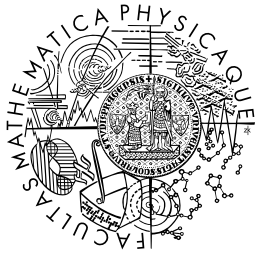

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Lucie Vomelová

Rhythm recognition

Department of Applied Mathematics

Supervisor of the bachelor thesis: Mgr. Martin Mareš, Ph.D.
Study programme: Computer Science

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date 8. 5. 2024 .....

Author's signature

I would like to thank my supervisor, Mgr. Martin Mareš, Ph.D., for his guidance and for all the recommendations he gave me. I am grateful for all the time he spent leading this work.

Title: Rhythm recognition

Author: Lucie Vomelová

Department: Department of Applied Mathematics

Supervisor: Mgr. Martin Mareš, Ph.D., Department of Applied Mathematics

Abstract: Rhythm is an important part of music. It is easy for humans to hear or feel the rhythm of a song, but automatic rhythm recognition is a complicated task. In this work, we introduce an algorithm for automatic rhythm recognition in songs with constant tempo. The algorithm follows these steps: onset detection, tempo analysis, beat detection, and rhythm detection. We introduce different approaches for each step. The goal is to compare the approaches to find the best one. We provide a Python package implementing all of the described approaches and steps. We also conduct experiments to determine the best approach in each step.

Keywords: music, rhythm, tempo, Fourier transform

Název práce: Rozpoznávání rytmu

Autor: Lucie Vomelová

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D., Katedra aplikované matematiky

Abstrakt: Rytmus je důležitou součástí hudby. Pro lidi je často snadné slyšet rytmus písně, ale automatická detekce rytmu je komplikovaná. V této práci představíme algoritmus na automatickou detekci rytmu v písních s konstantním tempem. Algoritmus sestává z následujících kroků: detekce počátků not, analýza tempa, detekce dob a detekce rytmu. Pro každý krok představíme různé postupy. Cílem práce je tyto postupy porovnat. Součástí práce je balíček napsaný v jazyce Python, který implementuje algoritmus na detekci rytmu a nabízí všechny popsané postupy. Balíček použijeme v experimentech, z jejichž výsledků určíme nejlepší postupy.

Klíčová slova: hudba, rytmus, tempo, Fourierova transformace

Contents

Introduction

Musical beat and rhythm is something that is fairly easy for people to detect. We often see people tapping their foot when listening to a song. Dancing is also based on rhythm and on our ability to hear it.

But in some cases we cannot use our ability to detect rhythm, we need to do it automatically — for example in musical mobile games. For me, this problem arose when I wanted to develop an application for automatic choreography generation for games like Dance Dance Revolution or StepMania. These games are based on dance pads — a square pad that people stand on that has arrows pointing in every direction. The goal is to press these arrows with our feet (and sometimes hands) based on the displayed choreography. There is a large database of songs that people can choose from, but I wanted to expand the possibilities and choose songs that are not in this database. That is why I wanted to develop an application that would generate a choreography for any given song.

To do that, I had to first look into automatic rhythm recognition. There are many approaches that people tried over the years, many publications, papers, books and projects. I decided to test different approaches and see which of them work the best or if there are some specific situations where some methods are better than other. In this thesis, I will describe my findings.

I decided to split the task of rhythm recognition into several sub-tasks. The proposed algorithm will always have the following four steps, but different approaches in each step will be examined. The four main steps are:

1. Onset detection.
2. Tempo analysis.
3. Beat detection.
4. Rhythm detection.

Onset refers to the beginning of a musical note or other sound, so onset detection is an obvious first step in rhythm recognition. Rhythm is usually determined by repeating patterns in musical notes. In order to detect rhythm, we must first detect when notes start.

Musical tempo, also known as beats per minute, defines the pace of a song. It doesn't have to be constant, it can change throughout the song — for example in classical music. But in this thesis, we will focus on songs with constant tempo.

Beat is the element that drives music forward. It is often described as a sequence of perceived pulse positions that are equally spaced in time. Beats are considered to be the fundamental component of rhythm.

After detecting beats, we can finally move to rhythm detection. Rhythm is the pattern of sounds, silences, and emphases in a song. It is dependent on the dynamics of strong and weak beats, played beats and rest beats, or long and short notes.

In this thesis we will introduce different methods for onset detection, tempo analysis, beat detection, and rhythm detection. The first chapter will describe the necessary theoretical background for this topic. The second chapter will provide

an overview of related work. Then, the following four chapters will look at each subtask of the proposed algorithm for rhythm recognition — onset detection, tempo analysis, beat detection, and rhythm detection. Different approaches for each subtask will be described and tested. In the last chapter, we will look at the implementation of the proposed approaches and we will compare them by conducting experiments.

1 Theoretical background

What is music? Many people tried to define it, but music is a complicated concept. Most definitions are circular or incomplete. Personally, I like this definition:

"Music is an organized sound." — Edgard Varèse [1]

Music is a format of art using sound. It is a combination of sounds that express ideas and emotions through rhythm, melody, harmony and color.

1.1 Basic musical terminology

Music has many fundamental elements. In this thesis, the main focus will be on the rhythmic part, but we will briefly introduce other elements of music. After that, important concepts for rhythm will be introduced more thoroughly.

Sound

Sound is and acoustic wave that travels through a medium such as air. Waves that have frequencies lying approximately between 20 Hz and 20 kHz can be captured by human ears. Music is composed of sounds in this frequency range.

Musical tone

A musical tone is a steady periodic sound, which has many properties such as duration, intensity, pitch, or timbre. Duration refers to the length of the sound and intensity refers to its loudness.

Pitch is the perceived frequency of a sound. It allows us to order sounds on a scale based on how high or low we perceive them.

Timbre, also known as tone color or tone quality, is the perceived sound quality of a sound. It allows us to distinguish different instruments or voices that produced the sound.

Musical note

Notes are the basic building blocks in music. They are closely related to the concept of tone. Notes have a value, which corresponds to their relative duration. The basic note values are: a whole note (semibreve), a half note (minim), a quarter note (crotchet) and an eight note (quaver), where each note is half of the duration of the previous one. There are other note values corresponding to different fractions (or multiples) of the whole note.

Melody and harmony

Melody is a series of musical tones sounding one after another. It is often perceived by humans as one entity.

Harmony refers to combined pitches that are sounding together at the same time. An important harmonic concept are chords. Chord is a set of notes sounding simultaneously. Chords play a key part in most modern music genres.

1.2 Important concepts for rhythm recognition

As I already mentioned, rhythm will be the main focus of this work. Rhythm recognition will consist of these steps: onset detection, tempo analysis, beat detection, and rhythm detection. In this section, we will describe onset, tempo, beat, and rhythm. Specific methods used in each step will be described later.

1.2.1 Onset

Onset refers to the beginning of a musical note or other sound. It is related to two additional musical concepts: the attack and the transient of a note. At the beginning of a musical tone, there is often a sudden increase of energy. The attack of a note refers to the sound build up. Transient is a short sound with high amplitude and it typically occurs at the beginning of a sound event.

Onset refers to the single instant that marks the beginning of the transient.

1.2.2 Beat

In music, beat is the basic unit of time. It is the element that drives music forward. It can be defined as a repeating pulse that underlies a musical pattern. Beat is specified by two parameters: a phase and a period.

Beat is a fundamental component of rhythm. It is often the pattern that humans tap their foot to when listening to a musical piece.

1.2.3 Tempo

Musical tempo, also known as beats per minute, defines the pace of a song. It describes the number of beats that occur per minute in a song. It is given by the reciprocal of the beat period. Tempo doesn't have to be constant, it can change throughout the song — for example in classical music.

Tempo of a musical piece can be described by a number denoting beats per minute, or it can be described by words. There is a standardized vocabulary for this purpose that uses mostly Italian terms — for example lento (slow, 52–108 beats per minute) or allegro (lively, 120–156 beats per minute). But this terminology is mostly used in classical music. Modern songs often use just a number specifying beats per minute (BPM).

Metronome

A metronome is a device that produces a click (or some other sound) at a regular interval. The interval can be set by the user and it is typically set in beats per minute.

1.2.4 Rhythm

Rhythm is a fundamental part of music. It is the pattern of sounds, silences, and emphases in a song. The pattern is usually short enough for humans to memorize.

As previously stated, beats are the fundamental component of rhythm. Rhythm is dependent on the dynamics of strong and weak beats, played beats and the inaudible but implied rest beats, or long and short notes.

Measure, tactus and tatum

Measure, tactus and tatum are three terms closely related to musical rhythm. A measure (also known as a bar) is a defined segment of time in a musical piece. In musical notation, it is denoted by a vertical line on each side. A measure provides structure to music notation since it groups notes together. It is usually defined by the number of note values it contains.

Tactus (or pulse) corresponds to the underlying pulse in music. It usually corresponds to the quarter note levels. It sets the tempo of a musical piece, because it defines a series of beats equally spaced in time. Not all beats need to be audible, they can also be implied (rest beats).

Tatum is the smallest time interval between successive notes.

1.3 Audio signal processing

Audio signals are electronic representations of sound waves. Audio signal processing is focused on electronic manipulation of those waves. It is a subfield of signal processing.

An important feature in audio signal processing is sound loudness, which is measured in decibels.

Decibel

Decibel (dB) is a relative unit of measurement. One decibel is equal to 0.1 bel (B). The bel scale is logarithmic, it uses logarithm with base 10. That means that an increase of 1 B is equal to a tenfold relative increase.

The energy of audio signal (perceived as sound loudness) is measured in decibels, relatively to the intensity of $1\,\mathrm{pW/m^2}$, which is approximately the threshold of hearing. This is useful because sensitivity of human ears is also logarithmic — if two sounds differ in intensity by 1 B (10 dB), we will perceive the louder one as approximately twice as loud.

But the decibel scale is used as a unit for ratio in general. That is why throughout this thesis, we will see decibel scale portraying relative changes in power, but it will not necessarily correspond to the classical sound scale.

1.3.1 Analog and digital signal

Analog signal

Analog signal is a continuous-time signal. Sound itself is an analog signal. Analog recordings capture the continuous sound wave of the signal.

Analog signal typically represents sound using a changing level of electrical voltage. The most common recording mediums are magnetic tape and vinyl records.

Analog signal cannot be processed by computers — that is why we need to convert it to digital signal. Converting an analog audio to a digital format usually requires the use of an analog-to-digital converter.

Digital signal

Digital signal is obtained from an analog signal using sampling and quantization. Sampling is a process of making the time-axis discrete, quantization is a process of making the value range discrete. Digital signal represents the original analog sound sampled at regular intervals with values rounded (or truncated) to the nearest discrete value.

1.3.2 Sampling rate

Sampling rate f_s (or sample rate, sampling frequency) defines the number of samples per second taken from a continuous (analog) signal to make a discrete (digital) signal. It is measured in Hertz. The higher the sampling rate, the higher frequencies can be captured in the digital representation.

In order to obtain a lossless representation of the continuous signal, we need to use a sampling rate such that it is possible to capture the highest frequency f_{max} present in the signal. The connection between the sampling rate f_s and the highest frequency f_{max} that can be captured is defined by the Nyquist-Shannon sampling theorem.[2]

Nyquist-Shannon sampling theorem

The Nyquist-Shannon sampling theorem states that: Let f_{max} be the maximum frequency value of the continuous-time signal $x_{cont}(t)$. In order to get a discrete-time copy $x(t)$ of $x_{cont}(t)$ that can be reconstructed perfectly to its original form $x_{cont}(t)$, we need to use a sampling rate f_s, such that:

$$f_s > 2f_{max}.$$

The highest frequency that can be captured when using sampling rate f_s is called the Nyquist frequency:

$$f_{Nyquist} = \frac{1}{2}f_s.$$

Standard sampling rates

There are some standard sampling rates that are used in most places. The first one is 44.1 kHz, or 44 100 samples per second, which is the standard for CDs and other audio formats. This rate was chosen because it can reproduce the whole sound frequency range audible to the human ear (because 20 kHz, which is the highest audible frequency, is smaller than the Nyquist frequency).

Another common sampling rate is 48 kHz, or 48 000 samples per second. This sampling rate is often used in audio for video.

1.3.3 Audio file formats

An audio file format is a file format used for storing digital audio. There are several different file formats used for storing audio, each has its specifics. Some formats use data compression to reduce the file size, other formats have higher quality.

There are many different audio file formats, for example Waveform Audio File Format or MPEG Audio Layer 3.

Waveform Audio File Format

Waveform Audio File Format is a lossless audio file format. It provides excellent sound quality, but the file size is quite large. This format is used when the highest audio quality is needed. Files of this format have an extension .wav or .wave.

This is the file format that is used in the provided Python package. Waveform Audio File Format is used because the audio quality is high so the program can work with complete information. Also, the conversion from other audio formats to Waveform Audio File Format is simple.

MPEG Audio Layer 3

MPEG Audio Layer 3 is probably the most popular audio format. It uses a lossy data compression, so the sound quality is not as high as with Waveform Audio File Format, but it is still good. A big advantage is smaller file size. Files of this format have an extension .mp3 (or .bit).

1.4 Audio signal domain features

Audio features capture some aspect of an audio signal. Different features capture different aspects of the signal. After extraction, these features can be used for audio signal processing.

1.4.1 Frame size and hop length

Frame size and hop length are two very important feature extraction parameters. Usually, audio features are computed over some input signal and they return one number. If this computation was applied over the whole signal, we would get only small amount of data. That is why framing is used — the input signal is cut into smaller frames and features are calculated over them. That way, we get more information about the overall behavior of the signal.

Frame size specifies number of samples in a frame (number of samples per second is specified by sampling rate). Hop length refers to the number of samples by which we have to advance between two consecutive frames. In other words, it determines the overlap between frames.

These two parameters are related to the temporal scope categorization in the next section. But they are introduced sooner, because they are one of the most important parameters in audio signal processing.

1.4.2 Categorization of audio signal features

Audio features can be categorized based on different properties, such as level of abstraction, temporal scope, musical aspect, or the signal domain.

Level of abstraction

Feature categorization can be based on their level of abstraction. There are three levels: low-level, mid-level and high-level audio features.

Low-level features are statistical features and they contain detailed information about the signal. They make sense to computers, but they are not easily interpretable by humans.

Mid-level features provide information about the structure and composition of sound. They are associated with musical attributes such as pitch, beat or note onsets. These features can be perceived by humans.

High-level features are abstract features that give us a deeper understanding of the musical aspects of the signal. They are understood and enjoyed by humans. These features include concepts such as chords, melody, rhythm, or tempo.

Temporal scope

In this case, feature categorization is based on the time duration over which the feature is calculated. There are three temporal scope categories — instantaneous, segment-level and global.

Instantaneous features contain information about the signal at a specific time instant. They capture very short chunks of the signal, in the range of milliseconds. The minimal temporal resolution that humans can perceive is 10 ms.

Segment-level features are calculated over segments of audio, typically in the range of seconds.

Global features are aggregate features that summarize the information from lower temporal resolution features. They provide information about the entire audio signal by aggregating results from lower-level features.

Musical aspect

Feature categorization can also be related to musical aspects of the signal such as rhythm, pitch or note onsets.

Signal domain

Feature categorization can be based on the signal domain they belong to. There are two main categories — time domain and frequency domain features.

Time domain features are extracted directly from the audio waveform. They provide information about the temporal characteristics of the sound. Time domain feature extraction is considered to be simpler than frequency domain extraction, because no transformation is required.

Frequency domain features focus on the frequency components of sound. These features can be obtained by applying the Fourier transform to the time-domain representation. They provide information about spectral characteristics.

I decided to categorize features primarily by the signal domain they belong to. In the following sections, I will introduce specific time domain and frequency domain features and other related concepts.

1.4.3 Time domain

Time domain signal processing techniques are based on the audio waveform observed over a period of time.

Waveform

A waveform is a graphical representation of an audio signal as it moves through some medium over time. It displays amplitude or level changes over time.

The simplest type of waveform is a sine wave. It represents a single sound frequency:

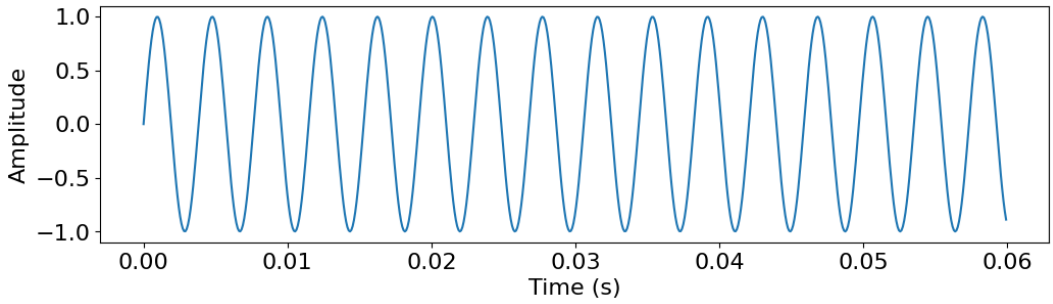

Figure 1.1 Sine wave

Waveforms can be both periodic and aperiodic. An example of a periodic waveform is the previously mentioned sine wave, an example of an aperiodic waveform is a waveform depicting some noise. The main focus of this thesis are song waveforms, which are complex periodic waveforms. They are composed of multiple different sine waves.

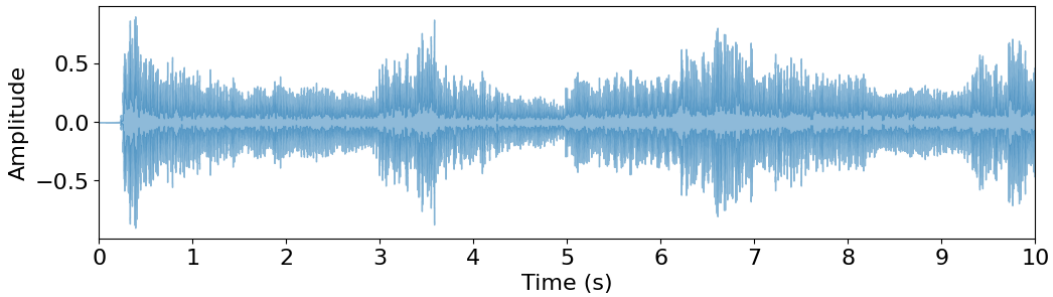

Figure 1.2 Waveform

The song used in the visualization of the waveform and all of the visualizations in this thesis (unless specified otherwise) is Valhalla Calling Me by Miracle of Sound.

Amplitude envelope

Amplitude envelope describes the maximum amplitude of an audio signal in one frame. It is very easy to compute, but still very useful.

Mathematically, for a frame k containing samples $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$, the amplitude envelope is:

$$AE(k) = \max_{j=1}^{\ell} x_{i_j}$$

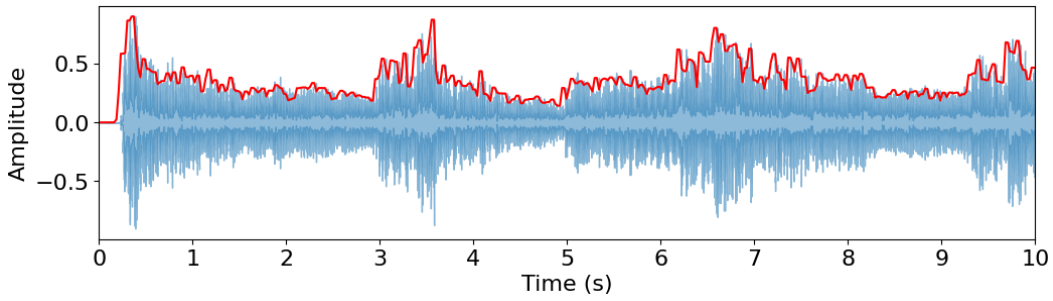

Figure 1.3 Amplitude envelope

Zero crossing rate

Zero crossing rate is simply the number of times the signal crosses zero (the time axis) in one frame.

Mathematically, for a frame k containing samples $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$, the zero crossing rate is:

$$ZCR(k) = |\{j \mid i_1 \leq j < i_\ell \wedge \operatorname{sgn} x_j \neq \operatorname{sgn} x_{j+1}\}|.$$

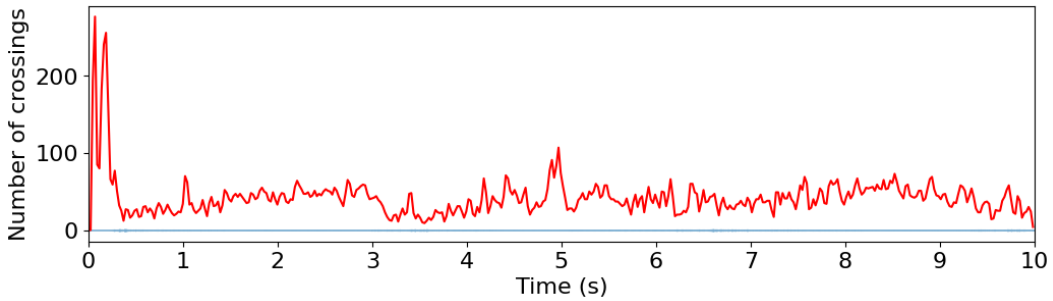

Figure 1.4 Zero crossing rate

Root mean square energy

The root mean square energy is obtained by taking the square root of the mean of the square of all the amplitude values in a frame. The root mean square energy contains information about the overall intensity of an audio signal by taking into account both positive and negative excursions of the waveform. It provides an accurate measure of the signal's power.

Mathematically, for a frame k containing samples $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$, the root mean square energy is:

$$RMSE(k) = \sqrt{\frac{1}{\ell}\sum_{j=1}^{\ell} x_{i_j}^2}$$

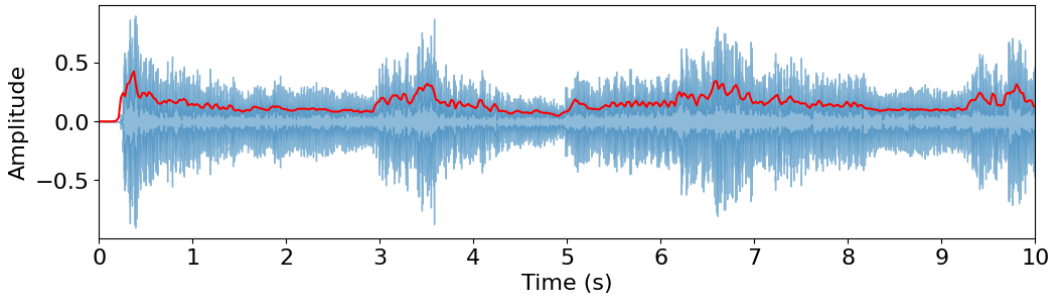

Figure 1.5 Root mean square energy

1.4.4 Frequency domain

In the frequency domain, the signal is represented as a function of frequency. The audio signal is decomposed into its constituent frequencies, revealing the amplitude and phase information associated with each frequency.

Fourier transform

The Fourier transform is a frequently used technique in signal processing. It is the most common tool used for converting a signal from time domain to frequency domain. It transforms a function of time $x_{cont}(t)$ to a function of frequency $X_{cont}(f)$. Formally, the Fourier transform equation is:

$$X_{cont}(f) = \int_{-\infty}^{+\infty} x_{cont}(t)\mathrm{e}^{-2\mathrm{i}\pi ft}\,\mathrm{d}t$$

It takes a continuous signal $x_{cont}(t)$ as input and decomposes it into a sum of sine and cosine waves of different frequencies each with specific amplitude and phase. The sine and cosine waves are included in the complex exponential $\mathrm{e}^{-2\mathrm{i}\pi ft}$. That can be seen with the Euler's formula: $\mathrm{e}^{\mathrm{i}x} = \cos x + \mathrm{i}\sin x$.

If we want to get back from the frequency domain to the time domain, we can simply apply the inverse Fourier transform:

$$x_{cont}(t) = \int_{-\infty}^{+\infty} X_{cont}(f)e^{2\mathrm{i}\pi ft}\mathrm{d}f$$

Discrete Fourier transform

Fourier transform works with a continuous signal. Since digital signal is discrete, we need to use a discrete version of it — the Discrete Fourier transform (DFT):

$$X(f) = \sum_{t=-\infty}^{\infty} x(t) \cdot e^{-i2\pi ftT}$$

where T is the duration of one sample (so for sampling rate f_s, the duration of one sample is $T = 1/f_s$).

Since the time function is discrete, the resulting frequency function will also be discrete. For an input signal of length ℓ we will get N discrete frequency bins, where $N = \ell/T$. A frequency bin n_i, $i \in \{0, \ldots, N-1\}$ covers following interval of frequencies: $n_i = \left[\frac{f_s \cdot i}{2N}, \frac{f_s \cdot (i+1)}{2N}\right]$

For our usage — audio signal processing — the input time function $x(t)$ will always be non-zero in finite number of points. That means that the infinite sum will be computed from finite number of non–zero values, so the result will also be finite.

Fast Fourier transform

Fast Fourier transform (FFT) [3] is an algorithm that computes the Discrete Fourier transform of a sequence, or its inverse. Computing DFT directly from definition is often too slow and impractical. FFT reduces this complexity from $O(N^2)$ to $O(N \log N)$, where N is the frequency domain size. FFT is based on the divide-and-conquer principle and it needs for frame size to be a power of 2.

Short time Fourier transform

DFT describes frequency components in the whole input signal. But sometimes it is useful to look at shorter parts of signal separately — when we take a song, the frequency components change over time based on which part of the song is played. That is when Short time Fourier transform (STFT) can be used. It is simply a sequence of Discrete Fourier transforms applied on short parts of the input signal. In our case, STFT is applied on frames.

STFT can also be used in the continuous domain, but for the purpose of this work, only the discrete domain is useful.

Typically, STFT uses a window function $w(n)$ applied over a time-domain function $x(t)$. Two commonly used windows are the rectangular window and the Hamming window. The rectangular window simply takes the desired part of a signal and replaces all values but those with 0.

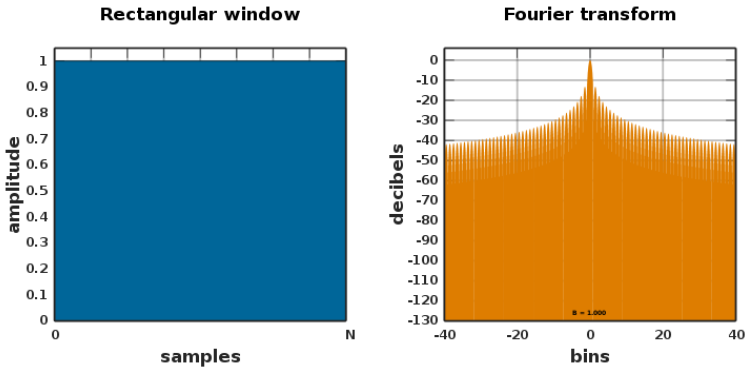

Figure 1.6 Rectangular window (Source: wikipedia.org)

The Hamming window applies tapers to the ends. It is part of the cosine-sum window family, which has the following formula:

$$w(n) = \sum_{k=0}^{K} (-1)^k a_k \cos\left(\frac{2\pi kn}{N}\right), \ 0 \leq n < N.$$

The parameter N specifies the length of the window. Cosine-sum windows for case $K = 1$ have the form:

$$w(n) = a_0 - (1 - a_0) \cos\left(\frac{2\pi n}{N}\right), \ 0 \leq n < N.$$

If we set a_0 to approximately 0.54 we will get the Hamming window.

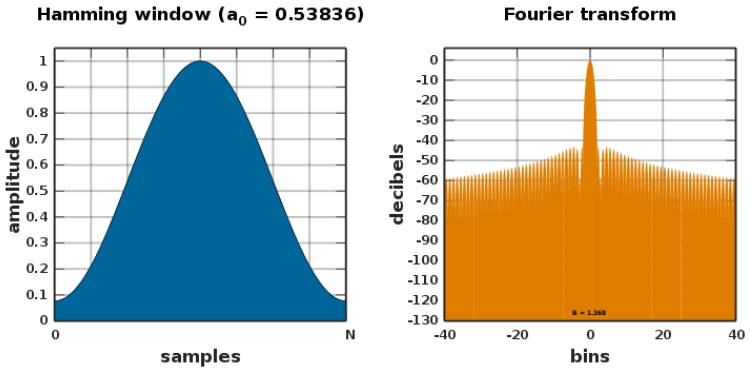

Figure 1.7 Hamming window (Source: wikipedia.org)

Mathematically, the discrete STFT can be expressed as:

$$\mathcal{X}(m, f) = \sum_{n=0}^{N-1} x(n + mH)w(n) \cdot e^{-i2\pi fn/N},$$

where m specifies the frame index and f is the frequency index. The maximal frequency index is $n_{max} = N/2$ and the corresponding frequency is the Nyquist frequency.

The frame size is specified by N and H is the hop length. w is a sampled window function $w \in [0 : N-1]$ of length $N \in \mathbb{N}$. So $x(n+mH)$ is the discrete-time signal in the n-th sample of the m-th frame.

The number $\mathcal{X}(m, f)$ is also called the f-th Fourier coefficient for the m-th frame. It is sometimes denoted as $\mathcal{F}(m, f)$.

Magnitude spectrogram

A magnitude spectrogram is a two-dimensional representation of the spectrum of the audio signal frequencies over time. It is usually depicted as a heat map — the x axis represents time, the y axis represents frequency bins and the loudness (or magnitude) of a particular frequency is represented by the color of each point.

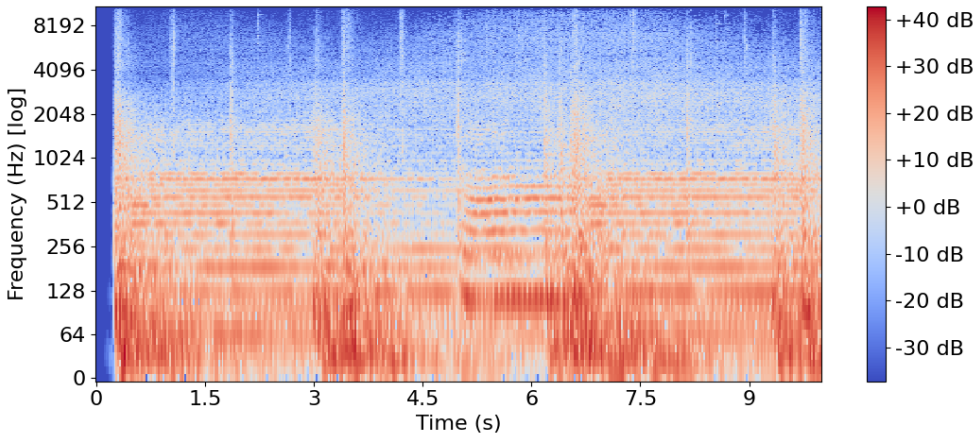

Figure 1.8 Spectrogram

Tempogram

Tempogram indicates specific tempo relevance for each time instance in a song. It is similar to a spectrogram, which indicates frequency relevance at each time instance.

Mathematically, a tempogram is a function

$$\mathcal{T} : \mathbb{R} \times \mathbb{R}_{>0} \to R_{\geq 0}$$

depending on a time parameter $t \in \mathbb{R}$ and a tempo parameter $\tau \in \mathbb{R}_{>0}$ measured in beats per minute (BPM). The value $\mathcal{T}(t, \tau)$ indicates how much the signal contains a locally periodic pulse of tempo τ in the neighborhood of time t.

In practice, discrete time-tempo grid is used. From that, we get a discrete tempogram

$$\mathcal{T} : \mathbb{Z} \times \Theta \to \mathbb{R}_{\geq 0},$$

where $\Theta \subset \mathbb{R}_{>0}$ is a finite set of possible tempo values measured in BPM. The value $\mathcal{T}(t, \theta)$ indicates how much the signal contains a locally periodic pulse of tempo $\theta \in \Theta$ in the neighborhood of time $t \in \mathbb{Z}$.

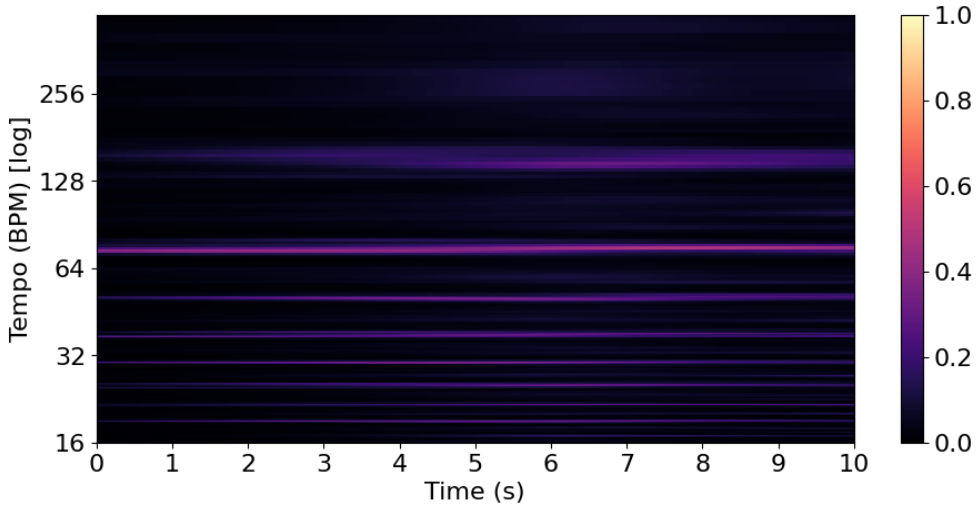

Figure 1.9 Tempogram

2 Related work

Automatic rhythm recognition in music is a field that has been studied extensively. This chapter will provide an overview of some of the resources that inspired this thesis.

Fundamentals of Music Processing

Fundamentals of Music Processing [4] is a book focused on the whole area of music processing, not just rhythm recognition. It includes a whole chapter on Tempo and Beat Tracking. This book also has additional resources — called FMP Notebooks [5] — that show how to use the theoretical concepts from the book in practice.

Fundamentals of Music Processing is also the main resource that I used for this thesis. The chapters about onset detection and tempo analysis are based mostly on information from this book.

The book offers several approaches for onset detection:

1. Energy-based approach
2. Spectral-based approach
3. Phase-based approach
4. Complex-domain approach

We will look at the first two approaches — the energy-based and the spectral-based approach.

Fundamentals of Music Processing also describes several approaches for tempo analysis:

1. Fourier tempogram
2. Autocorrelation tempogram
3. Cyclic tempogram

We will use the first two approaches — the Fourier tempogram and the autocorrelation tempogram.

Musical note onset detection based on a spectral sparsity measure

Musical note onset detection based on a spectral sparsity measure [6] is a research paper focused on the task of note onset detection. The authors propose a method for onset detection based on spectral sparsity. Spectral sparsity is a frequency-domain feature which indicates how many frequencies are needed to represent the audio signal in a particular time frame.

A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds

A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds [7] provides a very thorough overview of different audio signal features. It does not focus solely on musical audio features, but many of the mentioned features are applicable for music processing.

The authors mention for example amplitude-based features, zero-crossing rate or STFT-based features.

Music Audio Rhythm Recognition Based on Recurrent Neural Network

Music Audio Rhythm Recognition Based on Recurrent Neural Network [8] is a research paper published in March 2022.

The paper proposes a machine learning approach for rhythm recognition. It uses recurrent neural networks with gated recurrent units to build a short-term music audio rhythm extraction and recognition model. It also examines the impact of different activation functions on the model's accuracy.

Librosa

Librosa [9] is a Python package for music and audio analysis. It provides many tools for onset detection, tempo analysis and beat detection.

The Librosa package is used in the experimental part of the thesis. Audio file manipulation and tempogram calculations are done using the package.

2.1 Automatic step file generators

As I already mentioned, this thesis was inspired by games such as Dance Dance Revolution, because I wanted to create an automatic choreography generator (or more precisely, automatic step file generator). That's why I researched this particular area extensively. There are some systems created exactly for this purpose. They are closely related to the task of rhythm recognition, because rhythm is useful for good step placement.

Dancing Monkeys

Dancing Monkeys [10] is a project for automatic step file generation from 2003. This project was implemented in MATLAB.

Just like in this thesis, it is assumed that the input song has constant tempo. Tempo and beat detection in this project is based mostly on the assumption that beats are played by instruments with high energy (like bass instruments). This approach is similar to the energy-based approach that will be introduced in this thesis in the onset detection part.

Dance Dance Convolution

Dance Dance Convolution [11] is a machine learning conference paper from 2017 focused on the task of learning to choreograph. The authors combined

recurrent and convolutional neural networks for step placement prediction and long short-term memory model for step selection.

The step placement prediction starts by computing multiple-timescale STFT using three different window lengths, because shorter window sizes contain information related to low-level features while larger window sizes provide information for high-level features. Then the STFT magnitude spectrum dimensionality is reduced to 80 frequency bands.

The output of this part is then used as an input to a convolutional neural network. After that follows a recurrent neural network with 2 layers of long short-term memory units. The final part is a simple sigmoid unit which estimates the probability that a step will be placed.

3 Onset detection

Onset refers to the beginning of a musical note or other sound. It is a single point that marks the beginning of a transient of a note.

For onset detection, we will try two different approaches: an energy-based approach and a spectral-based approach. Both follow the same steps, but each one uses different signal properties. The steps are as follows:

1. Extract suitable audio features from the input signal.
2. Derive a novelty function from the extracted data.

Each approach will be tested on the same song and the resulting graphs will be displayed, so we can see advantages and disadvantages of both approaches.

Novelty function

Before looking at the onset detection approaches, we must introduce the concept of novelty functions. Novelty function $\Delta : \mathbb{Z} \to \mathbb{R}$ is a function that denotes local changes in signal properties. When computed from certain signal properties, peaks in novelty function should indicate note onsets.

3.1 Energy-based approach

The first approach uses root mean square energy (RMSE) audio feature. RMSE is a time domain feature that contains information about the overall intensity or strength of an audio signal. This approach is based on the assumption that note onsets correspond with sudden increases in energy.

3.1.1 Extract root mean square energy

The first step is to calculate RMSE of the input signal. That will give us information about energy changes in the signal.

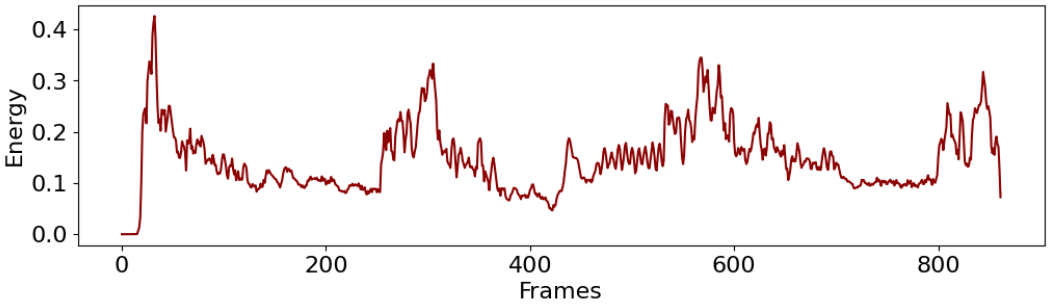

Figure 3.1 Root mean square energy

3.1.2 Derive energy based novelty function

After extracting root mean square energy, we want to derive the energy-based novelty function. In order to do that, we need to apply the following steps:

1. Calculate the discrete derivative.
2. Apply half-wave rectification.

Discrete derivative

In order to find points with sudden energy changes, we can use the derivative of the energy function. The function is discrete, so the easiest way to obtain a derivative of it is to calculate the difference between two subsequent energy values for the whole function (this operation is also called first-order difference).

Half-wave rectification

After obtaining the discrete derivative, half-wave rectification should be applied to the result. We are only interested in energy increases, so we want to keep only the positive differences. Negative differences are irrelevant for locating note onsets, so they can be set to zero.

This is exactly what half-wave rectification does. For a discrete input function $f(t)$ it sets all its negative points to zero:

$$y(t) = \max(f(t), 0)$$

Putting everything together, the energy-based novelty function is obtained by computing

$$\max\left(RMSE(k+1) - RMSE(k),\ 0\right)$$

for each two consecutive frames in the input signal k and $k+1$.

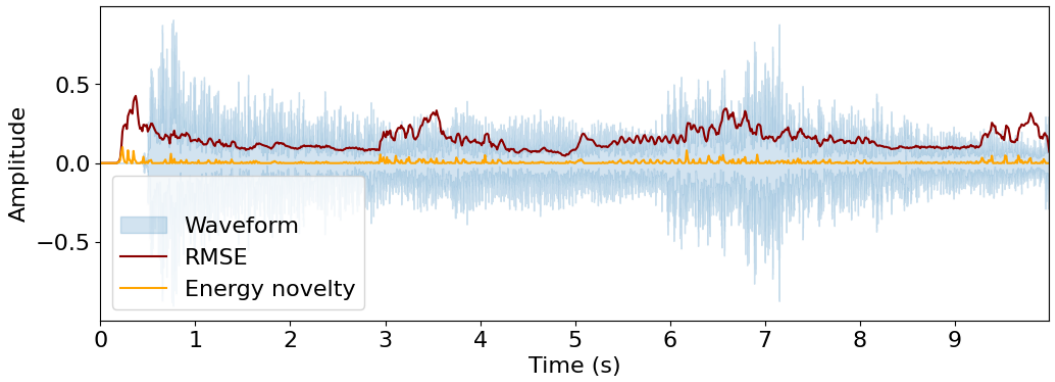

Figure 3.2 Comparison of root mean square energy function and the computed novelty function

The problem with the computed novelty function is that some note onsets do not contain nearly as much energy as other note onsets, this can depend for example on the musical instrument. We can partially fix this by applying logarithmic compression to the root mean square energy of the signal, so that energy increases are amplified and it is easier to detect them.

Logarithmic compression

Logarithmic compression is a powerful tool. Let $\gamma \in \mathbb{R}_{>0}$ be a positive constant. Then, the logarithmic compression $\mathcal{L}_\gamma(x)$ of a positive real value x is computed as:

$$\mathcal{L}_\gamma(x) = \log(1 + \gamma x).$$

The function $\mathcal{L}_\gamma(x)$ is equal to zero when $x = 0$, but it behaves like $\log(\gamma x)$ when x is large. Parameter γ, also called the compression factor, regulates the degree of compression.

After applying logarithmic compression, we will again calculate the discrete derivative and apply half-wave rectification. The resulting novelty function will be:

$$\Delta_{RMSE}(k) := \max\left(\mathcal{L}_\gamma(RMSE(k+1)) - \mathcal{L}_\gamma(RMSE(k)),\ 0\right)$$

If we choose a high enough compression factor γ, energy increases will be amplified. The downside is that the logarithm can amplify noise-like sound components, which may result in peaks that do not indicate note onsets. So we need to choose the compression factor carefully.

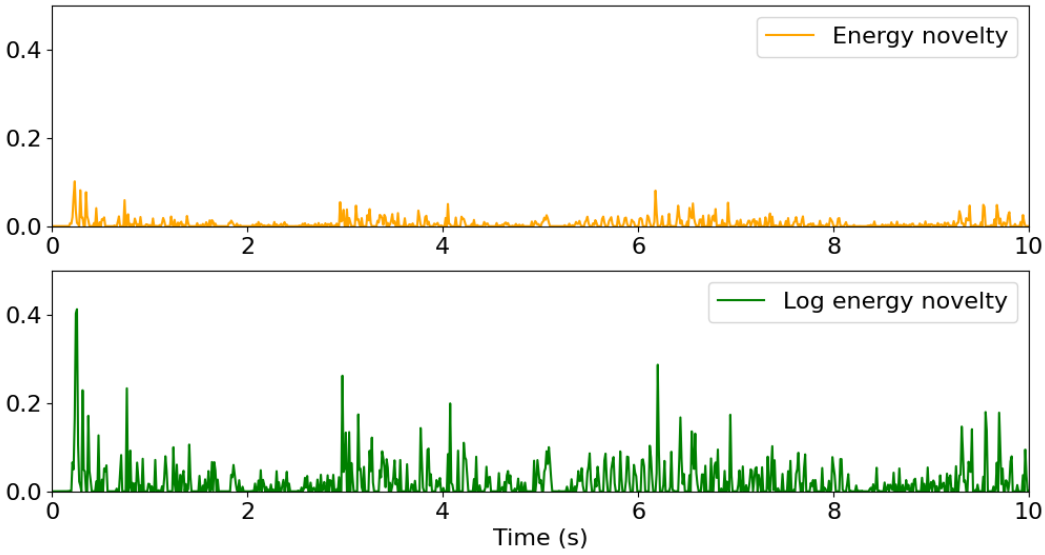

Figure 3.3 Comparison of energy based novelty function before and after logarithmic compression ($\gamma = 10$)

We can see that peaks are much more distinct when we applied logarithmic compression. These peaks should indicate note onsets.

3.2 Spectral-based approach

The second approach is based on spectral-based novelty function, also known as the spectral flux. It is computed from time-frequency representation of the signal. The idea is that by tracking changes in frequency content of the signal, we can detect note onsets.

3.2.1 Spectral based novelty function

Computing spectral based novelty function is more complicated than energy based novelty function. Following steps are needed:

1. Compute spectrogram.
2. Apply logarithmic compression.
3. Compute the discrete derivative.
4. Apply half-wave rectification.
5. Sum up results across all frequency bins.
6. Subtract local average and normalize the result.

Spectrogram

First step is computing the magnitude spectrogram $\mathcal{X}$.

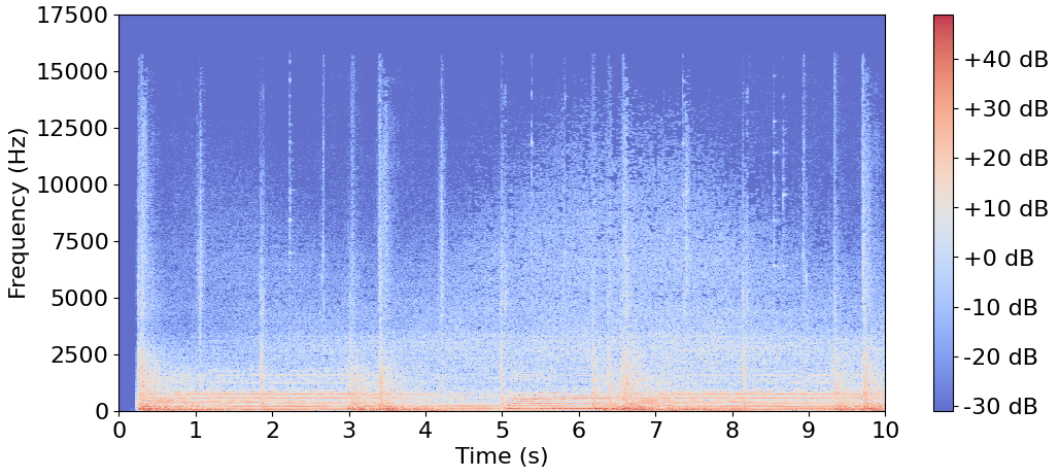

Figure 3.4 Spectrogram

Logarithmic compression

To enhance weak spectral components, we will apply logarithmic compression to the spectrogram $\mathcal{X}$:

$$\mathcal{Y} = \mathcal{L}_\gamma(|\mathcal{X}|) = \log(1 + \gamma|\mathcal{X}|)$$

By increasing γ, low-intensity components are brought out. But with too large compression factor, irrelevant components (like noise) can be amplified as well.

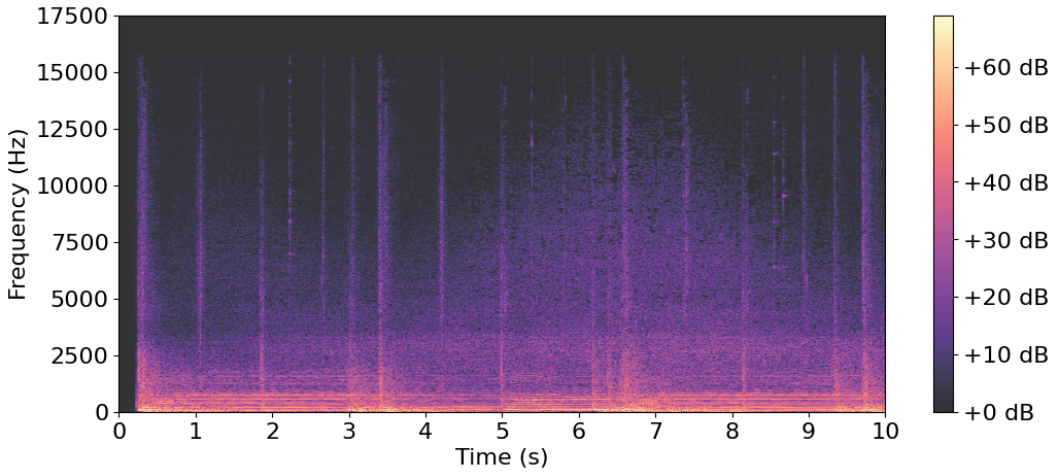

Figure 3.5 Spectrogram after logarithmic compression ($\gamma = 10$)

Discrete derivative

Just like in the previous approach, we are interested in amplitude changes, so we need to compute the discrete derivative. But in this case, the discrete derivative is computed for each frequency bin separately.

Half-wave rectification

Just like in the energy-based approach, we are interested only in amplitude increases, so we need to apply half-wave rectification. Again, it should be applied on the results of the previous step and on each frequency bin separately. We can look at it in a way that we are computing a novelty function for each frequency bin.

Sum up results across frequency bins

Now we need to combine the results of all frequency bins. To do that, we need to sum up the results from the previous step at each point in time. In other words, we take the functions for each frequency bin and we sum up their values at the same time point.

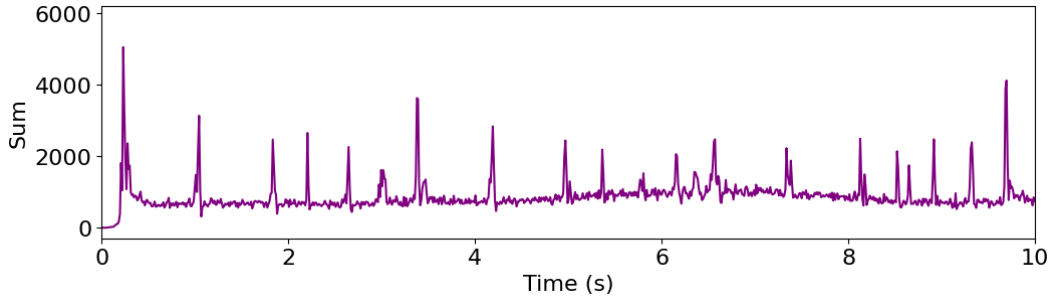

Figure 3.6 The result of summing up novelty functions across all frequency bins

Subtract local average and normalize the result

To enhance the properties of the spectral based novelty function, we can first subtract local average and then apply normalization.

Subtracting local average should suppress small fluctuations in the function. We just need to choose the number of subsequent frames M that will be used to compute local average. After computing the local average, we subtract it from the novelty function. If the result is less than 0, we set it to 0.

After that, we just apply simple normalization to get the normalized spectral-based novelty function $\Delta_{Spectral}$

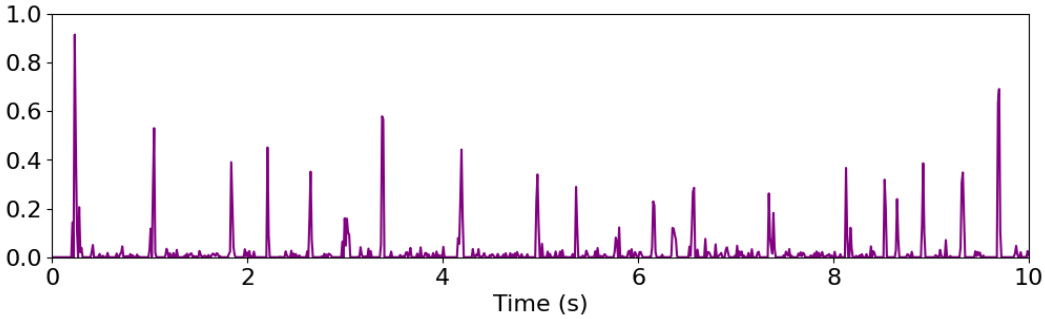

Figure 3.7 Spectral novelty function after local normalization ($M = 410,\ f_s = 44100$)

Just like in the first approach, the peaks should indicate note onsets.

3.3 Comparison of approaches

When we look at the resulting novelty functions (which were both computed on the same song part), we can see a very clear difference between the results. Some peaks are in both novelty functions, but some are only in one of them.

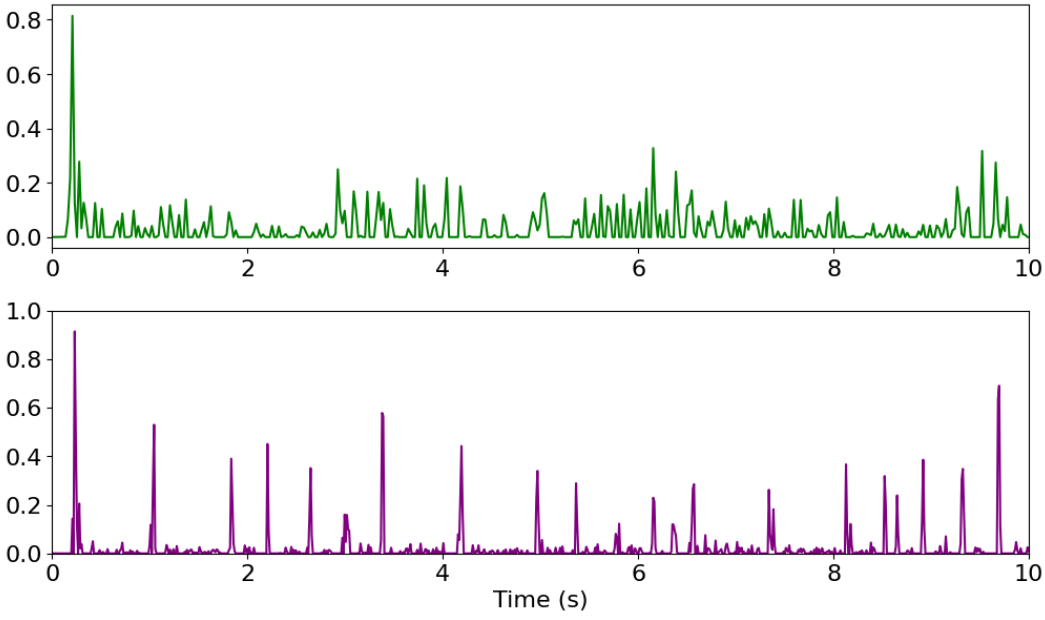

Figure 3.8 Comparison of energy-based and spectral-based novelty functions

The energy-based approach has some disadvantages. One was already mentioned — some musical instruments produce sounds that do not contain as much energy as other instruments. Fo example, bass or percussive onsets contain a lot of energy. But if these instruments are not present, the energy-based approach might not work well.

Another problem is that not all sounds are steady — for example a vibrato. Wih these types of sounds, energy novelty function can look different than we would expect and these note onsets would not be detectable.

Spectral-based approach solves these problems and works better in more situations. Peaks are more distinct and it works better even for instruments with non-steady sound. That does not mean that the energy-based approach is bad — in many cases it will work great (for example in songs with very distinct drums and bass). But spectral-based approach should generally work better.

4 Tempo analysis

Musical tempo, also known as beats per minute, defines the pace of a song. It describes the number of beats that occur per minute in a song. This thesis focuses on songs with constant tempo. We will also assume that beats are aligned with note onsets.

One problem that arises with tempo analysis is that there are various levels of notes that contribute to the overall rhythm. We then have to decide which level is the level where tempo should be detected. There is for example the measure level, tactus level or tatum level. Depending on which level we choose, the resulting tempo will be different. But generally all of the levels should result in numbers that are multiples of one another — for example 152 BPM and 76 BPM. 152 is a multiple of 76 and both of those numbers could be a correct tempo representation of the same song.

Tempograms

For tempo analysis, we will use tempograms, specifically discrete tempograms:

$$\mathcal{T} : \mathbb{Z} \times \Theta \to \mathbb{R}_{\geq 0}$$

Since tempograms indicate tempo relevance for each time instance in a song, they need some input time-based signal. That is where novelty functions will be used, because they enhance note onsets. Beats should mostly align with some note onsets and beats and tempo are closely related.

We will try two different approaches — using Fourier tempogram and autocorrelation tempogram. We will start with a discrete-time novelty function where peaks indicate note onsets (or rather note onset candidates). Since spectral novelty function should work better in more instances, it will be the novelty function used in all of the provided illustrations. But the steps would be the same for other novelty functions.

4.1 Fourier-based approach

The idea behind the Fourier-based approach is to detect local periodicities by comparing the novelty function with windowed sinusoids. We need to compute correlation between a section of the novelty function and the windowed sinusoid. High correlation indicates a periodicity of the sinusoids frequency. To compute this correlation, we can use short time Fourier transform (STFT).

4.1.1 Fourier tempogram

To understand the Fourier tempogram, we must first introduce the Fourier coefficient.

Discrete Fourier tempogram

The discrete Fourier tempogram is a function

$$\mathcal{T}_{\text{Fourier}}(m, \tau) := |\mathcal{F}(m, \tau/60)|$$

where τ is tempo value in BPM. $\mathcal{F}$ is the complex Fourier coefficient:

$$\mathcal{F}(m, f) := \sum_{t \in \mathbb{Z}} \Delta(t) w(t - m) \mathrm{e}^{-2\pi \mathrm{i} f t}$$

where Δ is the input novelty function.

In practice, the Fourier tempogram is computed only on some defined interval of reasonable tempi, because some BPM values (too high or too low) don't make sense in the musical sense.

The Fourier tempogram $\mathcal{T}_{\text{Fourier}}$ reveals the dominant tempo over time. The dominant tempo can be seen in the visualization of the tempogram:

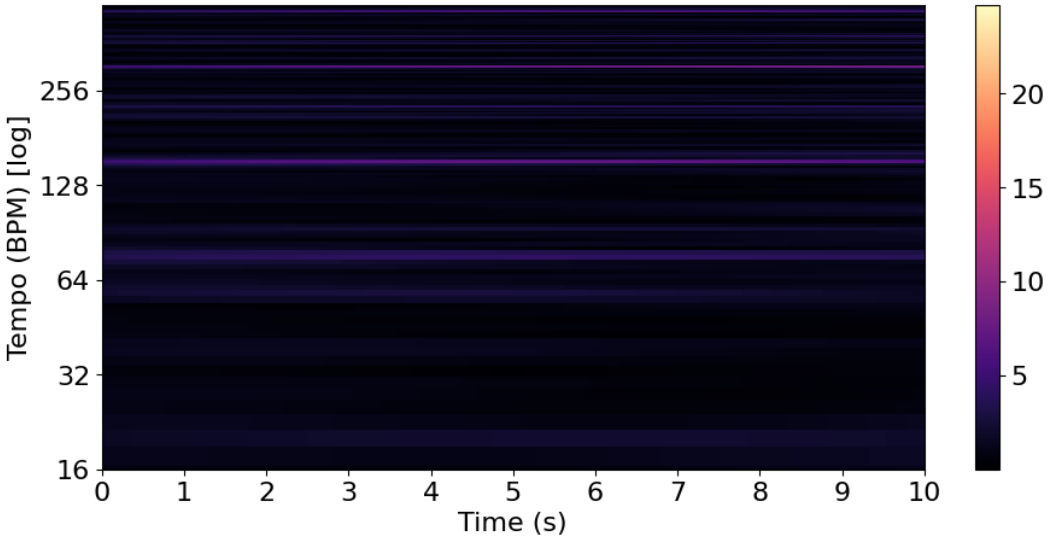

Figure 4.1 Fourier tempogram

For better visualization of dominant tempi, we can also sum up the Fourier tempogram for each BPM value and inspect this function. A higher value indicates a more dominant tempo.

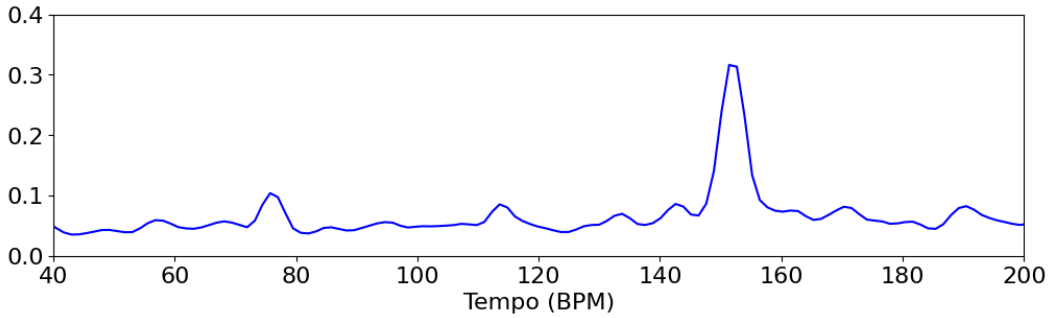

Figure 4.2 Fourier tempogram results summed up for each BPM value after normalization

We can see that the dominant tempi are at 76 BPM and 152 BPM. Both of these values make sense based on the note level we decide to use. And both are correct for the input song.

4.2 Autocorrelation-based approach

The second approach for tempo estimation is the autocorrelation-based approach.

4.2.1 Autocorrelation

Autocorrelation measures the similarity between a signal and a time-shifted version of it. Mathematically, autocorrelation $A_{xx} : \mathbb{Z} \to \mathbb{R}$ of a discrete-time signal $x : \mathbb{Z} \to \mathbb{R}$ is defined as

$$A_{xx}(\ell) = \sum_{m \in \mathbb{Z}} x(m)x(m - \ell).$$

This yields a function dependent on a lag parameter $\ell \in \mathbb{Z}$. It is maximal for $\ell = 0$ and symmetric in ℓ. The maximum in $\ell = 0$ results from the fact that the time-shift in this case is 0, so we are computing similarity of two identical functions.

4.2.2 Short-time autocorrelation

To obtain some information about tempo, we need to apply autocorrelation to the novelty function $\Delta : \mathbb{Z} \to \mathbb{R}$ locally — in the neighborhood of some time parameter. To do that, we will use a window function $w : \mathbb{Z} \to \mathbb{R}$. We need to define a windowed version of the novelty function Δ:

$$\Delta_{w,n}(m) := \Delta(m)w(m - n),$$

where w is the window function and n is the time parameter.

From that, we can define short-time autocorrelation $STA : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$. We can obtain it by computing the autocorrelation of $\Delta_{w,n}$:

$$STA(n, \ell) := \sum_{m \in \mathbb{Z}} \Delta_{w,n}(m)\Delta_{w,n}(m - \ell) = \Delta(m)w(m - n)\Delta(m - \ell)w(m - n - \ell)$$

By computing short-time autocorrelation of the novelty function Δ, we will obtain a time-lag representation of it. The parameter n is called a frame parameter and ℓ is a lag parameter.

4.2.3 Autocorrelation tempogram

Now, we need to convert the time-lag representation into time-tempo representation. To do that, we have to convert the lag parameter ℓ into a tempo parameter τ. We will need the sampling rate f_s for that. The formula for computing tempo from lag ℓ (given in frames) is then simple:

$$\tau = \frac{60 \cdot f_s}{\ell} \text{ BPM}.$$

The time-lag representation can look for example like the first function in the following visualization. Higher value indicates higher autocorrelation for the corresponding lag value.

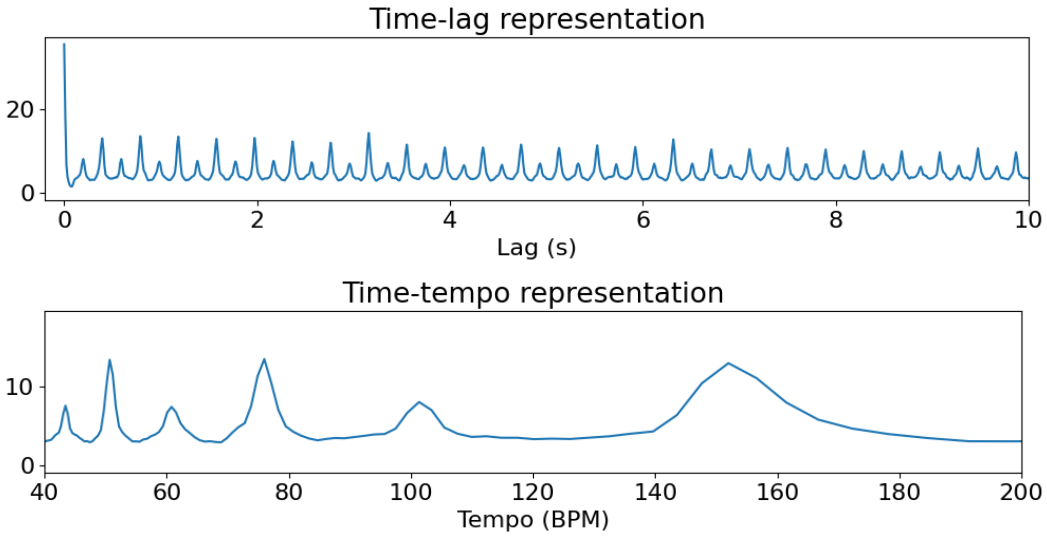

Figure 4.3 Time-lag and time-tempo representation

A higher value in time-tempo representation indicates that the corresponding tempo is more dominant. This is what is needed to obtain information about dominant tempi, but we will still introduce the autocorrelation tempogram.

The autocorrelation tempogram $\mathcal{T}_{Autocorrelation}$ is defined as

$$\mathcal{T}_{Autocorrelation}(n, \tau) := STA(n, \ell) = STA\left(n, \frac{60 \cdot f_s}{\tau}\right).$$

Just like in the previous approach, the autocorrelation tempogram reveals dominant tempo over time, which can be seen in the tempogram visualization:

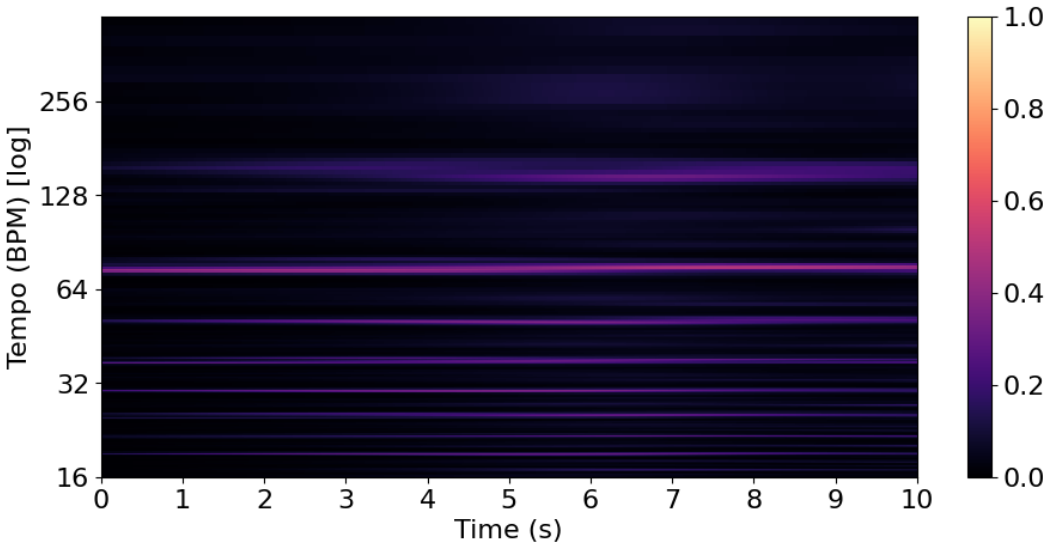

Figure 4.4 Autocorrelation tempogram

The dominant tempo can be seen either from the tempogram visualization or the time-tempo representation.

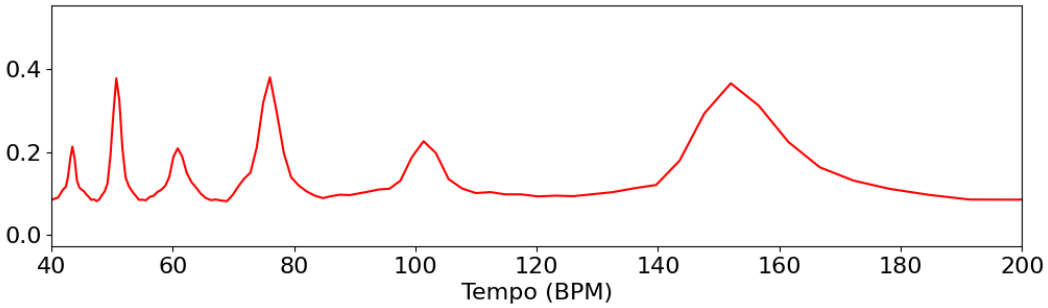

Figure 4.5 Time-tempo representation (normalized)

In this case, the most dominant tempi is at 75 BPM, which is almost correct. Then there are two other very high peaks but they are also not correct. Although the values are very close to the correct tempi (76 and 152 BPM), there is a small error.

4.3 Comparison of approaches

Both approaches were applied on the same song. Fourier tempogram had the best results, because the first two most dominant tempi were correct. But the autocorrelation-based approach also gave good results.

The autocorrelation tempogram is generally less accurate. Although the most dominant tempo is often very close to the correct value, it is rarely completely correct. The Fourier tempogram is more reliable. We will look into this more in the experimental part.

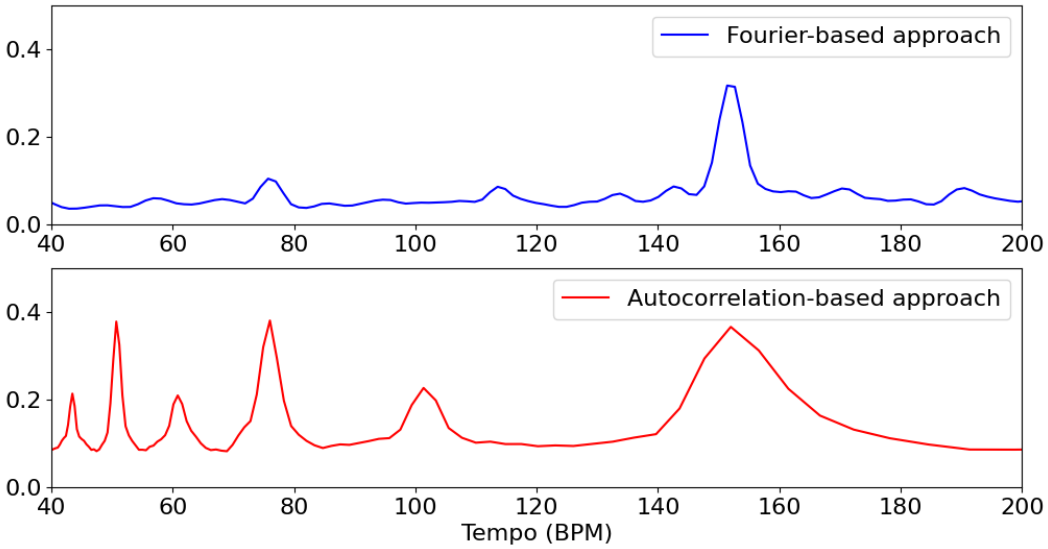

Figure 4.6 Comparison of approaches

5 Beat detection

Beat is the element that drives music forward. It is specified by two parameters: the phase and the period. The period p is given by the reciprocal of the tempo. The phase s then specifies a time shift. The beat function is a sinusoid with period p shifted by s.

We will assume that beat positions align with the strongest note onsets. And (as was previously stated), we will assume that the tempo is constant throughout the whole song.

5.1 Period and phase

5.1.1 Period

Period specifies the duration between two consecutive beats. It can be calculated directly from tempo (for songs with constant tempo). Since tempo was already obtained in the previous step, we can easily compute the beat period p as

$$p = \frac{60}{\tau}$$

where τ is tempo in BPM. The duration of beat period p is specified in seconds.

5.1.2 Phase

The beat phase specifies the shift of the sinusoid from 0. Since the beat function is a function of time, the phase s specifies a time shift. Intuitively, if we were to start a metronome set to the previously found tempo τ after a time shift s, its clicks would align with the song beats.

Time shift detection will be one of the main focuses of this chapter. But before we can start calculating the time shift, we need to extract dominant peaks.

5.2 Peak picking

Peaks in novelty function should indicate note onset candidates. Because our goal is to detect beats, we don't need to detect all note onsets. In theory, main beats should be indicated by more dominant peaks in novelty function.

Since we are interested primarily in beat detection, we should extract N note onset candidates so that the number N is the same or higher than the expected number of beats. We want to pick at least N onset candidates, where $N > \frac{\tau}{60} \cdot L$, where L is the song duration in seconds.

5.2.1 Strongest peaks

The first and most straightforward approach would be to just take the N strongest peaks. But this approach has one major problem — different song parts (for example chorus) have higher sound intensity. That would mean that

we would pick onset candidates mostly from louder parts and we will not have many candidates from other parts of the song.

5.2.2 Local strongest peaks

A better approach would be to partition the song into smaller sections and then apply the first approach on each section separately. This will result in more evenly distributed onset candidates, which should be more useful for beat detection.

The following visualization shows this approach applied to the first 15 s of spectral novelty function using partitions of 5 s.

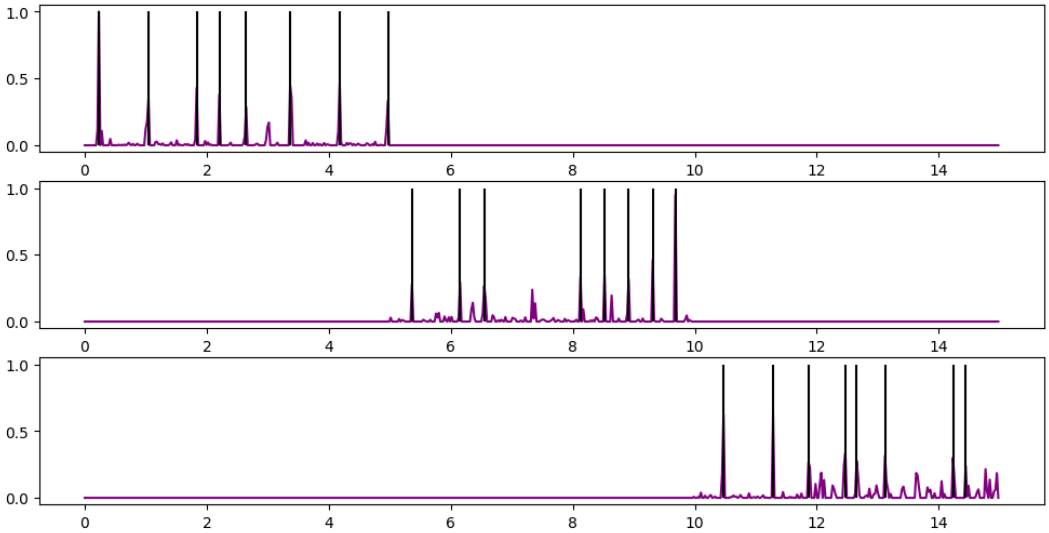

Figure 5.1 Detecting peaks in normalized spectral novelty function in partitions

5.3 Finding time shift

The beat phase specifies the shift of the sinusoid from 0. Since the beat function is a function of time, it specifies a time shift.

We will try three different approaches for phase calculation — based on score, based on score with penalty and based on the phase of tempo frequency.

We will also consider only time shifts in the interval $s \in [0, p]$. So we won't take into account that there might be a longer time interval before the actual start of the song, the possible time shift values are from 0 to the duration between two consecutive beats.

5.3.1 Based on score

The first approach is very intuitive. We will try all possible time shifts (or not all, but those that make sense in our case), give each one a score and then pick the one with the highest score.

We will start with 0 second time shift. Then we will start increasing the time shift by 1 millisecond until we reach the time shift of p seconds. We will calculate score for each of those time shifts and then pick the one with the highest score.

Time shift

We have a finite set of time shifts that we need to go through. We will calculate score for each time shift $s_i,\ i \in \mathbb{N}, i \in [0, p \cdot 1000]$. We multiply p by 1000, because we consider time shifts at multiples of 1 ms.

Click track

For calculating score we will use a click track. That is an audio signal that is 0 everywhere except for click times. Those click times are periodic with period p. So it is like a metronome set to a tempo τ.

In reality, we don't need the actual click track, it is just much easier to picture the calculations with it. But we will only use the click times in our calculations. These click times will be shifted by s_i milliseconds and then score will be calculated.

Let's call this click track $C = C_0, C_1, \ldots, C_{B-1}$, where B is the number of beats that can fit in the song duration with phase 0 and period p.

Tolerance interval

We will need to define a tolerance interval of size I, that will be used to calculate score. It will specify how close a note onset candidate needs to be to some click time so that it gets awarded a score point.

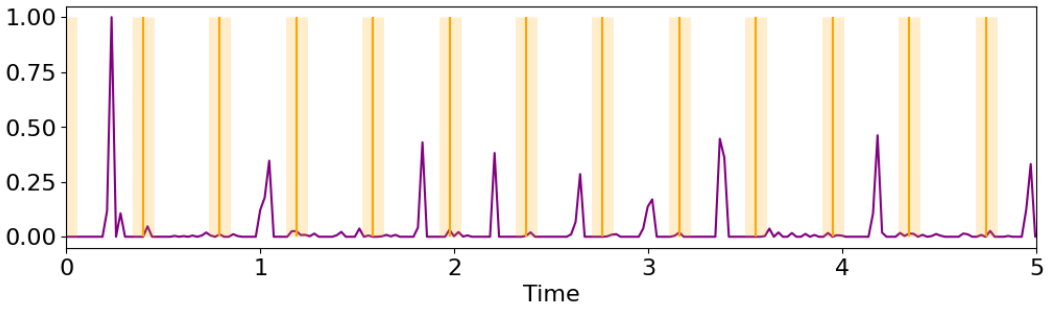

Figure 5.2 An example of click times (not shifted) with their tolerance intervals

Score

We need to calculate a score for each time shift s_i. The idea is that we will increase the click times by s_i milliseconds and then calculate their score S_i.

We will award 1 score point to each click time $C_b, b = 0, 1, \ldots, B-1$ shifted by s_i that is close to some note onset candidate. Then we will sum up the scores for all click times to get score S_i. After trying all possible time shifts, we will use the one with the highest score. Mathematically:

$$S_{i,b} = \begin{cases} 1 & \text{if } |(C_b + s_i) - \mathcal{O}_b| \leq I \\ 0 & \text{otherwise} \end{cases},$$

where C_b is the b-th click time, $C_b + s_i$ is its shifted version and $\mathcal{O}_b$ is the closest note onset candidate to $C_b + s_i$. The overall score S_i for a time shift s_i is simply

the sum of all $S_{i,b}$:

$$S_i = \sum_{b=0}^{B-1} S_{i,b}$$

The idea behind this approach can be seen from the following illustration, where shifted click times that have a note onset candidate (black) in their tolerance interval are highlighted with red:

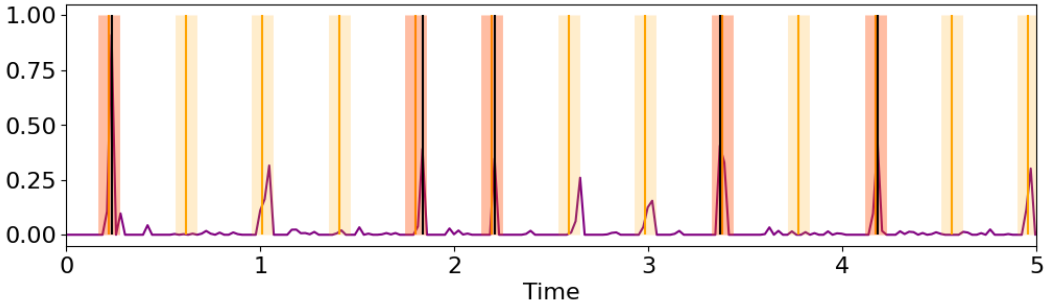

Figure 5.3 Shifted click times

The most important parameter in this approach is the size of the tolerance interval I. If the chosen size is too small or too big, the approach might not find the correct time shift. A good tolerance interval is around 10–20 ms, because it allows for some small fluctuations in beat times, but if the result is shifted by some milliseconds, human ears should not pick up on that.

Larger tolerance interval might result in a larger error in time shift, that would be audible to humans. Smaller tolerance interval might be too strict when awarding score points, because some small fluctuations are almost always present (for example because of different musical instruments).

5.3.2 Score with penalty function

The first approach is very straightforward — if there is a peak in the tolerance interval around a time click, we increase the score by 1. But shouldn't we award more points when a time click is closer to a note onset candidate? It is definitely better to have an onset candidate that is only 1 ms far than if it was 10 ms far. And what about onset candidates that are just outside of the tolerance interval? They are still "better" than those that are even further, but they all get awarded 0 points.

The second approach tries to solve these problems. Instead of simply awarding 0 or 1 score point, it awards a number of points based on how far the closest note onset candidate is from current time click. The closer the candidate is, the more points it gets. To do that, we will use a penalty function $\mathcal{P}_p$.

Score

The score $S_{i,b}^{\mathcal{P}_p}$ for a shifted time click $C_b + s_i$ will be calculated as:

$$S_{i,b}^{\mathcal{P}_p} = 1 - \mathcal{P}_p(d),$$

where d is the distance between the shifted time click $C_b + s_i$ and the closest onset candidate $\mathcal{O}_b$, so

$$d = (C_b + s_i) - \mathcal{O}_b.$$

The total score $S_i^{\mathcal{P}_p}$ for a time shift s_i will again be the sum of scores for each shifted time click.

Penalty function

The penalty function $\mathcal{P}_p$ will be used to penalize each shifted click time based on how far the closest note onset candidate is. The closer a candidate is to a shifted click time $C_b + s_i$, the higher the score should be. The penalty function $\mathcal{P}_p$ is dependent on the beat period p (that is why it has the subscript p). The function will be even ($\mathcal{P}_p(d) = \mathcal{P}_p(-d)$) and minimal in 0, so that the smallest distance between a time click and a candidate (0) has the lowest penalty. Larger distances should be penalized more until some specified distance, where the penalization will reach its maximum. The maximum penalization is 1, so that the resulting score is never less than 0.

We could use many different penalty functions, the only requirement is that smaller distances will get smaller penalization. This is true for example for a quadratic function. If the distance d to the closest note onset candidate is greater than half of the beat period p, the shifted time click will be get a penalization of 1, so its score will be $S_{i,b}^{\mathcal{P}_p} = 1 - 1 = 0$. Otherwise, it will get a penalization $\mathcal{P}_p \in [0, 1]$. More precisely:

$$\mathcal{P}_p(d) = \begin{cases} d^2 \cdot \frac{4}{p^2} & \text{if } d \leq \frac{p}{2} \\ 1 & \text{otherwise} \end{cases}$$

The multiplication by $\frac{4}{p^2}$ will result in normalization of the penalty, so that the value is between 0 and 1.

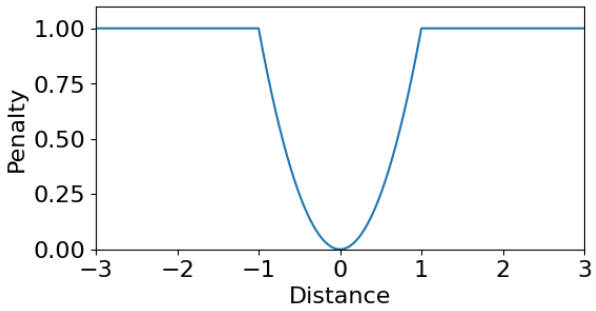

Figure 5.4 Penalty function

This function is clearly even, minimal in 0 and penalizes larger distances more than smaller distances.

5.3.3 Phase of the tempo frequency

This approach is based on the assumption that a song is just a sum of different sine waves of different frequencies. We will try to use the frequency of the already found tempo.

The tempo τ is defined in BPM. If we convert it to Hertz, we can find the phase of this frequency f_τ throughout the whole song and try to use it as a phase of beats.

Unfortunately, after testing this approach, the results were not correct. One reason for that might be that the found tempo wasn't precise. There could be some small error — too small to be detectable in terms of BPM, but still significant during the phase calculation. Another reason could be that the tempo frequency f_τ often fell in the middle of some frequency bin and not at the start. Because of that it was impossible to calculate the precise phase of it.

5.3.4 Comparison of approaches

The third approach didn't work at all, so it is clearly the worst approach out of the three. Because it didn't work, it won't be shown in the experimental part.

When we look at the first two approaches, the quadratic penalty function should be an improvement of the first approach. The penalty function ensures that the score given is proportional to the distance between the closest onset candidate and a time click. So it should perform better than a simple 0 or 1 like in the first approach and the results should be more precise.

However, the first approach works quite well if the tolerance interval is chosen wisely. It might not find the best time shift, but if the error is so small that it is not noticeable by human ears (for example around 10 milliseconds), then we can say that the result is somewhat correct. We usually need the result to be perceived as correct by humans, so a tiny error is insignificant in those cases.

6 Rhythm detection

Rhythm is the pattern of sounds, silences, and emphases in a song. It is quite complicated to detect rhythmic patterns automatically. So we will make the task a little bit easier — instead of trying to find complicated patterns, we will focus on strong note onsets that are frequently occurring after beats approximately after the same time as other note onsets. To do this, we will again use score.

In this chapter, the song used for visualizations is Spark by Vexento, since it has a simple rhythm.

6.1 Calculating score

We will consider peaks in novelty function and try to find peaks such that their distance from the closest preceding beat is approximately the same as for other peaks throughout the song. We will consider distances at multiples of 1 ms in the interval $d \in [0, p * 1000]$.

Just like with beat detection based on score, we will define a tolerance interval and we will assign one score point to all of the distances in the tolerance interval of a currently processed distance. This tolerance interval should once again be very small — around 10–20 milliseconds.

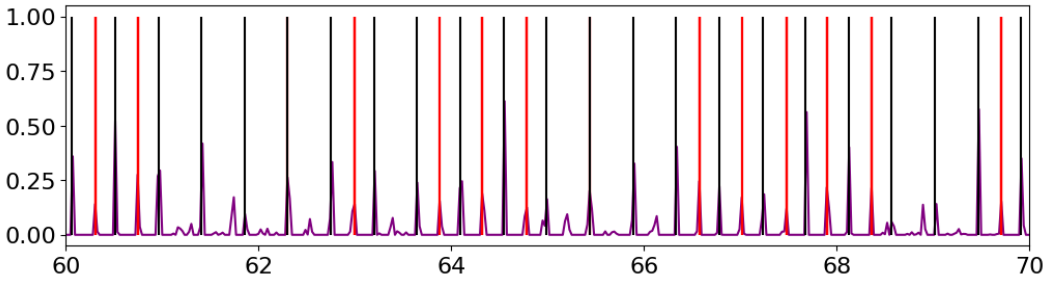

Figure 6.1 Very simple rhythmic pattern drawn over spectral novelty function — beats are black, other rhythmic peaks are red

6.2 Song partitioning

Scoring will not be calculated for the whole song at once. It will be calculated on smaller chunks of the song. We should get the best results if we calculate score separately for verses, choruses and other parts of the song. But verse and chorus detection is not an easy task.

6.2.1 Verse and chorus detection

One possibility for verse and chorus detection is to use root mean square energy. But in this case, instead of computing it as an instantaneous feature (in terms of milliseconds) we will compute it as a segment-level feature (in range of seconds). This will give us information about energy of longer parts of the song.

In theory, verse should have lower energy than chorus, so there should be very clear changes in the energy function when going from a verse to a chorus.

This can be clearly seen in the following illustration. We can see a very clear pattern, with two high-energy parts (choruses) and two low-energy parts (verses):

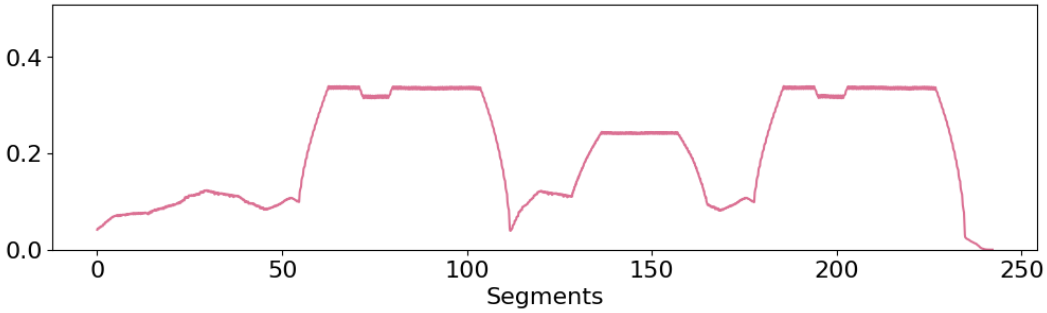

Figure 6.2 Result of RMSE calculated for 8 s segments

To detect the transitions between different song parts, we can compute discrete derivative. Then we will take the absolute value of the result, because we want to detect both verse-chorus (low-high) and chorus-verse (high-low) transitions.

Significant peaks in the derived function should indicate transitions between different song parts:

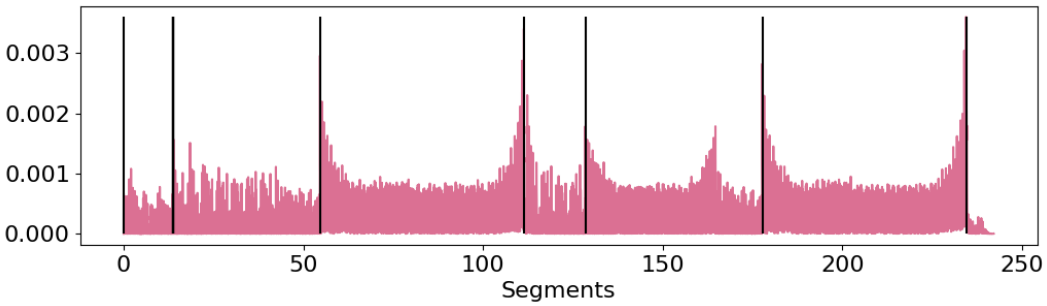

Figure 6.3 Peaks in the discrete derivative of segment-level RMSE

If we go back to the original graph before applying discrete derivative, we can see that our approach worked quite well:

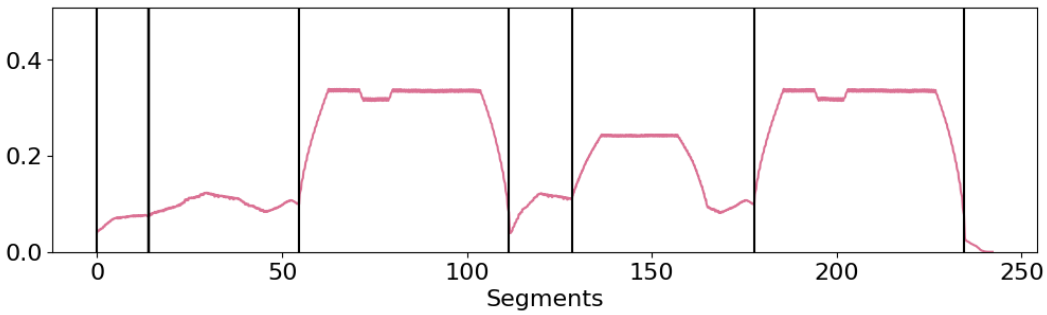

Figure 6.4 Song divided into chorus and verse parts

It is important to mention, that in reality, songs have other parts than just choruses and verses, like bridge. For simplicity, these were not mentioned in the

description of this approach. But in reality, this approach should work the same no matter what the song part is — the important thing is that each part has significantly different energy.

Unfortunately, this approach has many flaws. It relies on the assumption that there will be distinct energy changes between parts, but that is not always true. That means that with songs where energy fluctuations are not distinct, this approach will not work. In that case, we will try a different approach.

6.2.2 Parts of predefined length

If verse and chorus detection fails, we will simply divide the song into smaller chunks of predefined length. The length shouldn't be too small or too big, otherwise the scoring will not give any useful information. Smaller tens of seconds should be an optimal size.

Then, we will calculate score for each part separately and highlight peaks with the highest scores just like in the previous approach.

6.2.3 Comparison of approaches for song partitioning

In this case, comparing the approaches does not really make sense. It should be better to use the first approach (based on verse and chorus detection) if it detects song parts, but that will not work in most cases. The second approach will work for all songs, so it is more flexible.

Overall, rhythm detection is a very complicated task. The described approaches work for songs with very simple rhythmic patterns, but they do not return good results for more complicated songs.

7 Experimental results

This chapter will focus on implementation and experimental results of all of the described approaches. All of the approaches were tried out and tested on different songs.

7.1 Implementation

The code for rhythm recognition is delivered as a Python package. For the implementation, I extensively used the Librosa package [9], as was already mentioned. This package is used mostly for audio file loading and for tempogram related computations.

7.1.1 RhythmRecognition package

The provided package is called `RhythmRecognition`. Each step — onset detection, tempo analysis, beat detection, and rhythm detection — is implemented in a separate subdirectory — `onset, tempo, beat, rhythm`. The implemented approaches in each step are a straightforward implementation of the approaches described in the previous chapters.

All of the subdirectories have a very similar structure. There is base class defining the interface and common methods for each step and then there are classes inheriting from this base class that are implementing a specific approach.

Onset detection

Onset detection offers two approaches — energy-based and spectral-based approach. They are implemented in `EnergyNovelty` and `SpectralNovelty` classes with base class `NoveltyFunction`. The usage is very simple — after creating an instance of the specific class, we can simply call the `get()` method to obtain the computed novelty function. When creating an instance, we need to pass the audio file name as an argument.

Tempo analysis

Tempo analysis offers Fourier-based and autocorrelation-based approach. They are implemented in `FourierTempogram` and `AutocorrelationTempogram` classes with base class `Tempogram`. Depending on if we want to get only the most dominant tempo value or the whole tempogram, we can either call `get_tempo()` or `get_tempogram()` on a specific tempogram instance. When creating an instance, we need to pass a novelty function as an argument.

The `get_tempo()` method works as follows: it takes first few most dominant tempo values from a tempogram (default is first 5 values) and groups them into groups of similar values. Each value also gets assigned a weight based on how dominant it was in the tempogram. The group with the highest weight is selected and its weighted average is rounded to the nearest integer. This value is returned as the most dominant tempo.

Calculating weighted average helps in cases when the most dominant tempo value is close to the correct value, but not completely correct. It takes into account other dominant values, which are usually close to the correct value as well. Rounding up to the nearest integer also helps, because tempo is usually a natural number.

Beat detection

Beat detection offers two approaches — based on score and based on penalty. They are implemented in classes `ScoreBeatTracker` and `PenaltyBeatTracker` with base class `BeatTracker`. If we want to get the computed beat track for a given song, we can call `get_beat_track()` on a beat tracker instance. If we need just the computed time shift, we can call `get_time_shift()`. When creating an instance, we need to pass a novelty function and the song tempo as an argument.

In the `beat` section, there are also other modules used: `peak_picking.py` and `click_track.py`. The first module, `peak_picking.py`, offers methods for peak picking, which is used for extracting note onset candidates from novelty function. The second module, `click_track.py`, handles generating the click tracks that are used for beat detection.

Rhythm detection

Rhythm detection offers two approaches — by partitioning song into equal parts or by identifying chorus, verse and other song parts. They are implemented in classes `EqualPartsRhythmTracker` and `ChorusVerseRhythmTracker` with the base class `RhythmTracker`. If we want to obtain the computed rhythmic track, we can call `find_rhythmic_onsets()` on a rhythm tracker instance. When creating an instance, we need to pass a novelty function, the song tempo and its beat track as an argument.

7.1.2 Wrapper methods

Since the described steps need results of the previous steps as an argument, the usage might not be ideal. That is why there are wrapper methods provided as well. They simply take the song file name and compute everything inside, so that we do not have to compute each step separately. It is possible to specify which approach should be used in each step and it is also possible to change the default parameter values if it is needed.

These wrapper methods can be found in the file `detect.py`. The following methods are offered: `novelty_function()`, `tempo()`, `tempogram()`, `beat_track()`, `beat_time_shift()`, `rhythm_track()`. The only mandatory argument is the audio file name.

7.1.3 Other parts

Apart from the already described contents of the package, there are also other modules, for example a file with simple math functions (where things like logarithmic compression are implemented) or a file with constants (containing for example the frame size and hop length).

7.1.4 Parameters

The provided approaches are dependent on many parameters. Everything is written in a way that the user can change the parameters based on their needs, but there are default values set in most cases.

7.1.5 Usage

Usage of the provided package is very simple. The installation and usage is described in the provided `README.md` file. The file `requirements.txt` contains a list of required packages.

Documentation and examples

There is documentation provided for the whole package. It was generated using `pdoc` [12], which is an automatic documentation generator. There are also Jupyter Notebooks showing example usages of the provided code. The notebooks can be found in the `examples` directory.

7.2 Experiments

All of the described approaches were implemented and applied on different songs. The experiments were conducted on 31 songs of different genres and tempi. Some of the songs do not have constant tempo throughout their whole duration (for example the beginning might be slower). In that case, a cropped version of the song was used to satisfy the constant tempo requirement.

There will be some tables with results displayed, where there will be song name and song artist. Sometimes, there are more artists than just the one mentioned, but there is always just one artist mentioned for each song. This is only for better readability of the tables.

Some of the results are easy to test automatically, others are more complicated. Testing of the four main steps — onset, tempo, beat, and rhythm — will be described more thoroughly in the following sections.

Parameters

As was already mentioned, the provided methods are dependent on many parameters. I used the default values as they are set in the provided package. I will mention the values of the most important parameters that I used:

- sampling rate — 44 100 Hz,
- frame size — 2 048 samples,
- hop length — 512 samples.

All of the steps were tested extensively with different parameters. The provided configuration returned the best results.

Code

The code with tests and results can be found in the `tests` directory.

7.2.1 Onset detection testing

There are no tests specifically for testing onset detection. The results of onset detection are used in all other steps, so we can compare the approaches for onset detection by comparing the number of correctly detected tempi or by listening to found beats and rhythmic patterns. So we will decide later which of the two approaches was better based on the results of the other steps.

7.2.2 Tempo analysis testing

Tempo is the easiest step to test out of the four main steps. Each song has constant tempo and this value can be easily obtained — I used a combination of online resources and tap tempo tools. So we just need to compare the correct value with the value that the program will return.

Results

The following table shows results of testing for both tempograms and both novelty functions. The first two columns contain the song and the artist name, then the actual tempo in BPM is displayed and after that the results of the Fourier-based approach and autocorrelation-based approach computed over energy-based and spectral-based novelty functions are displayed.

Cell color indicates whether the calculated BPM was correct, close to the correct value or completely wrong. Correct values are those that are a multiple of the actual tempo or whose multiple is the actual tempo. They are indicated by a green cell color.

Yellow cell color indicates values that are very close to the correct value (with the difference of 1 or 2 BPM).

Red color indicates wrong results.

			Energy novelty		Spectral novelty	
Name	Artist	BPM	Fourier	Autocorr.	Fourier	Autocorr.
Alone	Alan Walker	97	194	48	194	185
Around the World	ATC	132	132	128	132	129
Baby Shark	Pinkfong	115	115	113	115	112
Beautiful Life	Ace of Base	135	135	131	152	67
Believer	Imagine Dragons	125	125	122	125	122
Call Me Maybe	Carly Rae Jepsen	120	120	117	180	117
Can't Stop the Feeling!	Justin Timberlake	113	113	111	113	111
Don't Speak	No Doubt	76	75	75	152	147
Faded	Alan Walker	90	180	88	180	89
Fight Song	Rachel Platten	176	176	44	176	86
Firework	Katy Perry	124	124	121	124	121
Hollaback Girl	Gwen Stefani	110	82	72	165	72
I'm Still Standing	Elton John	177	178	87	178	87
It's My Life	Bon Jovi	120	120	59	120	117
Love You Like a Love Song	Selena Gomez	117	117	114	117	114
Mambo No. 5	Lou Bega	174	174	86	174	167
On My Way	Sabrina Carpenter	170	127	42	128	42
Seven Nation Army	The White Stripes	124	123	121	124	61
Shake It Off	Taylor Swift	160	120	105	160	79
Shape of You	Ed Sheeran	96	144	48	144	127
Spark	Vexento	117	117	115	117	58
Stereo Hearts	Gym Class Heroes	90	180	89	180	45
Temple of Love	The Sisters of Mercy	166	166	82	166	82
The Nights	Avicii	126	126	122	190	62
Thunder	Imagine Dragons	168	105	55	168	42
Thunder	Gabry Ponte	135	135	131	135	133
Uptown Funk	Mark Ronson	115	115	113	115	113
Valhalla Calling Me	Miracle of Sound	152	152	149	152	75
We Will Rock You	Queen	81	163	40	163	40
Without You	Avicii	134	134	131	134	131
Y.M.C.A.	Village People	127	127	124	127	63

Table 7.1 Most dominant tempo obtained from the two tempogram approaches computed over energy-based novelty and spectral-based novelty functions.

The results are very clear — the Fourier tempogram detected the correct tempo (or almost the correct tempo) in most cases. The autocorrelation tempogram did not work in many cases. The most dominant tempo value was often very close to the correct value, but there was usually a small error.

The following table compares the results of the two tempograms computed over the two different novelty functions.

Tempogram	Correct results energy novelty	Correct and almost correct results energy novelty	Correct results spectral novelty	Correct and almost correct results spectral novelty
Fourier	22/31	26/31	23/31	25/31
Autocorrelation	2/31	15/31	2/31	16/31

Table 7.2 Comparison of tempogram approaches computed over different novelty functions.

When we look at the number of correct results, we can see how much more precise the Fourier tempogram is in comparison with the autocorrelation tempogram. It is the clear winner in all cases. Since it gives the best results, Fourier tempogram will be used in beat detection testing and rhythm detection testing.

When we look at novelty functions results, it is harder to decide which one was better. Both novelty functions performed very similarly. That means we will have to use both novelty functions in the following steps and compare the results once again.

7.2.3 Beat detection testing

Beat detection testing is more complicated than the previous step. But since the testing is done on a small amount of songs, I decided to conduct beat detection testing by simply listening to the song with the generated beat track over it. Small errors in the range of milliseconds are not detectable by human ears, so I was not able to detect these errors. But it does not matter, because the results should be used by humans anyway so even if there was a slight error, no one would notice.

Beat detection is dependent on the given song tempo. Just for the beat testing phase, we will use the correct known tempo even if the tempo analysis approaches could not find the correct value.

The following table contains results of the conducted testing. There are three possible values: ok, half and no. “ok” means that the beat track was aligned with the song beats, “no” indicates that the beat track was shifted completely wrong. The last option, “half”, indicates that the beat track was shifted in a way that it was right between the song beats. That is not correct, but it sometimes happened with songs that had twice as high tempo detected in tempo analysis. If the beat track was computed using the found tempo, it would be aligned correctly. That is why I decided to add this third category, because it is not correct but it might be correct if the calculated tempo value was used.

		Energy novelty		Spectral novelty	
Name	Artist	Score	Penalty	Score	Penalty
Alone	Alan Walker	half	yes	half	half
Around the World	ATC	half	half	half	half
Baby Shark	Pingfong	yes	half	yes	yes
Beautiful Life	Ace of Base	yes	yes	yes	half
Believer	Imagine Dragons	yes	yes	yes	yes
Call Me Maybe	Carly Rae Jepsen	yes	yes	yes	half
Can't Stop the Feeling!	Justin Timberlake	yes	yes	yes	yes
Don't Speak	No Doubt	yes	yes	yes	yes
Faded	Alan Walker	yes	yes	half	yes
Fight Song	Rachel Platten	yes	yes	yes	yes
Firework	Katy Perry	yes	yes	yes	half
Hollaback Girl	Gwen Stefani	yes	yes	yes	yes
I'm Still Standing	Elton John	yes	yes	yes	yes
It's My Life	Bon Jovi	yes	yes	yes	yes
Love You Like a Love Song	Selena Gomez	yes	yes	yes	yes
Mambo No. 5	Lou Bega	yes	yes	yes	yes
On My Way	Sabrina Carpenter	yes	yes	yes	yes
Seven Nation Army	The White Stripes	yes	no	no	half
Shake It Off	Taylor Swift	yes	no	yes	half
Shape of You	Ed Sheeran	half	half	yes	no
Spark	Vexento	yes	yes	yes	yes
Stereo Hearts	Gym Class Heroes	yes	yes	yes	yes
Temple of Love	The Sisters of Mercy	yes	yes	yes	yes
The Nights	Avicii	yes	yes	half	half
Thunder	Imagine Dragons	yes	yes	yes	yes
Thunder	Gabry Ponte	half	half	yes	yes
Uptown Funk	Mark Ronson	yes	yes	yes	yes
Valhalla Calling Me	Miracle of Sound	yes	yes	yes	yes
We Will Rock You	Queen	half	half	no	yes
Without You	Avicii	yes	yes	yes	yes
Y.M.C.A.	Village People	half	half	half	yes

Table 7.3 Results of beat detection approaches computed over different novelty functions.

The following table compares the results of the two beat detection approaches computed over the two different novelty functions:

Beat tracking approach	"ok" results energy novelty	"ok" and "half" results energy novelty	"ok" results spectral novelty	"ok" and "half" results spectral novelty
Score	25/31	31/31	24/31	29/31
Penalty	23/31	29/31	22/31	30/31

Table 7.4 Comparison of beat detection approaches computed over different novelty functions.

We can see that the score-based approach gave better results. As for differences between novelty functions, they again performed very similarly.

7.2.4 Rhythm detection testing

Rhythm detection testing was very similar to beat detection testing. Once again, I listened to the generated rhythmic track played over the song.

There were significant inconsistencies at the song start, especially if there is a silent part at the beginning. The program just randomly added some clicks before the song melody even started. A similar problem sometimes occurred at the start and end of each part. There were sometimes longer sections without any

clicks, probably because the song theme changed during that part and the scoring system did not give enough points to these small sections.

Overall, the approach with equal song partitioning worked better. It worked even for songs with significant energy fluctuations, where the verse and chorus tracking could have been used. Equal parting is also more flexible, so I would say it is the better approach out of the two.

It is almost impossible to classify the results of rhythm detection, since different people might have different opinions on the results. I personally think that the results are good if the song has a very simple rhythmic pattern. An example of a simple rhythmic pattern is when beats are on quarter notes and all eighth notes are played as well. In this case, the program correctly identifies the eight notes and generates a rhythmic track with a click on each eighth note.

But if the song has a complicated rhythm, the described rhythm detection does not give good results.

7.2.5 Discussion

In this chapter, the described approaches for onset detection, tempo analysis, beat detection and rhythm detection were applied on different songs. The results show that many approaches work well.

In onset detection, both approaches worked well. The results for both novelty functions were very similar, so there is no clear winner. The best tempogram approach was the Fourier tempogram. With beat detection, the score-based approach was slightly better, but both approaches worked quite well. And for rhythm detection, equal song partitioning was better.

The fact that the energy-based approach for onset detection worked so well was very surprising for me. I expected that the spectral-based novelty function would perform significantly better. I also expected the penalty-based beat tracking to work better than the score-based approach.

As for tempo analysis, I expected the Fourier tempogram to work better, so that was not surprising.

Overall, the proposed rhythm recognition algorithm works well. Especially for the first three steps (onset detection, tempo analysis and beat detection) — the results are very good and the algorithm detects beats correctly in most cases. The rhythm detection step is not working as well as the previous three steps, but it still gives good results in many cases.

Conclusion

Automatic rhythm recognition in music is a very complex task. We have introduced an algorithm for rhythm recognition in songs with constant tempo. The proposed algorithm always has the same four basic steps: onset detection, tempo analysis, beat detection, and rhythm detection.

We have researched several approaches in each of these steps. For onset detection, we described two different approaches — using the energy-based and the spectral-based novelty function. Each of these functions aims to enhance certain signal properties in order to bring out note onsets. Both novelty functions gave similar results in the experimental part.

For tempo analysis, we computed a tempogram over a novelty function. We tried two tempograms — Fourier-based and autocorrelation-based tempogram. The Fourier-based tempogram returned the best results.

To detect beats in a song, we tried two approaches — based on score and based on penalty. Both of these approaches aimed to find the best time shift so that the beat track would align with the song's beats. The score-based approach was slightly better in the end.

Finally, for the rhythm detection step, we tried two approaches — based on partitioning song into equal parts and based on detecting choruses, verses and other song parts. The equal song partitioning was better, mainly because it was more flexible and worked on all types of songs.

Based on the research, a Python package called RhythmRecognition was created. It is very flexible and it offers methods for each step separately or several steps combined.

Known limitations

The first limitation is, that the algorithm might not work in some cases. There are many complicated computations and if only one is incorrect, the result is not usable. This can be easily detected by listening to the results and it might be possible to change some of the parameters to get better results.

Another limitation is that the algorithm only works on songs with constant tempo. Unfortunately, many songs do not fulfill this requirement.

Rhythm detection also works only on songs with a simple rhythm. If a song has a complicated rhythm, the algorithm might not return good results.

Future work

To improve rhythm detection, it might help to use some machine learning approach. Right now, the programme computes an independent rhythm track each time. But it would definitely be useful if there were some precomputed rhythm tracks that could serve as a template for future rhythm track generation.

Another improvement could be to consider songs with fluctuating tempo. If the algorithm would work on this type of songs as well, the applicability would increase significantly.

Bibliography

1. Phillip, Robert. *The Classical Music Lover's Companion to Orchestral Music*. New Haven: Yale University Press, 2018. ISBN 9780300242720.
2. Shannon, Claude Elwood. Communication in the Presence of Noise. *Proceedings of the IRE*. 1949, vol. 37, no. 1, pp. 10–21. Available from DOI: 10.1109/jrproc.1949.232969.
3. Cooley, James W.; Turkey, John W. An Algorithm for the Machine Calculation of Complex Fourier Series. 1965. Available also from: https://community.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/S0025-5718-1965-0178586-1.pdf.
4. Müller, Meinard. *Fundamentals of Music Processing – Using Python and Jupyter Notebooks*. 2nd. Springer Verlag, 2021. ISBN 978-3-030-69807-2. Available from DOI: 10.1007/978-3-030-69808-9.
5. Müller, Meinard. FMP Notebooks. 2021. Available also from: https://www.audiolabs-erlangen.de/resources/MIR/FMP/C0/C0.html.
6. Mounir, Mina; Karsmakers, Peter; Waterschoot, Toon van. Musical note onset detection based on a spectral sparsity measure. 2021. Available also from: https://asmp-eurasipjournals.springeropen.com/articles/10.1186/s13636-021-00214-7.
7. Alías, Francesc; Socoró, Joan Claudi; Sevillano, Xavier. A Review of Physical and Perceptual Feature Extraction Techniques for Speech, Music and Environmental Sounds. *Applied Sciences*. 2016, vol. 6, no. 5. ISSN 2076-3417. Available from DOI: 10.3390/app6050143.
8. Chen, Bo. Music Audio Rhythm Recognition Based on Recurrent Neural Network. *Wireless Communications and Mobile Computing*. 2022, vol. 2022, pp. 1–11. Available from DOI: 10.1155/2022/6249798.
9. McFee, Brian; *et al. librosa/librosa: 0.10.2*. Zenodo, 2023. Version 0.10.2. Available from DOI: 10.5281/zenodo.8252662.
10. O'Keeffe, Karl. *Dancing Monkeys*. 2003. Available also from: https://monket.net/dancing-monkeys/.
11. Donahue, Chris; Lipton, Zachary C.; McAuley, Julian. Dance Dance Convolution. In: *International Conference on Machine Learning*. 2017. Available also from: https://api.semanticscholar.org/CorpusID:14219890.
12. Gallant, Andrew. *pdoc*. 2024. Available also from: https://pdoc3.github.io/pdoc/.
13. Vomelová, Lucie. *RhythmRecognition*. 2024. Available also from: https://github.com/lucievomelova/RhythmRecognition.

List of Figures

A Attachments

The implementation of the described rhythm recognition algorithm is provided as a zip file. The file contains the RhythmRecognition package, documentation, examples, tests and installation files. The implementation is also available on GitHub [13].

A.1 RhythmRecognition package

The Python package implementing the rhythm recognition algorithm is located in the `RhythmRecognition` directory.

A.2 Documentation

Documentation can be found in the directory `documentation`. It is in HTML format.

A.3 Examples

There are Jupyter Notebooks provided that show example usages of the `RhythmRecognition` package.They are located in the `examples` directory.

A.4 Tests

Tests used in the experimental part can be found in the `tests` directory. The test results are also located there.

A.5 Installation files

There are two files provided for installation. `README.md` contains installation instructions and `requirements.txt` contains a list of required packages.