Computer-assisted manual segmentation of music recordings

An on-the-fly approach to 'sample'-selection for music production

Toningenieur-Projekt

Felix Rothmund

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn Alois Sontacchi

Graz, January 9, 2018

institut für elektronische musik und akustik

Abstract

The goal of this project was to develop a tool that allows musicians to easily select interesting parts of a music recording during playback in order to re-arrange them for new compositions.
The manual selection and cropping of so called samples can be a tedious and often uninspiring task. This project aims to develop a way to select interesting parts in a recording more efficiently and intuitively:
While the user listens to a music recording, he or she marks interesting parts by pressing a key or switch. The computer then corrects the timing to select the parts that the user intended to select and that make sense musically.
The underlying algorithm considers the rhythmic structure of the musical piece, as well as the user's timing.

Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines Algorithmus, dass es MusikerInnen erlaubt, interessante Stellen einer Musik-Aufnahme zu markieren, um diese dann in neuen Kompositionen neu arrangieren zu können.
Das manuelle Auswählen und Schneiden von so genannten Samples kann langwierig und ermüdend sein. Das entwickelte Tool soll es ermöglichen, die Auswahl der Samples intuitiver und effizienter zu gestalten:
Während der Wiedergabe eines Musikstückes markiert der/die MusikerIn interessante Stellen durch das Drücken einer Taste. Der Computer korrigiert dann das Timing, um jene Teile auszuwählen, die der/die MusikerIn tatsächlich auswählen wollte und die musikalisch sinnvolle Einheiten ergeben.
Der verwendete Algorithmus berücksichtigt dabei sowohl die rhythmische Struktur des Musikstückes, als auch die Reaktionszeit der Nutzerin oder des Nutzers.

Contents

1 Introduction

1.1 Problem Description

In today's computer-based music production environments, the process of recording audio material into a Digital Audio Workstation (DAW) and selecting sound segments in order to re-arrange them later, can be tedious and uninspiring, as it usually involves a lot of mouse clicking.
The goal of this project is to develop an assistive tool that makes manual segmentation of music into so called 'samples' intuitive and efficient.

Sampling:
In music production, the term 'sampling' describes the act of re-using pre-recorded audio in a new context. So called 'samples' can be of varying length. For example, short samples can be used to achieve realistic virtual instruments, whereas longer song segments are often used as loops.
In certain genres of music, such as Hip-Hop or House, it is common practice to use sound segments found on records as a foundation for new songs. The most used samples in those genres come from recordings of Soul, Funk and Jazz music of the late 60s and 70s.
However, the techniques of sampling are not limited to pre-recorded music but can be applied to any source of audio [2].

General Idea:
The basic idea was to approach sample selection in a manner similar to looping using so-called loop pedals for electric guitars:
When playing the guitar with a loop pedal, the user hits the pedal at the start of a pattern or riff and hits it again, when the pattern ends.
When using the proposed algorithm, the user segments so-called 'samples' in a similar fashion. Audio clips are selected by pressing and releasing a pad on a hardware controller whenever an interesting audio segment starts and ends respectively.
'Sample' start and end points are then aligned with the desired positions, based on the rhythmic structure of the audio segment.

1.2 Related Work

In previous work at the *Institute of Electronic Music (IEM)*, Daniel Rudrich proposed an algorithm [5] that optimizes loop start and end times for guitar loop pedals based on beat estimation. This work is in great parts based on his master thesis [13].

1.3 Proposed Algorithm

The proposed prototype consists of five stages that can be seen in figure 1.

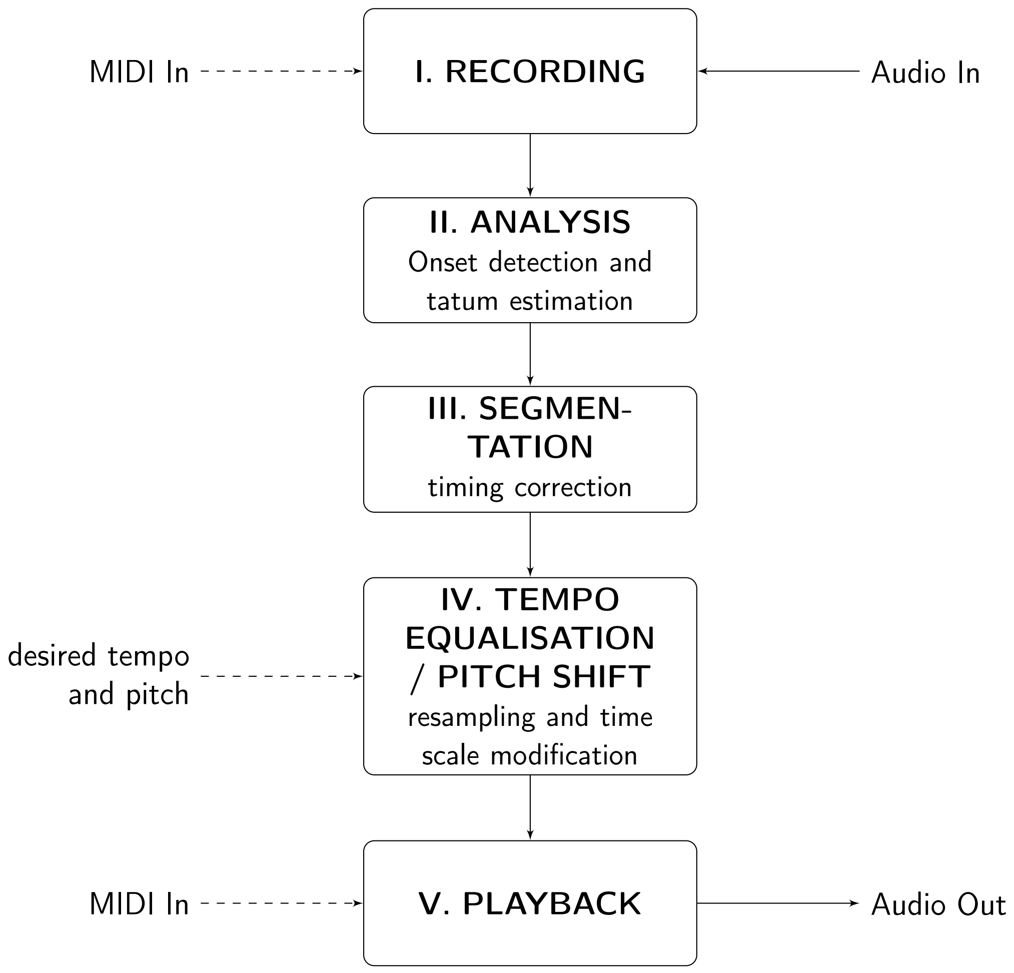

Figure 1 – Block diagram showing the five stages of the proposed prototype system and the basic signal flow. Dashed lines represent user input.

I. RECORDING During the first stage, audio and midi data is recorded. The user selects audio segments by tapping along with the music being recorded. Pressing a pad signifies the start of a desired audio segment, releasing it marks the end of the segment respectively.

II. ANALYSIS In this stage, an onset detection function is calculated for the recorded audio material. Based on the detected onsets, a tempo and tatum estimation is carried out, where 'tatum' describes the lowest metrical level in a musical piece [3].

III. SEGMENTATION In the third stage, the user's timing is corrected by taking the estimated tempo and tatum grid into account. The audio material is then segmented into audio slices, one for each recorded midi note event.

IV. TEMPO EQUALISATION / PITCH SHIFT In the fourth stage, tempo and pitch of the audio slices are manipulated based on desired tempo and pitch values provided by the user.

V. PLAYBACK In this last stage, the user triggers playback of the individual slices by pressing the respective pad on the midi device.

2 Implementation

The prototyping environment, as shown in figure 2, consists of the user input device (MIDI device) and a PC running a *Puredata (PD)* patch for recording and playback, as well as an analysis script in *MATLAB*. The audio interface (*RME HDSP 9632*) runs at 24bit and a sample rate of 44.1kHz, using a buffer size of 64 samples. *PD* runs on the standard MMIO windows drivers with a block size of 64 samples and a delay of 32ms. An analogue mixer is used to route audio to the near-field monitor speakers and the audio interface's inputs.

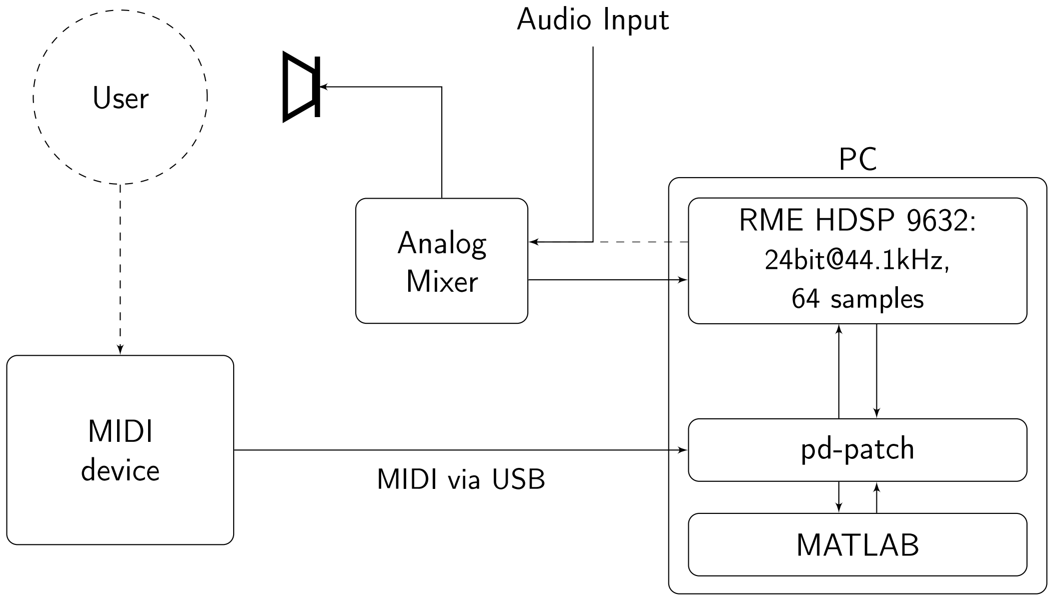

Figure 2 – prototyping environment consisting of a PC running *PD* and *MATLAB*, an analog mixer, near-field monitoring speakers and a MIDI Input device.

2.1 User Input

2.1.1 Hardware - Akai MPD 32

One of the most iconic series of hardware samplers is the *Akai MIDI Production Center (MPC)* series. The *MPC* has been around for decades and is still relevant today. Its pads are regarded as very ergonomic and highly expressive [10], which is why a similar pad-based controller *Akai MPD 32* (see figure 3) was chosen as an input device.

A custom settings preset was created for the device, so that it sends MIDI note values from 60 (C4) to 75 (E5). During recording, pressing and releasing pads marks start and end times of individual audio slices. When in playback mode, pressing pads triggers playback of the respective slices.
Besides its 16 pads, the device's transport buttons are used to start and stop recording, and trigger loading the sliced audio material for playback.

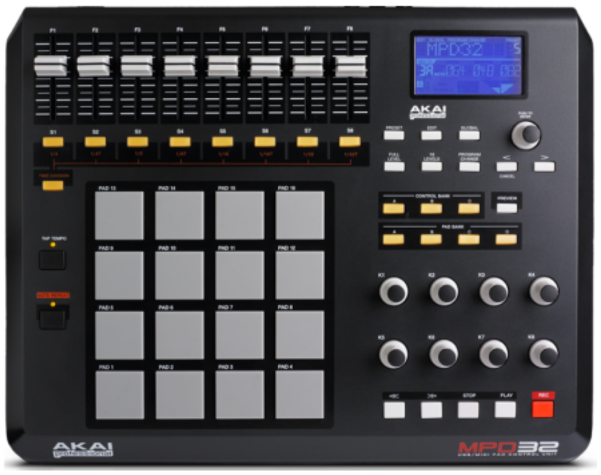

Figure 3 – Akai MPD 32 MIDI Controller

As the device uses standard midi protocol, the software should work with any other midi controller sending the respective values.

2.1.2 Puredata patch

While the use of *PD* has certain drawbacks, such as the added delay (see section 2.1.3), it can be a very flexible prototyping tool.

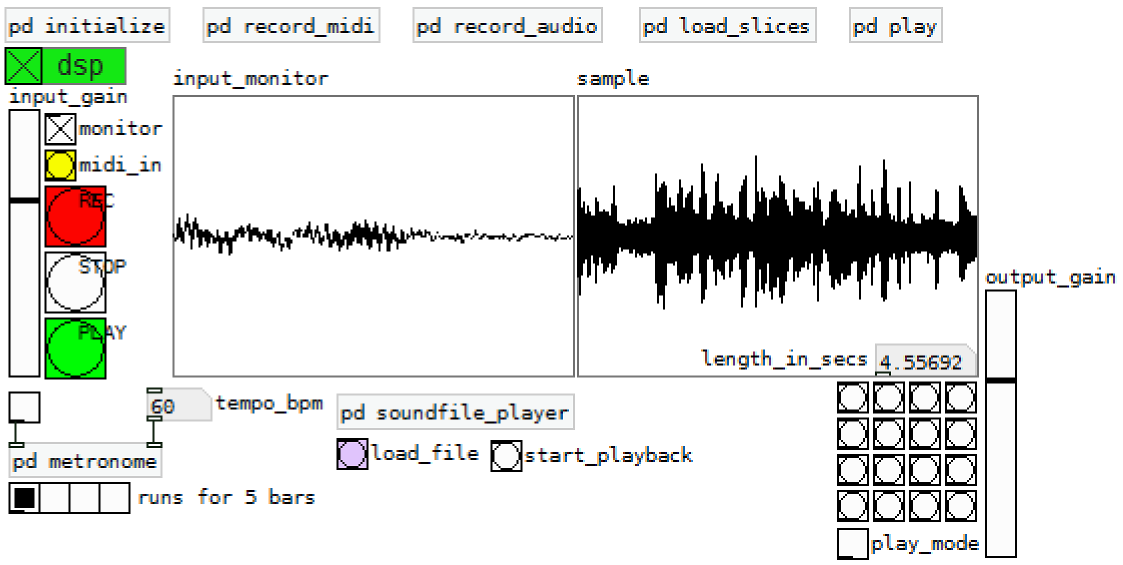

Figure 4 – GUI of the pd-Extended patch

The *PD* patch (shown in figure 4) records audio and midi data, and is also used for audio playback. Recording is triggered by pressing the red 'REC' button on the midi device (see figure 3) or within the *PD* patch. This will start the recording of audio and

midi events inside the sub-patches `[pd record_audio]` and `[pd record_midi]`. For now, the prototype allows 16 discrete midi notes as input, which correspond to 16 audio slices in the end.
The end of the recording is triggered by pressing the 'STOP' button on the device or inside the pd-patch. A short delay is added to ensure that all desired audio material is recorded.
The audio data is stored in .wav-files for analysis in *MATLAB*. Recorded midi events are stored in a text file.
After successful analysis in Matlab, the playback mode is triggered by pressing 'PLAY' on either the device or inside the pd-patch. `[pd load_slices]` will then load the audio slices stored in separate .wav-files. Playback is then triggered when hitting the respective pad. For now, playback is monophonic, meaning that triggering a slice mutes all previously playing slices (`[pd play]`).
The puredata patch also allows to play audio files (`[pd soundfile_player]`) and provides visual feedback for midi input, an audio monitor, as well as controls for input and output gain respectively.

2.1.3 Latency Compensation

The total amount of delay introduced by *PD* (and other parts of the prototyping environment) is not relevant during recording. What is of interest though, is the relative offset between a recorded midi event and the corresponding audio signal.
To measure the relative delay, the pads where tapped while recording midi events and impact sounds simultaneously using a microphone very close ($d < 5cm$) to the input device (see figure 5). Higher precision could have been achieved using an accelerometer, but was not considered necessary here.

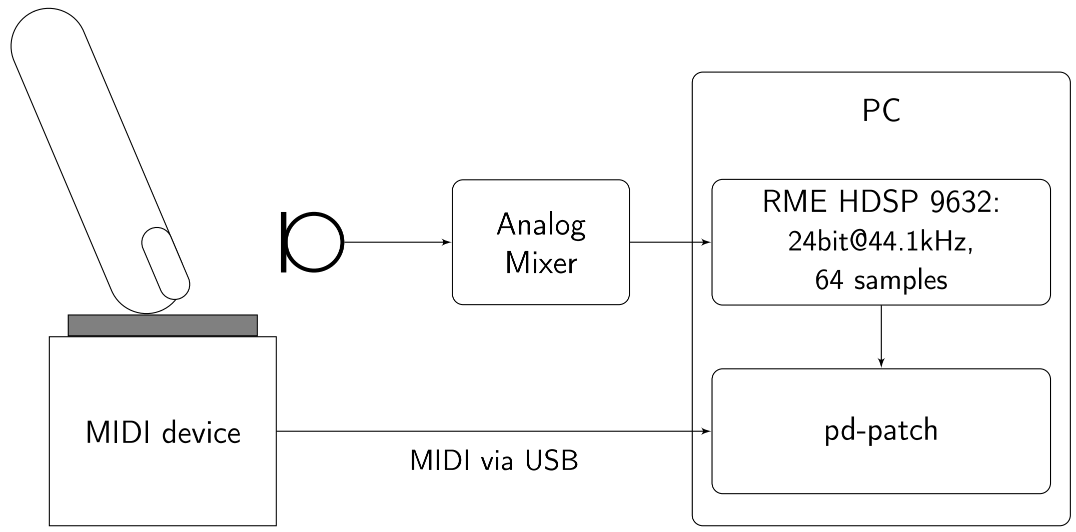

Figure 5 – measuring time offset between finger tap and registered MIDI note onset.

As shown in figures 6 (a) - (c), detected offsets range from around 3 ms to 10 ms with a median of around 7 ms and a variance of around 3 ms. All data for 56 recorded midi events is summarized in a histogram (figure 7). A respective delay compensation of 7 ms was added in the analysis stage.
The data shows that the current setup with *PD* is not ideal for time critical tasks. The offset may be due to the way the operating system (*Windows 10 Pro*) buffers MIDI events or the experimental implementation of MIDI handling for Windows in *PD*.
However, when putting the variance of 3 ms into perspective, it becomes negligible:
3 ms equal 0.6 percent of one beat interval at 120 bpm or 2 percent of one semi-quaver note.

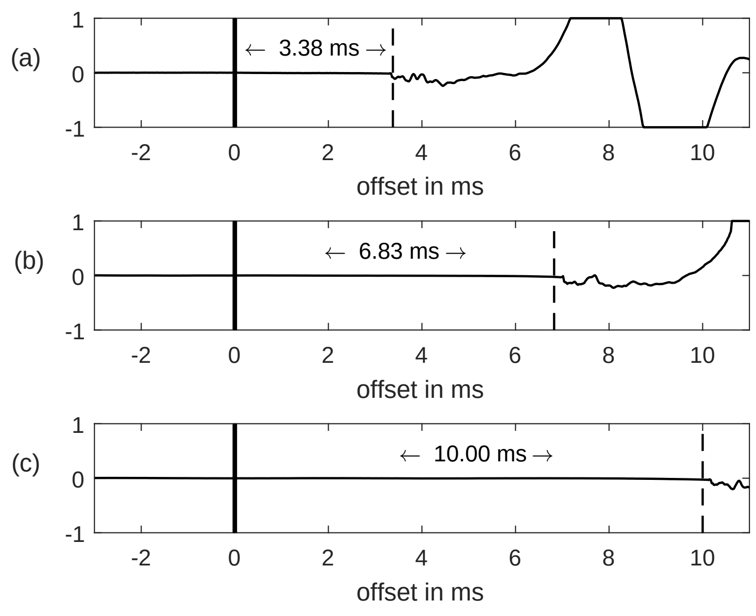

Figure 6 – smallest (a) to largest (c) latency offset - the vertical line at 0ms marks the registered MIDI note event, the dashed vertical line the corresponding audio onset (tapping noise).

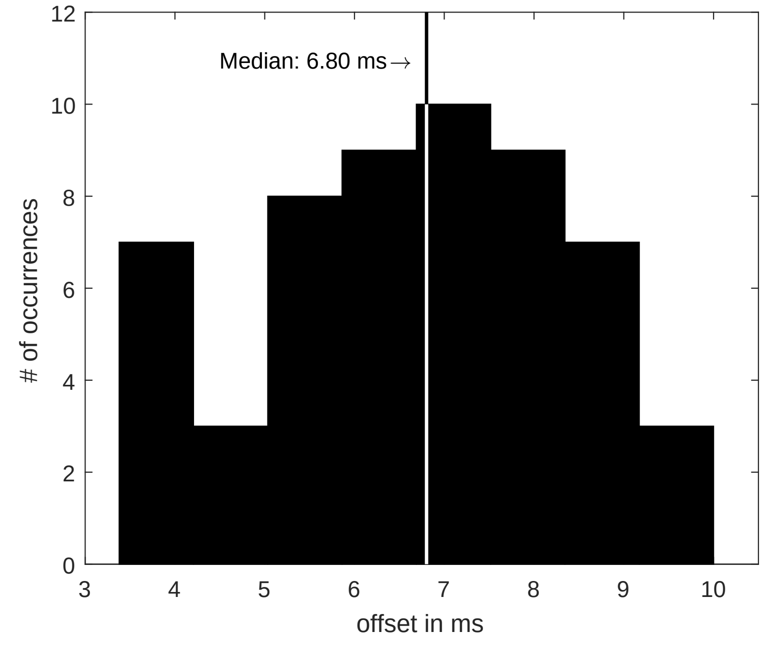

Figure 7 – histogram showing distribution of recorded latency offsets between MIDI note events and corresponding audio. The overall latency offset is around 7ms with a jitter of around 3ms.

2.1.4 Audio and MIDI input data

In figure 8 recorded audio and MIDI input data are shown. The dashed lines represent recorded slice start positions, the dotted lines recorded slice end positions. As can be seen, recorded MIDI note events are slightly off the desired onsets. Especially the first slice end position and the second slice start position are noticeably too early.

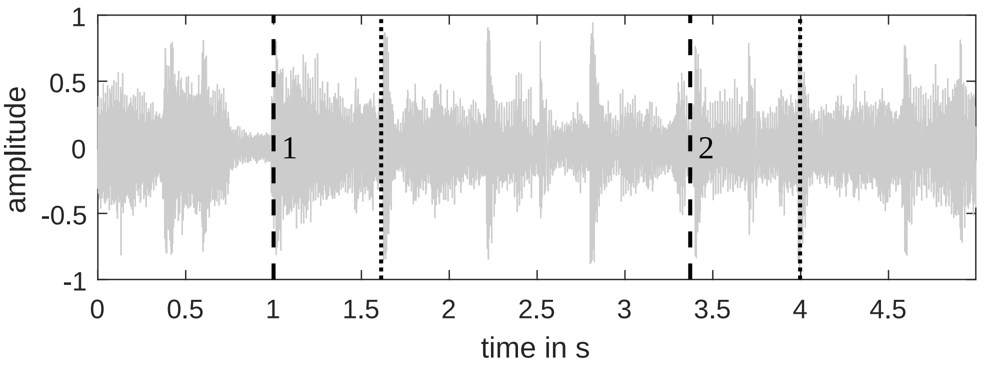

Figure 8 – recorded audio and midi data - dashed vertical lines mark detected MIDI note on events signalling slice start positions, dotted vertical lines mark MIDI note off events signalling slice end positions.

In order to better align the MIDI note events with their corresponding audio events, the MIDI and audio signals are analysed as follows (section 2.2). Timing offsets are then compensated based on prior experiments (see section 2.3.1).

2.2 Analysis

In order to align the recorded MIDI events with their corresponding audio events, an onset detection is carried out (section 2.2.1). Then, a tempo and tatum estimation based on [5] is calculated from the estimated onset detection function (section 2.2.2).

All analysis is done in *Matlab*. A more detailed explanation and a short tutorial are provided in the appendix A on page 26.

2.2.1 Onset Detection

A good onset detection is crucial for various tasks in music information retrieval (MIR), especially for segmentation tasks.
For this project, the onset detection method 'Spectral flux log filtered' (SFLF) proposed by Böck et al. [14] was used with adaptive whitening as proposed by Stowell and Plumbley [4] and implemented by Daniel Rudrich [13].

Spectrogram with Adaptive Whitening
First, the music signal is segmented into N overlapping frames, with $K = 1024$

$$x_n(t) = x(t + nh) \cdot w(t), \qquad n \in [0, N-1], t \in [0, K-1] \tag{1}$$

and transformed using the Short-Term Fourier Transform (STFT)[1]

$$X(n,k) = \sum_{t=0}^{K-1} x_n(t) \cdot e^{\frac{-2\pi jkt}{K}}, \tag{2}$$

yielding the so-called spectrogram $X(n,k)$ with time instances $n = 0, 1, 2, ...$ and frequency bins $k = 0, 1, 2, ..., K-1$.

The resulting magnitude spectrum $|X(n,k)|$ is then *whitened* as follows:

$$\begin{aligned} P(n,k) &= \begin{cases} \max\left(|X(n,k)|,\ r\right) & \text{for } n = 0 \\ \max\left(|X(n,k)|,\ r,\ \mu P(n-1,k)\right) & \text{otherwise} \end{cases} \\ \hat{X}(n,k) &= \frac{X(n,k)}{P(n,k)}, \end{aligned} \tag{3}$$

with $\mu = 0.997$ being the forgetting factor and $r = 0.6$ being the floor parameter.

1. In practice the so-called Fast Fourier Transform (FFT) is used, which is the most efficient implementation of the STFT for power-of-2 analysis window sizes.

Mel-scaling the Spectrogram:

Contrary to the human auditory system, the Fourier Transform's frequency resolution is linear across the frequency range. To match the human auditory system's non-linear frequency resolution (The noticable difference in pitch is smaller, i.e. more precise, in the lower frequency range and increases with rising frequeny.), the magnitude spectrum is combined to $B = 50$ overlapping sub-bands using a triangular filter matrix $M(k, b)$ which corresponds to the Mel-scale[2] with center frequencies between 94 and 15375 Hz. The triangular filters get wider with higher frequencies. No magnitude normalization is applied, so that higher frequencies are emphasized even more.

$$X_{mel}(n, b) = log(\lambda \cdot |\hat{X}(n, k)| \cdot M(k, b) + 1) \tag{4}$$

Finally, the spectral difference, or better known as Spectral Flux (SF) is computed:

$$SFLF(n) = \sum_{b=1}^{B} H\left(X_{mel}(n, b) - X_{mel}(n-1, b)\right) \tag{5}$$

A moving median threshold was used to find onset times, as can be seen in figure 9.

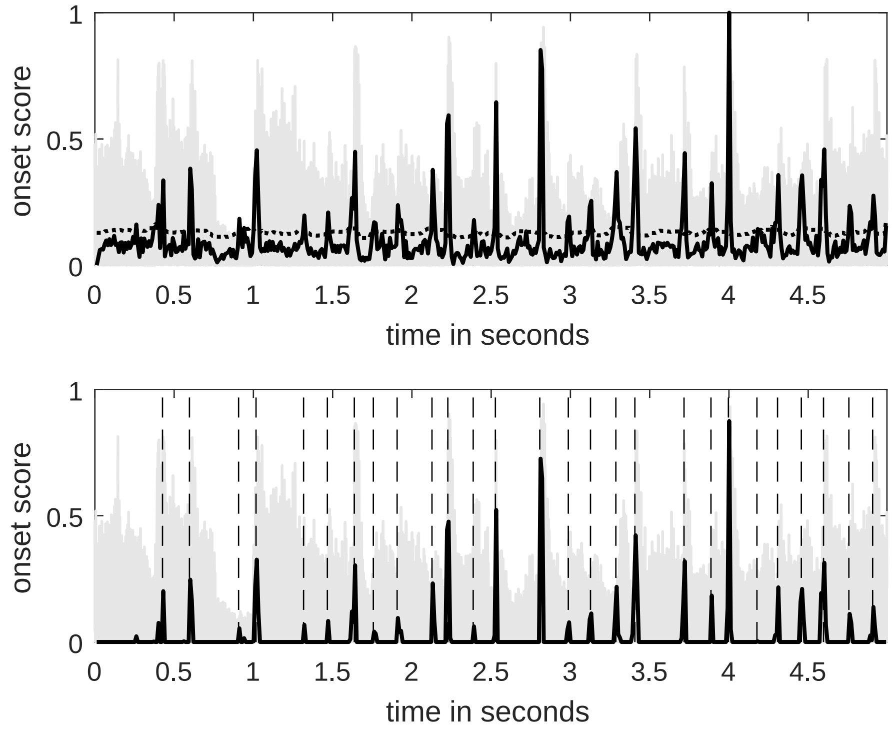

Figure 9 – onset detection and thresholding - **(top)** 'spectral flux log filtered' (SFLF) onset detection function (ODF) (straight line) and moving median threshold (dotted line).
(bottom) deducted threshold from ODF and applied temporal thresholding. dashed vertical lines mark detected onset locations.

2. The Mel-scale is a psychoacoustic scale, modelling the non-linear frequency resolution of the human auditory system and was first introduced by Steven et al. in 1937 [15].

2.2.2 Tempo and tatum estimation

In order to find possible 'sample' start and end points, a so-called tatum grid is estimated, where tatum describes the lowest metrical unit in a musical piece [3]. The tatum estimation is mostly taken from Daniel Rudrich's master thesis [13].

The basic idea is to place different tatum grids over an onset detection function and see which grid fits best (see figure 10).

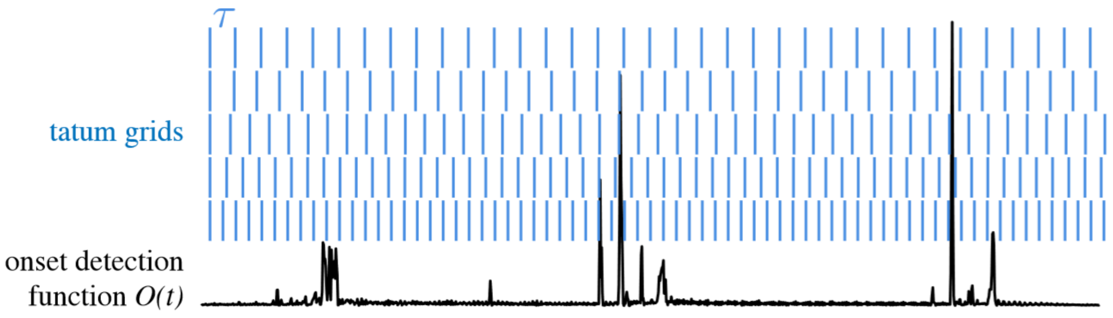

Figure 10 – Basic approach: Find tatum grid that best matches the onset detection function. Source: Daniel Rudrich [5, p.452]

First however, a quick tempo estimation is done based on the general state of the beat estimation proposed by Davies et al. [11]. Basically, the algorithm works by aligning the unbiased auto-correlation function of the onset detection function with comb templates for different tempi. Assuming that most music will be at tempi close to 120bpm, the different templates are weighted with a skewed distribution function, representing the estimated a-priori distribution of tempi. After estimating the beat-period or tempo, the beat locations are estimated by aligning the onset detection function with different beat alignment templates. Instead of the Complex Spectral Difference, the above-described SFLF onset detection function was used.
In essence, the tempo and beat estimation yields a musical piece's estimated tempo and on-beat locations. For a more detailed description please refer to the corresponding paper and comments in the Matlab code.

The estimated tempo $\hat{T}$ is expected to be within 50 to 150 beats per minute and used to define fine tempo tatum grids, as well as on-beat locations.

Tatum grids are defined for tempi 10 bpm slower than the estimated tempo and four times the estimated tempo plus 10 bpm. Tatum grid templates are spaced logarithmically. For example if the estimated tempo is 80 bpm, tatum grids will be defined to tempi ranging from 80 bpm - 10 bpm = 70 bpm to $(80 + 10) \cdot 4 = 360$ bpm.

A tempogram is then computed, describing the tatum presence, i.e. the degree of matching between the onset detection function and the defined tatum grids. The tempogram visualizes the temporal structure of a musical piece as shown in figure 11. To find the lowest underlying metrical level of a musical recording, the optimum tatum path is estimated by minimizing a utility function, which punishes jumps along the path and rewards high tatum presence. For a detailed description of the algorithm, please refer to [13].

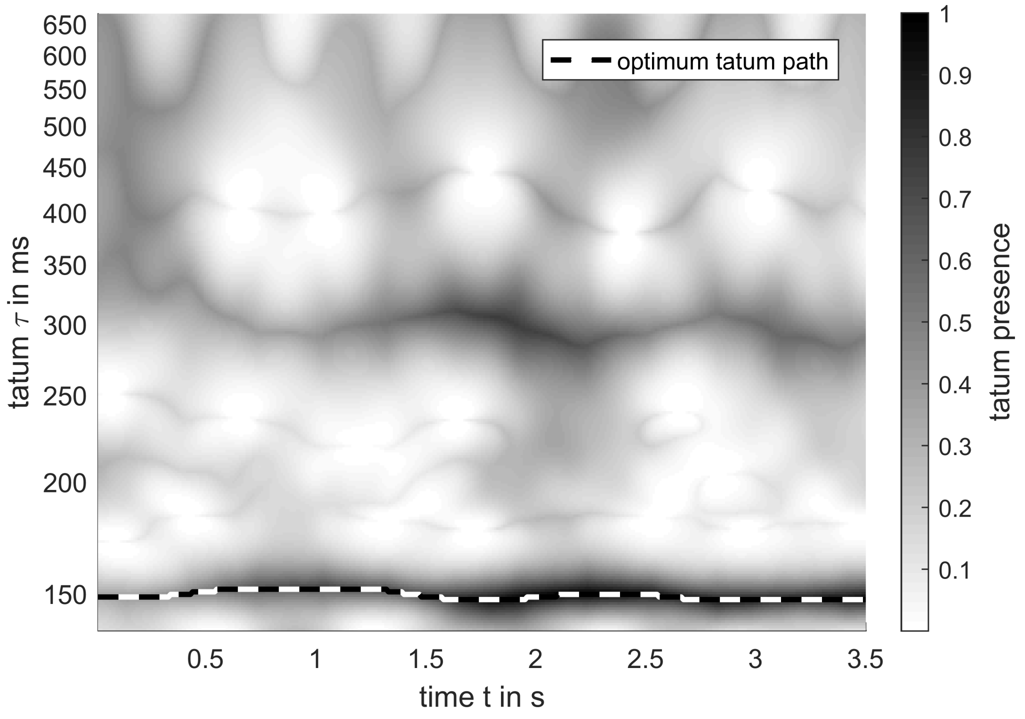

Figure 11 – tempogram and optimum tatum path

The on-beat locations estimated using the general state of the beat estimation by Davies et al. [11] (see above), are moved to the closest estimated tatum, as they are expected to be more precise.

From the tempo and tatum estimation, we now have the audio signal along with estimated onset locations, tatum locations and on-beat locations (as seen in the lower plot of figure 14). These are the candidates, where 'samples' can start or end.

2.3 Segmentation

Audio clips are segmented, considering both the rhythmic structure and the user's timing.

2.3.1 Auditory-motor interaction

According to Zatorre et al. [12], auditory-motor interactions during musical performances can be characterized as feed-forward and feedback interactions. In feed-forward interactions such as tapping along a rhythm, where there is no auditive feedback, timing is predominantly influenced by the auditory system in a predictive manner. As music is generally temporally organized, it allows the listener to make predictions about future events [9].

To understand the anticipative behaviour in auditory-motor interactions, a small informal empirical study was carried out. Two test subjects, both in their mid-twenties and with some musical background, where asked to tap along a metronome for four bars of a 4/4-measure at different tempi. In figure 12 the results of the measurement at 125bpm are shown. In the upper plot, the audio signal (metronome) and the corresponding recorded midi note events are shown (dashed lines). The lower plot shows the distribution of temporal offsets between midi note events and audio cues in milliseconds.

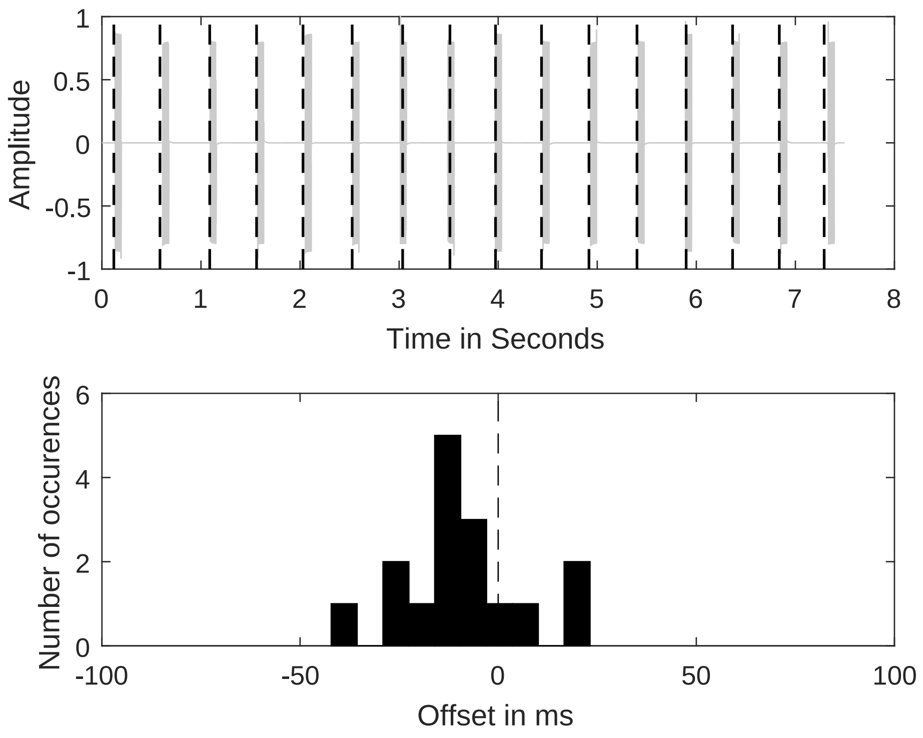

Figure 12 – **(top)** metronome input signal and detected taps, **(bottom)** histogram showing detected taps following a metronome at 125 beats per minute.

The results for all measurements are listed in table 1.

Tempo (bpm)	*60*	*70*	*80*	*90*	*100*	*105*	*110*	*115*	*120*	*125*	*130*	*140*
T$_{ISI}$ (ms)	*1000*	*857*	*750*	*667*	*600*	*571*	*545*	*522*	*500*	*480*	*462*	*429*
Median (ms)	-24	-7	-41	-8	-1	-44	-7	-8	-21	-11	-12	-11
Median (%)	-2.4	-0.8	-5.5	-1.2	-0.1	-7.7	-1.3	-1.5	-4.2	-2.3	-2.6	-2.6

Table 1 – tempi and results for the tapping experiment

In figure 13 the combined distribution in percent of one inter-stimulus onset interval (ISI) is shown for all measurements from 60bpm to 140bpm. As one can clearly see, the test subjects tend to hit the pads early. This corresponds to the findings of Mates et al. [8], who have observed that in tapping experiments, the response onsets precedes the onsets of the stimuli by some tens of milliseconds.

The median relative offset of 3% of one beat interval is to be considered when selecting the timing for 'sample' start and end points.

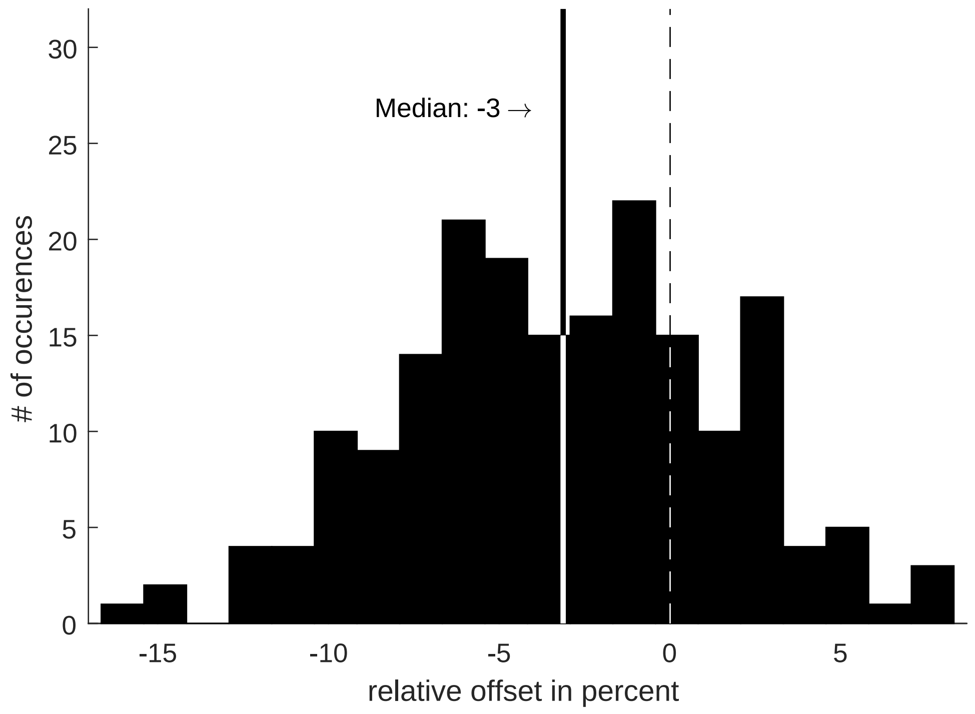

Figure 13 – Histogram showing relative offset times in percent of one beat interval/ISI

2.3.2 Timing Correction

Detected onsets, as shown in figure 9 on page 13, are possible slice start and end positions. In the segmentation step, recorded slice start and end positions are moved to the best possible candidate onset.

To do so, a candidate score is defined, resembling the values of the thresholded ODF at onset times. On-beat onset scores are emphasized by multiplying with scaling factor 2 and clipped to limit 1. An onset is considered on-beat, when it is within the range of one 32th note of the estimated beat location (assuming a 4/4- measure). Figure 14 shows the enhanced ODF and respective onset score at onset times.

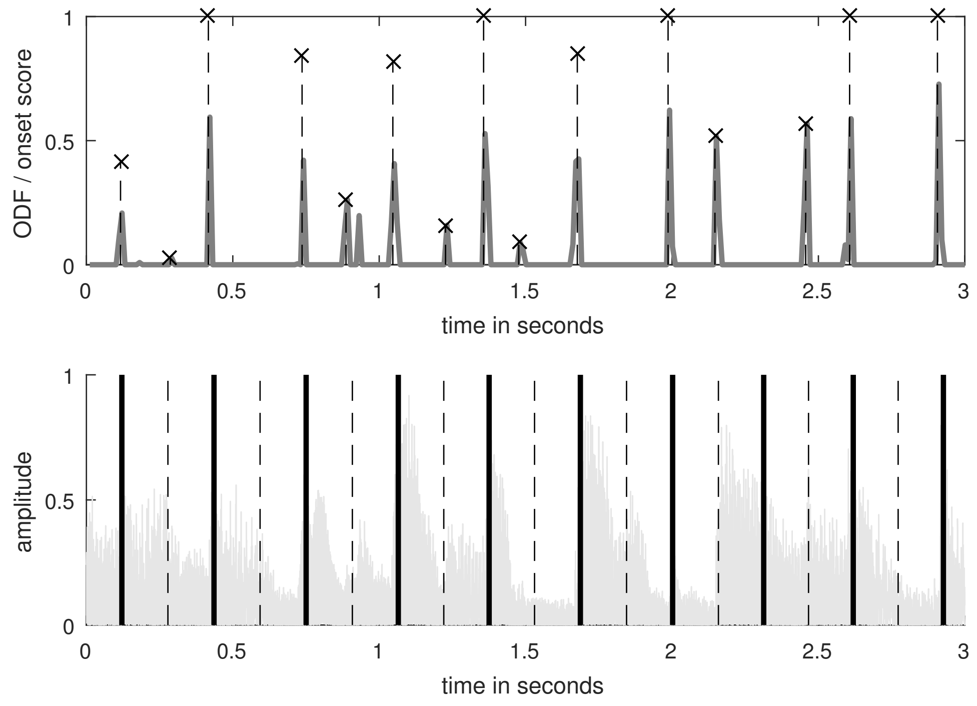

Figure 14 – **(top)** thresholded onset detection function (gray line) and onset scores at detected onsets (marked with x), **(bottom)** estimated tatum grid (dashed lines) and on-beat locations (bold lines)

To take the predictive nature in feed-forward auditory-motor interactions into account (see section 2.3.1) an asymmetrical analysis window and a right-skewed distribution function was used, in order to weight the candidate score to find the most likely onset that the user anticipated to select within an analysis frame.

Several distribution functions were evaluated. The best results were achieved using a beta distribution with $\alpha = 2$ and $\beta = 4$ or a Rayleigh distribution with $\sigma = 0.15$. The beta distribution is defined as

$$f(x, \alpha, \beta) = \frac{1}{B(\alpha, \beta)} \cdot x^{\alpha-1} \cdot (1-x)^{\beta-1}, \tag{6}$$

with $B(\alpha, \beta)$ being the beta function, which normalizes the total integral to 1.
The Rayleigh distribution is given as

$$f(x, \sigma) = \frac{x}{\sigma^2}\, e^{-\frac{x^2}{2\sigma^2}}\ , \qquad x \geq 0. \tag{7}$$

Both distribution functions are shown in figure 15 with different values for α, β and σ:

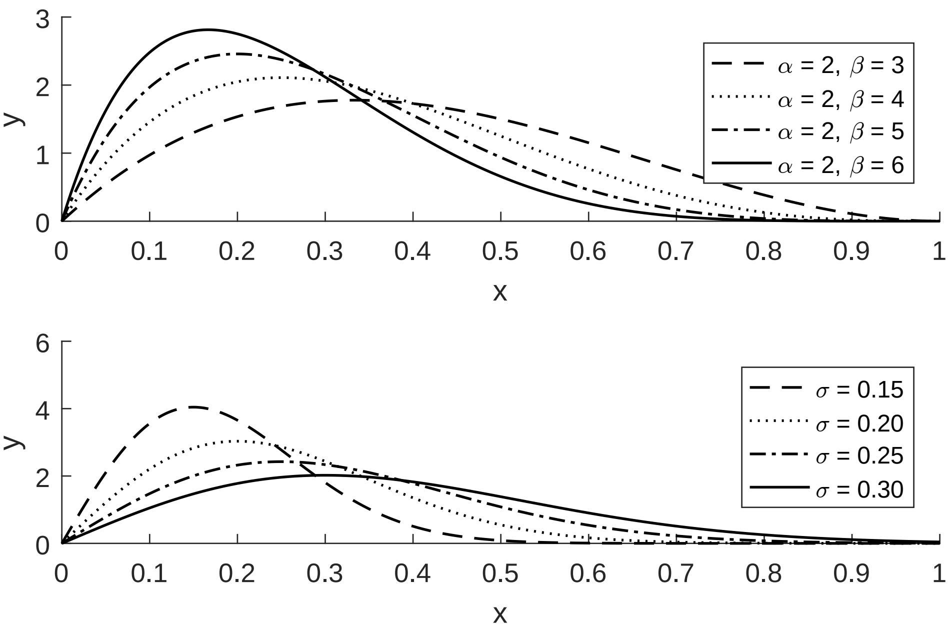

Figure 15 – right-skewed distribution functions - **(top)** Beta distribution $f(\alpha, \beta, x)$, **(bottom)** Rayleigh distribution $f(\sigma, x)$

An analysis frame length of one beat interval (a quarter note assuming a 4/4-measure) was chosen. A weighted analysis frame is placed around each recorded midi event, so that the distribution function reaches its maximum 3% of one beat interval after the recorded MIDI event.

The candidate within the frame that has the highest weighted onset/candidate score is then taken as the new slice start or end position.

In figure 16, an onset analysis frame is shown. The midi note event is shown as a thick vertical line. The weighted onset candidate with the highest score is chosen as the new slice start position (dotted line). This applies to both 'sample' start and end points. As intended, the onset shortly after the registered MIDI event is considered the most likely candidate.

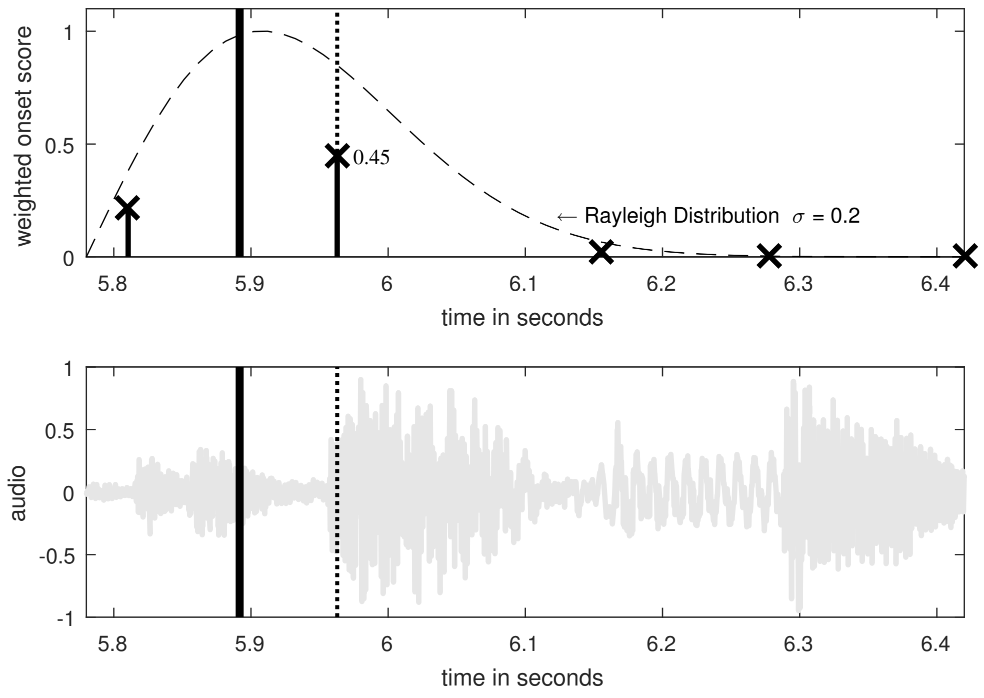

Figure 16 – timing correction - **(top)** recorded MIDI note event (thick vertical line) and onset candidates (marked as x) after applying weighting with Rayleigh distribution. **(bottom)** audio signal (gray) with original timing (thick line) and corrected timing (dotted line)

If no onset candidates can be found within an analysis window, the closest tatum will be chosen.

To reduce cracking noise at sample start positions, the exact position is moved to the next zero-crossing.

2.4 Pitch Shift and Tempo Normalization

In order to use the audio segments in a loop-based music production environment, tempo normalization is applied to compensate variances in tempo in the original recording. This not only allows the tempo-synchronous playback of 'samples' and tempo changes to any desired tempo but also allows the change of pitch. This is done by resampling the audio prior to time-stretching.

All time-scale modification was done using the *Time-Scale Modification* (TSM) toolbox for *Matlab*, developed by J. Drieger and M. Müller at Audio Labs Erlangen [6]. The TSM toolbox implements several time-stretching algorithms, which can be selected in the main analysis Matlab script (see also appendix A). Figure 17 shows the original and modified signal are shown for one audio segment.

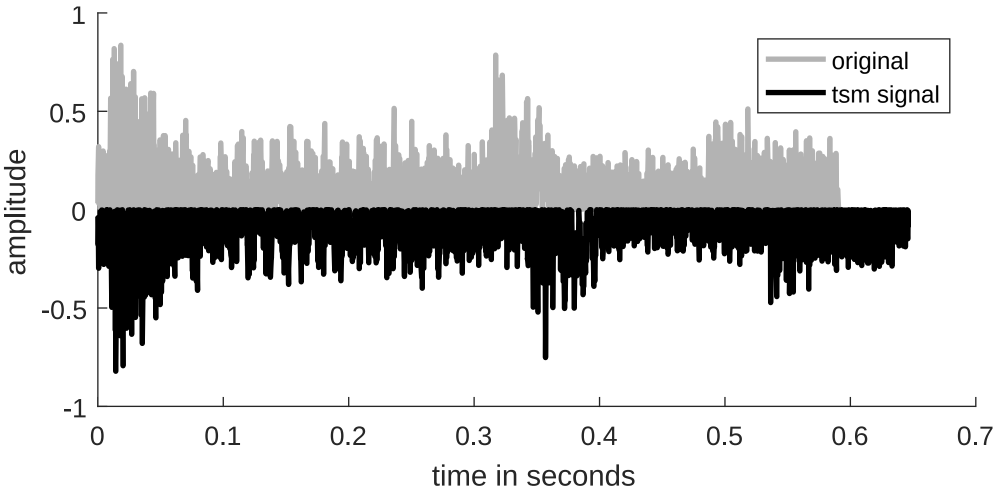

Figure 17 – time-scale modification of an audio segment using the TSM toolbox [7].

TSM was added, to provide a proof-of-concept demonstration of the segmentation algorithm. The quality of the different TSM methods was not evaluated. Different algorithms work better for different kinds of audio signals. For a detailed description and advantages/disadvantages for each algorithm, please refer to [7].

3 Evaluation

As the audio material is subject to the user's musical taste, it is difficult to find an objective measure of how well the algorithm works. The evaluation is thus rather subjective.

Another problem is that it is highly dependant on how well the user knows the musical pieces. If a user listens to a song for the first time, there is no chance of him or her being able to hit the pads on time. On the other hand, if the user knows the songs well, even short sounds like isolated drums can be easily sliced using the prototype system.

There is a also a strong learning effect. The authorgot more and more confident when slicing 'samples' on the fly, after playing with the prototype for while.

To evaluate the prototype environment samples from 13 songs were recorded with a total of 191 recorded slices (see table 2). All songs are among the most sampled songs according to whosampled.com [1] and well-known to the author. Most selected samples started and ended at on-beat locations (no syncopation). All are either 4/4- or 6/8-measure. Most feature drums, which leads to a strong onset detection.

Artist	Title	# of slices	correct
Bobb Byrd	Hot Pants	8	8
Bob James	Nautilus	16	15
Bob James	Take Me to the Mardi Grass	13	13
Commodores	Assembly Line	11	11
Curtis Mayfield	Move On Up	16	8
Dexter Wansel	Theme For the Planets	16	16
Isaac Hayes	Ike's Rap II	16	16
James Brown	Funky President	16	16
James Brown	The Playback	16	15
Kool & the Gang	Summer Madness	15	15
Lyn Collins	You can't love me if You Don't Respect Me	16	16
Marvin Gaye	Got to Give it Up	16	16
Parliament	Mothership	16	16
Total		191	181

Table 2 – evaluation set

181 slices or around 95% were segmented correctly. A 'sample' was considered correct when both start and end time of the slice were as intended by the author, when listening to them. Of the 10 slices that were sliced incorrectly, 6 started at the right onset but ended too early. This indicates that releasing a key or pad in a musical context might follow different principles than tapping-experiments suggest and should be investigated more thoroughly.

8 out 10 incorrectly sliced segments are from the song 'Move On Up' by Curtis Mayfield. This song features a complex rhythm with high-tempo percussion patterns, leading to a complex onset detection function.

Note that the examples chosen for this evaluation only cover a limited range of musical genres and all are rhythmically expressive, i.e. with prominent drums or rhythm sections. The 95% thus only provide an indication for best-case performance.

Other test with songs featuring less percussive instruments, or songs not as well known to the author have not been as successful. However, interesting *happy accidents* occurred quite often especially when experimenting with extreme TSM settings.

In summary, the developed prototype works as intended at reasonable tempi, especially when there are drums present and the rhythmic structure is not overly complex - given the user knows the musical pieces well. For less percussive sounds, when the onset detection function is not reliable, the algorithm fails to find the right timing.

Outlook:
While the current prototyping setup is limited in performance (see section 2.1.3), maximum slice count, ..., it did allow the author to easily select different song segments of various kinds from a wide range of input audio material.

For future projects, i.e. when integrating an algorithm like this into hardware and software samplers, recording, analysis and playback should be done in a more robust and efficient environment (i.e. with native code, such as C/C++).

References

[1] WhoSampled.com - Most Sampled Tracks. `https://www.whosampled.com/most-sampled-tracks/1/`. Accessed: March 2017.

[2] Wikipedia - Sampling (music). `https://en.wikipedia.org/wiki/Sampling_(music)`. Accessed: March 2017.

[3] J. Bilmes. Timing is of the essence: Perceptual and computational techniques for representing, learning, and reproducing expressive timing in percussive rhythm. PhD thesis. Massachusetts Institute of Technology, Program in Media Arts and Sciences, 1993.

[4] M. Plumbley D. Stowell. Adaptive whitening for improved realtime audio onset detection. In *Proceedings of the International Computer Music Conference (ICMC)*, 2007.

[5] Alois Sontacchi Daniel Rudrich. Beat-aligning guitar looper. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 451–458, Edinburgh, UK, 2017.

[6] Jonathan Driedger and Meinard Müller. Time Scale Modification (TSM) Matlab Toolbox. `https://www.audiolabs-erlangen.de/resources/MIR/TSMtoolbox/`. Accessed: April 2017.

[7] Jonathan Driedger and Meinard Müller. Tsm toolbox: Matlab implementations of time-scale modification algorithms. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, pages 249–256, Erlangen, Germany, 2014.

[8] T. Radil Ernst Pöppel J. Mates, U. Müller. Temporal integration in sensimotor synchronization. *Journal of Cognitive Neuroscience*, 6:4:pp.332 – 340, 1994.

[9] E. W. Large and C. Palmer. Perceiving temporal regularity in music. *Cognitive Science*, 2002.

[10] Patrick McGlynn. Interaction design for digital musical instruments. PhD thesis. National University of Ireland Maynooth, 2014.

[11] M.D. Plumbley M.E.P. Davies. Context-dependent beat tracking of musical audio. In *IEEE Transactions on Audio, Speech and Language Processing*, volume vol. 15, no. 3, pages pp. 1009–1020.

[12] Virginia B. Penhune Robert J. Zatorre, Joyce L. Chen. When the brain plays music: auditory-motor interactions in music perception and production. *Nature Reviews Neuroscience*, 8:547 – 558, July 2007.

[13] Daniel Rudrich. Timing-Improved Guitar Loop Pedal based on Beat Tracking. Master Thesis. Institute of Electronic Music, Univiersity of Music and Performing Arts Graz, 2017.

[14] M. Schedl S. Böck, F. Krebs. Evaluating the onset capabilities of onset detection methods. In *International Society for Music Information Retrieval Conference - ISMIR*, 2012.

[15] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

A Code Handbook / Tutorial

The code is structured as follows:

Puradata:
The folder ./puredata/ contains the puredata patch chopper.pd which is used for recording audio and midi data, as well as playback of the sliced audio 'samples'. It contains several subscripts [pd record_audio] and [pd record_midi], [pd load_slices] and [pd play] as already described in section 2.1.2. The folder is also where the recorded samples will be stored as .wav-files. All audio is summed to mono and stored with 24 bit and a sample rate of 44100Hz. Registered midi events are written to text files, which will be analysed by the *Matlab* scripts along with the recorded audio.

Note that the script requires pd version 0.43.4-extended.

Matlab:
The folder ./matlab/ contains the main *Matlab* script for analysis, segmentation and time-scale modification.

At the top of the main script ./matlab/main.m, some global preferences can be set, such as desired output tempo, pitch shift, minimum slice length and several logging options:

```
...
min_slice_length = 1; % minimum slice length in tatums

% parameters for time scale modification and pitch shift
TS_type = 'WSOLA'; % 'OLA' 'WSOLA' 'PV' 'PVPL' 'HPSS' 'none'

desired_tempo = 0; % 0 for average detected tempo
transpose = -3; % in semitones -12 to 12, 0 for no change
...
```

./matlab/functions/ contains required functions called by the main analysis script.

The code requires the TSM toolbox [6] to be installed and was designed for Matlab Version R2016a.

Running the Code:
To use the prototype make sure *Puredata* is configured to receive audio and MIDI data. Start the the recording as described in section 2.1.2.

After recording, simply run the main matlab script (press 'Run' or hit F5). The audio slices are saved to ./puredata/.

To activate playback of the slices using the respective pads on the MIDI device, hit 'PLAY' on the device or in the Puredata patch chopper.pd.