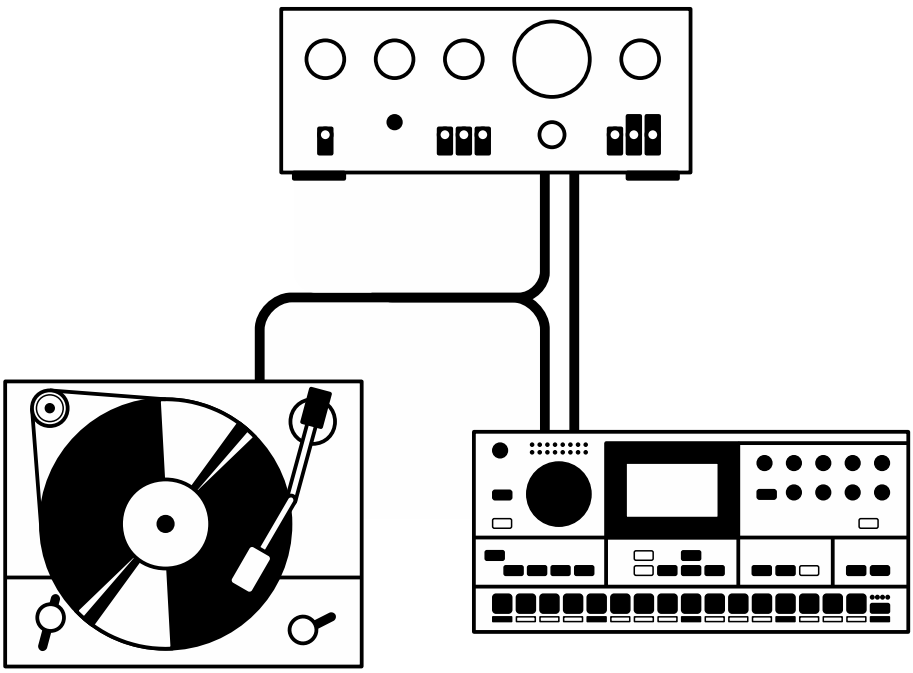
CHALMERS



Automated Synchronization of a Sequencer to Polyphonic Music

Master of Science Thesis

DAVID REVELJ

Department of Signals and Systems
Signal Processing Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2010
Report No. 041/2010

REPORT NO. 041/2010

Automated Synchronization of a Sequencer to Polyphonic Music

David Revelj

August 29, 2010

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2010

Automated Synchronization of a Sequencer to Polyphonic Music

DAVID REVELJ

Technical report no 041/2010
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 GÖTEBORG
SWEDEN
Telephone +46 (0)31-772 1000

Cover:
Schematic picture of a record player (bottom left) and a drum machine sequencer (bottom right) connected to an amplifier (top), in a configuration where the sequencer is synchronized to the record player.

Abstract

This is a master thesis report dealing with the subject of synchronizing a sequencer to polyphonic music. The task is divided into three subproblems: onset detection, tempo/meter analysis, and synchronization.

An in-depth study of selected methods for solving these problems is presented. Based on this study, a complete system has been designed.

Onset detection is performed by dividing the musical signal into several frequency bands, and differentiating their amplitude envelopes. Onsets are discriminated by peak-picking with an adaptive threshold function.

The tempo is evaluated continuously, by keeping a leaky histogram of inter onset interval times. The tempo is decided by finding the most frequent time in the histogram.

The time signature of the music is found by evaluating the onset autocorrelation at lags corresponding to tempo fractions. The bar length is decided as the tempo fraction where the autocorrelation is the strongest.

Synchronization is carried out with a phase-locking loop, where the system clock output is aligned to the detected onsets. The clock output is sent to the sequencer over a MIDI bus.

The system has been implemented on a DSP56303 evaluation module (except for time signature identification). The performance of the system depends on the input; music with a strong beat has shown to yield good results, while non-percussive music is very difficult for the system to correctly classify.

Keywords: beat tracking, onset detection, musical tempo, musical meter, BPM detection

ACKNOWLEDGMENTS

The work involved in this thesis has been carried out at the facilities of Elektron Music Machines MAV AB. The author would like to use this opportunity to send a warm thank you to everybody at Elektron for their support, and especially to Anders Gärder who tutored the project.

The author would also like to thank the examiner, professor Mats Viberg, at the Department of Signals and Systems.

Contents

List of Tables

List of Figures

ACRONYMS AND ABBREVATIONS

acf autocorrelation function

AM amplitude modulation

BD bass drum (or kick drum)

BPM beats per minute

CPU central processing unit

DF1 direct form 1 filter topology

DF2 direct form 2 filter topology

DFT discrete Fourier transform

DMA direct memory access

DSP digital signal processor

FIR finite impulse response

FFT fast Fourier transform

FWR full-wave rectification (taking the absolute of every sample)

HF high frequency

HFC high frequency content

HWR half-wave rectification (setting negative samples to zero)

IIR infinite impulse response

IOI inter onset interval (the time between two given onsets)

IRQ interrupt request

LF low frequency

LFO low frequency oscillator (typically below 20 Hz)

LP low-pass

LPF low-pass filter

MAC multiply-accumulate, see (Freescale Semiconductor, 2005a, ch. 13)

MIDI Musical Instrument Digital Interface (an industry-standard protocol for communicating with electronic musical instruments)

MM Mälzel's metronome (beats per minute)

NCO numerically controlled oscillator

PCB printed circuit-board

pdf probability density function

PFD phase frequency detector

PLL phase-locked loop

RAM random access memory

RMS root mean square

SCI serial communication interface

SD snare drum

STFT short-time Fourier transform

Chapter 1

INTRODUCTION

This master thesis was submitted to the department of Signals and Systems at Chalmers University of Technology for the program in Integrated Electronic System Design in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

The work has been carried out during 20 weeks in the first half of 2010 at the facilities of Elektron Music Machines MAV AB[1].

This thesis deals with the problem of automatically synchronizing a step sequencer to arbitrary polyphonic musical input with the help of a digital signal processor (DSP) from the Freescale 56300 family. The first chapter introduces the problem and its different aspects. It also clarifies some fundamental concepts in musical theory. Chapter 2 gives an in-depth theory of onset detection, tempo/meter analysis, and synchronization, based on the work of other authors and introducing some novelties. Chapter 3 describes the system that solves the problem, as well as the algorithms that have been employed to realize it on the DSP hardware.

1.1 Problem Description

The task at hand is that of synchronizing a sequencer to a polyphonic musical input signal. The idea is to develop a system to work in a configuration as described by figure 1.1. That is, controlling the progress of a MIDI sequencer so that it is always in phase with the musical source, their beats coinciding. A synthesizer controlled by the sequencer will the "play along" to the music. This section of the report describes the key concepts of this task.

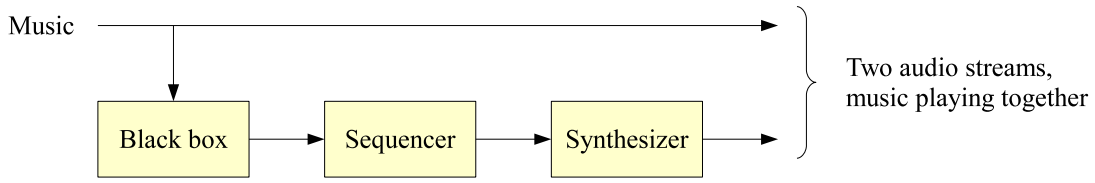


Figure 1.1: The system described in this report acts as the "Black box" in this configuration

[1]Visit `http://www.elektron.se` for more information about this company.

For improved manageability, the problem is decomposed into three parts from a bottom-up perspective. This decomposition is conformed to consequently throughout the report. These are the subproblems:

Onset detection - the extraction of low-level temporal features inside the music.

Tempo/meter estimation - the analysis of rhythmic properties of the music.

Synchronization - phase locking to the music and controlling the sequencer.

1.1.1 Onset Detection

An onset is the event that marks the beginning of a sound being played. The extraction of onsets can be seen as a form of real-time transcription of the music, only excluding information about pitch. An onset is defined by its location in time, and its weight (degree of onset).

The onsets are important because they form the basis of higher level rhythmic analysis. Their arrangement reveals properties like tempo and time signature of the music.

This section introduces the problem of onset detection, beginning with a description of the concept of *a sound*.

Sound

When an instrument is played, a sound is generated. Sound, in this sentence, is a finite duration entity, in contrast to sound as a general phenomenon. A sound could be, for instance: a note, a chord, a click, or a burst of noise.

A simple model of the how the sound amplitude varies over time (figure 1.2) is employed in this thesis. It must not be regarded as true, but as an approximation of reality.

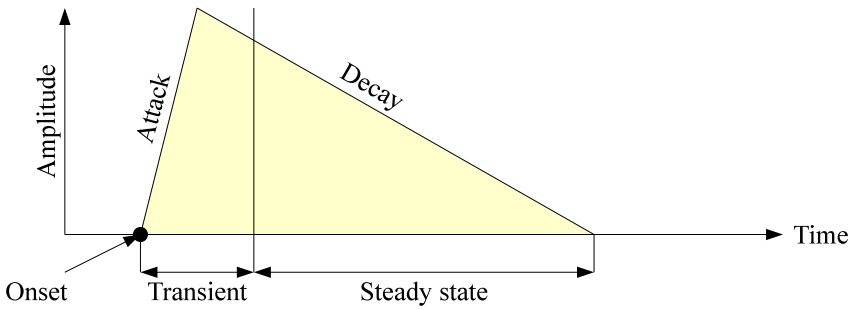


Figure 1.2: Sound envelope model

The moment when the sound begins is called the *onset*. The onset marks the beginning of the *attack* phase, during which the amplitude increases rapidly. The attack phase is followed by the *decay* phase where the amplitude slowly converges to zero.

The time period enclosing the attack is said to be a *transient*. Unfortunately, there is no strict definition of the term transient. It is used simply to denote the vague concept of "sudden change" in the signal. The transient has strong non-periodic content, and energy spread over a wide frequency range. However loosely defined, transients are very important in onset detection because their presence is an onset indicator.

The transient is followed by the *steady-state* region, were the tonal part of the sound resides. Here the sound typically consists of a number of regular sinusoidal oscillations. The amplitude is typically in decay here, but some instruments have the ability to sustain the steady state for a long time.

Different sounds have different amplitude envelopes. The presence of an attack transient relies on some kind of physical strike involved in the sound generation. This could be the hit of a drum, the plucking of a guitar, or the striking of strings in a piano. The steady-state comes from the instrument pitch, typically a physical vibration.

Human voice is pitched (except whisper) and most often contains transient regions. These transients are not restricted to the beginning of a word, and if you sing only vowels (like *aaa*) there transients are very small.

The main flaw of this amplitude model is that there are many situations were the envelope is not monotonically decreasing in the decay phase. Such artifacts can be introduced for instance by audio effects. Tremolo is an audio effect where the signal amplitude is modulated by an LFO, which gives rise to several local maxima in the amplitude envelope. Vibrato is a similar effect, only the pitch is modulated instead of the amplitude. The pitch modulation has a tremolo-like effect on the amplitude if you look only inside a narrow frequency band (which is the strategy of several onset detectors). *Reverberation* is an effect which simulates how the sound behaves in some physical space (typically the reflections inside a room). The impulse response of a reverberation filter typically have one or more strong peaks early on constituting the first reflections. Such a reflection could be falsely detected as a sound onset, even though it is in fact only an echo of an onset.

Polyphony

Polyphonic music contains several *voices*. A voice, in this sentence, is a source of sound that could be any musical instrument or (but not necessarily) human voice. Voices add together to form the polyphonic music. In other words: polyphonic music is just normal music. Music that is *not* polyphonic is for instance solo instrumental performances or a capella song.

Figure 1.3: Example of polyphony; 4 voices are playing together and their sum constitutes the perceived signal

The fact that the music is polyphonic adds complexity to the problem. For starters, the onset pattern becomes more difficult to analyze; just imagine the union of every sound being played by every voice. It is not really evident that you'll find a pattern, at least not as easily as you would by looking at for instance the drum onsets alone. There is also the problem of onsets which are intended to be simultaneous, but are in reality slightly misaligned.

Furthermore, the voices mask each other. A sound with some duration and relatively high amplitude may very well hide other lower-amplitude onsets that occur during its

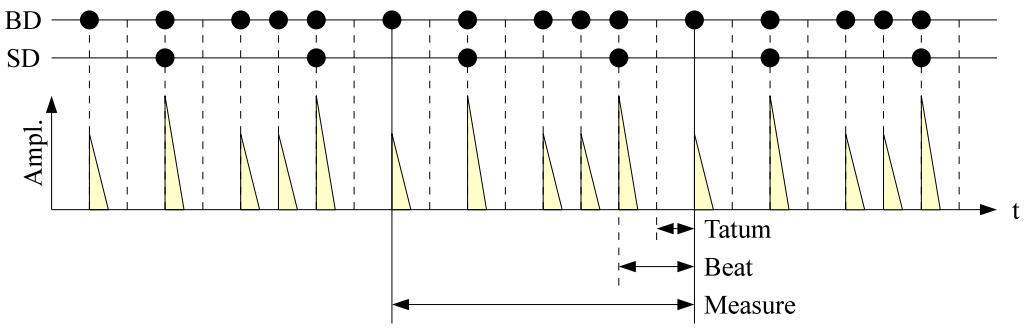


Figure 1.4: Hierarchical rhythm structure

course. These onsets are most likely appearing in frequency regions where the high amplitude sound is not present. This means that they most probably are distinguishable to the human ear, but also very probably not visible in the signal envelope.

Onset masking is a problem especially in modern music, which is usually dynamically compressed so that the overall loudness curve is flat rather than dynamic.

1.1.2 Tempo/Meter estimation

Tempo and meter estimation is the extraction of higher level temporal features from the music.

Rhythm is the sense of a reoccurring pattern in music. It is a vast concept, as there are infinite ways of constructing rhythm and there is no universal description of it. Contemporary western music does however tend to follow a well-defined hierarchical structure, which will serve as the model of rhythm in this thesis.

Rhythmical components

Figure 1.4 shows a common rock rhythm repeated three times. The scatter plot on the top is a drum transcription marking the bass drum (BD) and snare drum (SD) being played. The area plot below is the amplitude envelope.

The smallest, most low-level component is the *tatum* (Seppänen, 2001). It is the lowest common time denominator, i.e. any interval between two onsets is an integer multiple of the tatum. The tatum is not a widely accepted musical term, but it is common in papers on rhythmic analysis.

The *beat* (or tactus) is a higher level component of the rhythm. It is composed of an integer number of tatums (in this case: 2). The beat defines the tempo of the music.

The beat is usually found in a rhythmic percussion instrument like drums, but not always. Performances without rhythmic instruments too have a sense of a beat, or something as subtle as when the beat "should" occur.

The *measure* is the period of time into which the beats are grouped. It is also referred to as a *bar*, since that is how it the measure is marked in sheet music. The beat that marks the beginning of the measure is called the *down-beat* (from the arm-movement of a conductor).

Tempo

The musical *tempo* is the repetition rate of beats.

Humans have the ability to easily identify musical tempo. This is why the tempo is also called the foot-tapping rate. Interestingly, two persons listening to the same music can have different opinions on the foot-tapping rate, without anyone of them being wrong. The situation would then typically be that one person taps twice as fast as the other.

The tempo is usually not constant throughout a song, even when so has been stated. It tends to vary somewhat over time as a result of indeliberate deviation caused by the human inability to keep an absolutely steady pace. It can however be close to constant if a metronome has been used, and absolutely constant if a machine (like a sequencer) has been involved.

A time-varying tempo could also be deliberate in form of ramp (increasing or decreasing over time) or step (sudden) changes.

Tempo is specified in beats per minute (BPM) or Mälzel's metronome (MM), both having the exact same meaning. For applications such as beat tracking it is also important to note that the beats have a certain alignment in time (cf. the phase of a sinusoid). In this report this property is referred to as the beat phase.

To specify relationship between different tempi, the term *tempo harmonic* denotes an integer relationship, and the term *tempo octave* is used to denote a power of 2 relationship.

1.1.3 Meter (Time Signature)

The structure of the beats inside the bar is called the *musical meter* or *time signature*. The meter is described by two numbers, indicating number of beats per measure, and their duration. For example, 3/4 means that the measure consists of three beats, and they have a duration of one quarter each. Notice that for this particular example, the whole note consists of three quarter notes (not 4 as you could expect). It is a *triple* meter is known as waltz, which can be experienced by counting *one*-two-three-*one*-two-three... It is not very common in popular music today, but it has been used a lot throughout history. Figure 1.5 shows an example of a triple meter drum loop.

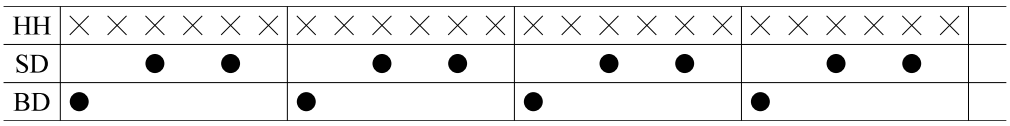


Figure 1.5: Example of simple triple meter drum loop

The possible meter variations are infinite, the only limitation being the imagination of the composer. Fortunately (from an analysis point of view) most music is written in 4/4 meaning that the bar has four quarter notes. This is a *quadruple* meter since the number of beats is a multiple of 4, but it also *duple* since it is also a multiple of 2. A simple duple/quadruple meter drum loop is shown in figure 1.6. It has time signature 4/4 or 8/8 depending on what you consider to be the beat period. You could also make it 2/4 or 4/8 without changing the rhythm in any way, by simply putting bar marks between every second quarter beat in the transcription.

The point of this elaboration was to show that to be 100% sure about the time signature you must look at the sheet music, everything else is subjective opinion. From an identification point of view, however, it doesn't matter what you call this particular beat, as long as you correctly state that it is simple duple.

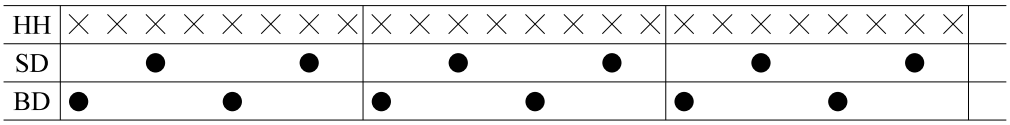


Figure 1.6: Example of simple duple/quadruple meter drum loop

Both 3/4 and 4/4 has a subdivision of the beat into two lesser time units. If instead a subdivision of three is used, the meter is called compound. Figure 1.7 shows such a rhythm: 12/8 with 4 quarter notes per bar, and 3 eighths per quarter. Notice that the time signature notation does not always reveal whether the meter is compound or simple. For example the time signature 6/8 could imply compound duple (2 quarters of 3 eighths) or simple triple (3 quarters of 2 eighths).

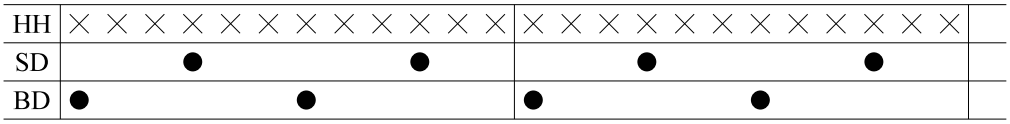


Figure 1.7: Example of compound meter drum loop

1.2 Synchronization

The synchronization problem has two main aspects. The first is phase-locking to the musical beat. The second is controlling the sequencer.

Successful synchronization means that the sequencer ”plays along” with the music. To achieve this, the sequencer must be playing at the same tempo as the music at all times. It must also be in phase with the music, so that beats occur simultaneously.

1.2.1 Phase locking

Phase locking means keeping the sequencer beat aligned with the input music over time. It is not enough to just clock the sequencer with a constant tempo, as it will eventually drift away from the musical source. Continuous phase locking is also important because of tempo variations in the music.

1.2.2 Sequencer control

As sententiously put by the New Oxford American Dictionary, a sequencer is

> “a programmable electronic device for storing sequences of musical notes, chords, or rhythms and transmitting them when required to an electronic musical instrument”

In other words, the sequencer holds a sequence of notes, and has the ability to “tell” a synthesizer to play a certain note at a certain moment in time.

The notes are arranged on a time scale which is relative rather than absolute, meaning that it uses fractions of a bar rather than seconds to determine when a note is to be played. The absolute time of the notes is then decided by the tempo, the rate at which the sequencer advances, i.e. fractions of a bar per time unit, most commonly beats per minute.

A sequencer usually has an internal clock to control its own tempo, but it can also use an external source of timing information. In this project, the sequencer’s progress is controlled via clock messages on a Musical Instrument Digital Interface (MIDI) bus. The tempo is determined by the rate of the incoming messages.

1.3 Background

The topic of this master thesis was suggested by Elektron, producers of electronic musical instruments. Elektron has the ambition to make automated synchronization a feature in some of their products (including drum machines, synthesizers, samplers and sequencers).

This is a desirable feature in such devices, as it enables a range of interesting applications for their users. A musician could, for instance, use it to get automatic accompaniment while playing his instrument freely. It could also be used to synchronize digital synthesizers to their analog equivalents. Possible applications for DJs include facilitated sampling of measures (defining durations and loop points), and automatic beat matching when mixing two songs.

1.4 Previous work

There are numerous papers dealing with the topic of onset detection and many on tempo estimation. But when it comes to meter recognition and synchronization there are very few.

1.5 Goals

The goal of this thesis is a DSP56300 implementation of a system that synchronizes a MIDI sequencer to arbitrary musical input. The initial ambition was to meet the following specifications.

- Real-time operation with negligible latency
- Instantaneous tempo response, producing output already after two beats
- Time signature detection
- Ability to follow small tempo variations around some constant tempo
- Ability to detect large tempo changes
- Ability to hold a tempo during silent passages in music

The system should also be possible to run on a specific platform, which is described in detail in section 1.6.1.

1.6 Methodology

A comprehensive literature study was the first part of the master thesis.

Then, off-line experimental algorithms were developed in Matlab for evaluation. All plots in this report are the results of such Matlab simulations, unless explicitly stated otherwise.

As an intermediate step, (almost) real-time versions of some algorithms have been implemented in Java using the JavaSound API. This was very useful, as JavaSound has support not only for sampled audio, but also for MIDI. These experiments could be performed on-line with a sequencer.

Finally, the final version of the system was partially implemented in machine language on a Freescale DSP.

The system that is described in this report is intended to run on a specific platform described below. Therefore it is important to consider strengths and limitations of this platform in the system design process. Unfortunately, the system is not implemented on the actual target platform within the scope of this thesis. As a substitute there is an evaluation board which runs the system and communicates the tempo information over MIDI to the target platform. The reason for this decision is that software development is easier on the evaluation module, as the target platform has other software running simultaneously.

1.6.1 Target Platform

The target platform is a hardware music sequencer that holds two main computational units. One is a Freescale **Coldfire MCF54454**, a 32-bit microprocessor running at 266 MHz with eMAC[2] unit. The other is a Freescale **DSP56721**, a dual core 24-bit digital signal processor at 200 MHz, code compatible with the Freescale DSP56000 and DSP56300 families. The DSP also contains 248K x 24-bit words of random access memory (RAM).

The DSP has hardware support for basic arithemetics (addition, subtraction) and logics (and, or, eor). It also features multiplication and the handy multiply-accumulate (MAC) operation. The DSP does *not*, however, have hardware support for more complex numerical operations such as square root, trigonometry, exponentials or logarithms. These operations have to be implemented in software if they are to be used. Division is supported to the extent that there is an instruction for performing an iteration of a nonrestoring division algorithm (Freescale Semiconductor, 2005a, ch. 13).

1.6.2 Evaluation module

The algorithm is implemented on a DSP evaluation board, which then signals the tempo to the target platform via a serial interface. This board has a Motorola **DSP56303** which is opcode compatible with the 56721 of the target platform. It has the same instruction set and arithmetic processing, but differs in that it is single core and runs at approximately 100 MHz. The evaluation board has A/D and D/A converters for analog sound input and output. It has two serial ports; one for debugging and one for communication. It has several LEDs, of which one can be easily controlled by user software.

[2]Enhanced Multiply-Accumulate

Chapter 2

THEORY

This chapter is a theoretical study. It describes how the problems have been, or can be, solved. There are three sections in this chapter, one for each of the subproblems: onset detection, tempo/meter estimation, and synchronization.

2.1 Onset detection

Musical onset detection is a subject that has been covered by many different authors. There is a range of different approaches to the problem, but they have much in common. Several comparative studies concerning onset detection techniques have been carried out earlier, such as (Hockman, 2008; Gouyon et al., 2006; Collins, 2005; Bello et al., 2005; Alonso et al., 2003).

The general onset detection strategy is to find some property of the signal that strongly relates to the presence of onsets, and then process the signal so that all other properties are removed. This process is referred to as *signal reduction*. The reduced signal in turn is called the *(onset) detection function*.

The purpose of this signal reduction is dual. First of all it should emphasize the onsets to enable further analysis. Secondly, it should reduce the amount of information in the signal so that it contains only the necessary. Most of the information in a musical signal is not related to rhythm.

The (onset) detection function, $d[n]$, should reflect the *degree of onset* at sample n. Degree of onset is a hazy concept, but it is intended to describe the "rhythmical weight" of the onset. A quarter note, for instance, is of more value than an eighth. This property turns out to be very difficult to establish, and that is a major problem in onset detection theory.

Downsampling

The detection function is typically downsampled compared to the original signal. This is because the dense temporal resolution required to describe sound is not at all necessary to describe rhythm. Matching onset locations within a few hundreds of a second would have to be considered a decent match. Consequently, downsampling of the detection function can be employed with advantage. With it follows substantial data reduction which reduces

the computational load of the system. Another benefit of downsampling is amplification of the signal derivative (proportional to $1/\Delta t$).

Tempo is however, the reciprocal beat period. Downsampling leads to reduced precision in the measuring of this time. Consider that even the most accurate observation of it can be assumed as accurate only within a ± 0.5 sample interval. As insignificant as this may sound, at a sample rate of 100 Hz and a tempo of 120 BPM (=50 samples per beat period), an error of 1 sample corresponds to 2.5 BPM (!). This effectively means that several periods must be observed before accurate tempo estimation is possible.

According to Scheirer (1998), at least 100 Hz should be used. Some example values used by different authors are: 86 Hz (Laroche, 2003), 100 Hz (Seppänen, 2001), 200 Hz (Scheirer, 1998), 250 Hz (Klapuri, 1999), 690 Hz (Masri and Bateman, 1996), 1 kHz (Uhle and Herre, 2003).

Several methods use the *continuous* onset detection function as input to higher level rhythmic analysis. It is then typically processed in frames of a few seconds in length. It is equally common to explicitly determine the onset locations and base further analysis on this set of data. The latter approach discards a lot of information, but it is quite natural if you consider that an ideal onset detection function should contain nothing but pulses marking onset locations, and can therefore be expressed as a sum of weighted, time shifted Kronecker delta functions.

Explicit onset detection requires *thresholding* and *peak picking* in the detection function.

2.1.1 Subband Amplitude Envelopes

Onset detection based on subband amplitude envelopes is a popular strategy. It is probably the method which has seen the most incarnations. This section is based largely on the works of Scheirer (1998); Klapuri (1999); Seppänen (2001). Their approaches do have differences, but when it comes to onset detection they have much in common and are interesting to compare.

Subband Decomposition

The audio stream is decomposed into a number of frequency bands. The purpose of this operation is to isolate musical events that are not "visible" in the original audio stream, because of "masking" as a result of polyphony and compression (see section 1.1.1).

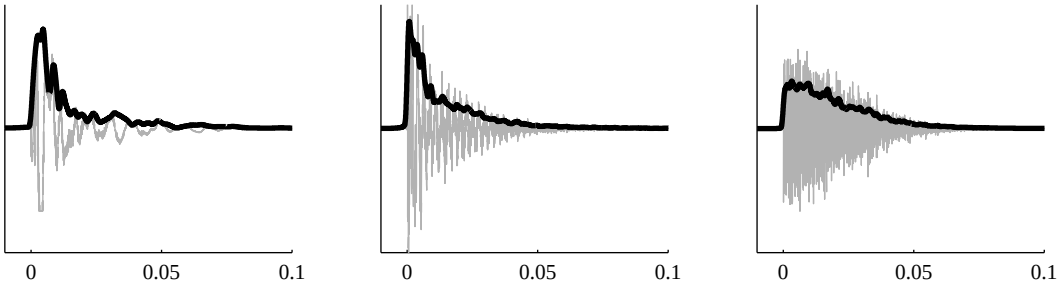
Not much have been written about how to choose the specific bands. Scheirer (1998) states that his algorithm is "not particularly sensitive to the particular bands", and concludes that a small number of octave bands is sufficient. His system uses 6 adjacent non-overlapping bands width edges at 0, 200, 400, 800, 1600, 3200, and 22050 Hz.

Klapuri (1999) uses 21 bands distributed from 44 Hz to 17 kHz. The lowest three bands are full octave, while the upper 18 are third-octaves. The intention of using third octave filters is to roughly capture the critical bands of hearing. In a later paper, Klapuri et al. (2006) uses 36 critical bands distributed from 50 Hz to 20 kHz, and states that "The exact number of subbands is not critical".

Subband decomposition is often implemented with a bank of bandpass filters. It is also common to use short-time Fourier transform (STFT) and group frequency bins together to form bands.

Envelope detection

The envelope is an imaginary line (in the signal plot) describing the general shape of the sound. It does not follow the rapid oscillations, but rather shows the contour or outline of the sound. Unless the signal consists of just one single sinusoid, it is difficult to talk about a *true* or *ideal* envelope, it has to be some kind of estimation. As an example, figure 2.1 shows the envelopes of three common drum sounds.



(a) Kick (b) Snare (c) Hi-hat, closed

Figure 2.1: Drum envelopes, extracted with Hilbert transform and smoothing FIR filter

The demodulation of an AM radio signal is one example of envelope detection. The AM signal consists of an HF carrier wave with a time-varying amplitude decided by an LF signal (audio). The LF signal is the envelope of amplitude modulation (AM) signal. The case of a sound envelope is analogous to that of an AM signal; pitch is the carrier wave and loudness is the envelope.

One fundamental prerequisite for envelope detection is that the frequency content of signal must be significantly higher than that of the envelope. If the envelope has variations that are as fast as those of the signal, it is not possible to detect. This is a major problem in on-line audio envelope detection, as the lowest audible frequencies (around 20 Hz) are very much comparable in speed to rapid loudness variations (such as attack transients).

Full-Wave Rectification is a non-linear operation that removes sign information from the signal so that only magnitude is left.

The full-wave rectified signal is visually a good descriptor of the amplitude envelope, especially when the signal contains high frequency compared to the envelope. The problem is of course the oscillatory behavior with many zero-crossings.

A good envelope approximation can be extracted from the rectified signal by applying a low-pass smoothing filter. Using this method, the envelope e at sample n is written as

$$e[n] = (|x| * h)[n] \tag{2.1}$$

where x is the signal and h is the filter.

Another way of detecting the envelope takes off from the root mean square (RMS) level of the signal. The RMS level is given as the square root of the average power of the signal (Smith, 2007a, pp. 74-75). The RMS can be calculated for a smaller window in time to get a measure of the local level. Equation (2.2) shows how to calculate the RMS level for a window of N samples around time n.

$$e[n] = \sqrt{\frac{1}{N} \sum_{k=-(N-1)/2}^{(N-1)/2} (x[n+k])^2}, \quad N \text{ odd} \tag{2.2}$$

The RMS approach makes more physical sense than full-wave rectification (FWR), as it is based on a physical quantity (energy), and agrees with the notion of loudness. The integrator can be implemented very efficiently (from a computational point of view) in the DSP, but it needs a lot of memory. A low order IIR LP filter could also function as a leaky integrator. The result is typically smoother than that of using a brick-wall FIR integrator. Causal RMS calculation can be written as

$$e[n] = \sqrt{(x^2 * h)[n]}, \tag{2.3}$$

where x is the signal and h is the integration filter.

Smoothing is employed to remove noise and make the envelope (and its derivative) more well-behaved. It is also used to perform energy integration as a part of RMS approximation.

Designing the smoothing filter for the envelope detector is not trivial. On one hand it needs to be fast enough to capture rapid attacks. At the same time it needs to be rather slow, so that it does not follow all the amplitude variations of the sustain- and decay phases of the sound.

Using a brick-wall integrator makes perfect sense considering equation (2.2), the RMS definition. Unfortunately this approach gives a rather jerky output, and because the integration window tends to be rather long, the implementation of such a filter is either slow or memory consuming. If a long filter is acceptable, it is possible to use a more sophisticated convolution kernel, like an LP filter with well defined design parameters.

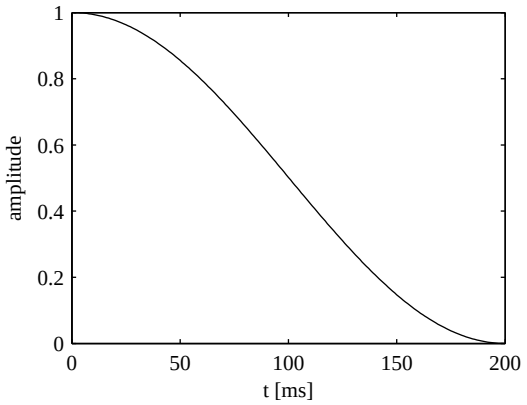
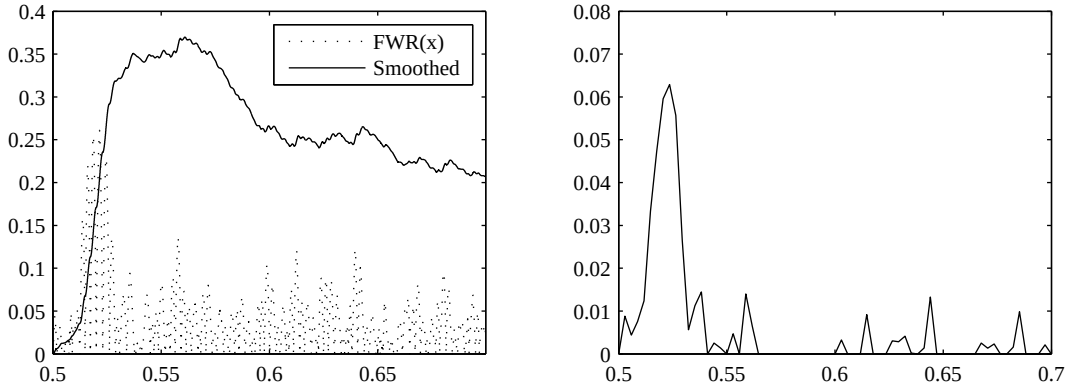


Figure 2.2: Right half Hann-window smoothing filter

An interesting option is the psychoacoustically motivated filter described by Scheirer (1998). It is designed to perform energy integration in a way similar to the human ear. It's impulse response (figure 2.2) is constructed from the right half of a 400 ms Hann window. The sudden transition to 1 at $t = 0$ makes it good at capturing attacks, while the tail hides the problematic oscillations of the decay phase. Since its introduction, this filter

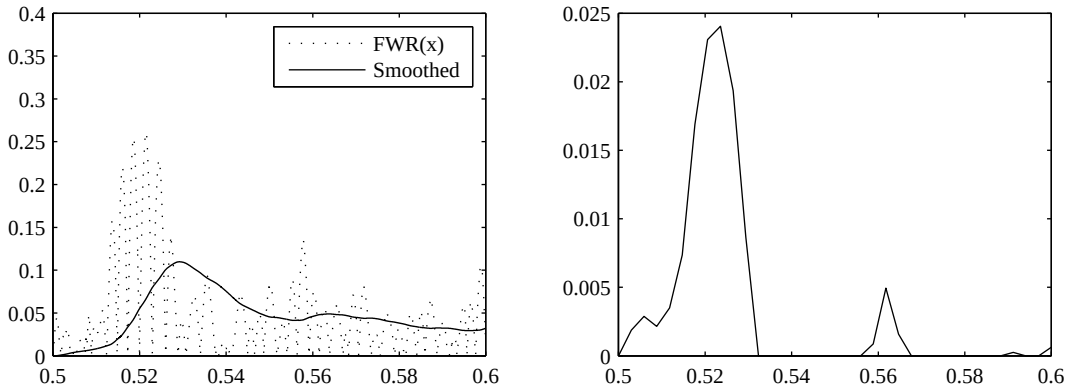
has successfully been used in several onset detection designs (Klapuri, 1999; Seppänen, 2001; Schuller et al., 2007).



(a) Amplitude envelope (b) Differentiated envelope

Figure 2.3: Envelope detection with FWR and custom FIR integrator

A simple, yet efficient, alternative is to use an IIR integrator. From a computational point of view it is very efficient compared to the FIR filter. It is possible to create a smooth amplitude follower with only a few coefficients. As an example, Seppänen (2001) uses a third order IIR LP filter with a cut-off at 30 Hz for the RMS approximation. The main problem with using an IIR filter is that you have to choose between making it fast or slow. Ideally it should be fast during attacks and slow during decays. Seppänen (2001) avoids this pitfall by using an FIR integrator at a lower sampling rate after the IIR integrator.

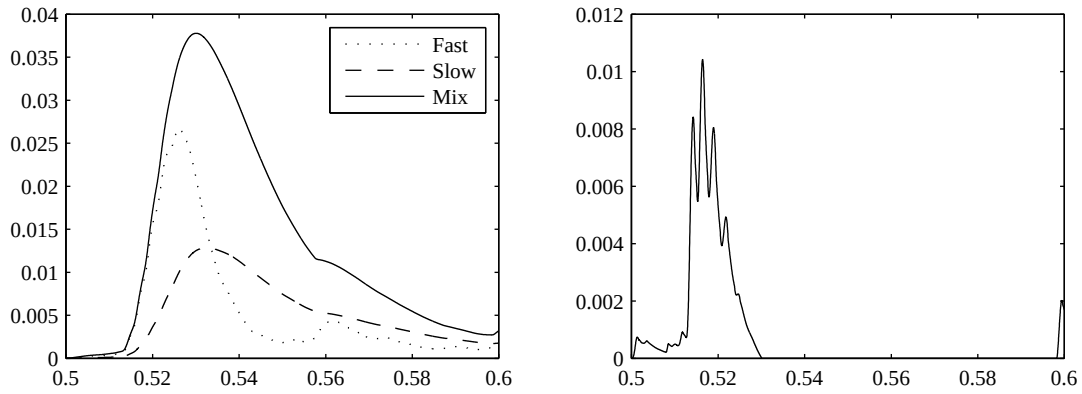


(a) Amplitude envelope (b) Differentiated envelope

Figure 2.4: Envelope detection with FWR and smoothing IIR filter

Another way to achieve both slow and fast behavior is to switch the filter coefficients in real time. That is, having one set of coefficients for attacks, and another for decays. To determine whether the signal is in attack or decay phase, the filter output is compared to the signal. If output level of the filter is above that of the signal, then decay coefficients

are chosen. If the signal level is above that of the filter output, the attack coefficients are chosen. The filter output is the amplitude envelope.



(a) Amplitude envelope

(b) Differentiated envelope

Figure 2.5: Envelope detection with coefficient switch

The method of switching coefficients works very well if the objective is to study only the amplitude level. Figure 2.5(a) shows an envelope extracted with a fast filter ($f_c = 45$ Hz), a slow filter ($f_c = 15$ Hz), and a mix where the filter coefficients have been chosen depending on if the filter is in attack or decay phase. Unfortunately the coefficient switching results in a jerky first order derivative, as you can see figure 2.5(b). This makes peak-picking much more difficult, as there are several local maxima for every attack.

The Hilbert Transform is another way of extracting the envelope of a narrowband signal (Smith, 2007a, sec. 4.3.7). The mathematics behind this transform are somewhat complicated, but it's enough to understand the result to make use of the transform in envelope detection.

Consider the signal $x(t) = A(t)\sin(\omega t + \phi)$. The amplitude $A(t)$ is varies slowly over time, and it is always positive. The value of $x(t)$, however, takes both positive and negative values, and varies more quickly because of the $\sin()$ expression. It is a non-trivial task to determine $A(t)$ for all t in this signal.

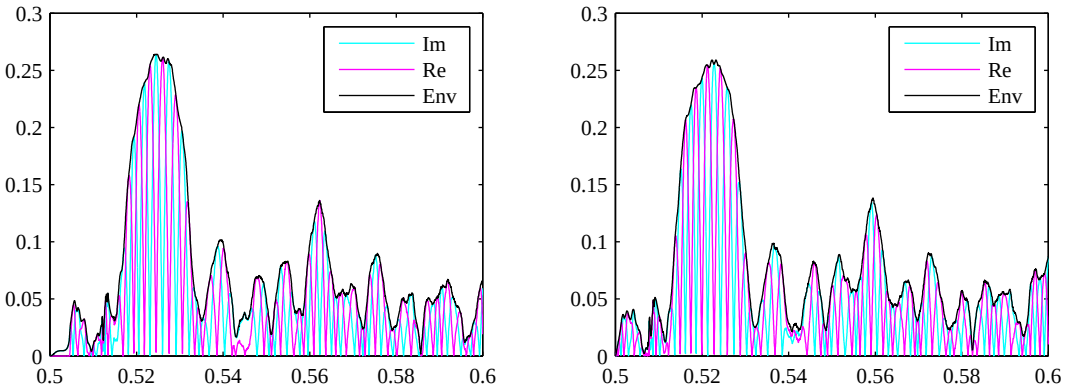
If, however, we somehow had access to the signal $A\cos(\omega t+\phi)$, we could find A from the trigonometric identity $A^2 = \sin^2(\theta)+\cos^2(\theta)$. This is what the Hilbert transform does for us; it gives us access to that signal. The transform is actually an allpass filter with a phase response such that all frequencies are shifted one quarter period (+90 degrees for positive frequencies, -90 degrees for negative).

The filter input $x[n]$ (in-phase component) and output $\hat{x}[n]$ (phase-quadrature component) are used to form the complex signal $y[n] = x[n] + j\hat{x}[n]$. The magnitude of $y[n]$ is the amplitude envelope:

$$e[n] = |y[n]| = \sqrt{x^2[n] + \hat{x}^2[n]}. \tag{2.4}$$

As an alternative to calculating the actual magnitude of $y[n]$ it is possible to simply add the magnitudes of x and $\hat{x}$ together. This is not mathematically correct but computationally beneficial on systems with no hardware support for the square root operation (such as

the DSP56300 family), and together with a smoothing filter the resulting envelope is not that bad at all.



(a) FIR (200 muls)

(b) IIR (12 muls)

Figure 2.6: Two ways of calculating the envelope using Hilbert transform

The actual implementation of the Hilbert transform, unfortunately, has to be an approximation. Its impulse response is given by equation (2.5) (Gold et al., 1969). This is a non-causal and infinite function that must be delayed and truncated before it can be implemented in practice. The in-phase component must of course be delayed the same amount before calculating the envelope in equation (2.4).

$$h[n] = \begin{cases} \frac{1-e^{j\pi n}}{\pi n}, & n \neq 0 \\ 0, & n = 0 \end{cases} \tag{2.5}$$

Notice that equation (2.5) evaluates to zero for even n, meaning that the number of MACs needed to calculate it is only half the filter order. Still, the filter needs to be quite long for good results. The envelope in figure 2.6(a) was calculated with a 400^{th} order filter.

It is possible to have more efficient calculation of the Hilbert transform using IIR filters, as demonstrated by Niemitalo (2003). Two lines of cascaded allpass filters with carefully designed phase responses are used to create two signals, which have the mutual relation of in-phase and phase-quadrature components. This relation means that the latter is the the Hilbert transform of the former, which is enough to calculate the envelope even though neither the original signal nor its transform is known. An envelope created with this method is shown in figure 2.6(b) where you can see that it is very similar to that calculated with a long FIR filter in figure 2.6(a).

Compression and differentiation

The amplitude envelope shows the magnitude of the signal at a given time, but the human ear does not perceive loudness in an amplitude-linear fashion. The hearing mechanism is not easily described, and the relation between amplitude and loudness is different for different frequencies, but we can arrive at a decent approximation based on some simple observations. For a sound that is not very loud (almost silent) it is possible to hear very

small changes in amplitude. If an amplitude variation of the same magnitude was applied to a louder sound, it would go completely unnoticed. Thus, changes in loudness are better described by the *relative difference*, meaning the size of change over the size of the number.

Another advantage of using relative difference compared to absolute is the onsets are less weighted by the absolute energy in each frequency band. This is good especially for HF bands as they typically have much less energy compared to LF bands. If the differences of several bands are going to be added together, this property is important because it reduces onset masking between bands.

A simple way to arrive at a function with these properties is to take the logarithm of the envelope, a large improvement compared to the linear scale. Klapuri (1999) used the logarithmic difference, equation (2.6) in his implementation, and made some valuable observations. Firstly, the response to changes in amplitude is much quicker meaning that the local maximum in the difference function better describes the actual location of the onset. Secondly, the smaller oscillations that give rise to several local maxima for just one offset are significantly reduced. This is because the oscillations are small compared to the overall level of the envelope at that point.

$$d[n] = log(e[n]) - log(e[n-1]) \tag{2.6}$$

The continuous derivative of the logarithm of a function $f(x)$ is given by equation (2.7).

$$\log(f(x))' = \frac{f'(x)}{f(x)} \tag{2.7}$$

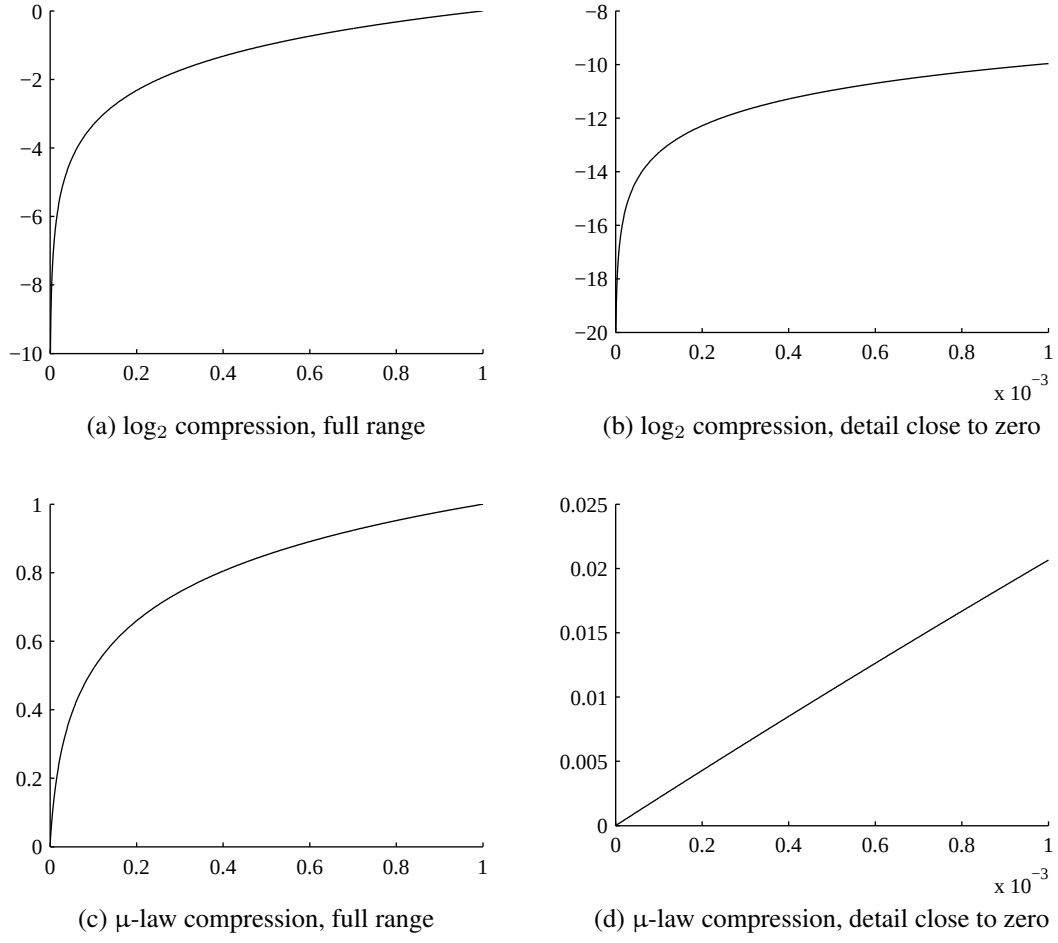
As you can see, the actual logarithm operation is not necessarily needed to get the logarithmic difference. This observation explains the formula suggested by Seppänen (2001), where the relative difference is calculated according to equation (2.8) by scaling the absolute difference with the inverse of the two samples added together. This yields roughly the same results as equation (2.6), but is computationally beneficial in a system with hardware support for division.

$$d[n] = \frac{e[n] - e[n-1]}{e[n] + e[n-1]} \tag{2.8}$$

Unfortunately the logarithmic difference as such does not hold well for very small numbers. For starters, $\log(0) \rightarrow -\infty$, meaning that any change from zero will result in huge difference peaks. Using this function, silence with added noise of very low amplitude gives rise to peaks that are much larger than normal onsets in music. This gives a problem not only of false detections, but also makes it much more difficult to create an adaptive threshold for detecting peaks.

As a solution to this problem one could look at other types of compression. For speech encoding in telecommunication applications, a method called μ-law compression, equation (2.9), is often used. An application of this algorithm in onset detection can be seen in (Klapuri et al., 2006). This type of compression gives a linear behavior around zero.

$$d[n] = \text{sgn}(e[n]) \frac{\ln(1 + \mu|e[n]|)}{\ln(1 + \mu)} \tag{2.9}$$



(a) $\log_2$ compression, full range

(b) $\log_2$ compression, detail close to zero

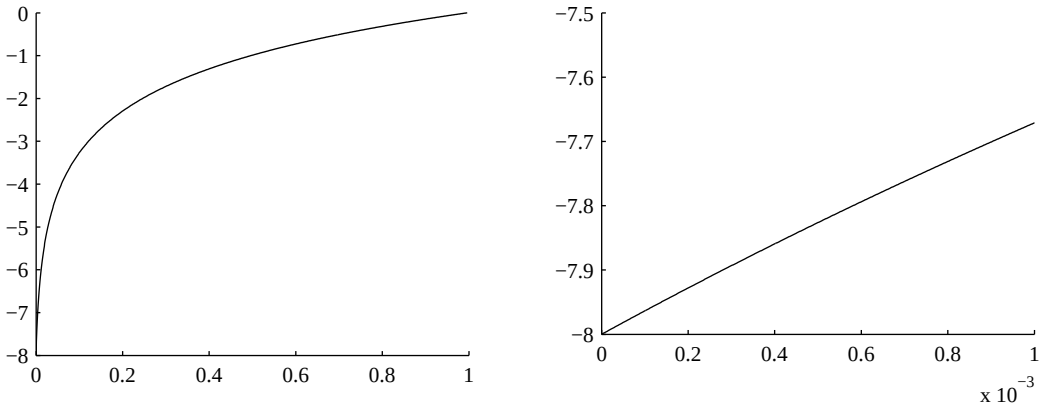
(c) μ-law compression, full range

(d) μ-law compression, detail close to zero

Figure 2.7: Logarithmic vs. μ-law compression ($\mu = 100$)

Another way of altering the relative difference to behave more nicely around zero could be to simply add a small bias to the envelope before taking the logarithm. This means that if the input signal needs to be of a size comparable to the bias in order to create significant change on the output. When the signal is large compared to the constant, the effect of the constant becomes negligible. This as achieved simply by adding the small constant c to the denominator in equation (2.8) or inside each $\log()$ in equation (2.6). This compression function is given by equation (2.10). Figure 2.8 shows a plot of this function. As you can see, it's close to logarithmic in the full range and close to linear near zero.

$$d[n] = \log_2(e[n] + c) \approx \begin{cases} \log_2 e[n] & \text{if } e[n] \gg c \\ \log_2 c & \text{if } e[n] \ll c \end{cases} \tag{2.10}$$



(a) full range

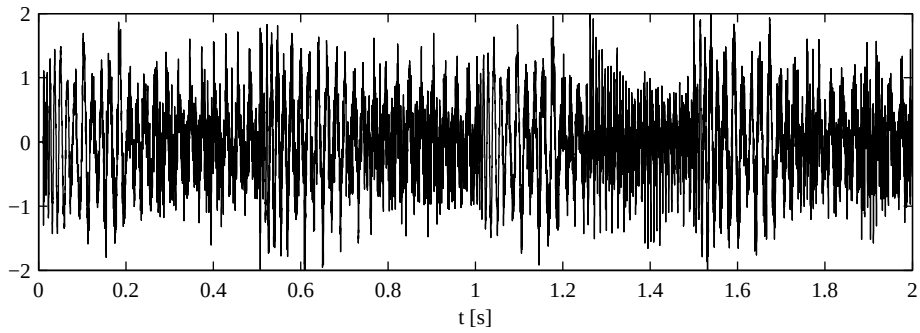
(b) detail close to zero

Figure 2.8: Logarithmic compression with added constant $c = 2^{-8}$

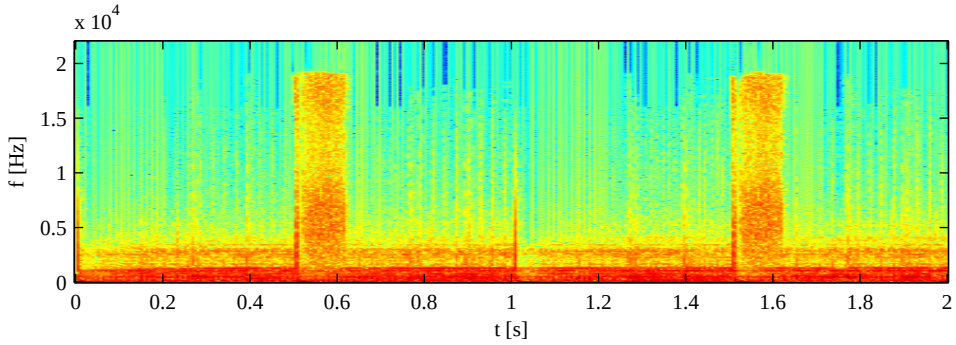
2.1.2 Spectral Analysis

Spectral analysis methods for onset detection have a lot in common with the subband envelope approach. They, too measure the energy in a range of frequency bands. The focus is however to make use of dense spectral resolution rather than sophisticated envelope differentiation schemes.

This approach makes sense, as spectral analysis reveals many things that are not visible in the time domain representation of the signal. Compare figures 2.9(a) and 2.9(b). It is not easy to identify the onsets in the time domain plot, but they are clearly visible in the spectrogram.



(a) Time domain plot



(b) Time-frequency plot (spectrogram)

Figure 2.9: Time domain and time-frequency domain plots

Short Time Fourier Transform (STFT)

STFT is the process of DFT-transforming windowed snapshots of the audio signal to get a two-dimensional time-frequency representation of it.

Equation (2.11) describes how the STFT is calculated. Consecutive, overlapping frames are extracted from the signal, inside which the samples are multiplied with a window-function. The amount of overlap is decided by the hop length parameter.

$$X[n,k] = \sum_{t=0}^{L-1} \left(x[t+nM]\; w[t]\; W_L^{tk} \right), \tag{2.11}$$

where $x =$ Audio stream,
$w =$ Window,
$k =$ FFT frequency bin,
$n =$ Frame number,
$L =$ Frame length,
$M =$ Hop length,
$W_L = e^{-2\pi j/L}$, FFT twiddle factor

The result is a matrix form representation of the audio with frequency on one axis and time on the other. The frequency resolution is determined by the window length, and the temporal resolution is determined by the hop size.

Figure 2.10 shows the extraction of five 11.6 ms length frames with a Hamming window and 50% overlap. The signal is the upper plot, and the windowed frames are the five waveforms below.

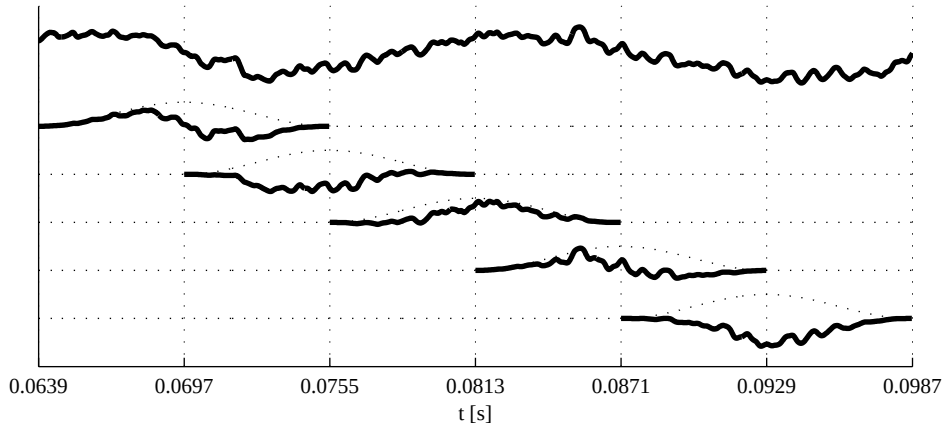


Figure 2.10: STFT time domain frame extraction

After windowing, each frame is FFT transformed to form a column in the spectrogram. The purpose of windowing the frame before transforming is to remove the sharp edges that would otherwise be present at the frame edges. In fact, not windowing the frame really means using a rectangular window. By choosing some other function the frequency domain distortion can be reduced. The Hamming window, equation (2.12), is a good choice for STFT since it smooth, symmetric, and satisfies the COLA[1] property at 50% overlap, meaning that all samples of the signal are weighted equally.

$$w[n] = 0.54 - 0.46cos(\frac{2\pi n}{N-1}) \tag{2.12}$$

[1]Constant-Overlap-Add

2.1.3 Energy Based Methods

Spectral energy based methods use the time-frequency representation of the audio stream to generate a detection function. Only the magnitudes of the bins are used, phase information is simply discarded.

Plain Spectral Difference

The spectral difference is a simple yet powerful approach to onset detection. It exploits the fact that transient attacks are broadband events that give a rise in energy over all the frequency spectrum. It is calculated by differentiating all frequency bands and then summing the differences together, according to equation (2.13). Figure 2.11(a) shows the result of this operation on the spectrum from figure 2.9(b). As you can see, the onsets are clearly visible, but there are also a number of disturbing peaks. This detection function was calculated with an 11.6 ms running window with 50% overlap.

$$d[n] = \text{HWR}\left(\sum_{k=f_{min}}^{f_{max}} |X[n,k]| - |X[n,k-1]|\right) \tag{2.13}$$

Spectral Energy Flux

Spectral energy flux is a small but intelligent improvement to the plain spectral difference. Before differentiation each bin is compressed (by means of square root). This method was used by Laroche (2003) with convincing results.

$$d[n] = \text{HWR}\left(\sum_{k=f_{min}}^{f_{max}} \sqrt{|X[n,k]|} - \sqrt{|X[n-1,k]|}\right) \tag{2.14}$$

The resulting function is depicted in figure 2.11(b). Notice how the spurious peaks are attenuated compared to plain spectral difference.

High Frequency Content

Masri and Bateman (1996) proposed construction of an onset detection function based on two properties. Firstly, the increase in high frequency content, and secondly the amount of high frequency content in relation the the overall energy of the signal. This approach is based on ratios, not differences. To determine these properties a time-frequency representation is used.

From each STFT frame, the total energy is calculated as the sum of the bin amplitudes squared:

$$E[n] = \sum_{k=f_{min}}^{f_{max}} (X[n,k])^2 \tag{2.15}$$

where the summation range $[f_{min}, f_{max}]$ is selected so that non-audible frequencies are excluded.

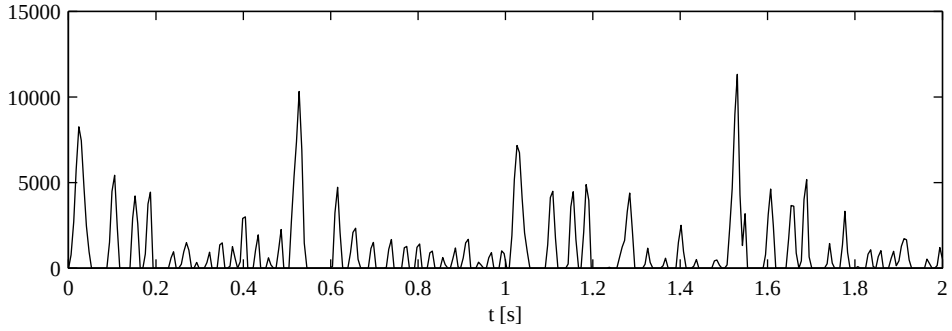
By weighting the frequencies linearly, a measure of the high frequency content is retrieved:

$$HFC[n] = \sum_{k=f_{min}}^{f_{max}} k|X[n,k]|^2 \tag{2.16}$$

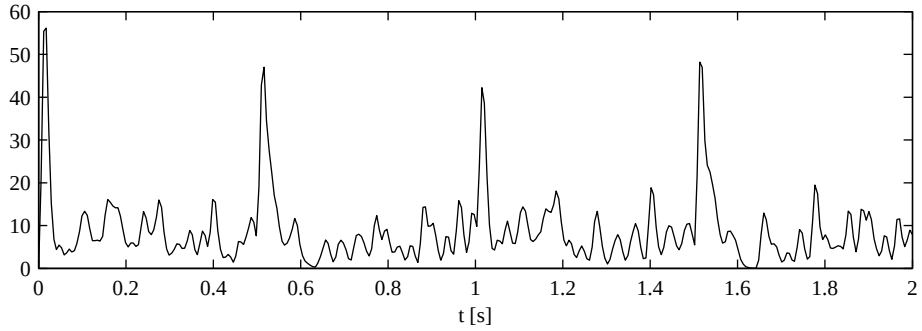
The detection function is then constructed as the product of two properties. The first one is the HFC ratio between two consecutive frames. This will capture the broad band energy burst associated with transients. The second property is the relation between HFC and overall energy. This will attenuate the degree of onset if the energy distribution is not biased towards higher frequencies.

$$d[n] = \left(\frac{HFC[n]}{HFC[n-1]}\right)\left(\frac{HFC[n]}{E[n]}\right) \tag{2.17}$$

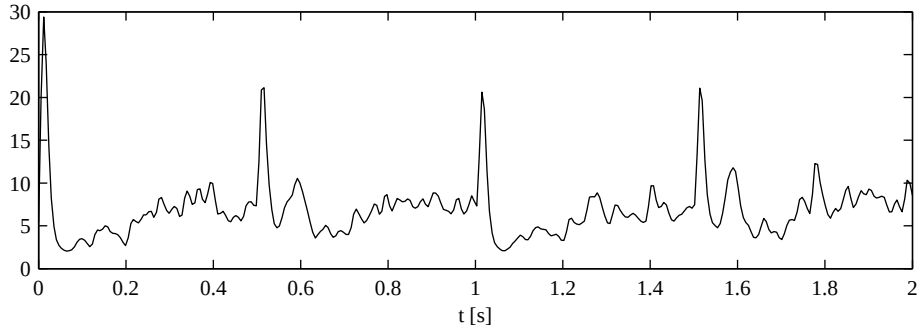
Figure 2.11(c) shows the HFC detection function.



(a) Plain Spectral Difference



(b) Spectral Energy Flux



(c) High Frequency Content

Figure 2.11: Detection functions based on spectral energy analysis

2.1.4 Phase Based Methods

Although energy methods perform well for a wide range of input, they have some shortcomings. There are note onsets, such as those from non-percussive instruments, which do not generate rapid energy increase. Detection methods which consider the phase information in the time-frequency representation have proven to perform well in this case.

Phase deviation

Phase based onset detection was presented by Bello and Sandler (2003). It is a method that relies on the unpredictiveness of signal phase in transient regions, in contrast to phase behavior in steady-state regions, which is said to be predictable to a large extent.

Each FFT bin k, holds a complex number consisting of amplitude and phase. The amplitude is neglected here, and an unwrapped[2] phase value $\tilde{\varphi}(n,k)$ is extracted. For each k, a phase prediction based on the two previous frames is calculated. Ideally, the phase should progress linearly with time (see figure 2.12); that is the case for steady oscillations. Thus the difference between predicted phase, and that which is actually present, is a measure that describes the "unpredictiveness" of the signal. This is described by equation (2.18).

$$\Delta\varphi[n,k] = \underbrace{\tilde{\varphi}[n,k]}_{\text{measure}} - \underbrace{2\tilde{\varphi}[n-1,k] + \tilde{\varphi}[n-2,k]}_{\text{prediction}} \tag{2.18}$$

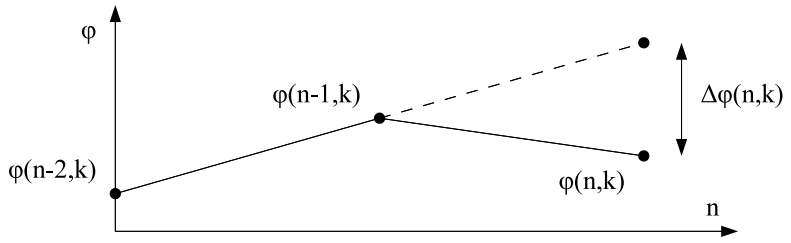


Figure 2.12: Phase prediction difference

The distribution of phase deviations inside the given frame n is then examined. A transient frame should have high phase deviation for several FFT bins, while a steady-state frame should contain phase deviations mostly spread around zero. Thus the phase deviation distribution consists of a peak at zero for steady-state, and a more spread out (flat) distribution for transients. The mean value measures this, as it is larger when the values are distributed further away from zero, while being close to zero for the peak distribution. The mean for frame n is given as $\mu[n]$ in equation (2.19).

$$\mu[n] = \sum_k |\Delta\varphi[n,k]| \tag{2.19}$$

This method works well in many cases, but Lee and Kuo (2006) showed that it does not hold when there are several sinusoids of relatively equal amplitude inside one FFT bin. The method is reliable only when there is a single dominant frequency component in

[2] see (Smith, 2007b, sec. 7.6.2) for explanation of phase unwrapping

the bin (other frequency components must be of negligible amplitude). This can be taken into account by weighting the phase deviation with the energy in the bin, as high energy suggests the presence of a dominant sinusoidal component. This detection function is given by equation (2.20).

$$d[n] = \frac{\sum_k \left(|X[n,k]|\ |\Delta\varphi[n,k]|\right)}{\sum_k |X[n,k]|} \tag{2.20}$$

Complex Prediction Difference

This method, developed by Bello et al. (2004), can be seen as the union of the spectral difference and phase deviation approaches. This is said to be a better measure than the the previous individually, as they make up for each others weaknesses. The spectral difference is very good for onsets with a rapid increase in amplitude, while the phase deviation is a better measure for soft, non-percussive onsets. The idea is to create a prediction $\hat{X}[n,k]$, based on previous input, and then measure the distance to what is actually measured. The prediction at a given STFT bin has the same amplitude as the same bin one frame earlier, while its phase is estimated as linearly progressing as in equation (2.18). The prediction $\hat{X}[n,k]$ is given by (2.21).

$$\hat{X}[n,k] = |X[n-1,k]| \angle\, 2\tilde{\varphi}[n-1,k] + \tilde{\varphi}[n-2,k] \tag{2.21}$$

The prediction differences are then summed over the frequency spectrum to produce a quantity (2.22) that is large for transients and small for steady-state.

$$\Delta[n] = \sum_k |X[n,k] - \hat{X}[n,k]| \tag{2.22}$$

2.1.5 Peak picking

Peak picking is not necessarily a part of a beat detection system. Many approaches, such as (Scheirer, 1998; Laroche, 2003; Alonso et al., 2004; Davies and Plumbley, 2007) use the continuous detection function as an input to metrical analysis. There are also several authors that have chosen to use explicit peak detection, including (Masri and Bateman, 1996; Seppänen, 2001; Uhle and Herre, 2003; Bello et al., 2004; Tanghe et al., 2005). An obvious advantage of using peak picking is that it results in significant data reduction. A time frame that would require thousands of symbols to describe with the continuous function is reduced to a handful of symbols describing peak locations and amplitudes.

Peaks are often clearly visible to the eye in detection function plots. But setting up conditions for their detection in a causal context is not that trivial.

Normally some amplitude level is selected as a threshold for peak detection. Both (Klapuri, 1999) and (Seppänen, 2001) use constant threshold levels. This requires good knowledge about the amplitude levels in the detection function, something that can be accomplished by normalizing the input.

It is more common to use some kind of adaptive threshold which is a function of past input.

A simple approach is to use the (scaled) signal mean as a threshold, as shown in figure 2.13(a). A running median filter is an even better option. It has the advantage of being

able to capture steps while masking impulses. Figure 2.13(b) shows the median over 51 samples scaled with a factor of 3. If you compare with the mean threshold in figure 2.13(a) (which has the same window and scale) you'll notice that the median is more stable and does not change notably at the peak locations, which is good. On the downside, the median filter is more difficult to compute. The values in the range needs to be sorted and then the middle value is selected as the median.

A peak-hold filter is also an alternative. This filter selects the output as the maximum of its previous output scaled with an exponential decay, and its input:

$$\Theta[n] = \max\left\{c_d\Theta[n-1], d[n]\right\}. \tag{2.23}$$

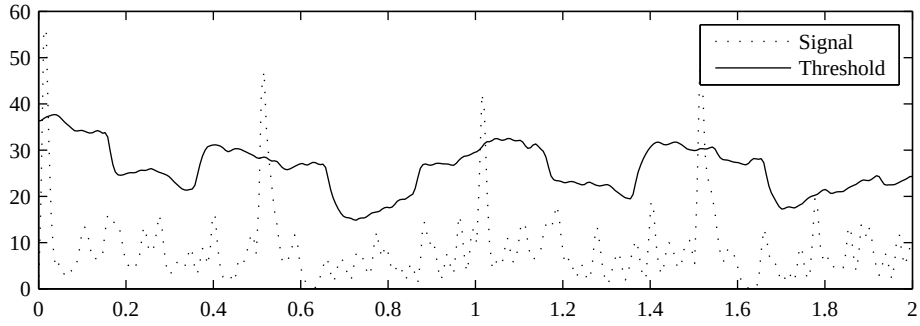
where $\Theta[n]$ is the threshold level, $d[n]$ is the detection function, and c_d is the decay constant.

This means that when the the threshold is exceeded by the signal, the threshold will be set to the signal level. The threshold will then decay exponentially until it is exceeded by the signal again. The decay constant c_d for a half-life t_{HL} is calculated according to equation (2.24). This method has the advantage of adapting to different input levels very quickly, and it is also possible to tune the decay to favor some certain BPM value. The main problem with this method is that very strong peaks will raise the threshold so much that peaks following closely after might go undetected.

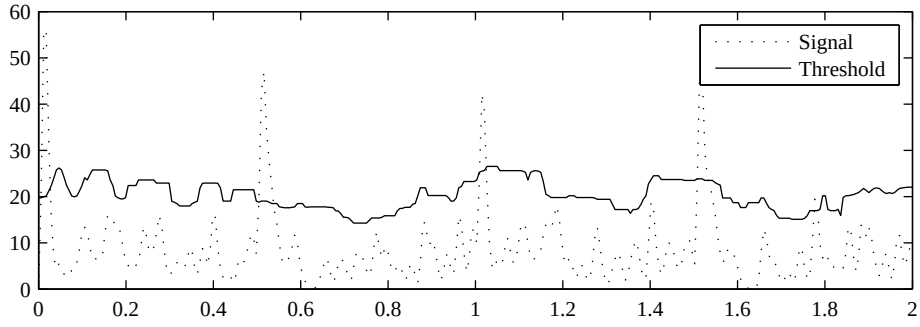
$$c_d^{t_{\text{HL}}} = 0.5 \quad \Longleftrightarrow \quad c_d = \exp\frac{\log 0.5}{t_{\text{HL}}} \tag{2.24}$$

Figure 2.13(c) shows an example of this thresholding method, with the exponential decay constant set so that the threshold has a half-life of 0.5 seconds.

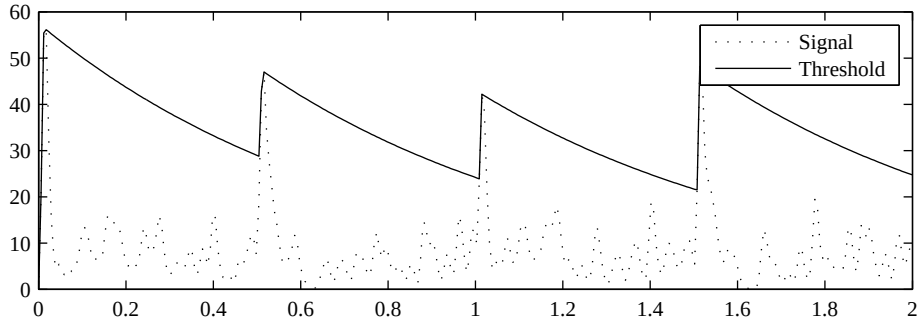
It is also common to simulate hysteresis by introducing a lower threshold which the signal needs to fall below before another peak detection is possible. This makes sense when thresholding the envelope derivative, as it is normally expected to fall below zero between onsets. It should, however, be used with care when thresholding energy, because then it is difficult to say if and how much it will fall between two onsets.



(a) Mean



(b) Median



(c) Peak-hold

Figure 2.13: Different methods for thresholding

2.2 Tempo estimation

In this section, "input signal" refers to the onset detection function, *not* the original audio stream which constituted the input in section 2.1.

2.2.1 Autocorrelation Function

Brown (1993) investigated the use of autocorrelation as a method for determination of periodicity in music. The analysis was performed on symbolic data rather than sampled music, but it shows that for error-free input, the tempo can be extracted through peak-picking inside the autocorrelation of a window covering a few beat periods.

The reason why autocorrelation can be used for tempo detection is that it finds periodicity within a signal. If the signal exhibits a strong degree of periodicity at an interval of τ samples, then the autocorrelation function (acf) of the signal will have a peak at lag (index) τ, and also at multiples of τ (fractions of tempo). The tempo can thus be decided by peak-picking in the acf.

Correlation is time reversed convolution, so autocorrelation can be expressed as a sum of products,

$$a[l] = \sum_{i=-\infty}^{\infty} \left(d[i]\; d[i-l]\right) \tag{2.25}$$

where $a[l]$ is the autocorrelation at lag l, and $d[i]$ is the signal sample i. This cannot be calculated for all the whole signal at once for practical reasons, so a frame is extracted from the signal and analyzed. This is expressed as

$$a[l] = \frac{1}{L-l} \sum_{i=n-L}^{n-L-1} \left(d[i]\; d[i-l]\right). \tag{2.26}$$

where n is the current sample in the input signal, and L is the length of the analysis window. The sum in equation 2.26 is biased, because zeros are inserted when $d[i-l]$ is shifted. The scaling factor $(L-l)^{-1}$ makes the acf unbiased, but $a[l]$ for l close to L are still uncertain because they have been calculated with few samples as basis. For this reason, $a[l]$ should not be calculated for all $l < L$, but only for $l < \frac{2}{3}L$ or so, i.e. the window must be significantly longer than the range of lags to be analyzed.

Autocorrelation can also be calculated in the frequency domain (Smith, 2007b), as has its frequency domain equivalent in conjugation. This is a three-step process:

$$\begin{aligned} D[k] &= \mathcal{F}\left\{d\right\}[k], \\ A[k] &= D[k]\; D[k]^*, \\ a[l] &= \left(\frac{1}{L-l}\right) \mathcal{F}^{-1}\left\{A\right\}[l]. \end{aligned} \tag{2.27}$$

Davies and Plumbley (2007) has successfully used autocorrelation for tempo estimation in music from the onset detection function suggested by Bello et al. (2004). In that approach, the tempo was selected by investigating which acf lag, together with their multiples, was the strongest.

There are a few problems associated with using autocorrelation for tempo estimation. In order to get reliable results, the analysis window needs to be relatively long so that it contains at least a few quarter beats at the slowest plausible tempo. As an example, 4 beat periods at 40 BPM are 6 seconds long. The analysis must also take place quite often, as the user requires fast response on tempo changes (perhaps 0.5 s). In order to process 6 s of audio twice every second, the detection function must be heavily downsampled.

2.2.2 Comb Filter Bank

Periodicity estimation by the means of a comb filter bank has been implemented in (Scheirer, 1998; Klapuri et al., 2006) among others. The idea is to feed the continuous onset detection function into a bank of comb filters. Each comb filter has a delay that corresponds to a certain tempo. If there is a strong periodicity at some tempo in the detection function, the comb filter corresponding to that specific tempo will resonate and give strong output.

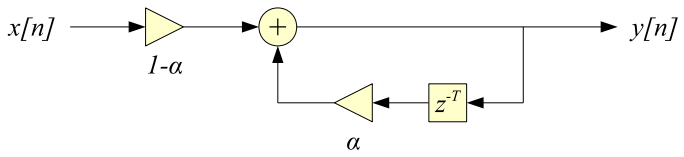


Figure 2.14: Comb Filter

The comb filters are implemented with positive feedback, according to figure 2.14, and the following equation:

$$h_\tau = \frac{1-\alpha}{1-\alpha z^{-\tau}} \tag{2.28}$$

where α sets the decay rate and τ the delay time in samples.

If the signal $x[n]$ has a periodic recurrence of peaks with a period corresponding to τ, it will align with the filter impulse response in the convolution sum and produce high level output. Fractions and multiples of τ will also resonate (which makes it different from autocorrelation, where only multiples resonate), but weaker. If the periodics of x does not have any relation to τ, the filter output will be of low amplitude.

A bank of filters h_τ is set up for a range of τ values corresponding to the range of tempos that should be investigated. The index τ of the filter with the maximum output is selected as the tempo estimation (remember that the tempo is just the inverse period).

The comb filtering method of tempo estimation has proven to give reliable results. The main problem with this method is the large amount of comb filters, which are not expensive to calculate, but consumes a lot of memory because each filter needs to have its own delay line of length τ.

2.2.3 Inter Onset Intervals

Seppänen (2001) suggested explicit discrimination of onset locations and using them to extract a large number of inter onset interval (IOI) times. Inter arrival times are extracted from all combinations of the latest few onset times and collected in a histogram. The IOIs are extracted band-wise, but the histogram is global. Based on the fact that all onsets are

(ideally) aligned to the tatum grid, all IOI times should be integer multiples of the tatum period. So the histogram examined for a greatest common divisor through an approximate minimum square error algorithm. The histogram is designed to decay with time so that weight of old input decreases, and becomes more adaptable to changes, making it possible to follow variations in tempo. Informal tests imply that the algorithm proposed by Seppänen (2001) performs well.

Desain (1992) developed a causal method based on creating an "expectancy function" at every onset in the music based on a small number of recent onset locations. The basic idea is that, if an onset occurred n samples ago, then there is some probability that another onset will occur n samples from now in the future. Basically, a probability function of a future onset is constructed for a small time frame, with a large peak at t and smaller peaks at multiples an fractions. Such a probability function is created by evaluating every pair of the set of most recent onsets. The functions are added, and the sum is a sensible function describing the probability of an onset in the near future.

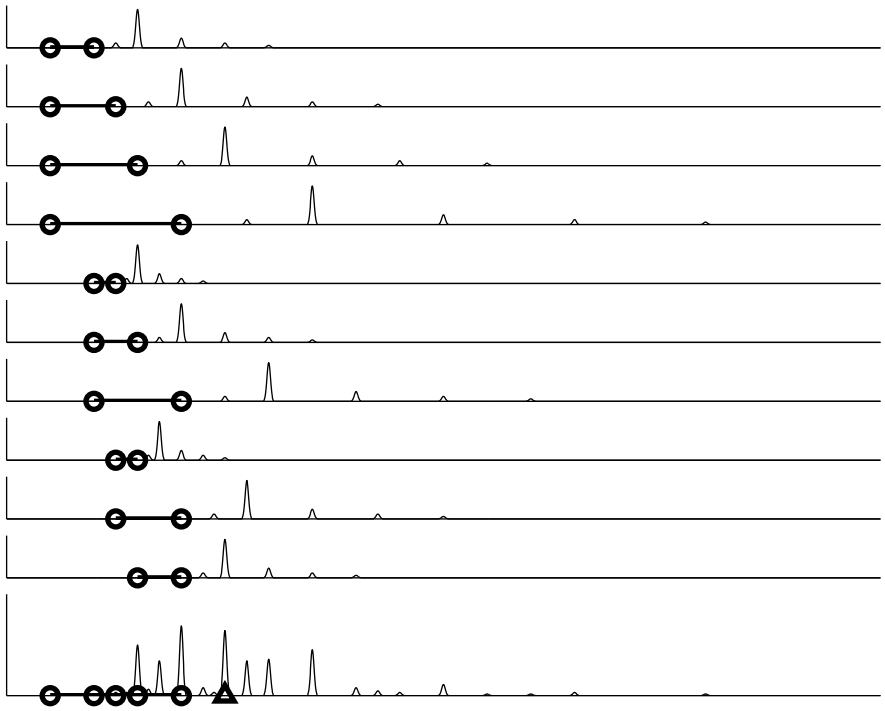


Figure 2.15: Onset expectancy function (Desain, 1992)

Figure 2.15 shows the expectancy for a a small number of onsets. Each subplot is the basic expectancy of an interval (marked as thick line between two onset marked as circles). The lower plot is the sum of expectancies, with the strongest future prediction marked with a triangle.

Jensen and Andersen (2004) presented a method extending that of Desain (1992). Their system is causal, and holds a "beat probability vector" which is updated at every onset detection. The update is made such that a set of IOIs is formed from a) the current onset location, and b) a small number N of onsets back in time. With their centers at locations corresponding to these IOIs, Gaussian pdf shapes are added to the vector. These pdfs are weighted as the product of the weights of the onsets from which the IOI was formed. The values that are already in the histogram are scaled with a value creating a gradual decay. The decay is a function of time elapsed since the previous onset detection. The beat probability is recalculated upon onset detection as

$$P_B[i] = \overbrace{c^{t_k - t_{k-1}}}^{\text{leakage-scaling}} P_B[i] + \sum_{j=1}^{N} \underbrace{A_k A_{k-j}}_{\text{combined weight}} \underbrace{G[\overbrace{|t_k - t_{k-i}|}^{\text{distance}} - i]}_{\text{Gaussian pdf}} \tag{2.29}$$

where A_k and t_k are amplitudes and times for the onsets, G is the Gaussian pdf, and c is the decay constant.

The amount of time to update the histogram at every onset is the length of the Gaussian probability density function (pdf) times the number of IOIs considered. This is a small number compared to many other approaches, and there are no divisions, square roots, exponentials or other complicated operations involved. The pdf can be stored as a lookup table in RAM.

Figure 2.16 shows an example of how the beat probability evolves over the 20 seconds of a song. The true tempo (130.5 BPM, 0.46 s) is not dominating, but the half-tempo (60.25 BPM, 0.92 s) is clearly visible, and you can also see how spurious onsets arrive and decay. Time is not linear in this figure; one time unit corresponds to one detected onset. The decay effect has been exaggerated to make it more visible.

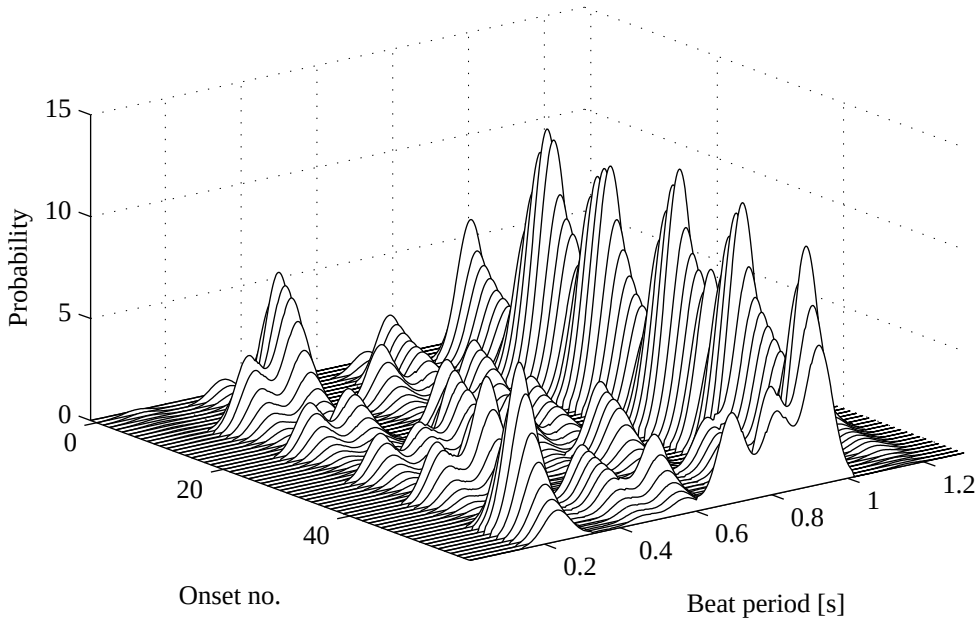


Figure 2.16: Beat probability vector, extracted from "Blue Monday" by New Order

2.3 Meter estimation

Brown (1993) showed that the autocorrelation of a musical piece is useful for finding its time signature. This section describes the behavior of the acf for different time signatures.

In the example plots of this section, the input detection function is constructed from a MIDI file format representation of the song. Onset locations where retrieved from the MIDI file in Matlab using the MIDI parser by Schutte (2009). An artificial detection function was then constructed by inserting ones at onset locations, leaving the rest of the signal zero-filled. If several onsets coincide they are added together. Finally the signal is convolved with a 50ms Hann-window to make it a little bit smoother and dissolve onsets that are very close to each other.

All autocorrelation plots in figures 2.17, 2.18, and 2.19 are constructed from 10 s excerpts of the respective songs. The autocorrelations are unbiased, and the plots are cropped to show a time frame appropriate to the respective meter.

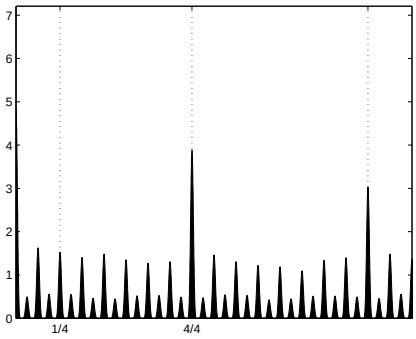
2.3.1 Duple/quadruple meter

Figure 2.17(a) shows the autocorrelation of a song in (4/4). You can see that the bar is divided into 4 quarter notes, which are in term subdivided into 2 eighth notes or 4 sixteenths. This is the most common time signature in western popular music. The peaks corresponding to the bar length is noticeably strong compared to the others. This is a feature which could possibly be used to determine the meter of music in general.

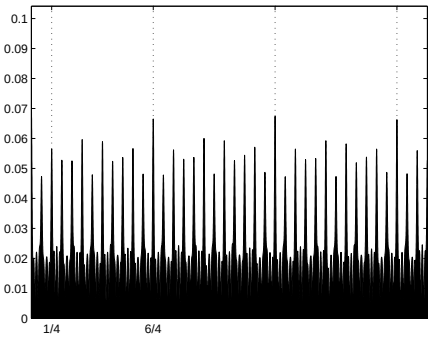
Compare with figure (d) which has the more uncommon *compound* duple meter. Here there are also 4 quarter notes per bar, even though it's not evident in the graph. Notice however that there are 3 eighth notes per quarter, which makes it different from the *simple* duple meter in figure (a). This introduces a problem for some IOI based tempo detectors, as the quarter beat tempo is not an octave of the eighth beat tempo. Such detectors often work by simply doubling intervals below a certain threshold, which would result in the tempo estimation being a factor of 1.5 off from the true value. If however, the tempo has been correctly estimated, the observation that bar length lags in the autocorrelation exhibits strong peaks can be used to determine the meter.

2.3.2 Triple meter

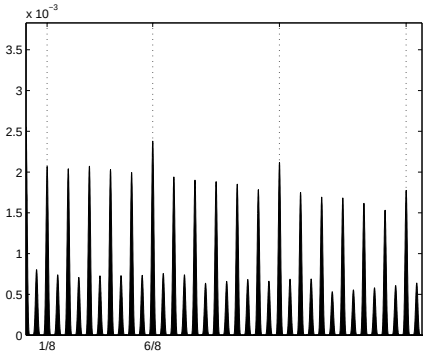
Figure 2.18 shows the autocorrelation of two songs with triple meter. Figure (a) is a piano track with no other instruments and it is of the time signature (3/4), commonly known as waltz meter. Just as in figure 2.17(a) you can see strong peaks at lags corresponding to the bar length (in this case quarter note times 3). The quarter is subdivided into 2 eighths, just as in simple duple meter. The music in figure (b) is of the time signature (9/8). Here the quarter note is divided into three eighths, just as in the previously mentioned compound duple meter. Three quarter periods form the bar, and we could again find the bar length by looking for peaks that are high and multiples of the quarter. This is not a very common time signature.



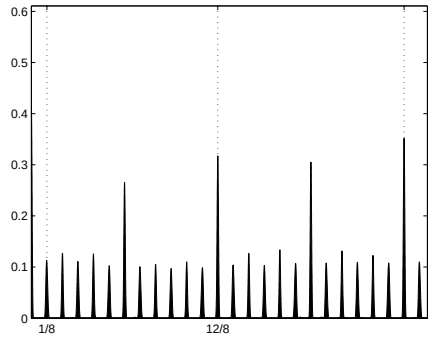
(a) “Tour de France” by Kraftwerk (4/4)



(b) “Schism” by Tool (6/4)

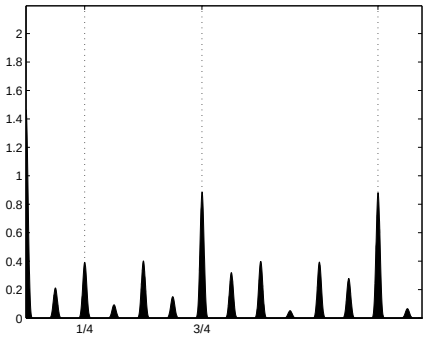


(c) “House of the Rising Sun” by Animals (6/8)

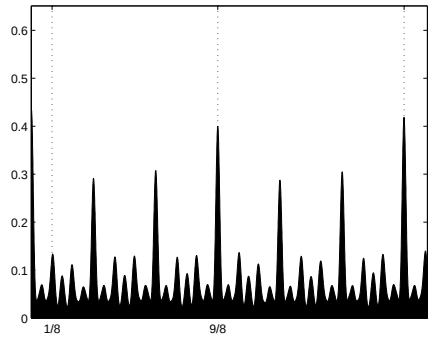


(d) “Everybody Hurts” by R.E.M. (12/8)

Figure 2.17: Autocorrelation of songs with duple/quadruple meter



(a) “Piano Man” by Billy Joel
simple (3/4)



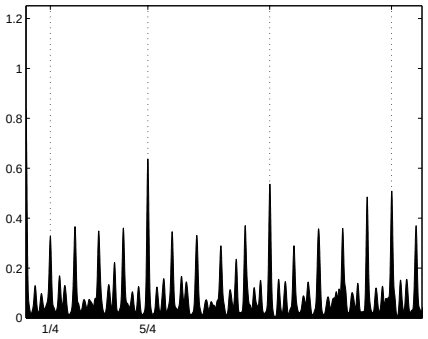
(b) “Ride of the Valkyries” by Richard Wagner
compound (9/8)

Figure 2.18: Autocorrelation of songs with triple meter

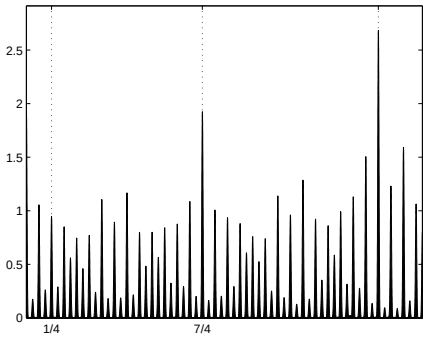
2.3.3 Complex meters

Figures 2.19 shows examples of songs with complex time signatures. In general the number of quarters per bar is a prime number, and they are more difficult to “feel” for the listener. These examples were chosen to support the idea of detecting the number of beats

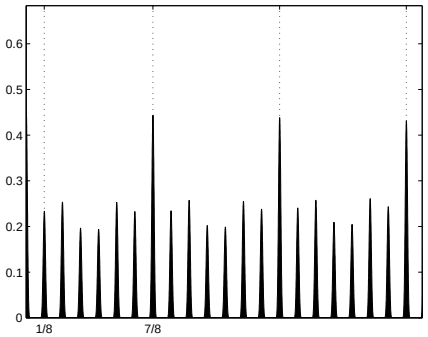
per bar by examining the tempo multiples of the quarter beat.



(a) “Take 5” by The Dave Brubeck Quartet (5/4)



(b) “Solsbury Hill” by Peter Gabriel (7/4)



(c) “Jocko Homo” by Devo (7/8)

Figure 2.19: Autocorrelation of songs with less common time signature

2.4 Synchronization

Several authors have pointed out synchronization as an application for beat detection systems. But, based on the literature study prior to this report, it seems very few papers actually dealt with the problem.

For synchronization, both the beat phase and the tempo must be known. But even if you start the sequencer with the right tempo and at the right moment, it will soon be out of synchronization with the music. This is because the tempo is not perfectly matched. Even a misjudgment of the beat period of only 1 hundredth of second would be disastrous in such a set-up. At 120 BPM it would only take a few seconds before the sequencer has drifted so far from the music that it’s clearly audible.

2.4.1 Phase-locked loop

The problem of keeping the sequencer synchronized to the input tempo bares some resemblance with the problem of keeping clock oscillators synchronized in a digital electronic system. In such designs, it is common to have a reference oscillator of some frequency,

and use it to create some higher frequency oscillation with a frequency that is an integer multiple of the reference. The oscillators needs to be phase-aligned, but the reference contains imperfections. The oscillator must be continuously adjusted to the reference in order to avoid skewing and drifting. As a solution to this problem, a control system called phase-locked loop (PLL) is used.

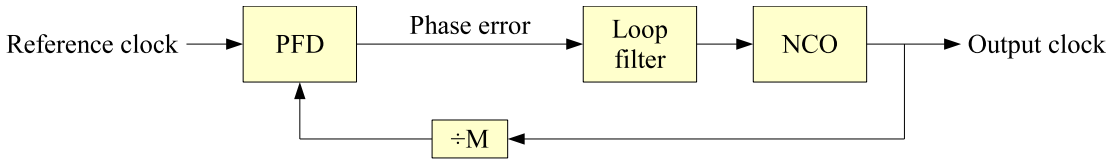


Figure 2.20: Typical phase-locked loop control system

Figure 2.20 shows a typical PLL design. The phase frequency detector (PFD) measures the phase difference between the reference and output clocks (for instance with the help of an XOR-gate for square waveforms). This error is the fed through a loop filter to give a certain error response (removing jitter etc). The output of the loop filter tells the numerically controlled oscillator (NCO) to increase or decrease its frequency to improve phase alignment. To apply this to the problem of synchronizing a sequencer to music, the PLL concept must be placed in a different context. The reference clock is the musical beat, and the output clock is the MIDI timing messages.

2.4.2 MIDI clock

MIDI is a standard for communication between electronic music instruments. The actual communication is achieved through messages sent over unidirectional serial buses. Hass (2005, p. 8) describes the messages used for controlling the progress of a sequencer in real time: *start*, *stop* and *clock*. A sequencer moves forward in discrete time steps of 1/24 of a quarter note. Every time a sequencer receives a clock message, it advances the song position one discrete time step. Thus, the unit controlling the time should send 24 clock messages per quarter note at the desired tempo.

Chapter 3

SYSTEM DESIGN AND IMPLEMENTATION

As the name implies, this chapter serves a dual purpose. It describes the system design, that is a detailed explanation of its operation. It also accounts for how the system has been implemented in practice.

The system takes sampled audio as input, and outputs clock messages on a MIDI bus. The task of the system is to output the clock messages such that they are synchronized to the beat of the input music.

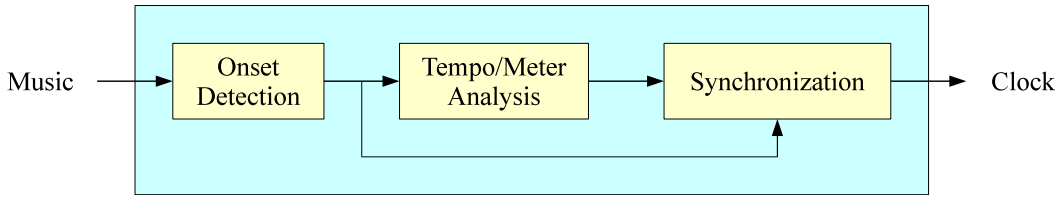


Figure 3.1: General system overview

This functionality is accomplished with three main software modules: *onset detection*, *tempo/meter analysis* and *synchronization*. Figure 3.1 shows the data flow.

The onset detection module extracts explicit onset locations from the audio stream. This is accomplished by the subband envelope method described in section 2.1.1.

The tempo/meter analysis module determines the tempo and metrical structure of the music based on the onset locations. The tempo is estimated with a beat probability vector (section 2.2.3), and the meter is found using autocorrelation (section 2.3).

The synchronization module synchronizes the output clock to the detected onsets at the estimated tempo. A kind of PLL design is used here.

The software runs on the DSP platform described in section 1.6.2.

3.1 General Implementation Aspects

3.1.1 DSP Operation

The DSP is a CPU designed especially for signal processing. It has two data-memories (named x and y) that can be read simultaneously into data registers to be used in the

following cycle. Figure 3.2 shows the MAC operation used for implementing filters on the DSP. The signal is placed in one memory and the coefficients into the other. Upon each clock cycle, the product of one coefficient and one signal sample is multiplied and added to the accumulator. When the filter output has been calculated, it is saturated and quantized to 24 bits before written back to the signal memory.

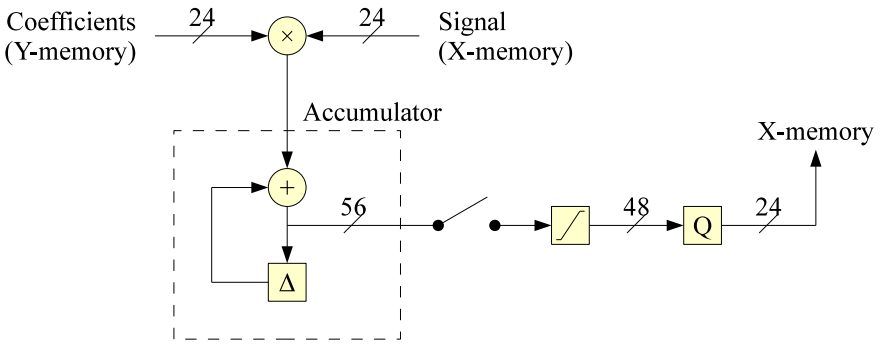


Figure 3.2: DSP multiply-accumulate operation

3.1.2 Task Scheduling

The system uses two tasks in a form of basic multitasking scheme. Task 1 handles beat detection, synchronization and thresholding, while task 2 handles the tempo and meter analysis. Task 2 is started at system startup and uses polling to wait for a successful beat detection. Task 1 is triggered by the event that a frame of samples is available in memory. This event sets off a direct memory access (DMA) interrupt request (IRQ), causing the CPU to push its program counter and status register onto the stack and begin execution of task 1. Task 1 saves the context of task 2 in memory, and then performs beat detection on the input samples. If a beat is detected, a flag is set to tell task 2 to begin execution (task 2 is still blocked by task 1 though). When task 1 is done, it restores the context of task 2 and performs an RTI (return from interrupt) to continue execution in task 2.

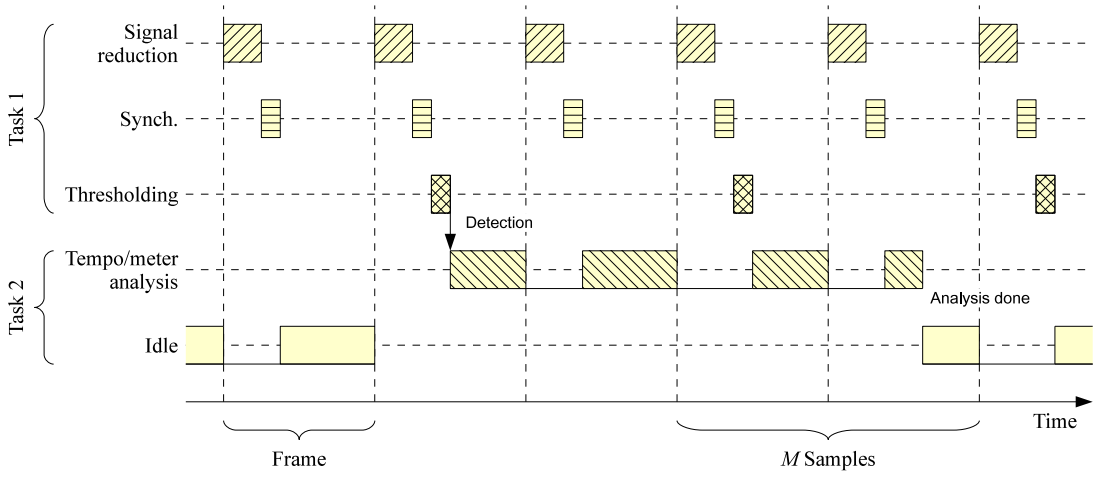


Figure 3.3: Task scheduling

3.2 Onset Detection

The onset detector takes the full resolution audio signal as input and outputs a sequence of onsets described by their *location* and *weight*. The module is described in principle by figure 3.4; the input audio x is processed bandwise and joined into a detection function d. Peaks are extracted from the detection function and collected in the onset queue.

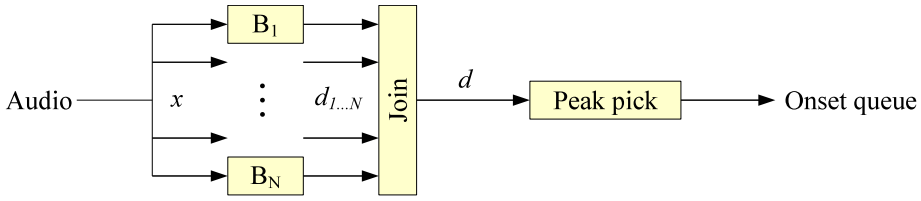


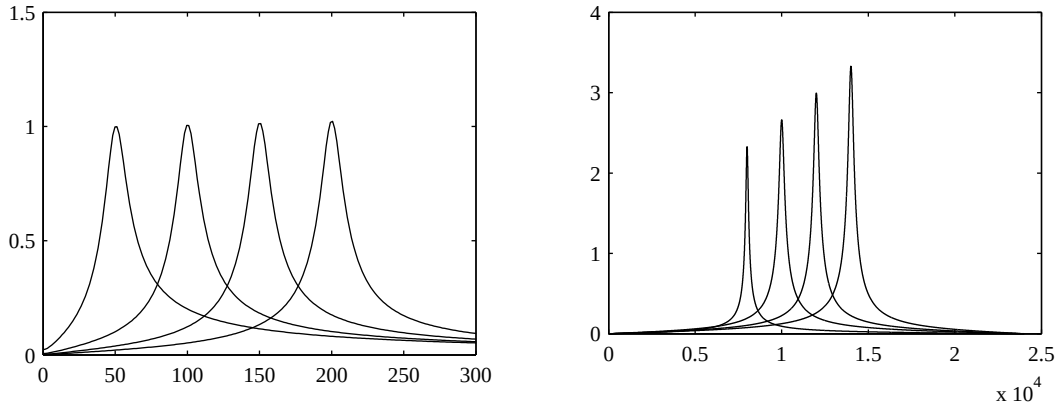
Figure 3.4: Onset detector overview

3.2.1 Detection function

The stereo input is mixed to mono by summation ($x[t] = \frac{1}{2}x_{\text{left}}[t] + \frac{1}{2}x_{\text{right}}[t]$). This leads to both destructive and constructive effects in different frequency regions because of phase differences between the channels.

Frequency band decomposition

In this design, the bands have been chosen to favor voice separation, with the sacrifice of good frequency coverage. A small number of narrow bands are selected, their layout is depicted in figure 3.5.



(a) Lower bands

(b) Upper bands

Figure 3.5: Frequency bands

The LF bands are selected to emphasize kick drums. This instrument is used in most popular music, and it as an important carrier of rhythmic information. Tanghe et al. (2005)

performed extensive measurements to arrive at average spectra for different types of drum sounds. Based on this information, the resonance filters are designed to have their peak amplitudes at 50-200 Hz. The reason why there are four bands is that the kick drum onsets may be masked by other bass instruments, but with several bands the probability increases that it will be isolated in at least one of them.

The HF bands are selected to capture onsets in general. Strong transients create energy peaks in this area of the spectrum. The amount of energy is however small, that is why these filters have been boosted compared to the LF filters, compare the amplitudes in figures (a) and (b).

The bands are extracted with a bank of IIR resonators. The filter used is the "constant resonance gain filter" described by Smith (2007b, section B.6.2). Equation (3.1) is the filter transfer function.

$$H(z) = \frac{(1-R)(1-Rz^{-2})}{1-(2R\cos\omega_c)z^{-1}+R^2z^{-2}} \tag{3.1}$$

The parameter R defines the width of the band, the closer R is to 1, the narrower the band. The filter has unity gain at the resonance frequency ω_c.

Envelope extraction and differentiation

From each frequency band signal an envelope function is extracted. Figure 3.6 shows the steps involved in this procedure.

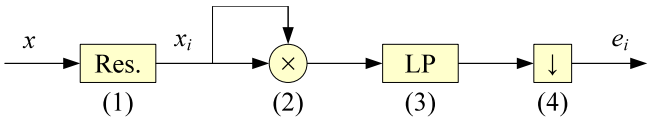


Figure 3.6: Band-wise envelope extraction

The resonator (1) extracts a narrow frequency band, as you can see in figure 3.10(b). The difference equation used for implementing the filter is

$$y[n] = (1-R)x[n] - (R-R^2)x[n-2] + (2R\cos\omega_c)y[n-1] - R^2y[n-2]. \tag{3.2}$$

The resonance filters are implemented as direct form 1 (DF1) biquad sections. Figure 3.7 shows such a filter topology. The Δ-block is a one-sample delayer, and the Q-block is the 24 bit quantization (which also includes saturation upon overflow). For the resonance filter, the coefficient b_1 equals zero so that one MAC operation is actually excluded from the implementation (but included in the picture as it shows the general case).

The reason for choosing this filter topology is that has better noise performance than the DF2 topology on DSPs with quantization only at the accumulator output, as opposed to at every MAC operation (Wilson, 1993).

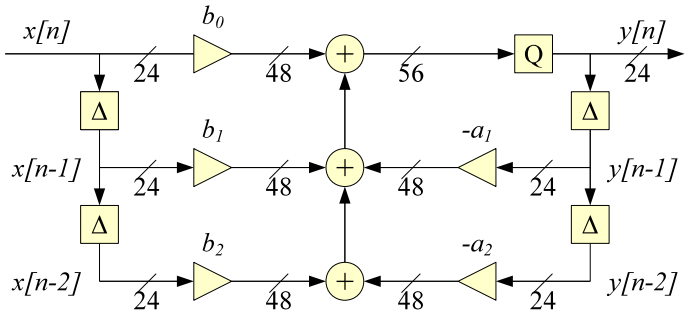


Figure 3.7: Biquad section, direct form 1

The signal square (2) is the first stage of the rectify-and-smooth operation. This is the first step of implementing an RMS approximation according to equation (2.3). The output is the energy of each sample, in the range [0,1[. You'll see in figure 3.10(c) that this signal is expanded compared to the original.

The IIR integrator (3) or smoothing filter is an LP filter with a cut-off frequency of 25 Hz. This filter consists of two second order elliptic filters in series, each having 1.5 dB ripple and 20 dB stop band attenuation. Their combined passband ripple is 3 dB and the stopband attenuation is 40 dB.

The filter has very low cut-off frequency (25 Hz) compared to the sample rate of the signal (48 kHz). Because of this unfavorable ratio, and the DSP using merely 24-bit arithmetics, a standard DF1 implementation gives very poor results. The filter filter would benefit from 48-bit calculations, but that would more than double its computation time. Instead, the performance can be improved by adding first order error feedback (Dattorro, 1988). Figure 3.8 shows how this is achieved.

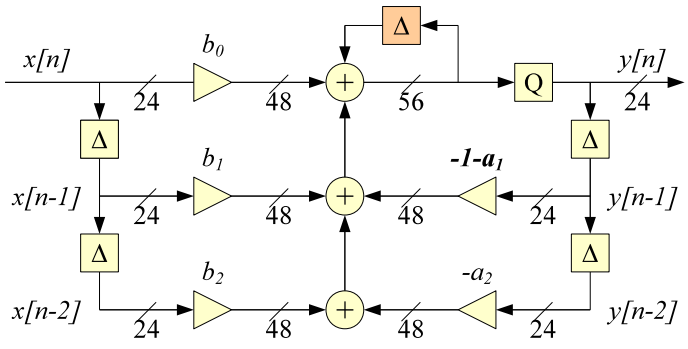


Figure 3.8: Biquad section, direct form 1 with first order error feedback

An extra feed-back link is added **before** the output is quantized. This adds the term $y[n-1]$ with double precision to the output sum. To compensate for this, the value 1 must be subtracted from coefficient a_1 in the standard feedback. For this particular LP filter which has $a_1 \approx -1.9$, the subtraction of 1 also means that the coefficient fits in the range [-1,1[and does not need to be scaled down (which would have been necessary otherwise) so the coefficient maintains maximum precision.

In practice the error feedback is implemented by not clearing the DSP accumulator in the filter loop (see appendix A.1.1). This adds one execution cycle compared to the standard DF1 implementation.

The smoothed envelope can be seen in figure 3.10(d).

Downsampling (4) of the envelope to $48000/128 = 375$ Hz is performed to reduce the amount of data without losing too much temporal precision. There is very little energy above 25 Hz in the envelope thanks to the smoothing filter, but for reasons discussed in section 2.1 this oversampling is necessary.

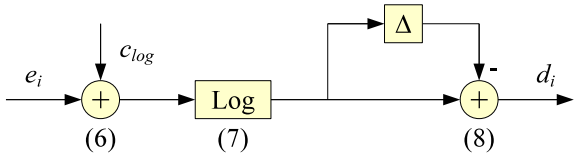


Figure 3.9: Envelope differentiation

Compression (7) is performed only after addition of a small constant (6), to suppress very-low amplitude oscillations taking place just above zero. The actual compression is achieved by taking the binary logarithm. The DSP does not have hardware support for this operation, but it can be approximated with high accuracy.

The integer part of the 2-logarithm can be found by counting the number of leading bits y of a binary number x. The DSP has a special instruction to do precisely that (Freescale Semiconductor, 2005a, ch. 13). Then, to find the fractional part, the number x is first scaled with 2^y,

$$log_2(x) = log_2(2^y x/2^y) = log_2(2^y x) - y, \tag{3.3}$$

to form a number in the range $0.5 \leq 2^y x \leq 1.0$. A polynomial gives the 2-logarithm in the range $[0.5, 1.0]$ with 8 bits of accuracy:

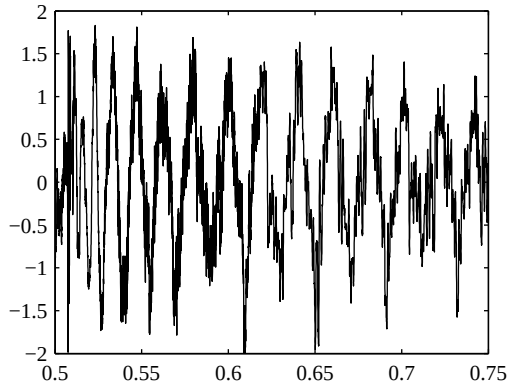
$$\log_2(x) \approx 4(-0.3372223x^2 + 0.9981958x - 0.6626105), \quad 0.5 \leq x \leq 1.0\,. \tag{3.4}$$

These coefficients were found in the DSP programming examples from Freescale, but the same values can be achieved using Matlab's `polyfit`.

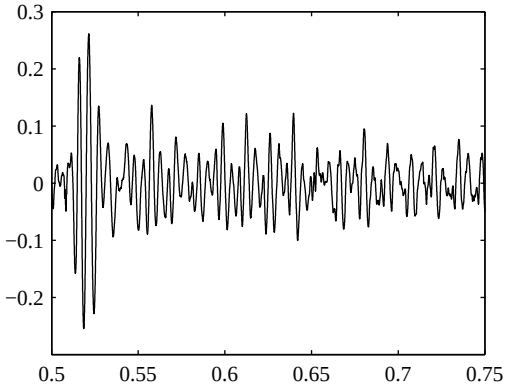
The downsampled and compressed envelope is shown in figure 3.10(e).

Differentiation (8) of the compressed envelope is the final step in the construction of the detection function for the band. The onset is now characterized by a distinct peak in the detection function, see figure 3.10(f).

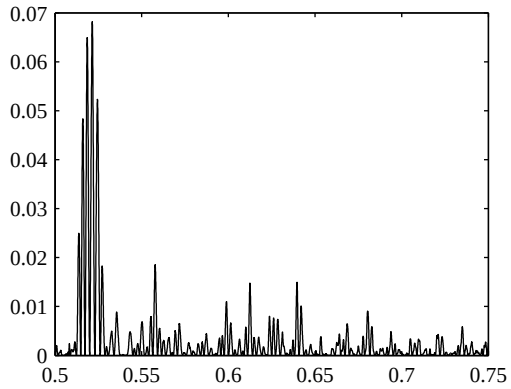
As you may have noticed, the square root operation has been omitted from the RMS approximation here (between steps 5 and 6). This is intentional, as taking the square root before the logarithm is equivalent to scaling the the logarithmic output with 1/2 (as $\log \sqrt{x} = \frac{1}{2} \log x$). The scaling has been omitted as linear scaling has no effect on the peak picking algorithm. It would only serve to reduce the dynamic range of the detection function (which is not desired).



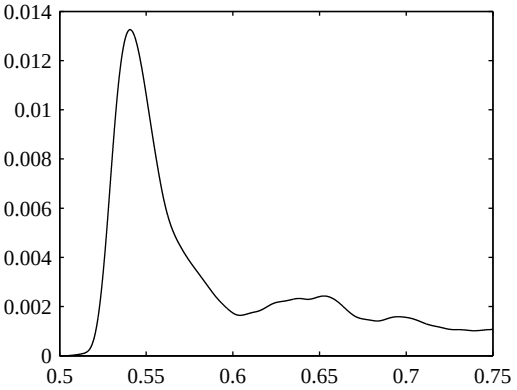
(a) Unaltered audio stream



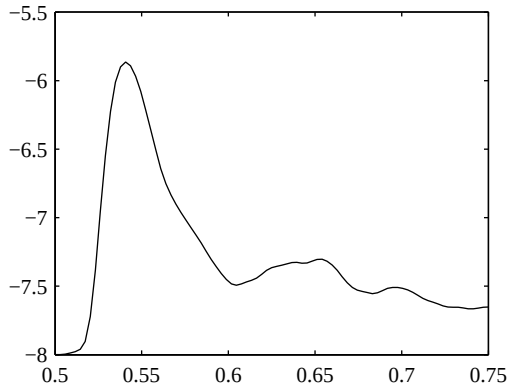
(b) After resonance filter (200 Hz)



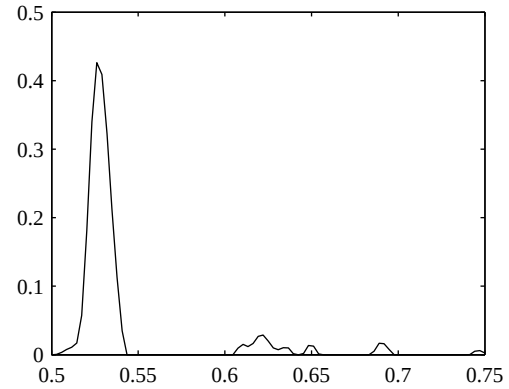
(c) After squaring



(d) After smoothing filter



(e) After downsampling and compression



(f) After differentiation

Figure 3.10: Finding the detection function of a single band

Joining bands

The band-wise differences are joined by summation. Since only positive amplitude changes are of interest the band differences are half wave rectified before summation. If HWR was not applied, a decay in one band could possibly hide the attack of another. This allows

only constructive addition. Figure 3.12 shows how four bands are summed together to form the combined detection function.

$$d[n] = \sum_{i=1}^{N_{bands}} \max\{d_i[n], 0\} \tag{3.5}$$

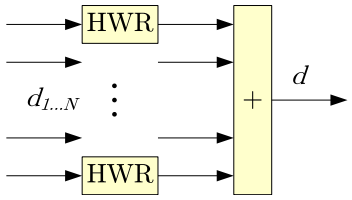


Figure 3.11: Joining differences

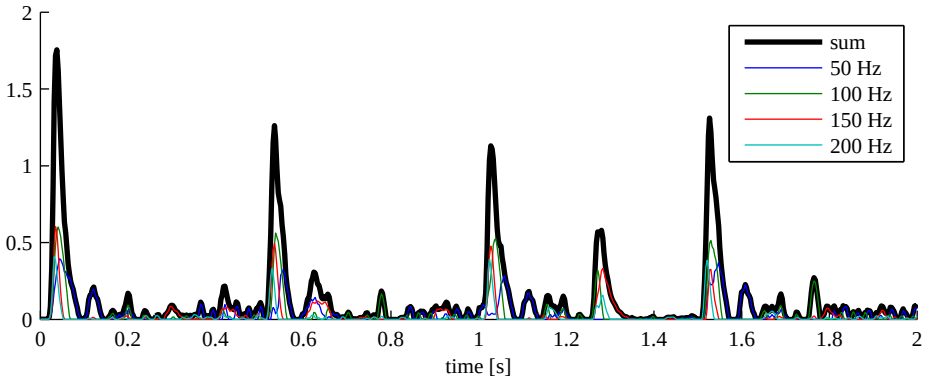


Figure 3.12: Sum of half-wave rectified differences

3.2.2 Thresholding and Peak Picking

The three most recent samples (including the current) are used for thresholding. The actual peak selection is calculated for $d[n-1]$ (delayed one sample), simply because it would otherwise be impossible to determine whether it is a local maximum or not.

The threshold $\Theta[n-1]$ is calculated with a peak-hold filter (described in section 2.1.5) as

$$\Theta[n-1] = \max\{c_d\Theta[n-2],\ d[n-1],\ \Theta_{\text{min}}\} \tag{3.6}$$

where c_d is the decay constant according to equation (2.24), $d[n]$ is the onset detection function, and Θ_{min} is the minimum threshold level.

Each sample in the detection function is tested against three conditions. If all three are true, then the sample is selected as an onset and inserted into the onset queue. The first condition is that the sample must raise the threshold, i.e. be selected as the maximum in equation (3.6). This can be expressed as satisfying the condition

$$c_d\Theta[n-2] < d[n-1] > \Theta_{\text{min}}. \tag{3.7}$$

The second condition is that the sample must be a local maximum:

$$d[n-2] < d[n-1] > d[n]. \tag{3.8}$$

The third and final condition is that at least 50 ms must have passed since the previous detection, i.e.

$$\frac{(n-1) - n_{\text{prev}}}{f_d} > 50 \text{ [ms]} \tag{3.9}$$

where f_d is the sample rate of the detection function, and n_{prev} is the sample number of the most recently detected onset.

The onset queue is implemented as a circular buffer. Whenever a new onset is inserted, the oldest is overwritten and a pointer is updated. The onsets in the buffer are described by two values: *location in time*, and *weight*.

3.3 Tempo and meter estimation

3.3.1 Beat probability vector

The tempo is estimated from an evolving *beat probability vector*, similar to that described by Jensen and Andersen (2004). This vector holds the probability of the beat period being a specific value, such that if bpv[t] is high, then the probability of the beat period being t is high. The index t is an integer of detection function samples. The size of the vector is 2 s (=750 samples), which sets the lower limit of representable tempi to 30 BPM.

IOI insertion

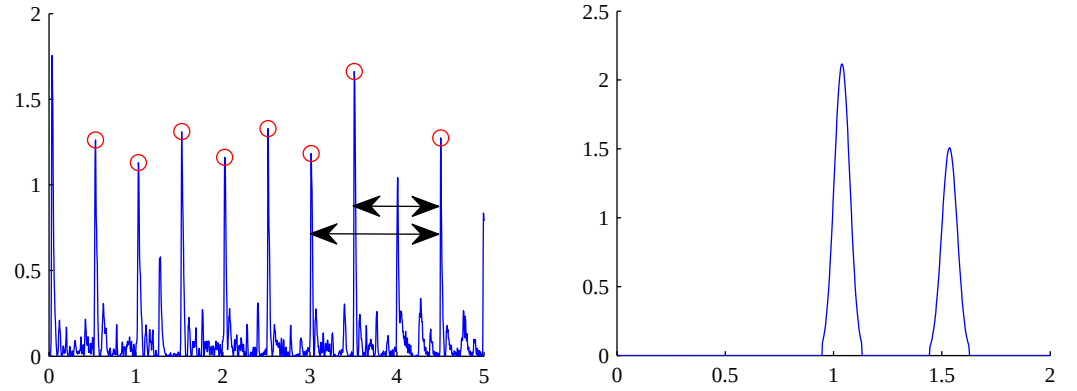
A signal is received from the onset detector when a new onset has been detected. Two IOI values are calculated; the time difference between the latest onset and the two most recent of the earlier. Figure 3.13(a) shows the two intervals annotated with double arrows. Detections are marked with circles, and as you can see there is a false negative before the last onset in this particular example. The intervals related to the undetected onset can of course not be analyzed by the system.

Both IOI values are inserted into the histogram, meaning that Gaussian pdfs are added with their center points at times corresponding to the IOIs. The weight of the Gaussian pdfs are decided by the product of the weights of the onsets that were used in resolving the IOI. Figure 3.13(b) shows this. You can see that the shorter interval adds more to the vector because it was constructed from stronger onsets.

Decay

Upon each update, the entire vector is scaled depending on how long time has passed since the previous update, so that the half-life of insertions can be controlled. The scaling factor is exponential and is applied to gradually reduce the influence of older input over time.

The vector could for instance be multiplied by a constant c^{dt} at regular dt intervals. It does however make more sense to perform the decay whenever the vector is read, that is



(a) Selecting the intervals from the onset queue

(b) Their addition to the beat probability vector

Figure 3.13: IOI selection and their addition to the beat probability vector

at onset detections. So c^{dt} is calculated, with dt being the time elapsed since the previous onset detection.

In the implementation, the exponential decay constant c_d is approximated with a second order polynomial. In order to maintain good accuracy in this approximation, the maximum time between vector updates is limited to 2 s. The interval in samples is converted to time in seconds and mapped to [-1,1[which is the range of the DSP fixed point registers.

The selected half-life t_{hl} is 5 seconds, giving $c = 0.87055$. The polynomial coefficients are found with Matlab's `polyfit` which minimizes the squared error over the evaluation points.

The factor c^t is calculated as

$$0.87055^{(t-1)} \approx 0.0049717(t-1)^2 - 0.1124766(t-1) + 0.8699401 \tag{3.10}$$

where 1 is subtracted from t to make it fit in the DSP register. The maximum error of the approximation for $0 < t < 2$ is 0.000158.

3.3.2 Meter by autocorrelation

The meter analysis is restricted to finding the number of beats to the bar. The meter is found using the autocorrelation function, according to section 2.3.

To determine the meter, the acf $a[l]$ should be analyzed at lags corresponding to multiples of the beat period. First, a subsampled acf $a_m[i]$ is constructed, where i is the i:th multiple of the beat period τ:

$$a_m[i] = a[l_i], \quad l_i = i\tau \tag{3.11}$$

The problem of calculating a_m is that the observed beat period $\hat{\tau}$, even when correctly estimated, is an integer and must be assumed as correct only within a ± 0.5 interval.

$$\hat{\tau} = \tau + \tau_e, \quad |\tau_e| < 0.5 \tag{3.12}$$

When $\hat{\tau}$ is multiplied with i, the error τ_e is also multiplied, so the acf cannot simply be sampled at integer multiples of $\hat{\tau}$, because for some i the error will be greater than 1, and then then the wrong value is sampled.

This problem can be resolved by calculating the acf index iteratively, under the assumption that the acf $a[l]$ has local maxima at lags corresponding to multiples of τ.

Assume is that if l_i is known within ± 0.5, then l_{i+1} can be found by adding $\hat{\tau}$, so that l_{i+1} has a total error within ± 1. This means that l_{i+1} can only take one of the three values $\{l_i + \hat{\tau} - 1,\ l_i + \hat{\tau},\ l_i + \hat{\tau} + 1\}$. The correct one is that for which $a[l]$ is the maximum (because the multiple of τ is a local maximum).

The initial index l_1 can be found within ± 0.5 by setting it equal to $\hat{\tau}$. Thus, l_i where $i > 1$ can be found iteratively and the acf can be correctly subsampled at integer multiples of the true beat period τ.

A *meter probability vector* is constructed for different bar length candidates n:

$$P_m[n] = \frac{1}{\lfloor N/n \rfloor} \sum_{i=1}^{\lfloor N/n \rfloor} a_m[ni], \quad n = 2 \ldots 6 \tag{3.13}$$

where N is the length of a_m. The bar length is then selected as the maximum in this vector.

The subdivision of the beat is found by looking for local maxima in the acf at lags corresponding to fractions of the beat period. There is no need to calculate the fraction indices iteratively, as the error in $\hat{\tau}$ is made smaller with division.

The beat subdivision is decided by calculating the average of the acf sampled at indices corresponding to fractions of the beat period.

$$P_s[n] = \frac{1}{n-1} \sum_{i=1}^{n-1} a[\hat{\tau}/n], \quad n = 2 \ldots 4. \tag{3.14}$$

If $P_s[3]$ is greater than $P_s[2]$ and $P_s[4]$, the meter is concluded to be compound.

Sparse autocorrelation

Autocorrelation can not be performed directly on the onset detection function, simply because the detection function has been discarded at this stage. Instead the acf has to be calculated directly from the onset queue, which is a set of time and weight values for the most recent onsets.

First of all, consider an approximation of the original detection function constructed by placing the contents of the onset queue as impulses in an otherwise zero signal. This signal holds all the necessary information, but it is not appropriate to calculate its acf because onsets that are slightly misaligned will not count properly. The acf will appear jittery and poorly represent the periodicity of the signal. So in order to improve the approximation the signal is convolved with a Gaussian pdf representing the uncertainty in the onset locations. Now also onsets which are just slightly misaligned will have a reasonable impact on the correlation sum. The reconstructed detection function $\hat{d}[n]$ is calculated as

$$\hat{d}[t] = G * \sum_i A_i \delta[t_0 - t_i] \tag{3.15}$$

where A_i and t_i are the amplitude and time of onset i, and G is the Gaussian pdf. Figure 3.14 shows a detection function reconstruction from eight onsets of song no. 15 (see table 4.1).

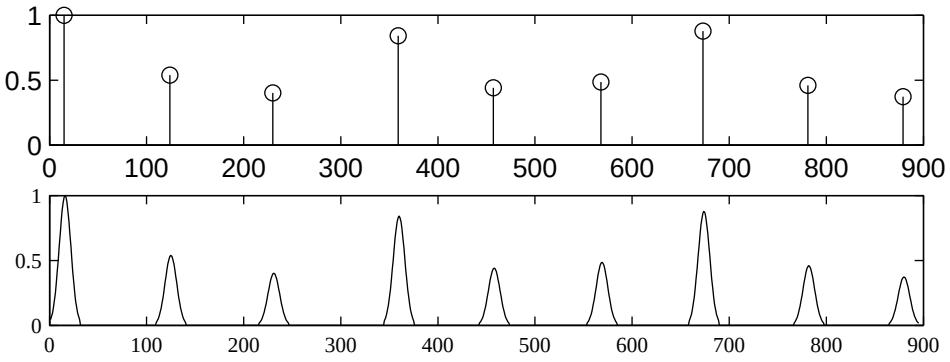


Figure 3.14: Reconstruction of detection function (lower) from onset queue (upper)

Figure 3.15 shows autocorrelation for some lags of the reconstructed detection function. This figure will be used to support the explanation of how an approximated autocorrelation can be calculated from the onset queue directly. The figure is structured so that the left plots show the reconstructed (thin line) and the reconstructed delayed (thick line) functions. Ranges where the functions overlap are marked with gray background, and the area where they overlap are marked in black.

Notice how the overlapping areas are always the result of two, not more, onsets overlapping. This is guaranteed, since overlap occurs when

$$|t_a - t_b| \leq \frac{N_G}{2} \tag{3.16}$$

where t_a and t_b are the pulses' locations and N_G is the length of the Gaussian pdf, and the minimum distance between two pulses, controlled by equation (3.9) is set to exceed half N_G. Because of the thresholding scheme a certain distance is guaranteed between the onsets. The Gaussian pdf is not as wide as this distance, therefore there is no overlapping of the onset pulses in the reconstructed detection function.

Only the areas where the functions overlap contribute to the correlation sum, as the rest is just zero, and the contribution of each overlapping area is the correlation between the overlapping onsets. This means that the autocorrelation of this signal is the sum of its pairwise inter onset correlations.

The correlation between two onsets i and j,

$$R_{i,j}[l] = \sum_t \left(A_i G[t - t_i] A_j G[t - t_j + l]\right), \tag{3.17}$$

is the correlation between two shapes which are identical except for a scaling factor. Correlation is a linear operation so the pulse amplitudes can be applied outside the sum, as

$$R_{i,j}[l] = A_i A_j \sum_t \left(G[t - t_i] G[t - t_j + l]\right) \tag{3.18}$$

and then remains only the autocorrelation of the Gaussian pdf scaled with the product of the pulses' amplitudes. Autocorrelation is the same for positive and negative lags, so it is a function of the distance between the pulses without regarding the direction. This function is written as

$$R_G[\Delta t] = (G * G)[\ |\Delta t|\ ] \tag{3.19}$$

where correlation has been exchanged for convolution since they are equivalent in this case (because of symmetry). The inter onset correlation between two onset i and j are thus given by the equation

$$R_{i,j}[l] = A_i A_j R_G[t_i - t_j + l] \tag{3.20}$$

where l is the lag.

Now, since the total autocorrelation is just the sum of the inter onset contributions it can be described accordingly:

$$R[l] = \sum_i \sum_j A_i A_j R_G[t_i - t_j + l]. \tag{3.21}$$

Consequently, it is possible to calculate the autocorrelation of the reconstructed detection function without actually reconstructing it. The function $R_G[\Delta t]$ is implemented using a lookup table. This algorithm can also be used to calculate sparse crosscorrelation.

The computational complexity of using this method to calculate the autocorrelation of a length L acf from N onsets is then $\mathcal{O}(N^2 L)$.

This method compares favorably to calculating the autocorrelation sum in a straightforward way, which has $\mathcal{O}(L^2)$ complexity, as L is typically larger than N^2.

Autocorrelation can also be calculated more efficiently through FFT, see equation (2.27), because correlation is time reversed convolution, and time reversal is equivalent to conjugation in the frequency domain. This operation has an $\mathcal{O}(L \log L)$ complexity. The signal must however be zero-padded to twice its length since it is not periodic, and it must also be adjusted to a power of two because of radix-2 FFT implementation.

The major drawback of using FFT is however that you need to calculate all the lags, i.e. a large amount of correlation lags are calculated but never used. Also it requires the full data range to be stored in memory. Both the regular autocorrelation and the sparse ditto gives the possibility to calculate the correlation of arbitrary lags, i.e. the acf can be sampled at any lag, without calculating all the other lags too.

Sparse autocorrelation is thus a fast, memoryless approximation to regular autocorrelation.

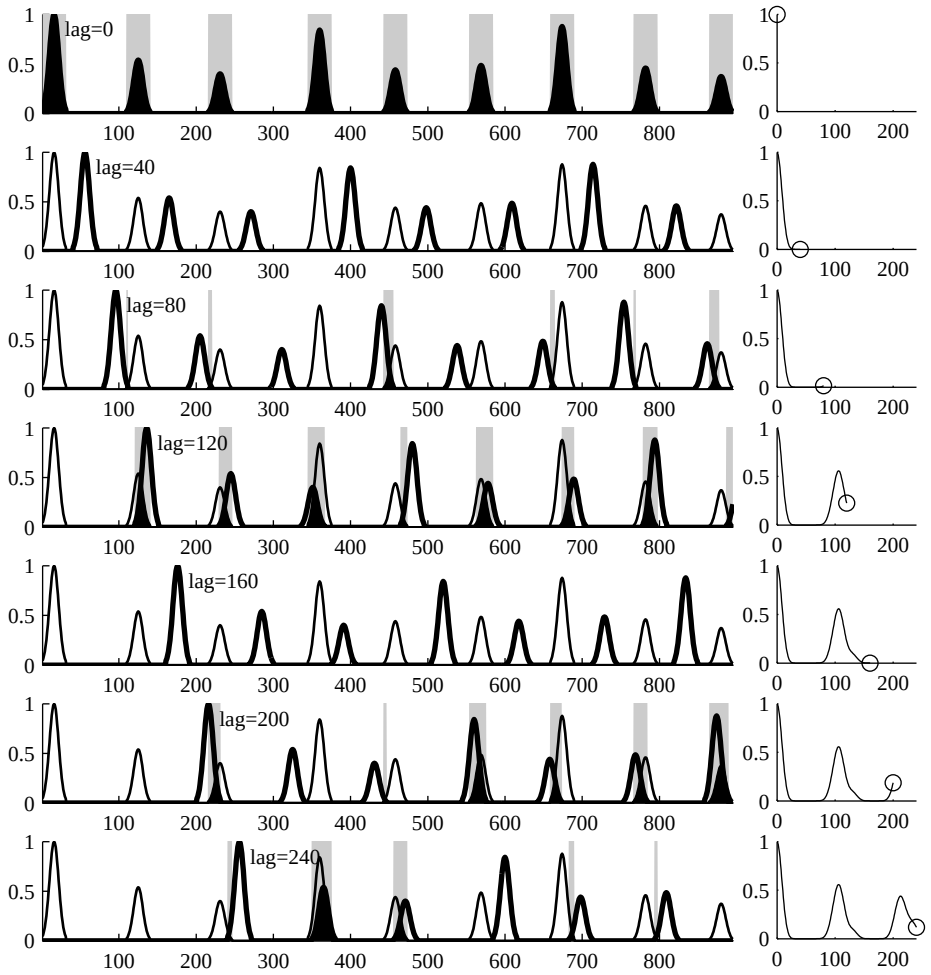


Figure 3.15: Autocorrelation of reconstructed detection function

3.4 Synchronization

The clock module should deliver MIDI clock messages at the pace of the input music.

The main purpose of the synchronization module is to phase-align the output clock to the input music. It should prevent the clock-drifting problem introduced from the inaccuracy of the beat period estimation, but also allow the system to follow minor tempo deviations in the music.

The operation is loosely designed around the PLL concept. The phase difference between the output clock and the input beats is measured and fed back through a low-pass filter. The clock uses the filter output to adjust its tempo so that it stays in phase with the music.

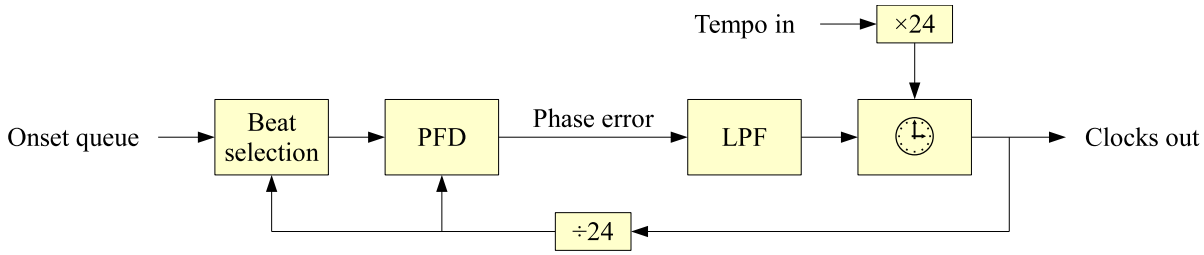


Figure 3.16: Synchronization overview

3.4.1 Beat selection

The onset queue does not only contain beat onsets, but also any other note onset that is detected in the audio stream. This is a problem when measuring the phase difference, as only beats/quarter notes should be considered. If the phase difference was measured for all onsets, there would be a lot of error values representing for instance the difference between the output clock and non-beat eighth and sixteenth notes.

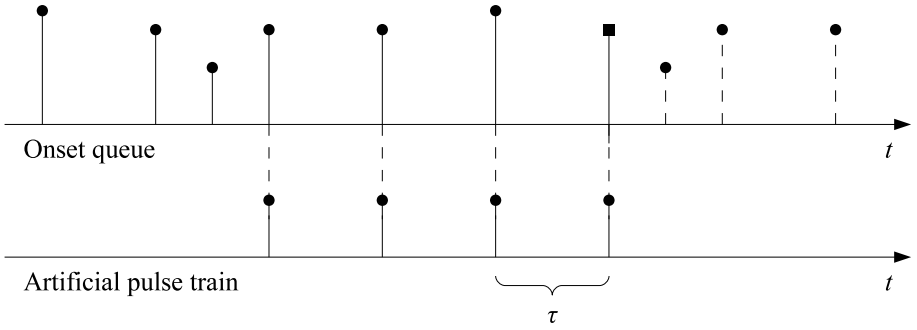
Beat selection is the process of excluding non-beat onsets from the queue, so that only beat onsets are fed to the PFD. This is an approximative process, as there is no way of knowing with 100% certainty whether an onset is a beat or not.

Davies and Plumbley (2004) suggested the use of crosscorrelation to determine the beat phase at any point in time. The upsampled onset detection function would be correlated against a comb filter representing the beat period. The maximum of the crosscorrelation would correspond to the time difference between now and the closest beat.

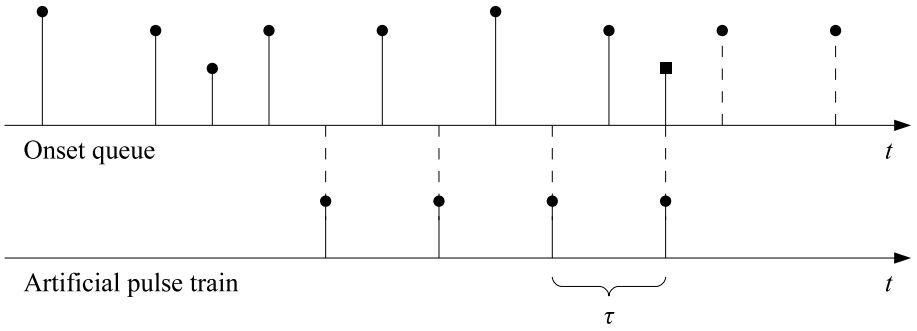
A similar approach is employed here, but the crosscorrelation is evaluated at every onset in the detection queue, and only lag 0 is calculated. This means that for every onset we get a measure of how well it correlates to an ideal detection function with the same alignment. This is calculated with *sparse* crosscorrelation, which is identical to the sparse autocorrelation described in section 3.3.2 only two different functions are used instead of delaying a copy.

An artificial pulse train is constructed, with 4 pulses at intervals corresponding to the current clock period. Then the crosscorrelation at lag 0 between this pulse train and the onset queue is calculated. If the current onset is a beat, then the correlation is strong, and if it is not a beat the correlation is weak.

If a beat has occurred, the situation will be something like figure 3.17(a). The subject (onset) of evaluation is marked with a square, while other onsets are marked with circles. As you can see the onset queue aligns well with the artificial pulse train with this onset as reference. This onset will be interpreted as beat. Figure 3.17(b) shows the case of a non-beat onset. Here the artificial pulse train aligns poorly with the onset queue resulting in a correlation sum which is significantly smaller than that of the beat.



(a) beat



(b) non-beat

Figure 3.17: Crosscorrelation of onset queue and artificial pulse train

The beat/non-beat decision is taken on the following criterion: if the correlation is stronger than 75% of the average of the last 4 correlation, then the onset is considered a beat. If this is not the case, it will be considered a non-beat.

3.4.2 Phase error and filtering

When an onset is accepted as a beat, the phase difference $\hat{\phi}_e$ is measured as the distance between the onset and the last output beat location. Figure 3.18 shows the procedure. As you can see, in this particular example the beat prediction (dashed line) is late compared to the detected beat. The phase difference $\hat{\phi}_e$ to the last beat output is almost as large as the beat period τ. This means that there is actually a small negative difference ϕ_e which is found by subtracting τ from $\hat{\phi}_e$ according to equation (3.22).

$$\phi_e = \begin{cases} \hat{\phi}_e & \text{if } \hat{\phi}_e \leq 0.5\tau \\ \hat{\phi}_e - \tau & \text{if } \hat{\phi}_e > 0.5\tau \end{cases} \tag{3.22}$$

The phase error is fed through a 3 tap FIR averaging low-pass filter. This filter has a normalized cut-off frequency of 0.25 (where 1.0 corresponds to π).

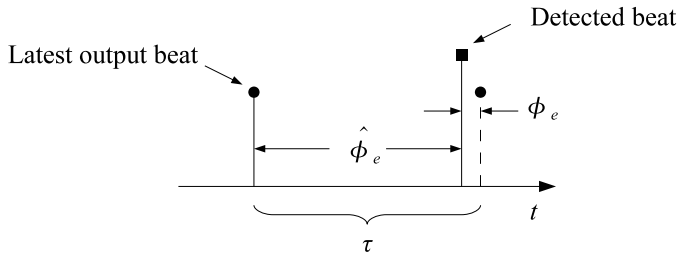


Figure 3.18: Measuring the phase difference

3.4.3 MIDI

MIDI messages are sent with the serial communication interface (SCI) of the DSP56303. The SCI, described in (Freescale Semiconductor, 2005b, ch. 8), is configured to asynchronous mode with 1 start bit, 8 data bits, and 1 stop bit (in accordance with the MIDI specification). The serial clock is divided from the 12.288 MHz system clock by 39 to give a speed of 31508 baud, which is within the 31250±1% in the MIDI specification.

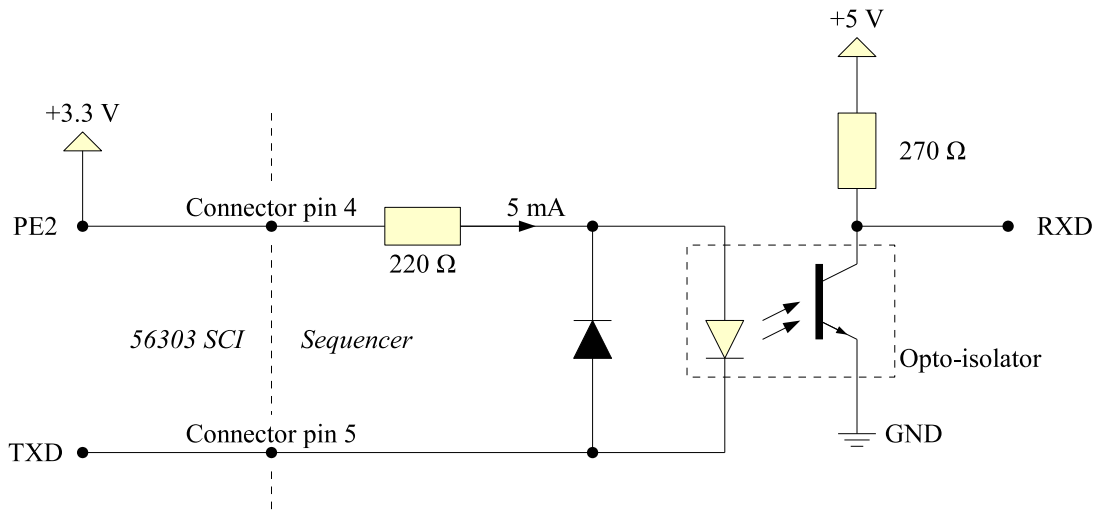


Figure 3.19: MIDI Hardware

Figure 3.19 shows the receiving side in MIDI communication. Bits are transmitted by regulating the voltage of the TXD pin. When its voltage is set to zero, current flows from PE2 through the LED in the opto-isolator, thereby opening the opto-transistor and pulling RXD low. When the voltage of TXD is set to high (3.3 V), there is no current and the opto-transistor is closed so that RXD is pulled high by the 5 V voltage source. There is always an opto-isolator at the MIDI input side to ensure that instruments are electrically isolated from each other.

The MIDI connector has simply been connected to the SCI without any additional logic. This is possible because the current specification of 5 mA is not critical. The current will probably be slightly above that in this system, but that depends on the voltage drop over the LED

The current on the transmitter side is drawn from the PE2 pin, which is a data output set to constant high. It has been chosen because it is physically adjacent to the TXD pin (it

is in fact the serial clock pin SCLK, reconfigured to general purpose I/O in software). The location of pins SCLK and TXD on the PCB can be found in (Freescale Semiconductor, 1999, p. A-5).

Chapter 4

RESULTS

4.1 Evaluation

The individual components of the system are evaluated in their Matlab implementation form. The decision to do so was made for practical reasons. Using Matlab, data can easily be collected and analyzed. To achieve the same amount of flexibility with the DSP would require a large amount of work.

This means that the functionality of the actual DSP implementation is not verified. But the Matlab evaluation verifies the concepts that make the system.

4.1.1 MIREX Training Data

This data set, which was used for training onset detectors in the past Music Information Retrieval Evaluation eXchange (MIREX) competitions, will be used here for evaluation. It is a publicly available[1] collection of 20 musical excerpts. Each excerpt is 30 s long and comes from different songs covering a variety of genres. The songs are listed in table 4.1. Some of them have unfortunately not been possible to identify, and when it comes to the classical pieces (songs 7, 10, and 12) it is not guaranteed that the correct performer is stated in the table.

The MIREX data contain not only the actual audio, but also manually annotated onset traces. These have been carried out for each song, by 40 different listeners. Each onset trace is stored as a collection of onset locations. Onset weight information is not available.

Each song has also been labeled with two musical tempi. Both are considered correct; their difference lies in the listener's subjective impression of the beat. There is usually a factor of 2 between the suggested tempi, or 3 for triple meter songs.

This information is very valuable, but it has some problems that reduces the quality of the evaluation in this report. First of all, the onset annotations tend to be of a foot-tapping character. That is, mostly beat onsets are annotated, not all the minor onsets in-between. The system described in this report was designed to deal with a situation where all beats are annotated. Secondly, the spread of the onsets is quite large, i.e. different persons have put the same onsets on slightly different locations. This means there is no "ground truth" available.

[1] `http://www.music-ir.org/mirex/2006/index.php/Audio_Beat_Tracking`

No.	Artist	Song	BPM	Meter
1	Barry White	My Everything	129.5	duple
2	Billy Bragg	New England	83.5	duple
3	[unknown]	[unknown]	167.5	duple
4	Erykah Badu	Mama's gun	126.0	triple
5	Passe and Medio - Den iersten gaillarde	Capilla Flamenca	68.5	triple
6	China Dolls	Wo Ai Ni	82.0	duple
7	Zubin Mehta	Stravinsky : Le sacre du printemps (Rite of Spring) : IV Spring Rounds	56.5	duple
8	Aphex Twin	Flim	148.0	duple
9	Bob Dylan	Hurricane	129.0	duple
10	Nikolaus Harnoncourt	Domine ad adiuvandum me festina - Vespro della Beata Vergine	122.5	duple
11	Zillertaler Schürzenjäger	Komm nach Tirol	140.0	duple
12	Nikolaus Harnoncourt	Bach, JS : St Matthew Passion BWV2 : Part 1 "Kommt, ihr Tochten helft mir klagen" [Chorus]	54.0	duple
13	Goran Bregowic	Ederlezi	180.0	duple
14	Tatu	200 Km/h in the wrong lane	130.0	duple
15	Le Bruit du Frigo	Mano Negra	186.0	triple
16	[unknown]	[unknown]	90.5	duple
17	Tom Waits	The Piano Has Been Drinking	45.5	duple
18	Radiohead	Exit Music (For a Film)	121.5	duple
19	Suicidal Tendencies	Possessed To Skate	93.5	duple
20	Ska-P	Gato Lopez	220.5	duple

Table 4.1: The MIREX training data

4.1.2 Onset Detection

This section provides an evaluation of the onset detection algorithm. The convention to count successful vs erroneous detections has not been followed here. The decision to do so was made because it is difficult to classify detections as false, when the annotated onsets are so spread out around the actual locations.

Instead, the onset detection results are compared to a probabilistic interpretation of the annotated onsets; a function of time reflecting the probability of an onset occurrence.

The onset probability function is constructed by joining the annotation traces of all listeners for each piece. Then Kronecker deltas are placed in an otherwise zero function, at locations reflecting the onsets. Finally this function, now a pulse train, is smoothed with a Gaussian kernel to introduce some natural spread.

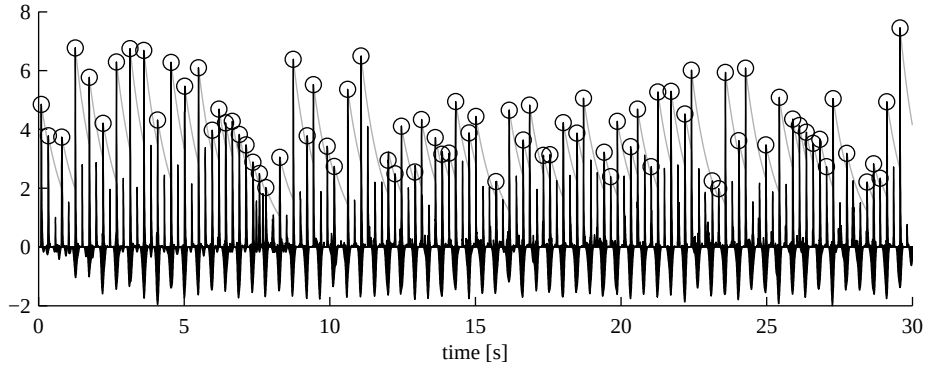
In the plots presented in this section, the onset probability function is plotted in the same graph as the onset detection function. The former is on the negative y-axis, while the latter is on the positive. The detected onsets are marked with circles and the adaptive threshold with a light gray line.

An investigation of how accurate in time the detections are has also been performed. The distances between detected onsets and annotated has been measured and placed in an histogram of the range ± 20 ms.

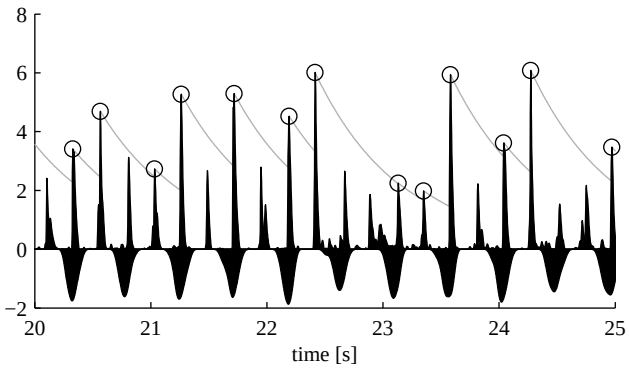
The result of onset detection varies a lot for the different songs. For a few of them, the system fails miserably. This includes music played with bowed string instruments. Onsets from such instruments are difficult to capture with a purely energy based onset detector. This is because there are no attack transients between notes when they are played continuously.

There is also the unfortunate mix of occasional hard onsets in a song mostly defined by soft onsets, such as song 7. Here there are a few drum strikes which overshadow the softer bowed string onsets in between, and the drum strikes are so far apart that the adaptive threshold reaches the noise floor before the next drum. The results from onset detection on this piece is simply disastrous.

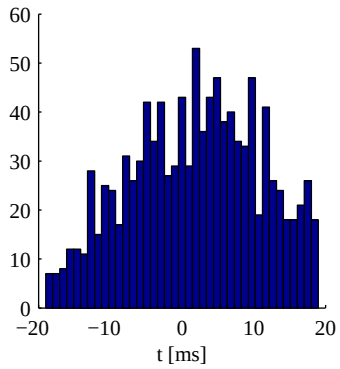
Three of the songs in the test data have been selected for display in this section. The first is the the well-known “My everything” by Barry White (song no. 1 in table 4.1), a pop/rock piece containing drums, guitar, bass, strings, and vocals. Figure 4.1 shows the onset detection results for this song. The upper plot (a) shows the full 30 s signal. If you’re reading the digital version of this paper you should be able to zoom in on that plot for greater detail, but if you’re reading a printed version you’ll have to make do with the detailed excerpt in plot (b). You’ll notice that the onset detection function seems to lag a bit behind the reference (onset probability function). The histogram of detection differences in plot (c) confirms this. The onset detector appears to be some 5-10 ms late. You’ll also see that the non-beat onsets goes undetected most of the time. This is mainly related to the thresholding method. A mean or median threshold would most likely capture also the non-beats.



(a) Full



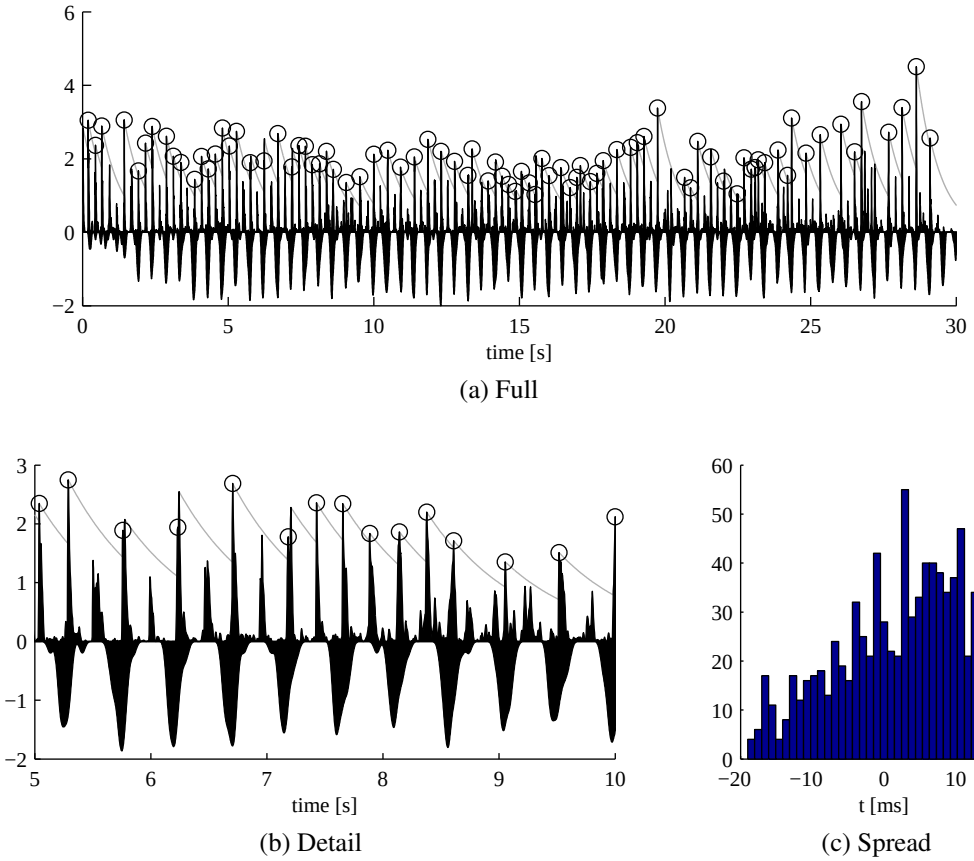
(b) Detail



(c) Spread

Figure 4.1: Onset detection for song no. 1

Song no. 9 (figure 4.2) is the first 30 s of another famous song: “Hurricane” by Bob Dylan. It contains drums, guitar, bass, vocals and harmonica. You’ll see the same lag behavior as in song 1 (figure 4.1), and also a similar tendency of leaving non-beat onsets undetected. The lag appears to be a systematic error in the design.



(a) Full

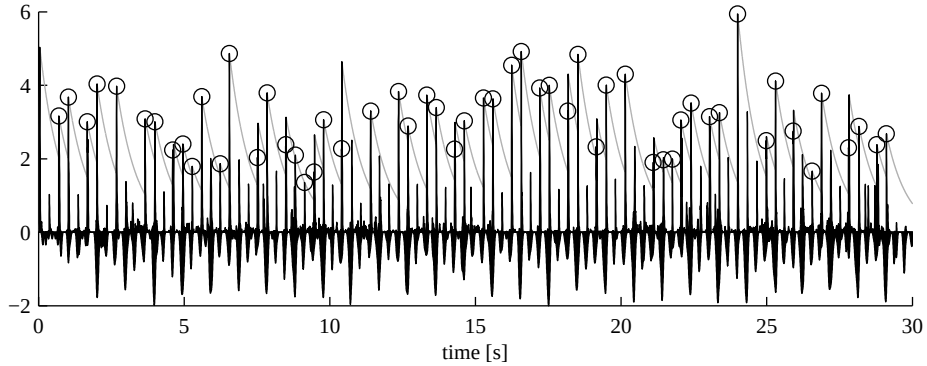
(b) Detail

(c) Spread

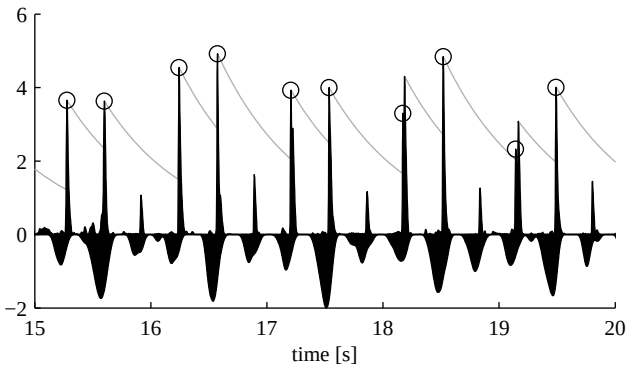
Figure 4.2: Onset detection for song no. 9

Song no. 15 (figure 4.3) is a triple meter piece with strongly accentuated down-beat. The down-beat is a bass note with long duration, the presence of which effectively attenuates the following onset. The onset after that (the third in the series) sounds exactly the same but appears much stronger in the detection function. This problematic behavior is visible as the repeating sequence of strong-weak-strong in the detection function. This is a system design flaw that might have negative consequences in the tempo analysis stage. The lag problem seen in the two previous histograms is visible here also.

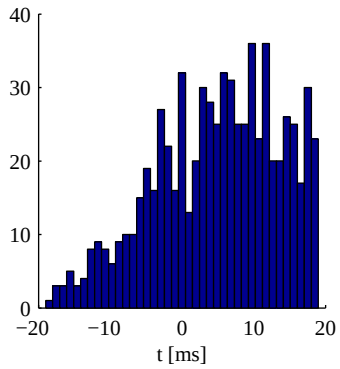
Notice that the overall strength of the onsets in song 15 is about half that of song 1. This confirms the need for adaptive thresholding.



(a) Full



(b) Detail



(c) Spread

Figure 4.3: Onset detection for song no. 15

4.1.3 Tempo estimation

The tempo estimation is evaluated with manually annotated onsets as input. This is because the accuracy of tempo estimation depends on the quality of the input, so in order to make a fair evaluation of the tempo estimation algorithm there must be no reliance on the onset detection algorithm.

At the same time, the input should not be perfect because the ability to correctly estimate tempo with onset detection errors present is a desired feature of the system, a design goal.

The manually annotated onsets from the MIREX training data are chosen as input. Then a controlled amount of error has been introduced. Some amount of the onsets are removed to simulate false negative detections, and some amount of random onsets are inserted to simulate false positive detections.

Table 4.2 shows the two BPM values that are regarded as correct for each song. The four rightmost columns show for how much of the song the tempo detector outputs a value within three samples from any of the two correct tempi. The "OK" column is for the error free case, while "fn" and "fp" denotes the presence of false negative and positive detections.

Song	BPM_1	BPM_2	OK	fn	fp	fn,fp
1	129.5	64.5	100%	92%	93%	75%
2	167.5	83.5	95%	92%	83%	69%
3	153.0	76.5	100%	97%	99%	84%
4	126.0	42.0	79%	56%	75%	30%
5	205.5	68.5	94%	63%	76%	49%
6	82.0	41.0	41%	20%	16%	3%
7	113.5	56.5	14%	14%	11%	5%
8	148.0	74.0	100%	100%	99%	87%
9	129.0	64.5	72%	77%	97%	92%
10	122.5	61.0	95%	86%	91%	64%
11	140.0	70.0	100%	92%	86%	79%
12	54.0	27.0	8%	16%	13%	13%
13	180.0	90.0	100%	94%	80%	65%
14	130.0	65.0	94%	94%	85%	63%
15	186.0	62.0	62%	70%	93%	84%
16	90.5	45.0	37%	28%	39%	20%
17	91.5	45.5	46%	42%	41%	26%
18	121.5	61.0	64%	44%	38%	38%
19	188.0	93.5	98%	97%	98%	93%
20	220.5	115.5	98%	91%	95%	90%

Table 4.2: Tempo detection results

A closer look at the result of tempo detection in song 8 helps in understanding the effect of onset detection errors. The plots show how the beat probability vector evolves over time for different error injections. The upper left plot is based on the correct onset locations and it shows stable tempo output. For the upper right plot 25% of the onsets

have been removed. This weakens the correct estimation, and creates ridges of tempo fractions (at integer fractions of the true tempo). For the lower left plot no onsets have been removed, but 25% erroneous onsets have been inserted at random locations. These false positives create tempo peaks at tempi above the correct. Notice that how false positives and false negatives show up in distinctively different ranges of the vector. Finally the lower right plot has 25% of the true onsets removed, as well as 25% random incorrect onsets inserted. Here the errors are spread all over the vector.

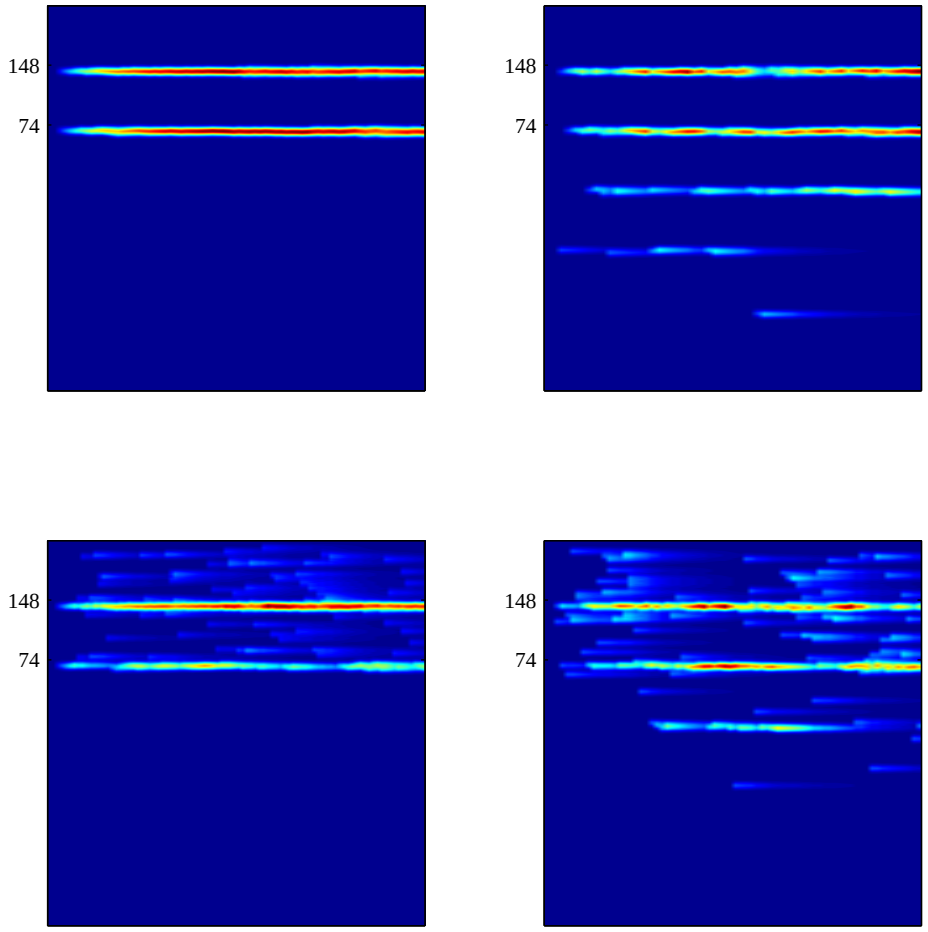
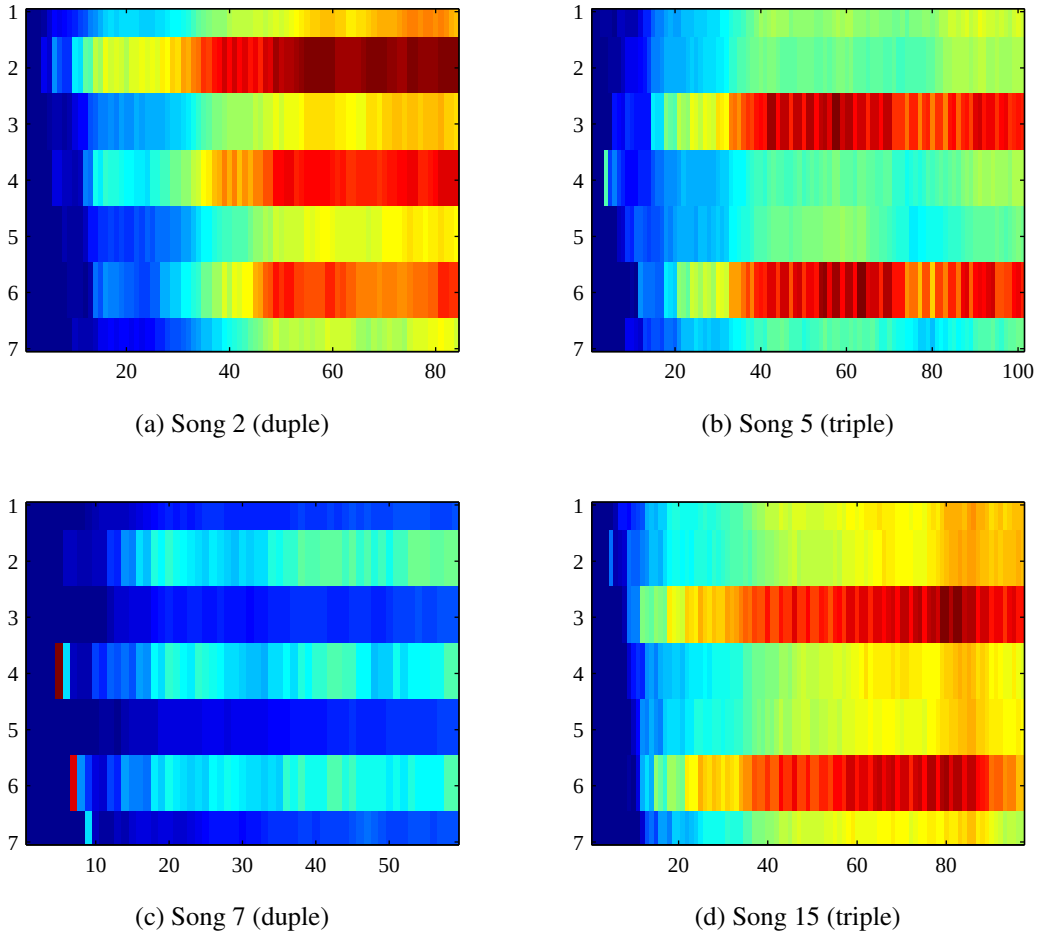


Figure 4.4: Beat probability vector over time for song 8: without errors (top left), with false negatives (top right), with false positives (bottom left), and with both false positives and negatives (bottom right)

4.1.4 Meter Estimation

The meter estimation algorithm does not perform well on the MIREX training data. This is probably because most of the songs are duple meter, and there is no difference in accentuation between beats of different weights, which is a prerequisite for successful meter identification. In the triple meter songs, however, several listeners appear to have annotated only the down-beat, causing it to be accentuated compared to the other two beats of the measure.

Figure 4.5 show the meter probability vectors of four different songs. On the y-axis is beats per bar, and the x-axis is time (non-linear, onset number). Notice how the triple meter songs (b) and (d) have strong indications on 3 and 6, while the duple meter songs (a) and (c) have strong indications on 2, 4. and 6.



(a) Song 2 (duple)

(b) Song 5 (triple)

(c) Song 7 (duple)

(d) Song 15 (triple)

Figure 4.5: Meter probability vector of different songs

4.1.5 Synchronization

The synchronization is only implemented on-line on the DSP platform. That means that the results cannot be easily compared to reference data. Consequently, this report does not

contain an objective measure of the synchronization performance. Instead the subjective impression of the algorithm performance is described.

To investigate the operation of the system when it is not exposed to any input errors, it is subjected to a click-track. For this kind of input, the onset detector has no problems finding then onset locations.

The system appears to have no problems aligning its output clock to this kind of input. When the system has found the correct tempo it manages to align to the input within two or three beats.

When subjected to spurious step changes in phase, simulated by increasing the time between only two of the clicks in the track, the system adapts and readjusts its own phase within a few beats (if the tempo has been estimated correctly).

The system has also gained the ability to follow a tempo which is not representable with its internal temporal resolution. A small error is continuously fed back as the beat predictions fail, so that it adds time to the next prediction thereby adjusting to the actual tempo of the input.

The system does however appear to have some problem to align properly. If the input tempo is faster than what the system has expected, it tends to always lag a little behind, as if it was "chasing" the input.

When the system has estimated the input tempo as higher than it actually is, it tends to make early predictions, then adjusting them but never exactly aligning to the input, even if it keeps the same pace.

When exposed to a more complex pattern of input signals, the beat selection algorithm tends to be successful most of the time. It does, however, inevitably fail at some point, sooner or later, for any complex input. Single failures are suppressed by the loop filter, but when there are several, the synchronization goes out of phase.

Chapter 5

CONCLUSIONS

5.1 Discussion

Strictly speaking, the system presented in this report does not meet the initial goals and expectations. It does however cover a large part of the way there. Most of the remaining work lies in improving the synchronization module, making it more stable. The future work section contains several suggestions on how to continue the work.

About the Results

The results accounted for in the previous section are not exactly jaw-dropping. On the other hand, those test data were carefully selected to test onset detectors for weaknesses. The number of songs is also very small, so the statistical significance of the results are questionable at least. The impression of the author is that the system performs better for most common pop music, than it did for the songs in the test set.

Nevertheless, it is interesting to speculate in what makes the algorithms presented in this report fail in so many situations.

One of the main problems in the designing of this system has been the presence of input errors at all levels of analysis. These errors are essentially introduced in the onset detector, and then propagated through the following processing stages.

The reason why the onset detector does produce errors is that the sound envelope model is not an accurate description. The detected envelope does have rapid increases also in locations not related to onsets, and there are onsets which does not give rise to rapid attacks. Depending on the type of input, the amount of error is different.

There is also the ever present possibility that even onsets which have been correctly detected, are not accurate carriers of timing information. Only a machine can hold a tempo perfectly. Musicians play the notes slightly misaligned to the intention.

Fundamental Problems

The inadequacy of the sound envelope model is really just a symptom of a fundamental, underlying problem: the wide range of possible input to the system. The models employed in this report are simply not enough to describe the vast concept of music. Sounds and rhythms can take any shape, and they do!

Another fundamental obstacle in the designing of this system is the everlasting contradiction between *responsiveness* and *stability* of a system. This problem has demanded a substantial amount of compromise in design and sacrifice in performance. Errors are present at stages of analysis in the system. The system has methods for dealing with the errors, based on probabilistic approaches. This resistance towards errors does however make the system stiff, less responsive. It cannot act immediately on a tempo change, because it must assume that such an unexpected event is more likely to be due to some false onset detection, rather than an actual tempo change. Only when the new tempo is reinforced by reoccurring beats, can it be acted upon.

This problem is also present in the synchronization algorithm, which is supposed to react fast, while still being resilient towards error. The beat selector used to remove unwanted input is a difficult design. The method proposed in this thesis fails quickly in passages where there is no strong beat available.

From a philosophical point of view, an attempt at flawless synchronization is an attempt to predict the future. The system can only make a qualified guess of when the next beat is going to occur, based on previous input. The system can never be so good, that it is not possible to find an input for which it fails.

5.2 Future Work

General

Something that would be beneficial for the system, at all stages of analysis, is to find some way of quantifying the quality of the information available. One example of when this could be useful, is when a song enters some passage where the onset detector starts failing. It could be for instance that the drums stop playing. If the detector starts finding false positives, it will sabotage the tempo/meter analysis. It would be of great value if the system had some way of realizing that right now, it is producing nonsense, and it would be better to just stick with what it knew from before. The quality of the information has not been improved, but the awareness of the quality has been introduced. Such a feature would greatly improve the usefulness of the system.

Onset detection

When it comes to envelope detection, a more thorough investigation of frequency band selection would be desirable.

The following suggestions are mainly related to lowering the computational load of the envelope detector.

- The signals subjected to envelope detection are narrow-band. It should be examined whether downsampling could be applied directly to the band-pass filtered signals. The idea being that the downsampling operation will fold the energy down to a lower band.

- The envelope detector presented uses IIR filtering followed by downsampling. This could be replaced with a polyphase FIR filter, performing both filtering and downsampling. The advantage of this should be the possibility to use a tailored impulse-response, like that the psychoacoustically motivated integrator in (Scheirer, 1998).

Tempo/meter analysis

The beat probability vector is practical, but it has a few problems. The first is of course that it's resolution is limited to that of the detection function. Considering that many IOIs are collected, it should be possible to introduce some refining, averaging scheme to arrive at a sub-sample beat period estimate.

One idea is to replace the probability vector with an adaptive bucket sort algorithm, where the buckets correspond to the peaks of the beat probability vector. The bucket centers could then be gradually refined with time.

When it comes to meter estimation, the most urgent thing is to determine the location of the down-beat. As long as this information is unknown to the system, it will not able to automatically align sequencer patterns to bars in the input music. From what other authors have written about this particular topic, it is very difficult.

Synchronization

The beat selector is the weakest link of the synchronization chain. It's poor performance makes synchronization impossible for anything but very clear onset patterns. It could possibly be complemented with some kind of expectancy function to improve the results. This is one of the most problematic issues of this system and it is one of the first things that should be looked in to in the future.

The effect of the synchronization loop filter has been investigated only very briefly when designing this system. The use of a median filter, or filters that weigh the input based on its deviation could prove beneficial in this critical part of the synchronization loop.

Appendix A

SOURCE LISTINGS

A.1 DSP

A.1.1 Biquad Sections

```
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;
        ; direct form I
        ;
        ; coefficients in y-mem: .5*( a2, a1, b2, b1, b0 )
        ; state vars in x-mem:    y[-2], y[-1], x[-2], x[-1]
        ;
        ;   5 cycles per sample
        ; ~20 cycles overhead
        ;
        ; last output in b

biquad_df1      macro   length,work,coeffs,state

        move    #coeffs+2,r4
        move    #2,n4
        move    #5-1,m4
        move    #state,r3
        move    #4-1,m3
        move    #work-2,r2
        move    #-1,m2
        move    #work,r1
        move    n4,n1     ; #2,n1
        move    m2,m1     ; #-1,m1
        move    #work-3,r0
        move    m2,m0     ; #-1,m0

        bset    #11,sr    ; scale up
        nop

        move                        x:(r3)+,x1        y:(r4)+,y0
```

```
        mpy     y0,x1,a         x:(r3)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r1),x1       y:(r4)+,y0
        mac     y0,x1,a         x:(r3)+,x1      y:(r4)+,y0
        mac     -y0,x1,a        x:(r3)+,x1      y:(r4)+,y1
        mac     -y1,x1,a        x1,x:(r0)       y:(r4)+,y0

        do      #length/2-1,_loop
        mpy     y0,x0,b         x:(r1)+,x0      y:(r4)+,y0
        mac     y0,x0,b         x:(r1)-,x0      y:(r4)+,y0
        mac     y0,x0,b         x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,b        a,x:(r2)+       a,y0
        mac     -y1,y0,b        x:(r1)+,x0      y:(r4)+,y0
        mpy     y0,x0,a         x:(r1)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r1)-,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,a        b,x:(r2)+       b,y0
        mac     -y1,y0,a        x:(r1)+,x0      y:(r4)+,y0
_loop
        move                    x0,x:(r3)+
        mpy     y0,x0,b         x:(r1)+,x1      y:(r4)+,y0
        mac     y0,x0,b         x:(r1)-,x1      y:(r4)+,y0
        mac     y0,x1,b         x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,b        a,x:(r2)+       a,y0
        mac     -y1,y0,b        x1,x:(r3)+
        move                    a,x:(r3)+
        move                    b,x:(r2)-
        asl     b               b,x:(r3)+

        nop
        nop

        bclr    #11,sr  ; no scale

        endm


        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;
        ; direct form I with first order error feedback
        ;
        ; coefficients in y-mem: a2, 1+a1, b2, b1, b0
        ; state vars in x-mem:   y[-2], y[-1], x[-2], x[-1],
        ;                        acc a0, a1, a2
        ;   6 cycles per sample
        ; ~20 cycles overhead
        ;
        ; last output in b

biquad_df1e     macro   length,work,coeffs,state
```

```
        move    #work,r0
        move    #-2,n0
        move    #-1,m0

        move    #state,r1
        move    m0,m1

        move    #coeffs,r4
        move    #5-1,m4

        move                    x:(r1)+,a0
        move                    x:(r1)+,a1
        move                    x:(r1)+,a2
        move                    x:(r1)+,x0      y:(r4)+,y0
        mac     -y0,x0,a        x:(r1)+,x1      y:(r4)+,y0
        mac     -y0,x1,a        x:(r1)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r1),x0       y:(r4)+,y0
        mac     y0,x0,a         x:(r0)-,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r1),x0       y:(r4)+,y1
        tfr     a,b             x0,x:(r0)-      y:(r4)+,y0

        do      #length-2,_loop
        mac     -y1,x1,a        a,x:(r0)+       a,y1
        mac     -y0,y1,a        x:(r0)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r0)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r0)+n0,x0    y:(r4)+,y0
        mac     y0,x0,a         b,x1            y:(r4)+,y1
        tfr     a,b                             y:(r4)+,y0
_loop
        mac     -y1,x1,a        a,x:(r0)+       a,y1
        mac     -y0,y1,a        x:(r0)+,x0      y:(r4)+,y0
        mac     y0,x0,a         x:(r0)+,x1      y:(r4)+,y0
        mac     y0,x1,a         x:(r0)+n0,x0    y:(r4)+,y0
        mac     y0,x0,a         x0,x:(r1)-
        move                    x1,x:(r1)-
        tfr     a,b             a,x:(r0)+       a,y0

        move                    y0,x:(r1)-
        move                    y1,x:(r1)-

        move                    a2,x:(r1)-
        move                    a1,x:(r1)-
        move                    a0,x:(r1)-

        endm
```

A.1.2 Resonance Filter

```
        ; direct form 1 biquad section for filters with b1=0

resf    macro   wbuf,coeffs,state

        move    #coeffs+2,r4
        move    #2,n4
        move    #4-1,m4
        move    #state,r3
        move    m4,m3       ; #4-1,r3
        move    #wbuf-2,r2
        move    #-1,m2
        move    #wbuf,r1
        move    n4,n1       ; #2,n1
        move    m2,m1       ; #-1,m1
        move    #wbuf-3,r0
        move    m2,m0       ; #-1,m0

        bset    #11,sr      ; scale up

        move                        x:(r3)+,x1      y:(r4)+,y0
        mpy     y0,x1,a             x:(r3)+,x0
        move                        x:(r1)+,x1      y:(r4)+,y0
        mac     y0,x1,a             x:(r3)+,x1      y:(r4)+,y0
        mac     -y0,x1,a            x:(r3)+,x1      y:(r4)+,y1
        mac     -y1,x1,a            x1,x:(r0)       y:(r4)+,y0

        do      #FRAMESIZE/2-1,_loop
        mpy     y0,x0,b             x:(r1)-,x0      y:(r4)+,y0
        mac     y0,x0,b             x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,b            a,x:(r2)+       a,y0
        mac     -y1,y0,b            x:(r1)+n1,x0    y:(r4)+,y0
        mpy     y0,x0,a             x:(r1)-,x0      y:(r4)+,y0
        mac     y0,x0,a             x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,a            b,x:(r2)+       b,y0
        mac     -y1,y0,a            x:(r1)+n1,x0    y:(r4)+,y0
_loop
        move                        x0,x:(r3)+
        mpy     y0,x0,b             x:(r1)-,x1      y:(r4)+,y0
        mac     y0,x1,b             x:(r0)+,x0      y:(r4)+n4,y0
        mac     -y0,x0,b            a,x:(r2)+       a,y0
        mac     -y1,y0,b            x1,x:(r3)+
        move                        a,x:(r3)+
        move                        b,x:(r2)-
        asl     b                   b,x:(r3)+

        nop
        nop

        bclr    #11,sr      ; no scale
```

```
        endm
```

A.1.3 Beat probability vector operations

```
bpv_minperiod   equ     #60*FS/DOWNSAMPLE/300       ; 300 BPM
bpv_maxperiod   equ     #60*FS/DOWNSAMPLE/50        ; 50 BPM

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bpv_clear       ; set all values to zero

        move    #bpv,r0
        move    #-1,m0
        clr     a
        rep     #512
        move    a,x:(r0)+
        rts

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bpv_decay       ; calculate decay and scale

        move    #exp_decay,r6
        move    #-1,m6

        move    #1.0*DOWNSAMPLE/FS,x0
        move    #-1.0,b             a,y0
        mpy     x0,y0,a
        asl     #23,a,a
        add     b,a                 ; a=dt-1 [seconds]

        move    #bpv,r2
        move    #-1,m2

        move    a,x0                y:(r6)+,b       ; x0=x          b=p0
        mpy     x0,x0,a             y:(r6)+,y0      ; a=x^2         y0=p1
        mac     x0,y0,b             y:(r6)+,y0      ; b+=p1*x       y0=p2
        move    a,x0                                ; x0=x^2
        mac     x0,y0,b                             ; b+=p2*x^2

        move    r2,r3
        move    m2,m3

        move                        x:(r2)+,x0      b,y0
        mpy     y0,x0,a             x:(r2)+,x1
        mpy     y0,x1,b

        do      #512/2-1,_loop
        move                        x:(r2)+,x0
```

```
        mpy     y0,x0,a         a,x:(r3)+
        move                    x:(r2)+,x1
        mpy     y0,x1,b         b,x:(r3)+
_loop   move                    a,x:(r3)+
        move                    b,x:(r3)+

        move    y0,a            ; return decay value
        rts

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bpv_findmax     ; find maximum

        move    #bpv+gauss_pdf_len/2+bpv_minperiod,r0
        move    #-1,m0

        nop
        nop
        nop

        clr     b       x:(r0)+,a
        do      #bpv_maxperiod-bpv_minperiod,_loop
        cmp     a,b     x:(r0)+,a
        tlt     a,b     r0,r1
_loop
        nop
        move    r1,a
        sub     #bpv+gauss_pdf_len/2,a

        rts

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

bpv_insert      ; add gaussian pdf at ioi

        ; a=time, b=ampl

        cmp     #bpv_maxperiod,a
        bhs     _abort          ; out of bounds

        move    #gauss_pdf,r6
        move    #-1,m6

        add     #bpv,a

        move    #2,n2
        move    m6,m2
        move    a,r2
```

```
        ;asr    #8,b,b
        nop
        nop
        move    b,x0            ; x0=ampl

        move                    x:(r2)+,a       y:(r6)+,y0
        mac     x0,y0,a         x:(r2)-,b       y:(r6)+,y1
        do      #gauss_pdf_len/2,_loop
        macr    x0,y1,b         a,x:(r2)+n2
        move                    x:(r2)-,a       y:(r6)+,y0
        macr    x0,y0,a         b,x:(r2)+n2
        move                    x:(r2)-,b       y:(r6)+,y1
_loop   move                    a,x:(r2)+

_abort  rts
```

A.1.4 Zero lag cross correlation

```
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;
        ; cross correlation at lag 0 between artificial pulse
        ; train corresponding to n beats at detected tempo,
        ; and onsets in queue

xcorr   macro   npulses,nonsets

        move    x:onset_queue_pt,r0
        move    #onset_queue_size-1,m0

        move    m0,m1

        move    #gauss_acf,r2           ; r2->gaussian acf
        move    #-1,m2

        move    x:(r0),x0               ; x0=time of most recent beat
        move    x:beat_period,x1        ; x1=beat period in df time

        clr     a                       ; a=correlation sum

        do      #npulses,_loop

        move    r0,r1                   ; r1->beat queue
        do      #nonsets,_beats

        move    x:(r1)-,b
        sub     x0,b
        abs     b                       ; dt=|t_onset-t_pulse|
        cmp     #gauss_acf_len,b
        move    b,n2
```

```
        nop
        move    y:(r1+1),y1             ; onset amplitude
        move    y:(r2+n2),y0            ; acf lookup
        mac     y0,y1,a         ifle
_beats
        move    x0,b    ; next artificial pulse
        sub     x1,b
        nop
        move    b,x0
_loop
        endm
```

A.1.5 MIDI

```
midi_init_sci   ; initialize SCI for MIDI output

        movep   #$000202,x:M_SCR        ; tx enable, 1+8+1 bit async
        movep   #$0026,x:M_SCCR         ; TCM=0 RCM=0 SCP=0 COD=0 CD=39(-1)
                                        ;   --> 31508 baud (almost 31250)

        movep   #%011,x:M_PCRE          ; select pe2,txd,rxd
        movep   #%100,x:M_PRRE          ; make pe2 output
        movep   #%100,x:M_PDRE          ; set pe2 high (midi current source)

        rts

txwait  jclr    #1,x:M_SSR,*            ;wait for tx to finish
        jclr    #0,x:M_SSR,*            ;wait for tx to finish
        rts

midi_clock
        bsr     txwait
        movep   #$f8,x:M_STXL
        rts

midi_start
        bsr     txwait
        movep   #$fa,x:M_STXL
        rts

midi_cont
        bsr     txwait
        movep   #$fb,x:M_STXL
        rts

midi_stop
        bsr     txwait
        movep   #$fc,x:M_STXL
        rts
```

```
midi_message    ; send 3 bytes in x0
        bsr     txwait
        move    x0,x:M_STXH
        bsr     txwait
        move    x0,x:M_STXM
        bsr     txwait
        move    x0,x:M_STXL
        rts
```

A.1.6 Thresholding

```
        move    #minififo,r0      ; three sample fifo
        move    #-1,m0

        move    x:threshold,y0
        move    x:th_decay,y1

        move    x:(r0+1),x0
        move    x0,x:(r0+2)
        move    x:(r0+0),x0
        move    x0,x:(r0+1)       ; x0 = sample of interest
        move    x1,x:(r0+0)       ; insert new sample from above

        mpyr    y1,y0,b           x0,a     ; b=threshold*decay      a=diff(y)
        max     a,b               x:th_min_level,a
        max     a,b               x0,a
        nop
        cmp     b,a               b,x:threshold
        blt     _no       ; below

        move    x0,a
        move    x:(r0+2),b
        cmp     b,a
        blt     _no       ; not local maximum
        move    x:(r0+0),b
        cmp     b,a
        blt     _no       ; not local maximum

        move    x:df_time,a
        move    x:t_prev,b
        sub     b,a                         x:th_min_time,b
        cmp     b,a
        blt     _no       ; too early since last detection

_yes    move    x:beat_queue_pt,r0
        move    #beat_queue_size-1,m0

        bset    #detection_pending,x:flags
        debugcs ; previous detection not processed
```

```
        bset    #lclick_pending,x:flags

        move    (r0)+
        move    x0,y:(r0)           ; amplitude
        move    x:df_time,x0
        move    x0,x:(r0)           ; time
        move    r0,x:beat_queue_pt

        move    #>48000/FRAMESIZE/10,x0 ; 0.1 s
        move    x0,x:led_counter
        bset    #M_DO,x:M_TCSR0

        move    x:df_time,x0
        move    x0,x:t_prev
_no

        move    x:df_time,a
        add     #1,a
        nop
        move    a,x:df_time
```

Bibliography

M. Alonso, B. David, and G. Richard, "A Study of Tempo Tracking Algorithms from Polyphonic Music Signals," in *Proceedings of the 4th COST 276 Workshop*, Bordeaux, France, 2003.

——, "Tempo and beat estimation of musical signals," in *Proceedings of International Conference on Music Information Retrieval*, 2004, pp. 158–163.

J. P. Bello and M. Sandler, "Phase-based note onset detection for music signals," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, Apr. 2003, pp. v/441–444. doi: 10.1109/ICASSP.2003.1200001

J. P. Bello, C. Duxbury, M. Davies, and M. Sandler, "On the use of phase and energy for musical onset detection in the complex domain," *IEEE Signal Processing Letters*, vol. 11, no. 6, pp. 553–556, 2004. doi: 10.1109/LSP.2004.827951

J. P. Bello, L. Daudet, S. A. Abdallah, C. Duxbury, M. Davies, and M. Sandler, "A tutorial on onset detection in music signals," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 1035–1047, Sep. 2005. doi: 10.1109/TSA.2005.851998

J. C. Brown, "Determination of the meter of musical scores by autocorrelation," *The Journal of the Acoustical Society of America*, vol. 94, no. 4, pp. 1953–1957, 1993. doi: 10.1121/1.407518

N. Collins, "A Comparison of Sound Onset Detection Algorithms with Emphasis on Psychoacoustically Motivated Detection," in *Audio Engineering Society 118th Convention*, Barcelona, Spain, 2005, p. 12 pp.

J. Dattorro, "The implementation of recursive digital filters for high-fidelity audio," *Journal of the Audio Engineering Society*, vol. 36, no. 11, pp. 851–878, 1988. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=5125

M. E. P. Davies and M. D. Plumbley, "Causal Tempo Tracking of Audio," *ISMIR 2004 Fifth International Conference on Music Information Retrieval Proceedings, Barcelona*, pp. 164–169, 2004.

——, "Context-Dependent Beat Tracking of Musical Audio," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1009–1020, 2007. doi: 10.1109/TASL.2006.885257

P. Desain, "A (de)composable theory of rythm perception," *Music Perception*, vol. 9, no. 4, pp. 439–454, 1992.

Freescale Semiconductor, *DSP56300 Family Manual*, 2005. [Online]. Available: http://www.freescale.com/files/dsp/doc/ref_manual/DSP56300FM.pdf

——, *DSP56303EVM User's Manual*, 1999. [Online]. Available: http://www.freescale.com/files/dsp/doc/ref_manual/DSP56303EVMUM.pdf

——, *DSP56303 User's Manual*, 2005. [Online]. Available: http://www.freescale.com/files/dsp/doc/ref_manual/DSP56303UM.pdf

B. Gold, A. V. Oppenheim, and C. M. Rader, "Theory and Implementation of the Discrete Hilbert Transform," in *Proceedings of the 1969 Polytechnic Institute of Brooklyn Symposium*, Brooklyn, NY, USA, 1969, pp. 235–250.

F. Gouyon, A. Klapuri, S. Dixon, M. Alonso, G. Tzanetakis, C. Uhle, and P. Cano, "An experimental comparison of audio tempo induction algorithms," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1832–1844, 2006.

J. Hass, *Introduction to Computer Music*. Indiana University, 2005, ch. 3.1. [Online]. Available: http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI.shtml

J. A. Hockman, "An Overview of Beat Tracking Techniques," 2008. [Online]. Available: http://www.music.mcgill.ca/~hockman/coursework/MUMT_611/final/jhockman_611final.pdf

K. Jensen and T. H. Andersen, "Real-Time Beat Estimation Using Feature Extraction," *Computer Music Modeling and Retrieval*, vol. 2771, pp. 155–178, 2004. doi: 10.1007/b12000

A. Klapuri, "Sound onset detection by applying psychoacoustic knowledge," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 6. IEEE, 1999, pp. 3089–3092 vol.6. doi: 10.1109/ICASSP.1999.757494

A. Klapuri, A. J. Eronen, and J. T. Astola, "Analysis of the Meter of Acoustic Musical Signals," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 14, no. 1, pp. 342–355, 2006. doi: 10.1109/TSA.2005.854090

J. Laroche, "Efficient Tempo and Beat Tracking in Audio Recordings," *Journal of the Audio Engineering Society*, vol. 51, no. 4, pp. 226–233, 2003. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=12235

W.-C. Lee and C.-C. J. Kuo, "Musical onset detection based on adaptive linear prediction," in *Proceedings of IEEE Internaional Conference on Multimedia and Expo (ICME)*. Toronto, Canada: IEEE, 2006, pp. 957–960. doi: 10.1109/ICME.2006.262679

P. Masri and A. Bateman, "Improved modelling of attack transients in music analysis-resynthesis," in *Proceedings of the International Computer Music Conference*, 1996, pp. 100–103. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.3987

O. Niemitalo. (2003) 90 degree phase difference IIR allpass pair. [Online]. Available: http://yehar.com/blog/?p=368

E. D. Scheirer, "Tempo and beat analysis of acoustic musical signals," *The Journal of the Acoustical Society of America*, vol. 103, no. 1, pp. 588–601, 1998. doi: 10.1121/1.421129

B. Schuller, F. Eyben, and G. Rigoll, "Fast and Robust Meter and Tempo Recognition for the Automatic Discrimination of Ballroom Dance Styles," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1, Apr. 2007, pp. 217–220. doi: 10.1109/ICASSP.2007.366655

K. Schutte. (2009) MATLAB and MIDI. [Online]. Available: http://www.kenschutte.com/midi

J. Seppänen, "Tatum grid analysis of musical signals," in *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, 2001, pp. 131–134. doi: 10.1109/ASPAA.2001.969560

J. O. Smith, *Introduction to Digital Filters with Audio Applications*. W3K Publishing, 2007. [Online]. Available: https://ccrma.stanford.edu/~jos/filters/

——, *Mathematics of the Discrete Fourier Transform*. W3K Publishing, 2007. [Online]. Available: https://ccrma.stanford.edu/~jos/mdft/

K. Tanghe, S. Degroeve, and B. De Baets, "An algorithm for detecting and labeling drum events in polyphonic music," in *Proceedings of the First Annual Music Information Retrieval Evaluation eXchange, London, UK*, London, UK, 2005.

C. Uhle and J. Herre, "Estimation of tempo, micro time and time signature from percussive music," in *Proc. Digital Audio Effects Workshop (DAFx)*, 2003. [Online]. Available: http://www.elec.qmul.ac.uk/dafx03/proceedings/pdfs/dafx46.pdf

R. Wilson, "Filter Topologies," *Journal of the Audio Engineering Society*, vol. 41, no. 9, pp. 667–678, 1993. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=6990