one hand, about 0.5% more percussive onsets are labelled correctly: some perceivable onsets were discarded with the silence gate. On the other hand, the rate of false positives increased for pitched onsets. Hence, the threshold for the silence gate must be set so that no correct detections are discarded, which can be verified when using $\alpha = 0$. We found that using a gate at -70 dB discarded onsets found in background noise without reducing the number of correct detections. This value may need to be fine tuned depending on the level of noise present in the input signal.

Automatic parameter optimisation

Using either the F-measure or a maximum rate of false detection, the parameters can be fitted over a given subset of the database, providing a statistical analysis of the influence of each parameter. The various parameters are very important for the onset detection and the temporal peak picking. Amongst the most important are the size of the window and overlap ratio of the phase vocoder, the silence threshold, and finally the threshold value for the peak picking. Optimising these parameters can be very time consuming, and the automation of the search for optima is an important gain of time. The design of our software library, described in Chapter 6, allows for the easy implementation of various combinations and automatic search of best parameters values. The process to automate the optimisation of the parameters relies on a simple iterative hill-climbing process, stopping when the maximum F-measure value has been reached.

2.6 Summary

Simple but perceptually motivated modifications to existing peak picking algorithms were proposed, and experiments on large databases have shown that the impact of the peak picking algorithm is limited to a few problematic timbres. The causal implementation opens the way to new applications, with live resampling and on the fly construction of annotated segments. Moreover the fast and robust extraction of onset can significantly improve the speed of systems that require temporal segmentation.

We have presented a complete framework for the evaluation of the performance of these functions. Evaluation on large databases showed that various methods could achieve a precise extraction of the onset trains without tuning of any parameters. Using a single parameter, a perfect match can be obtained on more than 90%

of the sound examples. The evaluation framework has highlighted the benefit of computing simultaneously two different functions. As all functions use the same spectral frame, computing several detection function is computationally inexpensive. This dual mode is the one proposed as the default settings.

The evaluation framework could also be used to calibrate precisely the location of the onset. Logarithmic based onset detection functions showed the advantage of a shorter delay in the peak rises.

Algorithm	% F	% P	% R	GD	FP	FN	M	D	Dist	aDist	Run
Lacoste & Eck 2	80.07	79.27	83.70	26.82	6.05	4.85	0.65	0.17	0.00613	0.0115	4713
Lacoste & Eck 1	78.35	77.69	83.27	26.55	7.91	5.12	0.62	0.19	0.00572	0.0115	1022
Ricard, J.	74.80	81.36	73.70	23.97	5.18	7.70	0.63	0.01	0.00593	0.0138	154
Brossier, P.	74.72	74.07	81.95	25.81	10.71	5.86	0.62	0.15	-0.00384	0.0111	50
Röbel, A. 2	74.64	83.93	71.00	22.62	3.46	9.05	0.51	0.52	0.00380	0.0084	159
Collins, N.	72.10	87.96	68.26	21.27	2.13	10.40	0.52	0.12	-0.00120	0.0069	12
Röbel, A. 1	69.57	79.16	68.60	21.40	5.05	10.27	0.48	0.88	0.00525	0.0087	158
Klapuri et al.	58.92	60.01	61.62	19.41	15.08	12.25	0.73	0.18	-0.02209	0.0276	56
West, K.	48.77	48.50	56.29	18.46	24.05	13.21	0.46	0.00	-0.00499	0.0138	179

Table 2.2: Overview of results of the MIREX 2005 Audio Onset Detection Contest [MIREX, 2005b]: overall average F-measure (F), precision (P) and recall (R); average number of correct detection (GD), false positives (FP), false negatives (FN), merged (M) and doubled (D); mean distance (Dist) and absolute mean distance (aDist) to hand labelled onsets; average runtime per file (Run).

Algorithm	Thresh.	% F	% P	% R	GD	FP	FN	M	D	Dist	aDist
complex	0.4	74.1	81.1	68.2	22.5	5.22	10.4	0.78	1.14	0.00439	0.00906
energy	0.8	63.8	70.9	57.9	19.1	7.82	13.8	0.67	1.00	0.00875	0.01286
phase	0.3	74.4	79.0	70.2	23.1	6.13	9.79	0.81	1.52	0.00133	0.00872
hfc	0.3	76.6	80.7	72.9	24.0	5.72	8.91	0.98	2.01	0.00656	0.01131
specdiff	0.3	75.0	77.0	73.2	24.1	7.17	8.82	0.88	1.58	0.00643	0.01095
kl	0.4	73.1	79.4	67.7	22.3	5.78	10.6	0.86	1.27	0.00368	0.00850
mkl	0.2	67.7	75.7	61.3	20.2	6.49	12.7	1.24	3.95	-0.00005	0.01107
dual	0.3	76.1	76.9	75.3	24.8	7.44	8.12	1.18	2.92	0.00355	0.01061

Table 2.3: Onset detection results obtained after training with our aubio real-time implementation on database MIREX 2005. The peak-picking threshold is indicated in the second column. Following column legends are identical to the ones in Table 2.2.

Chapter 3

Pitch analysis

The aim of a pitch detector is to determine the frequency perceived by the listener as the "height" of a sound, its *pitch*. Many sounds, including percussive ones, are perceived as having such a height. Some sounds, such as a crash cymbal or other percussive timbres, will instead not be perceived as pitched. Musical tones often present mixtures of pitched and unpitched sounds, which can be articulated in rapid variations. The goal of a pitch detection system is to identify the sounds forming a sensation of pitch, follow the frequency corresponding to this perceived height, and avoid unpitched sounds in the auditory scene.

Pitch perception models are essential for the analysis of harmony in music signals. They are used in different systems, such as music transcription and score following, music recognition and classification, melody modifications, time-stretching and other audio effects. A large number of methods has been proposed for the estimation of the fundamental frequency of speech signals, nowadays used in various applications, from speaker recognition to sound transformations [Rabiner, 1989, Gómez et al., 2003b]. Several approaches for the determination of the pitch of musical tones have been proposed in the past; reviews of pitch detection methods for music signals were given in [Roads, 1996, Klapuri, 2000, de Cheveigné, 2004]. Unlike speech, musical signals may have a very rich harmonic content and cover a wider range of the spectrum. The harmonic structure of different instruments will affect the reliability of different pitch models. Designing a robust pitch model is not trivial, even on monaural solo recordings as we will see in Section 3.4.3, and this task becomes increasingly difficult in the context of polyphonic music.

In this chapter, we describe different methods adapted to real time implementation. The methods selected are implemented as a collection of C routines for real time applications and their robustness is evaluated on monophonic recordings of several instruments. Their ability to extract the predominant melodic line from complex recordings is also tested. Different pitch perception models can be adapted to different applications, and the estimation of a pitch detection algorithm is a complex task, since each system can be evaluated along several criteria. Evaluation techniques for the performance of pitch detectors have been proposed, first for speech signals, and more recently for music signals. We give an overview of these techniques and discuss the results obtained with our implementation.

3.1 Introduction

The *fundamental frequency* f_0 of a periodic signal is the inverse of its period. The period may be defined as the "smallest positive member of the infinite set of time shifts leaving the signal invariant" [de Cheveigné and Kawahara, 2002]. For speech and music signals, which are not perfectly periodic, this definition must be applied to a local context around the analysis instant, within a limited set of time shifts. The *subjective pitch* refers to the auditory sensation of height. The fundamental frequency of a tone usually corresponds to the perceptual measure of its pitch, but there are exceptions. Periodic sounds may be outside the *existence region* of pitch, the frequency region in which a pitch may be evoked [Ritsma, 1962, Pressnitzer et al., 2001]. A sound may also not be periodic yet still evoke a pitch [Miller and Taylor, 1948, Yost, 1996].

Musical instruments are often harmonic and the different partials contribute to the sensation of pitch. Different instruments will have different harmonic structures. These structures change across the musical scale within each instrument timbre and evolve in time. Here we will denote the second partial of a harmonic spectrum as the first harmonic of this spectrum. The frequency of the n^{th} harmonic of a perfectly harmonic signal can be expressed as $f_n = (n+1)f_0$. However, most musical instruments are not perfectly harmonic. For vibrating strings, the frequency of the n^{th} harmonic can be modelled as a function of f_0 as follows:

$$f_n = (n+1)f_0\sqrt{1+Bn^2}, \tag{3.1}$$

where B is the *inharmonicity factor*, which varies according to the physical properties of both the string and the body of the instrument [Fletcher and Rossing, 1998]. Ideally harmonic signals are obtained when $B = 0$. Important inharmonicity factor are found on several music instruments, most notably the piano, where the higher

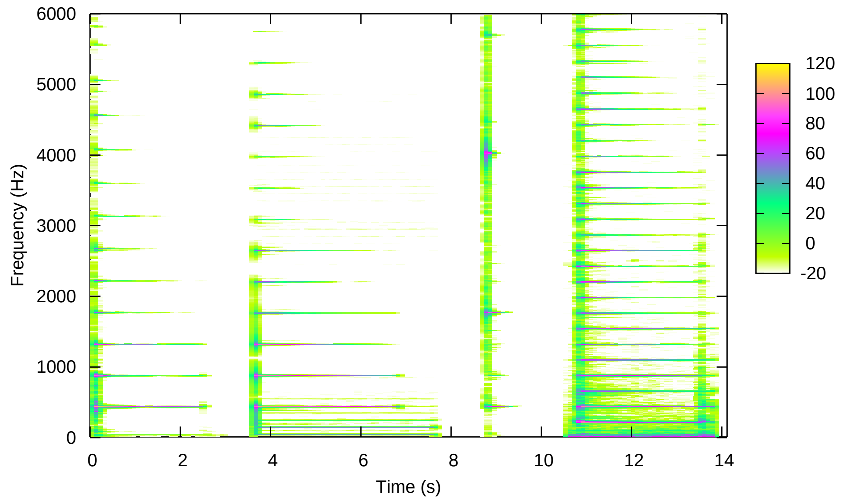


Figure 3.1: Spectrogram of a sound file containing four instruments playing the note A4 (440 Hz): a piano (0.3-3.1 s), a guitar (3.6-7.5 s), a vibraphone (8.6-9.7 s), and a harpsichord (10.7-13.8 s); analysis window 2048 points, 25% overlap; The strong transient components at the offset of the harpsichord correspond to the release of the key (13.5 s).

partials are consistently displaced towards the highest part of the spectrum.

The relation between the partials, their respective amplitude, participate in the sensation of timbre. The magnitude of each partial of an instrument is generally found to be lower than that of the fundamental, but this is not always the case. Sung voice for instance may present strong magnitudes at their first harmonic. Opera singers often sing with such a strong partial so that they can be heard over the orchestra. The clarinet favours the development of odd harmonics, so that partials at frequencies f_n will be found with a weak magnitude at even values of n. In all these cases, the perceived pitch remains the same.

The presence of percussive transients in the attack of musical sounds makes the determination of the period more complex. On sounds with sharp attacks, the search for a fundamental frequency in the short transients may cause spurious or missing estimates. Some instruments with long and breathy attacks may take more than 100 ms to settle on a steady state, and as they become longer, the transients will confuse pitch detectors, delaying considerably the decisions of the system [Fry, 1992]. Other transient sounds such as breathing, key clicking, bow scraping and other sounds from the instrument and the performer, are likely to complicate the fundamental frequency estimation. Within a voiced region, small variations may be observed in the period, for instance created by a glissando or a tremolo, and rapid articulation of notes will require a high temporal resolution. Pitch detectors should follow these rapid variations, yet avoid the spurious estimates caused by the transients.

Figure 3.1 shows the spectrogram of A4 notes (440 Hz) played consecutively on a piano, a guitar, a vibraphone, and a harpsichord. Each sound was extracted from the Musical Instrument Sounds RWC database [Goto, 2004]. The harmonic structure of the different timbres can be observed up to 6000 Hz. The harmonics of the piano and the guitar are distorted towards the highest part of the spectrum: the tenth harmonic of the piano is found around 5050 Hz for the piano, whereas the tenth harmonic of the guitar is around 4900 Hz – instead of 4840 Hz for an ideally harmonic sound. The difference between both timbres is due to different inharmonicity factors B in Eq. 3.1, larger for the piano string than for the nylon guitar string. The spectrogram of the vibraphone presents several differences with the piano and guitar timbres: a strong transient component can be observed during the attack, and the energy of the signal rapidly decays after the bar has been struck. Moreover, only a few of the harmonics are being developed – in particular the third (1750 Hz), eighth (4000 Hz) and twelfth (5750 Hz) harmonics – which is typical of struck bars instruments [Fletcher and Rossing, 1998, Fabre, 2001]. The harpsichord sound, at the right side of the figure, presents a strong partial one octave below the fundamental frequency, and twice as many harmonics as the piano or guitar. The harpsichord recorded in RWC database has two strings for each note: one is tuned to vibrate at the desired frequency and the other one to vibrate one octave below. Additional harmonics correspond to the lower of the two vibrating strings. Finally, the harpsichord sample was recorded with a significant background noise, which can be seen in Figure 3.1 after the silence separating the two notes and before the harpsichord attack, from 10.6 to 10.8 s.

Nuances and playing modes of different instruments might significantly alter the waveform of the signal and affect the performance of a pitch detection algorithm. Transitions from regions of low-level voiced music to background noise can be very subtle, and the estimation of the fundamental frequency will be more difficult when the signal presents a very low amplitude. The presence of background noise in the recordings, typically the additive noise due to air displacements close to a microphone, complicates the identification of the fundamental frequency. Reverberation and other room effects – early reflections, ambient noises – may cause several consecutive notes to overlap and alter the spectral components of the signal. Finally, the frequency range of an audio signal is broad: valid frequency estimates could be found from 20 Hz up to 5 kHz, and designing a method able to perform reliably on the whole spectrum is difficult.

A distinction is made between monophonic signals, where only one sound at a time occurs, and polyphonic signals, where various sources can produce multiple notes simultaneously, of various periods and possibly with various timbres. Tracking pitch on monophonic signals is not trivial, and the complexity of this task significantly increases when dealing with polyphonic sounds. We focus here on the analysis of pitch in monophonic music on a frame by frame basis. The choice of the pitch detection algorithms evaluated is driven towards their implementation in real time. The complexity of each algorithm and its computational load are considered, along with its robustness across the spectrum and on several instrument timbres. The ability of these algorithms to extract the predominant melody line from complex polyphonic recordings is also evaluated.

3.2 Pitch detection methods

Many of the pitch detection models used on music signals were derived from speech processing techniques [Rabiner et al., 1976, Wise et al., 1976]. Two tasks can be distinguished in speech pitch detection: identifying the voiced and unvoiced segments and estimating the fundamental frequency in the voiced segments. The use of a separate voiced-unvoiced decision is often required to avoid producing spurious detections in unvoiced segments, although a limited number of pitch algorithms can identify voiced and unvoiced segments by design. The estimation of the fundamental frequency can be subdivided into three steps [Hess, 1984]: pre-processing of the audio signal; extraction of the rough estimate; and post-processing for error correction and temporal smoothing of the pitch track.

Algorithms for fundamental frequency estimation are generally classified into two main categories: methods estimating periodicities in the waveform of the signal and methods which look for harmonic patterns in the spectrum. This separation between temporal and spectral approaches is not so clear, since some algorithms can be computed in both time and frequency domains. Spectral approaches tend to give a fine resolution in the highest part of the spectrum, but are limited in the low frequencies. As the period of the signal become shorter and closer to the sampling period, the accuracy of its estimation in the temporal domain becomes limited by the resolution of the digital signal, causing quantisation in the high frequencies. Trade-offs between time-domain methods for low frequencies and spectral methods for high frequencies were discussed in [Lyon and Dyer, 1986].

The time required to find a fundamental frequency estimate over a local context, the delay of a pitch detector, is directly related to the number of samples, the length of the signal windows, used to obtain the f_0 estimate. If this number can be kept small, and the algorithm to find the estimate does not take longer to calculate than these samples take to arrive, a real time implementation can be successful and yield short delays – typically 5 to 20 ms. Short windows are required to estimate rapid variations on short periods. The main limitation of the real time implementation of a pitch detection algorithm is thus its computational cost. Computing a 4096 Fourier transform every 5 ms is just about accessible for recent desktop computers, but yields a significant system load. Because pitch detection is meant to be used intensively in different applications, and eventually embedded in low resources devices, minimising their computation time is important. In this section we describe a selection of algorithms we implemented. Section 3.4 gives quantitative results obtained on different databases for each of these algorithms.

3.2.1 Preprocessing

To maximise the efficiency of an algorithm over a broad range of signals, the signals can be preprocessed. The aim of this step is to enhance the mid-range frequencies to reflect the perception of loudness by the human auditory system and to maximise the energy in the region of possible pitch candidates. A-weighting and C-weighting filters are designed to this effect, increasing the loudness of the frequency components in the 1 kHz to 5 kHz range, and decreasing the weight of low and high frequency components of the spectrum. To achieve this, time domain filters are efficient in real time, resulting in short delays and with a linear cost. However, designing such filters is complex and their computation in floating point requires the cascading of short filters to reduce the accumulation of errors [Schlichthärle, 2000]. In the spectral domain, a precise equalisation can be done across the spectrum using a weighting window, for instance to model the frequency response of the outer and middle ear. These steps are optional, but tend to increase the accuracy of the system and its robustness to the presence of noise. Moreover, they are in most cases computationally inexpensive. An interesting approach to this pre-processing step is to remove or reduce non-stationary components of the signal to keep only the sinusoidal components [Cano, 1998, Duxbury et al., 2001, Klapuri et al., 2001]. This approach is more complex but has been shown to improve the estimates of f_0 in transient attacks.

3.2.2 Spectral domain pitch detection

According to Klapuri [2000], two types of methods can be distinguished for modelling pitch in the spectral domain: *spectral place* methods rely on the localisation of the fundamental frequency by selecting spectral components according to their spectral location, and *spectral interval* methods rely on the estimation of distances between different partials of the sound. Temporal methods based on the ACF can be seen as spectral place approaches. An analogy can be made between these two spectral approaches and the *temporal theory* and *place theory* of pitch perception models [Moore, 1997, Chapter 5]: the temporal theory assumes the pitch is perceived as the least common period in different auditory channels, whereas place theory assumes that the perceived sound is compared to a bank of harmonic templates.

Musical timbres can have very different spectral patterns yet produce the same sensation of pitch. Different examples of spectral patterns perceived at the same pitch are shown in Figure 3.2. To detect the pitch of each of these patterns, de Cheveigné [2004] adopts the following reasoning. For pure tones (Figure 3.2 a), the position of the peak in the short term magnitude spectrum can be used as a cue to pitch. This approach fails for harmonic sounds where several peaks are present in the spectral frame (Figure 3.2 b). Selecting the largest peak will work in some cases, but fails on sounds with a first harmonic stronger than the fundamental frequency (Figure 3.2 c). Selecting the lowest frequency peak will identify the correct peak in all the above spectra, but will fail on sounds where the fundamental frequency is missing (Figure 3.2 d). A reasonable modification is to measure the interval between consecutive partials, which corresponds to the fundamental frequency of the tone. As the interval between partials varies according to the inharmonicity of the sound, spectral interval methods are more likely to be robust on inharmonic sounds than spectral place approaches. However this approach fails on sounds with missing partials (Figure 3.2 e), which brings to a final approach: for each partial in the spectral frame, sum the energy found in each subharmonic of this partial. The sum of the energy found for each bin are stored in a histogram. The histogram bin

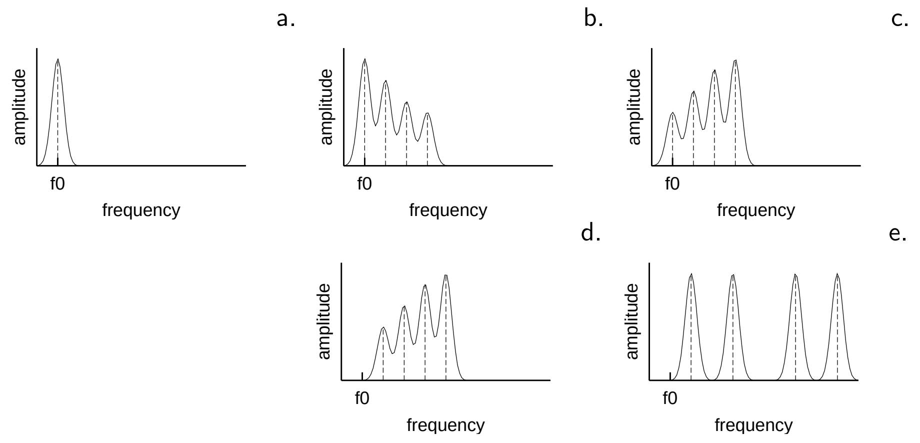


Figure 3.2: Different spectral patterns producing the same pitch cue: a. pure tone; b. harmonic tone; c. harmonic tone with strong partial components; d. missing fundamental frequency; e. inharmonic timbre; after [de Cheveigné, 2004]

found with the largest energy in the right-most part of the spectrum corresponds to the pitch cue. This technique illustrates a spectral pattern matching approach to model pitch perception. The multi comb pitch detection reviewed below can be seen as an implementation of this approach.

Spectral domain techniques are based on the Fourier transform, which can be optimally implemented. Long windows are required to obtain sufficient spectral resolution in the lower part of the spectrum. Longer windows implies a longer delay, a higher computational cost and less temporal precision in the high frequencies. Using zero-padding on the windowed signal before computing the Fourier transforms up-samples the spectral frame, so that finer analysis can be achieved in the lower part of the spectrum with no impact on the temporal resolution of the higher frequencies. However, when short delays and low computational cost are required, short time windows without zero padding are preferred to maximise the trade off between low frequency resolution and temporal resolution. For example, on a signal sampled at 44100 Hz, a window of length 4096 samples gives a resolution of 43.1 Hz per spectral bin, whereas the frequency interval of the entire lower octave of the piano corresponds to a range of less than 30 Hz: MIDI notes 20 (A0, 27.5 Hz) to 32 (A1, 55.0 Hz). To achieve sufficient precision in the low frequencies, the instantaneous frequency can be computed using the phase shift between two consecutive analysis frame. The precision of this computation depends on the overlap between frames, and at most half of the window should be used as the step size between two consecutive windows.

Fast comb spectral model

As a first approach to spectral pitch detection, we have derived an implementation of a simple pattern matching algorithm as found in [Lang, 2003]. The algorithm processes spectral frames as follows: the N peaks with the most energy are extracted from the spectral frame, and their magnitude and interpolated frequency are stored in an array. The predominant peak is then compared to the $N-1$ remaining peaks. If one of the $N-1$ peaks is found to be a subharmonic of the predominant peak within a tolerance of inharmonicity, and with a magnitude higher than half the one of the main peak, then this peak is selected as the new fundamental. The log of the magnitude is used for the peak comparison. The proportion of a half for magnitudes comparison is set empirically, as well as the inharmonicity criteria, written as follows: $n - 0.2 < f_{n_2}/f_{n_1} < n + 0.2$. The array of peaks is processed iteratively until no peaks are left in the spectral frame and each of them has been compared to their subharmonic. This approach allows the correct identification of the pitch cue for spectral patterns shown in Figure 3.2 (a) to (c).

Multi comb spectral filtering

Several implementations of spectral pattern matching using frame histogramming, similar to the one described by [de Cheveigné, 2004], have been proposed in the past. The following f_0 estimation algorithm is derived from [Lepain, 1999] and the improvements described in [Bello, 2003]. The method is based on the spectral frame $\mathbf{X}[n]$ of a phase vocoder, similar to the one used for the onset detection functions in Section 2.3. The input signal is first pre-processed through an A-weighting IIR filter to enhance medium frequencies and reduce the high and low parts of the spectrum. On each frame, the magnitude spectrum is low pass filtered in both directions and normalised to smooth out smaller peaks and minimise the effect of transient components. After pre-processing, peaks are detected in the spectral magnitudes frame and the list of peaks is passed to a harmonic comb. The assumption is made that one of the P strongest peaks corresponds to one of the partials of the present notes – for monophonic signals, we will limit to the case where $P = 1$. Each of these peaks generates a set of pitch hypotheses defined by the first Z subharmonics as:

$$f^0_{p,z} = \frac{f_p}{z} \text{ with } \begin{cases} 1 \leq p \leq P, & p \in \mathbb{N} \\ 1 \leq z \leq Z, & z \in \mathbb{N} \end{cases}, \tag{3.2}$$

where f_p is the frequency associated with the bin of the p^{th} peak, computed using a quadratic interpolation method. For each of these $f^0_{p,z}$ hypotheses, a harmonic grid is constructed over the spectral bins as:

$$C_{p,z}(k) = \begin{cases} 1 & \text{if } \exists\, m \text{ s. t. } \left| \frac{1}{m}\frac{1}{f^0_{p,z}} - \frac{1}{k}\frac{N}{f_s} \right| < \frac{\omega_b}{k} \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

where f_s is the sampling frequency, and m is an integer between 1 and M, the maximum number of harmonics considered. The tolerance ω_b is set to allow for some uncertainty in the harmonic match of the the comb filter and is typically set to a quarter of a tone. Different criteria are checked during the evaluation of each candidate comb. The two most important are the number of partials matching the comb harmonic grid, and the comb energy, estimated as the total energy carried by the set of partials.

3.2.3 Time-domain pitch detection

An intuitive method to detect the fundamental frequency of the signal is to observe the periodicity of the waveform. Time-domain techniques attempt to detect such periodic patterns directly in the waveform of the signal. One of the fastest ways of calculating the pitch is to count the number of zero-crossings within a given time frame, which requires a single exhaustive search for sign changes in a signal window. This method is reliable for very simple sounds such as pure sine tones, but fails on more complex tones. For instance, the rate of zero-crossings of a harmonic sound is often not related to the wavelength, as the waveform might change sign more than once within a period. The presence of noise or transient components is also likely to cause additional problems for the selection of relevant zero-crossings, either by increasing or decreasing the number of sign changes in a given window. An intuitive variation of this method is to count the number of peaks in a time frame, but faces similar limitations. If two peaks are present within one period, the estimation of the interval between peaks will fail. Generally, finding a reliable landmark in the waveform for a robust estimation of the period is difficult [de Cheveigné, 2004].

Schmitt trigger

A more successful approach than zero-crossing or peak counting methods is to implement a Schmitt trigger [Simpson, 1987, Sec. 10.19]. This is a special comparator circuit with two thresholds. When the input voltage becomes higher than the "upper" threshold, the output is a high voltage. When the input is instead smaller than the lower threshold, the output is a low voltage. The trigger acts as a memory, which describes a hysteresis cycle, and constitutes a period detector. In the case of musical audio, the input voltage is the audio signal. To cope with amplitude changes, the switching thresholds of the Schmitt trigger are modified to a proportion of the highest and lowest samples in the current buffer. The fundamental frequency candidate is directly given as the inverse of the rate at which the Schmitt trigger switches back and forth from one power rail to the other. The program can be written as a simple scan through each time window with a list of comparison and assignments. In our experiments, we have used the implementation proposed by Lang [2003] as a baseline for our evaluation. The simplicity of this model is also its drawback, and the complexity of musical timbres requires further improvements.

Autocorrelation

Correlation functions compare the similarity between two signals on a sample-by-sample basis. The autocorrelation function compares the signal with delayed versions of the same signal. Different versions of the ACF have been proposed. The *modified autocorrelation* of a discrete signal x_t may be defined as:

$$r_t(\tau) = \sum_{j=t+1}^{t+W} x_j x_{j+\tau}, \tag{3.4}$$

where $r_t(\tau)$ is the modified autocorrelation function of lag τ at time index t [Rabiner and Schafer, 1978]. With a periodic input signal, this function produces peaks at integer multiple of the period, as can be seen in Figure 3.3. A slightly different expression of Eq. 3.4 commonly referred to as the *autocorrelation* (ACF) in signal processing [Klapuri, 2000] is computed using:

$$r'_t(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau}. \tag{3.5}$$

The number of terms in the summation of Eq. 3.5 is reduced for long τ, which causes the autocorrelation to be tapered to zero towards long periods. The effect of the tapering is shown in Figure 3.3, where the normalised version of this function, $r'_t(\tau)/r'_t(0)$ is plotted.

For both functions, an exhaustive search of the maxima is done within a range of lags to find the peak at the shortest non-zero lag, corresponding to the pe-

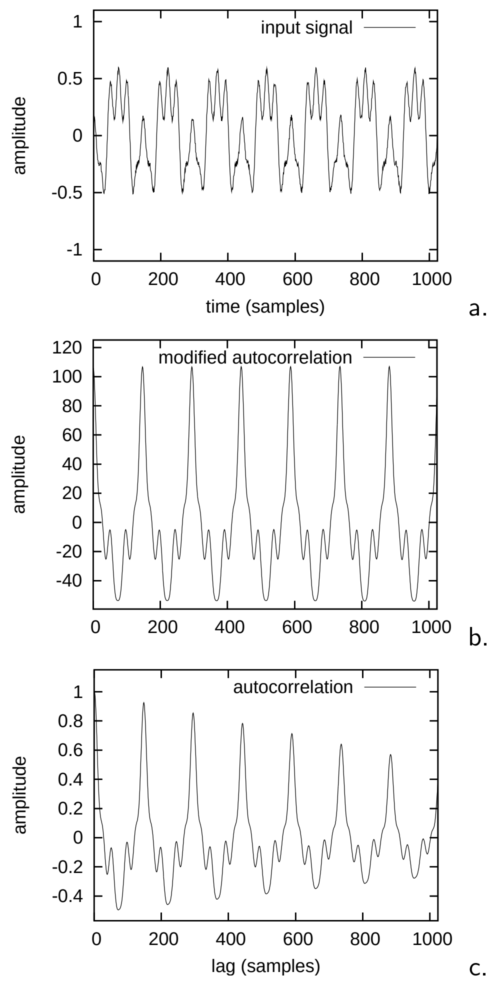


Figure 3.3: Example of modified ACF and ACF on a harmonic signal. a. 1000 samples of the input signal – 22.7 ms at 44.1 kHz. b. modified ACF according to Eq. 3.4. c. ACF according to Eq. 3.5 or Eq. 3.6. Because less terms are used in the summation for long lags, the envelope of the ACF is tapered to zero.

riod estimate. The modified autocorrelation method in Eq. 3.4 is prone to loss of relative precision when the period is small, whereas the tapered autocorrelation looses precision when the period becomes large [de Cheveigné and Kawahara, 2002]. Autocorrelation based methods are found to be efficient at detecting mid and low frequencies, and are commonly used in speech processing, where the range of possible frequencies is limited. As the spectrum is broader for music signals, the computational cost, $O(n^2)$, becomes significantly higher.

The computational cost of the ACF can be reduced by computing the autocorrelation in the spectral domain. The ACF (Eq. 3.5) can be expressed as the cosine transform of the square spectral magnitudes:

$$r'_t(\tau) = \sum_{k=0}^{N/2+1} |X_t[k]|^2 \cos\left(\frac{2\pi k\tau}{N}\right), \tag{3.6}$$

where $X_t[k]$ is the Fourier transform of a zero-padded window of the signal. This yields a computational cost $O(n\log(n))$, which is significantly lower compared to the temporal domain cost, $O(n^2)$. Expressed in this manner, the ACF can thus be seen as a *spectral place* approach, which selects the fundamental frequency by weighting the spectral components according to their spectral location [Klapuri, 2000]. Both approaches are prone to produce estimates at twice the period, since harmonic components of the fundamental frequency are given a positive weight, and less likely to produce errors in the higher octave, since in this case odd harmonics are given a negative weight [Klapuri, 2000]. ACF based approaches are generally found to be robust in the presence of noise, but are sensitive to formant and spectral peculiarities found in both speech and music signals [de Cheveigné and Kawahara, 2002].

YIN

The YIN algorithm [de Cheveigné and Kawahara, 2002] is a temporal pitch perception model which provides a simple yet fairly robust way to extract pitch candidates from a wide range of frequencies. The underlying assumption in this model is that $x_t - x_{t+\tau}$ is minimal when τ is the period of the signal. Let W be the size of the analysis window and $d_t(\tau)$ the square difference function at time t with delay τ, given by:

$$d_t(\tau) = \sum_{j=t+1}^{t+W} (x_j - x_{j+\tau})^2. \tag{3.7}$$

The YIN function is a normalisation of the square difference function, obtained by dividing the square difference found at a given lag by the average difference found over shorter lag values. The cumulative mean normalised difference function is expressed as a function of $d_t(\tau)$ as follows:

$$d'_t(\tau) = \begin{cases} 1, \text{ if } \tau = 0 \\ d_t(\tau) \Big/ \left[\frac{1}{\tau}\sum_{j=1}^{\tau} d_t(j)\right] \text{ otherwise.} \end{cases} \tag{3.8}$$

A minimum is then searched in d'_t for increasing values of τ and selected when found under a fixed threshold, typically set to 0.1. Figure 3.4 shows the square difference function $d_t(\tau)$, defined in Eq. 3.7, and the YIN function $d'_t(\tau)$, defined in Eq. 3.8, computed on 1024 samples of a saxophone recording. The minimum of the function, around 213 samples, corresponds to the period of the note being played by the saxophone. Minima at sub-multiples of the period, which highlight the presence of harmonics, are "lifted up" in $d'_t(\tau)$ by the cumulative normalisation in Eq. 3.8. Minima at integer multiples of the period are pronounced but discarded in favour of the first minimum found for the smallest value of τ. On the right hand-side plot of Figure 3.4, the window of samples contains the beginning of a note sung at the same frequency 207 Hz. The first minimum in $d'_t(\tau)$ is a correct estimate found above the tolerance threshold. This peak selection mechanism will prove useful to avoid doubling and halving errors. When no minimum is found, no period estimates are selected and the segment is marked as unvoiced. The value of the minimum can be used as a measure of confidence of the period estimates. Increasing the threshold value would correctly select the period estimates in some frames, but using too high values would lead to octave errors, the minimum of a harmonic of the signal being likely to be selected. Alternatively, if no valley is formed under the tolerance threshold, the minimum of $d'_t(\tau)$ can be used as the period estimate. This approach slightly reduces the overall voiced error rate but significantly increases the rate of non-voiced frames detected as voiced.

The longest period that can be selected by this model is half the length of the window of samples. For very short periods, the resolution of the period estimate is quantised at the sampling rate of the signal, and a quadratic interpolation around the minimum is used to refine the estimate. This interpolation is important for musical signals which may contain high fundamental frequencies. With appropriate interpolation, the algorithm is able to select arbitrarily high frequencies up to half the sampling rate.

Unlike the autocorrelation function, an advantage of the YIN function is that only the first part of the autocorrelation vector has to be computed: as soon as a minimum is detected under the threshold, the period value can be returned without

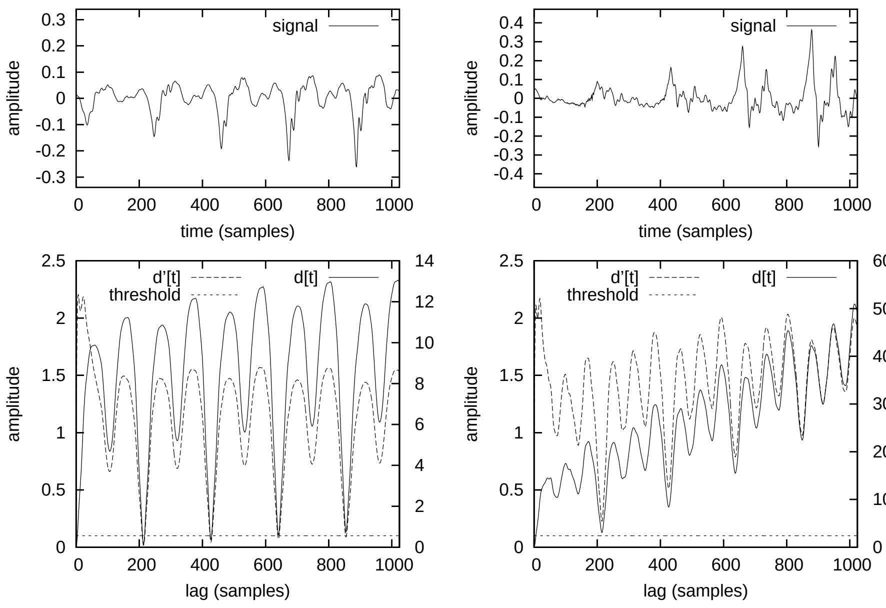


Figure 3.4: Example of YIN function on two music signals at 44100 Hz. Left: saxophone signal at 207 Hz (B3); Right: beginning of a voice signal (B3); In both case, a minimum is detected at 213 samples on $d'_t[t]$. On the saxophone signal, the period is found under the 0.1 threshold.

further computations. The latency of the system can thus be reduced on high frequencies to twice the length of the period. Moreover, this significantly reduces the cost of computing the entire function $d'_t(\tau)$, which is in $O(n^2)$. However, real time implementation is affected by this cost being strongly dependent on the fundamental frequency of the signal. To limit high computational costs on low frequency signals and silence regions, the search could be limited to shorter periods and frames containing enough energy, although this would prevent the detection of long periods.

3.2.4 Spectral domain YIN

We have designed a new pitch detection method based on the YIN algorithm. One approach to limit the cost of YIN is to compute the square difference function $d_t(\tau)$ in the spectral domain. The square difference function $d_t(\tau)$ can be written as a function of the autocorrelation:

$$d_t(\tau) = r_t(0) + r_{t+\tau}(0) - 2r_t(\tau). \tag{3.9}$$

To facilitate the selection of the minimum, we construct a tapered square difference function by replacing $r_t(\tau)$ in Eq. 3.9 with $r'_t(\tau)$ from Eq. 3.5. To minimise the delay, we assume constant energy over the lag τ: $r'_t(0) \approx r'_{t+\tau}(0)$, reducing the dependency of the computation to the current signal window. Using Eq. 3.6 in the above expression now leads to:

$$\hat{d}_t(\tau) = \frac{4}{N} \sum_{k=0}^{N/2+1} |X_t[k]|^2 \; - \frac{2}{N} \sum_{k=0}^{N/2+1} |X_t[k]|^2 \cos\left(\frac{2\pi k\tau}{N}\right), \tag{3.10}$$

which corresponds to a constant term – the sum of the squared spectral magnitudes – and the cosine transform of the squared magnitude taken at lag τ. This function can also be seen to be the square difference function between the spectral magnitude of the current window and a phase shifted version of the magnitude:

$$\hat{d}_t(\tau) = \frac{2}{N} \sum_{k=0}^{N/2+1} \left| \left(1 - e^{2j\pi k\tau/N}\right) X_t[k] \right|^2. \tag{3.11}$$

Based on the tapered function $r'_t(\tau)$ in Eq. 3.5, the function $\hat{d}_t(\tau)$ is also tapered towards long lags. This effect is shown in Figure 3.5, where both implementations of the square difference, Eq. 3.7 and Eq. 3.10, are plotted. Whereas the time domain version presents different minima at integer multiples of the period, in the spectral implementation, the lowest valley is formed at the period of the signal and other minima are found with a higher amplitude.

The normalised function $\hat{d}'_t(\tau)$ is computed similarly to the temporal domain implementation, using $\hat{d}_t(\tau)$ in place of $d_t(\tau)$ in Eq. 3.8. Comparative examples are shown in Figure 3.6. The upper plots in these two examples shows the two test signals, synthesised using several harmonics and small amount of white noise. Profiles obtained with the two methods, $d'_t(\tau)$ in Eq. 3.8 and $\hat{d}'_t(\tau)$, computed using Eq. 3.10, are plotted for each input signal in the lower plots of this figure. On the left, the signal contains a slowly rising harmonic sound. The same frequency is detected by both methods. In the temporal domain implementation, the threshold, marked by a horizontal line, had to be set to a ratio of 0.3 to select the correct minimum. For the spectral domain implementation, the minimum is found as the smallest value of $\hat{d}'_t(\tau)$ for all values of τ, displayed with the vertical line in the

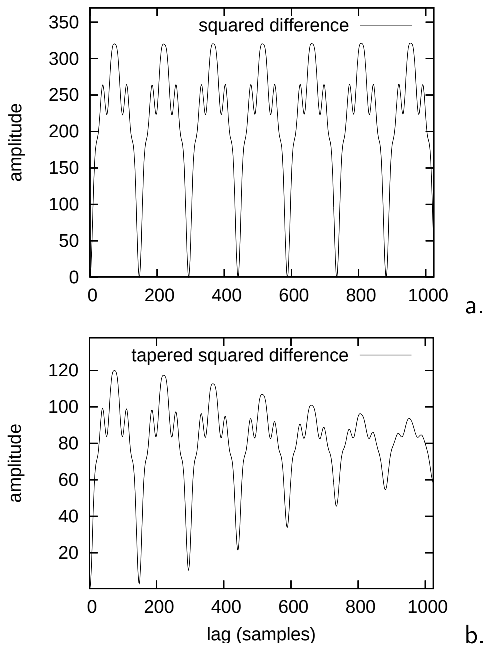


Figure 3.5: Example of square difference function and tapered square difference function obtained using the signal of Figure 3.3. a. square difference function according to Eq. 3.7. b. tapered square difference function according to Eq. 3.10.

figure. One of the success of YIN is due to its reduced number of parameters, limited to a single threshold parameter, which simplifies its implementation. In our modified implementation, the selection of the best period candidate is limited to the search for the minimum in $\hat{d}'_t(\tau)$, without depending on the threshold parameter.

To allow a better selection of period estimates within transition zones, no zero-padding is used to compute $X_t[k]$, the Fourier transform of the signal. The effect of zero-padding is to up-sample the representation of the signal in the spectral domain. Not using it smoothes out the valleys, increasing the resistance of the model to noise. The signal on the right side of Figure 3.6 simulates a transition between two notes, a low pitched sound and a higher tone. The second tone begins after about 800 samples. A strong transient component was simulated by the simplistic addition of white noise for a duration of 300 ms. The transient component has lifted up the function $d'_t(\tau)$ for all lags, and several minima appear, notably at the frequency of both tones found in this signal window. The selection of the smallest period becomes ambiguous. Moreover the valleys of the temporal domain function are affected by the transient component. The spectral domain implementation $\hat{d}'_t(\tau)$ instead contains deeper valleys. The selection of the minimum across all lags is less ambiguous. Moreover, the combined effect of the tapered square difference function and the absence of zero padding smooth out the valleys of the spectral domain implementation. The minimum is better defined, whereas small errors can be caused by the presence of high frequency components in $d'_t(\tau)$.

For very short periods, it may happen that the minimum of $\hat{d}'(\tau)$ is found at the lower octave of the signal period, because both valleys at τ_t and $2\tau_t$ are defined by a limited number of discrete lag values. Computing the value of the function at both interpolated minima locations will yield the selection of the correct minimum. The threshold can still be used to discard minima found above a fixed confidence, although this approach to the selection of the period reduces the success of the voiced/unvoiced decision.

This new difference function can be computed using two Fourier transforms, which reduces the cost across the whole spectrum to $O(n \log(n))$, significantly improving the computational load of the algorithm. The overall cost of the system also depends on the temporal resolution of the frequency estimates, and for applications demanding pitch tracks with very high resolution and containing high frequency components, it may be found more efficient to compute only the first few members of $d'_t[t]$ in the temporal domain. Finally, this implementation allows for a fine equalisation of different frequency components, for instance by applying a different weighting to each spectral magnitudes.

3.2.5 Post-processing

A reliable measure of the pitch period should give the exact beginnings and ends between consecutive segments of different pitches. Pitch detectors typically use fixed time frames from 5.6 ms to 50 ms. Spectral peculiarities, amplitude and frequency modulation are likely to cause spurious estimates, leading to pitch estimate changes, which will not be perceived as a change in pitch by the listener. The aim of the *post-processing* step is to reduce the number of spurious estimates. Meanwhile, the post-processing we introduce should not cause delay and preserve rapid variations and transitions between notes.

A common approach for smoothing the output of a system is its convolution with the impulse response of a low-pass filter. Low pass filtering was shown to be successful at removing jitter and noise [Hess, 1984], but does not remove gross errors and smears the transitions between notes and between voiced to unvoiced

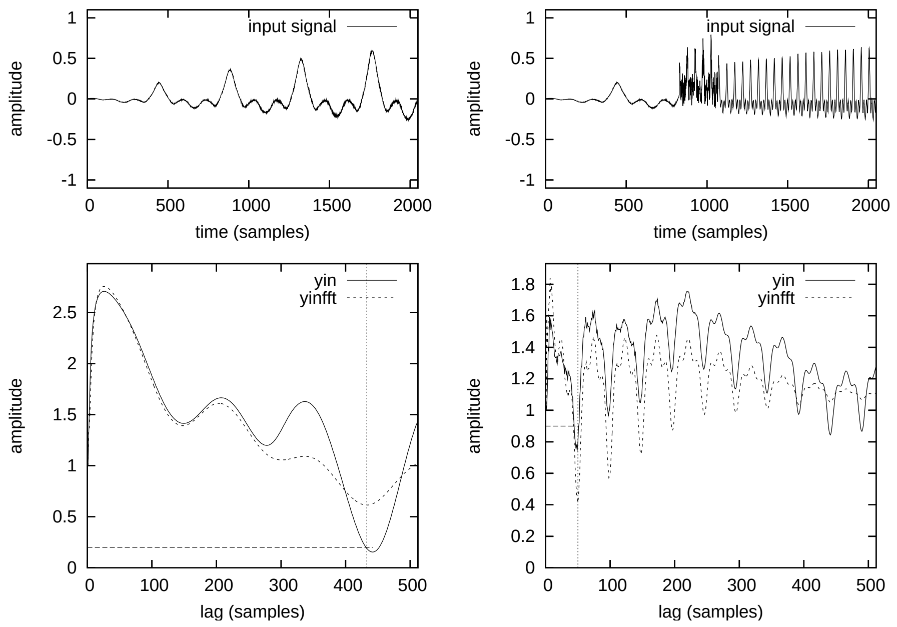


Figure 3.6: Example of YIN functions computed on two simulation signals at 44100 Hz using temporal (yin, $d_t(\tau)$ in Eq. 3.7) and spectral (yinfft, $\hat{d}_t(\tau)$ in Eq. 3.10) square difference functions in Eq. 3.8. Left: raising harmonic signal at 100 Hz; Right: transition from 100 Hz to 900 Hz.

segments. Non linear smoothing may thus be more appropriate. The use of median smoothing was proposed in [Rabiner et al., 1975], where a combination of linear smoothing, to remove jitter and noise, and median smoothing, to reduce short errors, is recommended. The moving median gives best results when using an odd number of observations, and smoothing over 3 to 5 estimates is generally considered long enough to reduce the impact of irregularities due to spurious detection and transients, yet short enough to reflect rapid glissandi and vibrato [Rabiner et al., 1975]. However, this implies an additional delay of 15 to 30 ms, depending on the size of the filtering and the rate of the analysis.

Other approaches that have been used to obtain smoother pitch track are based on the estimation of multiple candidates within a frame to find the best path across consecutive frames. In [Laroche, 1995], several estimates are stored for each frame, along with a confidence score. The optimal track can then be obtained by searching for the path with the best score. In [Maher and Beauchamp, 1994], a measure of the *mismatch* is computed between the measured partials of the estimated frame and

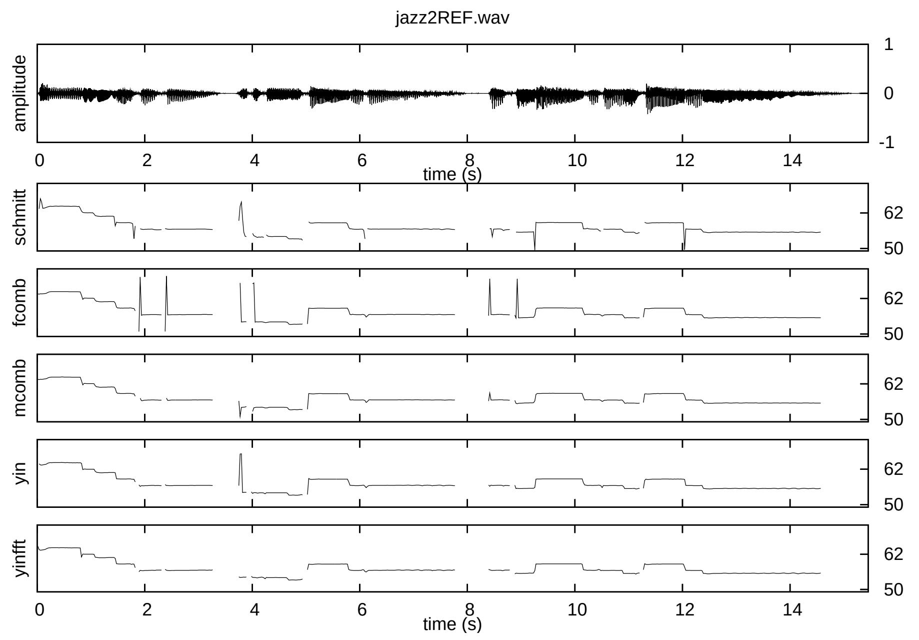


Figure 3.7: Examples of pitch tracks obtained for different methods on a saxophone signal. Top raw: original signal. Following rows from top: Schmitt trigger (schmitt), YIN (yin), spectral YIN (yinfft), multi comb with spectral smoothing (mcomb), fast spectral comb (fcomb).

the partials of a predicted frame. The set of partials resulting in the least mismatch between the predicted frame and the estimated partials is selected. This procedure helps avoiding octave errors and spurious detection. This two-way mismatch measurement is used in [Cano, 1998] as a confidence measure of the estimates, where past and future frames are combined to smooth the fundamental frequency function. Dynamic programming techniques such as hidden Markov models (HMM) can be used to find the best path across the estimates, as proposed in [Doval and Rodet, 1993].

The separation of the fundamental frequency estimation and the post-processing step is convenient for the implementation of the different modules, as well as for evaluation purposes. This allows us to estimate independently the frame-by-frame accuracy of the different pitch algorithms and the effect of the post processing. Further considerations on the post-processing of the pitch track are proposed in Chapter 5, where strategies to model notes are considered.

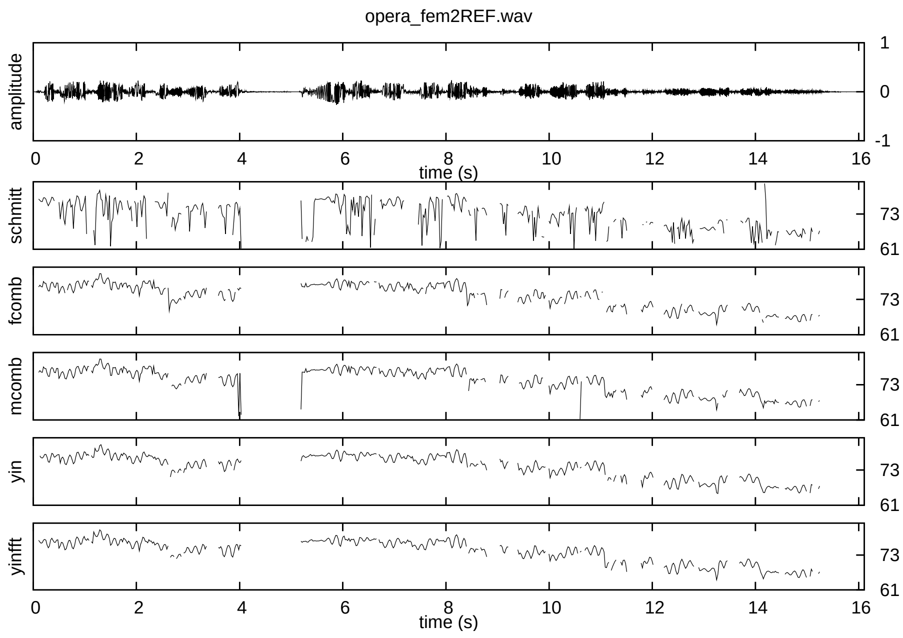


Figure 3.8: Examples of pitch tracks obtained for different methods on a soprano opera voice signal. Top raw: original signal; Following rows from top: Schmitt trigger (schmitt), YIN (yin), spectral YIN (yinfft), multi comb with spectral smoothing (mcomb), fast spectral comb (fcomb).

3.3 Multi-pitch and melody estimation

While various problems are encountered in the estimation of pitch on instruments playing solo, many instruments can play multiple notes at the same time, and several instruments can also play together. In addition to the difficulties of monophonic estimation, pitch period estimation of polyphonic signals presents a number of other problems as the signal complexity increases with polyphonic sources. Sources playing together may be perceived as a single coherent sound, or non-existent sound arises because of the combinations of multiple sources.

Simple polyphony may be successfully retrieved using monophonic detectors iteratively, which have proven to be useful for the detection of duets [Klapuri, 1999a]. However, a single monophonic pitch detector is generally not considered appropriate to perform multiple pitch detection. To estimate the pitches of different sources playing together, several approaches have been used. The comb filtering described in Section 3.2.2 was used iteratively in [Klapuri et al., 2002, Bello, 2003] to produce a set of candidates. Various approaches to the estimation of multiple fundamental frequencies have been proposed in the literature, including the use psychoacoustic models [Lyon and Dyer, 1986, Moore et al., 1997], or the integration of musicological rules to model the probability of transition between notes [Klapuri, 2003b]. Identification of each note of a single instrument was shown to be effective by explicit modelling of the notes [Kashino and Tanaka, 1993, Vincent and Plumbley, 2004]. In [Bello et al., 2002], a data base of isolated notes automatically learned from the signal was successfully used to maximise the robustness of the extraction of multiple fundamental frequency estimates. The design of multi-pitch estimation algorithms including musical knowledge were described in [Klapuri, 2003b, Bello and Pickens, 2005], with hidden Markov models (HMM) used to model the probability of the relationship between notes or chords across several segments. The complexity and the computational load of such systems makes their implementation in a real time context difficult.

Different methods have been proposed to extract the best melody lines amongst polyphonic pitch candidates [Klapuri, 2001, Paiva et al., 2004, Vincent and Plumbley, 2005]. In all these approaches, the assumption is that the melody line is the most salient note in the signal. A recent review of different strategies for the extraction of melody was proposed in [Gómez et al., 2003a]. In the scope of this research, we have limited our study to the robust extraction of pitch on isolated notes and monophonic recordings. Although the approaches we have selected for pitch detection are designed for monophonic signals, it is interesting to evaluate their robustness to follow the melodic line in polyphonic environments. Quantitative results for both monophonic and polyphonic signals are given in the next section.

3.4 Evaluation and experiments

To compare the different pitch detection methods we have described and implemented, we wish to evaluate their performance on a variety of music signals. Methodologies for the elaboration of audio signal databases and the evaluation of pitch extraction performance were proposed in [Rabiner et al., 1976], where a study of different pitch algorithm was conducted using a database of hand labelled voice signals. Unlike speech, for which the "ground-truth" estimates can be obtained using a laryngograph [de Cheveigné and Kawahara, 2002], a major difficulty in the evaluation of a pitch detection technique on musical signals is the construction of a database of annotated sounds. For monophonic signals, the ground truth can be created with the help of an existing pitch extractor and after manual corrections of spurious pitch candidates. Alternatively, when the score is available, the ground truth can be obtained with manual alignment of the score to the audio. For polyphonic recordings, access to the unmixed monophonic tracks would be convenient, although master recordings including the individual tracks are generally not provided by copyright holders. We have chosen to use three different databases: isolated notes, monophonic signals, and polyphonic recordings (see Section 3.4.2).

Another difficulty is the evaluation of the performance itself. Finding a good measure to pinpoint the various artefacts of a pitch detector is not trivial. For speech as for musical audio, the evaluation of pitch detectors requires the choice of meaningful criteria, and a criterion suitable for one type of application might not be suitable for all applications of a pitch detector. However, the characteristics of pitch detection algorithms influence the choice of a set of criteria: accuracy in estimating the pitch period, accuracy in making voiced-unvoiced decisions, and robustness across various signals characteristics – timbre, loudness, noise. These accuracies can be evaluated against many parameters and the importance of each evaluation criterion should be chosen according to the specific application targeted. For our real time implementations, the operating delay of the algorithms and their computational costs have also been evaluated.

Five pitch detectors were evaluated on our databases: fast spectral comb filter (fcomb, Section 3.2.2), multi-comb histogram with spectral smoothing (mcomb, Section 3.2.2), Schmitt trigger (schmitt, Section 3.2.3), time-domain YIN algorithm (yin, Section 3.2.3) and our modified spectral domain YIN (yinfft, Section 3.2.4). Each function accepts at least two parameters: the length of the buffer window and the overlap rate. In the following experiments, all other parameters, including thresholds, maximum number of peaks and number of partials, are fixed.

3.4.1 Performance metrics

The accuracy of each pitch detection method was measured against annotated data by counting the number of observations found correct within half a semitone. Reference values were time shifted and nearest neighbour interpolated to match the alignment and sampling rate of each method. A fixed time shift was used, computed as a function of the window and overlap sizes. To further refine this method, a corrective shift could be used to better align the ground truth estimate with the observation. Another approach for the comparison may include a search for the best match across a set of time shifts. In our case, using a fixed time shift was experimentally found appropriate when using the same buffer and overlap sizes for all evaluated methods. Besides voiced/unvoiced categorisation and gross error rate, pitch extraction algorithms will tend to make specific types of mistakes: octave errors are frequent; fifth, third and seventh intervals between the detected and actual pitches are also likely to happen. The following categories of errors were used to mark each time frame: *voicing detection*: errors on unvoiced segments correctly estimated; *raw pitch accuracy*: correct raw pitch estimates; *chroma pitch accuracy*: correct pitch estimates including octave errors.

When the attack of an instrument contains loud and long transient components, identifying a correct pitch can be difficult within the first frames of this attack, before the steady state has been reached. For real time applications, we are especially interested in knowing the speed of a pitch estimation algorithm, so we wish to evaluate the ability to recognise a correct pitch early in the attack of the sound. This is highly dependent on the timbre being analysed, and requires testing on a large database. The frequency range over which a pitch algorithm performs accurately has also to be measured. Another important matter is the perceptual accuracy of the algorithm, or how well the estimated pitch contour matches the one perceived by the listener. While full evaluation of this subjective criterion would imply the deployment of extensive perceptual testings, and the tests to be run by different listeners, our real time implementation of each of the pitch detectors facilitates informal tests to verify the perceptual accuracy of the detection on a large variety of signals.

3.4.2 Evaluation databases

Small databases have been labelled, either by hand or semi-automatically, by different research teams for their experiments [Rabiner et al., 1976, de Cheveigné and Kawahara, 2002]. Although large databases of music signals are available [Goto, 2004, Freesound, 2005], these databases have not been manually annotated and cross-validated. Large amounts of data could be obtained in different ways: running simulations on existing audio databases, recording or synthesising new material and finally annotate existing recordings. We have chosen to use the collection of isolated notes found in the Real World Computing database [Goto et al., 2003], and the databases gathered for the 2004 Audio Melody Extraction contest of the Music Information Retrieval Evalutation eXchange [MIREX, 2004b].

Instruments	Modes/Nuances	Low	High	Total
piano	13	21	108	1056
elecguitar	7	40	76	468
vibraphone	13	53	89	444
rhodes	5	28	100	292
clavinet	5	29	88	240
Total	43	21	108	2500

Table 3.1: Details of the instrument categories in the database of isolated notes. Each category contains several playing modes and nuances. The lower and higher values for each instruments are indicated in MIDI note number. The last column contains the total number of notes.

Isolated notes

To evaluate the robustness of the pitch algorithms against various instrument timbres, a first evaluation was made on recordings of isolated notes sampled at 44100 Hz taken from the RWC database [Goto, 2004]. The database comes with a large collection of instruments [Goto et al., 2003] recorded solo in different conditions: each note of their register is played in different modes – staccato, pizzicato, tremolo – using different methods – pedal, slapping, hard or soft mallets – and at different dynamics – *pp, p, mf, f, ff*. We selected sound samples containing several pianos, vibraphones, electric and acoustic guitars, clavinets and Rhodes. Problematic timbres such as the harpsichord or xylophone [Fletcher and Rossing, 1998] were avoided, but the choice of the instruments was made to represent a large range of timbres. The database is made of one single file per playing mode and instrument, containing individual notes played consecutively and separated with silence. The separation and labelling of each note was done automatically, using a simple silence detection. From the meta-data associated to each original file in the RWC, the generated files could be named after the RWC reference names and their MIDI note number. Table 3.1 shows the details of the database. The MIDI Numbers of some segments were found to have offset errors, either in the original files – 6 chords of the guitar played in a row – or due to over-segmentation within silences, and were corrected manually.

The ground truth of each note is assumed to be the frequency of the MIDI note at a constant value. This is a poor modelling of the actual instantaneous frequency, which follows variations due to the tuning of the instrument or the presence of vibrato. However what we know is the actual note being played, and identifying this note is precisely what we want to achieve.

Category	Files	Duration
pop	4	84 s
midi	4	80 s
daisy	4	75 s
opera	4	72 s
jazz	4	57 s
Total	20	368 s

Table 3.2: Duration of the sound files used for the evaluation of the pitch detection algorithms on monophonic and polyphonic signals. Both databases were obtained from the MIREX 2004 Melody Extraction contest [MIREX, 2004b]. Complete files listings are given Table A.3 and Table A.4.

Complex recordings

A database of 20 sound files, gathered for the 2004 MIREX Audio Melody Extraction contest [MIREX, 2004b], was used to evaluate the accuracy of our implementations on real recordings in monophonic and polyphonic conditions. The annotations were prepared from master recordings which included unmixed original tracks. The fundamental frequency estimation was done semi-automatically, with the help of a monophonic pitch detection method based on SMS [Cano, 1998]. Annotation of the monophonic track are used for the predominant melody of the polyphonic tracks. The set of files, described in Table 3.2 contains: 4 items consisting of a MIDI synthesised polyphonic sound with a predominant voice, 4 items of saxophone melodic phrases plus background music, 4 items generated using a singing voice synthesiser plus background music, 4 items of opera singing, two with a tenor male voice and two with a soprano woman voice, 4 items of pop music with singing voice.

3.4.3 Experiments

Several experiments were run to evaluate our real time implementations. Pitch detection algorithms have been implemented as a collection of C routines along with the code used for the onset detection methods described in Chapter 2. After verifying the tuning of the algorithm by listening to its output and testing on synthesised signals, frame by frame evaluation of the pitch tracks were done on different databases. A tolerance of half a semitone is used to cope with the detuning across the collection, but could be set to smaller to analyse instruments with non-Western and micro-tonal scales.

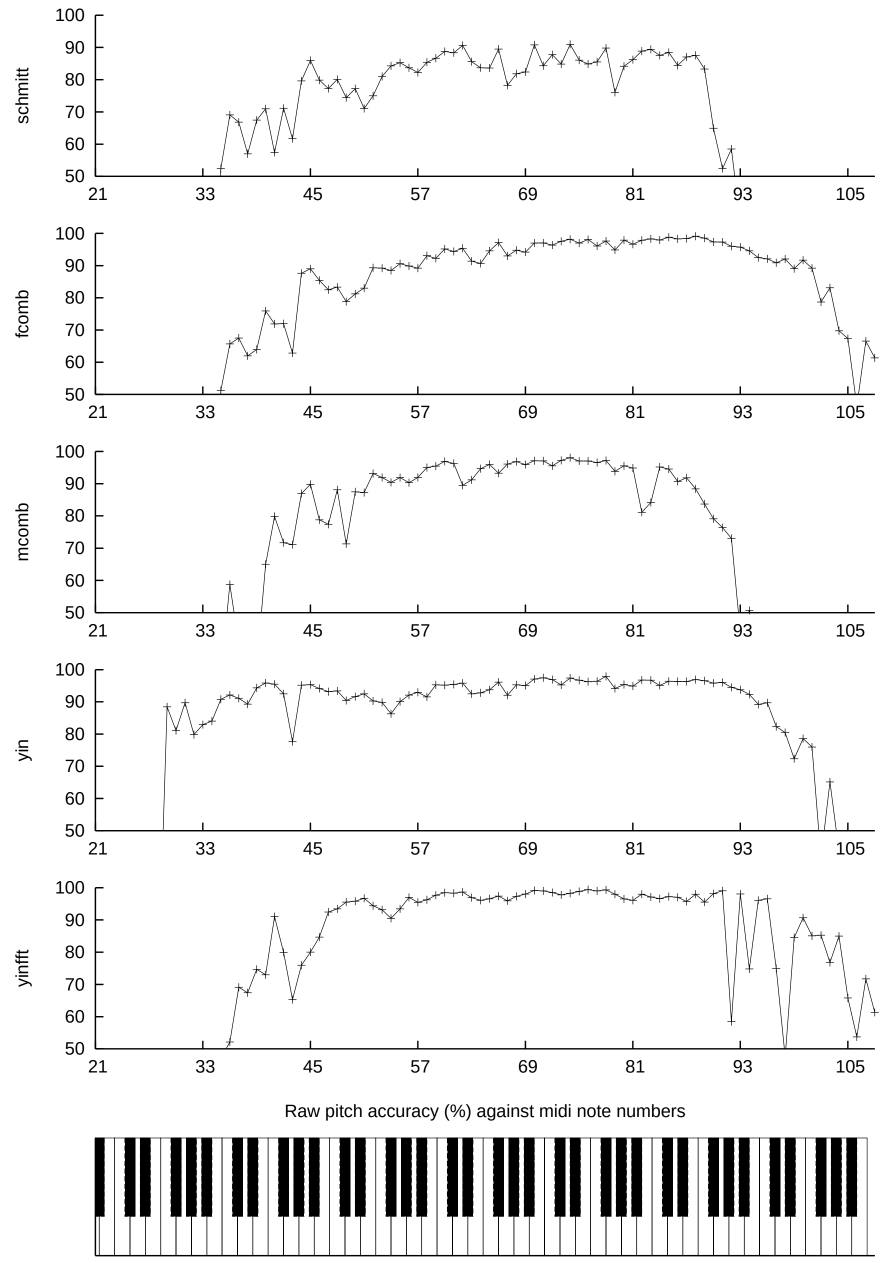


Figure 3.9: Percentage of raw pitch accuracy obtained on all 2500 isolated notes played on 5 different instrument timbres. Each plot corresponds to a different pitch detection method: Schmitt trigger (schmitt), fast spectral comb (fcomb), multi comb with spectral smoothing (mcomb), YIN (yin), spectral YIN (yinfft). See Table 3.1 for database details.

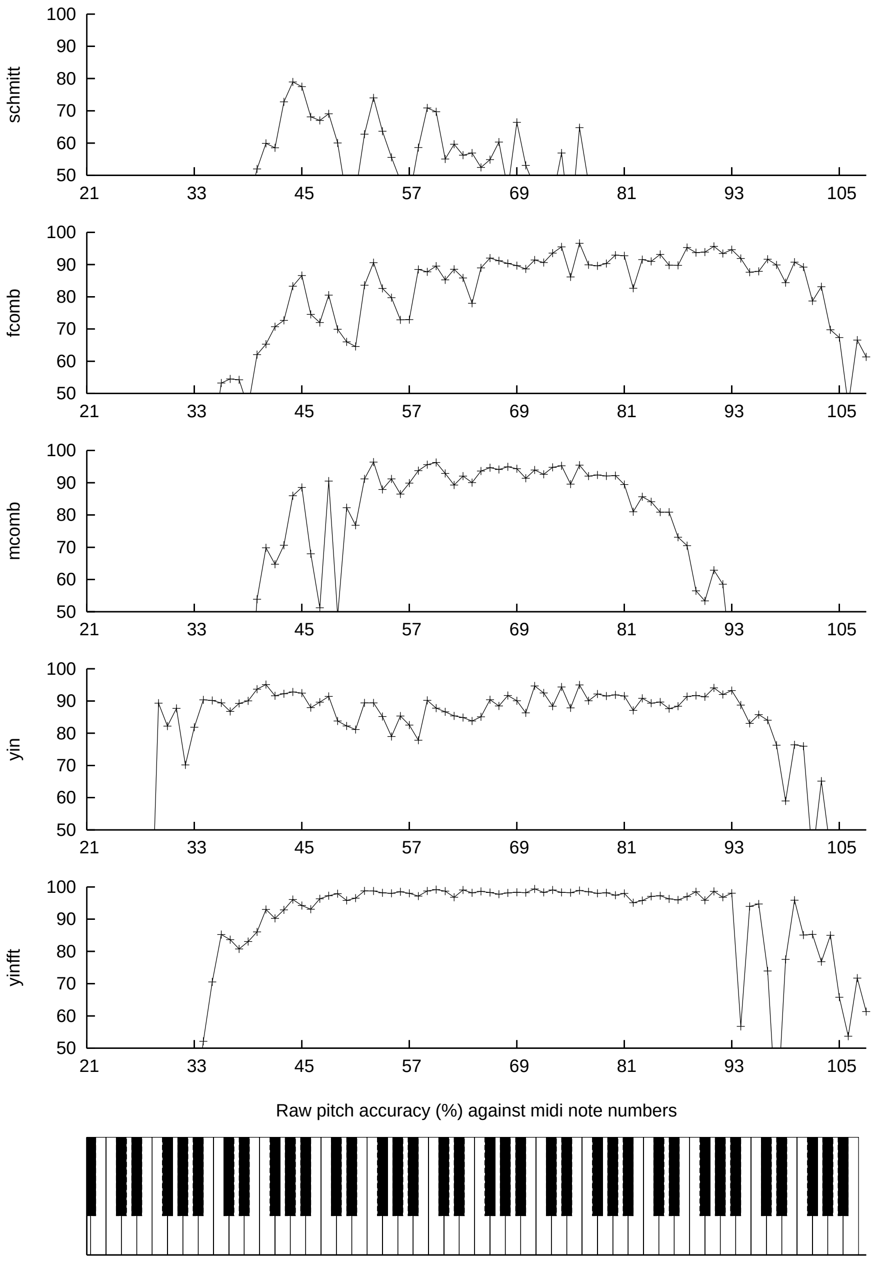


Figure 3.10: Percentage of raw pitch accuracy obtained on 1056 isolated piano notes played in 13 different modes. See Figure 3.9 for complete description.

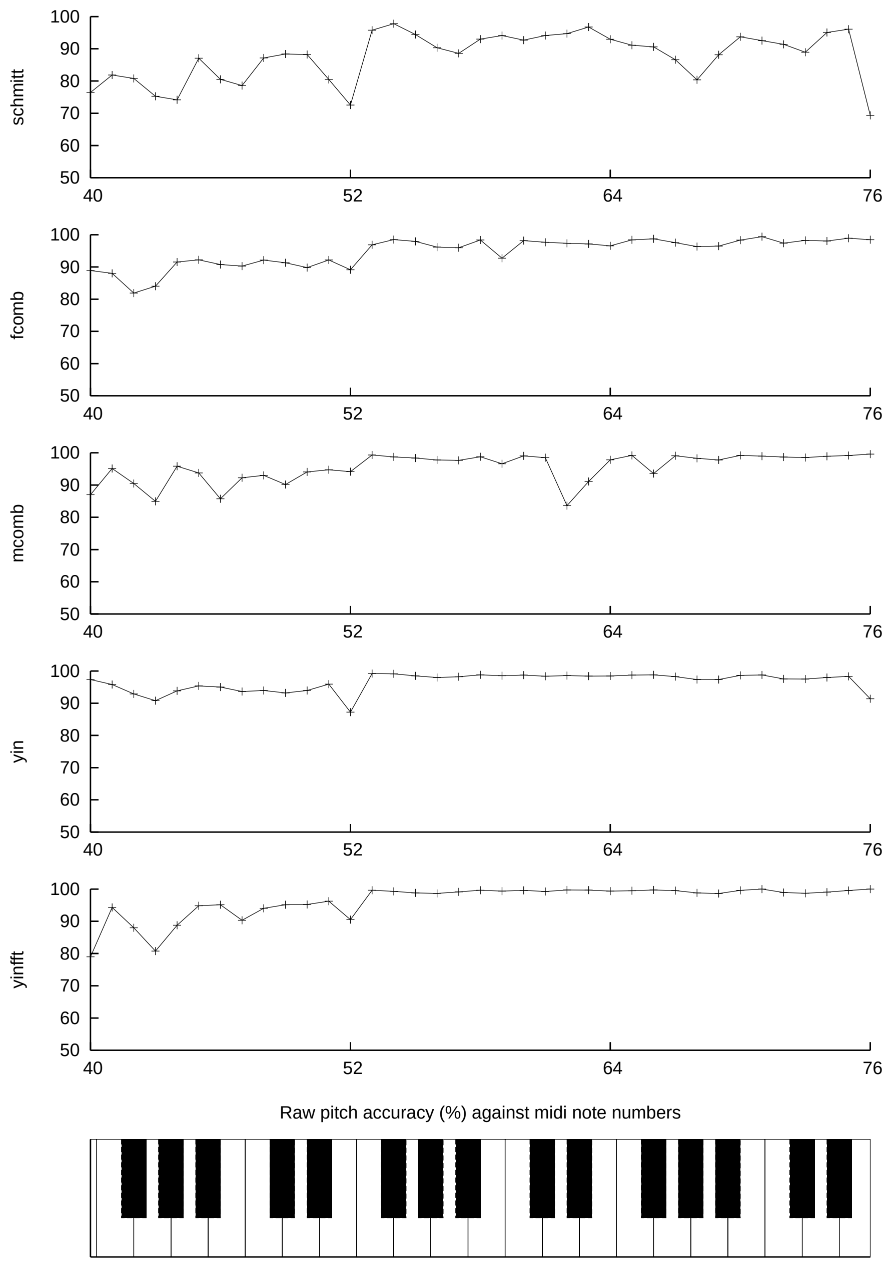


Figure 3.11: Percentage of raw pitch accuracy obtained on 468 isolated electric guitar notes played in 7 different modes. See Figure 3.9 for complete description.

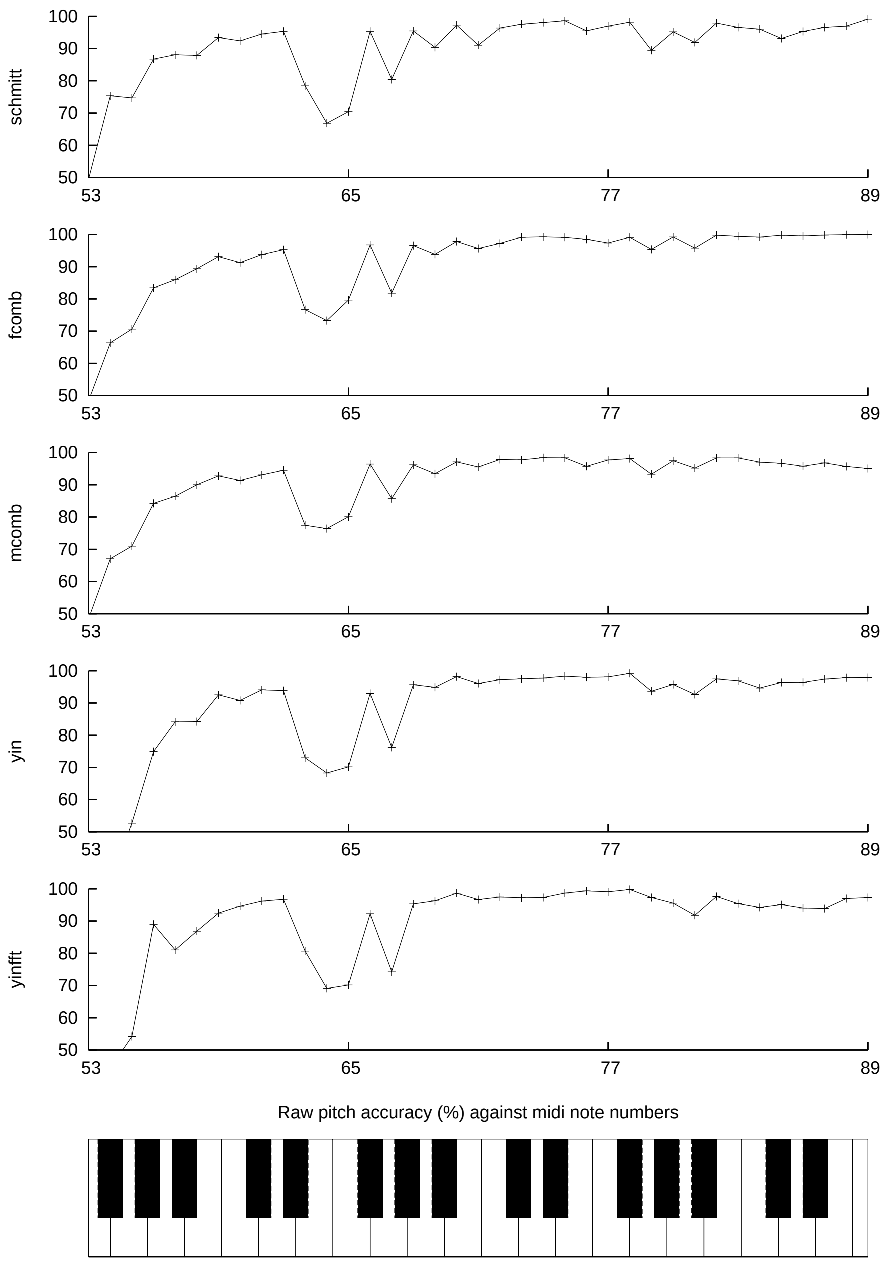


Figure 3.12: Percentage of raw pitch accuracy obtained on 444 isolated vibraphone notes played in 13 different modes. See Figure 3.9 for complete description.

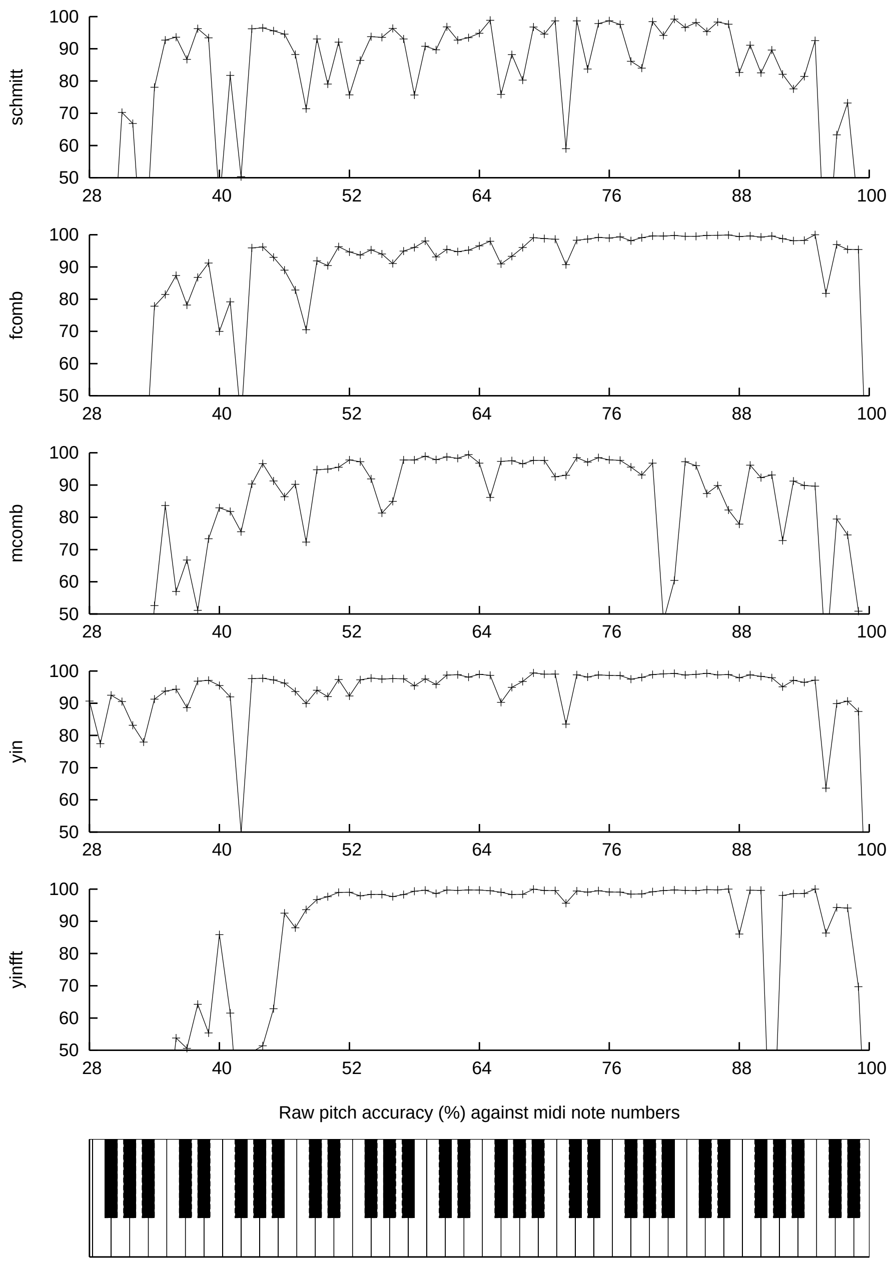


Figure 3.13: Percentage of raw pitch accuracy obtained on 292 isolated rhodes notes played in 5 different modes. See Figure 3.9 for complete description.

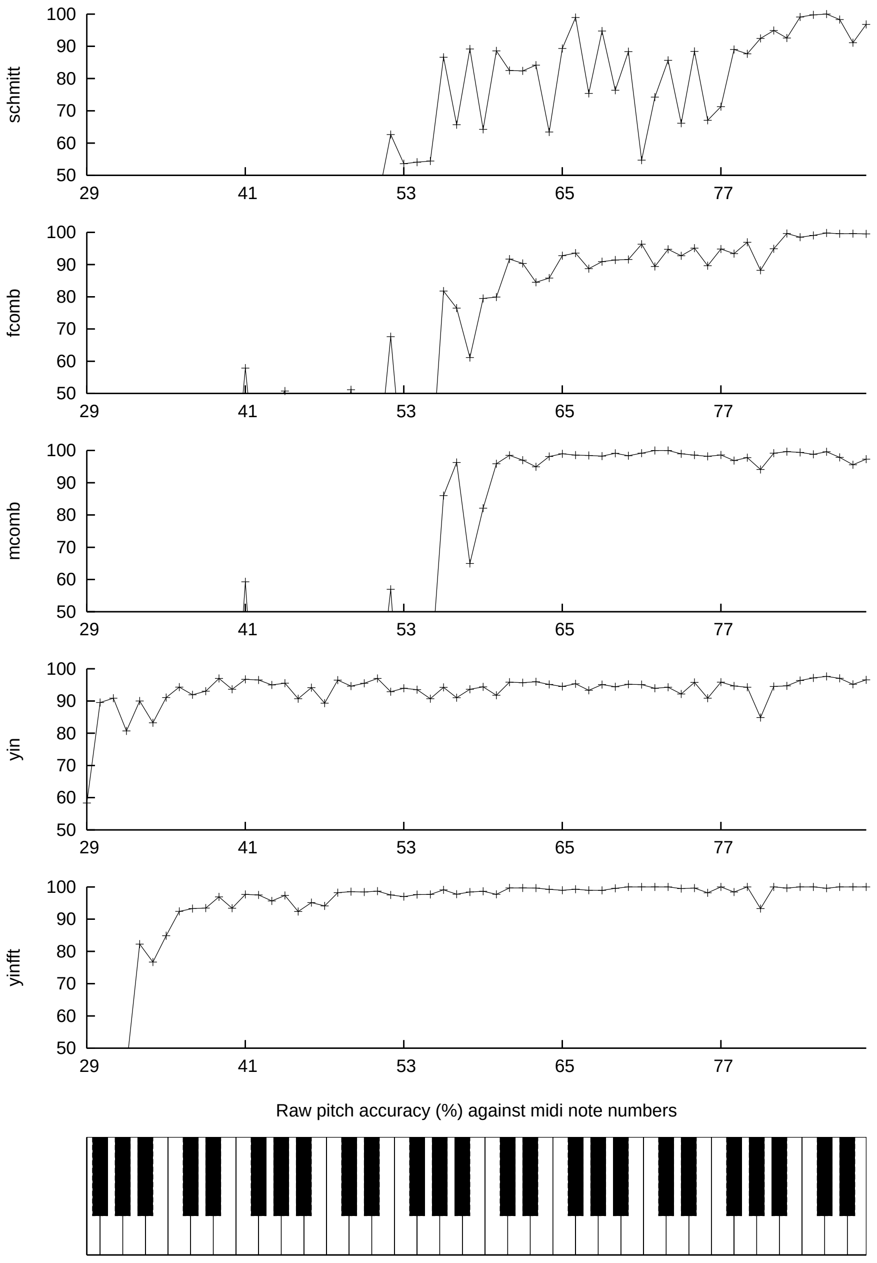


Figure 3.14: Percentage of raw pitch accuracy obtained on 240 isolated clavinet notes played in 5 different modes. See Figure 3.9 for complete description.

Isolated notes Figures 3.9 to 3.14 present the results obtained on the set of instruments with a window of 2048 points. The number of correct detections found within the tolerance of half a semi-tone was counted for each note in the database and averaged across different playing modes. The graphs are presented for a range of MIDI notes corresponding to the register of each instrument. The keyboard at the bottom of each graph represents all the notes in the instrument's register. For completeness, we included the lower octave of the piano, MIDI notes 21 to 33 in Figure 3.10, although the algorithms are not designed to achieve sufficient accuracy at these frequencies with a window size of 2048 points.

This approach allows us to pinpoint precisely the various difficulties that arise for different timbres and playing modes in different parts of the spectrum. Specific timbres, such as the one of the clavinet for which results are shown in Figure 3.14, highlight the tendency of the Schmitt trigger to produce octave errors, whereas YIN remains consistently more stable over the whole keyboard of the instrument. Some instruments are found to be problematic to all methods, as can be seen in Figure 3.12 with the results of the vibraphone.

The results obtained on the entire database, Figure 3.9, show that more than 60% of the frames could be retrieved accurately as low as MIDI note 36 using one of the three temporal domain methods, Schmitt trigger, YIN, or modified YIN. With the fast comb approach in the spectral domain, more than 90% of the frames could be retrieved up to MIDI note 100. The multi comb method does not perform as well as the fast comb algorithm in the highest notes of the database, from MIDI notes 92 to 108. On these notes, the spectral pattern matching method is less efficient because too few harmonics are detected in the spectrum, which causes the algorithm to select the wrong frequency, in most cases one octave below the actual fundamental frequency.

The overall score of the Schmitt trigger, which retrieves only slightly more than 70% correct estimates in the MIDI range 45 to 90, has to be balanced with its reduced computational load. Detailed results show that the Schmitt trigger has a robust behaviour on the electric guitar, Figure 3.11, while the results are unsteady on the Fender Rhodes and clavinet timbres, Figures 3.13 and 3.14. Although less versatile than the fast comb method, the multi comb approach is more robust in the MIDI range 48 to 80, especially on timbres with loud transient components such as the piano, Figure 3.10, or the clavinet, Figure 3.14. The behaviour of the multi comb method in the highest part of the registers could be improved by reducing the number of harmonics searched for when too few peaks are detected in the higher part of the spectrum. After reducing the number of comb candidates from 5 to 2, results achieved on the highest notes with the multi comb approach were comparable to that of the fast comb filter. However, reducing this number did also affect the overall results in the center octaves of the register, thus limiting the advantage of the pattern matching algorithm. We chose to leave the number of harmonic candidates at a constant value of 5, which was experimentally found to give best results in the center octaves of the keyboard.

Both the original YIN algorithm and our modified spectral domain YIN present a significant improvement to the Schmitt trigger and the spectral domain approaches, with as much as 90% of the frames retrieved in the MIDI range 60 to 95. This improvement is particularly noticeable on the piano and Rhodes timbres, Figures 3.10 and 3.13. Results obtained on piano notes show that the temporal domain YIN is more accurate than our spectral implementation on the lowest part of the keyboard, from MIDI notes 29 to 37. This is due to the distortion introduced in our approach when assuming $r'_t(0) \approx r'_{t+\tau}(0)$ in Eq. 3.10, and which becomes significant when τ is large. The temporal implementation of YIN is also more robust on the highest notes of the keyboard, from MIDI notes 91 to 103. In this case, despite the search for several interpolated minima described in Section 3.2.4, the spectral implementation of YIN tends to select a minima at twice the period, causing octave errors. However, overall results obtained on MIDI notes 35 to 90 confirm that our new detection method is significantly more robust than the temporal YIN algorithm in the MIDI range 45 to 90, with more than 95% of the frames consistently labelled with the correct fundamental frequency.

Monophonic and complex recordings The results obtained on the monophonic database [MIREX, 2004b] using the five detection algorithms are shown in Table 3.3. Window sizes of 2048 points were also used in these experiments. Best results on the solo recordings were obtained for our spectral YIN implementation, with 85.09% of raw accuracy, followed by the multi comb spectral peak picking, with 83.89% of accuracy. Overall, spectral methods appear to be more affected by octave errors than temporal methods, as can be seen in the detailed results (Table A.5 in Appendix A): the fast spectral comb method detects almost 10% of the frames in the wrong octave, whereas for the Schmitt trigger less than 2% of the frames are labelled in the wrong octave. The same is observed for mcomb and spectral YIN methods, with respectively 0.39% and 0.56% of frames causing octave errors. The accuracy of the multi comb filtering technique was significantly improved when pre-processing the spectral frame with normalisation and low-pass filtering of the spectral frame, reducing the error rate by more than 15%. The C-weighting filter

	schmitt	mcomb	fcomb	yin	yinfft
daisy	*91.13*	95.51	95.33	94.10	**95.84**
jazz	*91.04*	**95.01**	94.49	93.70	94.62
midi	*69.38*	**94.09**	91.23	90.26	93.68
opera	*23.37*	52.01	48.99	37.11	**56.06**
pop	*71.38*	82.83	79.03	79.80	**85.21**
Total	*68.46*	83.89	81.71	78.91	**85.09**

Table 3.3: Results of the raw pitch accuracy for the monophonic database Table 3.2; best scores are shown in bold, worst scores in italic; detailed results are listed in Table A.5.

accounts for about 0.5% of improvement for both spectral comb methods.

Results on the polyphonic database, Table 3.4, were obtained using the same parameters as for the monophonic simulations, apart from the threshold of the YIN algorithm, which was set to 0.7, instead of 0.3 for the monophonic database, to allow the selection of minima in the presence of strong transients. The Schmitt trigger clearly did not cope with the presence of other instruments: the result of this method dropped to less than 7%. The fast comb filter algorithm appears to be less affected by the presence of background music than the multi comb filtering. Detailed investigation of the results revealed that the multi comb method is actually tracking the bass line on several of the recordings, suggesting the weighting of the lower frequencies are over-fitted for monophonic sounds. Again, our modified YIN algorithm implemented in the spectral domain gave best results over the polyphonic recordings, with almost 60% of the frames correctly identified. Note that since the evaluation metric we use is stricter than the one used for the MIREX Audio Melody Extraction contest, we do not compare our results to the one obtained by other participants. However, the accuracy achieved by our new method yinfft represents significant improvements to the multi comb and YIN methods.

Computational costs Computational costs are not always predictable from the theoritical cost, as the actual cost can depend on the fundamental frequency of the signal and its complexity. The temporal domain YIN algorithm will tend to be less expensive in detecting shorter pitch periods, since the computation of the square difference function can be stopped as soon as a minima has been detected in the YIN function. The cost of spectral methods may slightly vary depending on the preprocessing steps, the number of peaks in the spectral frame, and the method used to find them. However, the overall cost of the spectral domain approaches is

	schmitt	mcomb	fcomb	yin	yinfft
daisy	*12.43*	**80.76**	76.80	73.12	72.42
jazz	*7.42*	60.64	54.18	61.74	**70.31**
midi	*3.17*	38.09	46.91	**47.20**	46.46
opera	*7.40*	39.63	35.97	16.00	**49.12**
pop	*4.85*	23.88	16.24	32.47	**62.42**
Total	*6.94*	47.72	45.75	45.57	**59.25**

Table 3.4: Results of the raw pitch accuracy for the polyphonic database Table 3.2; best scores are shown in bold, worst scores in italic; detailed results are listed in Table A.6.

mostly due to the Fourier transform, which remains the same for different signals.

Tests were run on the polyphonic database, described in Table 3.2, and which contains a variety of sounds with fundamental frequencies in different parts of the spectrum. Each sound file was processed multiple times in a row by the same algorithm to reduce the influence of disk access time and other input/output redirections. Computation times obtained for different window sizes with our C routines are presented in Figure 3.15. The overlap rate was fixed to 50%, so that each sample was processed twice regardless the window size.

To minimise the cost of loading and unloading the library at each run and on each sound file, the Python interface described in Section 6.4.2 was used to automate extraction and evaluation tasks, and proved useful at optimising computation times: about 15 minutes were required to compute the five detection methods on the database of 2500 isolated notes, corresponding to 140 minutes of audio, the equivalent of two compact discs. The single process Python program helped reducing the overall memory usage, and significant improvements were brought by maximising file caching in the computation sequences.

As expected, the computational cost of the Schmitt trigger algorithm is by far the lowest, the method being limited to a series of variable assignments for each buffer. Both spectral methods, fcomb and mcomb, are significantly more expensive than the Schmitt trigger. Moreover, their cost descreases when longer windows are used, which confirms that the Fourier transform, computed with the optimised FFTW library [Frigo and Johnson, 2005], has a cost $O(n\log(n))$. Unsurprisingly, the multi comb method, which includes preprocessing of the spectral frame and the computation of several harmonic combs, is more expensive than the fast comb method.

The YIN method reveals a behaviour different from the one of other methods:

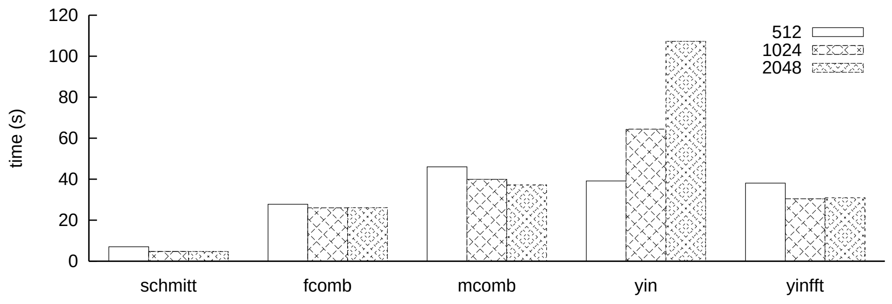


Figure 3.15: Computation times in seconds for different pitch algorithms on the Mirex 2004 database (Table 3.2, 368 seconds) at window sizes 512, 1024 and 2048, with 50% overlap. Tests were run on an Apple iBook G4 1.0 GHz running Debian GNU/Linux Etch.

as mentioned above, the cost becomes higher when longer windows are used, and despite the variable cost depending on the fundamental frequency of the signal, we measured longer runtimes with longer windows. Indeed, when no minima is found below the threshold, the whole square difference function, which costs $O(n^2)$, has to be computed. Listening tests using the real time implementation on synthesised signals also showed that this variable cost was problematic, causing some audio frames to be dropped on signals with low fundamental frequencies.

Our novel approach, yinfft, gave a cost in $O(n \log(n))$ as expected by its implementation using Fourier transform, and confirmed the improvement brought by our modifications. As we use the FFTW library to compute the Fourier transforms, which uses memory caching and other optimisation techniques, computing two transforms for each frame is even less expensive than the computations of the multiple harmonic comb.

3.5 Summary

We have presented and evaluated five different methods for the extraction of pitch and introduced a new spectral domain version of the YIN algorithm.

In the experiments, we compared fast spectral comb filter (Section 3.2.2), multi-comb filter with spectral smoothing (Section 3.2.2), Schmitt trigger (Section 3.2.3), YIN (Section 3.2.3) and a novel method, yinfft, derived from the YIN algorithm and computed in spectral domain (Section 3.2.4). Each method was evaluated against three different databases: isolated notes from different instruments playing in different modes, extracted from the RWC database [Goto et al., 2003], 20 monophonic tracks from MIREX 2004 [MIREX, 2004b], and 20 polyphonic mixes, also obtained from MIREX 2004.

Computationnaly simple methods, such as the Schmitt trigger and the fast comb filter, were successful at finding the fundamental frequency on synthetic sounds and on timbres with clear harmonic structures, such as the guitar or the Fender Rhodes. On the other hand, some instruments such as the vibraphone caused octave errors to all algorithms. Two of the detection techniques, yinfft and multi comb filter, gave the best results on monophonic recordings, with more than 83% of the frames correctly retrieved. The results obtained on isolated notes showed that YIN was the most versatile algorithm, able to retrieve more than 90% of the frames for notes from 55 Hz (A1) up to 1760 Hz (A6) with 2048 points window.

The results achieved by our novel detection method, yinfft, revealed a significant improvement to the other methods, for notes between 69 Hz (A2) and 1568 Hz (G6) on for monophonic and polyphonic databases. Overall, our spectral implementation of YIN appears more robust than all the other methods we tested. No parameters need to be adjusted for this method, which achieved best results on all databases, with almost 60% of the frames correctly retrieved on polyphonic signals, and 85% for monophonic signals.

Enhancements to these algorithms could be made in several ways. First, the higher and lower pitchlimits could be used as parameters for the pitch extraction algorithms, to look for pitch candidates inside a specific range, rather than discarding pitch estimates found out of this range. For high-resolution pitch tracks, peak continuation mechanisms could improve the continuity of the pitch track. This continuity is important to obtain perceptually relevant results. For this reason, a measure of continuity in the evaluation against manually annotated pitch tracks would be interesting.

Chapter 4

Tempo tracking

The localisation of beats is an essential part of an automatic annotation system. Performers will tap along the beat in order to keep in time, and the task of tracking tempo is well known to performers, Disc-Jockey or producers. Foot tapping is an intuitive movement, and the listener requires no musical knowledge to tap along the tempo [Dixon, 2001b]. Beat tracking is a well known task in the Music Information Retrieval community, and several approaches have been described in the literature [Goto, 2001, Dixon, 2001b, Klapuri, 2003a]. A review of several of these algorithms was given in [Hainsworth, 2004].

A causal beat tracking algorithm specifically designed to track tempo changes in real time was described in [Davies and Plumbley, 2004]. The algorithm, based on a phase vocoder and an onset detection function as described in Chapter 2, was tested over a large database of annotated music samples. We present comparative results obtained using different methods and describe our real-time implementation of the algorithm [Davies and Brossier, 2005].

4.1 Introduction

Although tapping may seem natural for a human listener, automatic tempo tracking is not a trivial task, especially in the case of expressive interpretations and tempo changes. Rhythmic structures can be very complex: their perception has been modelled using a hierarchy of streams forming patterns of different durations [Lerdahl and Jackendoff, 1983, London, 2002]. Three layers are accordingly described in the literature. The *tactus* is a time quantum dependent on the musical context, and is the shortest time interval found between onsets creating the perception of a rhythm. The tactus provides a fine grained rhythmical grid at around 40 ms to 100 ms resolution, related to the minimum perceptible inter-onset interval. The *metre*, often referred to as beat period, is the foot taping rate, with typical values ranging from 200 ms to 1.5 seconds. The metre is the phenomenon of entrainment which arises in a musical context from the combination of these patterns in rhythms. It is usually found to be at multiple values of the tactus. At a higher level, the *measure* is related to the time signature of the piece and often corresponds to harmonic changes and rhythmic variations. Music performances are not perfectly isochronous, and a system to extract beat locations from real musical audio must take into account important deviations from one beat to another.

In [Lerdahl and Jackendoff, 1983], a set of rules to derive rhythm patterns was described. The concept of *metrical dot notation* was proposed, formalising the three hierarchic levels – *tactus, metre, measure* – on score notations. Alternative rules drawing looser constraints for the generation of complex rhythm figures were proposed in [London, 2002]. The development of these rules and a better understanding of the perception of rhythm has helped the design of algorithms to extract the beat location from a MIDI score and from musical audio. For example, Brown [1993] describes a system to extract tempo from MIDI files using an autocorrelation of the onset times of the notes. The task of tracking tempo is highly subjective, and it has been shown that perception of tempo varies from one listener to another [Moelants and McKinney, 2004]. Systems for simultaneously tracking multiple tempo values were described, either based on a sequence of onset times [Allen and Dannenberg, 1990] or a MIDI score [Rosenthal et al., 1994]. Prior knowledge of a musical piece is used by Raphael to drive an automatic accompaniment system capable of following tempo changes [Raphael, 2001a,b].

Several approaches have been proposed to extract beat location from audio signals in the literature, a number of which are explicitly based on sequence of onsets or on onset detection functions described Chapter 2. The algorithm described in [Goto and Muraoka, 1995a,b] uses multiple "agents" with different strategies to detect temporal features such as onset times, inter-onset intervals and pre-defined spectral templates. Hypothesis from the different agents are then combined together to infer reliable beat locations. At the time, the system could run in real-time using a specific parallel computer. Improvements of this algorithm were described in [Goto, 2001] and include extraction of bar and tactus locations and enhancements for non-percussive audio signals. Scheirer [1998b] used impulse trains detected in several bands as the input of a bank of comb filters. The outputs of each filter are summed together and the function obtained is searched for peaks, corresponding to the best tempo estimates. Scheirer [1998b] implemented this algorithm as a computer program, and real time tracking of music was possible on an advanced desktop workstation. The BeatRoot algorithm described in [Dixon, 2001b] also uses an onset detection function to detect onset times in a first pass, then finds different beat period hypothesis by constructing a histogram of the inter-onset intervals. The locations of beats are infered by looking for sequences of events which match one of the period hypothesis and align to the onset times. In [Klapuri, 2003a], a probabilistic modelling of musical rules is used to infer the three layers of rhythmic structure from acoustic signals – tactus, metre, measure. The features used in the model are similar to a multi-band onset detection function described Section 2.2. The tactus, meter and measure bars are estimated based on a series of observations given a set of prior rules on the rhythmic structure of the music. A causal system based on particle filtering was presented in [Hainsworth, 2004]. The system was shown to be accurate on a wide range of audio signals, although its implementation is complex and computational cost of the algorithm may not be appropriate for real time implementation.

For interactive systems, a beat tracker should be computationally efficient, able to tap along live musical audio and follow rapid tempo changes. Davies and Plumbley [2004] described an algorithm for efficient causal beat tracking of musical audio. Further improvements of the algorithm were proposed in [Davies and Plumbley, 2005]. We describe here this algorithm and its software implementation in real-time [Davies and Brossier, 2005].

4.2 A context dependent algorithm

The algorithm we describe here is based on the onset detection methods seen in Chapter 2. With a signal sampled at $f_s = 44100\ Hz$, the onset detection is computed on $N = 1024$ points phase vocoder with a hop size of $M = 512$ samples. The onset detection function is then peak-picked using our adaptive thresholding operation defined Eq. 2.12, and observations from the past 3 seconds are kept in a buffer. These 3 seconds, corresponding to 256 frames of 512 samples, are required to detect slow beat periods, with at least two beat locations in the buffer. To evaluate the periodicity of the detection function, its autocorrelation function (ACF) can be computed. To improve the accuracy of the beat period measurement, several peaks in the ACF can be evaluated. The standard ACF, defined in Eq. 3.5, gives less weight to shortest lags. To facilitate the estimation of the beat period across multiple peaks of the ACF, the *narrowed autocorrelation* – or unbiased autocorrelation – was proposed in [Brown, 1993]. This function gives equal weights to contributions of different time lags:

$$\hat{r}_D[l] = |l - N| \sum_{n=0}^{N-1} \hat{D}[n]\hat{D}[n-l], \tag{4.1}$$

where $\hat{D}[n]$ is the peak picked detection function, as defined in Eq. 2.13, and l the time lag in steps of 512 samples. The scaling factor $|l - N|$ is used to lift up the contributions of longest lags, as opposed to the standard autocorrelation $r(l)$ defined in Eq. 3.5. An example of narrowed autocorrelation profile obtained using Eq. 4.1 is shown in Figure 4.1. The lag l, in detection function samples can be converted to tempo value in beats per minute (BPM) using the relation $l_{\text{bpm}} = (60 f_s)/(lM)$, where M is the hop size and f_s the sampling rate of the signal.

The location of the period peaks in the ACF does not depend of the alignment of the beat locations to the audio signal, or *beat phase*, since the autocorrelation is shift invariant. Using a bank of comb filters is thus an efficient way of finding location of period peaks in the ACF: each filter matches a given lag and searches for several peaks evenly spaced in the ACF. Searching for four consecutive was experimentally found to give successful results [Davies and Plumbley, 2005]. The comb filter resulting in the most energy corresponds to the lag of the function.

Equal weighting of the ACF may also causes issues where very short or very long lags are detected as the best matching candidate. To favour the detection of beats within realistic time lags, a weighting can be applied prior to the comb-filtering. The weighting of the ACF should match tempo values larger than 40 BPM and smaller than 250 BPM, given the inverse relation between lag and tempo. A perceptually motivated weighting is constructed using the Rayleygh distribution function:

$$L_w[l] = \frac{l}{b^2} e^{-\frac{l^2}{2b^2}}, \tag{4.2}$$

where a value of $b = 48$ detection function samples for the Rayleygh parameter gives the strongest weight to lag values around 60 samples, which corresponds to a tempo of 107.6 BPM, according to the above conversion formula. In the upper left plot of Figure 4.3, the Rayleygh distribution obtained for values of b, ranging from 30 to 50 frames of 512 samples, are shown. The slower the tempo is, the less probability it is given, so that a tempo of 90 BPM is preferred to one of 45 BPM. Very short lags are also given less weight, so that a tempo value of 90 BPM will

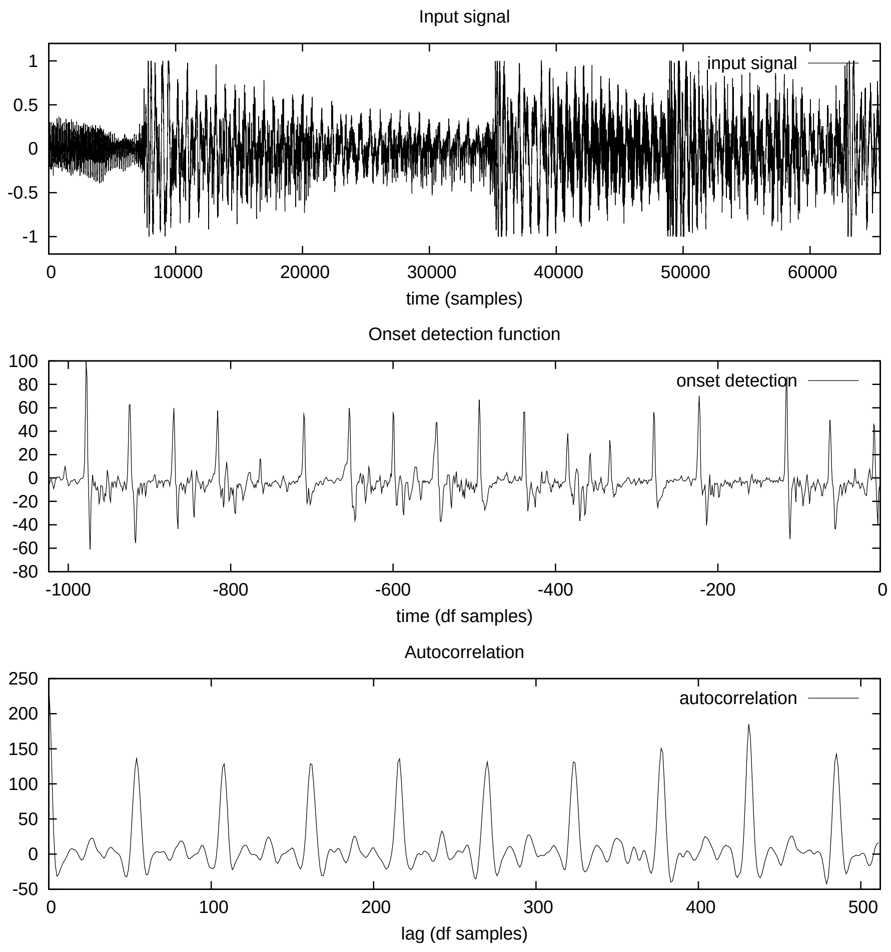


Figure 4.1: Example of autocorrelation function used for causal beat period detection. Top raw: current buffer of signal waveform, representing about 3 s of music. Middle raw: spectral difference onset detection function after dynamic thresholding. Bottom raw: autocorrelation function computed on the onset function according to Eq. 4.1.

tend to be selected by the system rather than 180 BPM.

After the time lag between two consecutive beats has been found, the phase alignment is found by cross correlating a train of impulses to the detection function. An exponential weighting is applied to the original function to favour the detection of the most recent events:

$$A_w[n] = e^{-\frac{n \log 2}{\tau}}. \tag{4.3}$$

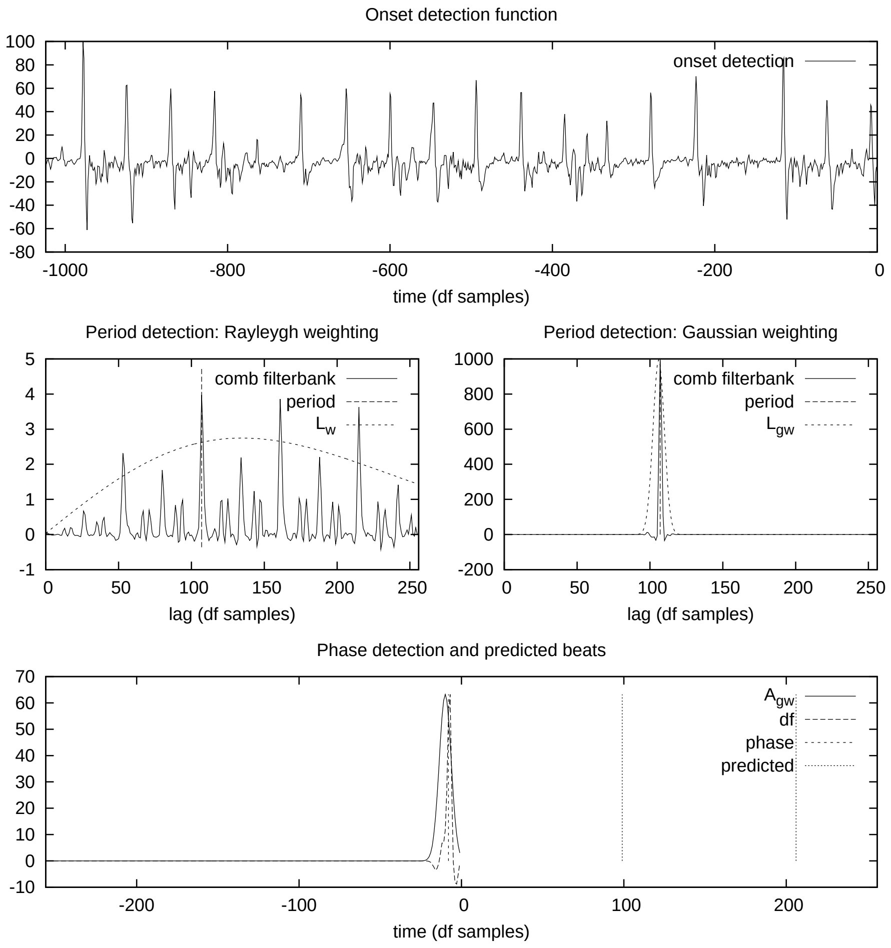


Figure 4.2: Example of causal beat period detection. Top raw: past window of onset detection function (origin at present time); Second raw left: Rayleygh weighting of the comb filter output (initial model); Second raw right: Gaussian weighting of the comb filter output (context-dependant model); Bottom raw: phase detection and predicted beat locations (origin at present time)

Given the phase value and the beat period, beats can be predicted up to the length of the autocorrelation function, about 1.5 s. To improve the continuity of the beat tracking algorithm, a context dependent model was developed [Davies and Plumbley, 2005]. After three consistent beat candidates have been found, the system enters an alternative mode, or *state*, which takes into account the past beats to refine the prediction of the following ones. A new weighting function is

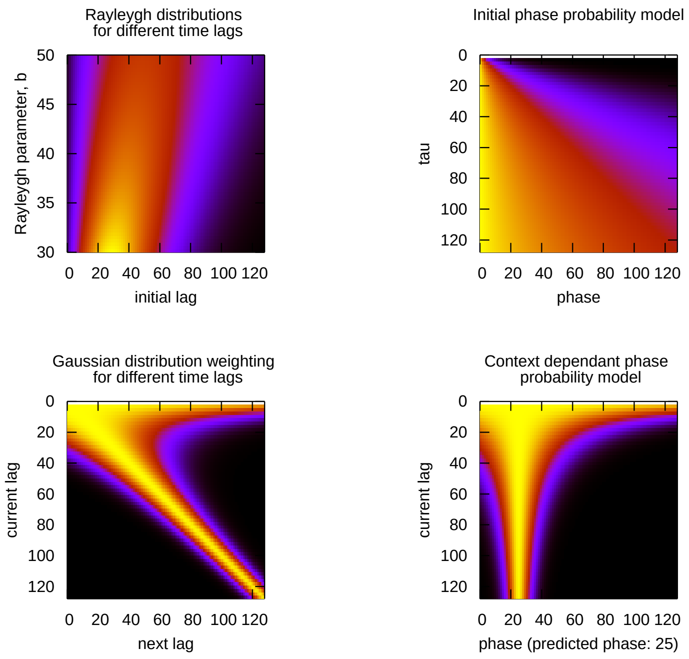


Figure 4.3: Probability distributions for the two state model beat tracking. Upper line shows the initial state model: Rayleygh distributions for the tempo lag (Eq. 4.2) and phase distribution functions for various lag candidates (Eq. 4.3). Bottom line shows the context dependent distributions for lag (Eq. 4.4) and phase probabilities (Eq. 4.5).

used to search for best matches in the ACF within a Gaussian distribution around the predicted lag:

$$L_{gw}[l] = e^{\frac{-(l-\tau)^2}{2\sigma^2}}, \tag{4.4}$$

where τ is the last predicted beat period and σ a variance empirically set to $\tau/8$ to allow for some deviation. Similarly, the phase alignment of the beat location is derived from the detection function but using a Gaussian distribution:

$$A_{gw}[n] = e^{-\frac{(n-\gamma_{last})^2}{2\sigma^2}}, \tag{4.5}$$

where γ_{last} is the last predicted beat location. An example of output obtained from the comb filters after Rayleygh and Gaussian weightings, L_w in Eq. 4.2 and L_{gw} in Eq. 4.4, are given in Figure 4.2. While the Gaussian weighting of the comb filter output prevents the beat period from switching between metrical levels, the Gaussian weighting, A_{gw} defined in Eq. 4.5 and shown in Figure 4.2, applied on the detection function for the phase alignment, prevents the system from switching from on-beat to off-beat.

The switch between the two states, initial hypothesis and context dependent model, is made according to the following criteria. If consistency has been found across the past three lags, the system enters the context dependent model. The continuity criteria is $|2\tau_r(i) - \tau_r(i-1) - \tau_r(i-2)| < \sigma^2$ with τ_r the time lag obtained from the Rayleygh weighted ACF and σ defined in Eq. 4.4. Lag candidates are now given by the context dependent model, which adapts to small variations across time. Both models are used simultaneously, and their consistency is evaluated: $|\tau_r(i) - \tau_g(i)| < \sigma^2$. When a new lag candidate τ_r is found to differ from the context dependent model τ_g, the system uses the Rayleygh weighting model as a new set of hypothesis until sufficient continuity has been found. This mechanism favours continuous tracking of the beat period, while allowing abrupt tempo changes.

4.3 Evaluation

An implementation of the two-state model algorithm was written in C, based on the onset detection functions described in Chapter 2. Experiments with the real time implementation showed how the Rayleygh distribution plays an important role in the final output, as it drives the behaviour of the context model by setting the initial parameters of the Gaussian weighting. The parameter b proves to be a useful "handle" to manually correct the system where it tracks the tempo at the wrong value. Another important parameter is the phase of the beat locations. As the beats are predicted, the results can be played slightly in advance or slightly delayed, which is an important feature for real time accompaniment systems. Simple mechanisms for manual corrections of off-beat tracking situations can also be employed, for instance using a "tap button" which users can tap at the correct beat period.

4.3.1 Performance metrics

Issues in evaluating beat tracking systems have been described in [Goto and Muraoka, 1997]. The primary performance metric is proposed to be the ratio of the longest continuous correctly tracked segment to the total length of the signal. This metric has been used in recent beat tracking studies [Klapuri, 2003a, Hainsworth, 2004, Davies and Plumbley, 2005]. Beats locations are considered as correct if there phase is found within $\pm 15\%$ of the annotation and if the tempo value corresponds to the annotations within $\pm 10\%$ of deviation. In order to cope with the ambiguity of the beat tracking task, four cases are identified: continuity at the correct metrical level (CML cont.); the total number of beats at the correct level, with the continuity constraint relaxed (CML total); continuity where tracking can occur at the metrical level above or below the annotated level (AML cont.); and total number of beats allowing for ambiguity in metrical level (AML total). One issue with this metric is that a single missed beat may cause the CML to drop from 100% to 50%. Evaluating the results of the human annotations (Table 4.1) gave poor CML scores, suggesting the continuity criteria are too strict.

A straightforward application of a tempo tracker is the estimation of the average tempo period of a musical piece. The task of automatic tempo extraction was evaluated at the second edition of the MIREX contest [Gouyon et al., 2006, MIREX, 2005c] using a metric specifically designed to take into account the ambiguity of beat tracking. The evaluation was made on 144 sound files, each annotated by 40 listeners. Algorithms were tested according to several tasks: ability to extract one or both most salient beat periods; ability to extract the correct phase alignment of one or both beat periods.

4.3.2 Experiments

We have tested the real time implementation using the database gathered by Hainsworth [2004], which consists of 222 annotated files of about one minute duration each, sampled at 44100 Hz and separated in different categories: rock/pop, dance, jazz, folk, classical and choral. Details of the database are provided in Table A.7. Table 4.1 shows the results obtained on the database by different algorithm. The causal implementation of the algorithm does not perform as accurately as the non-causal implementation but represent a significant improvement from the BeatRoot algorithm.

Our C implementation was also evaluated along with several other approaches during the MIREX 2005 tempo extraction contest [MIREX, 2005c]. To simulate the extraction of the two most salient tempo periods, as required by the contest, beat locations were extracted using the causal algorithm and the average period was selected as the most salient period. To provide to the evaluation algorithm of the contest with a complete output, the second most salient period candidate was selected arbitrarily to half or double of the most salient period: half when the first tempo period is found above 110 BPM, and double when found below. A sum-

Beat Tracker	CML Cont. (%)	CML Total (%)	AML Cont. (%)	AML Total (%)
State model (causal)	46.7	53.9	58.4	69.3
State model (non-causal)	54.8	61.2	68.1	78.9
Dixon Beatroot	25.1	30.9	46.2	59.6
Klapuri non-causal	55.7	62.4	71.2	80.9
Klapuri causal	52.3	59.7	65.2	69.3
Listener annotations	52.3	80.0	56.3	86.6

Table 4.1: Evaluation metrics (see Section 4.3.1) for different algorithms: Causal and non-causal State switching model [Davies and Plumbley, 2005], Scheirer's Beatroot [Scheirer, 1998b], Hainsworth's Particle filter [Hainsworth, 2004], Klapuri's probabilistic modelling in causal and non-causal modes [Klapuri, 2003a] and listener annotations. Database details are given in Table A.7.

mary of the results obtained for the various algorithm are reproduced in Table 4.2. The computational costs obtained show an important difference in the different approaches. Most systems analysed the 140 sound files in 1000 to 3000 seconds, while our causal implementation took 180 seconds to compute the whole database extraction. This confirms the very low computational complexity of this implementation, which mostly consists of the phase vocoder used for the onset detection function, with only a small level of additional cost from the autocorrelation and the tempo detection model.

4.4 Summary

We have reviewed several algorithms for beat tracking of music signals and described a real time implementation of a causal beat tracking system, based on the multi-comb autocorrelation of an onset detection function. This algorithm is efficient and able to predict beat locations from a variety of percussive and non-percussive music signals. Although the non-causal implementation proved to be more reliable than our causal implementation, the real time system achieves encouraging results, especially on percussive signals.

The evaluation of beat tracking system has been approached, and a number of issues are raised by this process. In order to better understand the rhythmic structures in music, several applications would benefit from higher level elements such as the time signature and the bar locations.

Participant	Score (std. deviation)	At Least One Tempo	Both Tempi	At Least One Phase	Both Phases	Mean Abs. Diff.	Run-time (s)	Machine
Alonso, David, Richard	0.689 (0.231)	95.00%	55.71%	25.00%	5.00%	0.239	2875	G
Uhle, C. (1)	0.675 (0.273)	90.71%	59.29%	32.14%	7.14%	0.222	1160	F
Uhle, C. (2)	0.675 (0.272)	90.71%	59.29%	32.86%	6.43%	0.222	2621	F
Gouyon, Dixon (1)	0.670 (0.252)	92.14%	56.43%	40.71%	7.86%	0.311	3303	G
Peeters, G.	0.656 (0.223)	95.71%	47.86%	27.86%	4.29%	0.258	2159	R
Gouyon, Dixon (2)	0.649 (0.253)	92.14%	51.43%	37.14%	5.71%	0.305	2050	G
Gouyon, Dixon (4)	0.645 (0.294)	87.14%	55.71%	48.57%	10.71%	0.313	1357	G
Eck, D.	0.644 (0.300)	86.43%	53.57%	37.14%	5.71%	0.230	1665	Y
Davies, Brossier (1)	0.628 (0.284)	86.43%	48.57%	26.43%	4.29%	0.224	1005	R
Gouyon, Dixon (3)	0.607 (0.287)	87.14%	47.14%	36.43%	6.43%	0.294	1388	R
Sethares, W.	0.597 (0.252)	90.71%	37.86%	30.71%	0.71%	0.239	70975	Y
Davies, Brossier (2)	0.583 (0.333)	80.71%	51.43%	28.57%	2.14%	0.223	**180**	B0
Tzanetakis, G.	0.538 (0.359)	71.43%	50.71%	28.57%	3.57%	0.295	7173	B0

Table 4.2: Summary of evaluation results from the MIREX 2005 Audio Tempo Extraction contest [MIREX, 2005c]. Non-causal (1) and causal (2) implementations of the two states tempo tracking algorithm are indicated in bold font. The details of the other algorithms can be found in [MIREX, 2005c].

Chapter 5

Note modelling

Based on the pitch and onset annotations we have obtained from the techniques described in the previous chapters, Chapter 2 for temporal segmentation and Chapter 3 for pitch detection, we are now looking for ways to combine the spectral and temporal observations, and group them into musically meaningful annotations. Specifically, we want to identify notes in a MIDI like format, with their precise beginning and end, their velocity, and their pitch.

We give here an overview of several methods designed to extract note objects from musical audio. We then describe two approaches we have implemented. The evaluation of the performance of these methods is then tackled. Different metrics for the evaluation of the note extraction task are described, and our implementations are tested against a database of various musical scores.

5.1 Introduction

The extraction of note objects is an important step towards the understanding of higher-level structures in music. Direct applications of such extraction include audio to score transcriptions, score following and query by performance. A variety of systems have been proposed for monophonic [Monti and Sandler, 2002] and polyphonic transcription [Bello et al., 2002, Klapuri et al., 2001, 2002]. Recent transcription systems use an explicit note modelling technique to extract MIDI like notations from speech [Ryynänen and Klapuri, 2004] and music [Ryynänen and Klapuri, 2005]. In these, hidden Markov model (HMM) is used for each note to model the likelihood of a note given a set of features extracted from the acoustic signal. A second Markov model is used to estimate the likelihood of transitions between notes given a set of musicological priors. In this second model, for instance, the probability of third and fifth intervals can be set to higher values than the probability of a second interval.

Real time estimation of the attributes of the note object, such as the fundamental frequency and its temporal boundaries, involve a trade-off in the time frequency domain. Onset times are precise for percussive onsets but less well defined for smoother attacks. The observation of pitch following onset will be affected by the presence of transients in percussive or breathy attacks, but easier on bright timbres with a sharp harmonic structure. Here we concentrate on the problem of labelling notes with a pitch as quickly as possible after the note onset, on solo recordings. We describe a system for the low-latency characterisation of note events [Brossier et al., 2004a], based on the information provided by our note segmentation algorithm, described in Chapter 2, and on the estimation of the fundamental frequency, described in Chapter 3. The system is designed to achieve robust labelling of notes on a large variety of solo musical recordings, in various acoustic environments. In Section 5.3, different methods to evaluate the performance of the note extraction task are reviewed, and the results of our experiments we ran to evaluate our implementation.

5.2 Real-time oriented approaches

One way to think about the task of extracting note events is to consider the extraction of a MIDI score corresponding to the observed audio signal. Real time implementation of a note modelling system is made complex by the requirements in terms of latency and computational costs. Modelling notes in real time consists in deciding of Note-On MIDI events, where the note start with a given pitch and velocity, within a short latency. Errors in the estimation of onset times and fundamental frequency may happen, but the combination of the temporal analysis and the pitch estimations can help maximising the success of the operation. As a first approach, a set of simple rules are expressed as follows: the detection of temporal onsets triggers the creation of a new note event; after a consistent pitch value has been found, the note event is labelled and can be sent; if a new pitch value is consistently found, a new note event is created. The main issue remains the estimation of a consistent pitch value after the onset of the note and during its transient.

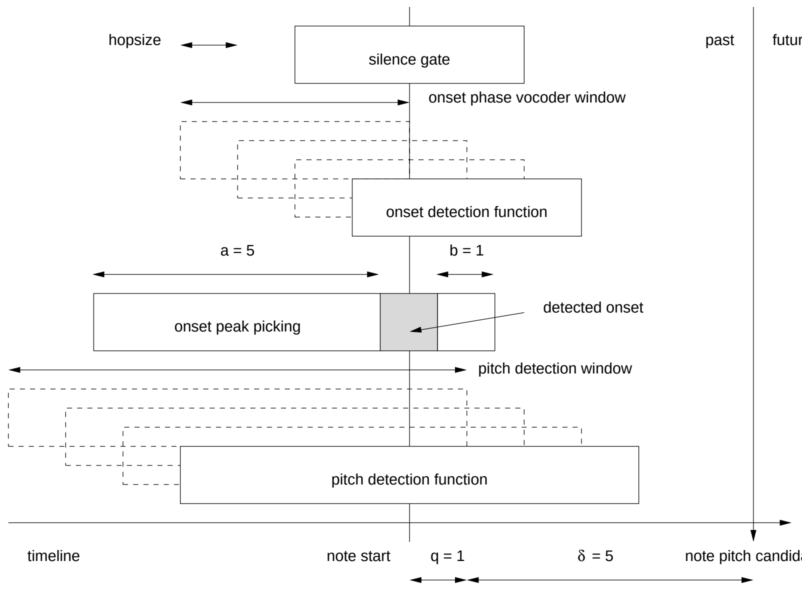


Figure 5.1: Detail of the synchronisation of the different modules used in parallel for note estimation: silence gate, onset detection function, onset peak picking and pitch detection modules use different window lengths and must be synchronised.

Median based approach To evaluate a consistent note pitch candidate in a small number of frames after the onset, we have choosen to compute the median over the candidates that appear in the frames after the onset:

$$P_{\text{note}} = \text{median}(P_q, P_{q+1}, ..., P_{q+\delta}), \tag{5.1}$$

where δ is the number of frames considered for the pitch decision and will determine the total delay of the system. The first q frames are not considered, so as to take into account the delay between both pitch and onset phase vocoders and optionally ignore early pitch candidates. This parameter significantly increases the delay, but allows spurious detections in the attack of the sound to be ignored. Another potential source of error occurs when the amplitude of the different partial change within the note: octave or fifth errors may then occur. The details of the different modules operating simultaneously is given in Figure 5.1. To limit the delay of the decision, we can use a varying δ depending on the context: short after the onset, with only 3 frames – 33 ms; long during steady states, up to 7 frames. When three frames have occured after the last detected onset, δ is incremented

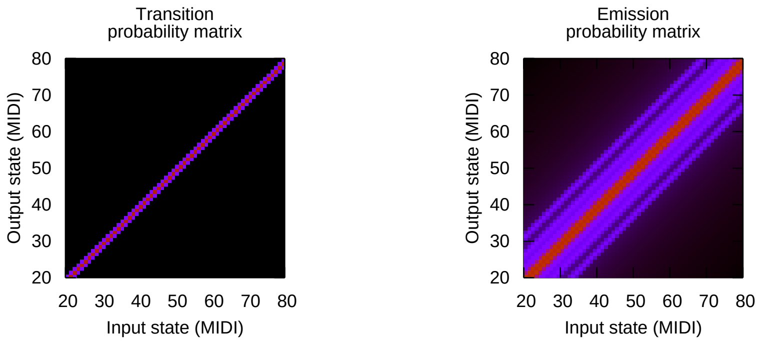


Figure 5.2: Emission and transition probability matrixes used in the hidden Markov model for temporal pitch smoothing using Viterbi decoding.

by 1 for each consecutive frames. This mechanism allows us to have a variable measure of consistency, based on a short time delay at the beginning of notes, and a longer delay during the steady state. The parameter δ allows short spurious detections to be ignored, including jumps to harmonic or sub-harmonic components. The computational cost of the search remains low as the moving median can be computed as a simple sort algorithm.

Viterbi decoding approach Another technique commonly used to determine the most likely prediction of future observations given a set of past observations is the use of the forward Viterbi algorithm. This dynamic programming technique involves the use of hidden Markov models [Rabiner, 1989]. A simple prototype can be built to compute the most likely pitch path given a set of observed values. The observations are are encoded as pitch values in MIDI note number, rounded to the nearest integer, and silences are encoded with 0. To each observation corresponds one hidden state. Fixed probabilities distributions are used in the Markov chain: initial probabilities of all states are equal. The self-transition probability is set to a very high value to reflect the steady states, while the transition to other states are kept low. The emission probability is built to reflect a high probability of transitions from one notes to the other, with a small weight given to semi-tones interval, and a higher probability for 2, 4 and 7 intervals. The probability matrixes are shown in Figure 5.2.

.

Given this probability model, the forward Viterbi algorithm can compute the

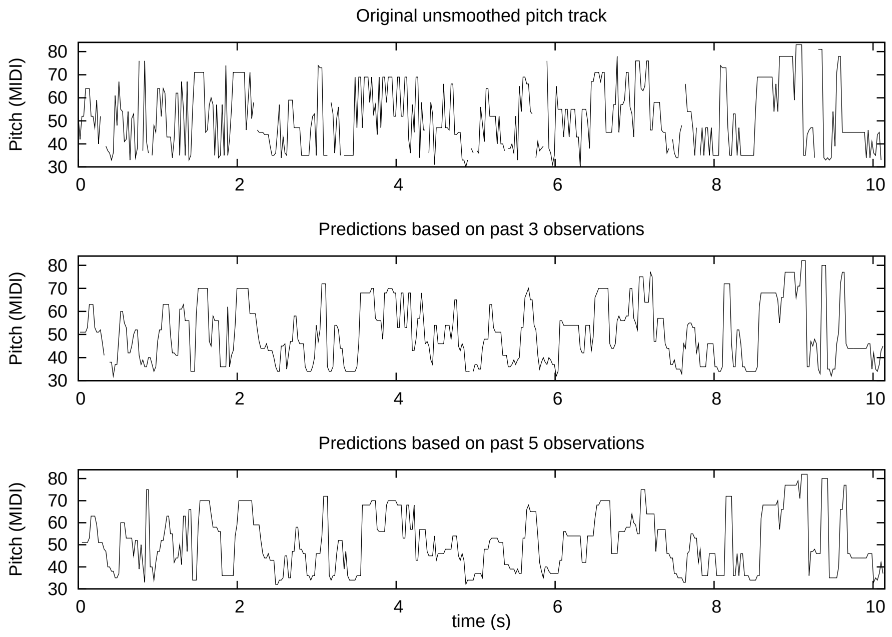


Figure 5.3: Example of pitch track obtained using the multi-comb algorithm, with and without smoothing using the Viterbi decoding approach with 3 and 5 past observations. Jitter and noise are removed, gaps are filled. The sound file is the one used in Figure 2.2

likelihood of a sequence of observations and give a prediction for the next observed state. Experiments were run using sets of 3 to 10 pitch observations. As can be seen in Figure 5.3, the pitch track can be significantly cleaned by the Viterbi algorithm, with short gaps within notes being filled and a limitation of the spurious candidates in the transients. Although the results are encouraging, the method has a serious drawback for real time implementation: the computational complexity of the algorithm is in $O(NM^2)$ with N the number of observations and M the number of states. Computing the likelihood of more than a hundred notes for a single frame, typically with a duration of 11 ms, would not apply easily to a real time system. About 2.5 minutes were required to compute the predictions of the 10 s long sound plotted in Figure 5.3 using our Python implementation of the Viterbi decoding algorithm.

.

5.3 Evaluation

We describe existing approaches to the evaluation of transcription systems, and the technique we used to estimate the performance of our note identification system. We then describe the method we have followed to construct an evaluation database which contains several scores played by different instruments. Evaluation results we obtained on this database are discussed.

5.3.1 Performance metrics

Several proposal have been made for the evaluation of transcription system. The *edit distance*, also called *Levenshtein distance* [Crochemore and Rytter, 1994], is commonly used to measure the distance between two polyphonic scores. This distance counts the number of operations needed to transform one string into another; three operations are allowed: edition, insertion and deletion. This measure gives a good idea of the overall robustness of a monophonic transcription and the amount of work needed to obtain a correct score from the computed annotations. However it may not reflect the perceptual annoyance caused by spurious notes, and its extension to polyphonic signals is complex.

Alternatively, a measure of the precision and recall of the algorithm can be based on the number of notes correctly defined, the total number of notes found in the original and in the extracted score. These metrics are easier to extend to polyphonic transcription and overall faster to compute. An additional measure, the overlap ratio, was proposed in [Ryynänen and Klapuri, 2005] to reflect the temporal correctness of the transcription: the measure is defined as the ratio of the time delay where correctly extracted and original notes overlap to the length occupied by both notes:

$$overlap\ ratio = \frac{min\{offsets\} - max\{onsets\}}{max\{offsets\} - min\{onsets\}}, \tag{5.2}$$

where $onsets$ and $offsets$ are two pairs of extracted and original times. This measure is useful in the context of real time extraction, as it will be affected by the delay of the system.

For our experiments, we chose to measure the number of notes correctly detected within a time tolerance of 50 ms, similarly as in Chapter 2 for the onset evaluation, and within half a semi-tone, like was done in Chapter 3 for the pitch evaluation. Notes evaluated as correct will have been found within these two tolerance criteria, both in term of temporal localisation and pitch value.

5.3.2 Score database

To evaluate the performance of the note modelling in realistic monophonic situation, simulations need to be run on real recordings. Several notes may overlap each other, and the presence of reverberation in the recording may affect the overall performance. Two approaches are available to obtain the ground truth of the samples: annotate manually a small collection of recordings, or obtain the exact MIDI score of the sound sample. The first process is a long task, which can be semi-automated to help the alignment of the manual annotations. However, cross-validation of the database is time-consuming and although MIDI files created from real recordings are available for instance in the Real World Computing (RWC) database [Goto, 2004], the MIDI scores are not aligned to the audio signal. Another approach is to obtain directly the MIDI score that generated the recorded sound sample, either by recording an instrument equipped with MIDI captors, or by synthesising the audio directly from a MIDI score. Databases of piano recordings with corresponding MIDI scores have been gathered [Bello, 2003], but these recordings were made to the evaluate polyphonic transcription system. A drawback of generated files is that the quality of MIDI rendered files often too mechanical, especially the one that have been created from a score rather than from a performance. The rendering depends not only on the MIDI file, but also on the synthesis engine. Modern software solutions such as Timidity can achieve perceptually relevant results. This technique facilitates the automation of the benchmarks, and endless combination of audio samples can be generated from just a few MIDI files. The process also has the advantage of a perfect alignment of both the actual note onset and offset times, which allows for a precise estimation of the system delay.

We used MIDI files from the Mutopia Project [Mutopia project, 2000], which aims at gathering a collection of public domain music sheets. The collection used for our experiments contains nine different scores, instrument solos or leads extracted from chamber and symphonic music (see Section A.4 in Appendix A for detailed score references and availability). Each file was rendered with various instruments in order to evaluate the influence of the timbre on the pitch extraction algorithm. The most realistic samples have been selected to render the MIDI files. Waveform rendering from MIDI files was done using Timidity [Toivonen and Izumo, 1999], a state-of-the-art MIDI to PCM conversion utility. Timidity provides a command line interface that can convert MIDI files using different sound synthesis formats (GUS,

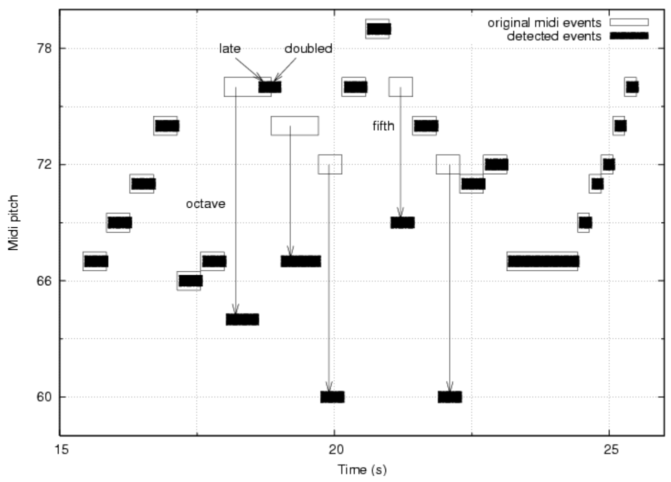


Figure 5.4: Example of typical note segmentation errors: detected events are shown in filled box, the original score is outlined. Extract from *Partita in A minor for Solo Flute, J. S. Bach, BWV* 1013, 1^{st} Movement: Allemande.

PAT, SoundFont2). To enhance the rendering of the MIDI files, some amounts of reverberation and chorusing were added, enough to be perceptually noticeable on each instrument.

5.3.3 Experiments

A test bed for the evaluation of our system has been implemented. Audio waveforms are generated using MIDI files and analysed through our note labelling program. The evaluation consist of the comparison between the original MIDI score and the list of candidate event detections we obtain, both against pitch and start time. `NOTE-ON` events are extracted as a pair of *MIDI pitch* and *start time*. If the detected event corresponds to an existing note within a tolerance window of length ϵ_t (ms) and with the correct MIDI pitch rounded to the nearest integer, the event is labelled as a correct detection. Incorrect detections can be characterised by their frequency error (e.g. octave or fifth interval errors), and their temporal error (e.g. doubled or late detections). An example of automatically retrieved score is given in Figure 5.4. In this piano-roll like graph, the original notes are drawn in solid lines, the detected events in filled black squares. The plot illustrates various types of errors, including octave and fifth jumps.

Using a fixed set of parameters for the onset detection and the pitch tracking,

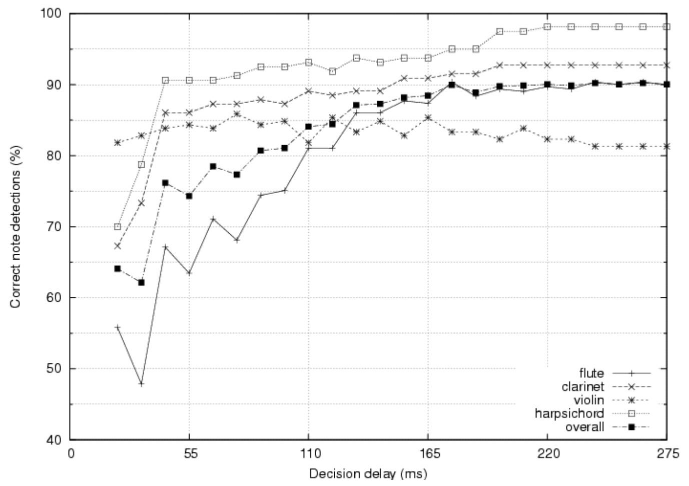


Figure 5.5: Correct note estimation results for different values of δ in Eq. 5.1, the number of pitch frames the decision is taken on, and for different instruments. The onset detection threshold α in Eq. 2.12 is fixed at 0.300.

we have estimated the number of notes which were correctly labelled for different values of δ in Eq. 5.1. The results obtained with different instruments are given in Figure 5.5 show the strong dependency of the number of pitch candidates required to obtain a solid note labelling and the nature of the instrument playing. The sharp attacks of the harpsichord or the clarinet lead to correct note results after only 4 frames, a delay of 44 ms. Most harpsichords have two strings tuned with one octave difference, leading to transcription errors. The SoundFont sample used for the harpsichord is simplistic and contains only one string, as opposed to the real recording seen in Figure 3.1. Soft attacks such as the flute will require up to 10 observations, a 110 ms duration, to obtain consistent note candidates. The breathy attacks are causing spurious pitch detections. The decrease of the performance found for the violin is explained by the presence of spurious octave or fifth errors within long steady states. In this case, a large δ and thus a strong criteria of consistency in pitch did not help the results.

The success of the median filter is improved by when δ is odd, because the number of pitch correct candidates is likely to be higher than the number of spurious pitch observations when the total number of candidates is odd. This explains the irregular profile of the curves in Figure 5.5. Overall results show that more than 80% of the notes could be correctly labelled within 110 ms. However for all instruments except flute, more than 84% of the notes can be retrieved within 45 ms.

We have tested successfully our software implementation (Chapter 6) in real time on an AMD Duron 700 MHz, where the process was using about 50% of the CPU, as well as on other machines.

5.4 Summary

We have presented and evaluated a new method for the determination of note objects within short latencies. The algorithm is based on the simultaneous analysis of temporal (Chapter 2) and spectral (Chapter 3) features. The pitch estimation runs in parallel with the onset detection functions, onset peak-picking and silence detection, and can run along other analysis such as tempo tracking (Chapter 4).

We have implemented this new note estimation method as a software program able to run in real time, and we have evaluated the performance and the delay of our approach. Details of our software solution and examples of integration with other applications are given in Chapter 6.

Chapter 6

Software implementation

Software applications and computer systems are evolving quickly, and drawing a state of the art in the field of computer science is not an easy task. The topic of computer music programming is the subject of several text books [Roads, 1996, Miranda, 2002, Zoelzer, 2002]. Some of the issues specific to the implementation of *Computational Audio Scene Analysis* systems have been discussed in [Scheirer, 2000, Amatriain, 2004, Tzanetakis, 2002]. Programming environment, graphical user interfaces, or protocols for musical data storage and exchange: different tools can create and process musical audio in different ways and for different applications. To understand how automatic annotations of musical data could be helpful to computer users, and illustrate some of the approaches adopted in the design of computer music programs, we first review several of these applications. Choices for the architecture of our system are made in view of their integration into other existing software environments. Guidelines for the design of an automatic annotation system are given. We then describe the approach we have followed in the implementation of our annotation system, the *aubio* library. In Section 6.4, examples of integration of this system are gathered.

6.1 Computer music environments

Programming music applications has attracted a large interest in the past decades. Computer music covers a large field of applications, and drawing clear boundaries in this field is not a trivial task. For the purpose of this document, we distinguish different approaches followed in the design of computer music solutions: programming environments and prototyping frameworks, targeted for research and creation; graphical applications designed to be intuitive to the end-user; protocols and formats, designed to exchange and store musical data.

Studying existing source code of musical programs helps understanding how these applications are constructed, and when this source is available, contributors can submit modifications, improvements and extensions to these programs. Open-source is thus a guarantee of flexibility and durability for a software solution and is therefore useful to the research community. Most of the software solutions we describe here are available under an open-source licence, although the names of some of the most famous commercial applications are also given.

6.1.1 Programming environments

Textual programming environments have pioneered the world of computer music in the late 1950's with several generation of the MUSIC-N languages. The different generations of this language rely on two grammars: the *orchestra* on one hand, where elementary functions or *opcodes* can be assembled to define instruments and effects; the *score*, on the other hand, for the control of these functions. Latest implementations of this software such as Csound 5 [Boulanger, 1998] include high level operations such as spectral modification and MIDI control. The Csound paradigm was also implemented as part of the MPEG-4 Structured Audio standard [Scheirer, 1998a], with the Structured Audio Orchestra Language (SAOL) and Synthesis Language (SASL) [Scheirer and Vercoe, 1999].

Another programming language for sound synthesis and processing, *SuperCollider*, was developed in the mid 1990's [McCartney, 1996]. The SuperCollider environment consists of a sound synthesis server, `scsynth`, and a language interpretor, `sclang`, which can parse a syntax derived from the Small-Talk object oriented language. Snippets of code written for sclang can be modified, interpreted and executed on the fly by the scsynth server, even when other instructions are already executed by scsynth, in a complete multi-task and multi-channel fashion. The SuperCollider environment contains several features, including functions for spatialisation and spectral processing, and control of external MIDI devices.

Chuck is a recent multi-platform and open-source programming language for real-time audio synthesis, composition, and performance [Wang and Cook, 2003, Wang et al., 2004]. The Chuck environment allows the management of timed structures at multiple time rates, with fine control over different parameters of the system. The Synthesis Tool-Kit (STK), a library for physical modelling of musical instruments, is integrated within the Chuck language, so that different instruments can be created and controlled seamlessly.

Csound, Chuck and SuperCollider are all environments adapted for the analysis, synthesis and composition of music in a real-time fashion. They have active communities of users, using them to teach, understand and create music with computers. A common paradigm to all these environments is the separation between signal rate and control rates: processing units or modules generate audio at the desired sampling rate, and their parameters are evaluated at a slower rate. This separation is helpful conceptually for the user, who can generate complex signals within a minimal amount of code. The distinction between signal and control also contributes to the efficiency of the sound rendering engines, by avoiding the evaluation of all parameters at each sample and optimising the scheduling of different computations.

6.1.2 Graphical interfaces

Visual programming environments such as *PureData* (PD) [Puckette, 1996a,b] or *Max/MSP* [Matthews et al., 2006] can be described as a graphical implementation of the MUSIC-N paradigm. Opcodes are represented as boxes with inputs and outputs and connected with virtual "wires". Two types of signal flow coexist: control and audio. The audio is computed at fixed block sizes, and the control event are polled in a scheduling queue. To extend basic functionalities of the PureData environment, a system of dynamically loadable plug-ins, also called externals, is used. Some of these externals include advanced functionalities such as onset detection (`bonk~`) and pitch tracking (`fiddle~`) [Puckette et al., 1998], which are used for instance to gather expressive informations from a live performance for the control of a generative music algorithm [Dobrian, 2004].

Computer music users are often more familiar with intuitive applications allowing the visualisation and editing of sound files. Graphical applications for audio processing have been adopted by composers, musicians and sound engineers. A typical audio workstation supports the editing of multiple tracks and allows the acquisition and manipulation of both audio and symbolic data. Several modules provide extensions to the host application. Virtual instruments or sound effects, these *plug-ins* can be reused across different applications. Both instruments and effects produce new sounds; their parameters can be controlled using MIDI-like control signals. Programs such as Pro Tools, Cubase and Logic Audio Pro are currently some of the most widely spread commercial applications. All these solutions understand the

VST plug-in standard interface, which comprises both a signal processing effects and virtual instruments.

Advanced audio editors include ways to add label tracks along the waveform. For instance, *Audacity* [Mazzoni et al., 2000] uses label tracks which contain annotated time stamps to ease the selection of segments – see for instance Figure 6.7. To include new functionalities, Audacity uses the Nyquist interface, a programming language derived from the LISP (LISt Processing) functional language and oriented towards sound processing. This language can be used to create new sound processing, analysis or synthesis modules. A description of Nyquist is given in [Miranda, 2002]. *Praat* [Boersma et al., 1992], another powerful sound editing application, was initially designed for the annotation of spoken and sung voice. Praat features a collection of analysis routines and graphical panels to visualise spectral data. Data plots can be saved directly in the PostScript format for integration in research publications. This research tool has shown to be useful as a composition tool for the synthesis and manipulation of voice [Miranda, 2002, Chapter 6]. *WaveSurfer* [Beskow et al., 2000] is an open-source audio editor designed as an extensible framework for analysis and visualisation. WaveSurfer has been used for research purposes, for instance to extend it with an automatic beat pattern recognition interface [Gouyon et al., 2004], based on the command line tool Beat Root [Dixon, 2001a]. In addition to their plug-in interfaces, Audacity, Praat and WaveSurfer also provide label tracks to annotate sound files. However, while the integration of advanced features into these applications would benefit their users, only few means of generating these annotation are readily available in these solutions.

6.1.3 Research frameworks

Recently, a number of research group have made available software frameworks for processing audio signals are available. *CLAM* (C++ Library for Audio and Music) [Amatriain, 2004, Amatriain et al., 2001] provides an object oriented framework for the construction of graphical user interfaces around signal processing algorithms. Complex applications using different processing techniques have been constructed using this framework [Gómez et al., 2003a, Gouyon et al., 2004].

Another open-source software project, *Marsyas* (Music Analysis, Retrieval and Synthesis for Audio Signals), includes signal analysis routines, visualisation tools, and machine learning algorithms [Tzanetakis and Cook, 1999, Tzanetakis, 2002, Tzanetakis et al., 2002b], The Marsyas framework was used for different research experiments, such as beat tracking [Tzanetakis et al., 2002a] or musical genre classification [Tzanetakis and Cook, 2002]. Both systems are open-source and designed with flexibility in mind. Examples of real-time applications are provided, showing their ability to work in real-time. With visualisation functions and graphical elements, these systems are also constructed as self-consistent entities, in which the developer can build graphical applications writing a minimal amount of code and reusing existing elements.

6.1.4 Storage and exchange of meta-data

Establishing strong standards is important for the computer musician to facilitate the exchange of data across applications. A plethora of file formats and protocols has been developed in and around computer music applications. Lack of support for some of these formats can limit the usage of one or several applications. Common languages must be defined to convey musical signal and control parameters across different software environments. We can distinguish three main approaches in the design of these standards, according to their aim: storage formats, to store, compress and reuse musical data; transmission protocols, dedicated to exchange of data between applications; storage of annotation data, to facilitate the exchange of annotations across applications.

Storage

WAVE and AIFF formats are some of the most widely used formats to store raw audio data on hard disks. These formats are complex, and several variations exist, which complicate their implementation. In addition, several dozens of alternative formats can be encountered on computer music systems. To address the task of reading, writing and converting from and into these file formats, dedicated software libraries have been designed to access the signal data inside these files. For instance, *libsndfile* is used by numerous audio related open-source projects and supports more than a hundred different file formats [de Castro Lopo, 2006b]. Additionally, in the last decade, the use of compressed audio data has rapidly spread across the Internet, and consecrated with the success of peer-to-peer exchange software. Formats such as MPEG-I Layer 3 (MP3) [Brandenburg and Bosi, 1997] and Ogg Vorbis [Xiph.org, 2005] are now integrated in embedded audio systems such as in-car audio or portable devices. Similarly, dedicated software solutions exist to encode and decode these formats.

Transmission

To fulfil the demanding needs of diverse music creation processes, efforts have been made to standardise protocols for transparent communication across applications. One such standard is the Musical Instrument Digital Interface, MIDI, which in 1982 started being developed. The MIDI protocol provides a multi-track, polyphonic and real-time standard to store and exchange symbolic data, and has been rapidly adopted by musicians as a standard [Loy, 1985]. However, MIDI has also shown its limitations, notably with limited bandwidth, fixed time quantisation and finite number of control parameters [Moore, 1988].

The Open Sound Control (OSC) protocol was later developed and addresses most of these limitations [Wright and Freed, 1997]. OSC is designed for communication amongst computers, sound synthesisers and other multimedia devices. The protocol is based on modern networking technologies and following the URI (Universal Resource Identifier) scheme for naming conventions. Symbolic and numeric data can be sent across the network to different machines; features include high resolution time stamps, multiple recipients of a single message, possible grouping of simultaneous events. This protocol has already been widely adopted [Wright et al., 2003]; it is used for instance in SuperCollider (Section 6.1.1) to exchange data between the interpreter client and the sound synthesis server, and PureData externals (Section 6.1.2) for OSC support are available.

Annotation

In addition to MIDI, several formats have been designed to represent musical data. A text-based format derived from the LaTeX syntax was designed for Lilypond [Nieuwenhuizen et al., 1996], a program for typesetting sheet music. This format has been reused by several applications for graphical score editing – Denemo, Rosegarden, NoteEdit. MusicXML [Good et al., 2004] is another recent format dedicated to store musical scores in a portable and flexible way. The format, whose specifications are based on the eXtensible Meta Language (XML), is now supported by commercial applications – Finale, Sibelius – as well as open-source applications – Rosegarden, NoteEdit. To store complex and multi dimensional symbolic and numeric data, another format was required with the recent developments of complex musical applications. A standardisation effort has been made through the MPEG-7 specification [Manjunath et al., 2002], an ISO standard for the storage and exchange of multimedia meta-data. For interoperability and storage purposes, the MPEG-7 focuses on the issue of the storage format by defining a specific XML Document

Type Definition (DTD). The norm also provides test samples and code examples, to reproduce the extraction process with other implementations, from audio to MPEG-7 annotation files. Open standards such as Music XML and MPEG-7 are important to facilitate the storage and exchange of complex musical data over the network and across applications.

Accessing existing and new sound file formats is likely to be an ever going issue, which in itself is only remotely connected to the extraction of meta-data from raw audio signals. We have seen that several implementations and formats exist to address different applications. We have given an overview of a selection of applications, describing their concepts and some details on their implementation. A myriad of musical software applications are being designed, and not all fit all purposes. Users experiences could be enhanced with advanced functionalities based on the automatic extraction of annotations. Different software environments could benefit from advanced functionalities using a software solution designed for the robust extraction of annotations. Reciprocally, a centralised software solution could be enhanced by being integrated in various environments and used for different purposes.

6.2 Design requirements

Chafe et al. [1982] identify a number of "musical constructs" and possible ways of integration in software environments for the musician. These constructs include onset and beat location, pitch track and piano-roll scores. In the preceding chapters, we have discussed the difficulty of extracting some of these musical constructs from musical signal. Although piano-roll scores are commonly used to represent symbolic data such as MIDI, little software solutions exist to extract control and symbolic data from raw audio signals. The aim of our software library is to provide these functionalities to a large number of host application, and with a minimal number of requirements to ease their integration in existing software solution.

An approach well known to UNIX programmers is to design small programs to address a simple task. Different programs are then assembled together to form complex systems. This approach is described by Raymond [2003] as the "*rule of composition* : design programs to be connected with other programs". Open-source programming framework such as CLAM or Marsyas are interesting for research and educative purposes and allow rapid prototyping. However they are not always adapted for integration within other software environments, since they include elements of graphical interface. We have chosen to adopt this UNIX approach and create a system dedicated to the extraction of annotations, and designed to be integrated in other software platforms. Hence, we do not consider the tasks of accessing the raw signal and storing the extracted annotations as part of this software solution.

6.2.1 Automation and evaluation

As we target this software for its integration into other environments, we have to consider its role: on one side, the system aims at providing optimal results in a given context; on the other side, to test and improve the system, we must evaluate these results against manually annotated data. In parallel to the set of programs to extract annotations, evaluations routines should be deployed, in order to measure and discuss the results. This is also important to reproduce the benchmarks on diverse machine architectures.

The system can be separated in three folds: the extraction routines, the evaluation routines and the evaluation data. For several reasons, annotated sound files do not explicitly belong to the software package. Primarily, copyright issues on the sound file used prevent the redistribution of these files. Moreover, the code to extract and evaluate annotations is more likely to change than the annotated databases. Finally, the system should be able to run evaluation benchmarks on new databases, and access different meta-data storage formats.

We have hence identified two major parts in our software system. The first part, the extraction routines, should be optimised for efficiency and robustness. As such, these extraction routines can be seen as a "pipe" transforming audio signals into annotations. In order to facilitate its integration, this pipe should be designed with efficiency and generality in mind. Writing this part of the system in a low-level language such as C or C++ is important for the efficiency of the system. Minimising the dependencies of these routines against third-party softwares will also facilitate their integration.

Contrarily, the evaluation routines do not have such constraints, and although it can be advantageous to run benchmarks efficiently, there is no need to restrict its number of dependencies or to write it in a low-level language. The major constraint the evaluation stage must fulfil is to evaluate the extraction exactly as would be done from an external application. Access to each levels of the system is important for evaluation purposes, for instance to evaluate the influence of a parameter or the presence of pre-filtering. Therefore, the evaluation routines should be flexible and tightly binded to the extraction routines. However, both extraction and evaluation stages should be kept independent to ensure the correct behaviour of the extraction stage.

6.2.2 Guidelines

We give here guidelines for the implementation of a software system for automatic annotation of audio. Some of these considerations are conclusions derived from the precedent chapters. Others are more general recommendations for scalability, often derived from the UNIX philosophy [Raymond, 2003].

One of the most important feature for a software solution to be used in the most varied conditions is its portability: the system should run on common operating systems, such as Windows, Mac OS X and Linux, and on various processor architectures, for instance x86, Power-PC, or SPARC. This approach favours the use of well established coding standards, such as ANSI C, and discourages the use of architecture specific optimisations, such as assembly routines, which are optimised for a given processor and in most cases non-portable. This also means that the software should be independent of the different audio drivers used on each operating systems.

One of the most important aspect to consider when designing our system, which aims at transforming audio signals into semantics and control signals, is how its inputs and outputs will connect to other systems. As we want the annotation functions of our library to be embedded in other software environment, one of the goal of the software library is to remain reasonably small, and with limited dependencies: the system should be small and easily extensible. This constraint implies the software system input and outputs that are independent of the environment in which it is used: regardless of the audio input, which is also important for portability reasons, and regardless of the type of the output needed. For instance, the functions of the library should be usable on sound files and on live stream coming from a microphone. Moreover, it should be just as easy to create OSC packets from the results of the annotation functions than to output MIDI streams or store these results in the MPEG-7 file format.

So as to optimise and reuse algorithms, the different parameters of these algorithms should be accessible to the end user, as opposed to hard-coded values. Unlike an approach often adopted in the design of graphical interface, where some of the settings are hidden for clarity, we voluntarily follow an approach whereby all parameters are exposed and can be set by the user. This approach guarantees that the system is *programmable* and facilitates its integration into other systems.

We have seen that our software solution should be portable, by running on various hardware and operating systems, extensible, by reducing the scope of the software to their role and leaving support for various inputs and outputs outside the library, and programmable, by giving access to a maximum of parameters. In addition, a few more guidelines can be added as features of the system. These are not requirements but features which could benefit all implementations.

We have focused on the real-time extraction of annotation. In order to work in real-time, the algorithms should output the descriptors of a time-frame as soon as this time frame is over. In practice, we have algorithms that output a result within several milliseconds after this time-frame – 15 ms being already long for instrumental interactions. On the other hand, we have seen that predictive algorithms such as the tempo tracking algorithm in Chapter 4 could output the results before the corresponding event occurs in the signal. Indeed, the delay of the system is tightly dependent on the algorithm used. Regardless of this delay, the important constraint is to keep the system causal, so that live sound streams can be processed just as well as sound files. The causality is a warranty of limited memory usage of the algorithms, because no information has to be kept until the end of the sound file. For the same reasons of limited memory usage, the causality constraint also favours speed optimisations.

As discussed in Chapters 2, 3, 4 and 5, we have chosen to put a strong accent on the computation efficiency of the algorithm. In practice, we want to minimise the number of cycles spent to compute the results of a frame, which can be done not only in the design phase of the algorithms, but also in their implementation. For instance, constants used to compute a descriptor throughout the audio stream should be computed once, and only required computations should be executed over each time frame. Desktop computers are nowadays fast enough to compute several hundreds of operations within very short periods [Freed et al., 1993]. Because the system is targeted for integration into other environments, the constraint is not only for the algorithms to run on such computers, but to run fast in a minimum of processor cycles.

Another feature of the system is the reproducibility of the results. This is important for many environments. For instance, in live installations, the system should respond similarly to similar events, regardless of the time at which they occur. When annotating a database of songs, the analysis of two identical songs should result in the same annotations, even when one of them is preceded by a few seconds of silence. In practice, this time-invariance is not trivial to achieve, in particular for algorithms based on windows of samples, as opposed to sample-by-sample, which is the case of most of the algorithms we have studied.

6.3 Aubio: a library for audio labelling

We have just described our recommendations for the design of a software library dedicated to the automatic annotation of musical sounds. We have developed such an implementation which aims at addressing all of these recommendations. This library is named *aubio*, and is available online [Brossier, 2003] at the following address:

`http://aubio.piem.org/`

The aubio library aims at providing a set of high level music signal processing functions to use as building blocks for these user interfaces. Similarly to mathematical libraries, *libsamplerate* [de Castro Lopo, 2006a] for digital resampling, *FFTW* for fast Fourier transforms [Frigo, 1997], or graphical toolkits for the creation and manipulation of graphical elements, the *aubio* library provides tools for the detection and manipulation of high level audio objects, The aim of this library is not directly to provide user interfaces. Instead, annotations obtained using aubio are to be used in varied applications, graphical editor, musical search engines, or live audio effect.

For efficiency and portability, we have chosen to write the aubio library in ANSI C, a well established standard. Several music software environments are written in C, such as PureData or Max/MSP. Arguably, C++ code can also run efficiently on modern computers and has been adopted by various projects – SuperCollider, Marsyas, CLAM. In several cases, object-oriented programming languages are convenient for programming computer music applications [Roads, 1996]. Most notably the notion of processing and generator units are easily implemented with class inheritance mechanisms. However, memory management in C++ is partly automated, which makes it difficult to control the exact amount of memory occupied by the process. Conversely, all memory allocations and deallocations must be explicit in C, which may often lead to smaller memory usage than that of C++ programs. However, drawing general rules would be hasty, and the choice of language also reflects the preference of the author.

```
typedef struct _fvec_t fvec_t;  typedef struct _cvec_t cvec_t;
struct _fvec_t {                struct _cvec_t {
  int length;                     int length;
  int channels;                   int channels;
  float **data;                   float **norm;
};                                float **phas;
                                };
```

Figure 6.1: Definitions of the two elementary vector prototypes used in aubio. Real valued data are stored in an array of floats of size `channels` $\times$ `length`. Complex data are stored in two float arrays for norm and phase.

```
struct _aubio_task_t {
  fvec_t * state;
  uint_t threshold;
  uint_t parameter;
};
```

Figure 6.2: Example of structure prototype used to define object in the aubio library. The structure holds all the data required to execute a given task. Parameters and vectors are used to hold in memory past data and computation space.

Data structures and function prototypes

The two basic object types used in the aubio library to store vectors of complex and real valued data are listed in Figure 6.1. These two structures define multidimensional vector objects to manipulate data in both temporal and spectral domains. These objects fvec_t and cvec_t are used inside aubio to pass data between functions. The size and channel number are stored within the structures, so that the function can operate on objects of different sizes.

For each task defined in the aubio library, a structure is defined in the source file

```
typedef stuct _aubio_task_t aubio_task_t
aubio_task_t * new_aubio_task(int bufsize, int hopsize);
int aubio_task_set_param(aubio_task_t *o, float param);
void aubio_task_exec(aubio_task_t *o, fvec_t *in, fvec_t *out);
int del_aubio_task(aubio_task_t *o);
```

Figure 6.3: Definitions of typical function prototypes used in aubio to manipulate the structure described in Figure 6.2. Functions allows for the creation, execution, modification and deletion of a given routine.

typically as listed in Figure 6.2. This structure contains all the data and parameters required for the computation of the task and which needs to be stored across consecutive executions. Function prototypes to use this structure are defined in the corresponding header file, typically as listed in Figure 6.3. All function prototypes contain the word `aubio` in order to prevent name-space pollution. In the example of Figure 6.3, the execution function, `aubio_task_exec`, accepts input data in a vector of floats `fvec_t * in` and store the results in another vector `out`. Another function `aubio_task_set_param` can be used to change the parameter `task->param` after the object creation and between two consecutive executions. Functions to create and destruct an object are prefixed with `new` and `del`. All memory allocation and freeing operations take place respectively in these two functions, avoiding memory management calls during the execution of a task.

The programming interface we have chosen in *aubio* is a classic implementation of simple object models in C, used in several softwares, such as for instance PureData or some Csound implementations. Using this simple prototyping model, *stateful* objects can be created, executed, and deleted optimally. These objects behave like modules: simple modules operate simple tasks, such as Fourier transforms or filtering, and complex objects can be created by assembling several elementary modules into a new module.

Library organisation

Additional steps are followed to favour the modularity of these functions and simplify their prototypes. Firstly, not all operations demand the storage of temporary data or the allocation of additional memory space. For instance, computing the autocorrelation function or the exhaustive search for a maximum value in a vector are *stateless* operations. When possible and efficient, such functions are implemented without the use of a stateful structure. Most of these stateless functions are gathered into a collection of convenience routines and reused throughout the library.

Secondly, the main parts of a complex process are divided into smaller parts. For instance, the onset detection task consists of three modules: phase vocoder, onset detection function and peak picking. The output of the onset detection functions can be used as the input of the peak picking and the beat tracking routines. The output of the phase vocoder, which uses the FFT object, is also reused by spectral domain pitch detection routines. This would allow for instance to reuse spectral representations already computed by the host application, as is done for instance

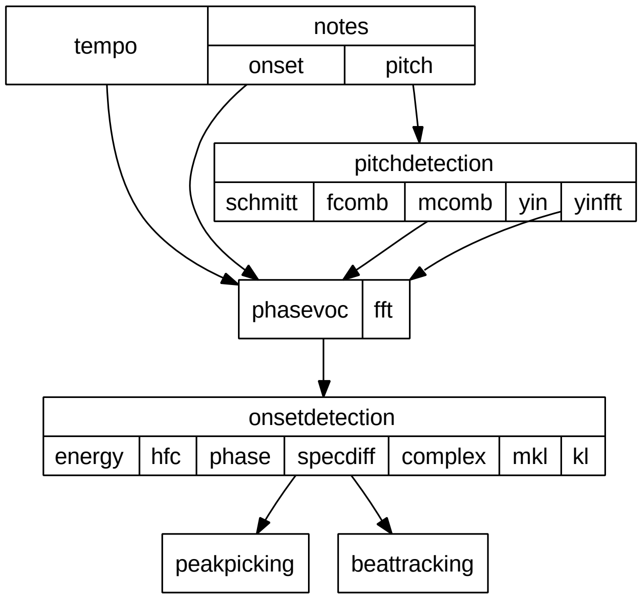


Figure 6.4: Diagram of the relations between different objects in the aubio library. Task objects tempo, notes, onset and pitch, reuse objects for pitch detection and onset detection functions.

in Marsyas when reading MPEG-I Layer 3 files (MP3), which are already stored as spectral data. Figure 6.4 shows a simplified diagram of different aubio modules.

Maintenance and extensions

These functions are to be used directly in C and C++ applications. Their prototypes have been chosen to be flexible yet efficient. Changing these prototypes would require changing accordingly the source code of programs using these functions. To prevent such binary incompatibility issues, the library is compiled with a version number, and multiple versions may coexist on the same system. To minimise the amount of code in the library, FFTW [Frigo, 1997] is used for FFT computations, optimised for different architectures and operating systems. libsndfile is used to read and write sound files in a portable way, and libsamplerate for efficient resampling of audio signals [de Castro Lopo, 2006a,b].

Programs written in C code may be efficient, but their maintenance requires rigour and costs compilation time. Interpreted languages minimise this cost and allow rapid prototyping. Several of these languages, such as for instance Perl or

Python, are commonly used in the research community. Other researchers might prefer compiled languages such as C++ or Java. To allow the integration of the *aubio* routines directly from languages other than C or C++, a description of the *aubio* application interface was written for the Simple Wrapper Interface Generator (SWIG) [Beazley et al., 1995]. This programs generates the files required to compile extensions for more than ten different languages such as Java, Perl and Python.

6.4 Integration examples

Along with the shared library of C routines described in Section 6.3, two types of command line tools are available: programs to analyse audio signals and extract annotations, and programs to evaluate the performance of the extraction algorithms against annotated databases. In addition, we have integrated the *aubio* library in a number of the applications reviewed in Section 6.1.2. We describe here how we integrated aubio in some of these graphical user interfaces: external for PureData, plug-ins for Audacity and WaveSurfer, integration into FreeCycle, CLAM Annotator and Sonic Visualiser. The complete source packages of these applications are available, as detailed in Appendix B.

6.4.1 Command line interface

The first series of command line is useful for the user to annotate one or multiple files. They come with a number of options and can be used from external applications. Each tool extracts one or more of the four descriptors we have described in the precedent chapters: onsets in Chapter 2, pitch in Chapter 3, tempo in Chapter 4 and notes in Chapter 5.

`aubiocut` outputs time stamps of note onsets in second on the console. When used with the option `--beat`, `aubiocut` attempts to detect beat locations instead. The buffer and hop sizes can be set with the parameters `--buffer` and `--hop`. The name of the sound file to analyse is passed with the parameter `--jack`. If no `--input` argument is found, the application attempts to launch the application in real time mode by connecting to the JACK audio server. In this mode, one audio input and one audio output are created. The input port receives the signal to be analysed, while the output ports plays the sound of wood-block each time an onset is detected in the input signal. Other interesting arguments include the `--threshold` option which is used to set the peak picking threshold parameters, the `--silence` parameter to set the silence gate, and the `--mode` function, used to select which detection function should be used. Functions currently available are: `complex` domain, `phase` based, spectral difference (`specdiff`), `kl`, `mkl`, `energy` and `dual`. A list of modes separated by commas can be passed to `--mode` to run several detection functions simultaneously. Another interesting option is `--plot`, which produces a plot of the sound sample, the detection function and the detected peaks. When used together with `--output`, the plot is saved in a file, either as a Post Script, a Portable Network Graphics (PNG), or a Scalable Vector Graphic (SVG) format, depending on the extension of the output passed as the argument to `--output` – virtually all terminals supported by Gnuplot are available [Williams and Kelley, 1986]. The plot in Figure 2.2 for instance was obtained using the following command line:

```
aubiocut --no-onsets --mode hfc,complex,specdiff,phase,mkl \
  --plot --input misterio_loop.wav --outplot misterio_loop.ps
```

Finally, when run with `--cut` a list of onset or beats is extracted in a first pass, then refined looking for local energy minima and zero crossing to slice at optimal locations. One file is created for each event detected with the name of the original file followed by the time stamp of the event in the file. The slices obtained are ready for use in software or hardware samples.

The program `aubiopitch` extracts pitch candidates from a sound file. The command line options available are similar to that of `aubiocut`, with `--buffer`, `--hop` and `--input`. The `--mode` option accepts here the following keywords: `fcomb`, `mcomb`, `schmitt`, `yin` and `yinfft`, which correspond to the different pitch algorithms evaluated in Chapter 3. Options specific to `aubiopitch` include `--units`, which accepts the arguments `midi` and `freq` and change the frequency unit of the output; `--maxpitch` and `--minpitch` determine the range of frequency out of which no frequency estimates should be found. Similarly as for `aubioonset`, `aubiopitch` supports multiple arguments for `--mode`, separated by commas, and the `--plot` option. The plot in Figure 3.7 was obtained using the following command line:

```
aubiopitch --mode schmitt,fcomb,mcomb,yin,yinfft \
  --minpitch 277 --maxpitch 880 --plot \
  --input opera_fem2REF.wav --outplot opera_fem2REF.ps
```

These command line programs can also be used to play the results in real time using the JACK audio layer, a low-latency audio server for Linux and Mac OS X, and could be extended to output sound to other audio drivers.

```
#! /usr/bin/python
from aubio.bench import bench
from aubio.tasks import onset
mybench = bench(onset,'/path/to/annotated/onsets/')
mybench()
```

Figure 6.5: Example of Python script used to evaluate the performance of the onset extraction routine with default parameters. The `bench` class runs the `onset` task class on every file found in the specified directory.

While several functionalities are available from these two programs, `aubiocut` and `aubiopitch`, several other programs are available in the source code. Additionally, these examples can be used to write custom programs.

6.4.2 Evaluation interface

The second series of programs included in aubio are the ones used to run the benchmarks whose results were described in previous chapters. They can be used to evaluate the influence of different processing steps and the performance of new extraction methods.

Python [van Rossum et al., 1991] is an object oriented language with a syntax similar to C. We have chosen this language to write the evaluation scripts. Each module written in C has its own corresponding object in Python. The different extraction routines are gathered along a class template, which defines the initialisation and parametrisation steps, the execution, and the evaluation. Additionally, functions to plot the results are written using the Python interface to Gnuplot [Williams and Kelley, 1986]. In parallel of these tasks, another type of class is derived to run benchmark over a collection of annotated files. A benchmark can thus be written as in Figure 6.5. This example is a complete program which evaluates the performance of the onset extraction task using default parameters. Here, the call to `mybench` first runs `dir_exec` to extract the features from sound files; then `dir_eval` to evaluate this data against manual annotations; finally, the results of the computation are printed on the console and plotted. When needed, each function of the template `bench` class can be redefined to operate differently, for instance to change the order of the executions or the way evaluated data are gathered. Similarly, for each tasks, pre-defined extraction and evaluation routine can also be changed, for instance to extract the data using an external command line, to read the ground truth from another file format, or to alter the default plotting function.

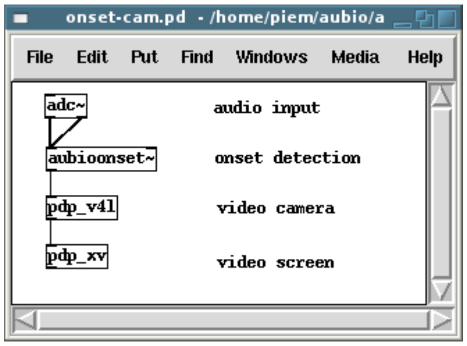


Figure 6.6: Using aubioonset~ in PureData to drive a video camera. The microphone adc~ sends signal to the onset detector aubioonset~, which in turns sends a bang to the camera pdp_v4l. Each picture taken is sent to the pdp_xv object and displayed on screen.

Several programs based on the python interface were used to generate some experimental results and figures included in the previous chapters. In Chapter 2, the results of the onset evaluation (Figure 2.10) were created using `bench-onset`. The pitch evaluation on the monophonic and polyphonic database in Chapter 3 were done by `bench-pitch` program, while the results of the pitch evaluation in Figure 3.9 were created by the program `bench-pitch-isolated`. To facilitate the reproduction of the results we have discussed in these chapters, the complete source code used to run the evaluation benchmarks is available along the C library in the aubio source package, as detailed in Appendix B.

6.4.3 PureData objects

The main functionalities of aubio have been binded to the PureData programming environment [Puckette, 1996a,b]. A PureData external – or plug-in – is included in the aubio source tree and gives access to different objects: `aubioonset~` for onset detection, `aubiotempo~` for tempo tracking, `aubiopitch~` for pitch detection, and `aubiotss~` for transient vs. steady-state separation. The source code of these objects are simple wrappers around the aubio functions, written in C.

Simple yet powerful applications can be built within PureData. Using the `pdp` external, audio signal and control objects can be combined with video capture, display and processing objects. Figure 6.6 shows a screen-shot of a PureData patch using the `pdp` video externals [Schouten et al., 2002] and aubio. The onset detection object, `aubioonset~`, processes sound from the sound-card input, `adc~` (Audio-to-

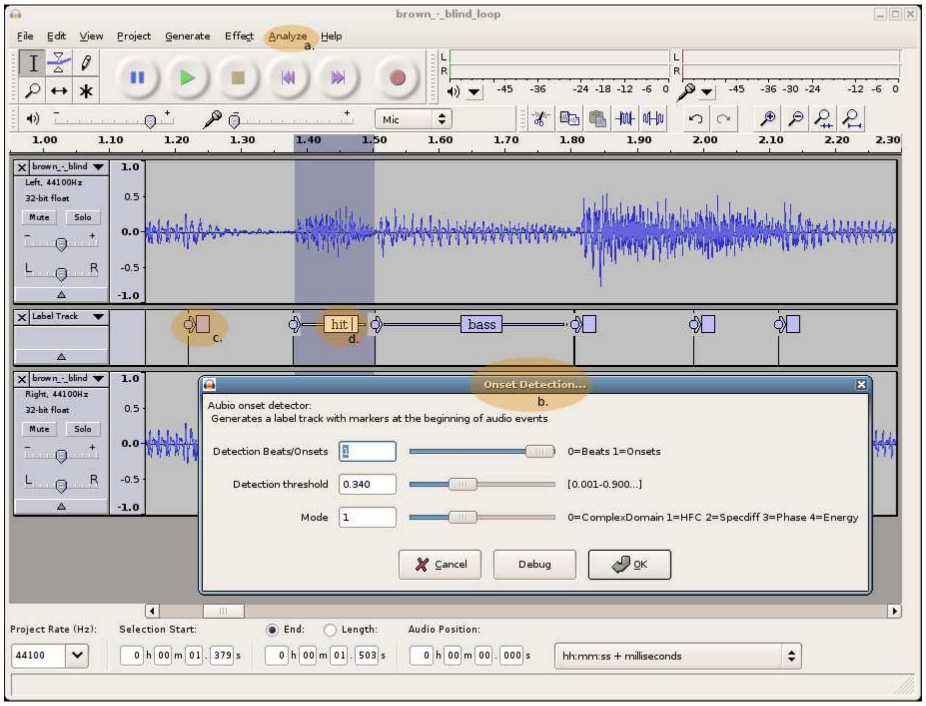
Digital Converter). Each time an onset is detected, `aubioonset~` sends a control message, `bang`, to the video camera `pdp_v4l`, which in turn takes a picture from a video device. The image is then displayed on the screen, `pdp_v4l`.

This simple patch takes a picture from a video camera each time an onset is detected on the microphone input. The first impression created by this system is often that of camera taking shots at random times, but when users start understanding how the camera is controlled, the installation can be amusing. An interesting artefact occurs when the onsets are triggered by the light switch of a room: after switching the light off, the screen displays an image of the room in the light; when switching the light back on, the screen displays the black pitched room. This is explained the delay of the camera, which can be expected to be longer than 30 ms at 30 image per second, and even longer when white balancing and image averaging is used to stabilise the image.

Several audio and video applications could be built using the different objects provided by aubio. More functionalities are to be added, including the possibility of changing analysis method, and the addition of several parameters. At this stage, these objects are already useful to test the performance of these functions and listen to their results in real time. The system was tested on a Pentium III Mobile 800 MHz: running simultaneously `aubiopitch~` (yinfft), `aubioonset~` (complex), and `aubiotrack~`, the processor load was about 55%, leaving enough processing power to draw detected features on the screen.

6.4.4 Audacity and WaveSurfer plug-ins

Audacity and WaveSurfer are advanced graphical audio editors with features such as track labels. Using the plug-ins extensions of these two software packages, we have added support for the onset and tempo detection. The Nyquist analysis plug-in written for Audacity works as follows: first, it saves the current sound selection to a temporary file, then runs `aubiocut` on this file. The time-stamps printed by `aubiocut` automatically generate a label track. The graphical interface of the Nyquist plug-in and an example of label track are shown in the Audacity screen-shot of Figure 6.7. Minor modifications to the Nyquist languages were required to add support for file saving and external commands. A similar approach was adopted to implement an aubio plug-in in WaveSurfer. The WaveSurfer plug-in architecture is written in Tk, a cross-platform graphical toolkit based on the Tcl scripting language, and provides functions for writing to disk and executing external commands. These implementations are not optimal, especially for very long files, since they require



Aubio onset detector:
Generates a label track with markers at the beginning of audio events

Detection Beats/Onsets 1 0=Beats 1=Onsets
Detection threshold 0.340 [0.001-0.900...]
Mode 1 0=ComplexDomain 1=HFC 2=Specdiff 3=Phase 4=Energy

Project Rate (Hz): 44100
Selection Start: 0 h 00 m 01 . 379 s
End: 0 h 00 m 01 . 503 s
Audio Position: 0 h 00 m 00 . 000 s
hh:mm:ss + milliseconds

Figure 6.7: Using `aubioonset` as an Audacity plug-in. a. Analysis menu where the aubio plug-in appears. b. Aubio onset detection settings dialog box. c. Label track automatically generated and marking detected onsets. d. Manually annotated label.

saving the file to disk. A direct integration of the aubio functions into the Nyquist or Tk language would require slightly more code maintenance, but increase the performance.

6.4.5 Freecycle

Freecycle is a graphical interface to segment audio files at onsets and edit these segments. All of the onset extraction functions included in aubio are accessible via the configuration menu. The threshold for the peak picking of the detection function can be set using the horizontal slider next to the transport control buttons. The toolbar at the top of the window also contains various informations about the current sample, including the detected BPM. A time envelope can be applied on each extracted segments, changing the attack, decay, sustain and release parts of the segments. The main waveform display shows the location of the extracted

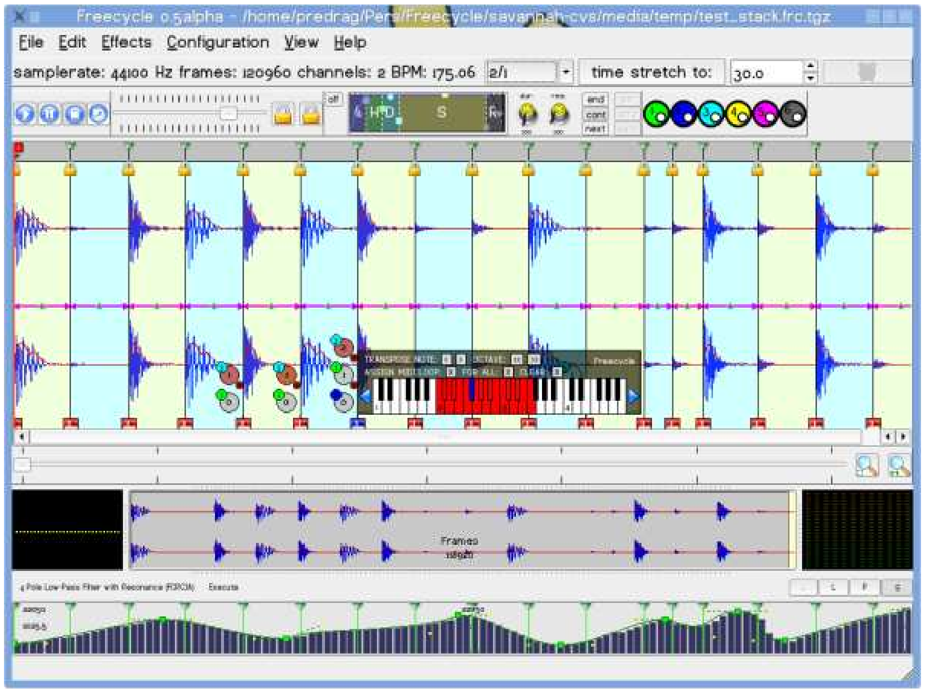
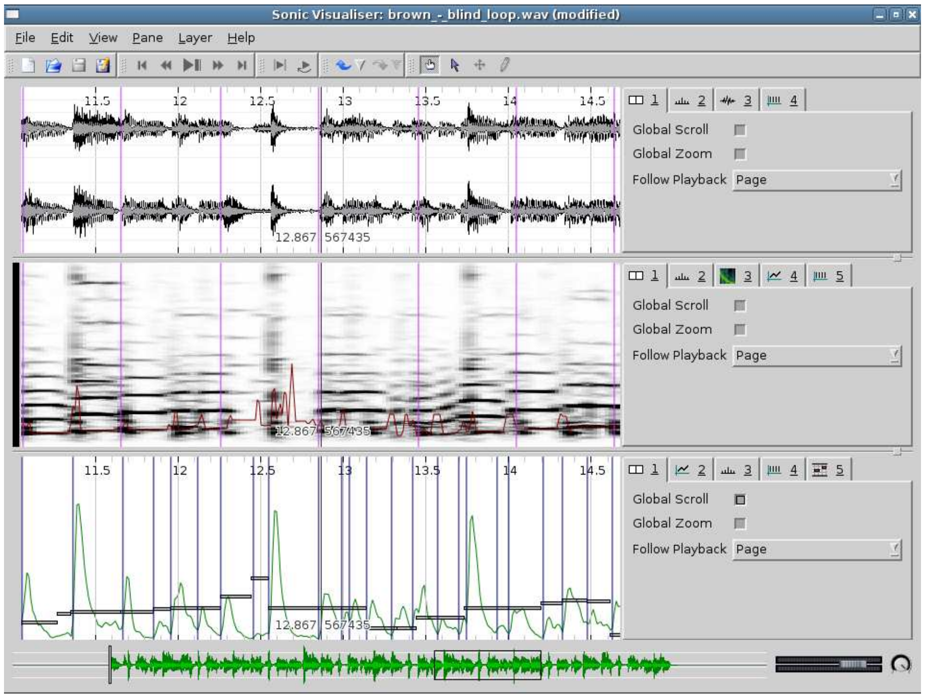

Freecycle 0.5alpha - /home/predrag/Pers/Freecycle/savannah-cvs/media/temp/test_stack.frc.tgz
File Edit Effects Configuration View Help
samplerate: 44100 Hz frames: 120960 channels: 2 BPM: 175.06
time stretch to: 30.0

Figure 6.8: Screen-shot of Freecycle [Viceic, 2006] showing a waveform sliced at detected onset times. Different onset detection function can be selected from the configuration menu, and the toolbar contains a slider to set the peak picking threshold.

onsets; each can be moved along the time-line, or locked to prevent future edition using the small lock symbols at the top of the label lines. The segments can also be swapped in different combinations, and each combinations can be stored in one of six configurations. Pressing the numbered colour circles in the toolbar recall one of each configuration. At the bottom of these lines is found the keyboard, which can be used to assign each segment to a range of MIDI values. Freecycle can also export the segmented sound in a number of formats, including SoundFount and AKP, the AKAI Sample Format used in hardware samplers.

6.4.6 Sonic Visualiser and CLAM Annotator

Sonic Visualiser is an application for viewing and analysing the content of sound files developed recently at the Centre for Digital Music [Cannam et al., 2006]. The aim of this graphical interface is to give the user visual and textual informations

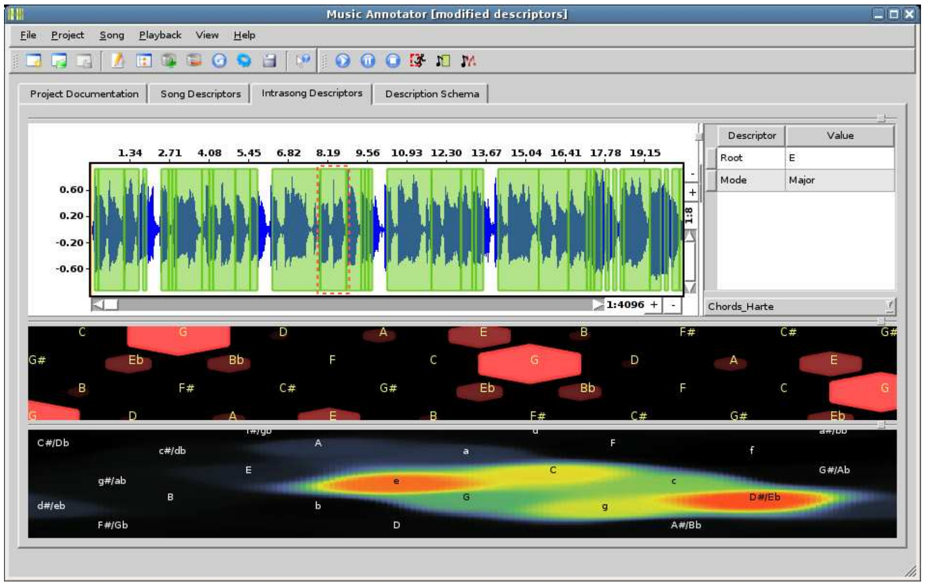

Sonic Visualiser: brown_-_blind_loop.wav (modified)
File Edit View Pane Layer Help
11.5 12 12.5 13 13.5 14 14.5
12.867 567435

Global Scroll
Global Zoom
Follow Playback Page

Figure 6.9: Screen-shot of Sonic Visualiser [Cannam et al., 2006], with three different panes displayed. Each pane contain several layers. Top: waveform and beat tracking layers. Middle: spectrum view, beat locations and pitch layer. Bottom: onset times, onset detection and notes.

about music signals. Obviously, this type of interface is interesting to visualise the results of our annotation routines. In a way similar to that of WaveSurfer, the interface supports multiple panes. Additionally, each pane may display several layers of data, representing different data in different ways. Transparency is used to visualise multiple layers at a time. In addition to built-in functions optimised for efficient representation of time and spectral domain data, feature extraction can be done using a specific plug-in interface, VAMP. A particularity of this interface is its support for multidimensional data and labels, which permits the use of complex meta-data formats within Sonic Visualiser.

To use the aubio features directly from Sonic Visualiser, several VAMP plug-ins were written that extract annotations using the aubio routines: onset detection, onset detection functions, pitch tracking, beat detection and notes modelling can now be used directly as Sonic Visualiser layers. Parameters for different algorithms



Descriptor	Value
Root	E
Mode	Major

Figure 6.10: Screen-shot of the CLAM Annotator, a graphical application to visualise pre-extracted features. In this configuration, segments of detected chords are overlayed on the waveform display (top), and two chord visualisation modules show the tonality probability of the piece (bottom).

are to be adjusted in a dialog box, containing drop-down lists of different onset or pitch algorithms, peak-picking and silence thresholds, maximum and minimum pitch values. An example of a Sonic Visualiser screen-shot is displayed in Figure 6.9. Towards the top of the window, the first pane contains the default layer with a waveform display, and an additional layer to display the extracted beat locations. The middle pane contains a spectrum layer, the beat locations and the pitch detection output. The last pane at the bottom of the window shows detected onsets, onset detection functions and extracted notes.

The CLAM Annotator is another application to visualise extracted features and edit annotation data [Gouyon et al., 2004]. The application, built using the CLAM framework, proposes advanced functionalities to annotate and visualise music sound files. A screen shot of the CLAM Annotator graphical user interface is shown Figure 6.10.

6.5 Summary and perspectives

We have given an overview different computer music environments, illustrated by a number of computer music software applications and formats. The overview aimed at highlighting different approaches in the design of these environments: programming languages dedicated to analysis and synthesis of audio signals, graphical user interfaces designed to facilitate the manipulation of music signals, or storage formats and protocols to store or exchange musical data.

We have then described the design of the *aubio* library, a system for automatic extraction of high-level features from audio streams. The library contains routines for the extraction of audio descriptors on one hand, and scripts for the evaluation of these descriptors on the other hand. The extraction routines were designed in a causal fashion to facilitate their integration not only in real-time systems, but also in various user interfaces. Describing implementation details is somewhat difficult, and the reader is invited to consult the source code of the library to understand how the different modules are constructed and assembled. The complete source code of the *aubio* library is available on the project website [Brossier, 2003] – see details in Appendix B. A detailed documentation is also provided, generated using Doxygen [van Heesch et al., 1997] from the comments included in the source code.

The *aubio* library has been integrated with new and existing software. Command line tools are provided to extract annotations; results can be played or printed on the standard output, to be stored in text files or further processed with other tools reading from standard input. Scripts for the evaluation of the different descripors are also provided so that users can easily reproduce the results discussed in this thesis, including the graphs to present these results. The different routines have been deployed in the PureData programming environment. Writing a PureData external for aubio was an important test for the library, since its requirements are typical of a real time environment. The announcement of these objects was made on the PureData mailing lists, and several users reported using them. The integration of *aubio* into graphical user interfaces has been described. Audio editors such as Audacity, Wavesurfer or Freecycle can benefit from the library functionnalities, such as automatic segmentation at onset and beat locations. The routines are also accessible from applications specifically designed to visualise audio annotations, such as Sonic Visualiser and CLAM Annotator. We hope that the *aubio* library will continue to grow and improve in the future, and become a widely adopted solution to integrate advanced annotation featuress into various music software environments.

Chapter 7

Conclusion

We have reviewed a number of existing and novel approaches to annotate musical audio streams, implemented some of these approaches in a real time fashion, and evaluated these implementation against several databases of real recordings. In Chapter 1, we gave an overview of some of the characteristics of the human listening system and we reviewed a number of techniques to process audio signals.

Chapter 2 included the review of several methods for the extraction of onset locations, the beginning of sound events. Several onset detection functions were implemented and a modified peak selection technique was designed to allow the selection of the onset times within a minimal delay. The robustness of these functions was evaluated against a large database of sounds.

Chapter 3 included a definition of the pitch, the perceptual attribute associated to frequency, and described a number of approaches to extract the fundamental frequency from audio signals. Five different method, including a novel spectral domain modified YIN, were evaluated on databases of isolated notes, polyphonic recordings and monophonic recordings. Our novel approach gave the best overall pitch accuracy, with a computational cost suitable for real time operation.

In Chapter 4, we reviewed different approaches to the extraction of tempo beats in musical audio, and we described the details of our real time implementation of a two-state causal beat tracking approach. The results of this method were compared to other techniques on two different databases of manually annotated sounds.

We looked at the issue of extracting MIDI like notes in Chapter 5, where an overview of different techniques are given, and two new methods for causal extraction of these notes are proposed. Preliminary experiments were made on sound files generated from MIDI scores, and the results of our approach are discussed in terms of performance and latency.

In Chapter 6, we described several types of computer music environments, and our software implementation, the *aubio* library, designed to operate in a real time fashion. Several examples of software using this implementation were described. We hope that the aubio library will continue to grow and improve in a near future.

Appendix A

Experiments details

To facilitate the reproduction of the experimental results discussed in this document, the list of files used during the experiments of Chapter 2 and Chapter 3 are gathered in this appendix. Additional details of the experimental results are also included.

A.1 Temporal segmentation

The name of the sound files used for the onset detection experiments of Chapter 2 are listed in Table A.1. For each sound file, three or more text files are used to store the hand annotations.

A.2 Pitch analysis

The lists of files used in the three databases of Chapter 3 are included below. In Table A.2 are listed the details of the files extracted from the RWC instruments database [Goto et al., 2003]. The files were first automatically splitted using aubiocut (Section 6.4.1), and the slices obtained renamed to the expected note name. All sound files were then played and compared across instruments and playing modes. Table A.3 and Table A.4 list the files of the monophonic and polyphonic databases obtained from the MIREX Audio Melody Extraction contest [MIREX, 2004b]. Text files are used to store the result corresponding to each sounds.

Categories	*solo drums*	*solo brass*	*poly pitched*
Filenames	acoustic10	sax	celesta
	acoustic11	trumpet	distorted_guitar
	acoustic12		guitar1
	acoustic1	*solo plucked strings*	guitar2
	acoustic2	berimbau	harpsichord1
	acoustic3	double_bass_pizz1	harpsichord2
	acoustic4	double_bass_pizz2	harp
	acoustic5	electric_bass	piano1
	acoustic6	guitar	piano2
	acoustic7	harpsichord	vibraphone
	acoustic8	piano	
	acoustic9	sitar	*complex*
	agogos	synth_bass	classic1
	castanets		classic2
	claves	*solo singing voice*	classic3
	electronic1	alto_voice	jazz1
	electronic2	bass_voice	jazz2
	electronic3	soprano_voice1	jazz3
	electronic4	soprano_voice2	pop1
	hi_hat	tenor_voice	pop2
	java_gourd		pop3
	kick	*solo sustained strings*	pop4
	ped_hat	cello1	techno1
	pig_drum	cello2	techno2
	shaker	double_bass	world1
	surdo	viola	world2
	tabla	violin1	world3
	talking_drum	violin2	
	timbales		
	udu	*solo winds*	
		accordeon	
	solo bars and bells	clarinet	
	bells	flute	
	glockenspiel	oboe	
	vibraphone		
	xylophone		

Table A.1: List of files used for the evaluation of onset detection algorithms [MIREX, 2005b]. Each file is annotated by 3 or 5 different listeners. The total number of annotation is shown in Table 2.1.

Instruments	piano	elecguitar	vibraphone	rhodes	clavinet
Filenames	011pfnof	131eglff	041vihnf	021epnof	023cvnof
	011pfpef	131eglfm	041vihpf	021eppef	023cvnom
	011pfref	131eglfp	041vihvf	022epnof	023cvstf
	011pfstf	131eglpf	041visdf	022eppef	023cvstm
	012pfnof	131eglpm	041visnf		
	012pfpef	131eglpp	041visvf		
	012pfref		042vihnf		
	012pfstf		042vihpf		
	013pfnof		042vihvf		
	013pfpef		042visdf		
	013pfref		042visnf		
	013pfstf		042visvf		

Table A.2: List of files from the RWC database [Goto et al., 2003] from which isolated notes were extracted. Total number of notes are given in Table 3.1, experiments results in Figure 3.9.

Group	pop	midi	daisy	opera	jazz
Filenames	pop1REF	midi1REF	daisy1REF	opera_fem2REF	jazz1REF
	pop2REF	midi2REF	daisy2REF	opera_fem4REF	jazz2REF
	pop3REF	midi3REF	daisy3REF	opera_male3REF	jazz3REF
	pop4REF	midi4REF	daisy4REF	opera_male5REF	jazz4REF

Table A.3: List of files from the MIREX [2004b] melody extraction contest, used for experiments on monophonic data in Chapter 3; files durations are listed in Table 3.2, experiment results in Table 3.3.

Group	pop	midi	daisy	opera	jazz
Filenames	pop1	midi1	daisy1	opera_fem2	jazz1
	pop2	midi2	daisy2	opera_fem4	jazz2
	pop3	midi3	daisy3	opera_male3	jazz3
	pop4	midi4	daisy4	opera_male5	jazz4

Table A.4: List of files from the MIREX [2004b] melody extraction contest used for experiments on polyphonic data in Chapter 3; files durations are listed in Table 3.2, experiment results in Table 3.4.

	schmitt			mcomb			fcomb			yin			yinfft		
	S	P	C	S	P	C	S	P	C	S	P	C	S	P	C
daisy	95.07	91.13	92.07	94.33	95.51	95.55	96.36	95.33	95.53	94.40	94.10	94.12	94.47	95.84	95.87
jazz	93.38	91.04	91.16	92.35	95.01	95.23	92.35	94.49	94.83	92.45	93.70	94.19	92.45	94.62	95.03
midi	75.71	69.38	69.67	75.14	94.09	94.49	75.14	91.23	92.75	75.14	90.26	90.78	75.14	93.68	94.05
opera	70.05	23.37	27.97	69.11	52.01	53.93	70.70	48.99	51.77	69.17	37.11	39.80	69.11	56.06	57.04
pop	75.53	71.38	73.18	74.53	82.83	83.07	88.11	79.03	80.51	74.53	79.80	80.02	74.62	85.21	85.39
Total	81.33	68.46	70.03	80.41	83.89	84.45	85.79	81.71	83.02	80.46	78.91	79.69	80.49	85.09	85.48

Table A.5: Detailed results for the experiment on the monophonic database with voice/unvoiced (S), raw pitch accuracy (P) and chroma accuracy (C); a description of the database can be found in Table A.3.

	schmitt			mcomb			fcomb			yin			yinfft		
	S	P	C	S	P	C	S	P	C	S	P	C	S	P	C
daisy	3.04	12.43	15.84	0.00	80.76	83.24	0.00	76.80	83.04	0.20	73.12	79.60	0.00	72.42	79.43
jazz	10.42	7.42	13.68	2.77	60.64	63.68	0.38	54.18	58.36	2.01	61.74	71.43	0.98	70.31	75.69
midi	28.15	3.17	7.73	15.10	38.09	43.17	14.53	46.91	53.10	15.89	47.20	56.54	14.98	46.46	56.29
opera	20.08	7.40	10.12	19.03	39.63	44.17	19.08	35.97	41.94	19.20	16.00	23.52	19.14	49.12	51.85
pop	14.81	4.85	9.61	5.22	23.88	28.36	4.15	16.24	22.72	3.84	32.47	40.45	4.34	62.42	67.29
Total	14.38	6.94	11.18	7.42	47.72	51.73	6.51	45.75	51.67	6.92	45.57	53.72	6.76	59.25	65.40

Table A.6: Detailed results for the experiment on the polyphonic database with voice/unvoiced (S), raw pitch accuracy (P) and chroma accuracy (C); a description of the database can be found in Table A.4.

A.3 Tempo tracking

The list of files used for the experiments on tempo tracking described in Chapter 4 and whose results were given in Table 4.1 are listed in Table A.7. This database was gathered by Hainsworth [2004]. Note that the details of the files used for the MIREX Audio Tempo Extraction Contest, whose results were listed in Table 4.2, are unknown as they were kept secret to the participants.

A.4 Notes modeling

The list of scores used for the annotation of our note modelling approach in Chapter 5 is given. The corresponding scores can be found on the Mutopia Project website [Mutopia project, 2000].

Six Partitas (Clavierubung part I), No. 1 by J. S. Bach (1685-1750), BWV 825, for Harpsichord, Piano, or Clavichord, 1731. Public domain.

Six Partitas (Clavierubung part I), No. 2 by J. S. Bach (1685-1750), BWV 826, for Harpsichord, Piano, or Clavichord, 1731. Public domain.

Six Partitas (Clavierubung part I), No. 5 by J. S. Bach (1685-1750), BWV 829, for Harpsichord, Piano, or Clavichord, 1731. Public domain.

Partita I for Solo Violin by J. S. Bach (1685-1750), BWV 1002, 1720. Public domain.

Octet by F. Mendelssohn-Bartholdy (1809-1847), Op. 20 for Violin, Viola and Cello, 1825. Creative Commons Attribution-ShareAlike 2.5.

Ein Sommernachtstraum - No.5 by F. Mendelssohn-Bartholdy (1809-1847), Op. 61, for Orchestra: Flute, Oboe, Clarinet, Bassoon, French Horn, Violins, Viola, Cello, and Double Bass. Public domain.

Symphony nr. 18 in F major K. 130 by W. A. Mozart (1756-1791), KV 130, for Orchestra: Flutes, Horns, Violins, Viola, Cello and Bass, May 1772. Public Domain.

Funftes Quartett, Movements 1 and 2 by W. A. Mozart (1756-1791), KV 158, for String Quartet, 1773. Creative Commons Attribution-ShareAlike 2.5.

Concerto for Bassoon and Orchestra in B flat major by W. A. Mozart (1756-1791), KV 191, for Bassoon and Orchestra: Horns, Oboes, Violins, Viola, Cello and Bass, 1774. Public domain.

Category	Class.	Solo Cl.	Jazz	60's Pop	Rock	Dance	Folk
Filename	035	007	002	038	001	015	056
	036	121	003	041	010	016	057
	037	122	008	048	011	017	058
	152	123	009	079	022	018	059
	188	124	076	080	023	019	060
	190	125	077	081	024	020	061
	193	126	078	082	025	021	062
	194	127	102	083	045	039	063
	195	128	103	084	046	040	064
	196	129	104	085	047	042	065
	201	130	106	086	049	043	066
	202	131	107	087	050	146	067
	205	132	108	088	051	147	068
	207	133	110	089	137	153	069
	208	134	111	090	138	154	070
	209	135	112	091	139	155	071
	210	144	113	092	140	156	072
	211	145	114	093	141	157	075
	240	218	115	094	143	158	
	241	219	117	095	150	165	Other
	242	220	118	096	151	166	073
	243	221	119	097	159	167	074
	244		120	098	160	168	224
		Big Band	212	099	161	169	225
	Choral	052	213	100	162	170	
	006	053	214	163	173	171	
	012	054	215	164	226	172	
	013	055	216	181	227	174	
	189	148	217	182	232	175	
	191	149	222	183	233	176	
	206	197	223	184		177	
	245	198		185		178	
		200		186		179	
				187		180	
				228		234	
				229		235	
				230		236	
				231		237	
						238	
						239	

Table A.7: List of files from the tempo extraction database used for the experiments in Chapter 4, with details of the different categories: Classical, Choral, Solo classical, Big Band, Jazz, 1960's Pop, Pop/Rock, Dance, Folk, and Other. Complete database details are found in [Hainsworth, 2004].

Appendix B

Additional material

These pages briefly describe the contents of the CD-ROM accompanying this document. A copy of this material is available at `http://aubio.piem.org/phdthesis/`.

B.1 Aubio source code and documentation

The `aubio/` directory contains the current version of the complete aubio source code (`aubio/pub/`), and the documentation for the project (`aubio/doc/`), generated using Doxygen [van Heesch et al., 1997]. The source code is in a compressed TAR file, and the documentation is in the HTML format. For other revisions of the aubio library, please see `http://aubio.piem.org`.

B.2 Sound examples

The `examples/` directory contains audio examples illustrating different results obtained with aubio. They are organised as follows:

- `onset/` Examples of click tracks obtained in real-time using our implementation are available in this directory. Examples obtained with `bonk~` [Puckette et al., 1998] are also provided for comparison. The sounds used in Chapter 2 are also included.

- `tempo/` Beat tracking examples obtained using `aubiotrack` on different recordings.

- `notes/` Examples of MIDI notes output obtained using `aubionotes` on different sounds.

B.3 Thesis document

The `thesis/` directory contains the present document in its electronic version. Note that this version, in the Portable Document Format (PDF), contains internal and external hyperlinks. Colour graphics were prepared to print correctly on a white and black printer.

List of Figures

List of Tables

Bibliography

Samer Abdallah and Mark D. Plumbley. Unsupervised onset detection: a probabilistic approach using ICA and a hidden Markov classifier. In *Cambridge Music Processing Colloquium*, Cambridge, UK, 2003.

Paul E. Allen and Roger B. Dannenberg. Tracking musical beats in real time. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 140–143, University of Glasgow, UK, 1990.

Xavier Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, October 2004.

Xavier Amatriain, Pau Arumi, David Garcia, Ismael Mosquera, et al. Clam, C++ Library for Audio and Music. `http://clam.iua.upf.edu/`, 2001. Last visited: Fri June 2 2006.

Xavier Amatrian and Perfecto Herrera. Transmitting audio content as sound objects. In *Proceedings of AES 22nd International Conference on Virtual Synthetic and Entertainment Audio*, pages 278–288, Espoo, Finland, 2002.

Jean-Julien Aucouturier and François Pachet. Ringomatic: A real-time interactive drummer using constraint-satisfaction and drum sound descriptors. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 412–419, London, UK, September 2005.

Jean-Julien Aucouturier, François Pachet, and Peter Hanappe. From sound sampling to song sampling. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 1–9, Barcelona, Spain, October 2004.

Dave Beazley et al. SWIG, Simplified Wrapper and Interface Generator. `http://www.swig.org/`, 1995. Last visited: Fri June 2 2006.

Juan-Pablo Bello. *Towards the Automated Analysis of Simple Polyphonic Music*. PhD thesis, Centre for Digital Music, Queen Mary University of London, London, UK, 2003.

Juan-Pablo Bello and Jeremy Pickens. A robust mid-level representation for harmonic content in music signals. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 304–311, London, UK, September 2005.

Juan-Pablo Bello, Laurent Daudet, and Mark B. Sandler. Time-domain polyphonic transcription using self-generating databases. In *Proceedings of the Audio Engeeniring Society 112th Convention*, Munich, Germany, May 2002.

Juan-Pablo Bello, Mike P. Davies, and Mark B. Sandler. Phase-based note onset detection for music signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 441–444, Hong-Kong, 2003.

Juan-Pablo Bello, Laurent Daudet, Samer Abdallah, Christopher Duxbury, Mike P. Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions in Speech and Audio Processing*, 13(5):1035–1047, September 2005.

Jonas Beskow, Kåre Sjölander, et al. Wavesurfer, a sound manipulation program. `http://www.speech.kth.se/wavesurfer/`, 2000. Last visited: Fri June 2 2006.

Paul Boersma, David Weeninki, et al. Praat, a program for speech analysis and synthesis. `http://www.praat.org/`, 1992. Last visited: Fri June 2 2006.

Richard Boulanger. *The Csound Book*. MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-52261-6.

Karlheinz Brandenburg. MP3 and AAC explained. In *Proceedings of the Audio Engineering Society 17th International Conference on High Quality Audio Coding*, Florence, Italy, 1999.

Karlheinz Brandenburg and M. Bosi. Overview of MPEG audio: Current and future standards for low bit rate audio coding. *Journal of the Audio Engineering Society*, 45(1/2):4–21, 1997.

Albert S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. MIT Press, Cambrigde, Massachusetts, 1990. ISBN 0-262-52195-4.

Paul M. Brossier. Aubio, a library for audio labelling. `http://aubio.piem.org`, 2003. Last visited: Fri June 2 2006.

Paul M. Brossier, Juan-Pablo Bello, and Mark D. Plumbley. Fast labelling of notes in music signals. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 331–336, Barcelona, Spain, October 2004a.

Paul M. Brossier, Juan-Pablo Bello, and Mark D. Plumbley. Real-time temporal segmentation of note objects in music signals. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 458–461, Miami, Florida, USA, November 2004b.

Judith C. Brown. Determination of the meter of musical scores by autocorrelation. *Journal of the Acoustical Society of America*, 94(4):1953–1957, 1993.

Carlos Caires. Irin: Micromontage in graphical sound editing and mixing tool. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 219–222, Miami, Florida, USA, November 2004.

Chris Cannam et al. Sonic visualiser, an application for viewing and analysing the content of music audio files. `http://www.sonicvisualiser.org/`, 2006. Last visited: Fri June 2 2006.

Pedro Cano. Fundamental frequency estimation in the SMS analysis. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-98)*, pages 99–102, Barcelona, Spain, 1998.

Michael A. Casey. Acoustic lexemes for organizing internet audio. *Contemporary Music Review*, 24(6):489–508, December 2005.

Michael A. Casey. Understanding musical sounds with forward models and physical models. *Connection Science*, 6(2):355–371, 1994.

Chris Chafe, Bernard Mont-Reynaud, and Loren Rush. Towards and intelligent editor of digital audio: recognition of musical constructs. *Computer Music Journal*, 6 (1):30–41, 1982.

Chris Chafe, David Jaffe, Kyle Kashima, and Bernard Mont-Reynaud. Techniques for note identification in polyphonic music. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 399–405, San Francisco, California, USA, 1985.

Nick Collins. On onsets on-the-fly: Real-time event segmentation and categorisation as a compositional effect. In *Proceedings of Sound and Music Computing Conference (SMC 04)*, IRCAM, Paris, France, October 20–22 2004.

Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, New York, USA, 1994. ISBN 0-19-508609-0.

Roger B. Dannenberg. An on-line algorithm for real-time accompaniment. In *Proceedings of the 1984 International Computer Music Conference (ICMC)*, pages 193–198, IRCAM, Paris, France, June 1985.

Roger B. Dannenberg and Bernard Mont-Reynaud. Following an improvisation in real time. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 241–248, University of Illinois at Champagne/Urbana, Illinois, USA, 1987.

Matthew E. P. Davies and Paul M. Brossier. Beat tracking towards automatic musical accompaniment. In *Proceedings of the Audio Engeeniring Society 118th Convention*, Barcelona, Spain, May 2005.

Matthew E. P. Davies and Mark D. Plumbley. Causal tempo tracking of audio. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 164–169, Barcelona, Spain, October 2004.

Matthew E. P. Davies and Mark D. Plumbley. Beat tracking with a two state model. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 3, pages 241–244, Philadelphia, USA, March 19–23 2005.

Erik de Castro Lopo. Libsamplerate, a sample rate converter for audio signals. `http://www.mega-nerd.com/SRC/`, 2006a. Last visited: Fri June 2 2006.

Erik de Castro Lopo. Libsndfile, a library for reading and writing files containing sampled sound. `http://www.mega-nerd.com/libsndfile/`, 2006b. Last visited: Fri June 2 2006.

Alain de Cheveigné. *Pitch: Neural Coding and Perception*, chapter Pitch perception models. Springer Verlag, New York, 2004. ISBN 0-387-23472-1. Edited by Christopher J. Plack, Arther N. Popper, Richard R. Fay and Andrew J. Oxenham.

Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. *Journal of the Acoustical Society of America*, 111(4): 1917–1930, 2002.

Amalia de Götzen, Nicolas Bernardini, and Daniel Arfib. Traditional (?) implementations of a phase vocoder: the tricks of the trade. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-00)*, pages 37–44, University of Verona, Italy, 2000.

Diana Deutsch. *The Psychology of Music*. Academic Press, Orlando, Florida, 2nd edition, 1982. ISBN 0-12-213565-2.

Simon Dixon. An interactive beat tracking and visualization system. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 215–218, Havana, Cuba, 2001a.

Simon Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, March 2001b.

Simon Dixon, Fabien Gouyon, and Gerhard Widmer. Towards characterisation of music via rhythmic patterns. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 509–516, Barcelona, Spain, October 2004.

Christopher Dobrian. Strategies for continuous pitch and amplitude tracking in realtime interactive improvisation software. In *Proceedings of Sound and Music Computing Conference (SMC 04)*, IRCAM, Paris, France, October 20–22 2004.

Mark B. Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4): 14–27, 2001.

Boris Doval and Xavier Rodet. Fundamental frequency estimation and tracking using maximum likelihood harmonic matching and HMMs. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 221–224, 1993.

Christopher Duxbury, Mike E. Davies, and Mark B. Sandler. Separation of transient information in musical audio using multiresolution analysis techniques. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-01)*, pages 1–5, Limerick, Ireland, 2001.

Christopher Duxbury, Mike E. Davies, and Mark B. Sandler. Complex domain onset detection for musical signals. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-03)*, pages 90–93, London, UK, 2003.

Benoit Fabre. Acoustique physique des instruments de musique. Lecture notes, ATIAM, IRCAM and Université Pierre et Marie Curie, Paris, France, 2001.

Harvey Fletcher and W. A. Munson. Loudness, its definition, measurement, and calculation. *Journal of the Acoustical Society of America*, 5(2):82–108, 1933.

Neville H. Fletcher and Thomas D. Rossing. *The Physics of Musical Instruments*. Springer-Verlag, New York, 2nd edition, 1998. ISBN 0-387-98374-0.

Jonhatan Foote and Shingo Uchihashi. The beat spectrum: a new approach to rhythm analysis. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2001)*, pages 881–884, Tokyo, Japan, August 2001.

Scott Foster, Andrew W. Schloss, and Joseph A. Rochmore. Towards and intelligent editor of digital audio: signal processing methods. *Computer Music Journal*, 6 (1):42–51, 1982.

Adrian Freed, Xavier Rodet, and Philippe Depalle. Synthesis and control of hundreds of sinusoidal partials on a desktop computer without custom hardware. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 98–101, Tokyo, Japan, August 1993.

Freesound. A collaborative database of Creative Commons licensed sounds. `http://freesound.iua.upf.edu/`, April 2005. Last visited: Fri June 2 2006.

Anders Friberg and Andreas Sundström. Swing ratios and ensemble timing in jazz performance: Evidence for a common rhythmic pattern. *Music Perception*, 19 (3):333–349, 2002.

Matteo Frigo. FFTW, the Fastest Fourier Transform of the West. `http://www.fftw.org/`, 1997. Last visited: Fri June 2 2006.

Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on Program Generation, Optimization, and Platform Adaptation.

Christopher Fry. MidiVox voice-to-MIDI converter. *Computer Music Journal*, 16 (1):94–95, 1992.

Michael Good et al. MusicXML, an Internet-friendly format for sheet music. `http://www.recordare.com/xml.html`, 2004. Last visited: Fri June 2 2006.

John W. Gordon. Perception of attack transients in musical tones. Technical Report STAN-M-17, Stanford University, Department of Music, California, USA, 1984.

Masataka Goto. An audio-based real-time beat tracking system for music with or without drums. *Journal of New Music Research*, 30(2):159–171, June 2001.

Masataka Goto. Development of the RWC music database. In *Proceedings of the 18th International Congress on Acoustics (ICA 2004)*, volume 1, pages 553–556, April 2004.

Masataka Goto and Yoichi Muraoka. Music understanding at the beat level – real-time beat tracking for audio signals. In *Working Notes of the IJCAI-95 Workshop on Computational Auditory Scene Analysis*, pages 68–75, August 1995a.

Masataka Goto and Yoichi Muraoka. A real-time beat tracking system for audio signals. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 171–174, Banff Center for the Arts, Canada, 1995b.

Masataka Goto and Yoichi Muraoka. Issues in evaluating beat tracking systems. In *Working Notes of IJCAI-97 Workshop on Issues in Artificial Intelligence and Music - Evaluation and Assessment*, pages 9–16, 1997.

Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Music genre database and musical instrument sound database. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 229–230, Baltimore, Maryland, USA, October 2003.

Fabien Gouyon and Simon Dixon. Dance music classification: a tempo based approach. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 501–504, Barcelona, Spain, October 2004.

Fabien Gouyon, Nicolas Wack, and Simon Dixon. An open source tool for semi-automatic rhythmic annotation. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-04)*, pages 193–196, Naples, Italy, 2004.

Fabien Gouyon, Anssi Klapuri, Simon Dixon, Miguel Alonso, Georges Tzanetakis, Christian Uhle, and Pedro Cano. An experimental comparison of audio tempo induction algorithms. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1–13, 2006.

John M. Grey. An exploration of musical timbre. Technical Report STAN-M-2, Stanford University, Department of Music, California, USA, 1975.

John M. Grey and James A. Moorer. Perceptual evaluations of synthetised instrument tones. *Journal of the Acoustical Society of America*, 62(2):454–462, 1977.

Emilia Gómez, Anssi Klapuri, and Benoit Meudic. Melody description and extraction in the context of music content processing. *Journal of New Music Research*, 32 (1):23–40, 2003a.

Emilia Gómez, Georges Peterschmitt, Xavier Amatriain, and Perfecto Herrera. Content-based melodic transformations of audio for a music processing application. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-03)*, pages 333–338, London, UK, 2003b.

Stephen Hainsworth. *Techniques for the Automated Analysis of Musical Audio*. PhD thesis, Cambridge University, Engineering Department, Cambridge, UK, 2004.

Stephen Hainsworth and Malcom Macleod. Onset detection in music audio signals. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 163–166, Singapore, 2003.

Wolfgang Hess. Pitch determination of speech signals. algorithms and devices. *Journal of the Acoustical Society of America*, 76(4):1277–1278, October 1984.

David Huron. *The Humdrum Toolkit: Reference Manual*. Center for Computer Assisted Research in the Humanities, Menlo Park, California, 1995. ISBN 0-936943-10-6.

Florent Jaillet and Xavier Rodet. Improved modelling of attack transients in music analysis synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 30–33, Havana, Cuba, 2001.

Tristan Jehan. Event-synchronous music analysis/synthesis. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-04)*, pages 561–567, Naples, Italy, 2004.

Emir Kapanci and Avi Pfeffer. A hierarchical approach to onset detection. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 438–441, Miami, Florida, USA, November 2004.

Matti Karjalainen, Vesa Välimäki, and Zoltan Jánosy. Towards high-quality sound synthesis of the guitar and string instruments. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 56–63, Tokyo, Japan, August 1993.

Kunio Kashino and Hidehiko Tanaka. A sound source separation system with the ability of automatic tone modeling. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 248–255, Tokyo, Japan, August 1993.

Anssi Klapuri. Qualitative and quantitative aspects in the design of periodicity estimation algorithms. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, 2000.

Anssi Klapuri. Multipitch estimation and sound separation by the spectral smoothness principle. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 5, 2001.

Anssi Klapuri. Musical meter estimation and transcription. In *Cambridge Music Signal Processing Colloquium*, 2003a.

Anssi Klapuri. Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *IEEE Transactions on Speech and Audio Processing*, 11(6):804–816, November 2003b.

Anssi Klapuri. *Signal Processing Methods for the Automatic Transcription of Music*. PhD thesis, Tampere University of Technology, Tampere, Finland, 2004.

Anssi Klapuri. Pitch estimation using multiple independent time-frequency windows. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 115–118, New Platz, New York, USA, October 1999a.

Anssi Klapuri. Sound onset detection by applying psychoacoustic knowledge. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3089–3092, 1999b.

Anssi Klapuri, Tuomas Virtanen, and Jarno Seppänen. Automatic transcription of musical recordings. In *Proceedings of the Consistent and Reliable Acoustic Cues Workshop (CRAC-01)*, Aalborg, Denmark, 2001.

Anssi Klapuri, Antti Eronen, Jarno Seppänen, and Tuomas Virtanen. Automatic transcription of music. Technical report, Department of Information Technology, University of Technology, Tampere, Finland, 2002.

Mario Lang. TuneIt, a simple command-line instrument tuner for Linux. `http://delysid.org/tuneit.html`, 2003. Last visited: Fri June 2 2006.

Jean Laroche. Traitement des signaux audio-fréquences. Lecture notes, Département Signal, Groupe Acoustique, École Nationale Supérieure de Télécommunications (ENST), Paris, France, 1995.

Philippe Lepain. Polyphonic pitch extraction from music signals. *Journal of New Music Research*, 28(4):296–309, 1999.

Fred Lerdahl and Ray Jackendoff. *A generative theory of music*. MIT Press, Cambridge, Massachusetts, USA, 1983.

Pierre Leveau, Laurent Daudet, and Gaël Richard. Methodology and tools for the evaluation of automatic onset detection algorithms in music. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 72–75, Barcelona, Spain, October 2004.

Justin M. London. Cognitive constraints on metric systems: Some observations and hypotheses. *Music Perception*, 19(4):529–550, 2002.

Gareth Loy. Musicians make a standard: the MIDI phenomenon. *Computer Music Journal*, 9(4):8–26, 1985.

Richard F. Lyon and Lounette Dyer. Experiments with a computational model of the cochlea. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1975–1978, Tokyo, Japan, 1986.

Rob C. Maher and James W. Beauchamp. Fundamental frequency estimation of musical signals using a two-way mismatch procedure. *Journal of the Acoustical Society of America*, 95(4):2254–2263, 1994.

Bangalore S. Manjunath, Philippe Salembrier, and Thomas Sikora. *Introduction to MPEG-7, Multimedia Content Description Interface*. John Wiley and Sons, 2002.

Paul Masri. *Computer modeling of Sound for Transformation and Synthesis of Musical Signal*. PhD dissertation, University of Bristol, UK, 1996.

Max Matthews, Miller Puckette, David Zicarelli, et al. Max/MSP, a graphical environment for music, audio, and multimedia. `http://www.cycling74.com/products/maxmsp`, 2006. Last visited: Fri June 2 2006.

Dominic Mazzoni, Joshua Haberman, Matt Brubeck, et al. Audacity, a cross-platform audio editor. `http://audacity.sourceforge.net/`, 2000. Last visited: Fri June 2 2006.

Stephen McAdams. Music: a science of mind? *Contemporary Music Review*, 2(1): 1–61, 1987.

Robert J. McAulay and Thomas F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 34(4):744–754, 1986.

James McCartney. SuperCollider: A new real-time sound synthesis language. In *Proceedings of the International Computer Music Conference (ICMC)*, Hong Kong University of Science and Technology, 1996.

International MIDI Association. Midi specification 1.0, 1983. Los Angeles.

George A. Miller and Walter G. Taylor. The perception of repeated bursts of noise. *Journal of the Acoustical Society of America*, 20(2):171–182, 1948.

Eduardo Miranda. *Computer Sound Design: Synthesis Techniques and Programming*. Focal Press, Oxford UK, 2nd edition, June 2002. ISBN 0-240-51693-1.

MIREX. 1st Annual Music Information Retrieval Evaluation eXchange. MIREX Organising Comitee, `http://ismir2004.ismir.net/ISMIR_Contest.html`, 2004a.

MIREX. Audio melody extraction, 1st Annual Music Information Retrieval Evaluation eXchange. MIREX Organising Comitee, `http://ismir2004.ismir.net/melody_contest/results.html`, 2004b.

MIREX. 2nd Annual Music Information Retrieval Evaluation eXchange. MIREX Organising Comitee, `http://www.music-ir.org/mirex2005/`, 2005a.

MIREX. Audio onset detection, 2nd Annual Music Information Retrieval Evaluation eXchange. MIREX Organising Comitee, `http://www.music-ir.org/evaluation/mirex-results/audio-onset/`, 2005b.

MIREX. Audio tempo extraction, 2nd Annual Music Information Retrieval Evaluation eXchange. MIREX Organising Comitee, `http://www.music-ir.org/evaluation/mirex-results/audio-tempo/`, 2005c.

Dirk Moelants and Martin McKinney. Tempo perception and musical content: What makes a piece slow, fast, or temporally ambiguous? In *International Conference on Music Perception and Cognition*, Evanston, Illinois, USA, 2004.

Dirk Moelants and Christian Rampazzo. A computer system for the automatic detection of perceptual onsets in a musical signal. In A. Camurri, editor, *KANSEI - The Technology of Emotion*, pages 141–146, Genova: AIMI-DIST, 1997.

Bernard Mont-Reynaud and Mark Goldstein. On finding rythmic and melodic patterns in musical lines. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 391–398, San Francisco, California, USA, 1985.

Giuliano Monti and Mark Sandler. Pitch locking monophonic music analysis. In *Proceedings of the Audio Engeeniring Society 112th Convention*, Munich, Germany, May 2002.

Brian C. J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, 5th edition, 1997.

Brian C. J. Moore, Brian R. Glasberg, and Thomas Baer. A model for the prediction of thresholds, loudness, and partial loudness. *Journal of the Audio Engineering Society*, 45(4):224–240, April 1997.

Francis Richard Moore. The dysfunctions of MIDI. *Computer Music Journal*, 12 (1):19–28, 1988.

James A. Moorer. On the transcription of musical sound by computer. *Computer Music Journal*, 1(4):32–38, 1977.

James A. Moorer. The use of the phase vocoder in computer music applications. *Journal of the Audio Engineering Society*, 26(1/2):42–45, 1978.

James A. Moorer and John M. Grey. Lexicon of analysed tones: Part ii: Clarinet and oboe tones. *Computer Music Journal*, 1(3):12–29, 1977a.

James A. Moorer and John M. Grey. Lexicon of analysed tones: Part i: A violin tone. *Computer Music Journal*, 1(2):39–45, 1977b.

James A. Moorer and John M. Grey. Lexicon of analysed tones: Part iii: The trumpet. *Computer Music Journal*, 2(1):23–31, 1978.

Mutopia project. Mutopia, a collection of sheet music in the public domain. `http://www.mutopiaproject.org/`, 2000. Last visited: Fri June 2 2006.

Jan Nieuwenhuizen, Han-Wen Nienhuys, et al. Lilypond, an automated engraving system for typesetting sheet music. `http://lilypond.org`, 1996. Last visited: Fri June 2 2006.

Nicolas Orio and François Déchelle. Score following using spectral analysis and hidden markov models. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001.

François Pachet. Multimedia at work - playing with virtual musicians: the continuator in practice. *IEEE Multimedia*, 9(3):77–82, 2002.

Rui Pedro Paiva, Teresa Mendes, and Amílcar Cardoso. A methodology for detection of melody in polyphonic musical signals. In *Proceedings of the Audio Engeeniring Society 116th Convention*, Berlin, Germany, May 2004.

Elias Pampalk, Tim Pohle, and Gerhard Widmer. Dynamic playlist generation based on skipping behavior. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, London, UK, September 2005.

Martin Piszczalski and Bernard A. Galler. Automatic music trancription. *Computer Music Journal*, 1(4):24–31, 1977.

Michael R. Portnoff. Implementation of the digital phase vocoder using the fast Fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24(3):243–248, 1976.

Daniel Pressnitzer, Roy D. Patterson, and Katrin Krumbholz. The lower limit of melodic pitch. *Journal of the Acoustical Society of America*, 109(5):2074–2084, 2001.

Miller S. Puckette. Pure Data. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 269–272, Hong Kong University of Science and Technology, 1996a.

Miller S. Puckette. Pure Data: another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, Tachikawa, Japan, 1996b.

Miller S. Puckette, Theodore Apel, and David D. Zicarelli. Real-time analysis tools for PD and MSP. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, University of Michigan, USA, 1998.

Elena Punskaya, Christophe Andrieu, Arnaud Doucet, and William J. Fitzgerald. Bayesian curve fitting using MCMC with application to signal segmentation. *IEEE Transactions on Signal Processing*, 50(3):747–768, March 2002.

Heiko Purnhagen and Nikolaus Meine. HILN - the MPEG-4 parametric audio coding tools. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, Geneva, Italy, 2000.

Lawrence R. Rabiner. A tutorial on HMM and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Lawrence R. Rabiner and Ronald W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1978.

Lawrence R. Rabiner, Marvin R. Sambur, and Carolyn E. Schmidt. Applications of a nonlinear smoothing algorithm to speech processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(6):552–557, December 1975.

Lawrence R. Rabiner, Michael J. Cheng, Aaron E. Rosenberg, and Carol A. McGonegal. A comparative performance study of several pitch detection algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(5):399–418, October 1976.

Christopher Raphael. Automatic rhythm transcription. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 99–107, Bloomington, Indiana, USA, October 2001a.

Christopher Raphael. Music Plus One: A system for expressive and flexible musical accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba, 2001b.

Emmanuel Ravelli, Mark Sandler, and Juan-Pablo Bello. Fast implementation for non-liner time scaling of stereo signals. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-05)*, pages 182–185, Madrid, Spain, 2005.

Eric S. Raymond. *The Art of Unix Programming*. Addison-Wesley, 2003. ISBN 0-13-142901-9.

Bruno H. Repp. Some empirical observations on sound level properties of recorded piano tones. *Journal of the Acoustical Society of America*, 93(2):1136–1144, 1993.

Jean-Claude Risset. Computer study of trumpet tones. *Journal of the Acoustical Society of America*, 38(5):912–920, 1969.

Jean-Claude Risset and Max V. Matthews. Analysis of musical-instrument tones. *Physics Today*, 22(2):22–30, 1969.

Roelof J. Ritsma. Existence region of the tonal residue. *Journal of the Acoustical Society of America*, 34(9A):1224–1229, 1962.

Curtis Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, Massachusetts, 1996. ISBN 0-262-68082-3.

John Roeder and Keith Hamel. *Introduction to the Physics and Psychophysics of Music*. Springer-Verlag, New York, 2nd edition, 1975.

Oarih Ropshkow. Reproduction of the original Fletcher and Munson equal loudness curves. `http://en.wikipedia.org/wiki/Image:FletcherMunson_ELC.png`, 2005. modified for this document, Last visited: Fri June 2 2006.

David Rosenthal, Masataka Goto, and Yoichi Muraoka. Rhythm tracking using multiple hypotheses. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 85–87, Danish Institute of Electronic Music (DIEM), Aarhus, Denmark, 1994.

Matti Ryynänen and Anssi Klapuri. Modelling of note events for singing transcription. In *Proceedings of the ISCA Tutorial and Research Workshop on Statistical and Perceptual Audio Processing*, Jeju, Korea, October 2004.

Matti Ryynänen and Anssi Klapuri. Polyphonic music transcription using note event modeling. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, New York, October 2005.

Pierre Schaeffer. *Traité des Objets Musicaux*. Editions Du Seuil, Paris, France, 1966.

Bertram Scharf. *Critical Bands, Foundations of Modern Auditory Theory*. Academic Press, Orlando, 1970.

Eric D. Scheirer. *Music-Listening Systems*. PhD thesis, Media Arts and Sciences Program, School of Architecture and Planning, MIT, Boston, Massachusetts, USA, June 2000.

Eric D. Scheirer. The MPEG-4 Structured Audio standard. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 6, pages 3801–3804, May 1998a.

Eric D. Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America*, 103(1):588–601, 1998b.

Eric D. Scheirer and Barry L. Vercoe. SAOL: The MPEG-4 Structured Audio Orchestra Language. *Computer Music Journal*, 23(2):31–51, 1999.

Dietrich Schlichthärle. *Digital Filters : Basics and Design*. Springer, Berlin, 2000. ISBN 3-540-66841-1.

Andrew W. Schloss. *On the Automatic Transcription of Percussive Music - From Acoustic Signal to High-Level Analysis*. PhD thesis, Department of Hearing and Speech, Stanford University, California, USA, 1985.

Tom Schouten et al. PDP, a graphics system for PureData. `http://zwizwa.fartit.com/pd/pdp/`, `http://directory.fsf.org/vid/Misc/pdp.html`, 2002. Last visited: Fri June 2 2006.

Earl D. Schubert. *Psychological Acoustics*, chapter Editor's comments on papers 1 through 5, pages 8–16. Dowden, Hutchinson and Ross, Stroudsburg, 1979.

Xavier Serra. *A System for Sound Analysis/Transformation/Synthesis based on a Deterministic Plus Stochastic Decomposition*. PhD thesis, CCRMA, Stanford University, California, USA, 1989.

Xavier Serra and Julius O. Smith. Spectral modeling synthesis: a sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

Robert E. Simpson. *Introductory Electronics for Scientists and Engineers*. Allyn and Bacon, Boston, Massachusetts, USA, 2nd edition, 1987.

Julius O. Smith, III. Efficient synthesis of stringed musical instruments. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 64–71, Tokyo, Japan, August 1993.

Leslie S. Smith. Using an onset-based representation for sound segmentation. In *Proceedings of the International Conference on Neural networks and their Applications (NEURAP)*, pages 274–281, Marseilles, France, March 20-22 1996.

Johan Sundberg, Anders Friberg, and Roberto Bresin. Musician's and computer's tone inter-onset interval in Mozart's Piano Sonata K 332, 2nd mvt, bar 1-20. Quarterly progress and status report, TMH, Speech, Music and Hearing, 2003.

Adam Tindale, Ajay Kapur, Georges Tzanetakis, and Ichiro Fujinaga. Retrieval of percussion gestures using timbre classification techniques. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, pages 541–545, Barcelona, Spain, October 2004.

Tuukka Toivonen and Masanao Izumo. Timidity++, a MIDI to WAVE converter/player. Published under the GNU General Public License, `http://timidity.sf.net/`, 1999. Last visited: Fri June 2 2006.

George Tzanetakis. *Manipulation, Analysis and Retrieval for Audio Signals*. PhD thesis, Faculty of Princeton University, Department of Computer Science, June 2002.

George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. *Organised Sound*, 4(3):169–175, 1999.

George Tzanetakis, Georg Essl, and Perry Cook. Human perception and computer extraction of musical beat strength. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-02)*, pages 257–261, Hamburg, Germany, 2002a.

George Tzanetakis et al. Marsyas, Music Analysis, Retrieval and Synthesis for Audio Signals. `http://marsyas.sourceforge.net/`, 2002b. Last visited: Fri June 2 2006.

Georges Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, July 2002.

Dimitri van Heesch et al. Doxygen, a documentation system for C, C++, Java, Python and other languages. `http://www.doxygen.org/`, `http://www.stack.nl/~dimitri/doxygen/`, 1997. Last visited: Fri June 2 2006.

Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworths, London, UK, 2nd edition, 1979. ISBN 0-408-70929-4.

Guido van Rossum et al. Python, a dynamic and interpreted programming language. `http://python.org/`, 1991. Last visited: Fri June 2 2006.

Barry L. Vercoe. The synthetic performer in the context of live performance. In *Proceedings of the 1984 International Computer Music Conference (ICMC)*, pages 199–200, IRCAM, Paris, France, June 1985.

Vincent Verfaille. *Effets audionumériques adaptatifs: théories, mise en œuvre et usage en création musicale numérique*. PhD thesis, Université Aix-Marseille II, 2004.

Predrag Viceic. Freecycle, a beat slicer using the QT toolkit. `http://freecycle.redsteamrecords.com/`, 2006. Last visited: Fri June 2 2006.

Emmanuel Vincent and Mark D. Plumbley. Instrument identification in solo and ensemble music using independant subspace analysis. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Barcelona, Spain, October 2004.

Emmanuel Vincent and Mark D. Plumbley. Predominant-f0 estimation using bayesian harmonic waveform models. In *Proceedings of the Music Information Retrieval Evaluation eXchange (MIREX)*, 2005.

Avery Wang. An industrial strength audio search algorithm. Invited speaker. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA, October 2003.

Ge Wang and Perry R. Cook. ChucK: A concurrent, on-the-fly, audio programming language. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

Ge Wang, Perry Cook, et al. Chuck, a concurrent, on-the-fly audio programming language. `http://chuck.cs.princeton.edu/`, 2004. Last visited: Fri June 2 2006.

Ian Cunliffe Whitfield. *The neural code*, pages 163–183. Orlando: Academic Press, 1983. Eds: Edward C. Carterette and Michael L. Friedmann.

Thomas Williams and Colin Kelley. Gnuplot, a portable command-line driven interactive data and function plotting utility. `http://www.gnuplot.info/`, 1986. Last visited: Fri June 2 2006.

James D. Wise, James R. Caprio, and Thomas W. Parks. Maximum likelihood pitch estimation. *IEEE Transactions on Acoustic, Speech and Signal Processing*, 24 (5):418–423, October 1976.

Matt Wright and Adrian Freed. Open sound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 101–104, Aristotle University, Thessaloniki, Greece, 1997.

Matt Wright, Adrian Freed, and Ali Momeni. Open sound control: state of the art 2003. In *Proceedings of the 2003 conference on New Interfaces for Musical Expression (NIME 03)*, pages 153–159, Montréal, Canada, 2003.

Foundation Xiph.org. Vorbis I specification. `http://xiph.org/vorbis/doc/Vorbis_I_spec.html`, 2005. Last visited: Fri June 2 2006.

Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International Conference of the Special Interest Group on Information Retrieval (SIGIR)*, pages 42–49, Berkeley, California, USA, August 1999.

William A. Yost. Pitch strength of iterated rippled noise. *Journal of the Acoustical Society of America*, 100(5):3329–3335, 1996.

Aymeric Zils and François Pachet. Musical mosaicing. In *Proceedings of the International Conference on Digital Audio Effects (DAFx-01)*, pages 39–44, Limerick, Ireland, 2001.

Udo Zoelzer, editor. *DAFX - Digital Audio Effects*. John Wiley and Sons, 2002. ISBN 0-471-49078-4.

Eberhard Zwicker and Hugo Fastl. *Psychoacoustics: Facts and Models*. Springer, 2nd updated edition, 1990. ISBN 3-540-65063-6.