OBTAIN: Real-Time Beat Tracking in Audio Signals

Ali Mottaghi, Kayhan Behdin, Ashkan Esmaeili, Mohammadreza Heydari, and Farokh Marvasti
Sharif University of Technology, Electrical Engineering Department, and
Advanced Communications Research Institute (ACRI), Tehran, Iran
aesmaili@stanford.edu

***Abstract*—In this paper, we design a system in order to perform the real-time beat tracking for an audio signal. We use Onset Strength Signal (OSS) to detect the onsets and estimate the tempos. Then, we form Cumulative Beat Strength Signal (CBSS) by taking advantage of OSS and estimated tempos. Next, we perform peak detection by extracting the periodic sequence of beats among all CBSS peaks. In simulations, we can see that our proposed algorithm, Online Beat TrAckINg (OBTAIN), outperforms state-of-art results in terms of prediction accuracy while maintaining comparable and practical computational complexity. The real-time performance is tractable visually as illustrated in the simulations.**

***Index Terms*—Onset Strength Signal; Tempo estimation; Beat onset; Cumulative Beat Strength Signal; Peak detection**

I. Introduction

Many works have been carried out in offline beat tracking. One can find effective algorithms in the literature which perform beat tracking in an offline fashion [1]. These methods have access to the entire signal data. However, in this paper, we are proposing a new algorithm towards real-time beat tracking. Beat tracking is an audio signal processing tool which is based on onset detection. Onset detection is an important issue in signal processing. It can be widely seen in different pieces of research that onset detection is used such as music signal processing [2], neural signal processing (EEG, ECoG, and FMRI), and other biomedical signal processing areas such as electrocardiac signals to name but a few [3], [4]. Figure 1 shows a typical onset response appearing in musical signal processing. Therefore, it is important to detect real onset locations in these signal. The practical importance in the biomedical region falls in approximating the precise onset in noisily recorded data. In musical signal processing, there would be many practical cases where this onset detection would prove to be important. Visual effects in musical applications may work based on real-time onset detection as in music player applications. The purpose is to capture abrupt changes in the signal at the beginning of the transient region notes [2].

In this work, we propose a modular method containing numerous blocks. We call our algorithm OBTAIN (a pseudo-abbreviation of Online Beat TrAckINg). We will elaborate upon the blocks of this system throughout the paper and compare it to a state-of-art method. The rest of this paper is organized as follows:

Section II elaborates the system module. Section III illustrates the simulation results. Section IV shows how we have implemented our algorithm And finally, we conclude the paper in section V.

II. OBTAIN Algorithm

A. Generating Onset Strength Signal (OSS)

We split the subject audio file into overlapping windows. The sample rate of the audio signal is assumed to be $F_{S0} = 44100Hz$. In order to detect beats, we require to perform our algorithm on a sequence of samples since working with one sample at a time we cannot derive any beat. We require processing a frame of samples to implement Fast Fourier Transform (FFT) in order to have access to an array of samples so that the pattern of beats could be learned. Therefore, we consider windows of samples where each window is of size 1024 samples, i.e. we suppose each window contains 1024 samples as explained in [2]. Thus, the sampling rate is $\frac{44100}{1024} = 43.06Hz$ We also consider the overlapping ratio equal to 87.5%. In other words, we choose the Hop size (H) parameter equal to 128 samples and consider the overlapping ratio of the new input series of samples with the stacked frame equal to 87.5%. The reason behind choosing large overlap is to enhance accuracy. If we try to maintain the structure of a specific frame for several stages, the efficiency of the algorithm performance increases since the desired frame stays somehow in the memory for a while. This is, in fact, equivalent to 87.5% decrease in sampling rate. Then, we compute FFT of each window. We normalize the data by dividing the components to a normalizing value. This normalizing value is chosen as follows: We consider a fixed span of the frequency band at the beginning of FFT of the audio signal and find the maximum absolute value in this span of time. We suppose this is a good approximate of the maximum component for the entire frequency range. Afterward, threshold the components below an empirical level $74dB$ to zero (An empirical noise level cancellation) [5]. Next, we apply log-compression on the resulted window [2]. The log compression is carried out as follows: Let X denote the resulted window, then the log-compressed signal is:

$$\Gamma_\lambda(X) = \frac{log(1+\gamma|X|)}{log(1+\gamma)} \tag{1}$$

It is worth noting that after log-compressing we perform a further normalization step in order to be assured the maximum

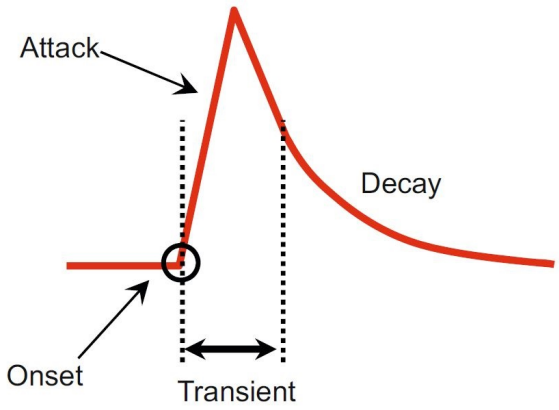

Fig. 1: Onset of the envelope of the audio signal. [2]

of the signal is set to 1. Log compression is carried out in order to reduce the dynamic range of the signal and adapt the resulted signal to the human hearing mechanism which is logarithmically sensitive to the voice amplitude [2].

We define $Flux$ function based on the log-compressed signal spectrum Γ_λ as follows:

$$Flux\ [n] = \sum_{k=0}^{K} |\Gamma_\lambda[n+1,k] - \Gamma_\lambda[n,k]|_+, \tag{2}$$

where $|x|_+$ is $max\{x, 0\}$.

This function is, in fact, discrete temporal derivative of the compressed spectrum. [2]. Now we apply a Hamming window ($h[n]$) of length $L = 15$ with the cutoff frequency equal to $7Hz$ in order to remove noise components from the OSS. OSS can be derived by applying the Hamming filter on the $Flux$ as follows:

$$OSS[n] = \sum_{k=n-[\frac{L}{2}]}^{n+[\frac{L}{2}]} Flux[k] \times h[k] \tag{3}$$

After denoising the OSS, the detected peaks of the OSS represent the onset times.

B. Tempo Estimation

We store the OSS we obtained in the previous phase into a buffer of length 256. Each buffer contains 256 OSS signals. Two intuitive reasons behind this choice of length for the buffer could be first, the robustness of real-time process, and second, the time required for the buffer to load enough samples for detection would be approximate 3 secs, which is approximately compatible to human hearing capability in beat detection. Before proceeding to the next step in tempo estimation, it is worth introducing the concept of generalized auto-correlation. We define the generalized auto-correlation of a signal with itself as follows:

$$A_m[n] = DFT^{-1}\left(|DFT(OSS[n])|^c\right) \tag{4}$$

Letting $c = 2$ leads to the well-known auto-correlation formula. Choosing smaller values for c yields sharper peaks

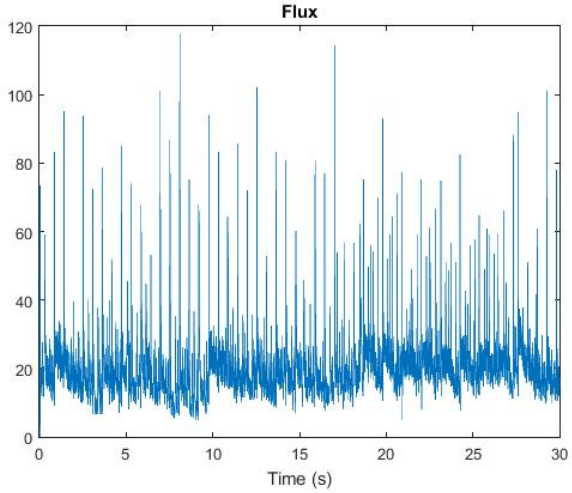

Fig. 2: The $Flux$ signal for audio signal no. 10 in dataset Open in [6].

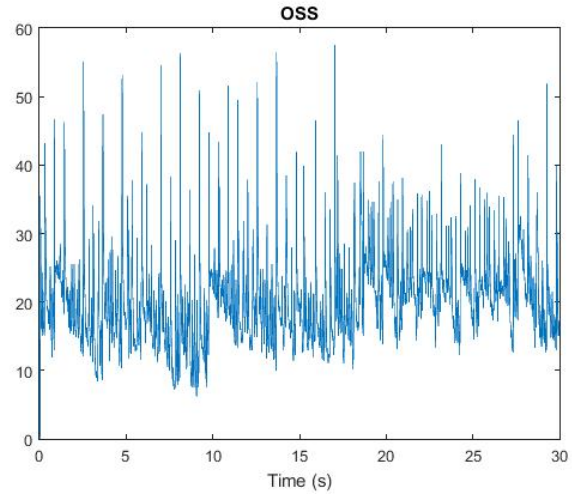

Fig. 3: The OSS signal for audio signal no. 10 in dataset Open in [6].

in the result which will be helpful in tempo estimation. We have chosen $c = 0.5$ as simulations have shown this value leads to better estimations. [7] Next, we enhance harmonics by defining the following function:

$$EA_m[n] = A_m[n] + A_m[2n] + A_m[4n] \tag{5}$$

The reason for including the downsampled terms $A_m[2n]$, and $A_m[4n]$ is that auto-correlation function is of a harmonic nature. Therefore, we expect to observe enhanced peaks in $EA_m[n]$. Since the main beat period (BP) we are looking for typically falls in the range $[0, 127]$ samples per BP, we only evaluate this function on this interval. In equation 5, we suffice to include only the terms $A_m[2n]$, and $A_m[4n]$. It is worth noting that these functions are set equal to 0 outside the interval $[0, 127]$. We experimentally can see that the interval $[50, 210]$ (BPM) is an exhaustive range which includes the tempo for most of the audio signals. If we let τ_b denote the

BP, it is trivial that:

$$BPM = \frac{60 \times F_S OSS}{\tau_b} \tag{6}$$

As a result, we achieve a limited range for the BPM, and τ_b consequently.After that, as discussed in [7] We observe the pattern of the $EA_m[n]$ and select the maximum 10 peaks in this interval whose locations are period nominees for BP and then by using pulse trains we score each BP candidate. Pulse Trains are constructed based on each BP candidate and correlated with buffered OSS Signal. Finlay, we choose the highest score as our instant BP. We also used a feedback system that uses overall estimation in [7] to prevent some abrupt changes in BP, because in most songs BP changes smoothly and abrupt changes rarely happen.

C. Cumulative Beat Strength Signal (CBSS)

We use a method similar to the method introduced in [8]. To generate CBSS for each frame, we initially look for the previous beat which is considered to be observed as a peak in this signal. CBSS for a frame is equal to the weighted sum of OSS in the corresponding frame and the value of CBSS of the last beat using different weights. Now, we explain how to find the last beat. To this end, we employ a log-Gaussian window. In order to specify the location of the beats, we use a recursive method to assign a score to each time slot which determines the beat power in the working frame. The maximum value among these scores specify the beat locations. CBSS is obtained via calculating the summation of two terms: one from the previous frames and the other is related to the current frame. Let τ_b be the estimated beat period from the tempo estimation. We consider a scoring window on the span $[t - \tau_{\frac{b}{2}}, t + \tau_{\frac{b}{2}}]$. Then, we form the log-Gaussian window as follows:

$$W[v] = e^{[-(\eta log(-\frac{v}{\tau_b})^2)/2]}, \tag{7}$$

where $v \in [-\tau_b/2, -2\tau_b]$, and η determines the log-Gaussian width. $Cu[n]$ denotes the CBSS at each time slot. $\Phi[n]$ is defined as follows:

$$\Phi[n] = \max_v W[v] Cu[t + v] \tag{8}$$

This value approximately determines what the score of the previous beat was. Implementations agree with this assumption in assigning scores to the beats [1]. Finally, the score for each frame is calculated as follows:

$$Cu[n] = (1 - \alpha) OSS[n] + \alpha \Phi[n] \tag{9}$$

This structure results in quasi-periodic CBSS. Therefore, even when the signal is idle, previous scores could be used in order to obtain the next beat. The periodic structure is improved throughout the learning process of the algorithm, and the estimation accuracy increases. The choice of α is carried out using cross-validation in several implementations on the training data set selected randomly from the main dataset (80% of the dataset).

D. Beat Detection

In this Section, periodic peaks of CBSS are detected in a real-time fashion, and the output signal "beatDetected" is a flag which is set to 1 if a peak is detected in that frame. This block takes advantage of two separate parallel systems to enhance reliability of the system performance. The initial system simply tracks periodic beats without taking into account the beat period, while the second system is totally dependent on the beat period.The main assumption is that if the beat period is not detected correctly, the cumulative signal still maintains its periodic pattern. Therefore, the second system is a correction system. The final system decides based on the comparison of the CBSS values in the peak locations detected by these two systems. The system yielding higher average is chosen. Each frame consisting of 512 samples of the CBSS is fed into this block in a buffer. Two consecutive buffers are overlapped with 511 common samples (similar to FIFO concept). In order to reduce the complexity of computations, both systems do not function for all frames. The first system only functions when the distance between the current sample and the time the previous beat is detected falls in the span $(BP-10, BP+7)$. This span is chosen since the beats must be detected within at most $0.1s$, and further delay is not practical for a real-time system. If no beat is detected in a frame, the system finally turns the flag to 1, which shows a peak is inside that frame. The second system works exactly in the middle of the two beats, i.e. when the half of the BP is passed since the last detected beat and stops detecting till the next beat is detected. Thus, it must be stored in a buffer until the next beat is detected for comparison. The correction made by the second system is through this buffer. When the second system achieves a higher average in comparison to the first system, changing this buffer for the location of the final peak detected by the second system, the first system is corrected. Now, we specify the mechanism of each system as follows:

A. The main (initial system):

Here, we take advantage of the method introduced in [9]. The main concept is that the periodic beats have the largest value in comparison to the rest of samples within windows of length τ_b; therefore, we initially look for the main BP and afterward look for the maximum values in windows of length BP. A brief summary the method provided in [9] is summarized as follows: We assume that the input series to this system is the CBSS (whose peaks should be detected). We initially subtract the linear predictor of the data from the samples and denote the resulted signal with x. Afterwards, the $LMS \in R^{\lceil \frac{N}{2} - 1 \rceil \times N}$ is defined as follows:

$$m_{k,i} = \begin{cases} 0, & x_{i-1} > x_{i-k-1} \wedge x_{i-1} > x_{i+k-1} \\ 1 + r, & \text{otherwise} \end{cases} \tag{10}$$

where r is a uniform random number in $[0, 1]$.

Then, the rows of the matrix LMS are added together. Let γ_k denote the result of summation for each column. Now, let $\lambda = argmin(\gamma_k)$. Clipping the matrix from the λ-th row, we will have the resulted $ScaledLMS$, finally the columns which

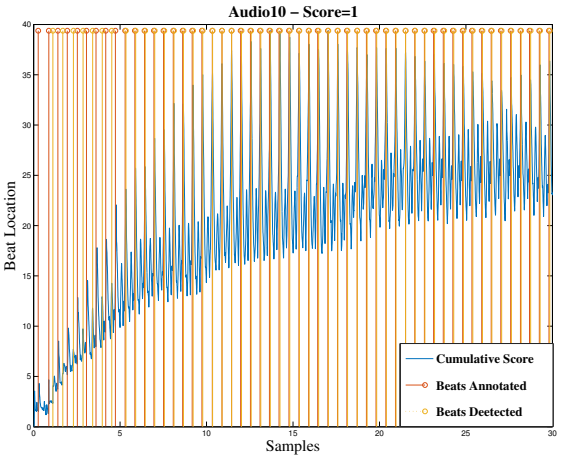

Fig. 4: Real-time beat detection.

have zero variance in the $ScaledLMS$ matrix determine the peak locations.

These evaluations are carried out for each frame. If the last sample of the frame is a peak, the flag "beatDetected" turns to 1.

B. The second system:

In this system, a pulse train with the same period as detected by the peak detection is generated, and afterward, is cross-correlated with the CBSS.

$$CCor[n] = CBSS[n] * PulseTrain[-n] \tag{11}$$

The peak location of the resulted signal determines the displacement required for the pulse train to be matched with the CBSS. Therefore, if the pulse train is shifted for this displacement, we are assured that the peaks of CBSS are detected. In order to prevent errors in this section, the correction system is proposed.

III. Simulation Results

The dataset we used in the simulations and measuring our method's accuracy and performance is derived from the [10]. Figure 4 shows our algorithm's performance on beat detection for Audio number 10 in the [10]. We also use ICASSP SP cup training dataset provided in [6] consisting of 50 musical excerpts. We also compare our method to IBT method in [11]. We evaluate the performance of methods based on the four metrics defined in the Section "Evaluation" as in [12]. The four metrics are CML_c, CML_t, AML_c,AML_t respectively. These four metrics are based on the continuity of a sequence of detected beats. CML_c is the ratio of the longest continuously correctly tracked section to the length of the file, with beats at the correct metrical level. CML_t the total number of correct beats at the correct metrical level. AML_t is the ratio of the longest continuously correctly tracked section to the length of the file, with beats at allowed metrical levels. AML_c the total number of correct beats at allowed metrical levels.

We also compare the two algorithms based on the P-score metric introduced in [13] and f-Measure as introduced in [14]. Let b denote number of correctly detected beats and p denote number of false detected beats and n denote number of undetected beats. P-score and f-Measure are defined in 12 and 13.

$$P = \frac{b}{b + \max(p, n)} \tag{12}$$

$$F = \frac{b}{b + \frac{p+n}{2}} \tag{13}$$

We have fixed the Tempo tolerance to 17.5% and Phase tolerance to 25% for the four metrics introduced in [12]. The tolerance window is set to 17.5% for f-Measure. The results of the comparison of the performance of the two algorithms on the mentioned two datasets are provided in Table I.

The four metrics AML_t, AML_c, CML_t, CML_c evaluated on dataset [6] for our method and IBT are plotted versus the phase tolerance as provided in figures 5, 6, 7, and 8.

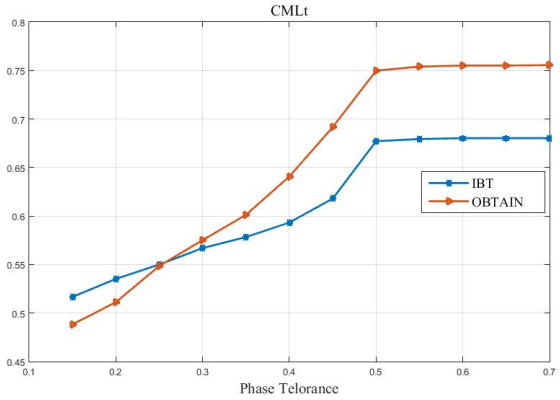

Fig. 5: CML_t vs. Phase tolerance.

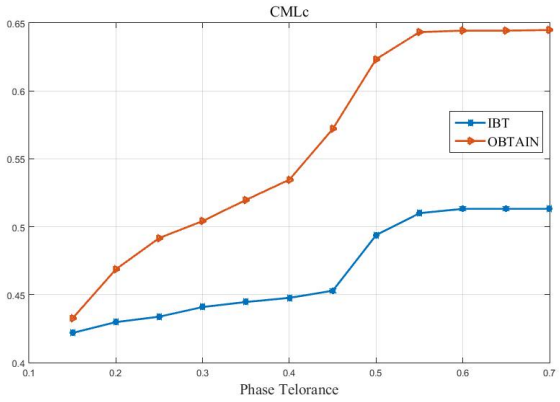

Fig. 6: CML_c vs. Phase tolerance.

IV. Implementation

We have implemented our method on an embedded system (Raspberry Pi 3) to perform the real-time beat tracking. We have selected Microsoft Windows the operating system for our software development platform. The algorithm developed in MATLAB Simulink may be converted to a C/C++ code with the cooperation of the algorithm developer and software developer, to make an implementation of the algorithm from

TABLE I: Comparison of performances of the two methods (in %).

method	AML_t	AML_c	CML_t	CML_c	$P-score$	$f-Measure$
OBTAIN on ICASSP SP cup dataset [6]	74.97	66.99	54.86	49.17	67.36	72.57
IBT on ICASSP SP cup dataset [6]	67.72	54.04	54.99	43.38	63.76	71.38
OBTAIN on ballroom dataset [10]	77.36	70.37	49.91	45.57	66.73	72.53
IBT on on ballroom dataset [10]	76.41	66.92	54.70	49.02	63.47	71.52

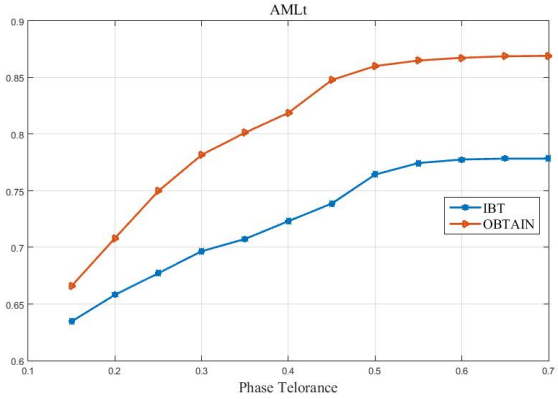

Fig. 7: AML_t vs. Phase tolerance.

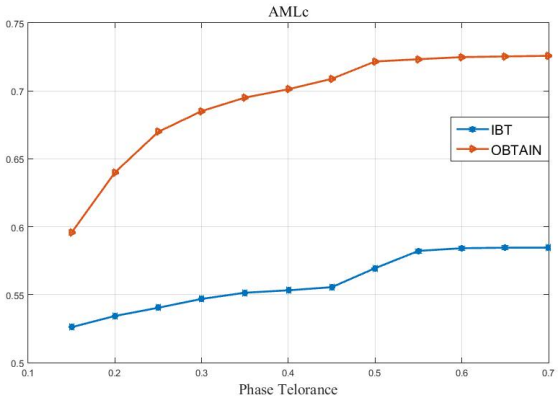

Fig. 8: AML_c vs. Phase tolerance.

scratch. The algorithm does not require any mathematical procedure more complicated than computing Fast Fourier Transforms (FFTs), so it would be easy to develop the software base to implement the real-time Beat Tracker. However, there is a great choice for direct implementation of Simulink blocks in software or even hardware (HDL). Simulink (and basically MATLAB) has the possibility to generate the software or hardware design that corresponds to block diagrams which can also contain M-file functions or scripts. We use this feature to generate the C code. After proper configuration of Simulink Code Generator and its solver, we are able to generate the C/C++ code that implements the exact same functionality. Basic mathematical operations and also some complex operations such as the FFT are directly performed in a plain source code, without using external libraries. Operations such as loading audio files and playback are implemented by connecting the generated application to MATLAB exclusive libraries. The most important thing in this procedure is the correct configuration of the Code Generator. For example, the solver should be set to discrete, fixed-step, and correct floating-point variable size must be selected. The recorded video of our system implementation can be accessed at [15] and the compressed version at [16].

V. Conclusion

In this paper, we propose an algorithm towards real-time beat tracking (OBTAIN). We use OSS to detect onsets and estimate the tempos. Then, we form a CBSS by taking advantage of OSS and tempo. Next, we perform peak detection by extracting the periodic sequence of beats among all CBSS peaks. The simulation results yield better results in comparison to IBT method. The algorithm outperforms state-of-art results in terms of prediction while maintaining comparable and practical computational complexity. The real-time performance is tractable.

VI. Acknowledgments

We appreciate the IEEE Signal Processing Society (SPS) to provide us with the opportunity of participating in IEEE SIGNAL PROCESSING CUP 2017 held by IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2017. The algorithm proposed in this paper was presented as Sharif University team algorithm and received honorable mention as one of the best teams with excellent beat tracking algorithm and annotation [17].

References

[1] D. P. Ellis, "Beat tracking by dynamic programming," *Journal of New Music Research*, vol. 36, no. 1, pp. 51–60, 2007.
[2] M. Muller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer, 2015.
[3] J. T. Mordkoff and P. J. Gianaros, "Detecting the onset of the lateralized readiness potential: A comparison of available methods and procedures," *Psychophysiology*, vol. 37, no. 03, pp. 347–360, 2000.
[4] R. Kiani, H. Esteky, and K. Tanaka, "Differences in onset latency of macaque inferotemporal neural responses to primate and non-primate faces," *Journal of neurophysiology*, vol. 94, no. 2, pp. 1587–1596, 2005.
[5] P. Grosche and M. Muller, "Tempogram toolbox: Matlab implementations for tempo and pulse analysis of music recordings," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR), Miami, FL, USA*, 2011.
[6] http://www.ee.usyd.edu.au/carlab/UserFiles/File/Downloads/training_set.zip.
[7] G. Percival and G. Tzanetakis, "Streamlined tempo estimation based on autocorrelation and cross-correlation with pulses," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1765–1776, 2014.
[8] A. M. Stark, "Musicians and machines: Bridging the semantic gap in live performance," Ph.D. dissertation, Citeseer, 2011.
[9] F. Scholkmann, J. Boss, and M. Wolf, "An efficient algorithm for automatic peak detection in noisy periodic and quasi-periodic signals," *Algorithms*, vol. 5, no. 4, pp. 588–603, 2012.
[10] [Online]. Available: http://www.audiocontentanalysis.org/data-sets/
[11] J. L. Oliveira, F. Gouyon, L. G. Martins, and L. P. Reis, "Ibt: A real-time tempo and beat tracking system," 2010.

[12] A. M. Stark, M. E. Davies, and M. D. Plumbley, "Real-time beat-synchronous analysis of musical audio," in *Proceedings of the 12th Int. Conference on Digital Audio Effects, Como, Italy*, 2009, pp. 299–304.
[13] M. F. McKinney, D. Moelants, M. E. Davies, and A. Klapuri, "Evaluation of audio beat tracking and music tempo extraction algorithms," *Journal of New Music Research*, vol. 36, no. 1, pp. 1–16, 2007.
[14] S. Dixon, "Evaluation of the audio beat tracking system beatroot," *Journal of New Music Research*, vol. 36, no. 1, pp. 39–50, 2007.
[15] https://youtu.be/GdmKvr1YZoA.
[16] https://youtu.be/hKMmpqaDYRc.
[17] http://www.ieee-icassp2017.org/sp-cup.html.