IMPROVING THE REAL-TIME PERFORMANCE OF A CAUSAL AUDIO DRUM TRANSCRIPTION SYSTEM

Marius Miron
INESC TEC, Porto, Portugal
mmiron@inescporto.pt

Matthew E.P. Davies
INESC TEC, Porto, Portugal
mdavies@inescporto.pt

Fabien Gouyon
INESC TEC, Porto, Portugal
fgouyon@inescporto.pt

ABSTRACT

In this paper we analyze and improve an audio drum transcription system with respect to real-time constraints. Furthermore, we propose a novel evaluation method, which allows us to systematically explore situations which are likely to occur in real-life drum performances. Then, we evaluate the architecture on a drum loops database, and discuss the influence of the size of the evaluation window, and of the classification method. Finally, we present the implementation in Pure Data and Max MSP, and propose a "do-it-yourself" technique which allows anyone to modify, and build a drum transcription system.

1. INTRODUCTION

Drum transcription aims to extract symbolic drum notation from audio signals. The task involves detecting a set of event candidates and labeling them. Additionally, a drum transcription system must face challenges related to the overlapping of events, presence of different sounds, or variation in timbre and amplitude of the drum events.

Assigning labels to drum sounds in audio recordings has been applied to two types of data: drum loops [1], [2], [3], [4], [5], and polyphonic music including a mixture of instruments [6], [7], [8], [9]. All of these systems work offline, and each one of them has a different way of solving the challenges of drum transcription. A classifier is used in [3] and [6], while other approaches rely on using a set of adaptive templates [8] and [9], or non-negative matrix factorization [7]. The issue of overlapping sounds was addressed in [5] by using decoy spectral templates along with the learned Gamma Mixture Model templates. Other methods, as [2] and [7], detect events in different frequency bands to separate different classes of sounds that might overlap.

Regarding online processing, a real-time classification algorithm for isolated percussion sounds, implemented in Pure Data, was proposed in [10]. The method trains a classifier using 300 features across ten overlapping windows, and does not address the subject of drum performances which are made of a mixture of sounds. Furthermore, another algorithm [11, p 101] implemented in Super Collider, is used for approximate bass drum and snare drum detection, as a part of a real-time beat-tracking system. This algorithm does not transcribe hi-hats, and was not evaluated in the scope of drum performance transcription. Both the Pure Data [1] and Super Collider [2] implementations are available as open-source.

We proposed a real-time system [12], along with an implementation for Pure Data and Max MSP. The algorithm detects drum onsets, extracts a set of features, and classifies each event as either bass drum (BD), snare drum (SD), or hi-hat cymbal (HH), from a mixture of the three classes, plus toms. Additionally, it uses an instance filtering (IF) stage to filter overlapping events fed to the three K-nearest neighbor (KNN) classifiers. Similar to other past methods,([6] and [8]) a global onset detector is used but the overall performance is improved by the IF stage.

The offline algorithms have a set of advantages because they can post-process the data, or use a large buffer. For instance, they can adjust onset times by shifting them backwards in order to start the analysis closer to the real onset of the drum, as in [8]. Tanghe et al. [6] proposed a system that works with streaming audio, but onset computation is not causal and the buffer used for onset detection is 105 ms, which is very large for drums. Moreover, during the MIREX 2005 contest, the time frame to locate an onset was $\pm$ 30 ms. Because of the symmetric window, the evaluation does not assume causality.

On the other hand, a system which can transcribe drums online can be useful in building machine listening driven interactive systems, in tasks such as instrument syncopation, or real-time beat tracking. This task is particularly challenging because a real-time system needs to be causal, and works under real-time constraints. Such a system faces several challenges. First, it should reliably transcribe audio as fast as possible. Furthermore, the minimum time necessary to detect a drum onset or compute features, might be lower than the time between two consecutive events. Additionally, the events might not be aligned to the metrical grid, and they might overlap.

We present an audio drum transcription system which responds to the challenges addressed above and has a better real-time response than the system discussed in [12]. Morevoer, we propose a novel evaluation of real-life situation, sound overlapping, and systematically explore it. For this purpose we design a database which comprises different levels of overlapping events. Additionally, we test

[1] http://williambrent.conflations.com/pages/research.html/
[2] http://www.sussex.ac.uk/Users/nc81/bbcut2.html

the system with a sequential K-means clustering algorithm which doesn't require a training phase, and classifies the instances on the fly. This algorithm requires less processing time, hence is a faster alternative when it comes to classifying data in real-time.

This paper is structured as follows. In Section 2, we present an overview of the existing system, as well as the system improved for real time. In Section 3, we evaluate the improved system along with the existing system. First we present a novel method, which allows us to systematically evaluate different levels of sound overlapping, which are likely to occur in real-life situations. Additionally, we evaluate both of the systems on a drum loops database. In Section 4, we introduce a modular implementation of the drum-transcription system, which can be easily modified for multiple live setups. Section 5 concludes this paper.

2. METHOD

2.1 Overview Of The Existing System

The existing transcription system, as described in [12], works with real-time stereo audio, sampled at 44100 Hz, and is implemented in Pure Data and Max MSP. The architecture of the system is depicted in Figure 1.

The algorithm uses a high frequency content onset detection stage to detect global onset candidates for the BD, SD and HH classes. For each detect event, it extracts three feature vectors, in the BD, SD, and HH frequency bands. These bands have fixed cutoff frequencies determined empirically: for BD we have a low-pass filter at 90 Hz, for SD we have a band-pass filter at 280 Hz with 20 Hz bandwidth, and for HH we have a high-pass filter at 9000 Hz.

The features are extracted over 10 overlapping frames of size 43 ms. The feature vectors are averaged with the energy in each frame. The salient part for extracting the features is 132 ms after the onset detection, but can be as small as 54 ms, if a new event comes earlier.

An instance filtering (IF) stage, for each of the BD, SD, and HH, comes next and filters the event candidates. The filtered event candidates are then fed to three KNN classifiers. The IF stage deploys a second onset detection stage, with higher frequency resolution in each of the BD, SD and HH frequency bands. Because for this stage we use a larger window size (1024 samples, 23.2 ms) and hop size (256 samples, 5.8 ms), thus there is better spectral discrimination, the IF is more efficient at filtering instances. The evaluation shows that the IF stage increases the performance of the existing system.

2.2 Improving The Existing System For Real-Time

There are three major improvements on the original system, which increase the real-time performance, and which are explained in this Section.

First, using separate onset detectors is a better solution when detecting class specific onsets because we are able to control the parameters of each onset detector separately.

Secondly, the system can give an output at any time, rather than waiting a response from the classifier. This is particulary useful when the interval between consecutive events

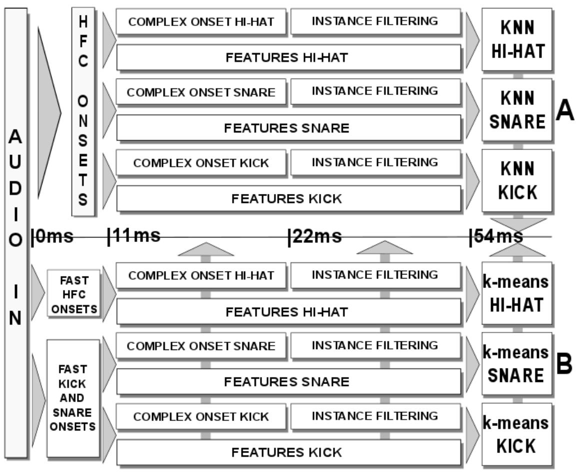

Figure 1: Comparison of the existing architecture (A), and the architecture improved for real-time (B), regarding the minimum response time

is smaller than the time to compute features or perform the IF. Additionally, during feature extraction, we do not take 10 overlapping 43 ms frames, but just one.

Thirdly, we use a sequential K-means clustering algorithm which is faster than the KNN classifier in a real-time situation [13, p 210].

2.2.1 Event Detection

Because the global onset detection can overdetect events for a class and misdetect for others, we propose an alternative to this system, using separate onset detectors than a single global one. By these means, it is less rather to have an overlapping situation since the event candidates are detected separately.

This fast onset detector uses a window size of 512 samples (11.6 ms) and hop size 128 samples (2.9 ms). Using such a small window does not yield a good enough frequency resolution to correctly separate between BD and SD. Moreover, the attack of the BD occurs at a higher frequency than the decay, and roughly in the same frequency band as the SD. Thus, we use a single complex onset detector in the SD band, 280 Hz with 20 Hz bandwidth, which captures the attacks of the BD and SD. We keep the high frequency onset for HH, since this type of drum has significant high frequency content.

2.2.2 Architecture

The architectures of the exiting system and of the one improved for real-time are portrayed in Figure 1, with respect to real-time constraints.

First, regarding the response time, the existing system can give a response in "maximum" of 143 ms. Because we need to give an answer as fast as possible, we extract features in a single window of 43 ms, instead of taking 10 overlapping frames. Thus, the improved system will give a response in maximum 54 ms: 11 ms, the time to detect an onset, plus 43 ms, the time needed to extract the features.

Secondly, the existing system has a "minimum" time response of 54 ms, the time needed to compute the features for the first window. Therefore, when a new onset occurs in the interval [0-54] ms, the analysis for the current event stops and the current event is missed. Thus, the existing system can not deal with situations when an event occurs earlier than 54 ms, and it misses events.

In the improved version, we would rather have an answer from the system rather than failing to detect events. If a new event occurs between 11 ms and 22 ms, then the system stops the current analysis, and offers the detected onset as an output. Similarly, if another onset is received between 22 ms and 54 ms, the system stops the current analysis, and offers as output the event filtered by IF stage. Thus, we reduced the minimum response time to 11 ms, the time needed to detect an onset.

In this way, the improved version deals properly with cases where the interval between two consecutive events is lower than the time needed to compute onsets or features.

2.2.3 *Sequential K-Means Clustering*

The existing system uses a binary KNN classifier for each BD, SD, and HH, in order to assign a class to each feature vector. Each instance is classified as member or non-member of the BD, SD and HH class.

The real-time performance of the classification algorithm depends on the number of points trained p. In this case, during the classification, finding the nearest neighbor takes $O(pn)$ time [13, p 210]. A faster alternative would be to have an algorithm which learns from the data gradually, rather than storing a database of learned instances. The sequential K-means clustering allows updating the centers of the clusters as new data points arrive. The time needed to classify an instance is $O(2n)$, assuming that we have two clusters. Thus, we obtain a faster, more real-time response by using the online K-means clustering rather than the KNN classifier.

The algorithm, as presented in [14], starts with the initialization of the initial means m_k, where k is the number of clusters. Having a new instance x, a distance is computed from x to each mean of each cluster. The closest cluster i is picked and its mean is updated with thew newly classified value $m_i = m_i + \frac{1}{n_i} * (x - m_i)$, where n_i is the number of the instances in a class. A "forgetful" K-means assumes replacing the counts $\frac{1}{n_i}$ with a constant $\frac{1}{a}$, and, thus, forgetting the older means by giving more weight to the recent instances.

We have three sequential k-means binary classifiers which can start with initial random means or can be initialized. Because we have binary classifiers, looking for either BD, SD, and HH, we initialize them with the features extracted from white noise filtered in the corresponding bands for BD, SD and HH. The parameters of the filters are as follows: the cutoff frequency for the low-pass filter is at 90 Hz, for the band-pass filter at 280 Hz with 20 Hz bandwidth, and for the high-pass filter at 7000 Hz. Because we are using a common onset detector for the BD and SD, both of the corresponding classifiers are initialized with white noise in BD and a SD bands. The HH classifier is initialized with white noise in HH and a SD bands.

3. EVALUATION

In Section 3.1, we present a novel method of evaluation for a drum transcription system by systematically analyzing the overlapping between events. We evaluate the existing architecture, and the one improved for real-time on the overlapping sounds database. Then, in Section 3.2, we test both of the architectures on the original drum loops database presented in [12]. Finally, we compare the performance of the k-means classifier with the one of the KNN classifier.

The evaluation window gives the maximum time deviation of a detected onset from the actual event. Because we impose real-time constraints, we set the size of the window to 18 ms, hence smaller than 35 ms, the one used in [12]. We analyze the consequences of choosing a smaller window size in Section 3.2.

The evaluation metrics, F-measure, F, precision, p, and recall, r, are described in [13, p 270]. Because we want to see distribution of precision across the BD, SD, and HH classes, we plot the $1 - p$ value instead of p. If this value is closer to 1, then the algorithm detects a high number of incorrect instances.

3.1 Drum Sounds Overlapping Analysis

Live performances are different from simple drum loops in terms of varying amplitudes and event displacement. We want to determine in which ways the existing transcription system described in [12] can be improved for real-time situations.

Furthermore, when building our system, we do not wish to make any assumptions about the metrical positions for any of the drum classes. We want to analyze every possible case. Thus, we build a database which contains systematic overlapping between BD, SD, or HH events. This database allows us to detect possible problems with our algorithm, when facing various situations that often appear in drum performances.

Moreover, the database is generated with sounds from a single drum kit. We are not using other drum kits because the only variable we want to analyze with this database is the sound overlapping. For the same reason, the effect of using the k-means classifier will be evaluated separately in Section 3.2.2. Therefore, we are using a KNN classifier for both of the evaluated systems.

Fewer instances in a KNN model result in faster performance [13, p 208] . In order to have a faster answer from the KNN classifier, we reduced the number of the training instances to 373, compared to 884 in [12], by removing different loudness excerpts when the sounds have the same timbre.

3.1.1 *Database*

We introduce a database made of different overlapping levels between various combinations of BD, SD and HH. Let $C = \{BD, SD, HH\}$ be an event with a duration of 100 ms. For each permutation of three C, we generate one file with a different interval between the events. Thus, we have a combination of events $E_i(a, b, c)$, where $a, b, c \in C =$

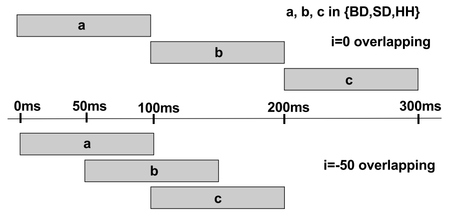

Figure 2: Two cases of overlapping sounds , $i = 0$ and $i = -50$

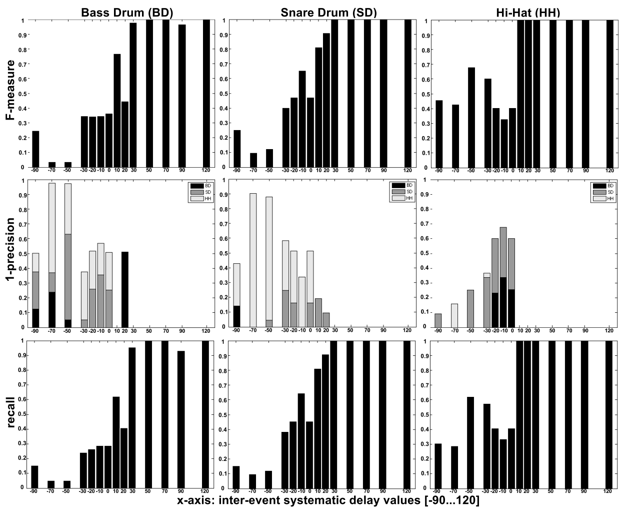

Figure 3: The original architecture, used in [12], tested with the overlapping sounds database

$\{BD, SD, HH\}$, and $i \in \{-90, -70, -50, -30, -20, -10, 0, 10, 20, 30, 50, 70, 90, 120\}$. For $i = 0$ an event starts where a previous event ends. For $i < 0$ the events overlap with i ms. For $i > 0$ there is no overlapping between events. An example of two situations is represented in Figure 2.

The database contains 210 MIDI files and the corresponding audio files, having nine events per file, in different combinations. The audio for the evaluation part is generated using Timidity++[3], by rendering midi drum loops from different genres, through a single drum kit.

3.1.2 *Existing System Evaluation*

Evaluating the existing system proposed in [12] , on this sound overlapping database, allows us to systematically analyze the behavior of the system along different variables. For instance, we can look at how the system performs when we have audio made of a single class of sounds, a combination of two sounds, or a combination of three different sounds. Furthermore, by comparing the performance on different overlapping levels, we can tell how well the system reacts in real-time.

The performance of the existing system when testing it on the overlapping sounds database is depicted in Figure 3. The system has few errors when sounds are not overlapping, $i > 0$. On the other hand, events are missed when

[3] http://timidity.sourceforge.net/

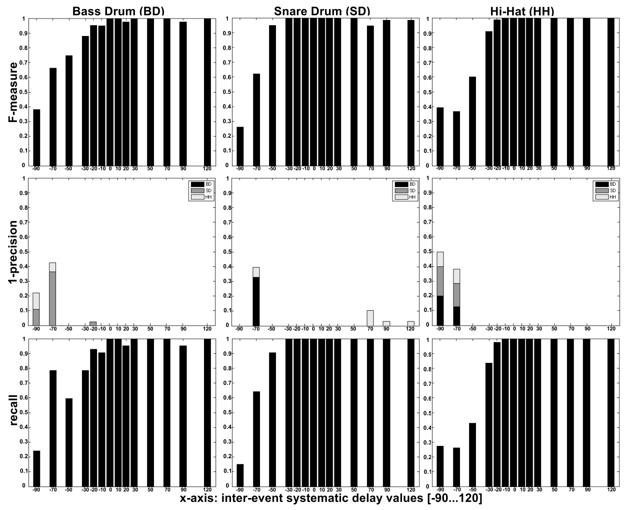

Figure 4: Results for testing the improved architecture with IF on the systematic sound overlapping database

sounds are overlapping, $i < 0$, even if the recall for onset detection is greater than 0.9 across all classes and delay times.

Furthermore, we can find important information about the system by analyzing the $1 - p$ distribution across all drum classes. For instance, we can tell that some events are wrongly classified as snares when we have hi-hats in audio. Moreover, there are few false positives in the -90 ms delay times because the interval between the two onsets, 10ms, is lower than the time resolution of the onset detection function. Thus, the onsets are detected simultaneously.

3.1.3 *Real-time Architecture Evaluation*

We tested the improved architecture, with the IF stage, on the overlapping sounds database. The results are presented in Figure 4. There is clear an improvement, in rejecting wrongly classified instances, and decreasing the number of the events missed.

The number of false positives significantly drops, especially for -50 ms delay time, when the system does not have the necessary time to compute the first feature window and classify the event. In this case, the interval between two consecutive events is 50 ms, and the minimum time to compute the features is 56 ms. For the -70 ms delay time, the interval between two consecutive events is 30 ms, and the algorithm exits with the output of the fast onset detector, because the IF stage needs 35 ms for processing. Finally, for event delay values of -90 ms, the interval is 10 ms, lower than the time needed to compute an actual fast onset, thus a lot of events are missed. In Figure 4, this yields a low value for recall.

3.2 Evaluation On The Drum Loops Database

The existing system [12] was originally tested on a drum loops database, comprising 177 drum loops, from different genres, at different tempi, and generated with 50 drum kits. First, we test the improved and the existing systems on this

	18 ms window			35 ms window		
	F	p	r	F	p	r
BD OG	0.76	0.88	0.67	0.81	0.93	0.71
BD RT	0.81	0.88	0.75	0.87	0.94	0.81
BD RTv	0.55	0.58	0.52	0.89	0.95	0.83
SD OG	0.74	0.85	0.66	0.78	0.88	0.69
SD RT	0.81	0.89	0.75	0.82	0.91	0.76
HH OG	0.77	0.90	0.67	0.81	0.93	0.71
HH RT	0.78	0.89	0.69	0.81	0.93	0.72

Table 1: The results of the original system (OG), and the improved systems (RT) with respect to the size of the evaluation window

database, by discussing the consequences of a smaller evaluation window, which imposes more real-time constraints. Secondly, we evaluate the k-means clustering method.

3.2.1 Evaluation Window Constraints

In [12], the size of the evaluation window was 35 ms, which is enough to capture all onsets, but not adjusted to the real-time constraints. Therefore, we want to see the if we can reduce this window to 18 ms, approximately half of the original size. We test the improved system (RT), along with the original architecture (OG), on the original drum loops database presented in [12], with window sizes of 18 ms and 35 ms. Furthermore, we test the hypothesis of using a separate onset detector for BD (RTv), which detects complex onsets below 90 Hz. We show that using a single onset detector for BD and SD is a better option, when dealing with real-time constraints.

As we mentioned in Section 2.2, because the attack of the BD occurs roughly in the same frequency band as the attack of SD, there is no reason to use separate onset detectors for BD and SD. This hypothesis is supported by the results in Table 1, when comparing the BD RT, the system with a common onset detector, with BD RTv, the system with separate onset detectors for BD and SD. There is a significant drop of performance when looking for the BD onsets in time intervals smaller than 18 ms. In the case of BD RTv, most of the correct BD events are detected in the [18-35] ms interval, on the decay of the sound, rather than on the attack.

Furthermore, the performance does not drop significantly for SD and HH, when comparing the OG and the RT systems across different window sizes. The onsets for the BD class are better detected in larger time span. As we showed above, this can be explained by the acoustic features of this drum.

3.2.2 Evaluation Of The Classification Method

We compare the performance of the system using a sequential K-means clustering algorithm (KM), with the one using a KNN classifier (KNN).

First we transcribe every file sequentially. This configuration is called KM ALL. The system learns continuously, as the means for each class are updated correspondingly when a new instance is classified.

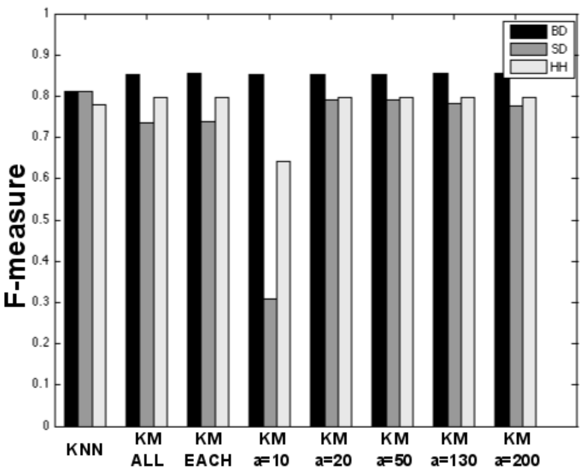

Figure 5: Comparison of the system using a KNN classifier (KNN), with the same system using the sequential K-means clustering (KM), in different configurations. ALL is running all all files through the K-means, EACH is resetting the K-means clustering before transcribing each file. Then we test the "forgetful" K-means configuration for different values of a

Secondly, in a real-life situation one has to use the sequential classification to detect the drum events performed with a single drum kit. We want to know how a sequential classifier performs regardless of the previous file. Thus we reset the K-means classifier each time a new file is transcribed. This configuration is called KM EACH.

Thirdly, we analyze the case of the "forgetful" K-means, as described in Section 2.2.3, and the influence of the constant a on the performance. We want to see how giving more weight to the recent instances affects the results. We assign several values for $a \in \{10, 20, 50, 130, 200\}$.

The results are presented in Figure 5. Using the K-means clustering algorithm gives better results than the KNN, when classifying BD. The F-measure is 4% higher for the KM. The results for the SD are lower with 3%, since other drums as toms occur in the same frequency band, thus classifying this type of drum requires some apriori knowledge about the spectrum of the sound. The results for the HH are similar for both of the KNN and KM. Additionally, resetting the K-means for each file KM EACH, does not decrease the performance.

Regarding the influence of the parameter a, the performance of the BD classifier is constant, regardless of the values of a. For the SD, the F measure shows that we need at least $a = 20$ instances for a good performance, but we get the highest value for $a = 130$. This happens because the SD needs to separate between many classes of events, including toms, thus it requires some long-term learning, rather than using the most recent instances. Furthermore, the HH class as well requires at least $a = 20$.

4. IMPLEMENTATION

The implementation of the transcription system can be downloaded as open-source from the Github repositories for Pure

Data [4] and Max MSP [5].

Users can choose from already built transcription patches or decide to built a new one. They can patch together different versions of the main modules: onset detection, feature computation, and classification. In this way, they can choose from different versions of the algorithm presented in this paper. For instance, they can use KNN classifier, or the sequential K-means. In the same way, the BD, SD and HH parts can use separate or common onset detectors, and different resolutions for the feature computation.

5. CONCLUSIONS

In this paper we proposed an audio drum transcription system improved for real-time performance. Additionally, we presented a novel evaluation which allowed us to systematically analyze the overlapping level between sounds. We discussed the problems of an existing causal drum transcription system, when facing real-time audio drum performances, and we proposed several improvements.

The evaluation shows that the improved system achieves better performance when dealing with different situations of overlapping sounds, which can occur frequently in real-life drum performances. Furthermore, the improved system achieves better performance than the original when tested on a drum loops database.

Finally, we proposed a "do-it-yourself" implementation in Pure Data and MaxMSP, which allows non-experts to build and customize their own drum transcription systems.

Acknowledgments

This work is supported by the ERDF – European Regional Development Fund through the (FCOM-01-0124-FEDER-014732) COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundacao para a Ciencia e a Tecnologia (Portuguese Foundation for Science and Technology) within project Shake-it, Grant PTDC/EAT-MMU/ 112255/2009. Part of this research was supported by the CASA project with reference PTDC/EIA-CCO/111050/2009 (FCT), and by the MAT project funded by ON.2.

6. REFERENCES

[1] D. Fitzgerald, R. Lawlor, and E. Coyle, "Drum transcription using automatic grouping of events and prior subspace analysis," *Proceedings of the 4th European Workshop on Image Analysis for Multimedia Interactive Services*, pp. 306–309, 2003.

[2] G. Tzanetakis, A. Kapur, and R. McWalter, "Subband-based Drum Transcription for Audio Signals," *2005 IEEE 7th Workshop on Multimedia Signal Processing*, pp. 1–4, Oct. 2005.

[3] O. Gillet and G. Richard, "Automatic transcription of drum loops," *International Conference on Acoustics, Speech, and Signal Processing ICASSP04, Montreal, Quebec, 2004*, pp. 269–272, 2004.

[4] ——, "Drum track transcription of polyphonic music using noise subspace projection," *Proceedings of the 6th International Conference on Music Information Retrieval*, pp. 92–99, 2005.

[5] E. Battenberg, V. Huang, and D. Wessel, "Toward live drum separation using probabilistic spectral clustering based on the itakura-saito divergence," in *AES 45th Conference: Applications of Time-Frequency Processing in Audio*, Helsinki, Finland, 2012.

[6] K. Tanghe, S. Degroeve, and B. De Baets, "An algorithm for detecting and labeling drum events in polyphonic music," in *Proceedings of the first Music Information Retrieval Evaluation eXchange (MIREX)*, 2005.

[7] J. Paulus and T. Virtanen, "Drum transcription with non-negative spectrogram factorisation," *13th European Signal Processing Conference*, 2005.

[8] K. Yoshii, M. Goto, and H. Okuno, "Drum sound recognition for polyphonic audio signals by adaptation and matching of spectrogram templates with harmonic structure suppression," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 1, pp. 333–345, Jan. 2007.

[9] V. Sandvold, F. Gouyon, and P. Herrera, "Drum sound classification in polyphonic audio recordings using localized sound models." in *ISMIR*, 2004.

[10] W. Brent, "Cepstral analysis tools for percussive timbre identification," *Proceedings of the 3rd International Pure Data Conference*, 2009.

[11] N. Collins, "Towards autonomous agents for live computer music: Realtime machine listening and interactive music systems," Ph.D. dissertation, 2006.

[12] M. Miron, M. Davies, and F. Gouyon, "An open-source drum transcription system for pure data and max msp," in *The 38th International Conference on Acoustics, Speech, and Signal Processing*, Vancouver, Canada, 2013.

[13] D. Hand, *Principles of data mining*. The MIT Press, Jan. 2001, vol. 30, no. 7.

[14] R. Duda, P. Hart, and D. Stork, *Pattern classification*, ser. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, 2001.

[4] http://github.com/SMC-INESC/drumtranscription_pd

[5] http://github.com/SMC-INESC/drumtranscription_maxmsp