

To The Graduate School:

The members of the Committee approve the thesis of Jan-Eric Duden presented on May 3rd, 2002.

Jeffrey Van Baalen, Chairman

Diana Gordon-Spears, Co-chair

William Spears

William B. Stacy

James L. Caldwell

APPROVED:

Jeffrey Van Baalen, Head, Department of Computer Science

Stephen E. Williams, Dean, The Graduate School

Duden, Jan-Eric, An Improved Approach to Real-Time Beat-Induction from Digital Audio Signals,
Master of Science, Department of Computer Science, May, 2002

In this thesis, I present an approach for extracting the tempo and the beat-times of a musical piece in real-time. The approach is based on a model developed by Eric Scheirer. Synchronizing computer graphics to the beat of previously unknown music requires a highly efficient algorithm that produces regular beat output. A bit of accuracy can be sacrificed in order to achieve these goals. I improved the Eric Scheirer's model to meet these requirements. The model splits the audio signal into multiple bands. A bank of comb-filters analyzes each band. Features computed from the comb-filter states yield an estimate of the tempo and of the phase of the musical piece. Beat-times are generated using these estimates. Some variants of the implementation achieve similar accuracy to Eric Scheirer's implementation on the test-data, but are about 3 times faster. Other variants are nearly 6 times faster than the implementation of Eric Scheirer, but they sacrifice some accuracy for efficiency. An implementation shows the applicability of the algorithm in a real-time environment.

AN IMPROVED APPROACH TO REAL-TIME BEAT-INDUCTION
FROM DIGITAL AUDIO SIGNALS

by
Jan-Eric Duden

A thesis submitted to the Department of Computer Science
and The Graduate School of The University of Wyoming
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE

Laramie, Wyoming
May, 2002

© 2002 by Jan-Eric Duden

ACKNOWLEDGMENTS

I would like to thank my advisor Diana Gordon-Spears for her helpful comments and guidance on this thesis. I would also like to thank William Spears and William B. Stacy for their help and ideas.

I appreciate the help of Jarno Seppänen for providing me a port of Eric Scheirer's implementation. I would like to thank both of them for answering my questions regarding their implementations.

For reading my thesis, I would like thank Meg Wood, Marco Helmers and Rianne van Dijk. Thomas Böhne helped me with some tricky issues with \LaTeX . Discussions with my brother Hauke Duden helped to clarify parts of this thesis.

Thanks to my parents Hildegard and Erich Duden for their support over all these years. I would like to thank my fiancée Rianne van Dijk for her love and support.

Jan-Eric Duden

CONTENTS

1. <i>Introduction</i>	1
2. <i>Definitions</i>	3
2.1 Music	4
2.2 Digital Signal Processing	4
3. <i>Background</i>	9
3.1 Goto and Muraoka	10
3.2 Seppaenen	10
3.3 Scarborough, Miller, and Jones	10
3.4 Scheirer	11
4. <i>Objectives</i>	13
4.1 Functional Goals	14
4.2 Performance Goals	14
4.3 Implementation	15
4.4 Scope	15
5. <i>Beat-Induction from Simple Signals</i>	17
5.1 Using comb-filters to detect the frequency and phase of a click-train	18
5.2 Using a filter-bank of comb-filters to detect the click-train's frequency	19
5.3 Using a filter-bank and selection function	20
5.3.1 A more accurate tempo estimate	22
5.4 Determining the phase	24
5.5 Evaluation of the selection function features	24
5.6 Evaluation of a linear combination of features	25
5.7 Evaluation of the tempo correction	27
5.8 Conclusion	28
6. <i>Beat-Induction Architecture for Polyphonic Music</i>	29
6.1 Preprocessing	31
6.1.1 Filter-bank	31
6.1.2 Envelope Extraction and Down-Sampling	32
6.1.3 Derivative	32
6.2 Resonant Filter-bank	32
6.3 Post-processing	32

6.3.1	Generating smooth tempo transitions	33
6.4	Evaluation	36
6.4.1	Evaluation Measures	36
6.4.2	Methodology and Test-Data	38
6.4.3	Test Results	38
6.5	Conclusion	39
7.	<i>An Optimized Implementation</i>	41
7.1	The beat-induction architecture in C++	42
7.1.1	Beat-Induction Class (1)	43
7.1.2	BandPath Class	43
7.1.3	Comb-Filter Class	44
7.1.4	Beat-Induction Class (2)	44
7.2	Test-Setup	44
7.3	Optimizations	44
7.3.1	First Iteration (Sample rate reduced)	45
7.3.2	Second Iteration (Number of comb-filters reduced)	45
7.3.3	Third Iteration (Phase computation optimized)	46
7.3.4	Fourth Iteration (on-the-fly optimization)	46
7.3.5	Influence of the optimizations on the quality	48
7.4	Performance Comparison with Scheirer's Approach	49
7.4.1	Future Work	50
7.5	Visualizer using the Beat-Induction Model	50
7.6	Synchronous Version	50
7.7	Conclusion	51
8.	<i>Conclusion</i>	53
8.1	Contributions	54
8.2	Summary	54
8.3	Future work	55
	<i>Bibliography</i>	57

LIST OF FIGURES

2.1	Unit Impulse	6
2.2	Click-Train	6
2.3	Shape of a one sided Hann window	7
2.4	Combfiler implementation in C++	8
5.1	The period of the click-train is a <i>full unit larger</i> than the delay of the comb-filter.	23
5.2	The period of the click-train is a <i>full unit smaller</i> than the delay of the comb-filter.	23
5.3	The period of the click-train is a <i>fraction of a unit larger</i> than the delay of the comb-filter.	23
5.4	The period of the click-train is a <i>fraction of a unit smaller</i> than the delay of the comb-filter.	23
5.5	Comb-filter delay equals click-train period.	23
5.6	the different pulse shapes	25
5.7	Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Unaligned	26
5.8	Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Aligned	27
5.9	Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Unaligned	27
5.10	Comparison of Tempo-Prediction-Errors	28
6.1	Scheirer's processing algorithm	30
6.2	Duden's processing algorithm	30
6.3	Transformation of the input signal	31
6.4	A perfect transition to half of the tempo	34
6.5	An awkward transition	34
6.6	Delaying the transition makes the transition smoother	34
7.1	Implementation	42
7.2	Data Flow between Client Application and the Beat-Induction Architecture	43
7.3	Bottle-neck analysis with no optimizations enabled	45
7.4	Reduced sample rate	46
7.5	Reduced sample rate and 99 comb-filters	47
7.6	Smoothing in GetPhase eliminated	47
7.7	After applying all optimizations	48
7.8	BeatVis Visualizer and Ashampoo Media Player	50

LIST OF TABLES

5.1	Evaluation of Features detecting Click-Train Periods	25
5.2	Average period where the features failed detecting click-train periods	26
6.1	Qualitative Comparison	38
7.1	Influence of optimizations on the quality of the output	49
7.2	Comparison of execution times	49

Chapter 1

INTRODUCTION

Most people have experienced the beat of a musical piece and they do not have any trouble tapping with their foot according to the beat. This makes most people think that foot-tapping, also beat-tracking or beat-induction, is trivial, but it turns out to be a non-trivial task when a computer should perform it. It is challenging, because the phenomenon “beat” cannot be easily described in terms of a raw data stream encoding the musical piece. The correctness of beat-induction algorithms is difficult to verify, since the beat information for a musical piece is usually not available. In most cases the algorithm has to be verified manually.

Applications for beat-induction algorithms are mainly related to multimedia systems. Real-Time Beat-Induction may be used to animate graphics such as displaying someone dancing to music or to control a light-stage ([Goto and Muraoka 1998](#)).

Query-by-Humming systems allow users to search for musical pieces by humming the melody. Chai ([2001](#)) states that the rhythmic structure plays an important role in characterizing the melody of songs. Feature descriptions with rhythmic information allow shorter queries than those without rhythmic information.

Scheirer ([2000](#)) mentions that tempo extraction might also play an important role to annotate multimedia content with meta data as it is proposed with the MPEG7 standard. A tempo extraction system could provide one feature in such a framework.

This thesis addresses the problem of how to ‘teach’ the computer to keep track of beat. It will build on a known approach and tries to improve its quality and performance, so that the approach is more suitable to animate computer graphics synchronized to the beat of a musical piece.

This thesis is structured as follows:

- Chapter 2, *Definitions*, defines essential terms in signal processing and music.
- Chapter 3, *Background*, discusses briefly previous approaches to real-time beat-induction
- Chapter 4, *Objectives*, states the goal that I strive to achieve for the thesis.
- Chapter 5, *Beat-Induction from Simple Signals*, discusses issues detecting the period and the phase of click-trains with comb-filter-banks.
- Chapter 6, *Beat-Induction Architecture for Polyphonic Music*, presents a generalized approach for polyphonic audio signals and results of a short evaluation.
- Chapter 7, *An Optimized Implementation*, presents the impact on efficiency and accuracy of a range of optimizations.
- Chapter 8, *Conclusion*, lists the contributions and summarizes the results.

Chapter 2

DEFINITIONS

2.1 Music

In the following section I present definitions of beat and associated terms.

Definition 2.1.1 (Beat and Tempo): *A pulse or beat is “one of a series of regularly recurring, precisely equivalent stimuli.” (Cooper and Meyer 1960). The series of pulses defines the tempo of the piece of music (Scheirer 1998b). The series can be perfectly described by two parameters:*

1. *frequency - proportional to the reciprocal of the distance of two adjacent beats. The unit of frequency is beats per minute (BPM).*
2. *phase - position of an arbitrary pulse or beat. Its unit is measured in samples which can be easily translated to a time scale.*

Also, note that the stimuli creating the beat may be omitted for some time without affecting the tempo.

Definition 2.1.2 (Beat Information): *Beat information consists of the frequency and the phase of a beat.*

Definition 2.1.3 (Inter-Beat-Interval (IBI)): *The inter-beat-interval (IBI) is the interval described by two beats. If the tempo is constant, then the IBI is equal to the reciprocal of the tempo (Goto and Muraoka 1998).*

Definition 2.1.4 (Rhythm and Meter): *The meter of musical piece is perceived as groups of beats. It describes only patterns of duration (Bernstein and Picker 1966). The rhythm is a pattern of accented and un-accented beats.*

2.2 Digital Signal Processing

This thesis will use digital signal processing as a tool to analyze digital signals that may be provided by a wide range of possible sources, such as sound-cards or multi-media players. The following section will introduce definitions that are used throughout the document. Most of them are based on definitions found in Sanijit K. Mitra (1993).

Definition 2.2.1 (Signal): *A signal is a mapping from the space or time domain to another domain usually the real numbers or the complex numbers.*

Examples of signals are sound wave forms, time series, or images. Here, I will focus on signals that are functions of the discrete time domain mapping to a discrete codomain. This leads to the next definition.

Definition 2.2.2 (Digital Signal): A digital signal is a signal whose domain and codomain are discrete.

Digital signals are denoted by bold letters and $\mathbf{s}[t]$ denotes the value of signal \mathbf{s} at time/sample t .

Definition 2.2.3 (Analog Signal): An analog signal is a signal whose domain and codomain are continuous.

Since the text deals with digital signals, it will use the terms signal and digital signal interchangeably.

Definition 2.2.4 (Sample): A sample refers to one value of the signal.

Definition 2.2.5 (Sample Interval): The interval between two adjacent samples in the discrete time domain is a sample interval. It is assumed that this interval is constant for a signal. Usually it is measured in seconds.

Definition 2.2.6 (Sample Rate): The sample rate is the reciprocal value of the sample interval, which is measured in Hertz where $\text{Hz} = 1/s$.

Definition 2.2.7 (Bits per Sample): Defines the number of bits that are used to store one sample.

Bits per sample is a measure of how well the digital signal resembles the analog representation of the signal. Digital Audio on a CD is stored with a sample rate of 44.1 kHz and 32 Bits per sample. There are 16 Bits for the left and 16 Bits for the right channel, since CD Audio is stereo.

Definition 2.2.8 (PCM): PCM is the abbreviation of pulse code modulation. It is the basic format to store digital audio signals. The PCM format is the discrete form of the analog wave audio signal. The algorithms described later in this thesis will all work on audio signals in PCM format.

Definition 2.2.9 (Unit Impulse): A unit impulse (see Figure 2.1) is a digital signal \mathbf{u} where

$$\mathbf{u}[k] \equiv \begin{cases} 1 & k = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Definition 2.2.10 (Click-Train): A click-train \mathbf{c} (see Figure 2.2) with period $P(c) = \lambda$ or frequency $F(c) = 1/\lambda$, phase $\Phi(c) = \phi$ and peak C is a discrete signal with

$$\mathbf{c}[x - \phi] \equiv \begin{cases} C \cdot \mathbf{u}[x \bmod \lambda] & \forall x \geq 0 \\ 0 & \text{otherwise} \end{cases} .$$

Definition 2.2.11 (One-Sided Hann Window): An one-sided Hann window \mathbf{hann} (see Figure 2.3) of length N is given by the following formula

$$\mathbf{hann}[x] \equiv \begin{cases} 0.5 \cdot \left(1 + \cos\left(2\pi \frac{x+N}{2N-1}\right)\right) & \text{where } 0 \leq x < N \\ 0 & \text{otherwise.} \end{cases}$$

Chapter 6 discusses the application of the one-sided Hann window. It is used to extract the envelope of the audio signal.

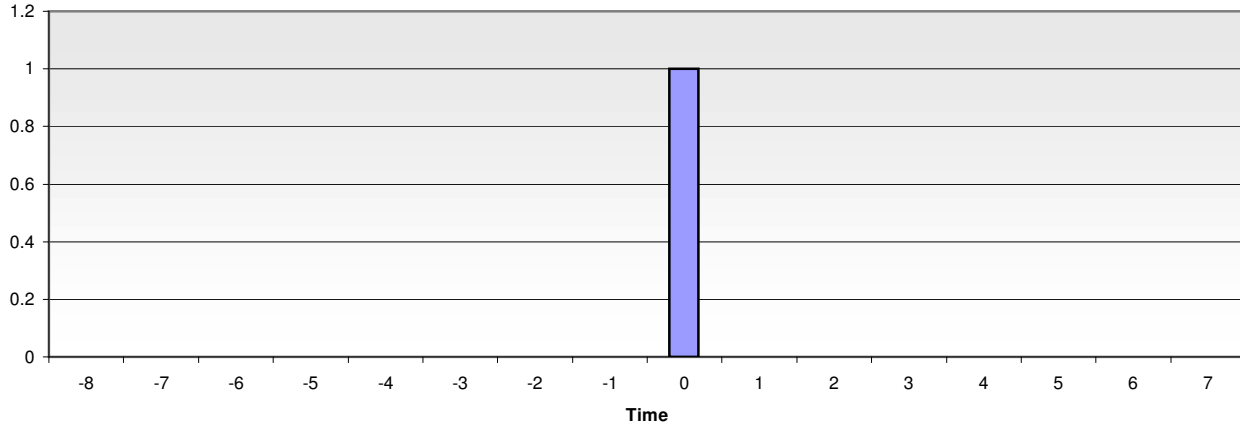


Figure 2.1: Unit Impulse

Click-Train c, P(c)=8, peak C=1

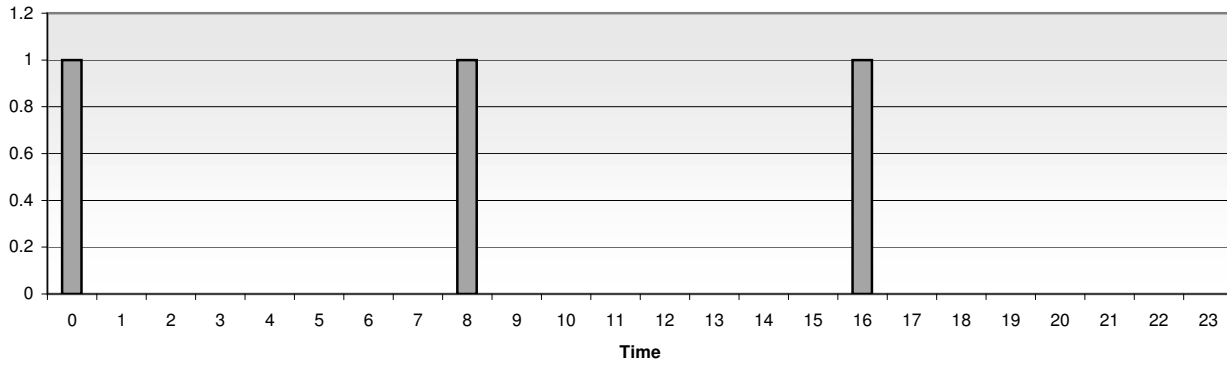


Figure 2.2: Click-Train

Definition 2.2.12 (Convolution): The convolution of two signals \mathbf{h} and \mathbf{x} is

$$\mathbf{x}[n] \otimes \mathbf{h}[n] \equiv \sum_{k=-\infty}^{\infty} \mathbf{h}[n-k] \cdot \mathbf{x}[k].$$

Definition 2.2.13 (Digital Filter): A digital filter \mathbf{T} transforms an input signal $\mathbf{x}[n]$ into an output signal

$$\mathbf{y}[n] \equiv \mathbf{T}\{\mathbf{x}[n]\}.$$

At several points in the thesis, I used filters to isolate frequency bands of a signal.

Definition 2.2.14 (Comb-Filter, Resonant Filter or Resonator): A resonant filter with delay or period γ is a filter

$$\mathbf{r}[n] = \mathbf{R}\{\mathbf{x}[n]\} \equiv \alpha \mathbf{r}[n-\gamma] + \beta \mathbf{x}[n].$$

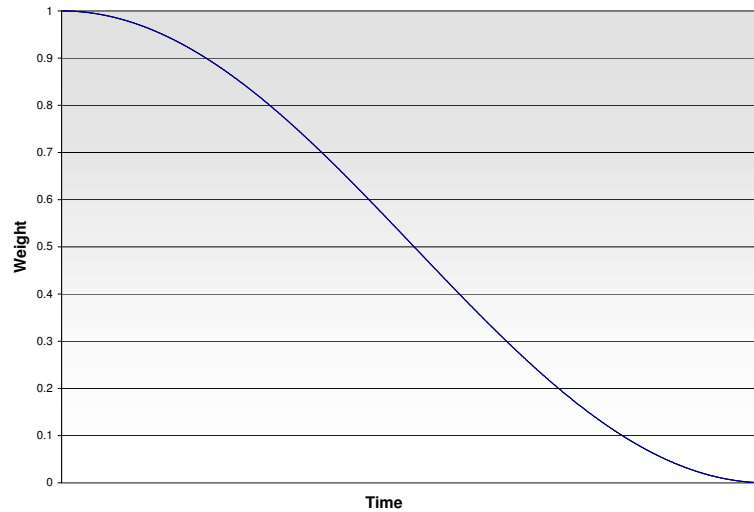


Figure 2.3: Shape of a one sided Hann window

The parameters α and β define how strong the comb-filter state is influenced by past input.

It can be implemented as shown in Figure 2.4. I used a slightly different approach to implement comb-filters. Instead of processing data as soon as it arrives, I rather store it in a buffer having a size that equals the delay of the comb-filter. Once the buffer is full, all samples in the buffer are processed using the comb-filter equation. I call this an update of a comb-filter. I hope to have more exact feature values by using this update mechanism. These features are discussed in Section 5.3.

Definition 2.2.15 (Filter Bank): *A filter bank consists of parallel working filters of the same type that filter the same input signal.*

Definition 2.2.16 (Envelope): *The envelope of a signal is a rough outline of the absolute values of the amplitudes.*

```
//This variable stores the state of the combfilter  
//The elements of this array are referred to as registers  
//or cells.
```

```
double aState[gamma];
```

```
//The cursor keeps track of where to write next
```

```
int nCursor=0;
```

```
double alpha;
```

```
double beta;
```

10

```
// The function reads one sample in and returns a filtered sample
```

```
double combfilter(double in)
```

```
{
```

```
    //apply comb-filter formula
```

```
    aState[nCursor]=alpha*aState[nCursor]+beta*in;
```

```
    double out=aState[nCursor];
```

```
    //advance cursor
```

```
    nCursor++;
```

```
    //take care of wrap around
```

```
    nCursor=nCursor % gamma;
```

```
    return out;
```

```
}
```

20

Figure 2.4: Combfilter implementation in C++

Chapter 3

BACKGROUND

This section presents approaches to real-time beat-induction. Different models have been evaluated. Models that analyze audio signals were presented in (Goto and Muraoka 1998), (Goto and Muraoka 1995), (Scheirer 1995), (Scheirer 1998b), (Foote and Uchihashi 2001), (Dixon 2001a), (Laroche 2001), (Tzanetakis, Essl, , and Cook 2001) and (Seppänen 2001). Summaries for most of these models were presented in (Seppänen 2001). I briefly summarize the models presented in (Goto and Muraoka 1998), (Seppänen 2001) and (Scheirer 1998b), since they are causal models for audio signals. The model by Scarborough, Miller, and Jones (1992) summarized below is similar to the comb-filter model used in this thesis.

3.1 Goto and Muraoka

M. Goto and Y. Muraoka describe a multi-agent beat-tracking model in (Goto and Muraoka 1998). The model not only extracts beat patterns, but it is also capable of detecting a hierarchical beat structure by incorporating heuristics that guide the algorithm. The algorithm computes onset times and drum patterns in the frequency domain. Using autocorrelation and cross-correlation, each agent infers the lowest level of beat structure. Higher-level checkers apply heuristics and report the results to agents which will use these to complete the prediction. Each agent evaluates the reliability of its hypothesis and the most reliable one is selected. The model they presented is suitable for real-time beat-tracking. Their implementation analyzes audio signals in real-time on a parallel machine.

3.2 Seppänen

In his master's thesis, Seppänen (2001) presents a model that uses audio signals to extract the tempo information and the underlying tatum (smallest rhythmic unit). The model has four stages. The first stage detects the onsets of the signal. The tatum grid of the musical piece is estimated in the second stage. In step three, accents are detected. The results of the previous three steps are used by the final stage that estimates the beat grid. The system is fast enough to work in real-time.

3.3 Scarborough, Miller, and Jones

Scarborough, Miller, and Jones (1992) present with BeatNet a model using a bank of oscillators. The model processes durations of notes and each oscillator can be excited by a pre-defined duration/phase pair. Through the interconnection of oscillators the system is able to induce metrical levels. Having units corresponding to period/phase pairs, the system is similar to the comb-filter bank used by Scheirer.

3.4 Scheirer

Scheirer (1998b) describes a tempo extraction algorithm using a filter-bank that splits up the incoming stream into 6 bands. Six staged pipelines analyze the bands. The first stage extracts the envelope and the second computes the first-order derivative. A bank of resonant filters analyzes the result of the first stages where each filter corresponds to a beat-frequency. A final step summarizes the analysis. It combines the output of the six resonant-filter banks for each frequency and picks the filter of the combined resonant filters with the highest energy. The corresponding frequency of the filter determines the tempo. The analysis of the cells of the filter provides the phase of the beat. In the following chapters, I try to improve Scheirer's approach and discuss the model in more detail.

Chapter 4

OBJECTIVES

In this thesis, I strive to achieve a range of goals, which I categorize into functional goals and performance goals. The functional goals aim for a high quality beat-tracking system while the performance goals aim for a system that is computationally fast. Achieving these goals should help to make the beat-induction system more suitable for synchronizing computer graphics with the beat of a musical piece in real-time.

4.1 Functional Goals

The system generates accurate output. The accuracy of the system's output will be compared to a manually created reference. It is important to note that recognizing a tempo that, for example, is twice as fast as the correct one, has to be considered as a partial failure. The system needs to find the correct tempo of a musical piece.

The system accurately predicts the frequency of the beat. It is also important that the system is trying to predict the accurate frequency of the beat. If the tempo is not accurate enough the beat-prediction might get out of phase after a while when the system does not correct the phase constantly.

The system strives for regularity. The tempo output might vary due to inaccuracy or problems detecting the beat. Without a graceful transition between different beat speeds, the system might for example generate two beats in a short succession, even though the tempo is much slower. This may cause flickering in an animation synchronized with the beat-output of the system.

The system uses samples in PCM data format. The system needs to handle PCM data which is the raw sound data that almost every PC is able to play and to provide.

The system provides results anytime. Even if the system has not received enough data to make a definite conclusion about the beat, the system has to provide an approximate solution if requested.

4.2 Performance Goals

The algorithm requires a negligible amount of processor time on a modern system. An application of this system is to present on-beat animations on a modern computer system. Since such a system has no special hardware and it has to perform other tasks such as displaying the animation, the beat-tracking system has to be as efficient as possible.

The algorithm works with a small write ahead buffer less than a second. The system should also be able to cope with music that is streamed to the computers of users over the Internet. Since the song is usually played before all data has arrived, the beat-tracking system can only rely on a small write-ahead buffer of sound data and has to provide the beat-information in time.

4.3 Implementation

In order to show that the architecture is capable of driving a visualization that shows on-beat-animation a sample application is developed using the beat-induction architecture.

4.4 Scope

In this thesis, I deal only with the tempo and beat of a musical piece. The metric and rhythmic structure are ignored. Furthermore, it is assumed that the tempo of the piece of music is constant. As pointed out in ([Scheirer 1998b](#)), the tempo and beats in the musical piece may be used for further analysis of the metric and rhythmic structure.

Chapter 5

BEAT-INDUCTION FROM SIMPLE SIGNALS

This chapter deals with the beat-detection of simple click-trains using comb-filters. I discuss issues that come along with this approach. The approach using comb-filters was first proposed in (Scheirer 1998b).

5.1 Using comb-filters to detect the frequency and phase of a click-train

Theorem 5.1.1: *Assume a click-train \mathbf{c} with $P(\mathbf{c}) = \lambda$, peak of height C and $\beta = 1 - \alpha$, then a comb filter \mathbf{R} with delay λ will eventually gain a maximum peak of height C (Scheirer 1998b).*

Proof. Let \mathbf{r} be the output signal of $\mathbf{R}\{\mathbf{c}\}$:

$$\begin{aligned} \mathbf{r}[0\lambda] &= (1 - \alpha)C \\ \mathbf{r}[1\lambda] &= \alpha(1 - \alpha)C + (1 - \alpha)C = (\alpha - \alpha^2 + 1 - \alpha)C = (1 - \alpha^2)C \\ \mathbf{r}[2\lambda] &= \alpha(1 - \alpha^2)C + (1 - \alpha)C = (\alpha - \alpha^3 + 1 - \alpha)C = (1 - \alpha^3)C \\ &\dots \\ \mathbf{r}[n\lambda] &= (1 - \alpha^{n+1})C \\ &\downarrow \\ \lim_{n \rightarrow \infty} \mathbf{r}[n\lambda] &= C \text{ if } \alpha \in [0 \dots 1) \end{aligned}$$

□

The position x of the peak in the register array of the comb-filter is the phase of the click-train $\Phi(\mathbf{c})$.

The result allows to induce the period of the click-train or an integer multiple of the period from a peak of height C in resonant filter state. Other periodicities than the filter delay and integer multiple of it do not cause the resonant filter to have a maximum peak of height C which is proven in the following theorem.

Definition 5.1.1 (Resonance): *The resonance of a comb filter \mathbf{R} to a click-train \mathbf{c} is denoted with $\mathbf{res}(\mathbf{R}, \mathbf{c})$ and is the height of the highest peak of the comb-filters output having an infinitely long click-train as input.*

Theorem 5.1.2: *Assume a click train \mathbf{c} with $P(\mathbf{c}) = \lambda$ and peak C . Then a comb filter \mathbf{R} with delay $\gamma \neq n\lambda; n \in \mathbb{N}$ will not gain a maximum peak C , instead the resonance $\mathbf{res}(\mathbf{R}, \mathbf{c})$ of the comb-filter to click-train is*

$$\mathbf{res}(\mathbf{R}, \mathbf{c}) \equiv \frac{(1 - \alpha) \cdot C}{1 - \alpha^{\frac{\rho}{\gamma}}} \text{ if } \alpha \in [0 \dots 1[$$

where

$$\rho = \text{lcm}(\lambda, \gamma) \text{ (least common multiple)}$$

The comb-filter receives impulses every ρ steps. Scheirer states this theorem in (Scheirer 1998b)¹.

¹ Note that this section corrects a small error in (Scheirer 1998b) and (Scheirer 2000).

Proof.

$$\begin{aligned}
\mathbf{r}[0\rho] &= (1 - \alpha)C \\
\mathbf{r}[1\rho] &= \alpha^{\frac{\rho}{\gamma}}(1 - \alpha)C + (1 - \alpha)C = (\alpha^{\frac{\rho}{\gamma}} + 1)(1 - \alpha)C \\
\mathbf{r}[2\rho] &= \alpha^{\frac{\rho}{\gamma}}(\alpha^{\frac{\rho}{\gamma}} + 1)(1 - \alpha)C + (1 - \alpha)C = (\alpha^{\frac{\rho}{\gamma} \cdot 2} + \alpha^{\frac{\rho}{\gamma}} + 1)(1 - \alpha)C \\
&\dots \\
\mathbf{r}[n\rho] &= \sum_{i=0}^n \alpha^{\frac{\rho}{\gamma} \cdot i} \cdot (1 - \alpha)C \\
&\Downarrow \\
\mathbf{res}(\mathbf{R}, \mathbf{c}) &\equiv \lim_{n \rightarrow \infty} \mathbf{r}[n\rho] = \frac{(1 - \alpha) \cdot C}{1 - \alpha^{\frac{\rho}{\gamma}}} \text{ if } \alpha \in [0 \dots 1)
\end{aligned}$$

□

Clearly, for delays not matching an integer multiple of the click-train's period $1 - \alpha^{\frac{\rho}{\gamma}} > (1 - \alpha)$ holds, since $\rho < \gamma$ and therefore $\mathbf{res}(\mathbf{R}, \mathbf{c}) < C$.

Suppose the resonant filter's delay is $\gamma = k \cdot \lambda$. Then $\text{lcm}(\gamma, \lambda) = \gamma$ implies that

$$1 - \alpha^{\frac{\rho}{\gamma}} = (1 - \alpha) \implies \mathbf{res}(\mathbf{R}, \mathbf{c}) = C$$

Thus inspecting only the peaks of comb-filters as presented in this section can only be a heuristic to detect the click-trains frequency. This behavior was also reported for the oscillator-filter-bank presented in (Miller, Scarborough, and Jones 1992). The following section addresses this issue and other problems that arise when a filter-bank consisting of many comb-filters detects the click-train period.

5.2 Using a filter-bank of comb-filters to detect the click-train's frequency

The section gives an overview of how to use a filter-bank of comb-filters with different delays to detect a click-train period. This approach was proposed by Scheirer (Scheirer 1998b). The input signal is dispatched to each of the comb-filters. A selection function, that examines the comb-filters' registers, evaluates the comb-filters and chooses the comb-filter with the best evaluation as best indicator for $F(c)$. Scheirer uses the comb-filters as oscillators, that try to resonate with the pulses of the click-train. As pointed out in Chapter 3, it is similar to the bank of oscillators presented in (Miller, Scarborough, and Jones 1992). In these two approaches, each oscillator represents one period. Large and Kolen (1994) and Toiviainen (1998) use oscillators, that adapt their period with the signal. On the first glance, the model used here can only detect a fixed number of periods. However, in Section 5.3.1 I show how to extend the comb-filter-bank approach to overcome this restriction.

In order to design a comb-filter-bank several points need to be addressed including:

- the number of comb-filters

- the delay λ of each comb-filter
- the α and β parameter of each comb-filter
- a selection function.

The number of comb-filters determines the number of click-trains the filter-bank is able to detect. It is an efficiency/quality trade-off. Less comb-filters improve the efficiency while more comb-filter improve the quality of the detection.

The filter-bank has a comb-filter with the corresponding delay for each click-train period. The parameter α of each comb-filter specify the decay of old information. If the half-time of impulse response of all comb-filters is equal then a click-train's reinforcement will have the same impact on all comb-filters. It is unclear how to choose appropriate values for the β parameter.

The selection-function has to select the comb-filter making the best prediction about the analyzed signal. The following issues emerge that make the decision difficult.

1. Two comb-filters may detect a single click-train period.
2. A comb-filter may have similar peaks with different click-train periods.
3. Assume three comb-filters \mathbf{R}_1 , \mathbf{R}_2 , \mathbf{R}_3 with delays γ_1 , γ_2 and γ_3 where $\gamma_1 < \gamma_2 < \gamma_3$. Given a click-train with period λ , where $\gamma_1 < \lambda < \gamma_2$, \mathbf{R}_3 may have the highest peak.

In the previous section theorem 5.1.2 showed that not only a comb-filter whose delay matches the period of click-train will have a high peak but also a comb-filter with a delay two times the period of the click-train will eventually have a high peak. Also the previous section indicated that comb-filter's peak does not uniquely determine the period of an analyzed click-train. The third point shows another issue when the number of comb-filters is not sufficient. Consider the following example:

$$\begin{aligned} &\text{Delays of the three comb-filters: } \gamma_1 = 5, \gamma_2 = 7, \gamma_3 = 9. \\ &\text{Period of the click-train: } \gamma = 6 \\ &\frac{lcm(\lambda, \gamma_1)}{\gamma_1} = \frac{30}{5} = 6, \quad \frac{lcm(\lambda, \gamma_2)}{\gamma_2} = \frac{42}{7} = 6 \quad \text{and} \quad \frac{lcm(\lambda, \gamma_3)}{\gamma_3} = \frac{18}{9} = 2 \\ &\implies \mathbf{res}(\mathbf{R}_1, \mathbf{c}) = \mathbf{res}(\mathbf{R}_2, \mathbf{c}) < \mathbf{res}(\mathbf{R}_3, \mathbf{c}) \end{aligned}$$

Since the quotient of comb-filter \mathbf{R}_3 's delay and the period of the click-train is harmonic, \mathbf{R}_1 and \mathbf{R}_2 have a lower resonance.

5.3 Using a filter-bank and selection function

The selection function is the most important aspect of the filter-bank. It ultimately determines the prediction of the beat's tempo by mapping comb-filters to a real value, that orders the comb-filters

according to their correctness. The comb-filter with the lowest value best predicts the tempo. The proposed beat-induction algorithm uses a selection-function **sel** that uses several features:

1. the shape of the comb-filter
2. the speed of how fast the peak shifts
3. the energy of the comb-filter, that was used by Scheirer.

In the following paragraphs, let n be the period of the comb-filter and $r'[0..n-1]$ be the registers of the comb-filter state. Furthermore this notation is extended such that $r[i] \equiv r'[(i+n) \bmod n]$. The peak of the comb-filter state is computed as follows:

$$\mathbf{Peak}(r) \equiv \operatorname{argmax}_{i=0,\dots,n-1} (r[i])$$

The shape of the comb-filter register should resemble just a peak only, if the delay matches the period. It measures how much the shape of the comb-filter looks like a triangle with a small base. A scan through the registers reveals the peak of the comb-filter. The squared values of each register are weighted by a function \mathbf{T} and summed up. The function \mathbf{T} is one at except a small interval. The weights in that interval approximate the shape of a triangle. The result of this weighted sum is normalized by the sum of the squares of the register values.

$$\mathbf{shape}(r) \equiv \frac{\sum_{i=0}^{n-1} \mathbf{T}(i, \mathbf{Peak}(r)) r[i]^2}{\sum_{i=0}^{n-1} r[i]^2}$$

$$\mathbf{T}(i, p) \equiv \begin{cases} 1.0 & \text{if } |i-p| > 3 \\ 0.8 & \text{if } |i-p| = 3 \\ 0.5 & \text{if } |i-p| = 2 \\ 0.2 & \text{if } |i-p| = 1 \\ 0.0 & \text{if } |i-p| = 0 \end{cases}$$

I experimented a bit with different triangle sizes. The triangle used here seems to work best.

The second feature computes how fast the peak shifts for each update of the filter's registers. If the filter is completely in resonance with the input signal the peak will not shift. If there is a slight mismatch the peak will shift to the left if the comb-filter's delay is greater than the input signal's period. It will shift to the right if the comb-filter's delay is smaller. The speed criterion only works if the comb-filter's delay is close to an integer multiple of the click-train's period. Otherwise the shifts of the peak get too large to track accurately.

$$\mathbf{shift}(r) \equiv \mathbf{Peak}(r) - \mathbf{Peak}(r_{\text{prev}})$$

where c_{prev} is the previous comb-filter state. So, the shift feature can only be used if a sequence of comb-filter states is provided.

Scheirer (1998b) uses the energy criteria in his approach. The energy is computed in two steps. First, the sum of the square of the cell values is determined. Second, the differences between the square-root of the sum and each cell-value is summed up and normalized.

$$\begin{aligned} \mathbf{energy}(r) &\equiv \frac{1}{n} \cdot \sum_{i=0}^{n-1} \sqrt{e} - r[i] \\ e &\equiv \sum_{i=0}^{n-1} r[i]^2 \end{aligned}$$

The selection function is the linear combination of these features. Once the comb-filter is selected the speed of the peak shifts can be applied to make a more accurate prediction of the actual speed.

5.3.1 A more accurate tempo estimate

The purpose of a more accurate tempo estimate is to help to fulfill the functional goal 4.1. Dixon (2001a) pointed out that Scheirer's approach needs to have a large bank of comb-filters to accurately detect a broad range of tempi. In order to reduce the size of the bank, this approach might result in a efficient and accurate model.

I assume that the click-train's period is close to the comb-filter's delay. Let λ , ρ be the comb-filter's delay and the click-train's period. If $\tau = \rho/\lambda$ is known the ρ can be determined by solving this equation. Since λ is the comb-filter delay, only τ is unknown. I state that τ can be approximated by examining the shifts of the peak in the comb-filter register array.

Let s be the number of cells the peaks shifts per update. Then

$$\tau = 1 + \frac{s}{\lambda} \implies \rho = \tau \cdot \lambda = \left(1 + \frac{s}{\lambda}\right)\lambda = \lambda + s$$

An easy example is a resonator with period λ and a click-train with period $\rho = \lambda + 1$. If the click-train is one cell too slow, the peak of the comb-filter will move one cell every update of the resonator's state. Figure 5.1, shows this phenomenon. The case, where the click-train is one cell too fast, is displayed in Figure 5.2. The change of the state over time, when the click-train is only a bit too slow or a bit too fast, is shown in figures 5.3 and 5.4. The height of the peak does not stay at one level, since the peak does not always get a reinforcement in the same cell. Just before the peak shifts to an adjacent cell, the adjacent cell already gets a reinforcement, but at that point the cell value is not high enough to be the new peak. Notice that the sample rate is 210 Hz, thus the resonance frequency of the filter with a delay of 71 is 177.464 BPM.

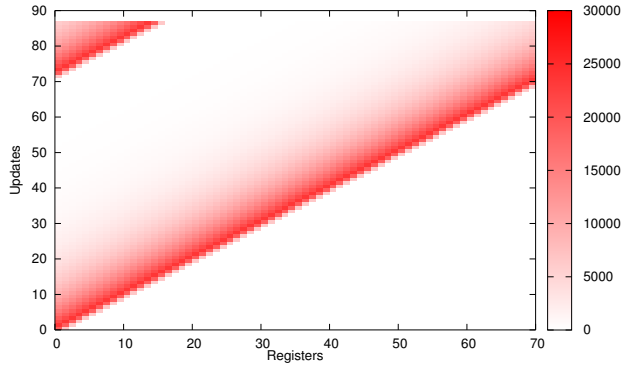


Figure 5.1: The period of the click-train is a *full unit larger* than the delay of the comb-filter.

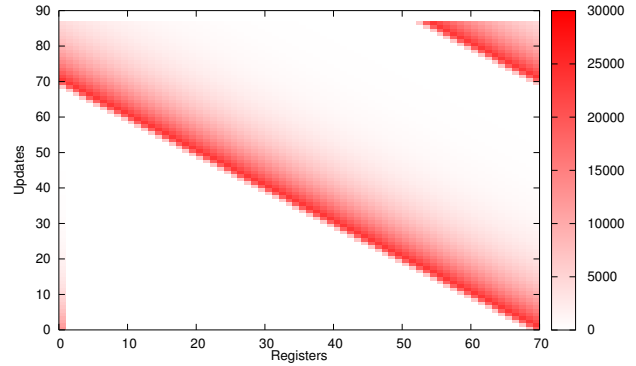


Figure 5.2: The period of the click-train is a *full unit smaller* than the delay of the comb-filter.

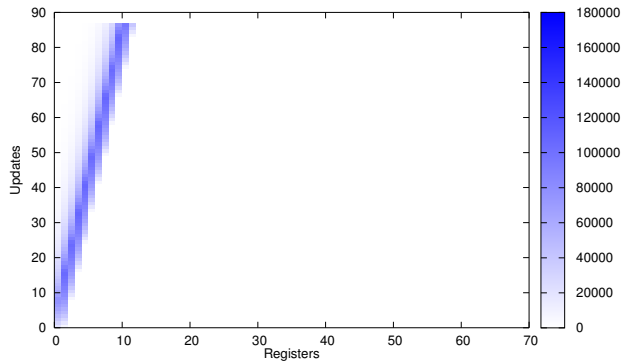


Figure 5.3: The period of the click-train is a *fraction of a unit larger* than the delay of the comb-filter.

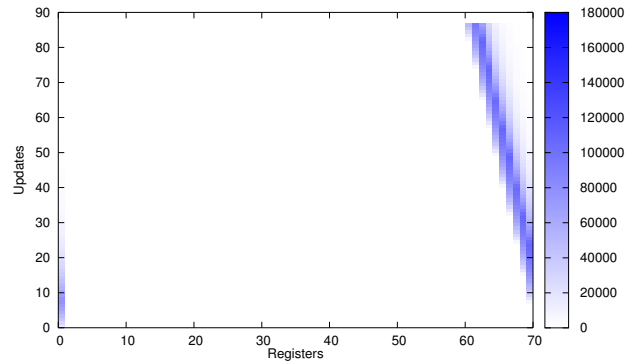


Figure 5.4: The period of the click-train is a *fraction of a unit smaller* than the delay of the comb-filter.

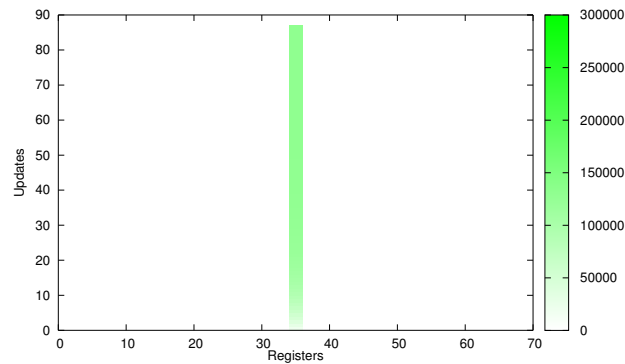


Figure 5.5: Comb-filter delay equals click-train period.

If the click-train's frequency cannot be accurately represented aliasing occurs and the comb-filter's state reflects this by showing steps in the movement of the peak. This step completes the prediction of the period of the click-train. The next step predicts the phase of the click-train to identify the input

signal.

5.4 Determining the phase

The position δ of the peak in the selected comb-filter registers indicates the phase of the click-train. The phase of the input signal is

$$\frac{\delta}{\lambda} \cdot (\text{Samples per second}) + \text{time associated with the first sample of the resonator's registers}$$

This formula maps the phase to a time relative to the start-time of the input-signal.

5.5 Evaluation of the selection function features

In order to select a set of features, I evaluate the features' performance detecting the tempo of click-trains. Ideally, the selection function has to overcome all of the issues described above, thus the selection function selects the resonator that most closely resembles the tempo of the click-train.

I examined two different types of pulse shapes. A pulse size of one indicates the ideal pulse identical to the click-train. A pulse size of four is a broader pulse but with a sharp onset. The pulse falls off slowly over the three consecutive samples (see Figure 5.6). I used two pulse shapes in order to see if the features have a better response to broader pulses.

I differentiate two types of click-train frequencies: aligned and unaligned frequencies. The resonant filter bank implementation reads in a signal with a sample rate of 210 Hz. Click-train signals that are aligned are not aliased. Their period is an integer multiple of $1/210$ s. Click-trains that are unaligned can have periods that may cause aliasing. At 210 Hz, some of these periods can not be exactly represented.

I ran tests examining four possible combinations:

1. Pulse size 1, aligned click-train frequencies
2. Pulse size 4, aligned click-train frequencies
3. Pulse size 1, unaligned click-train frequencies
4. Pulse size 4, unaligned click-train frequencies.

The results (see Table 5.1 and Table 5.2) show that all features work well, but not perfectly detecting high frequency and unaligned click-trains. At lower frequencies, the resonators detect frequencies

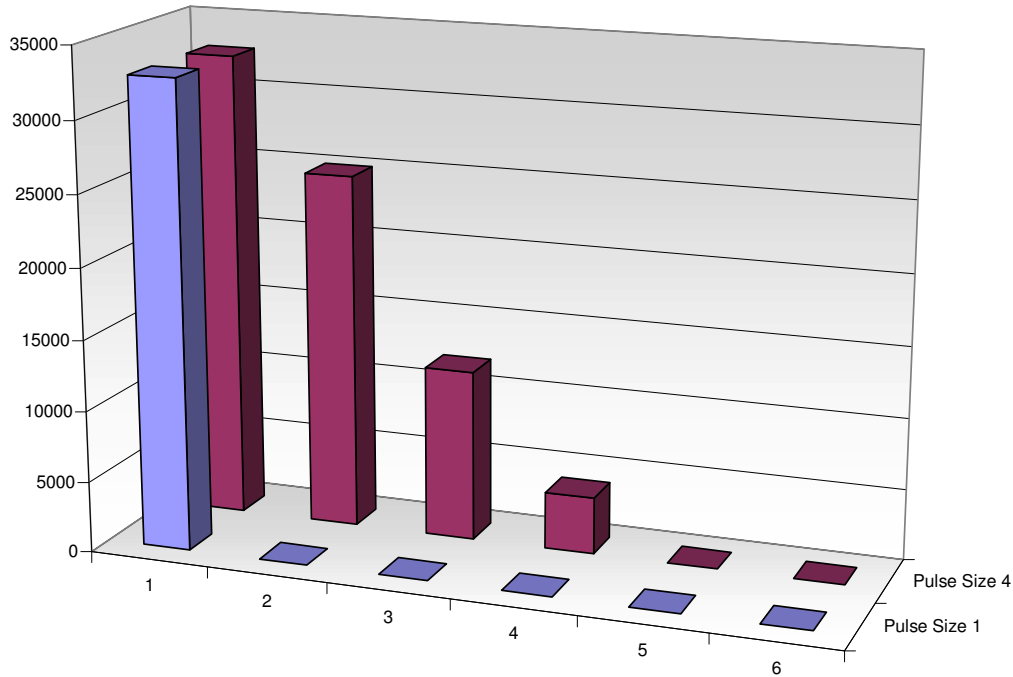


Figure 5.6: the different pulse shapes

Aligned	Pulse Size	Click-train periods	Shape	Energy	Shift
•	1	130	72 %	66 %	59 %
	1	199	77 %	62 %	72 %
•	4	130	68 %	75 %	59 %
	4	199	80 %	81 %	73 %

Table 5.1: The results of the evaluation of the features’ ability to detect different click-train periods. The columns “shape”, “energy” and “shift” show the percentage of the correctly detected click-train periods.

twice as fast as the click-trains frequency. There are also a few cases where other frequencies were detected. Increasing the size of the pulse helps quite a bit to get better results for unaligned click-trains. Not all possible resonant filters are implemented. Here, I used 99 comb-filters. Thus a resonator with a higher frequency might have more resonance than a resonator that is more appropriate, as shown in Section 5.2.

5.6 Evaluation of a linear combination of features

In this section, I constructed a selection function that linearly combines the shape feature and the energy feature and tried to find the optimal weights by enumeration. I choose an interval $[-10, 10]$

Aligned	Pulse Size	Click-train periods	Shape	Energy	Shift
•	1	130	75.7 BPM	75.0 BPM	85.2 BPM
	1	199	94.6 BPM	105.0 BPM	88.7 BPM
•	4	130	75.7 BPM	74.5 BPM	85.2 BPM
	4	199	85.5 BPM	95.9 BPM	86.5 BPM

Table 5.2: The table shows the average period where the features failed detecting click-train periods.

for each of the weights and enumerated points using a step-size of 0.3 which generates 4489 points.

I measure the accuracy of tempo estimation for each of these cases:

1. Pulse size 1, aligned click-train frequencies
2. Pulse size 4, aligned click-train frequencies
3. Pulse size 1, unaligned click-train frequencies
4. Pulse size 4, unaligned click-train frequencies

In order to reduce the computational requirements for this enumeration, I used only a few frequencies in each case. Thus for each point in the search space, I did several runs. In each run, a click-train period was chosen from the set of frequencies and analyzed by a comb-filter bank. The selection function was used to find the comb-filter whose period best represents the click-train's period. The root-mean-squared (RMS) error was used to summarize the errors over the used set of frequencies.

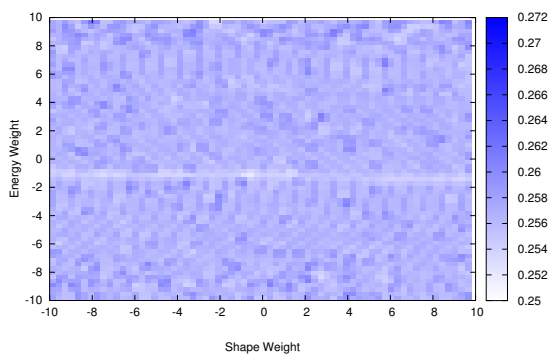


Figure 5.7: Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Unaligned

Unfortunately, Figures 5.7, 5.8, and 5.9 show that the search space does not have low peaks. There is no figure for the first test, since for all tested weight pairs the error was about 0.1833. Since this

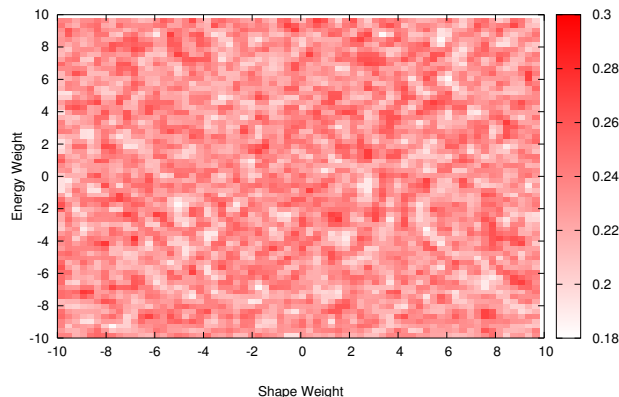


Figure 5.8: Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Aligned

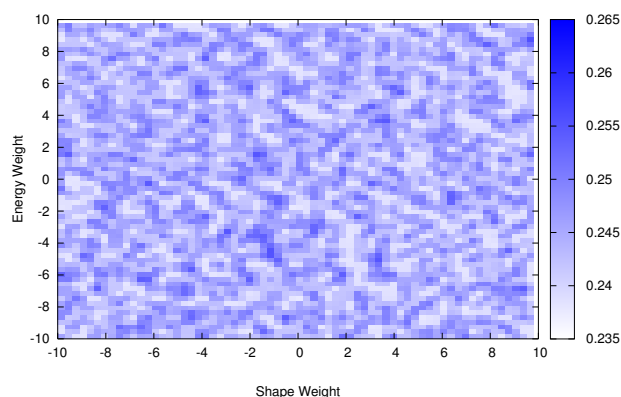


Figure 5.9: Evaluation of linear combinations measuring tempo accuracy: Pulse size=4, Unaligned

was a bit surprising, I investigated this case a bit further used a larger set of click-train periods to reevaluate a small number of points. It turned out that the error was different for different weight pairs. However, differences between errors were smaller than 0.01. Due to time constraints, I did not examine this case further. So probably no linear combination of these two features can yield a perfect tempo output. It is interesting that aligned frequencies yield a lower RMS error than the unaligned frequencies and this does not change with extend pulse sizes.

5.7 Evaluation of the tempo correction

The tempo-correction as presented in Section 5.3.1 was evaluated applying it to click-trains with a tempo of 60.0, 60.5, 61.0, ..., 160BPM. For each click-train, the comb-filter whose period best represents the tempo of click-train was selected. Without the correction enabled, the error is the difference of the comb-filter delay and the period of the click-train. When the correction is enabled, the error is the difference of the corrected comb-filter delay and the period of the click-train. The results are present in Figure 5.10. The corrected prediction works pretty well, whereas just using the closest tempo of the available comb-filter introduces quite a bit of an error. The root-mean-squared-error for the uncorrected approach is 0.3750 where the corrected approach causes only an error of 0.0095. However, this approach largely depends on the selection function choosing the best comb-filter. If the selection functions fails, the correction only slightly modifies a wrong tempo-prediction.

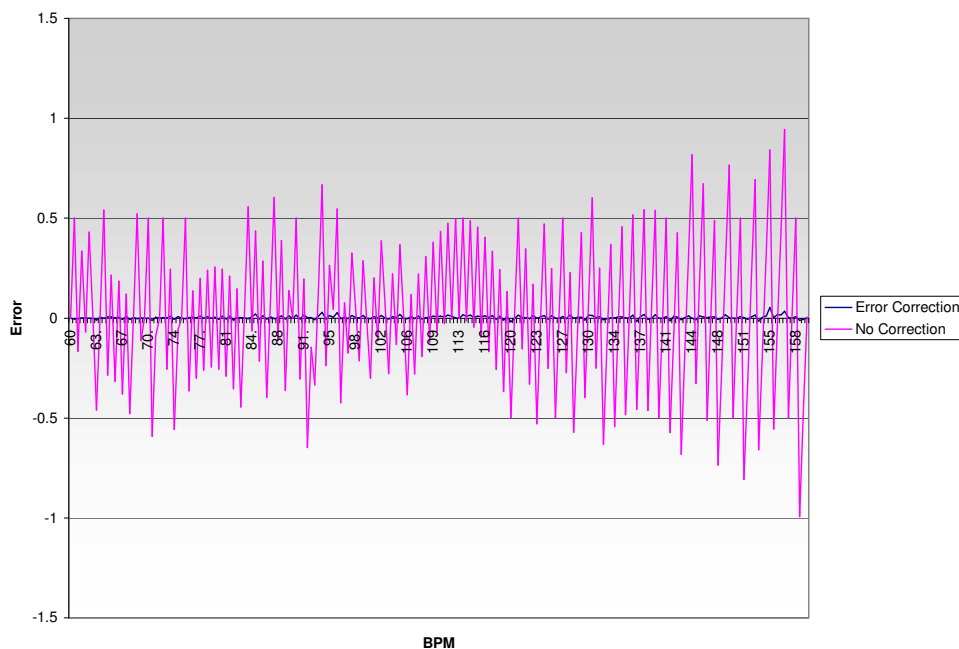


Figure 5.10: Comparison of Tempo-Prediction-Errors

5.8 Conclusion

The chapter discussed how to use comb-filters to detect the frequencies of a click-train. First, I presented a proof that comb-filters may have high-peaks, even though the frequencies of the click-train are not close to the resonance-frequency of the filter. Second, I used the concept of a selection function which examines a bank of comb-filters and selects the comb-filter that has the best resonance according to a feature or a linear combination of features. Then I discussed how to extract the phase of the beat and how to refine the prediction of the frequency. Finally, I verified my results empirically and showed that the refinement of the prediction is an advantage when used for click-train signals. The performance of the selection function was somewhat disappointing. Even at this level - a perfect click-train - the selection function is not capable of finding the right tempo in all cases. More work has to be done to get the detection with this approach right on this level, since the extension described in Chapter 6 depends on the resonant filter bank.

Chapter 6

BEAT-INDUCTION ARCHITECTURE FOR POLYPHONIC MUSIC

This chapter extends the approach to polyphonic digital audio signals. These signals are much more complex than the click-trains. The following sections describe how the audio signals are transformed so that they can be processed by a resonant filter bank. I propose a new method to determine the tempo from the comb-filters and how the tempo output is refined to get a smooth tempo output. I evaluate the approach using 14 songs. I will refer to the approach as Duden’s approach or Duden’s algorithm.

Figures 6.1 and 6.2 show the conceptual similarities and differences between both approaches. The first part discussed in Section 6.1 is almost identical to Scheirer’s (1998b) approach. Only the last step is omitted. Beginning with the resonant filter bank the system differences are noticeable and the final two steps try to make a tempo change smoother.

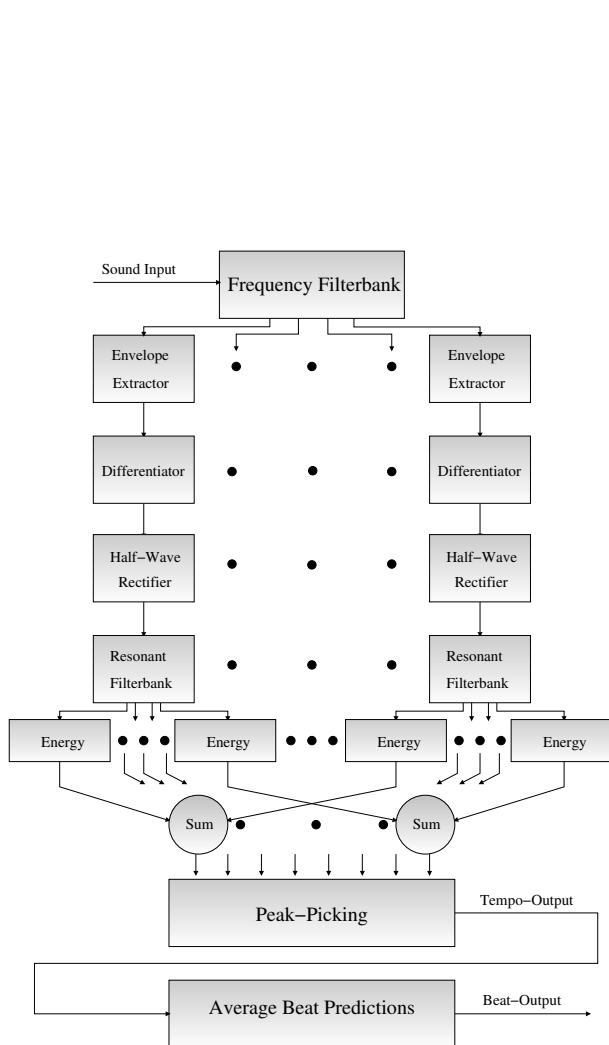


Figure 6.1: Scheirer’s processing algorithm

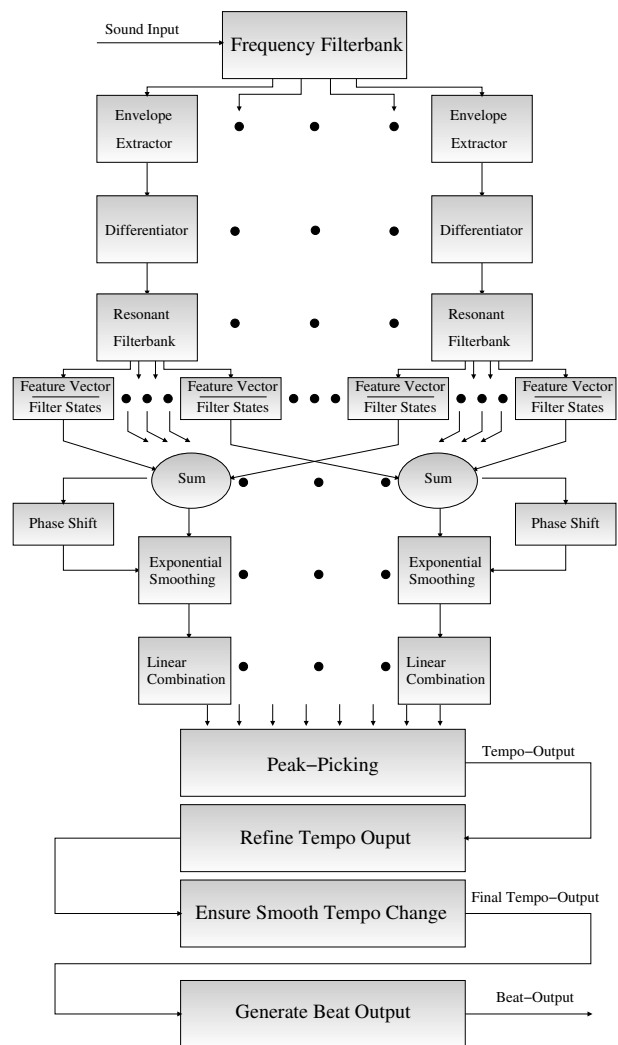


Figure 6.2: Duden’s processing algorithm

6.1 Preprocessing

The preprocessing step converts the polyphonic music signal into a signal similar to a click-train. This step is identical to the first steps of Scheirer's (1998b) algorithm. Figure 6.3 clarifies the process described next.

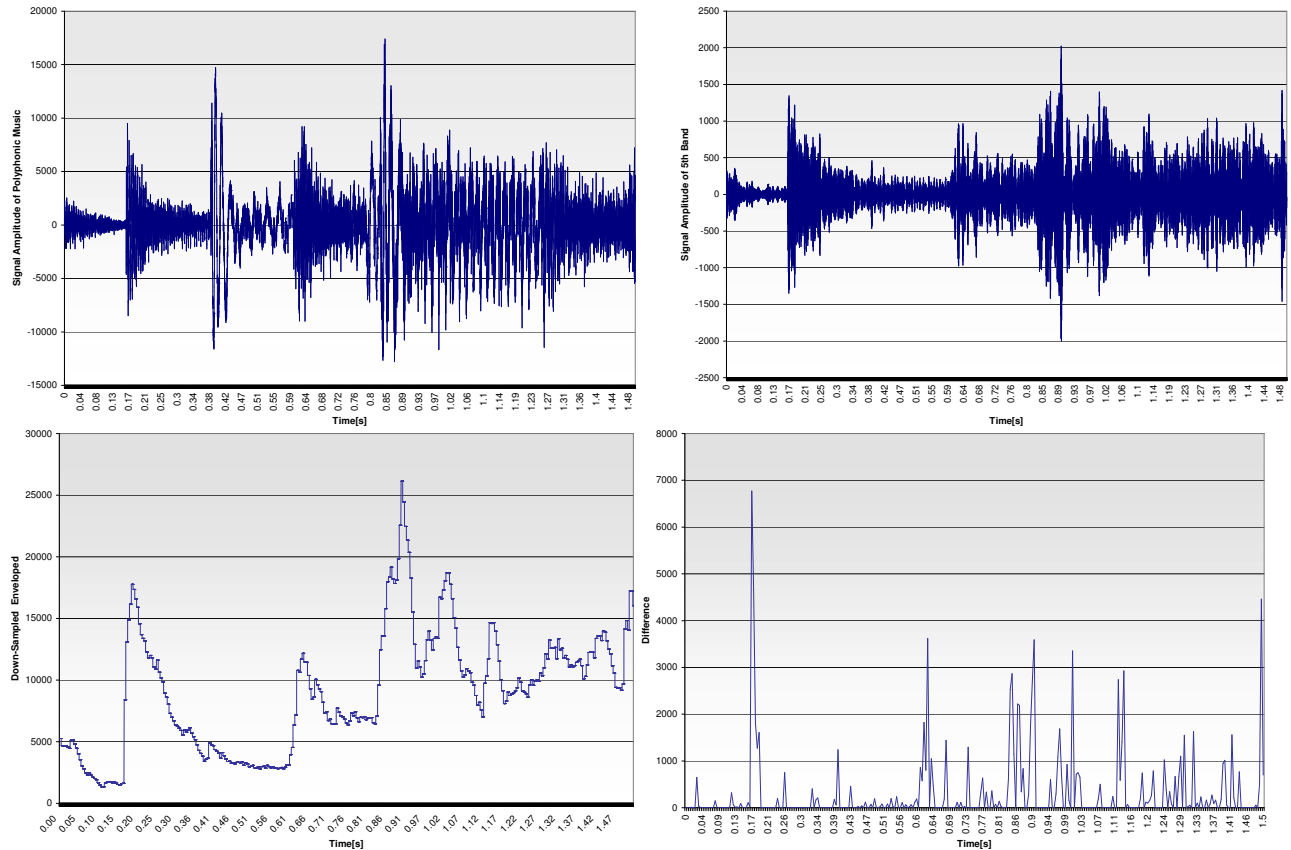


Figure 6.3: The figures show how the polyphonic signal is transformed in four steps. The result of this transformation is the input for the resonant filter banks.

6.1.1 Filter-bank

The filterbank splits the signal into six bands in the frequency domain. The lowest band carries frequencies 200 Hz and below. The other bands range from 200 Hz-400 Hz, 400 Hz-800 Hz, 800 Hz-1600 Hz, and 1600 Hz-3200 Hz. The sixth filter is a high-pass filter with a cut-off frequency of 3200 Hz. The band frequency ranges are chosen such that they split the spectrum approximately into octaves. As suggested in Scheirer (1998b), sixth-order elliptic filters are used to implement the filter-bank. These filters have sharp cut-offs to minimize the overlap between the bands. However, Scheirer (1999) points out that the sharp cut-offs might not be necessary.

6.1.2 Envelope Extraction and Down-Sampling

This step extracts the envelope from the signal for each band. A one-sided Hann window (Definition 2.2.11) is convolved with the output of the filter-bank. The filter reduces fast modulations and produces a signal that has the rough shape of the input signal (Scheirer 1998b). A similar smoothing process was discussed by Todd (1994) as a part of a model of the auditory system. The convoluted signal is down-sampled to 210 Hz. The low sample-rate allows fast processing, but it is high enough to represent the rough shape of the envelope.

6.1.3 Derivative

Computing the derivative detects onsets of the signal. It is easily implemented by sweeping over the sample buffer and subtracting the previous sample from the current sample. So the derivative is actually approximated through the difference. A cut-off at zero ensures that the process detects positive slopes only. High slopes represent sharp onsets as demonstrated in figure 6.3. As Scheirer (1998b) pointed out no thresholding or peak-picking is performed at this stage. Other approaches like (Seppänen 2001) use thresholding at this stage.

6.2 Resonant Filter-bank

The resonant filterbank works as described in Section 5.2. It analyzes the output of the preprocessing stage to detect the tempo. The state of the six resonant filter-banks is inspected by computing the features shape, phase-shift and energy that are presented in Section 5.3. For each comb-filter delay, the comb-filter states are combined across all bands by summing up their state-arrays. The combination results in 150 comb-filters states that are to be evaluated by the phase-shift feature. Whereas the shape and the energy feature are computed before the comb-filters are combined. The extracted feature values are combined instead.

6.3 Post-processing

The post-processing step tries to increase the regularity of the tempo-output. Since the selection-function may favor different resonant-filter in fast succession, the post-processing step uses an exponential smoothing filter to reduce such decisions. In addition to this smoothing an additional step smoothes the transition from one tempo and phase to another tempo and phase.

Note that this way of processing the comb-filter states is different from Scheirer's approach. In his approach, the comb-filter states are processed several times a second, usually between 10 Hz-20 Hz. Whereas this method extracts the features from the comb-filter states at between 2 Hz-3 Hz.

This processing is not done often enough to generate smooth output. Exponential smoothing is used to average the features over time. Therefore more stable feature values are used for the tempo estimation.

Exponential Smoothing The results of the feature evaluation are averaged for each comb-filter by

$$r := \alpha r + (1 - \alpha)n$$

where n is the new value and r is the old value.

Extracting the tempo Once the phase-shift and the shape are computed and processed by the smoothing filter the values of the features are simply added. The 'best' comb-filter is picked; that is the comb-filter with the lowest value. The next section continues with the output of the selection function which is the resonant-filter that is the best candidate to predict the beat.

Extracting the phase As described in Section 5.4, the phase is determined by picking the peak of the selected combined comb-filter state. A low-pass filter smoothes the combined comb-filter states before the peak position is determined making the peak-picking a bit more stable. Since the selected comb-filter is expected not to shift much, the phase output should be pretty stable. As opposed to the algorithm in (Scheirer 1998b), the a beat-time is estimated only once.

Refining the tempo output The estimated tempo is corrected using the technique presented in Section 5.3.1. The refinement should work well, since the phase does not shift much, and therefore the shift of the peak can be measured well.

Posting the beat-information to a container The frequency and the phase of the beat are posted to a container, that keeps track of a history of this information. It is important to note that the phase is specified relative to the start of the analyzed song. This enables the model to find applicable beat information for a given position in the song.

6.3.1 Generating smooth tempo transitions

In (Cambouropoulos, Dixon, Goebel, and Widmer. 2001), Cambouro et al. state that music listeners like to hear near-constant inter-beattime-intervals (IBI). The length of IBIs can vary a lot, since the beat-induction algorithm is not considered perfect. Even though the input is assumed to have a constant tempo, the algorithm might change its predicted tempo. This section proposes a method on how to make tempo changes 'smooth'.

The method Cambouro et al. propose averages the inter-beattime-intervals to generate a smooth sequence of beats. This method is not useful to smooth the output of the tempo prediction stage. The method is more applicable to smooth beats when the tempo is assumed to be constant and when changing the past beat-times is possible.

Scheirer's Algorithm (1998b) generates several estimates of a beat time. Beat times are averaged to a final estimate, only those beat-times lying within a certain margin refine the estimate. The method also does not take advantage of the additional information it could use: the tempo.

The methods described work on the beat-time level. On the contrary, the output generated by Duden's approach is a beat-time(the phase) and a frequency. Between two postings time can pass. The actual sequence of beat-times is generated using the beat-time and the frequency. Scheirer's method depends on several estimates per beat which are not provided by this process.

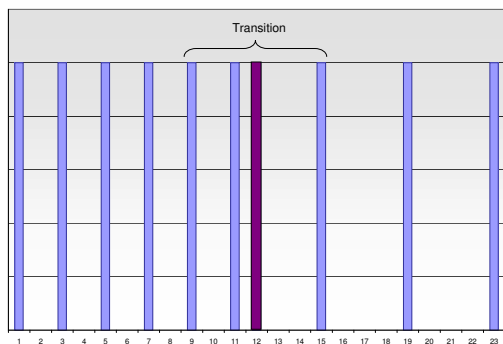


Figure 6.4: An example of a perfect transition to half of the tempo. The highlighted beat would be generated instead of the beat to its left side when averaging was used.

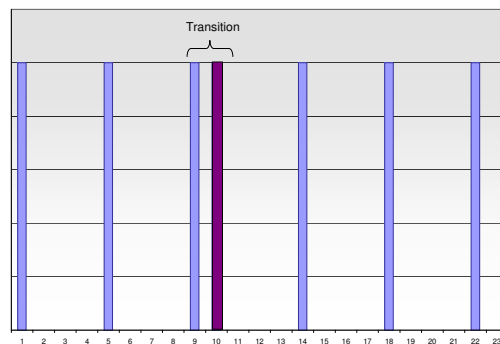


Figure 6.5: An example of an awkward transition. Skipping the highlighted beat would result in a much smoother transition

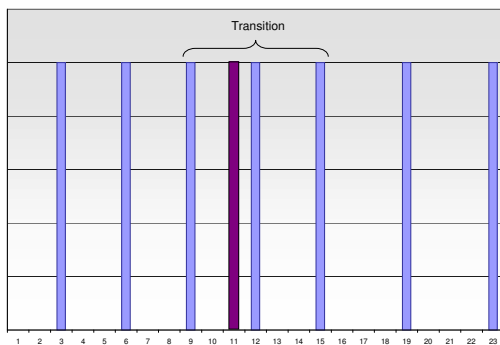


Figure 6.6: The highlighted beat would be generated when the tempo change was done immediately. A perfect transition is generated by waiting one beat before making the transition.

The following examples show problems which can occur at tempo transitions:

1. Assume that a transition has to be generated between two beat sequences where the second sequence's frequency is twice as large as the first sequence's frequency. Further assume, that both beat sequences have the same phase. The output that is desirable would be just the concatenation of the two sequences. No intermediate beats are necessary, since the two beats are in phase (Figure 6.4). Scheirer's Method should work in this case, since only beats occurring at

approximately the same time are averaged. Just averaging does not result in a good transition, since it would generate unnecessary intermediate beats, until the full tempo is reached.

2. A phase shift of 200 ms happens, but the beat maintains the tempo. Scheirer's ad-hoc method would not modify the beat time. An example: Assuming that the tempo is 120 BPM thus the IBI is 500 ms. The first beat-time of the new sequence appears as an artifact. Since the just generated IBI is 200 ms whereas the previous and future IBI are 500 ms (see Figure 6.5).
3. Figure 6.6 illustrates what can happen if the change of the tempo does not occur on a beat. In the example shown, it is better to delay the tempo change one beat to achieve a good transition.

Definition 6.3.1 (Transitional Beats): *Transitional beats are generated to make a smooth transition between two tempos.*

Definition 6.3.2 (Transition Interval): *The transition interval between two tempi is the last beat of the old tempo, transitional beats and the first beat of the new tempo.*

The approach described next tries to make transitions as smooth as possible. It is guided by the following guidelines:

1. The transitional IBI(s) should be as close as possible to the original IBI and the new IBI.
2. The transition interval is short - one beat at most.
3. It is not taken into account, whether the transitional beats coincide with beats in the analyzed song.

Let $f_1 = 1/\lambda_1, \phi_1 = 0$ be frequency and phase of the current beat, $f_2 = 1/\lambda_2, \phi_2$ frequency and phase of the next beat, and let N be the number of beats used as beats in the transition phase. The algorithm has to determine N beat times b_1, \dots, b_n where b_1 is the last beat of the current beat and b_N the first beat of the new beat. The algorithm below uses one transitional beat and $3 \leq N \leq 4$. Even though up to 4 beats are generated, only one transitional beat is used.

The algorithm uses a fixed beat b_1 and then generates the beat b_2, b_3 and in some cases b_4 :

$$\begin{aligned}
 \omega &= 0.85 \\
 \lambda_{\text{avg}} &= \frac{\lambda_1 + \lambda_2}{2} \\
 k &= \left\lceil \frac{b_1 + \omega \min(\lambda_{\text{avg}}, \lambda_2) - \phi_2}{\lambda_2} \right\rceil \\
 b_1 &\quad \text{determined by input} \\
 b_2 &= b_1 + \lambda_{\text{avg}} \\
 b_3 &= k\lambda_2 + \phi_2 \\
 \text{if } b_3 - b_2 > \lambda_{\text{avg}} &\text{ then} \\
 b_2 &= b_1 + \lambda_1 \\
 b_3 &= b_2 + \lambda_{\text{avg}} \\
 b_4 &= k\lambda_2 + \phi_2 \\
 \text{endif.}
 \end{aligned}$$

The algorithm starts by placing the beats b_1 and b_2 . The IBI of b_1 and b_2 is the average period of λ_1 and λ_2 . The right time to start with the new beat is determined by k . The parameter k is computed so that the new beat b_3 starts when the IBI to b_2 is at least $\omega \min(\lambda_{avg}, \lambda_2)$. It ensures that the IBI is not drastically smaller or larger than λ_{avg} and λ_2 . However, if the IBI is larger than λ_{avg} the transition interval is started one beat later to reduce the IBI. The algorithm does not solve all problems that arise since it generates one transitional beat by default, even though that might not be necessary.

The complete system consists of the preprocessing step which produces signals that look more like click-tracks. The resonant filterbank analyzes these signals, extracts tempo information from it and presents a prediction of tempo and phase to the post-processing part which favors only results that have been consistent for a while. Once a definite decision for the tempo and phase is done, it is ensured that tempo transitions are done in a smooth way.

6.4 Evaluation

In this section, I compare Duden's approach with Scheirer's approach. I use three different measures to compare the performance of these approaches.

6.4.1 Evaluation Measures

Three measures are important for a beat-tracking algorithm to excel:

1. Regularity
2. Beat placements
3. Tempo.

Dixon discusses in (Cambouropoulos, Dixon, Goebel, and Widmer. 2001) that humans prefer regular beat patterns over more accurate patterns. The regularity criterion is necessary to measure how consistently the beats were placed. Since the thesis concentrates on musical pieces with a constant beat, regularity is good measure.

The inspection of the times of beats will deliver the measure. Let b_i be the time of the i -th beat for $1 \leq i \leq n$, then

$$\mathbf{regularity} \equiv \sqrt{\frac{1}{n-2} \sum_{i=3}^n \left(\frac{(b_i - b_{i-1}) - (b_{i-1} - b_{i-2})}{b_{i-1} - b_{i-2}} \right)^2}.$$

The squared relative deviation of inter-beat-intervals to their previous inter-beat-intervals measures how well the beat tempo was preserved compared to the immediate predecessor. The square-root of

the average squared deviation summarizes this idea over the complete musical piece. Scheirer (1998b) uses a similar measure, but the deviation is not normalized, therefore the measure allows at fast tempi a relatively higher deviation than at slower tempi whereas the proposed measure will allow small deviations at high tempi and larger deviation at slower tempi.

Three methods are discussed that measure the accuracy of the beat-placement. All of these methods assume an estimated sequence of beat-time denoted with b_1, \dots, b_n and a reference sequence r_1, \dots, r_m .

The beat-time-accuracy is measured as proposed in Scheirer (1998b) using root-mean-square deviations. The distances of the beat to the closest ideal beat are squared and summed up. Taking the square-root of the sum retrieves accuracy. The tempo of the ideal beat is doubled thus the predicted beat can also match the mid-point between two beats. This allows to have a perfect measure if the beat is twice as fast as the solution or if the beat's phase is off-beat.

$$\mathbf{accbeattime}_1 \equiv \sqrt{\frac{1}{n} \sum_{i=1}^n \min_j \left\{ \min \left(b_i - r_j, b_i - \frac{r_j + r_{j+1}}{2} \right) \right\}^2}$$

Goto et al. (1997) presented a similar approach to measure the beat-placements. They suggest a simple measure that first matches the estimated beat-times with the closest reference beat-time and then they compute the relative deviation of the estimated beat-times to the matched references beat-times. The third step determines the longest correctly tracked sequence of beat-times. A sequence is correctly tracked when all relative errors are below a threshold. They compute the mean, standard deviation and the maximum error using this specific sequence. An advanced measure build on the latter measure was proposed in (Goto and Muraoka 1997) as well. It takes half-tempo, double-tempo and phase errors into account.

Cemgil et al. (2000) evaluated their beat-tracking system's performance by applying a Gaussian window to the deviations. The windowing function heavily discounts beat-placements outside a 40 ms tolerance.

$$\mathbf{accbeattime}_2 \equiv \frac{100}{(n+m)/2} \sum_i \max_j \{W(b_i - r_j)\}$$

$$W(x) \equiv \exp\left(\frac{-d^2}{2\sigma_e^2}\right) \text{ where } \sigma_e = 40 \text{ ms}$$

The measure emphasizes in a non-linear manner that beats should be close to their reference beats. More than the previous measures of Scheirer and Goto et al., this measure emphasizes the accuracy of the beat times. Beat-sequences having the correct tempo and a phase error greater than 40 ms, yield a fairly low score with Cemgil et al.'s measure, while Scheirer's and Goto's measure will still yield a good score. A perfect match of the estimated beat-sequence with the reference beat-sequence yields a score of 100. Since I will determine the accuracy of the tempo with the measure below, I will use the $\mathbf{accbeattime}_2$ measure.

Algorithm(Feature used)	regularity	acctempo	accbeattime ₂
Scheirer	3.45	47.50	52
Duden(Shift)	1.36	36.13	48
Duden(Shape)	1.30	26.63	50
Duden(Energy)	1.17	51.88	28

Table 6.1: Results of the evaluation. Note that the **regularity** and **acctempo** measure errors (lower is better) whereas **accbeattime₂** measures correctness (100 is best and 0 is worst)

The accuracy of the tempo is how well the algorithm detects the tempo of the musical piece. First, the tempo is determined by computing the average IBI Θ of the reference sequence. The mean squared relative error of the estimated IBI according to Θ determines the value of the measure, thus

$$\text{acctempo} \equiv \sqrt{\frac{1}{n-1} \sum_{i=1}^n \left(\frac{b_i - b_{i-1} - \Theta}{\Theta} \right)^2}.$$

The measure is similar to variance of the IBIs discussed in (Scheirer 2000).

6.4.2 Methodology and Test-Data

The test-data were MP3-files that were downloaded from mp3.com (MP3.com 2002). I selected 14 MP3-files and use them to compare the two approaches. Each algorithm analyzed the first 90 seconds of each song. In order to create reference beat-times for each song, I wrote a program that plays back MP3-files. By tapping on the keyboard, the program generates a reference beat-list. I tapped to each song three times. Beat times that were close to each other are merged by taking the average of the beat times. Afterwards, the list of beat-times was corrected by hand. Having references, Scheirer's and Duden's approach were evaluated using the three measures defined above on 14 songs.

I modified Scheirer's default parameter settings, so that the algorithm uses a comb-filter-bank of 150 filters. With these settings, I let Scheirer's algorithm analyze the test songs. These were converted to 22 kHz to have a fair comparison, since the Duden's approach also analyzes the songs at 22 kHz. The implementation presented was compiled without the optimizations discussed in Chapter 7. I evaluated the Duden's approach for each of the three features discussed in Section 5.3.

6.4.3 Test Results

The tests presented in Table 6.1 show that Scheirer's algorithm generates more accurate beat-times. However, Duden's algorithm using the shape feature is close to the accuracy of Scheirer's algorithm. The algorithm's performance using the shift feature is not as good as the algorithm's performance with the shape feature. The energy feature does not seem to be an adequate feature to maximize **accbeattime₂**. This is quite surprising, since Scheirer's approach uses this feature and performs best.

All variants of Duden's approach show a fairly regular beat sequence output. Whereas Scheirer's score is more than two times higher, thus Duden's approach improved the regularity quite a bit. Inspecting the output of Scheirer's implementation shows that IBI can greatly vary. Whereas the IBIs generated by Duden's approach are more regular.

Even though Scheirer's approach places the beats more accurately, the IBIs it generates do not provide a good feeling of the tempo of the analyzed musical pieces. Duden's algorithm using the shape feature achieve the best score of all variants.

The tempo-correction, which was introduced in Section 5.3.1, does not have much influence on the results. It seems that the errors introduced by the rest of the system, overshadow the small correction.

The variant using the shape feature generates the best results in the first two measures. The third score is close to Scheirer's score. However, in Chapter 7 I present a variant of the proposed algorithm that is best in all categories.

Unfortunately, the number of test-cases is limited, because there is no large music corpus available. One reason is that beat-tracking algorithms were evaluated with different goals in mind (Dixon 2001b). The large music corpus that Seppänen created for his master's thesis (Seppänen 2001) was not accessible due to copyright restrictions.

6.5 Conclusion

The chapter introduced an extended model to Scheirer's approach. I described the Duden's model and Scheirer's model. The two models are nearly identical for the first processing steps, following processing steps are similar but still different. The differences aim for a faster processing time and regular beat-time output. The evaluation shows that Scheirer's algorithm output is more accurate, but not as regular as the output of the variants of Duden's algorithm presented in this thesis. Also, the inter-beat-intervals generated are closer to the average inter-beat-interval of the reference beat-times.

Chapter 7

AN OPTIMIZED IMPLEMENTATION

This part of the text deals with the C++ implementation of the presented beat-induction architecture. First, I show how I designed the classes to implement the architecture and discuss important issues. In the second section, I examine different optimizations to improve the efficiency of the architecture. It turns out, that one optimization actually improves the quality.

7.1 The beat-induction architecture in C++

The architecture consists mainly of three classes *CBeatInduction*, *CBandPath*, and *CCombFilter*. Figure 7.1 shows the parts the classes implement. The following subsections describe the implementation of processing elements in order of the data-flow.

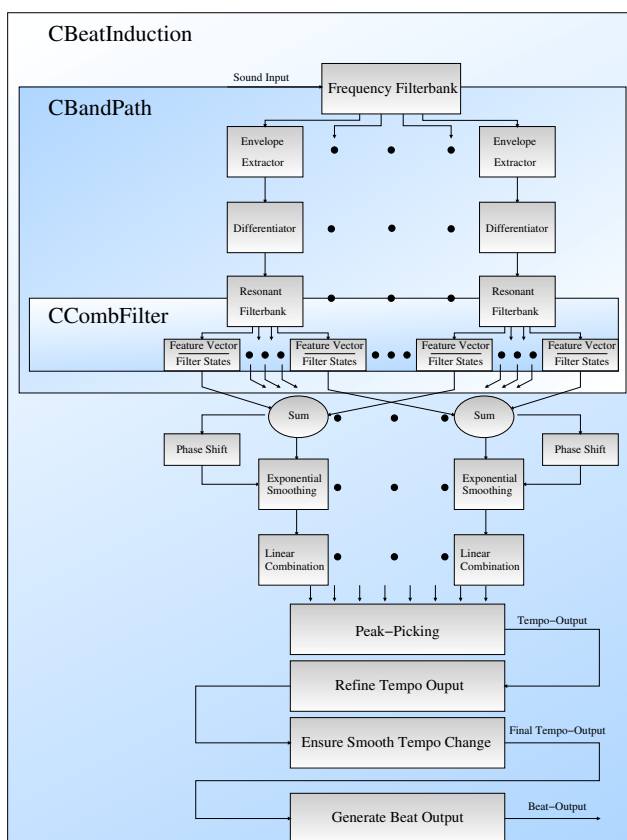


Figure 7.1: Implementation

7.1.1 Beat-Induction Class (1)

The Beat-Induction class presents the public interface to the client software. A *CBeatInduction* object creates its own thread to receive sound data. The implementation is thread-safe, thus while the object is accepting sound data, it can still provide beat time output (Figure 7.2). A second thread produces the sound data and reads the beat-output of the first thread. However, I evaluated the beat-induction's performance using a non-threaded version (see Section 7.6), which helps to correctly estimate the efficiency of the algorithm.

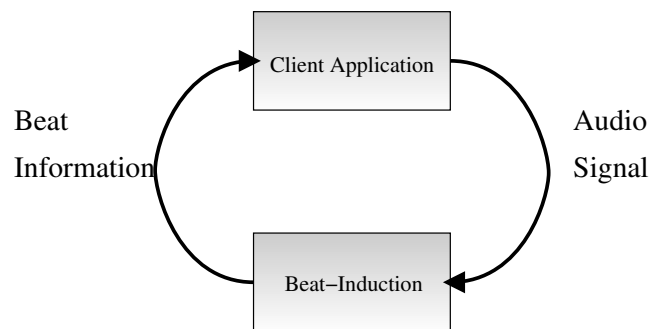


Figure 7.2: Data Flow between Client Application and the Beat-Induction Architecture

The client software provides a stream of music data by calling the function *OnNewSampleData* which takes any PCM format and converts it into the internal format (16 Bit, mono at 10080 Hz or 22050 Hz). The original conversion routines were provided by Hauke Duden and modified to fit the needs of the beat-induction. Converted sample data is written into one of a chain of buffers. A semaphore signals when a buffer is full. The data is then read by the six *CBandPath* objects that represent the six bands.

The buffer approach is used through-out the architecture. Different steps of the algorithm read their data from a buffer and write them to a buffer. Thus the output is always processed in chunks of samples. Samples are never processed separately.

7.1.2 BandPath Class

CBandPath uses filter-methods provided by the Intel Performance Library. The filtered samples are passed to the halfwave-rectifier (Section 6.1.2) which computes the envelope of the signal and reduces the sample to 210 Hz. The differentiate method computes the derivative. The resulting signal is then passed to each of the 99 or 150 comb-filters.

7.1.3 Comb-Filter Class

The comb-filter implemented in a similar way to the outline in Figure 2.4. The input is buffered, so a change of the actual comb-filter state will only be done if enough samples are present such that all registers are updated at once. The number of filter updates is counted. The comb-filter provides methods that compute the energy and shape measure. It also provides access to the state buffer.

7.1.4 Beat-Induction Class (2)

About every 500 ms, the post-processing stage extracts the features from the comb-filter-states. The computation of the features phase and tempo is straight forward. They are computed as described in Section 6.2. The phase and tempo are posted via the *PostBeatEstimate* method to a circular buffer. The algorithm for generating smooth tempo transitions (Section 6.3.1) is applied and the final beat-information as well as the transitional beats are written into slots of the buffer.

When *TimeUntilNextBeat* is called the method searches the buffer for the beat-information that applies and extrapolates - if necessary - to return the time until the next beat occurs. It also returns in an optional variable the estimated tempo, thus the caller can generate beat sequences on its own.

7.2 Test-Setup

The test-setup for the following performance evaluations was a Pentium III processor at 500 MHz, 512 KB secondary cache, 196 MB SDRAM on Windows 2000 SP2. I used the Visual C++ 6.0 SP5 compiler and compiled the source code with speed optimizations turned on. The Intel Performance Library that was linked is optimized for Pentium II and Pentium Pro machines. The execution times were determined by averaging the run-time of 10 runs. The variances of the results was less than 0.02 in all test-cases. The fraction of the execution time was determined using the Visual C++ profiler.

7.3 Optimizations

The analysis of bottlenecks allows to improve the performance of the system by concentrating on those parts that have the largest impact on the system performance. I used a profiler to measure the time the cpu spent executing functions of the program. One iteration consists of making the bottle-necks visible, changing the behavior of the the implementation, and measuring the resulting speed-up.

7.3.1 First Iteration (Sample rate reduced)

Original version of the beat-induction needs for a 90 second MP3 -file (44.1 KHz, 16 Bit, stereo) on average 15.4 seconds to analyze it, which implies a CPU usage of 17%. The results of profiling

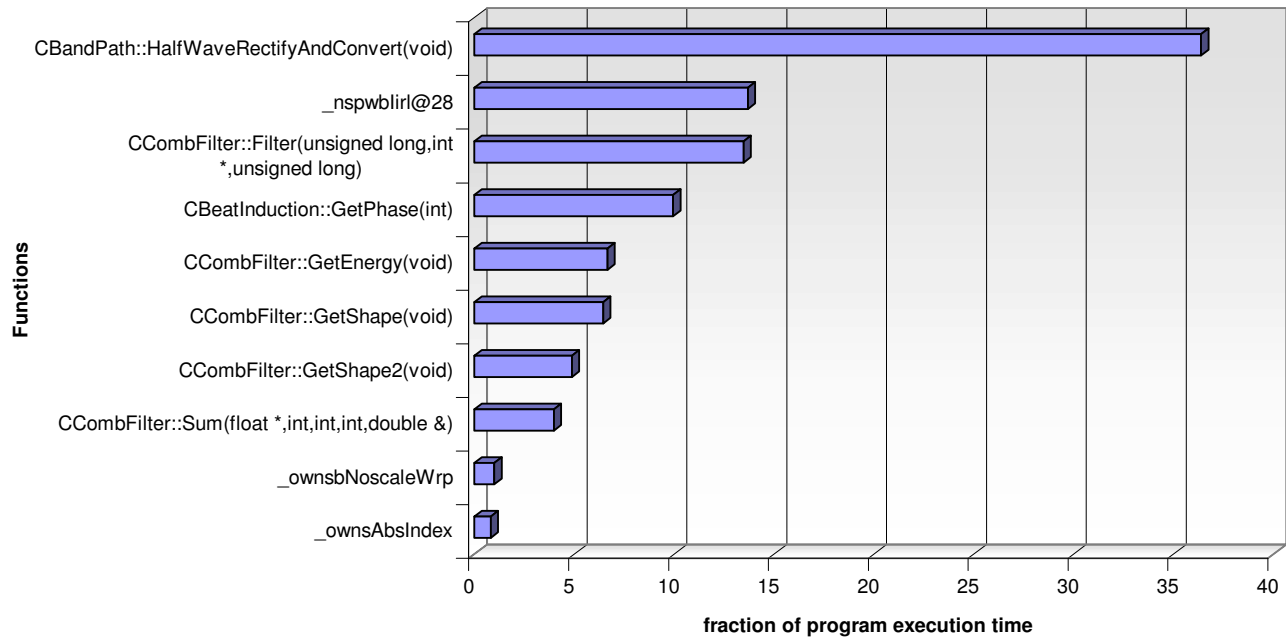


Figure 7.3: Bottle-neck analysis with no optimizations enabled

the application in Figure 7.3 show that the halfwave-rectify and conversion step (Section 6.1.2) is the bottleneck of the architecture. The function `_nspwblirl@28` performs the filtering described in Section 6.1.1 is the second bottle-neck. Reducing the sample rate from 22050 Hz to 10080 Hz, will reduced both bottle-necks, since both functions work on signals having a sample-rate of 22050 Hz. A speed-up of $1/0.46 \cdot (0.36 + 0.14) + 0.5 = 1.37$ should be generated by this reduction. Thus the execution time should be reduced to about 11.2 seconds. Indeed, the new version spends 11.1 seconds (12.3% CPU usage) analyzing the MP3-file, which is slightly faster than expected.

7.3.2 Second Iteration (Number of comb-filters reduced)

As presented in Figure 7.4, the filter-bank code is not a major bottle-neck anymore, whereas the halfwave-rectify function still takes a major part of the execution time. A better convolution algorithm might help, but due to time constraints I did not investigate in that direction. I decided to lower the execution time needed by `CCombFilter::Filter`. Lowering the number of comb-filters from 150 to 99 per band serves this goal. Naively, the speed up is $1/0.66 \cdot 0.18 + 0.82 = 1.07$ the resulting execution time should be 10.5 seconds, but the empirical results show a higher speed up. This version only has only a CPU utilization of 8.6% (7.7 seconds). The higher speed-up of 1.3 can be explained that not

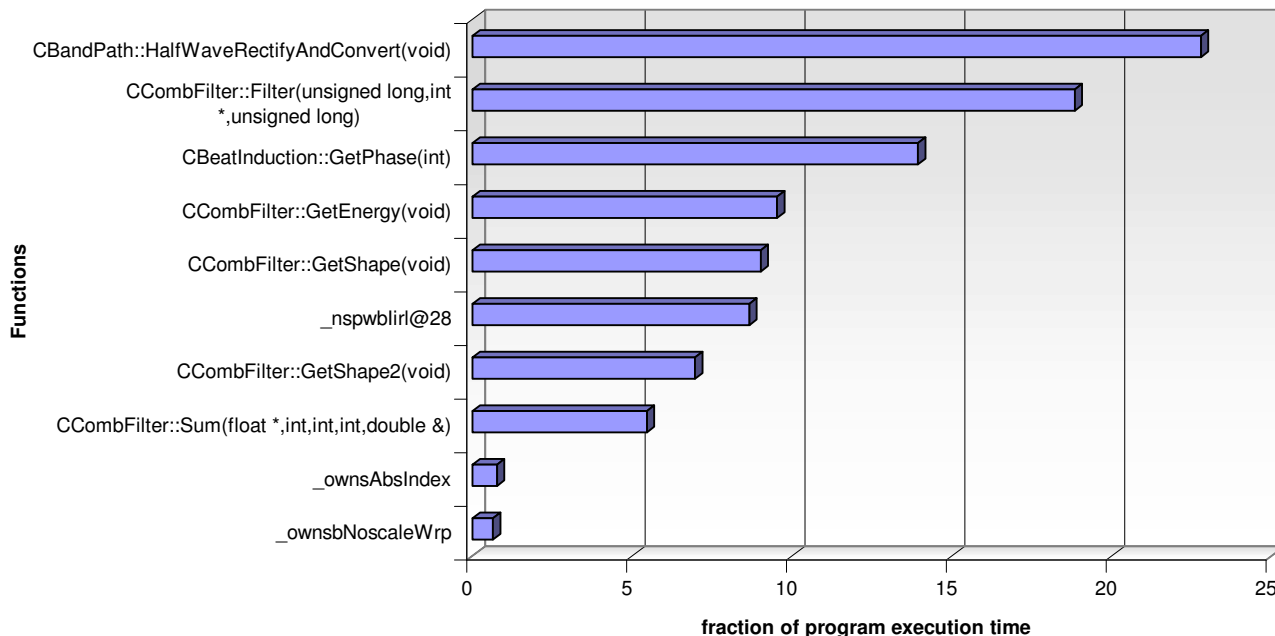


Figure 7.4: Reduced sample rate

only `CCombFilter::Filter` processed less data but also other methods like `CBeatInduction::GetPhase` and `CCombFilter::GetEnergy`. These methods will be not only called less frequently, but also the amount of data they process is reduced more than proportionally. It is more expensive to apply these methods to Comb-filters with a high delay. Eliminating Comb-Filters especially with a high delay results in a high speed-up.

7.3.3 Third Iteration (Phase computation optimized)

Figure 7.5 shows that `CBeatInduction::HalfwaveRectifyAndConvert` and `CCombFilter::Filter` have the largest impact on the program's performance. However, at this points optimizing these functions is difficult. I optimized `CBeatInduction::GetPhase` instead. The method determines the phase of a comb-filter across all six bands by picking the peak after smoothing the state with a hamming window (Scheirer 1998a). I eliminated the smoothing process, with the hope that it does not help to the accuracy of the algorithm a lot. The measured program execution time is 7.1 seconds (8%) which is a speed-up of 1.07.

7.3.4 Fourth Iteration (on-the-fly optimization)

`CCombFilter::GetEnergy`, `CCombFilter::GetShape` both use the sum of the squared values in the state array. It is not efficient to recompute it for each function-call. not efficient. I moved the computation

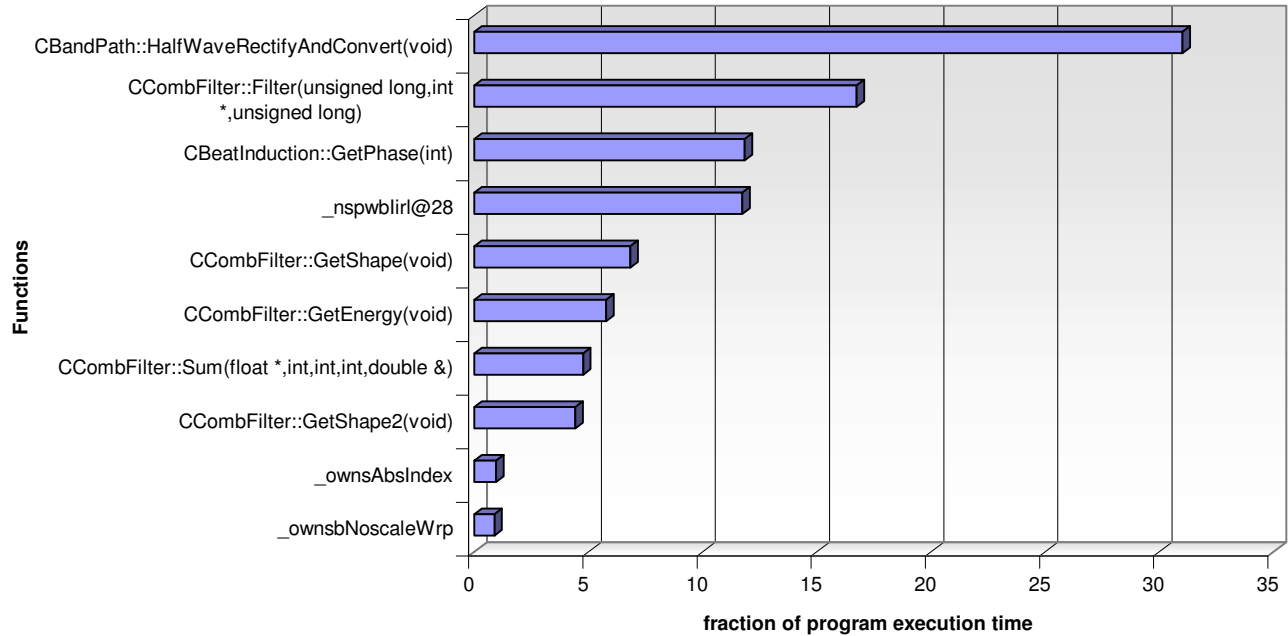


Figure 7.5: Reduced sample rate and 99 comb-filters

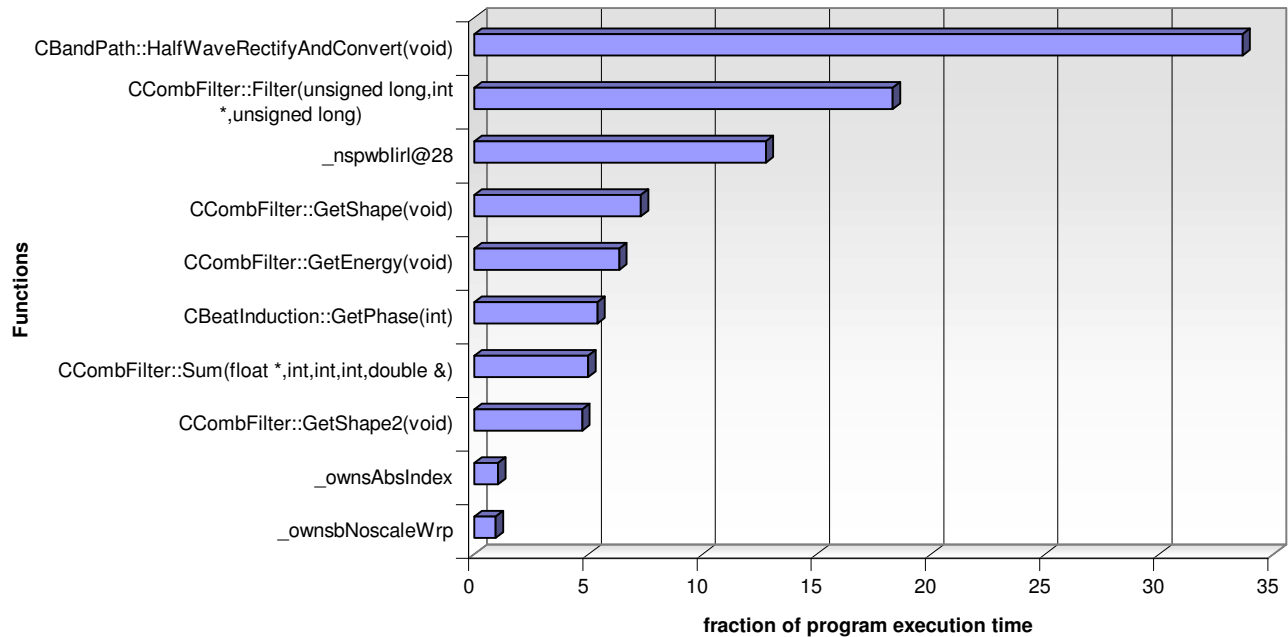


Figure 7.6: Smoothing in GetPhase eliminated

of the sum into *CCombFilter::Filter*. While computing the new state values, the method now also computes the sum of the squared values and the peak-position and peak-value. This should reduce memory accesses, since computing these values in the filter method maximizes cache hits. I ex-

amined two variants. One variant is accurate and the output is unchanged when accurate on-the-fly optimization is enabled. With that variant enabled the execution time was lowered to 6.4 seconds which is a speed-up of 1.1. It extracts the features for every update of the state array. The other variant omits to recompute the features in some cases, and this optimization resulted in a speed-up of 1.2 or 5.9 seconds.

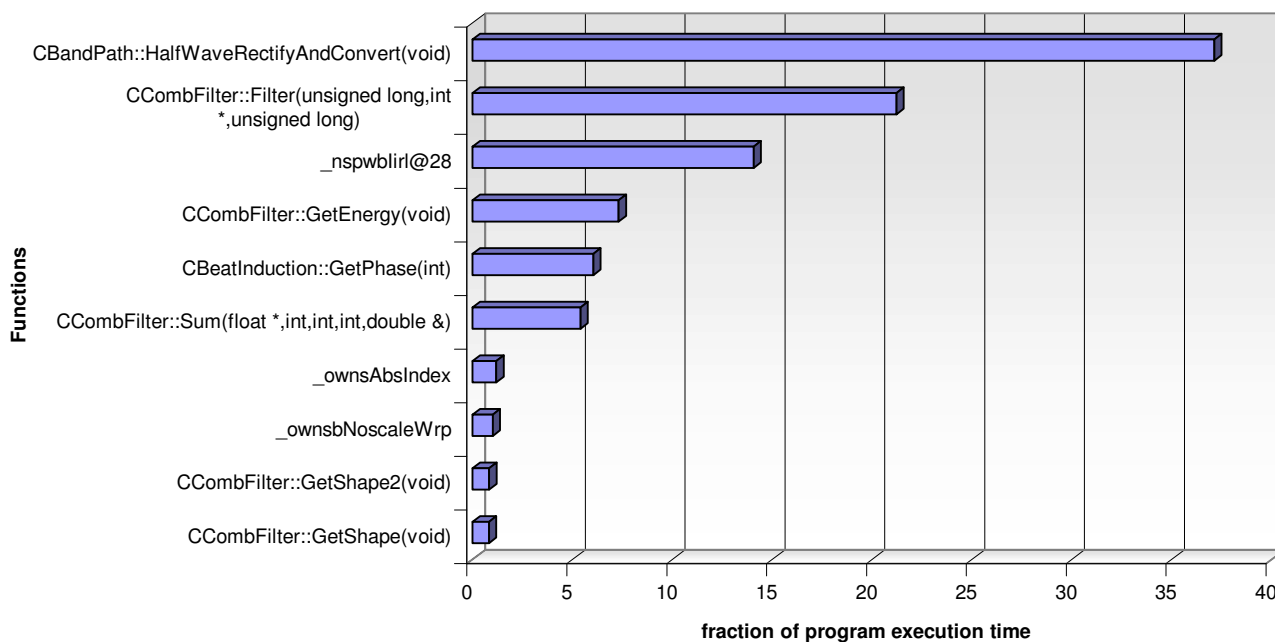


Figure 7.7: After applying all optimizations

The total speed-up of these optimizations compared to the original version is about 2.5. The CPU-utilization of the final version is 6.6 %.

7.3.5 Influence of the optimizations on the quality

Table 7.1 shows the impact of the optimization discussed on the quality of the algorithms accuracy. An important result is that *enabling the phase optimization is not only an performance improvement but also a quality improvement*. Furthermore, the quality considerably degrades when the sample rate is reduced. Configuration 7 has the best relation of performance and quality, since it just needs 5.98 seconds to analyze a 90 second sound-file.

#	SR	# Comb.	P. opt.	Acc. OTF	OTF	regularity	acctempo	accbeattime ₂
1	22.5 kHz	150				1.30	26.63	50.08
2	22.5 kHz	100				1.44	31.64	50.78
3	10.1 kHz	150				1.61	28.20	38.81
4	22.5 kHz	150	•			1.47	29.56	52.52
5	22.5 kHz	150			•	2.17	35.00	39.36
6	22.5 kHz	150		•		1.30	26.63	50.08
7	22.5 kHz	100	•	•		1.49	31.36	52.19
8	10.1 kHz	100	•	•		1.34	29.75	45.66

Table 7.1: Optimizations and their influence on the accuracy is shown in this table. Columns are configuration number, sample rate(SR), number of comb-filters, optimized phase computation (P. opt), accurate on-the-fly optimization (Acc. OTF), on-the-fly-optimization (OTF), regularity measure, tempo accuracy and accuracy of beat-times.

Sample Rate	Scheirer	Duden	Speed-Up
22.5 kHz	42.4 s	15.4 s	2.75
10.1 kHz ¹	35.7 s	11.1 s	3.22
10.1 kHz ²	35.7 s	6.0 s	5.95

Table 7.2: The table presents how much execution time both algorithms need to analyze 90 s of data. The column sample rate states, which sample rate the system used. The Scheirer and Duden column specify the execution time and the Speed-up column the speed-up Duden’s algorithm achieved compare to Scheirer’s algorithm.

7.4 Performance Comparison with Scheirer’s Approach

I ran Scheirer’s implementation of his algorithm on the machine described in Section 7.2. I used the same compiler and optimizations settings and the author’s default program settings. Scheirer’s implementation uses 100 comb-filters for each band, whereas the first two versions of my implementation use 150 comb-filters. The program execution times are listed in Table 7.2, Scheirer’s program was given an audio signal of 90 seconds at different sample rates. The algorithm’s execution time varies much, since the implementation does not convert the signal to a fixed sample rate before filtering it.

The implementation presented here, first converts the signal to the listed sample rates and then analyzes the signal. This maximizes the independence of the performance from the sample-rate. The trade-off is that the algorithm does not utilize the quality of signals with high samples rates.

In order to compare the performance of the algorithms, the first column in Table 7.2 specifies the sample-rate of the signal that is analyzed by the filter-banks of both approaches. The third rows shows the full speed-up when all optimizations are enabled.

¹ version presented in Section 7.3.1

² fully optimized version (Section 7.3.4)

7.4.1 Future Work

The method `CBandPath::HalfwaveRectifyAndConvert` should be the target of future optimizations. More sophisticated convolution algorithms might result in an acceptable speed-up. Optimizing the filter-bank as proposed in (Scheirer 1998b), by using more efficient filter implementations might result in a small speed-up.

7.5 Visualizer using the Beat-Induction Model

I implemented a visualizer plug-in for the Ashampoo Media Player. The plug-in uses the beat-induction algorithm to analyze the music that the media player is playing and shows a dot indicating a beat. It also displays the estimated BPM of the musical piece. I listened to MP3-files using this visualizer plug-in enabled and it turns out that in many cases is the visualizer not on-beat. However, this is only a subjective and fuzzy impression. I think that the beat-induction model needs to be improved, so that it can cope with all kinds of music.



Figure 7.8: BeatVis Visualizer and Ashampoo Media Player

7.6 Synchronous Version

I implemented an asynchronous version, which I used for the visualizer plug-in. The asynchronous version creates two threads (see Section 7.1.1). One thread produces the sound-data and reads the beat-output, the second thread analyses the sound-data and generates the beat-output.

I implemented variants of the beat-induction model that work synchronously. The same thread that produces the sound data analyzes it. This implementation was used to measure the implementation's efficiency and accuracy. It was also used to produce the evaluations shown in Chapter 5. The application is able to read in MP3-files using the MPG123 library ([Hipp 2001](#)) and prints a sequence of beat-times into a file.

7.7 Conclusion

I presented a short overview of how I implemented the concept of the beat-induction architecture. Since the focus of this thesis is not only the quality of the algorithm's output, but also the speed of the computation, I examined different optimizations. One optimization has no impact on the accuracy. Applying one of the other optimizations implies a lower quality of the output. However using the shape feature, one optimization not only improved the speed, but also improved the quality. I compared the performance of the presented approach with Scheirer's implementation and, depending on the enabled optimizations, speed-ups between 3 and nearly 6 were achieved. Briefly, I discussed the sample real-time application, which shows a point blinking on beat.

Chapter 8

CONCLUSION

In this thesis, I discussed a modified beat-induction model. The modifications make the model more applicable for real-time applications, such as synchronizing computer graphics to previously unknown music.

8.1 Contributions

I presented the following contributions:

1. A better way to estimate a click-train's period with a comb-filter-bank.
2. I showed, how the comb-filter approach can be mislead when detecting click-train periods.
3. A modified architecture was presented, that achieves speed-ups of 3 to 6 over the approach presented in (Scheirer 1998b).
4. Duden's architecture generates more regular beat-output than Scheirer's implementation. Also, the beat-output resembles the tempo of the musical pieces better than Scheirer's algorithm.
5. I proposed a method of how to generate a transition between two beat-sequences.
6. An implementation of a sample application was completed.

8.2 Summary

In Chapter 5, I discussed Scheirer's approach on how to use comb-filters to detect the period of click-trains. I evaluated three different features that help to estimate the period. The majority of the click-train period were estimated correctly. I showed that by analyzing comb-filter states click-train periods were estimated with higher accuracy.

An extended model to Scheirer's approach was introduced in Chapter 6. First steps of the two models are similar, but the following processing steps differ. The differences aim for a faster processing time and regular beat-time output. A comparison shows that Scheirer's algorithm output is slightly more accurate, but not as regular as the output of the variants presented in this thesis.

A short overview of the implementation is given in Chapter 7. Then optimizations and their impact on the efficiency and accuracy are discussed. It turns out that one optimization not only improves the performance, but also improves the quality. The efficiency of the presented approach with Scheirer's implementation was examined and speed-ups between 3 and nearly 6 were achieved. A short discussion of the sample application closes the chapter.

The main idea of this thesis was to make Scheirer's model more applicable to synchronize computer generated animations to the beat of previously unknown music. That leads to two types of objectives,

which were discussed in Chapter 4, concerning functional abilities and the efficiency of the system. The functional goals focus on the minimal functional services the system has to provide and on the quality of the system's output. The performance goals make sure that the functional goals have to be implemented such that they can be processed in a timely manner.

Most of the functional goals are met. The system is able to analyze arbitrary sound data in PCM format. The beat output is regular and smooth and graceful transitions between tempi are created. These two goals are important since the regularity of the beat output affects the regularity of a synchronized animation. The algorithm can deliver information about beat-times at any-time. Furthermore, the performance goals are met. The implementation is fast enough to run on a low-end machine with less than 10% CPU-utilization. This leaves enough processor time to animate computer graphics while analyzing incoming music data.

Accuracy of Duden's and Scheirer's model are acceptable, but there is room for improvement. The following section discusses possible ways to improve the accuracy.

8.3 Future work

The quality of the beat-induction model is not perfect. It is easy to find songs where the imperfections of the beat-induction model are visible. This is especially obvious when one is listening to music and is watching the visualizer. High-level concepts like drum pattern matching may improve the quality as it was applied in (Goto and Muraoka 1998). Also, the detection of accents might be a good way to improve the accuracy (Seppänen 2001).

More work should be done on the architecture's ability to detect click-train periods. Click-trains are the simplest signals that convey a beat, but the architecture is not able to detect the period in all cases correctly. Features that are based on more than one comb-filter state might achieve better results. A better detection of simple signals may also improve the architecture for polyphonic music.

The model should be able to detect when it is not able to estimate the beat well. This could be done by analyzing the entropy of the feature values of the comb-filters. Low entropy indicates that there is a good chance to find a beat. In this case, there are only a few comb-filters states that exhibit low feature values, which means only a few tempi are likely to be selected as the tempo of the song. A high entropy indicates that the musical piece may only have a subtle beat. The feature values are all about the same level, which does not provide much information about the tempo of the musical piece. Such a measure was introduced by Scheirer (2000).

I hope, that the development of beat-induction algorithms continues and that quality and performance improves.

REFERENCES

- Bernstein, M. and M. Picker (1966). *An Introduction to Music*. Prentice Hall.
- Cambouropoulos, E., S. Dixon, W. Goebel, and G. Widmer. (2001, August). Human preferences for tempo smoothness. In *VII International Symposium on Systematic and Comparative Musicology, III International Conference on Cognitive Musicology*, Jyväskylä, Finland, pp. 18–26.
- Cemgil, A. T., B. Kappen, P. Desain, and H. Honing (2000). On tempo tracking: Tempogram representation and kalman filtering. *J. New Music Research* 29(4).
- Chai, W. (2001). Melody retrieval on the web. Master's thesis, Massachusetts Institute of Technology.
- Cooper, G. W. and L. B. Meyer (1960). *The Rhythmic Structure of Music*. The University of Chicago.
- Dixon, S. (2001a). Automatic extraction of tempo and beat from expressive performances. *J. New Music Research* 30(1). To appear.
- Dixon, S. E. (2001b). An empirical comparison of tempo trackers. In *8th Brazilian Symposium on Computer Music*, Fortaleza, Brazil.
- Foote, J. and S. Uchihashi (2001). The beat spectrum: A new approach to rhythm analysis. In *IEEE International Conference on Multimedia & Expo 2001*.
- Goto, M. and Y. Muraoka (1995). Beat tracking based on multiple-agent architecture – A real-time beat tracking system for audio signals–. In V. Lesser (Ed.), *Proceedings of the First International Conference on Multi-Agent Systems*. MIT Press.
- Goto, M. and Y. Muraoka (1997). Issues in evaluating beat tracking systems. In *Workshop on Issues in AI and Music, International Joint Conference on Artificial Intelligence*, pp. 9–16.
- Goto, M. and Y. Muraoka (1998). An audio-based real-time beat tracking system and its applications. In *Proceedings of the International Computer Music Conference*, San Francisco CA. Computer Music Association.
- Hipp, M. (2001). <http://www.mpg123.de>. Library.
- Large, E. and J. Kolen (1994). Resonance and the perception of musical meter. *Connection Science* 6, 177–208.
- Laroche, J. (2001, October). Estimating tempo, swing and beat locations in audio recordings. In *IEEE Workshop on Applicat. of Signal Proc. to Audio and Acoust. (WASPAA)*, New Paltz, NY, USA, pp. 135–138.

- Miller, B., D. Scarborough, and J. Jones (1992). On the perception of meter. In M. Balaban, K. Ebcioglu, and O. E. Laske (Eds.), *Understanding Music with AI: Perspectives in Music Cognition*, pp. 427–447. Cambridge: MIT Press.
- MP3.com (2002). <http://www.mp3.com>. Website.
- Sanjit K. Mitra, J. F. K. (Ed.) (1993). *Handbook for Digital Signal Processing*. John Wiley & Son.
- Scheirer, E. (1998a). <http://sound.media.mit.edu/~eds/beat/tapping.tar.gz>. Source Code.
- Scheirer, E. (1998b). Tempo and beat analysis of acoustic musical signals. *J. Acoust. Soc. Amer.* 103, 588–601.
- Scheirer, E. (1999, May). <http://www.ffem.org/gdam/library/eric-scheirer-tapping-email.txt>. Mailing List Posting.
- Scheirer, E. (2000). *Music-Listening Systems*. Ph. D. thesis, MIT Media Lab.
- Scheirer, E. D. (1995). Extracting expressive performance information from recorded music. Master's thesis, Massachusetts Institute of Technology.
- Seppänen, J. (2001). Computational models of musical meter recognition. Master's thesis, Tampere Univ. of Tech., Tampere, Finland.
- Todd, N. P. M. (1994). The auditory “primal sketch”: A multiscale model of rhythmic grouping. *J. New Music Research* 23(1), 25–70.
- Toiviainen, P. (1998). An interactive midi accompanist. *Comp. Music J.* 22(4), 63–75.
- Tzanetakis, G., G. Essl, , and P. Cook (2001, October). Automatic musical genre classification of audio signals. In *Proc. Int. Symposium on Music Inform. Retrieval. (ISMIR)*, Bloomington, IN, USA, pp. 205–210.