Acoustic Drum-Kit Source Separation

with Inter-Limb Onset Detection for Timing Analysis and Training

Master's Thesis

Baldur Kampmann

Aalborg University
SICT

SICT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY

Title:
Acoustic Drum-Kit Source Separation with Inter-Limb Onset Detection for Timing Analysis and Training

Theme:
Sound & Music Computing, Master's Thesis

Project Period:
Spring Semester 2020

Project Group:
Individual Project

Participant(s):
Baldur Kampmann

Supervisor(s):
Cumhur Erkut

Copies: 1

Page Numbers: 128

Date of Completion:
May 24, 2020

Abstract:

This thesis contains research into methods for acoustic drum-kit source separation and onset detection for analysis of timing performance.
Taking into account the human motor and sensory system, a set of requirements are specified to guide the design and implementation of a prototype learning tool for use by drummers to improve their timing precision.
The prototype is evaluated from a system and user perspective and lastly the evaluation results and potential improvements are discussed.

Contents

Preface

Aalborg University, May 24, 2020

This thesis marks the conclusion of my journey from a decade-long career in sound engineering towards a future career in audio-related product development, where I hope to further explore my passion for sound and music, which has always motivated my work as a sound engineer.

While the transition has required hard work to grasp the concepts and methods that have enabled the use of digital computing devices to analyse, modify and enhance sound and music in so many ways, it has also been a path of personal development with both sacrifices, new friendships and lessons learned that hopefully will help me reach my long-term goals.

I would like to express my gratitude to Cumhur Erkut for his supervision throughout the thesis and for continually working to further the Sound & Music Computing programme together with Stefania Serafin, the faculty and staff. A special thanks to Sofia Dahl for additional literature and method suggestions.

I also wish to thank the user evaluation participants with a special thanks to Martin Pedersen for the drum performances used in the AV production and Teitur Árnason for use of camera equipment.

I would like to extend my sincere thanks to Allen & Heath Limited for the third semester internship position, which has been an immense help in relating my academic knowledge to real-world projects.

Finally, I would like to thank my family for their support during my education, thesis work and in life in general.

Baldur Kampmann
bkampm17@student.aau.dk>

Chapter 1

Introduction

As a musician, I have often wondered wherein the difference between a great and average drum performance lies. On one hand, you can find drummers with impressive technical abilities, but who for some reason do not elicit a musically gratifying performance. On the other hand, there are drummers, whose playing style is minimalistic, but who still manage to make the listener resonate with their performance. The distinction between these two cases is often attributed to the concept of *groove*, which Feld [1, p. 76] defines in a broad sense as:

> "[...] an unspecifiable but ordered sense of something [...] that is sustained in a distinctive, regular and attractive way, working to draw a listener in."

The vagueness of this definition illustrates that the concept of groove is difficult to do justice in a simple statement. Hofmann et al. [2] are somewhat more specific and state "that there is a common agreement that groove is a quality of music that makes listeners want to move or dance to the music".

Frühauf et al. [3] argue that although various musical styles require different timing strategies, such as playing a drum stroke slightly early or late relative to a musical timing grid, it is essential that drummers are able to deliberately manipulate their inter-pattern timing in order to induce a sense of groove - also referred to as playing "in the pocket" [4].

On the same topic, Iyer [5] is quoted by Danielsen [6, p. 9]:

> "In groove contexts, musicians display a heightened, seemingly microscopic sensitivity to musical timing (on the order of a few milliseconds). They are able to evoke a variety of rhythmic qualities, accents, or emotional moods by playing notes slightly late or early relative to a theoretical metric time point [...] All these musical quantities combine dynamically and holistically to form what some would call a musician's 'feel'. (Iyer 2002: 398)"

Drumeo instructor Dave Atkinson [7] provides an excellent video lesson, which demonstrates some of the issues and concepts mentioned above, and I urge the reader to watch the video to better understand the focus of this project.

The audio clips on the website[1] accompanying the study by Frühauf et al. [3] are also excellent examples of how timing modifications affect the perception of a drum performance.

Having established the importance of timing accuracy with regards to the perceived quality of a drum performance, I began to look into the current methods drummers apply to improve their timing skills and asked myself, is it possible to fast-track this process using digital signal processing and analysis?

After an initial research phase, it became clear that a range of timing practice tools exist on the market, but they are usually built into electronic drum-kits or similar devices. Software solutions also exist, but require the use of an electronic drum-kit or drum-pad as an input device. However, the temporal resolution of the feedback is often fairly coarse and the tools do not let the user study the results in detail after a performance.

Noticing a gap in the market for drummers, who play acoustic drum-kits, I decided to use the thesis as an opportunity to develop a prototype for this segment, which also attempts to improve upon some of the shortcomings of the currently available tools.

[1] http://musicweb.hmtm-hannover.de/groove/

Chapter 2

Problem Statement

An essential requirement for drummers is the ability to keep a steady rhythm, also known as sensorimotor synchronization (SMS). To obtain this ability, a metronome is often used during practice to establish an internal rhythmic sense. However, while practicing with a metronome helps with overall timing precision, it does not necessarily help the drummer improve the coordination of the hands and feet, which can result in rhythmic inconsistency and a musically suboptimal performance.

Although tools are available for electronic drum-kits and drum pads to help drummers improve their timing precision, these are not easily adapted to acoustic drums, which leaves out a significant number of potential users.

The aim of this project is to provide similar tools for acoustic drum-kit users and help increase drummers' awareness of their inter-limb timing precision by analysing the audio signal obtained from a monophonic microphone located in close proximity to the drum-kit. Following timing analysis, the user receives performance feedback on a graphical user interface based on the deviation from a known timing reference, i.e. a metronome.

System evaluation methods will be implemented to measure the performance of the underlying algorithms. By determining a suitable metric for comparing a drummer's performance prior and subsequent to practicing with the tool, the effect of providing timing feedback on a drummer's ability to perform sensorimotor synchronisation tasks will be evaluated within the constraints placed on user testing due to the COVID-19 lockdown in place during the finalisation of this project.

Chapter 3

Research

3.1 Timing

Timing in a musical context is generally understood as the ability of an individual to maintain synchronisation with an internal or external pulse as described by Janata et al. [8, p. 682]. The task is also known as sensorimotor synchronisation and the ability to perform this task, especially for drummers, is the foundation upon which music is built.

A subcategory of timing is microtiming, also referred to as expressive timing, which relates to minuscule variations in a musician's spacing of notes [9] and is generally what characterizes a human performance compared to a perfectly quantized computer-generated performance. The emotional expression in a performance can also be partly attributed to microtiming as the musician's decision to anticipate or lag certain notes can evoke a certain emotional response in the listener [10, p. 496].

Sometimes the two definitions will overlap in practice, since musicians might discuss the timing of a performance, but are actually referring to the underlying minute variations that change the "feel" or groove of a musical performance.

Research by Senn et al. [11] into microtiming and its effect on listeners, suggests that exaggerated microtiming deviations have a negative effect on groove ratings depending on the genre and the listener's musical expertise.

These findings are reinforced by Frühauf et al. [3], who found that when presented to a drum pattern where the kick and snare drum have been shifted in -25ms, -15ms, 0ms, +15ms and +25ms increments while the hi-hat is played exactly on the grid, music students rated the shifted performances worse than the 0ms version. The performances where the drums were shifted early were rated slightly lower than the late shifted performances.

Frühauf et al. argue that listeners prefer late shifts since it leads to a more relaxed "feel", whereas early shifts have a stressed "feel". Unfortunately, most people have a natural inclination to tap in advance of the beat when asked to synchronize to an external pulse. Research

indicates that the anticipation of the beat or negative mean asynchrony (NMA) is due to the difference in processing time of tactile and auditory information in the brain [12]. Studies also show that NMA differs between hands and feet, which indicates that the length and thus latency of the neural paths affects the timing strategies for the different limbs [12, p. 500].

An alternative theory presented by Thaut [13, p. 45] suggests that the tendency to anticipate the beat is a strategy to more efficiently compensate for inaccuracies in the motor system. Compared to lagging the beat and increasing the distance between the executed action and the actionable feedback, Thaut argues that it is more efficient to receive feedback immediately following the execution of an action as it optimizes the opportunity for self-correction for the upcoming event.

The reliance on an external pulse for self-correction could also explain why it usually is more difficult to synchronize to a metronome set to a slow tempo compared to a faster tempo and studies [14] shows that as IOI increases beyond approximately 2 seconds, non-musicians tend to produce increased NMA. However, while the standard deviation increases similarly to that of non-musicians as the IOI increases, trained musicians are able to maintain a near-constant mean asynchrony.

With the advent of music composing on computers, a testament to the importance of timing in music performances was the arrival of the quantization feature, which is used to align symbolic data in the MIDI format to a musical grid. An audio equivalent, which is able to locate and quantize onsets in multitrack audio recordings, was later introduced in the *Beat Detective*[1] tool available in the Avid/Digidesign Pro Tools[2] digital audio workstation (DAW). The introduction of the tool, which is now available in most digital audio workstations, has had a major impact on how modern music is produced as it made it possible to convert a mediocre performance into an expert performance at the click of a button. However, quantizing musical performances can have a negative effect on listeners rating of a performance as seen in the study by Hofmann et al. [2, p. 339], where it was found that listeners preferred performances with asynchronies below 19ms over fully quantized performances.

In summary, the research shows that timing plays an important role in listeners' rating of a musical performance, which suggests that it is beneficial for musicians to practice in a manner that improves their timing accuracy - whether their intentions are to achieve computer-like precision or to more precisely manipulate their microtiming in a specific manner fitting a particular musical style.

The research also indicates that it is useful to both consider timing in a broad sense - which can be determined from the mean asynchrony of the onsets - in order to provide insight into a drummer's ability to synchronise with the overall pulse, and also in a detail by evaluating the

[1] `https://www.soundonsound.com/techniques/pro-tools-using-beat-detective`
[2] `https://www.avid.com/pro-tools`

timing consistency on a microtiming level determined by the standard deviation of the onset asynchrony.

3.2 Drum Performance Basics

To better understand the factors that affect a drummers ability to perform consistently for an extended period of time, it is useful to look at the some common techniques, exercises and practice tools.

3.2.1 Drum Technique

There are essentially two schools on how to hold the drum sticks - traditional and match grip. The traditional grip is commonly used by jazz drummers and stems from marching band drummers, where the snare drum was carried with a strap across shoulder causing the drum to tilt to one side. By changing the grip, it was possible to hit the drum without raising the elbow into an awkward position [15].

The match grip on the other hand is probably the most common grip. It has the benefit of symmetry across both hands making it easier to achieve an identical motion between both hands and thus a similar sound when hitting the drum, regardless of which hand is used.

Fujii [16, p. 171] shows the ability to execute drum strokes at a high frequency is achieved by compliance in the wrist. Most research only considers wrist muscle activity when evaluating drum performances, but drumming exercise material [17] suggests that practicing specifically to strengthen the muscles in the fingers can help improve accuracy at higher speeds, since whether using traditional or match grip, most drummers will tend to switch from using their wrist to using their fingers as actuators to set the drum stick in motion as playing speed increases. While this reduces the maximum force that can be applied, it reduces fatigue and thus enables improved timing consistency at shorter IOI's.

Scientific research into feet tapping performance is limited, but according to Fujii et al. [12] "the NMA increases during foot tapping compared to manual tapping", which suggests that it is important to practice the feet in the same manner as the hands to achieve good coordination between all limbs, which is reinforced by drummer, Thomas Lang, who is renowned for his foot technique and author of the book *Creative Coordination & Advanced Foot Techniques* [18], in an article for Drum! Magazine [19].

3.2.2 Exercises

A set of exercises that are commonly used to improve drum technique are the so-called rudiments. Rudiments are exercise patterns that incorporate alternative strokes between both

hands, both feet, if playing a bass drum with a double-pedal, or a combination of hands and feet.

According to Meyer [20], an added benefit of practicing rudiments is improved sight-reading due to the rudiments consisting of common note-groupings, which leads to the drummer reading groups of notes rather than individual notes, much like reading words or groups of words rather than individual letters. Also, by alternating between which hand plays the first note in the rudiment, both hands are exercised - especially beneficial for the weaker hand.

There are forty standard rudiments that drummers should be acquainted with and an overview is available on the Percussive Arts Society's website[3].

It is useful to have an goal in mind when practising and Fujii et al. [12, p. 500] note that the data collected in their study can be used to determine a target performance level for music students. The data indicates that to perform comparatively to a professional drummer, the mean squared error (MSE) and synchronisation error (SE) should be below approximately 10ms.

3.2.3 Metronome

Fujii et. al [16] show that while non-drummers may be able to tap at the same rate as drummers, the asymmetry score of the onset distribution is significantly smaller for drummers, which is likely due to practicing in synchronization with a metronome or similar external pulse in a musical context.

A metronome set to generate an audio and/or visual pulse for each quarter-note of a given tempo is commonly used for synchronisation of multiple musicians and for practicing timing accuracy and consistency. Music students are therefore usually encouraged to practice with a metronome to improve their internal sense of timing.

Sometimes synchronization error is caused by the performer not being comfortable with performing to a metronome. A commonly employed technique to help ensure synchronization with an external audio pulse is to set the volume of the pulse to a setting, where it is masked by the sound of the drum-kit, when a drum hit perfectly matches the metronome.

This makes determining if sensorimotor synchronization has been achieved more intuitive and decreases the cognitive load of the task, since relating to the metronome is only necessary if the performer drifts out of synchronisation and has to self-correct.

A method to improve the internal sense of timing is known as *ghost clicks* or *quiet count* - an example of which is available in [21]. The concept is simple and the idea is to mute

[3] `https://www.pas.org/docs/default-source/default-document-library/pas-drum-rudiments-2018dcccc96de1726e19b`
`pdf?sfvrsn=fdbeaea5_6`

the metronome audio every other bar, so that the musician has to maintain a consistent tempo while the metronome is muted with the goal being that the synchronization is still intact when the metronome audio returns. As the performer's internal timing improves, the number of muted bars is increased.

3.2.4 Smart Timing Practice Tools

A tool that measures timing accuracy and provides performance feedback to the user is often referred to as a *Rhythm Coach* and can be found in certain metronomes[4] and even smartphone applications[5]. Using a sensor attached to a drum, the device measures the deviation of the user's performance from the metronome pulse used for synchronization. Electronic drum-kits often include similar features, but support input from multiple drums simultaneously. Some electronic drum-kits and drum pads also feature the *quiet count* method mentioned in the previous section. More recently, a software application, *Melodics*[6], expands upon the *Rhythm Coach* concept by gamifying drum practice.

There are however some shortcoming in the current timing tools. The metronomes only provide one sensor input, rendering it impractical to measure inter-limb timing, while the electronic drum-kits do not provide detailed information about the inter-pattern timing of the performance, but usually present the user with the total number of correct hits as a percentage value. *Melodics* requires an external MIDI input device and as such mainly caters to electronic drum-kit or drum-pad owners. The analysis is also limited to an indication of anticipated and lagging events on a piano-roll representation without a unit of measurement and does not provide a method for analysing the timing in detail after the performance.

3.3 Human Sensory and Motor System

According to Fujii et al. [16], the 2005 World's Fastest Drummer (WFD) competition winner was able to deliver just over 1200 drum hits per minute using both hands with drumsticks, formally referred to as bimanual stick drumming. This equals a inter-onset interval (IOI) of 50ms for bimanual or 100 ms for unimanual stick drumming.

According to Dahl [22], the fastest rate reported for double pedal playing, i.e hitting a bass drum alternating between both feet, is 1030 hits in 60 seconds, which is equal to an IOI of 117ms per foot.

These results provide a measure of the absolute minimum interval to be expected between hits and can be used in the detection algorithm to discard additional hits that lie within this

[4] `https://www.boss.info/us/products/db-90/`
[5] `https://makers4good.com/pages/backbeater`
[6] `https://melodics.com`

interval after the initial hit to reduce false positives in the onset detection algorithm.

However, since the goal of the prototype is to measure the asynchrony or deviation of each hit from a musical temporal grid, a higher resolution is required for the actual timing analysis. In [12, p. 496] the synchronization error (SE) of 15 professional drummers lies in the range from approximately -50ms to +50ms with a standard deviation of 10-15ms. The 2005 WFD winner mentioned above shows similar standard deviation values for SE, 9.5 and 6.6ms for the left and right hand, respectively. These figures suggest that the temporal resolution and accuracy of the onset detection method should ideally be less than 10ms to capture the asynchrony of expert drummers.

To reduce the risk of the user becoming confused or losing interest, any setup tasks required prior to a practice session must be as straightforward and quick to complete as possible. According to Doherty & Sorenson's paper on user experience in a computing setting [23], a user is likely to become inattentive, if a task takes more than 10 seconds to complete. For 5-10 second tasks, a user is likely to keep attention if progress feedback is provided. If a task takes less than 5 seconds to complete, the user is unlikely to lose attention. Ideally, processing time for the setup tasks would therefore be kept below 5 seconds per drum.

Another aspect to consider regarding the temporal resolution of the system is the limitations of the human sensory apparatus itself - more specifically the *window of simultaneity*, which Wykowska & Arstila [24] describe as the time range within which separate stimuli are perceived as one synchronous event. Their research suggests that the window size is 20-30ms for visual stimuli and approximately 4ms for auditory stimuli presented to different ears. This inherent limited ability to discern separate audio events should therefore theoretically limit a musician's conscious synchronization efforts to a resolution of approximately 4ms.

Echo threshold is another definition of the same phenomenon used by Brown et al. [25]:

> "The echo threshold is the briefest lead-lag delay at which subjects report perceiving "two sounds" or are able to accurately identify or discriminate the lag location on some criterion proportion of trials (e.g., on 50 % or on 75 % of trials"

An overview of various studies of echo thresholds is presented in the paper and for short burst signals most of the studies define the echo threshold as approximately 5ms.

This further supports using a 5ms temporal resolution for onset detection.

To summarize the findings on the human motor and sensory system as it relates to the features and requirements for the prototype:

- Minimum inter-onset-interval (IOI) = 100ms for each limb
- Synchronization error of expert drummers measured as standard deviation = ~10ms

- Task completion attention span <=5s
- Human visual temporal resolution = 20-30ms
- Human auditive temporal resolution = ~5ms

3.4 Automatic Drum Transcription Methods

The problem explored in this thesis is in many ways similar to the *Automatic Drum Transcription* (ADT) problem.

One of the first mentions of ADT can be found in the 1985 dissertation *On the Automatic Transcription of Percussive Music* by Walter A. Schloss [26]. Although the dissertation is more than 30 years old, it still provides useful insights into the intuition and problems related to ADT and shows that the although the field of ADT is constantly evolving and computing power is ever increasing, it still worthwhile to ensure that the chosen tools are a good fit for the problem at hand.

The operation of an ADT system can be divided into two parts - source separation and onset detection.

If the performance to be transcribed is monophonic, i.e. only a single drum-kit component is active, then the process is simpler, as it is only a matter of performing onset detection. Sometimes the performance is quasi-monophonic in the sense that the audio signal may contain multiple components, but only one is ever active at any given time. These situations require a method that can detect multiple components, but usually a minimum interval between events is specified to allow for classification of the event type.

However, in most ADT cases the signal is a polyphonic performance with a mixture containing multiple components with overlapping events. In some situations, there may also be pitched instruments in the mixture, but those cases fall into the domain of *Automatic Music Transcription (AMT)* of which ADT is a sub-problem.

In case of polyphonic performance, pre-processing is required using source separation methods to separate the components into individual audio streams.

The following provides a brief overview of some currently popular methods for source separation and classification as well as onset detection that were considered for the prototype. For a more complete treatment, the paper *A Review of Automatic Drum Transcription* by Wu et al. [27] presents the state of the art as of 2018.

3.4.1 Source Separation and Classification

3.4.1.1 K-Nearest-Neighbour Classifier

Stefanakis et al. [28], demonstrate an quasi-monophonic method to detect and classify audio events using a K-Nearest Neighbour Classifier with K=1.

Audio is captured via a microphone at a sampling rate of 22050Hz and STFT is performed with a Hanning window of 3ms and hop size of 0.73ms.

A template dictionary is generated using isolated audio recordings of the events to be detected. Audio events can be classified with 90% accuracy and a 5ms response time by comparing the real-time audio input to templates in the dictionary.

The method assumes no overlapping audio events, a minimum of 21ms between audio events and uses the so called "percussiveness" measure for onset detection.

Although the accuracy and response time is within range of the requirements for the prototype, the method is ultimately not a good fit, since it does not support polyphonic performances.

3.4.1.2 Bounded-Q Analysis

The software program Pure Data, commonly known as Pd [29], contains various tools for signal processing and analysis. Amongst these is Bonk, which is used to detect onsets and match them to pre-recorded spectral templates in quasi-monophonic performances.

The algorithm employed in Bonk is called Bounded-Q analysis [30]. The algorithm separates the audio signal into eleven individual streams using a filterbank and performs onset detection on each stream. The combination of onsets detected in the streams can then be compared to known onset combinations for various percussive instruments and the closest match is reported.

The minimum hop size possible with Bonk is 64 samples at 44.1kHz. This provides a temporal resolution of 1.45ms, which is within the prototype requirements. However, the algorithm does not support polyphonic performances and is therefore not suited for this project.

3.4.1.3 Non-Negative Matrix Factorization

Lee and Seung coined the term Non-Negative Matrix Factorization (NMF), which is described in their paper *Algorithms for Non-negative Matrix Factorization* [31]. The expression non-negative in this case refers to the overall assumption that the mixed signal is a sum obtained exclusively through summation of a number of components. Since the magnitude

spectrogram of an audio signal is inherently non-negative, it is readily applicable for source separation with NMF.

The NMF method is defined in [31] as follows:

Given a non-negative matrix V, find non-negative matrix factors W and H such that:

$$V \approx WH \tag{3.1}$$

In the case of drum source separation, the matrix V is usually a magnitude spectrogram obtained through a *Short-Time Fourier Transform (STFT)* of an audio mixture containing drum events.

The matrix W is the basis matrix, usually called the dictionary, which contains templates, i.e. the spectral fingerprint of each drum, more formally known as the components.

Similarly, H is referred to as the activation matrix and contains the activity of the components in the mixed signal.

The number of components that the algorithm attempts to isolate, also known as the rank or simply R, is defined by the user and in our case, the rank is equal to the number of drum-kit components in the mixture. If only standard drum patterns are to be identified, three components might suffice for identifying the kick, snare and hi-hat. However, additional components could be included for separation of a larger number of drums or to allow for timbre changes as a result of specific drum performance techniques.

A dictionary of drum sound templates is constructed by applying the NMF algorithm with rank R=1 to recordings of each drum-kit component in isolation, storing the W matrix in the dictionary and discarding the H matrix.

Subsequently for classification, the W matrix is initialized with the previously stored templates, while the H matrix is initialized either randomly or uniformly.

The W and H matrices are updated using an iterative method and the divergence between the estimated V and the actual V is minimized. The W matrix can optionally be fixed, which makes the source separation adhere more strictly to the content of the templates.

Lee and Seung proposed the use of a multiplicative update approach, which can be combined with a variety of divergence measurement methods, such as Euclidean distance, Kullback-Leibler divergence and Itakura-Saito divergence.

The mixture is ultimately decomposed into separate magnitude spectrograms for each component in the mixture and the phase information of the original mixture is combined with the spectrogram magnitudes. The resulting data is then reconstructed into separate audio streams by performing inverse STFT.

The NMFD method introduced by Smaragdis [32] adds a temporal aspect to the standard NMF method, which changes the NMF model to the following:

$$V \approx \sum_{t=0}^{T-1} W_t \cdot \overset{t\rightarrow}{H} \tag{3.2}$$

Whereas NMF assumes that a template is constant over time, the NMFD method expand the templates into MxT matrices, where M is the number of frequency bins and T is the number of continous STFT frames stored. This allows the algorithm to also take into account temporal variations in the templates when separating the components.

López-Serrano et al. provide the NMF Toolbox [33] for MATLAB, which includes an implementation of the NMFD algorithm by Smaragdis - although the T value is only used to fade a static template over time and therefore the implementation does not capture the actual temporal changes in the components.

3.4.1.4 Advanced Machine Learning Methods

As computing power has increased, development of ADT tools using advanced machine learning algorithms has expanded. The algorithms can typically be divided into two approaches:

- Algorithms trained on a large number of isolated audio recordings of the drums used within a mixture

- Algorithms trained on sequences of events, such as the drum pattern mixture itself

Şimşekli et al. [34] present a Hidden Markov Model (HMM) model, which is trained on isolated samples to detect and classify percussive events. Using an efficient HMM implementation, the method is able to operate in realtime, but does not support polyphonic audio.

Based on work on the previously described NMFD method, Smaragdis and Venkataramani [35] successfully created a linear autoencoder termed the Non-Negative Autoencoder (NAE). However, while the neural network adaptation provides more extensibility and better performance than standard NMF for high-rank decompostion using a multilayered NAE approach, it did not perform significantly better for low-rank decomposition, which makes it less useful for the three-component drum-kit source separation problem at hand.

Comparing RNN and NMF methods in the paper *A Review of Automatic Drum Transcription*, Wu et al. [27] state that the *lstmpB* method achieves the highest F-measure score of the RNN variants and similarly *NMFD* for the NMF methods - although it scores lower than all the RNN methods.

> "Based on our experiments, RNN-based methods seem to be the most promising approaches, and they are recommended when a large and diverse training dataset with high-quality annotations is available. NMF-based methods, on the other hand, provide decent performance with only little training data required; suitable for cases when large training datasets are not available." [27, p. 1479]

For the prototype to facilitate analysis of inter-limb accuracy in a user performance, it is important that the algorithm, in addition to correctly classifying overlapping drum events in a mixture, also provides accurate temporal onset locations. As seen in section 3.1, Frühauf et al. [3] show that temporally shifting individual drum-kit components as little as 15ms causes listeners to lower their rating of a performance. Since F-measure evaluations of ADT methods are often made with a fairly coarse window size of 50ms, as described in [27, p. 1466-1467], it is difficult to predict the performance of a given method at the increased temporal resolution required for the prototype to function properly in the intended use case.

Although neural network approaches typically outperform simpler source separation methods, while still achieving real-time operation, the main issue with neural networks is the amount of time required for training as noted in [36].

As mentioned in the research chapter, users can only tolerate a certain amount of waiting for setup tasks to finish. For the prototype, where the users train a dictionary using audio recordings of their own drumkit, the time required to complete training of a neural network would be detrimental to the user experience.

Training a neural network on a dataset containing a large number of generic drum samples to ensure sufficient generalization could be a workaround, but this would prevent the algorithm from taking into account the room acoustics and idiosyncrasies of the drumkit and microphone setup specific to a given user.

It could be argued that if the drumkit, acoustics, microphone position and audio settings remain constant between practice session, a one-time extended training procedure might be acceptable. However, a static setup is not considered feasible in real-life scenarios, since most musicians will regularly need to move their drum-kits for concerts and similar events.

After completing the initial research, a paper by Callender et al. [37] was published, which describes an adaptation of the *Onsets and Frames* project [38], which was initially trained to transcribe piano music, to perform ADT by training it on a new dataset, the *Expanded Groove MIDI Dataset* (E-GMD).

The E-GMD dataset was generated by expanding the GMD dataset [39], which contains a large number of human drum performances recorded as MIDI data, using the MIDI data to trigger

drum samples in 43 different synthetic drum-kits on a Roland TD-17 electronic drum-kit[7] and recording the resulting mixtures.

The original *Onset and Frames* project as well the adaptation, *OaF-Drums*, is based on the *Magenta* framework, which is powered by *TensorFlow* [40]. The training process for *OaF-Drums* took three days to complete using 16 tensor processing units (TPUv3).

Whereas the original *Onset and Frames* model splits onset and note activity detection into two separate stacks, the *OaF-Drums* model only uses the onset and velocity part. Further modifications are incorporated to tailor the model to percussive content. It is worth noting that a hop size of 441 samples is used resulting in a temporal resolution of 10ms. *Mir_Eval* [41] is used to calculate the *F-measure* [42] and as previosly mentioned, the criteria for a correctly classified onset requires it to fall within a 50ms window.

Although trained on synthetically generated audio mixtures, the approach is able to perform ADT on audio containing acoustic drumkit mixtures with support for transcribing velocity information.

3.4.2 Onset Detection

Following source separation, the resulting individual audio streams must be analyzed to determine the temporal location of onsets in the signal. Although this seems like a simple problem, onset detection has its own pitfalls starting with how to define an onset, since the perceived onset does not necessarily match the onset detected in the audio signal - also known as the perceptual center problem [43].

Even though percussive onsets are more pronounced than pitched instruments onsets, the onset location is a flexible definition and is usually specified as part of a study. For example, Fujii et al. [12] define the onset location as "the time at which the envelope exceeded 10% of the maximum amplitude of each sound burst".

Both time- and frequency-domain methods are available for onset detection with varying complexity. Most methods rely on peak-picking after an initial processing stage, in which the audio signal is reduced to an envelope with peaks at the temporal locations of the onsets. This can be accomplished either using a empirically determined fixed threshold or using an automatic threshold that is updated based on the signal content. A comparison of various methods is available in the paper *A tutorial on onset detection in music signals* by Bello et al. [44].

Assuming that percussive onsets are relatively constant between hits and if isolated audio recordings are available of the drum-kit components used in the mixture, an alternative solution to detecting the actual onset is to detect the peak and extrapolate the onset by subtracting

[7] `https://www.roland.com/global/products/td-17_series/`

a onset-to-peak lag value - similar to the method used in [12]. The lag can be determined through either manual or automated analysis of the isolated drum-kit component recordings. Detecting the peak reduces the risk of noise in the signal obscuring the exact onset location. It can also reduce the ambiguity of defining onset locations, since, if required, the lag value can be further adjusted manually to perfectly match the ground truth onsets - somewhat similar to the recording delay compensation settings available in some audio software, such as Logic Pro X[8].

[8] https://www.apple.com/logic-pro/

Chapter 4

Prototype Requirements

Based on the research and problem statement, a basic requirements specification was created for use as a guide during the implementation phase.

Purpose

The intended use case for the prototype is to provide an analytical tool for drummers to measure their timing performance relative to a metronome pulse without consideration for any genre-specific timing idiosyncrasies.

Although research shows that the timing accuracy of drummers changes with tempo [12] and that the location of an onset has an effect on the perceived timing [3, p. 254], the main goal is to provide an objective analysis of the incoming data and therefore the system will not take into consideration the tempo or the location of onsets in the rhythmic structure.

The underlying assumption is that the ability to synchronize with a metronome is essential regardless of the playing style and that the distribution of onsets on a microtiming level is an artistic choice. Nevertheless, to be able to make that choice, drummers must be aware of their timing.

However, if the user would like to practice a specific type of pattern or timing style, the ability to import a MIDI file with a new exercise drum pattern should be provided.

Features

- Facility for user to generate a custom drum template dictionary
- Setup process duration should be less than 5 seconds per drum
- Import of standard MIDI files containing exercise patterns to be used as ground truth data

- The sample rate of the audio processing should accommodate the frequency range of the drum-kit components
- Support for polyphonic drum performances
- Source separation of a monaural audio signal input into individual audio streams for bass drum, snare drum and hi-hat
- Onset detection performed on the individual streams to enable inter-limb timing accuracy evaluation
- Onsets occurring within 100ms of each other in each audio stream should be removed to reduce false positives
- Temporal resolution of onset detection algorithm should be at least 5ms
- User-friendly performance feedback in a graphical user interface showing detected onsets vs. reference onsets and timing accuracy per detected onset in milliseconds

Assumptions

Since the system cannot be expected to function correcly under adverse conditions, a set of guidelines to ensure optimal system performance was compiled. The guidelines in table 4.1 should be presented to the user during the setup process and could optionally be displayed to the user again in the case poor source separation and/or onset detection performance is detected.

Issue	**User Information**
Inadequate signal-to-noise ratio:	Reduce background noise
	Reduce microphone gain to avoid distortion
Template ambiguity:	Avoid using drums with a similar pitch or timbre
	Avoid drum patterns with overlapping events in similar drums
Temporal smearing:	Reduce drum resonances
	Reduce excessive reverberation

Table 4.1

Chapter 5

Prototype Development

5.1 Prototype Design

The main purpose of the current prototype is to facilitate an evaluation of the user experience, the performance of the source separation algorithm, the accuracy of the timing analysis and ultimately determine whether the user achieves improved timing performance as a result of performing exercises with timing feedback.

The design of the prototype therefore has two paths - user evaluation and system evaluation. The user evaluation aspect is functionally identical to what could be further developed into a release version. The system evaluation contains functions to test the underlying algorithms via benchmarking scripts and unit tests. Automated tests also enable comparison with other ADT methods using the IDMT-Drums dataset.

Although the ideal design process would be to finalize the design prior to the implementation process, due to the experimental nature of prototyping, it is expected that the implementation process and additional research will prompt modifications to the design.

5.1.1 User Evaluation Design

When starting the prototype software, the user must first complete a setup process. Next, the user is asked to examine the exercise pattern, listen to an audio preview of the pattern and perform the pattern in synchronization with a metronome. Initially, the user does not receive timing feedback, but after a predetermined number of exercise iterations, the user is presented with timing feedback on a graphical user interface following each exercise iteration.

To accomplish this, the following tasks must be implemented in software:

Setup

1. Record audio samples of each drum-kit component with automatically triggered record start and stop
2. Perform NMFD with rank = 1 and T = 8 to generate templates
3. Compile templates into a dictionary and store as a .mat file

Analysis

1. Import and convert a standard MIDI file to an suitable format for use as an exercise pattern and as ground truth data
2. Generate metronome audio pulses at a specified tempo
3. Record the user performance as an audio signal containing a mixture of the previously sampled drums played according to the exercise pattern in synchronization with the metronome.
4. Separate the drum-kit mixture into individual component audio streams
5. Detect onsets in the separated streams
6. Match detected onset to ground truth onsets
7. Calculate the timing error between the detected onsets and ground truth onsets
8. Depending on the test modality, plot the results to provide timing performance feedback to the user
9. Store detected onsets for subsequent analysis

Statistics

The overall steps are as follows:

1. Load detected onsets from files containing user performances with and without timing feedback
2. Perform timing analysis on detected onsets relative to ground truth data
3. Pool analysis result from the last three exercise attempts of each modality into two vectors corresponding to the modalities
4. Calculate statistic and present the results of each modality

In addition to calculating the mean and standard deviation of the onset distribution, the use of a combined metric that could express the difference between two performances as one score would be beneficial.

However, since the user's onsets may anticipate or lag the beat relative to the ground truth onsets, the data will potentially contain both negative and positive values, which could distort the results of analysis methods based on mean and standard deviation calculations if not processed correctly.

Therefore the final design of the statistical analysis will be determined as a result of analysing the data and experimenting with different metrics.

5.1.2 System Evaluation Design

To verify correct operation of the implementation during development and to evaluate the performance under different conditions, a number of system tests are required that programmatically modify the ground truth data and other test variables prior to generating synthetic audio mixtures to be used as input to the prototype.

The source separation and onset detection process is accomplished by utilising the same functions as during the user evaluation to keep the results aligned.

The list below shows the basic tasks of the system tests:

1. Import and extract onsets from a standard MIDI file
2. Modify test variables programmatically on each iteration as required for the selected test
3. Perform NMFD with rank = 1 and T = 8 to generate templates using drum samples from the selected synthetic drum-kit
4. Compile templates into a dictionary and store to a *.mat* file
5. Generate a synthetic audio mixture from the MIDI data
6. Perform source separation and timing analysis using the synthetic mixture
7. Optionally plot the timing feedback for verification of correct operation
8. Compare test iteration data and present results

A method to handle audio and ground truth data from the IDMT-Drums dataset [45] is required to enable comparison of the prototype to other ADT methods. The process is outlined below and is slightly different to the other system tests due to the format of the data in the dataset.

1. Import dataset audio files and ground truth onset data
2. Segment training data into individual events and extract a single event for each drum-kit component
3. Calculate onset-to-peak lag in the drum-kit component audio data
4. Perform NMFD with rank = 1 and T = 8 on audio data for each drum-kit component to generate templates
5. Compile templates into a dictionary and store to a *.mat* file
6. Perform source separation and timing analysis using the dataset mixtures
7. Optionally plot the timing feedback for verification of correct operation
8. Export the detected onsets and the ground truth onsets to text files suitable for F-measure calculation using Mir_Eval

5.2 Prototype Implementation

MATLAB [46] was chosen as the development environment to facilitate efficient prototype implementation using the built-in tools in MATLAB and third-party toolboxes, such as the NMF Toolbox by López-Serrano et al. [33], the MIR Toolbox by Lartillot et al. [47] and the MIDI Toolbox by Eerola & Toiviainen [48].

Having established the overall design in section 5.1, the first step of the implementation process was to divide the prototype into parts based on the intended use and designating a main function file for each part.

The main function names and usages are listed in table 5.1 and correspond to the overviews in figures 5.1 and 5.2.

Function Name	**Usage**
- setupDictionary.m	Record drum samples and generate dictionary
- performAnalysis.m	Record user performance audio and perform analysis
- sourceSeparation.m	Separate mixture into individual component streams
- detectOnsets.m	Detect onsets in separated streams
- matchOnsets.m	Compare detected onsets to MIDI onsets
- plotResults.m	Plot the timing feedback presented to the user

Table 5.1

Additional helper functions were created for various operations, such as converting from tempo in beats per minute to inter-onset-interval in milliseconds, which are called from the functions listed above.

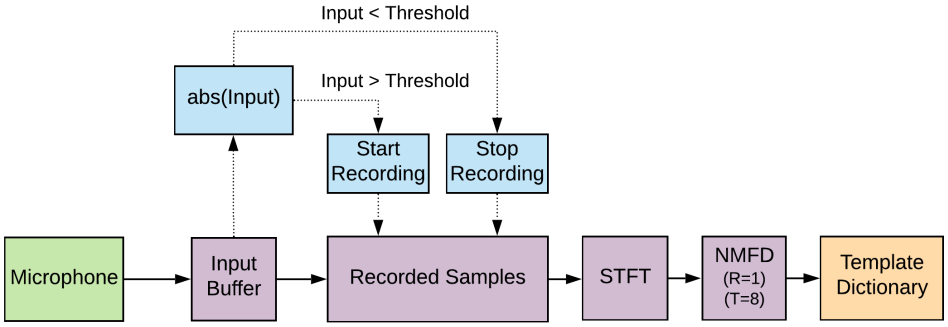


Figure 5.1: Overview of the NMFD dictionary setup process with start and stop of audio recording controlled using the audio signal level with pre-determined thresholds. The process is repeated for each drum-kit component.

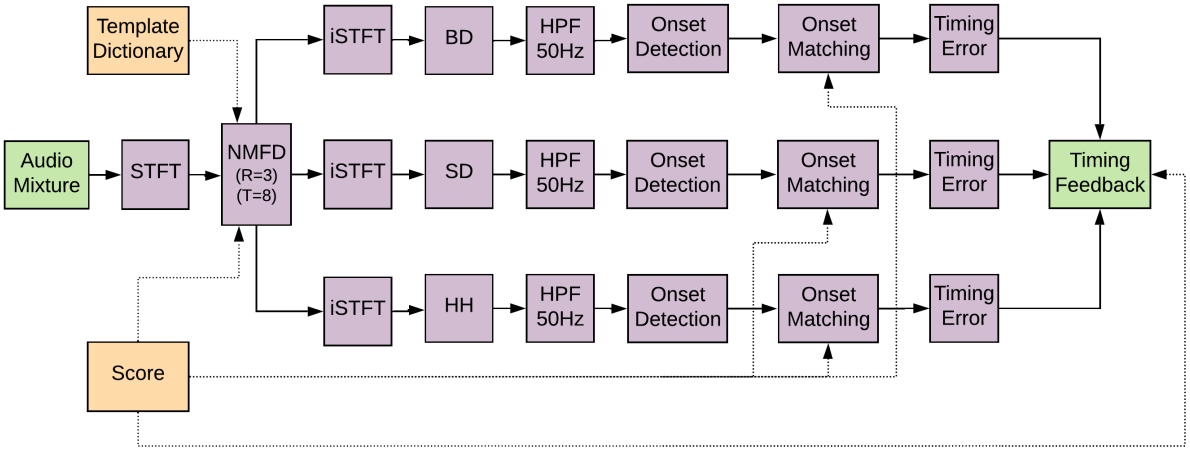


Figure 5.2: Overview of the prototype system from input to output. The score onset locations is used as a starting point for the activation update process for score-informed separation

In order to create a seamless evaluation process and ensure repeatability, two scripts were created to automate the calling of the above functions during the user evaluation and the system evaluation process named *userEval.m* and *sysEval.m*, which are available in appendix K and J, respectively. A separate script was created for the the IDMT-Drums dataset test, which is available in appendix L.

The source code is available in the supplementary material.

5.2.1 Source Separation

Based on research into the state-of-the-art source separation methods and due to the requirement for efficient template training, the Non-Negative Matrix Factor Deconvolution (NMFD) approach described in [32] was chosen for source separation.

To speed up the implementation phase, the NMF Toolbox by López-Serrano et al. [33] provided the required functions. In addition to other NMF variants, the NMF Toolbox supports the NMFD method utilising Kullback-Leibler divergence for the multiplicative update function.

As mentioned in section 3.4.1.3, the temporal aspect of NMFD in the NMF Toolbox is implemented in a fashion that fades out the same spectrogram drum sample over T number of frames by multiplying it with a coefficient that decreases to zero over T iterations.

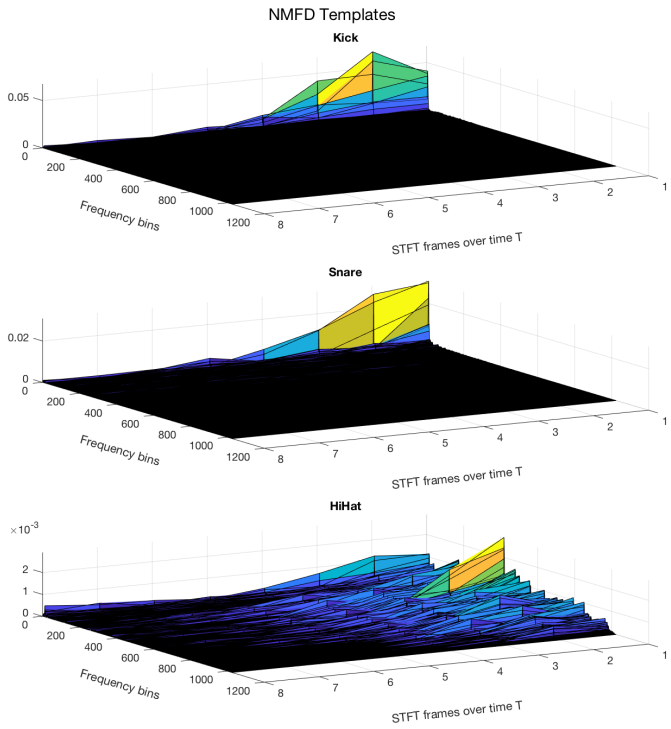


Figure 5.3: NMFD templates showing normalized audio signal energy over T=8 STFT frames

Some modifications were therefore required to the toolbox to create templates that contain the required number of frames extracted from the actual drum spectrograms and to enable the algorithm to use the expanded templates during the source separation process. Additional information about the NMF Toolbox modifications is available in appendix I.

An example of the content in NMFD dictionary templates is shown in figure 5.3. While it can be argued that the spectrum of a drum following the attack portion is more or less a fade-out over time, the simplified approach in the original implementation would likely perform worse with toms and similar drums that have a pitched component in the sustain period. Being able to discern changes in the pitch component over time could potentially improve the separation quality when similar sounding drums, such as bass drums and toms, coexist in a mixture.

Various settings for the NMFD algorithm were tested and notably, using adaptive templates over fixed templates did provide better perceived separation quality, but also more cross-talk making it less suited for onset detection, which is in line with the findings by Dittmar & Gärtner in [45]. This prompted the use of fixed templates in the prototype.

5.2.2 Multi-threading and Buffering Considerations

Since MATLAB does not support multi-threading, a limitation presented itself with regards to simultaneously recording audio, performing the source separation and timing analysis as well as plotting the results.

Ideally, the calculations and the plotting would be completed during each buffer period and using sufficiently large buffers, this is indeed possible with a correspondingly large latency between input and output. Benchmarks with different audio buffer sizes showed that a buffer size greater than 8192 samples was required to avoid buffer underrun.

While pseudo-realtime performance can be achieved, another observation was that performing the NMFD processing on shorter audio segments as would be required for a real-time implementation, resulted in a degradation of the source separation quality. This is most likely caused by the number of components in the mixture varying in each frame, which confuses the algorithm when it expects to find activations for all of the components specified, similar to the issue mentioned in section 4.2 in [45].

Although the NMF Toolbox implementation provides an option for score-informed processing, i.e. the ground truth onsets are used as a starting point for the activations, it does not fully solve the issue. Another approach would be to dynamically update the expected number of components for each frame to correspond with the components active in the mixture at a given time as per the ground truth data. Exploring this further was however beyond the scope of this project.

In [45], Dittmar & Gärtner successfully implemented a real-time NMF-based ADT system as a VST[1] plugin, which performs source separation and onset detection with a systemic delay of approximately 6ms. This shows that a real-time implementation of the system is possible, but will require a different approach than currently used in the prototype.

Seeing that a processing lag was unavoidable either due to buffer size latency or the need for a large frame size for the NMFD algorithm for the prototype, it was decided to first record a complete exercise pattern and then process the entire recording to avoid the issues described above, which also bypasses the need to finish plotting in time for the next audio buffer being acquired. The exercises can then be repeated indefinitely with the added benefit of allowing the user to solely focus on playing the drums during the exercise - and having unlimited time to study their timing performance between each exercise attempt.

5.2.3 Automatic Thresholding

Initially, a fixed threshold value was empirically determined in conjunction with the built-in *findpeaks* MATLAB function used to detect peaks in the separated audio streams. However, a

[1] `https://www.steinberg.net/en/company/technologies.html`

fixed threshold requires tuning depending on the incoming signal and relying on the user to set a threshold for each drum would be detrimental to the user experience.

An automatic threshold feature was therefore implemented based on detecting the envelope of the audio signal with two separate detectors by applying the *movmean* function in MATLAB to a rectified version of the original audio.

A slow detector was tuned to smooth out peaks leaving an envelope that shows the trends in the signal over time and a fast detector was set to smooth the rapid fluctuations in the signal, but leave the prominent peaks intact as seen in figure C.1.

As seen in equation 5.1, averaging the difference between the envelopes after adding an empirically determined offset results in a vector containing the automatic threshold values. Figure C.2 shows the threshold overlaid on the rectified audio signal.

$$\begin{aligned} &average_{fast} = m(|X|), K = 5000 \\ &average_{slow} = m(|X|), K = 500 \\ &diff = average_{fast} - average_{slow} \\ &autoThreshold = m(diff + offset), K = 500, offset = 0.2 \end{aligned} \tag{5.1}$$

where X = all audio samples and
m is the *movmean* MATLAB function with window size K

In equation 5.2, setting the output to one when the rectified audio signal is above the automatic threshold value and zero otherwise, we get a vector of binary states with the same length as the number of samples in the audio signal.

$$g(x) = \begin{Bmatrix} 0 & |x| <= autoThreshold \\ 1 & |x| > autoThreshold \end{Bmatrix} \tag{5.2}$$

Multiplying the rectified audio signal with the vector as per equation 5.3 either mutes the audio or passes it through unchanged resulting in a cleaner signal for the *findpeaks* function to process eliminating the need for a fixed threshold. Figure C.3 shows the audio signal before and after thresholding is applied.

$$y(X) = |X| * g(X) \tag{5.3}$$

where X = all audio samples and $*$ denotes element-by-element multiplication

The MATLAB implementation can be seen in appendix F.

5.2.4 Onset Detection

The currently used onset detection method simply attempts to find the largest peaks in the audio signal separated by the minimum IOI value of 100ms. Therefore, a slight discrepancy can occur if a given drum sound has an extended attack duration and is compared to a ground truth onset.

However, since the separated audio might contain low-level spurious noise from overlapping audio events, it was decided to use the audio signal peak instead of attempting to locate the exact onset. The logic behind this is that audio recordings of each drum in isolation are stored during the NMFD dictionary setup process and assuming that the onset to peak distance remains constant for each drum-kit component, it is possible to automatically extrapolate the onset location from the peak location.

As seen in appendix G, the value of the audio peak is calculated from the isolated recordings using the *max* function in MATLAB and then the onset location is found by assuming that the onset is placed at the start of the recording at the temporal location where the rectified audio level is equal to 1% of the peak value. Trimming the audio so the first sample is at the onset location, the audio peak sample index is equal to the onset-to-peak lag value, which can then be used to compensate for the difference in the detected onset locations compared to the ground truth onsets. This method is similar to the approach used by Fujii et al. [12], where an onset threshold of 10% of the peak value is used.

Another benefit of detecting peaks instead of onsets is that the audio signal can be pre-processed with the automatic threshold function described above. If detecting the actual onsets, their exact location would likely be shifted due to low-level signals being muted, when below the threshold as seen in figure C.3 in the appendix.

The onset detection utilises the *findpeaks* function in MATLAB, which accepts a number of parameters to tune the detection process, such *Minimum Peak Height*, which was set to the default 0 (i.e. disabled), *Minimum Peak Distance*, which was set to 100ms based on the IOI requirements established during the research phase and *Minimum Peak Prominence*, which was empirically determined and set to 0.3. The remaining optional parameters were left at their default values.

If required, the automatic threshold function described above could be further developed to also handle peak detection and since the peak locations are already segmented and separated by zero value samples, it would be trivial to use the *max* function to detect the peak locations.

Initially, the system detected peaks in the rectified audio signal, but after additional system testing it became clear that attempting to find peaks in the rectified audio without smoothing

the signal was causing onsets to be incorrectly detected by a couple of milliseconds as shown in figure D.1. Inspecting the signal revealed that the issue occurred when the absolute peak of a given hit had shifted due to the source separated signal waveform changing shape slightly when two drum-kit components were overlapping in the mixture.

An exponential moving average (EMA) filter [49] was therefore used to smooth the thresholded audio signal prior to processing with *findpeaks*.

As seen in equation 5.4, the EMA filter is a simple recursive filter [50]. The amount of smoothing is set with the parameter a, which controls the weighting or ratio between the input and the previous output inside the filter. The input and output of the filter are designated x and y, respectively. The variable t is the current time step in samples.

$$y(t) = a \cdot y(t-1) + (1-a) \cdot x(t) \tag{5.4}$$

For the hi-hat, setting *a=0.95* provided the best results, while *a=0.9991* worked best for the snare and bass drum. After smoothing the signal, the setting for the *findPeaks* function needed to be modified by reducing the *Minimum Peak Prominence* setting to 0.03 for the hi-hat and 0.001 for the snare and bass drum. Also, since the filter introduces a delay in the signal depending on the value of a, this must compensated for in the detected onset location values.

The result is that the overall detection consistency has degraded somewhat, but the detection algorithm is more robust to changes in the shape of the audio signal peaks. The trade-off between a higher precision, but less stable system versus a more stable, but less precise system is worth considering when introducing real-world data, which is more unpredictable than the synthetic data used for system evaluation. The EMA filter was implemented after the preliminary and revised user evaluations, but was used for the system evaluation.

5.2.5 Onset Matching

To calculate the accuracy of the onsets detected in the user performance relative to the ground truth or reference onsets, an onset matching method is required.

The concept is to simply calculate the Euclidean distance from each detected onset to each reference onset. Finding the minimum value returns the error or asynchrony in samples between a given detected onset and its closest matching reference onset. The resulting asynchrony values are later converted to milliseconds and used as accuracy indicators for each drum-kit component in the timing feedback visualisation as seen in the second row of figure 6.21.

The number of detected onsets are not always equal to the reference onsets. For example, the player might hit a drum too many times or miss a hit. Also, noise might distort the signal that

triggers the peak detection algorithm resulting in excess onsets or the input signal might be too low resulting in missing onsets.

Currently, duplicate and missing detected onset are reported as-is in the timing feedback to keep the user's performance aligned with the feedback. It could be argued that removing duplicates onset from the timing feedback could improve the user experience, but further user testing is required to determine if this is the case.

However, when analysing the data after the fact and calculating statistics, it can be useful to perform post-processing to ensure fair comparisons between datasets and reduce clutter during visual inspection of the data. The functions described below were therefore added as options, which can be activated by setting the correct parameters when calling the function *matchOnsets*. The implementation is available in appendix H.

In the case of duplicate onset, it is usually safe to assume that the detected onset closest to the reference onset should be left intact and any excess onsets should be removed. This is accomplished by iterating detected onsets, which have been matched to the same reference onset during the onset matching process, and only retaining the onset with shortest distance to the reference onset.

For missing hits, a simple solution is to interpolate the missing onset asynchrony value. Determining whether an onset is missing can be accomplished by inspecting the list of detected/reference onset pairs and looking for reference onsets without a matching detected onset. When a missing onset is found, the asynchrony values of the adjacent onsets are used to calculate the mean, which is then inserted as the asynchrony value for the missing onset.

In addition to removing duplicate onsets and inserting missing onsets, it can also be useful to remove outliers. Since the absolute distances from each detected onset to its closest reference onset are already known and stored as a vector, the vector can be searched for values larger than a specified distance limit and the resulting indices used to remove onsets accordingly.

Chapter 6

Prototype Evaluation

6.1 System Evaluation

The goal of the system evaluation is to gather data on the performance of the source separation algorithm and the onset detection implementation. The system evaluation is intended to run semi-automatically enabling the simulation of a wide range of scenarios. The system evaluation operations are consolidated in a single script, *sysEval.m*, which is presented in appendix J.

For clarity, a separate script was created for the IDMT-Drums dataset test, which is available in appendix L.

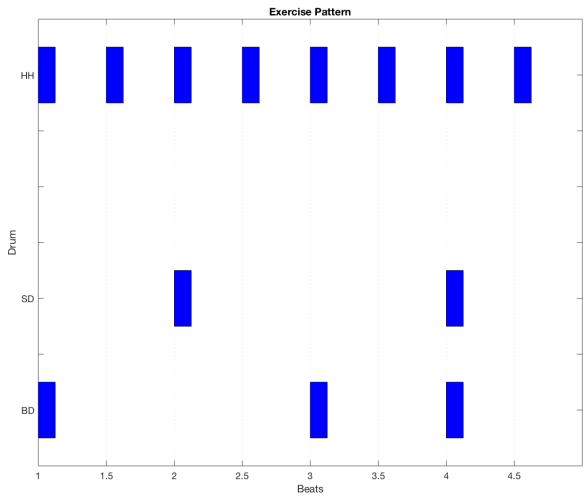


Figure 6.1: Piano-roll representation of the original drum exercise pattern

Symbolic representations of drum patterns, such as the one in figure 6.1, are imported from standard MIDI format (*.mid*) files to both serve as ground truth data and as input data for generating synthetic audio mixtures that simulate a drummer playing a drum-kit. Using symbolic data enables programmatically manipulating parameters of the drum pattern, such as the tempo, prior to generating the synthetic audio mixtures. The template dictionary setup process is also included in the automated test to enable evaluating the effect of different drum-kits on the system performance.

Comparing the analysis results of each test iteration, it is possible to evaluate the quality of the source separation and the timing analysis process.

A specific test is selected by setting the variable *testMode* accordingly.

6.1.1 Method

Based on the list of tasks compiled in the design phase in section 5.1.2, the following method was established:

1. Set the user-specified test modality, such as tempo or pattern, via the *testMode* variable
2. Import drum pattern as MIDI data and use as ground truth data
3. Modify the ground truth data programmatically
4. If the drum pattern is the independent variable, import MIDI data currently specified in test
5. Generate a synthetic audio mixture using the modified ground truth data onsets and pre-recorded drum samples
6. Set up a template dictionary with currently selected drum samples (Only necessary to repeat if independent variable is the drum-kit)
7. Perform source separation and timing analysis on the synthetic audio mixture
8. Compensate for onset-to-peak lag
9. Compare ground truth data onsets with detected onsets
10. Present results as plots of onset detection accuracy and range
11. Repeat steps 3 to 10 as required by the specified number of test iterations

6.1.2 Results

6.1.2.1 Baseline Performance

Prior to automated testing with different test modalities, the baseline performance was established using ideal drum samples, i.e. drum-kit components recorded in a studio environment without background noise, distortion or excessive reverberation, referred to as drum-kit 1 (see scalograms in figure 6.3). The scalograms show that the peaks in the spectral signature of the samples are distinct and spaced evenly throughout the frequency spectrum with little overlap between the individual drums.

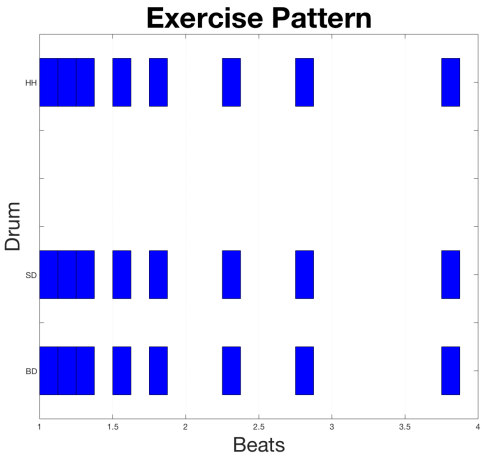


Figure 6.2: Piano-roll representation of drum pattern used for baseline performance tests

A test was implemented in the system evaluation script (*testMode = 'combinations'*) that tests the consis-

tency of the system with increasing inter-onset-intervals (IOI) ranging from 100ms to 800ms as seen in the drum pattern in figure 6.2. The range can then be inspected to ensure it is lower than the 5ms temporal resolution limit required as defined in section 4.

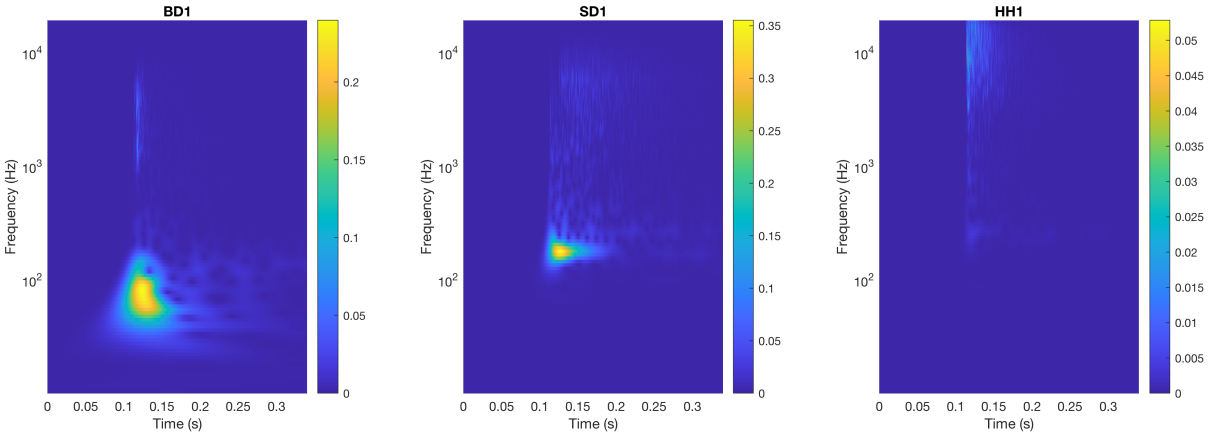


Figure 6.3: CWT scalograms of reference drum-kit samples for bass drum, snare drum and hi-hat

Ideally, the accuracy value should stay constant regardless of the IOI and thus the range should be as close to zero as possible. This proves that the algorithm is able to consistently discern between onsets with changing IOI's.

Using range as a metric removes differences caused by the onset-to-peak lag in the drum samples and exclusively reports discrepancies in the system. However, for clarity and to enable direct comparison of plotted results, the option to automatically compensate for the onset-to-peak lag as described in section 5.2.4 is used to normalize the data, so perfectly timed onsets should have an accuracy value of 0ms regardless of the onset-to-peak lag of different drums.

As seen in figure 6.4, the snare drum has the largest range value at 1.22ms, but is still well within the temporal resolution limit.

Testing the same pattern with a 50ms lag deliberately added to the snare drum (*testMode = 'combinations-staggered'*), figure 6.6 shows that all snare drum onsets are correctly detected as shifted in time by 50ms. Figure 6.5 shows that the range has decreased due to the bass drum and snare drum not overlapping as in *testMode = 'combinations'*.

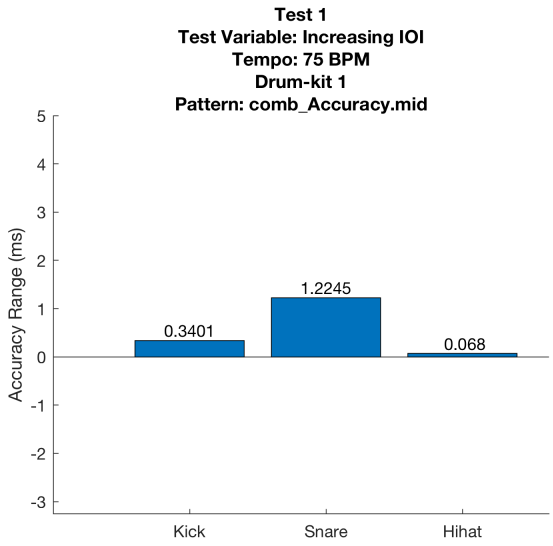


Figure 6.4: Onset detection consistency as range with increasing IOI's

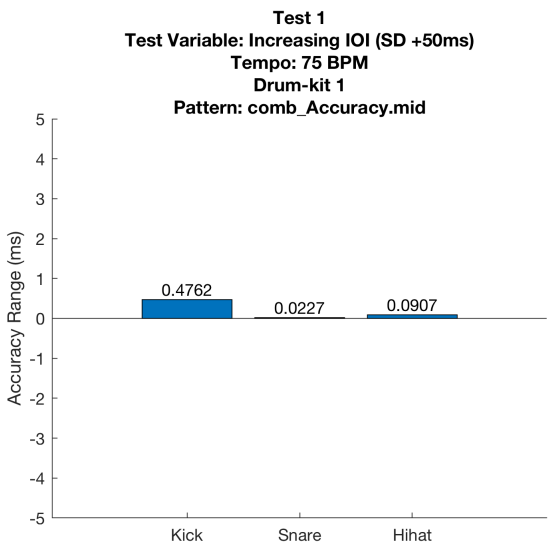


Figure 6.5: Onset detection consistency as range with increasing IOI's and snare drum shifted 50ms

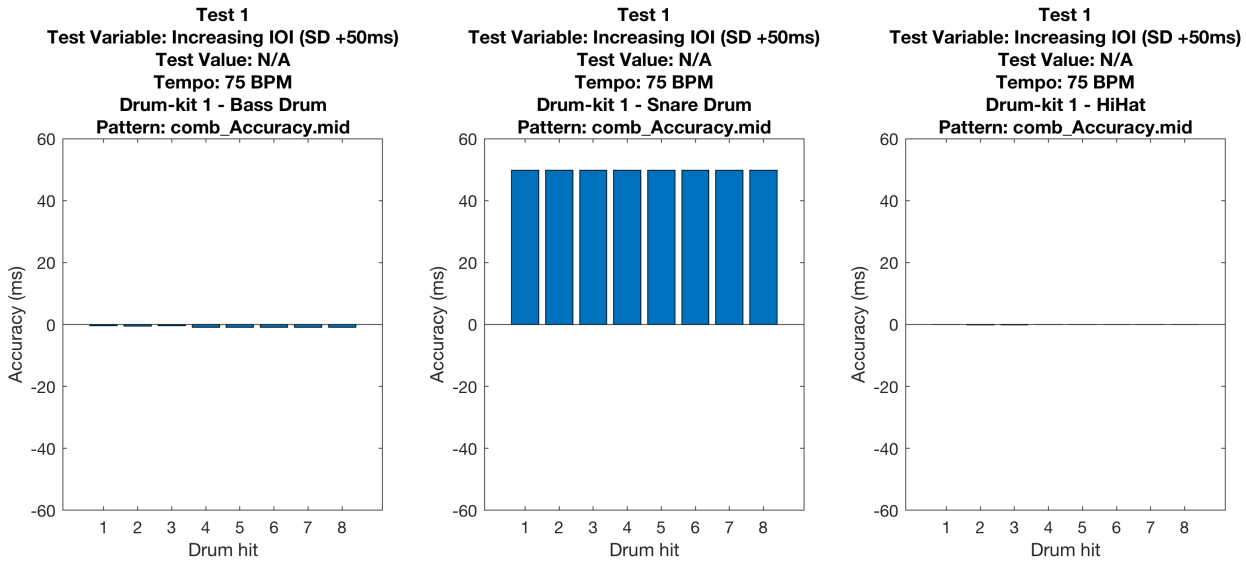


Figure 6.6: Onset detection accuracy with increasing IOI's and snare drum shifted 50ms

6.1.2.2 Inter-Limb Onset Detection Accuracy

The prototype must be able to detect minute inter-limb timing discrepancies in order to analyse the coordination of the hands and feet. The exercise pattern in figure 6.7 was created to test the inter-limb asynchrony detection capabilities of the system with various combinations of simultaneous events likely to occur in a drum pattern.

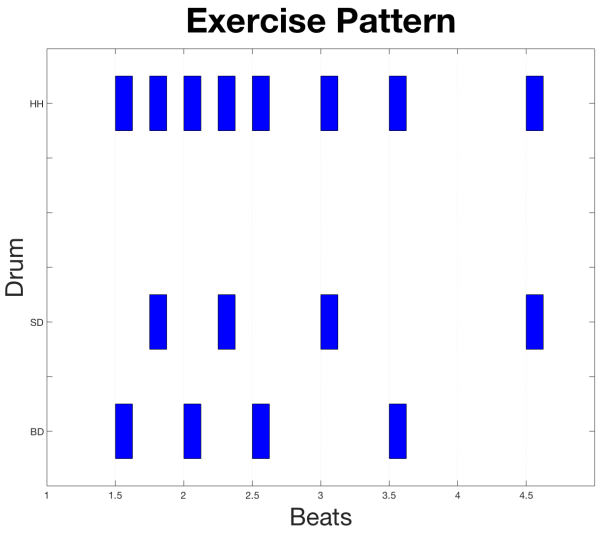


Figure 6.7: Piano-roll representation of the drum pattern used to test the system's inter-limb onset detection capabilities

In figure 6.8, a series of results are shown for the *'testMode=inter-limb'* test, where the drum pattern is modified so the second and third bass are shifted backwards in time and the second and third snare drum hits are shifted forwards in time, while the hihat is unmodified. The test was performed with shifts of 1, 2.5 and 5 ms to simulate the expected inter-limb timing asynchrony of an expert drummer. The results show that the algorithm is able to detect errors as small as 1ms in both the bass and snare drum - although with a 1ms overall shift in the accuracy value for the bass drum, which is the result of a slight error in the onset-to-peak lag detection. See figure D.2 for additional plots of the 1ms test.

In the following, test results are presented for each of the currently implemented automated tests, *pattern*, *tempo*, *drum-kit* and *velocity*.

6.1.2.3 Pattern as Independent Variable

In addition to the drum patterns used for the baseline tests, the *Simplified*, *Shuffled* and *Combinations* patterns shown in figure B.1 were used to test the system with the pattern as the independent variable (*testMode=pattern*). The patterns were designed to evaluate different aspects of the system; performance with basic rhythms with minimal overlap of drums, performance with shuffled rhythms and performance with different combinations of drum-kit components being active simultaneously.

Comparing the results in figure 6.9 shows that performance is nearly identical for the first two patterns. There is a slight inconsistency in the bass drum in the first two patterns and for the third pattern there is also decreased consistency in the snare drum for the second and third events - a result of the bass drum and snare drum being active simultaneously. However, all the values are still within the 5ms temporal resolution limit.

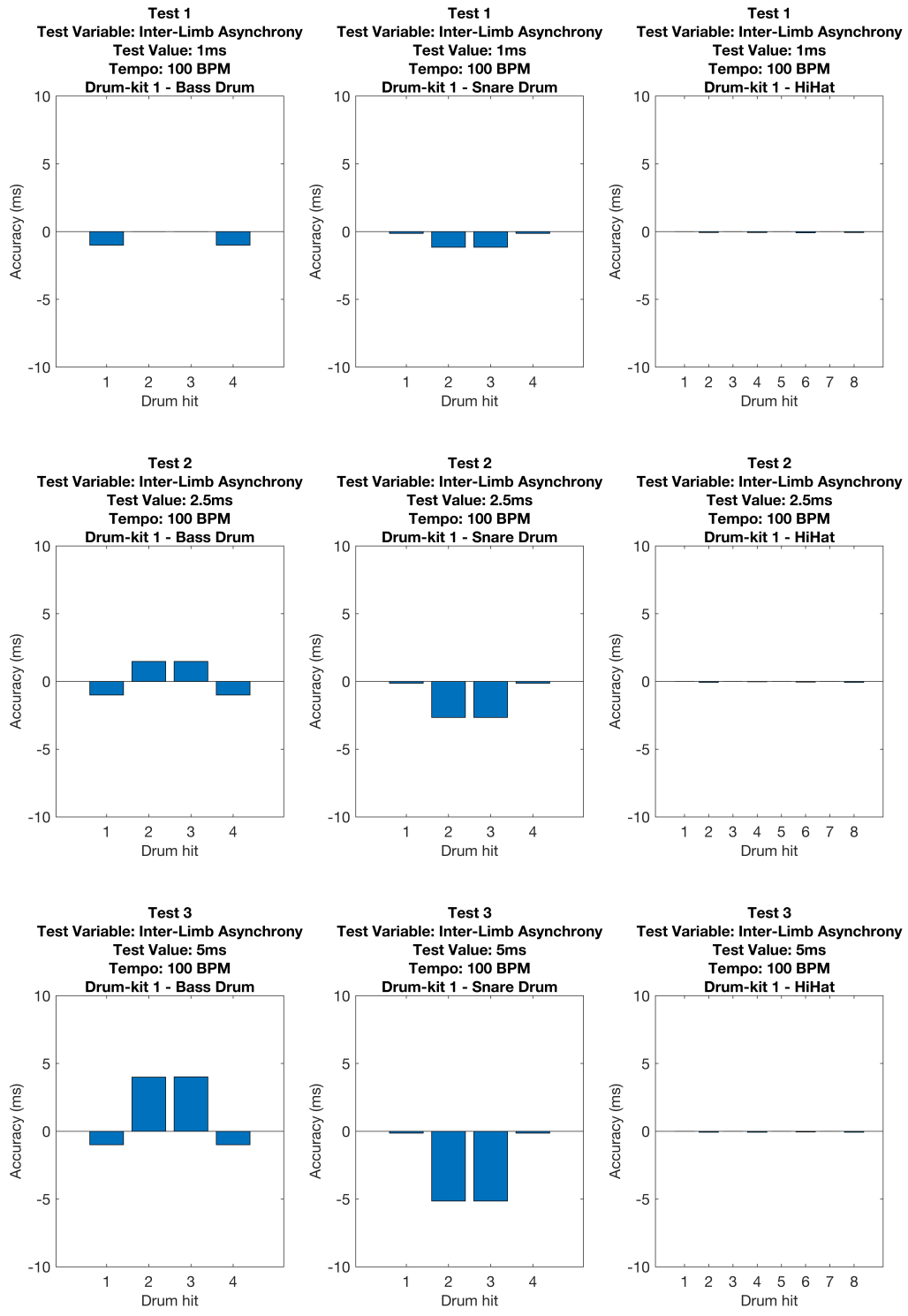


Figure 6.8: Inter-limb onset detection accuracy with asynchronies of 1, 2.5 and 5 ms to simulate an expert drummer

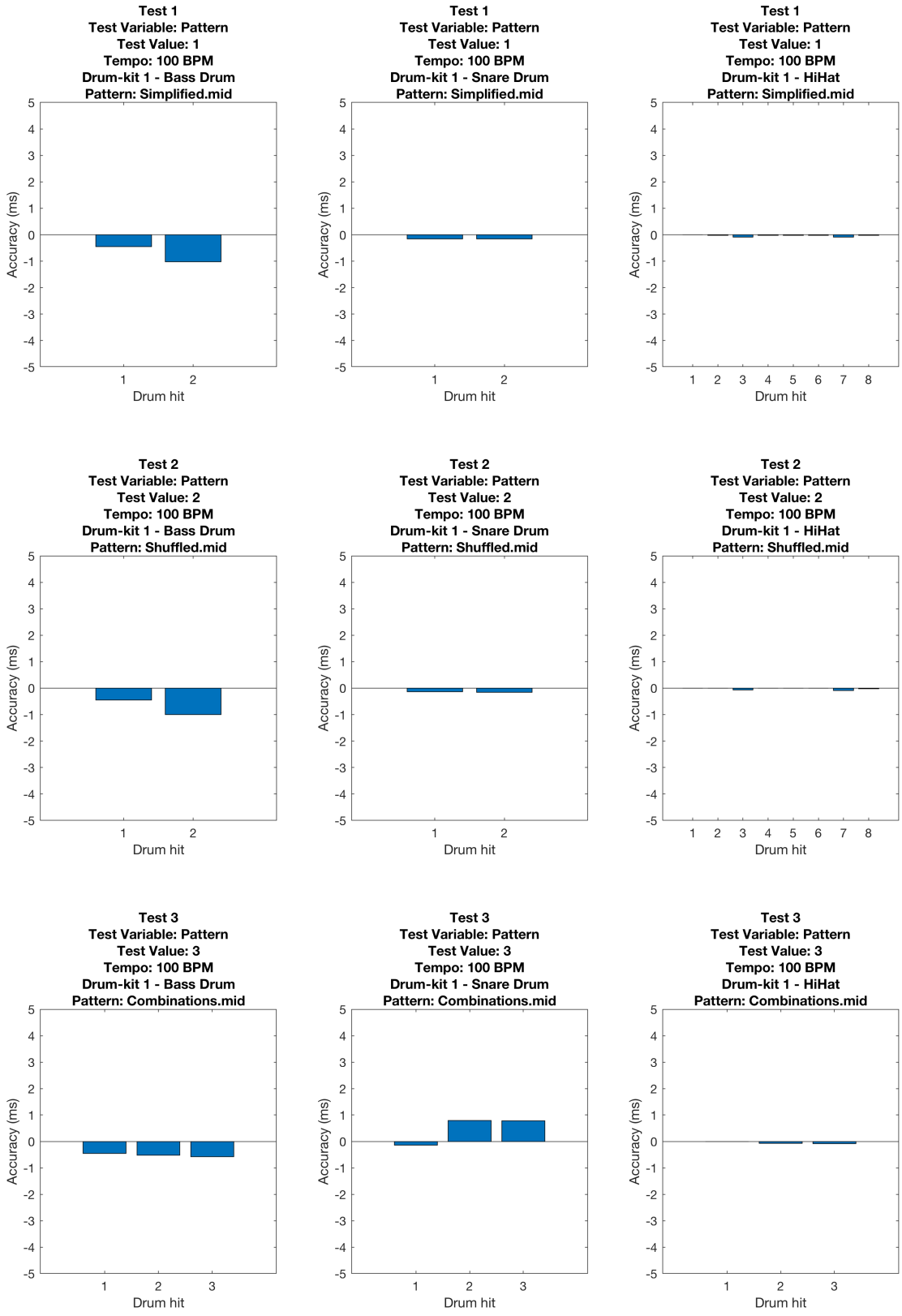


Figure 6.9: Onset detection accuracy with different drum patterns

6.1.2.4 Drum-kit as Independent Variable

Synthetic drum-kits were generated for system testing using audio recordings of various drum-kits, which were gathered as follows:

- Acoustic drum-kit recorded with an iPhone 6S in a large theater
- Virtual drum-kit supplied with the Logic Pro X digital audio workstation software
- Acoustic drum-kit recorded with a MacBook Pro's built-in microphone in a medium-sized, reverberant room - the same drum-kit and room as used for the user evaluation.

Figure 6.11 shows the onset detection accuracy result for the bass drum, snare drum and hi-hat using all three drum-kits (*testMode=drumkit*). While the results are within the temporal resolution limit, a marked increase is seen in the snare drum in the third test.

While the accuracy value shows an error of approximately 3ms, the range value for the snare drum in the third test in figure 6.10, is 0ms, which indicates that the problem is caused by an issue in the onset-to-peak lag calculation. Comparing the snare drum scalograms in figure C.4 shows that the energy distribution in *SD2* is more diffuse than in *SD0* and *SD1*. This could be caused by either reverberation in the recording or due to insufficient damping of drum resonances. Both conditions can make the peak indistinct and thus reduce the precision of the envelope extracted by the exponential moving averager.

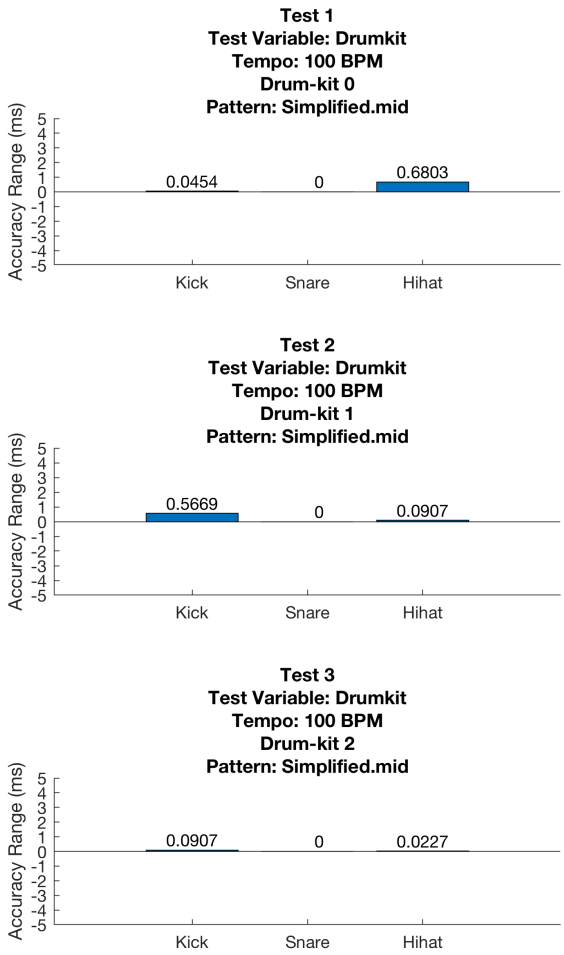


Figure 6.10: Onset detection consistency as range with different drum-kits

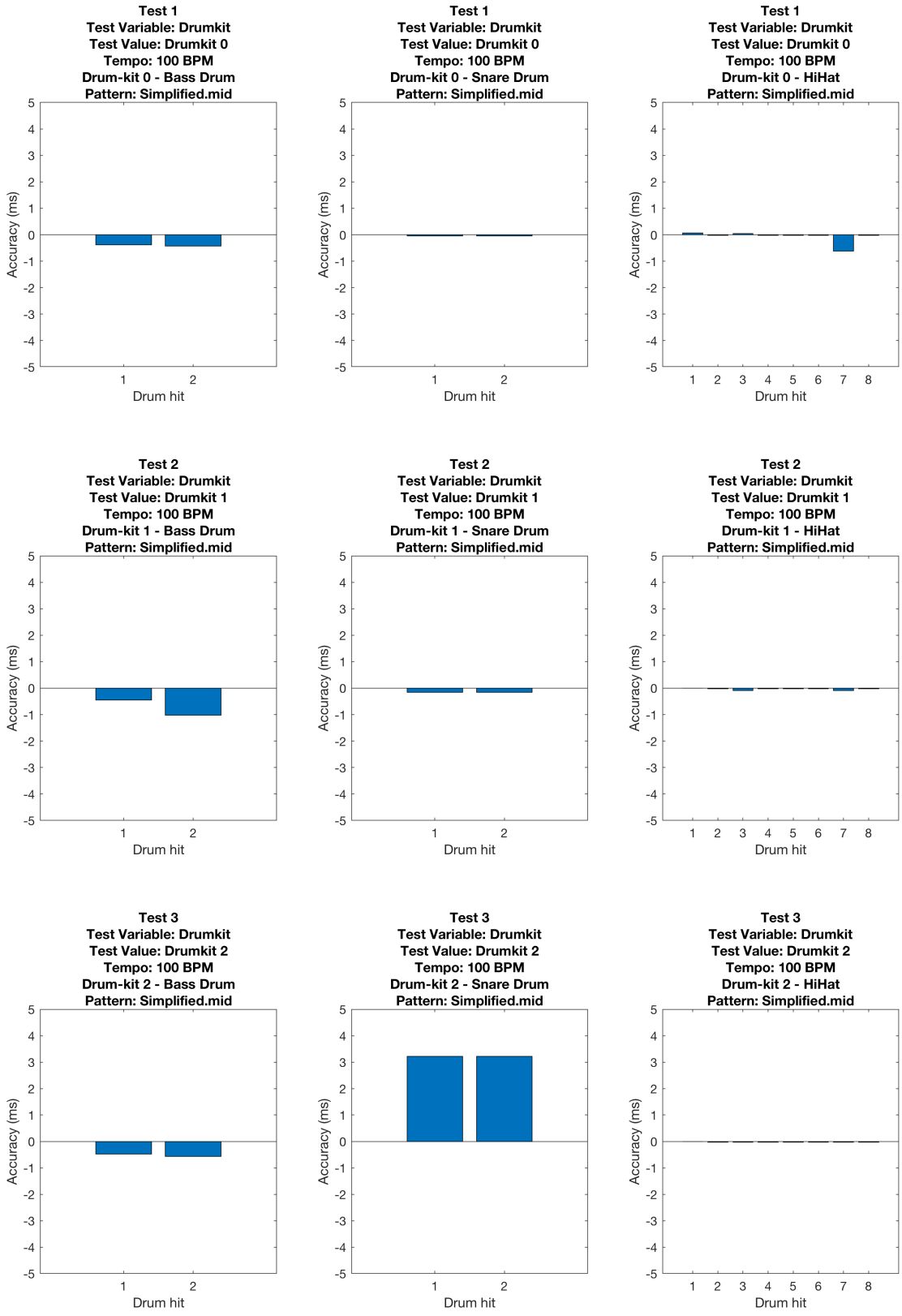


Figure 6.11: Onset detection accuracy with different drum-kits

6.1.2.5 Tempo as Independent Variable

Figure 6.12 shows the onset detection consistency for *testMode=tempo* using the reference drum-kit 1 with the *Simplified* drum pattern, when tempo is increased in steps of 60, 120 and 180 BPM. The nearly identical range values indicate that the system performance remains constant as tempo increases.

Figure 6.13 shows the result of performing the same test with drum-kit 2, which is comprised of samples recorded during the preliminary user test. Again, the range values indicate that the performance is identical regardless of tempo.

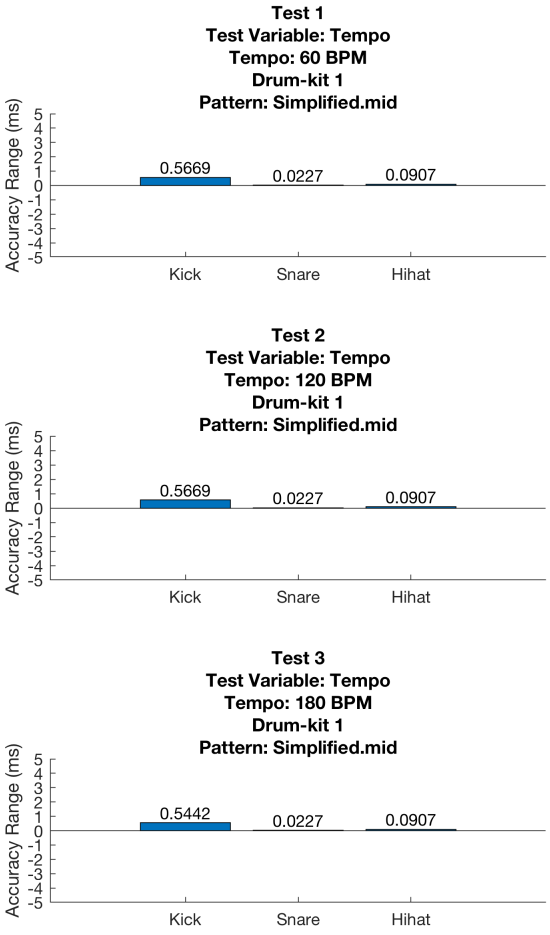


Figure 6.12: Onset detection consistency as range with increasing tempo using drum-kit 1

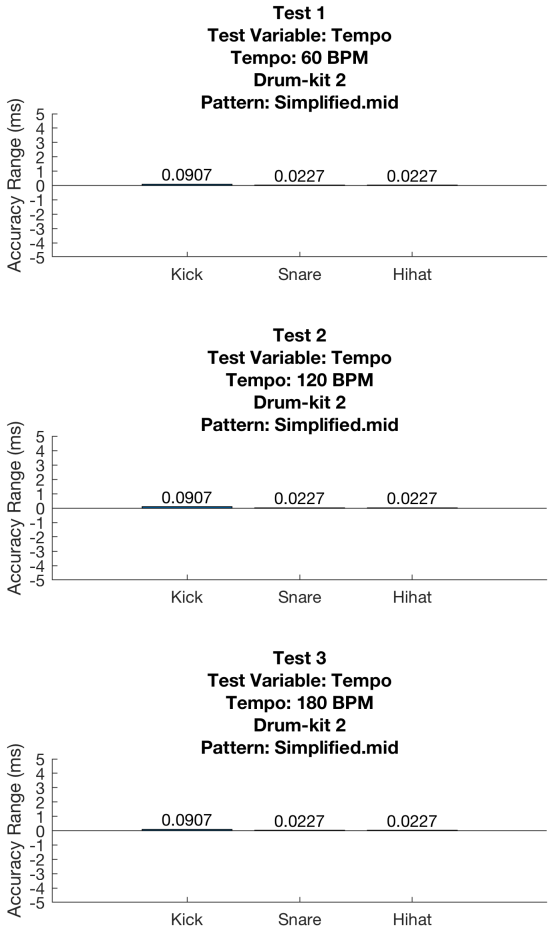


Figure 6.13: Onset detection consistency as range with increasing tempo using drum-kit 2

6.1.2.6 Velocity as Independent Variable

To determine the effect of a performer's drum stroke velocity on the onset detection accuracy, the *testMode=velocity* test modifies the level of the hi-hat in the pattern (see figure 6.15) by changing the velocity value in the MIDI data to 1/3, 2/3 and full velocity over the course of three test iterations. The results in figures 6.14 and 6.16 show that the detected values are constant, which indicates that the system is robust to changes in velocity.

Figure 6.14: Onset detection consistency as range with varying hi-hat velocity

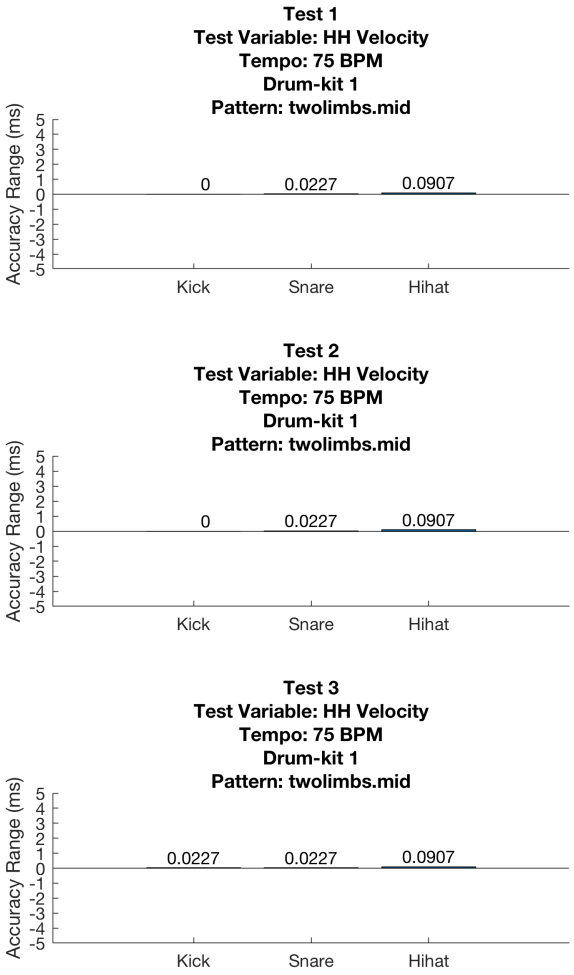
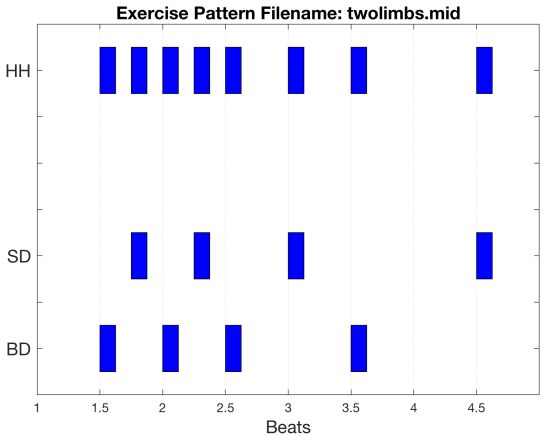


Figure 6.15: Drum pattern used for velocity test



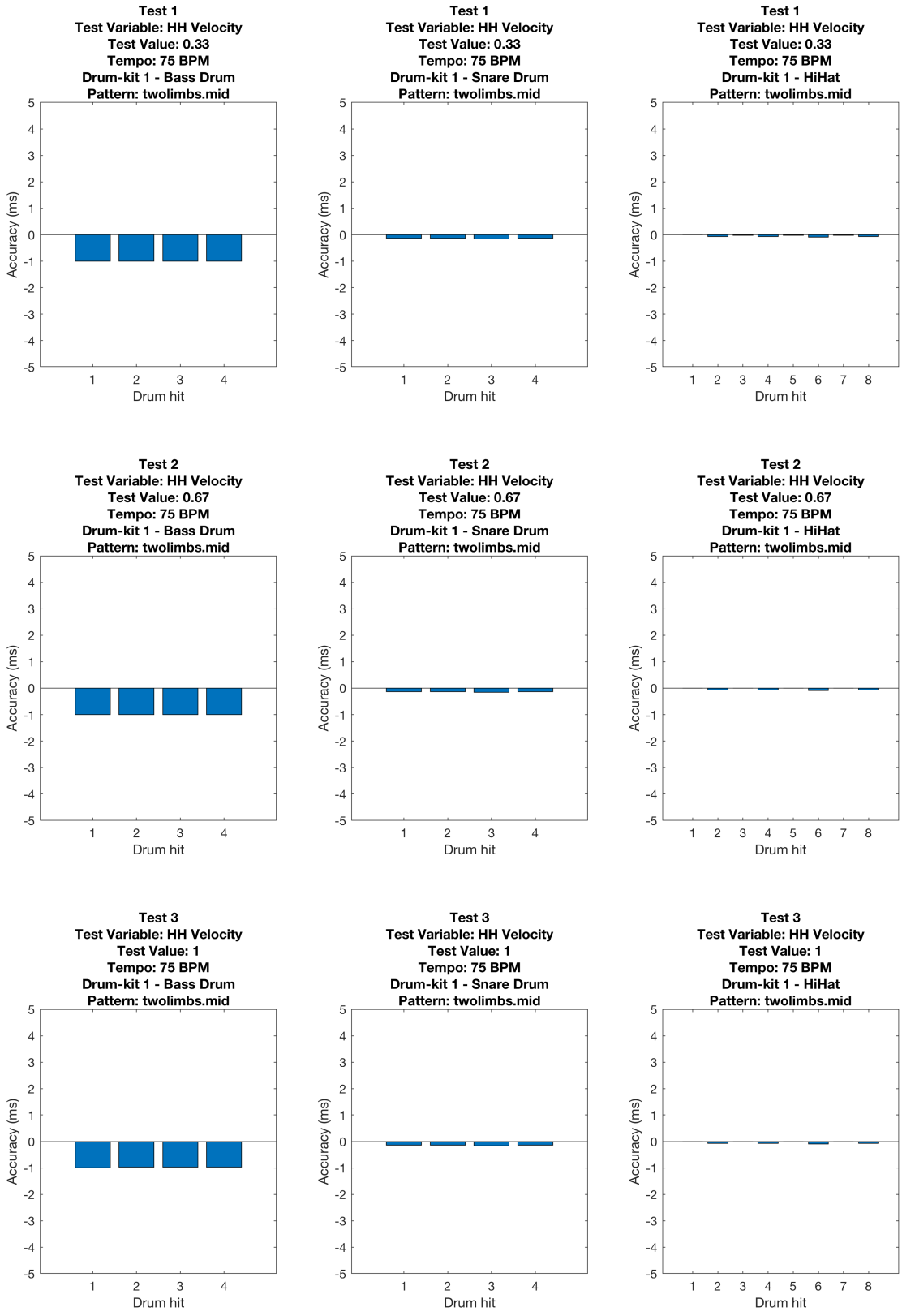


Figure 6.16: Onset detection accuracy with varying hi-hat velocity

6.1.2.7 Benchmarking

Basic benchmarking of the system processing performance was implemented by adding the *tic* and *toc* functions, which measure the time required for operations to complete, to relevant locations in the code. A more comprehensive performance profile was also genereated with the *Profiler*[1] tool running the *sysEval.m* script with a single iteration of the *testMode = 'inter-limb'* test. A condensed list of the most time-consuming operations is available in appendix O.

On a 2014-model MacBook Pro 15" with a 2.5 GHz intel Core i7 processor and 16 GB memory, the processing time for the setup procedure using three components (R=3) with 8 template frames (T=8) and a convergence limit of 1000 iterations took approximately 7.7 seconds.

Processing two bars of audio at 100BPM equal to 211,680 samples at 44.1kHz using the same number of components and template frames with a convergence limit of 100 iterations and plotting the timing feedback took approximately 7.4 seconds. Subsequent runs - also with different data - completed in approximately 2.4 seconds, which is below the limit specified in section 4. The reduction in processing time is likely due to optimizations occurring in MATLAB's internal pipeline.

6.1.3 IDMT-Drums Dataset Test

For comparison with other ADT methods, the prototype was tested with the IDMT-Drums dataset [45]. The IDMT-Drums dataset[2] was chosen, because it exclusively uses bass drum, snare drum and hi-hat and thus enables direct comparison with a number of ADT methods.

Training data is provided for each mixture as separate audio recordings of each drum-kit component in isolation containing multiple hits on each drum-kit component. The mixtures are divided into 14 acoustic drum-kit performances (RealDrum), 60 sample library performances (WaveDrum) and 10 drum synthesizer performances (TechnoDrum). In addition to the audio data, annotations of the onset locations are provided as *SVL* and *XML* data.

Mir_Eval [41] was used to calculate the F-measure with a window size of 50ms, which is identical to the method used in the OaF-Drums paper [37] and in [27].

6.1.3.1 Method

In order to set up the NMFD dictionary, the training data for each drum-kit component belonging to the selected mixture was segmented into individual events using *mirSegment* from the MIR Toolbox [47] and the second event of each drum-kit component was used for training, since the prototype only uses one NMFD template per drum. The onset-to-peak lag was

[1] `https://www.mathworks.com/help/matlab/matlab_prog/profiling-for-improving-performance.html`
[2] `https://www.idmt.fraunhofer.de/en/business_units/m2d/smt/drums.html`

also automatically detected in the training samples for use in the onset detection process as described in section 5.2.4.

Next, source separation and onset detection was performed on the mixtures and the actual onset locations were extrapolated using the previosly detected onset-to-peak lag. Finally, the detected onsets and the reference onsets for each drum-kit component were exported to text files in a format suitable for *Mir_Eval*. Optionally, the detected onset can be plotted for visual comparison with the reference onsets.

This procedure was repeated for all the mixtures in the dataset.

The source separation and onset detection functions used in the IDMT-Drums test are identical to those used in the user evaluation and the other system tests, but for clarity a separate script was created for the IDMT-Drums dataset test, which is available in appendix L. Some functions were duplicated to separate *.m* files and edited for readability by removing code, which was not utilised in the test.

A Python [51] script was created (see appendix M), which imports the onset data text files generated by the MATLAB script and calculates F-measure, precision and recall of each drum-kit component for all the tests in the dataset using the *mir_eval.onset* function. The F-measure results are written to a *CSV* file for subsequent analysis in Excel.

6.1.3.2 Results

The averaged results of all tests for the individual drum-kit components are shown in table 6.1 and indicate that the prototype performs equally well for each component with an increase in accuracy for the snare drum. Although the hi-hat is usually easier to classify compared to the bass drum and snare drum, since there is less overlap occurring in the frequency spectrum and it typically has a distinct peak shape as seen in the scalogram in figure C.4, there are also a larger of number of events occurring for the hi-hat in a normal drum pattern and as a consequence a greater possibility of erroneous classification.

As indicated by the results in table 6.2, the prototype compares favourably to the other methods. In fairness, the prototype is trained on drum samples used in the mixtures and therefore has a slight advantage. However, only one drum sample per component, out of the approximately five samples available per drum, was used as training data. The templates were also fixed, which disables the adaptive update in the NMFD algorithm and makes it less flexible with regards to variations in the components.

Figures 6.17 and 6.18 shows the change in F-measure as a function of *Mir_Eval* window size ranging from the standard 50ms to 10ms. Results for *OaF-Drums* and other methods are unfortunately only available with window sizes of 50ms.

The performance with a window size of 10ms does not quite align with the requirements specified in section 4. The reason for the discrepancy between the other system tests and the IDMT-Drums dataset test could be caused by the patterns in the IDMT-Drums dataset not being optimized for the use case of this prototype, i.e. no overlaps between snare and bass drum events or the frequency content of the samples might be too similar in some cases, causing the source separation quality to decrease. It is also possible that certain training samples cause issues in the automatic onset-to-peak lag calculation. Also, the score-informed option in the NMFD algorithm was not activated for the IDMT-Drums dataset tests, since it was not used in any of the other models in the comparison.

Component	F-measure
BD	90.34
SD	94.77
HH	89.31

Table 6.1: Prototype F-measure per component using the IDMT-Drums dataset with a standard *Mir_Eval* window size of 50 ms.

Model	Training Dataset(s)	F-measure
Prototype	IMDT-Drums	91.47
OaF-Drums	E-GMD	85.27
DT-Ensemble	TMIDT(-BAL),MDB.ENST,RBMA	91.49
ADTLib	ENST-3	83.12

Table 6.2: Prototype compared to other methods using the IDMT-Drums dataset with a standard *Mir_Eval* window size of 50ms. Adapted from [37]

.

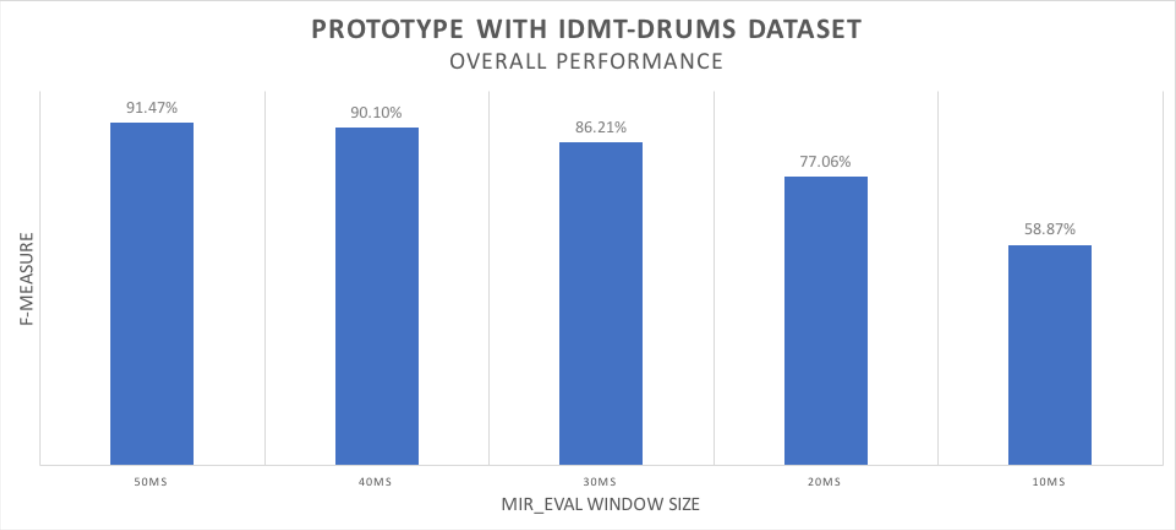


Figure 6.17: Overall F-measure for decreasing *Mir_Eval* window sizes

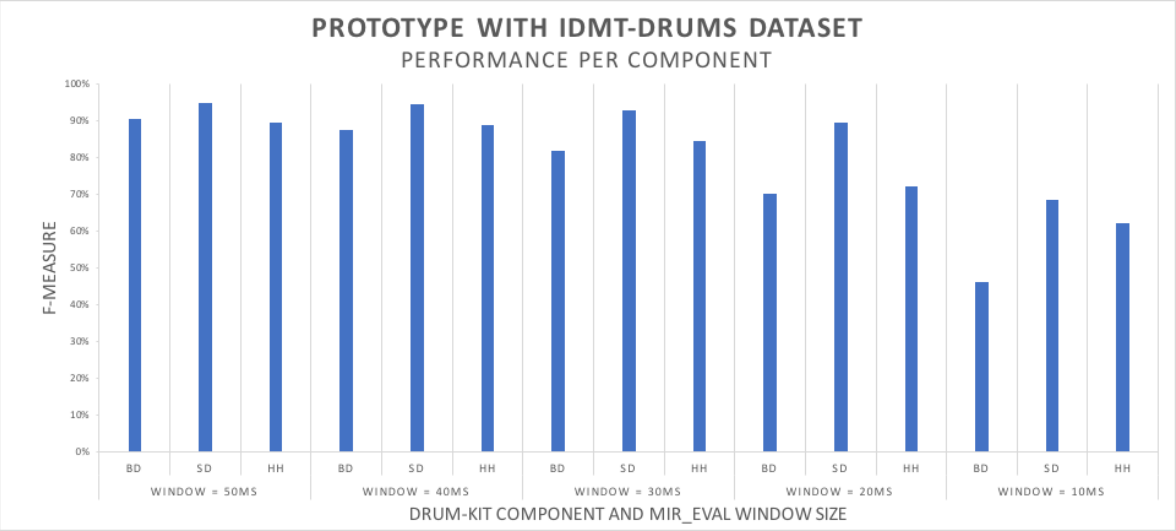


Figure 6.18: F-measure per drum-kit component for decreasing *Mir_Eval* window sizes

6.2 User Evaluation

The goal of the user evaluation is to gather data from a total of ten performances per participant, where the first five are without timing feedback and the following five are with feedback. The tests have been designed to focus on the timing aspect of the user's performance by using a simple exercise pattern consisting only of kick drum, snare drum and hi-hat events.

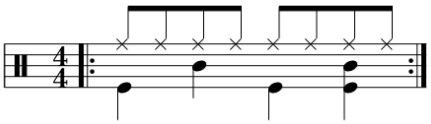


Figure 6.19: Original exercise pattern

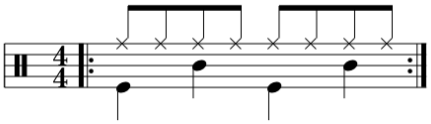


Figure 6.20: Simplified exercise pattern

6.2.1 Preliminary Method

The list below outlines the steps of the preliminary user evaluation process.

1. The participant is presented to a one-bar drum pattern shown as a piano-roll (see figure 6.1) of kick, snare and hi-hat events
2. The participant is asked to press a key to hear a audio preview of the drum pattern
3. The participant is again asked to press a key, which starts the exercise. Following a one bar pre-count, the participant plays two bars of the drum pattern along to a metronome audio pulse played back through headphones
4. The audio and analysis data is stored automatically after the performance as *.wav* and *.mat* files, respectively, but feedback is not provided to the participant about the timing performance
5. Finally, the participant is asked to press a key to restart the test and the process is repeated for a total of five performances
6. The participant is informed by the operator to repeat steps 1-5, but this time timing feedback is provided as shown in figure 6.21

Due to the COVID-19 lockdown in place during the finalization of this project, access to participants was limited, but to progress the thesis a preliminary user evaluation was performed with two participants between the age of 20 and 36, who are musicians, but not drummers. The author also performed the test to verify correct operation of the system, but the results were not used in the evaluation.

The template recording process was performed by the author on location in advance on a three-piece drum-kit shown in figure 6.22 to more efficiently use the limited time available for

testing. This could cause a slight inaccuracy in the detection process due to variations in the playing style of participants.

After the test, the participants were presented with a number of informal questions, which served as the basis for the formal questionnaire presented in appendix A.

Based on the insights gained from the preliminary evaluation, the evaluation method was revised as described in section 6.2.3.

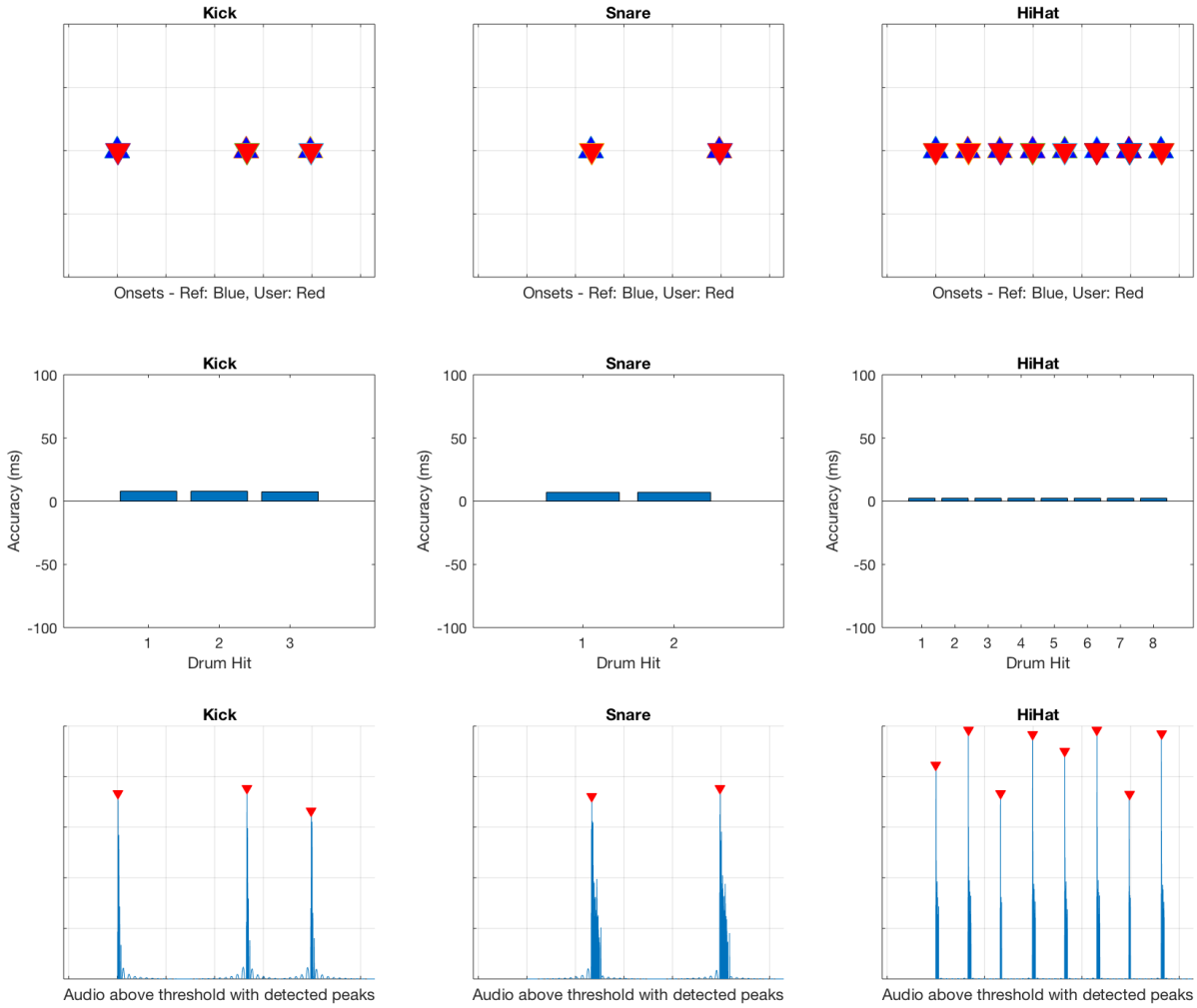


Figure 6.21: Example of timing feedback presented to the user. Red triangles in first row indicate the performer's onsets, while blue triangles indicate reference or ground truth onsets as per the exercise pattern. The second row shows the accuracy of each hit chronologically, where anticipated onsets are negative and lagged onset are positive. The third row shows the detected peaks overlaid on the audio waveform

6.2.2 Preliminary Results

Overall, the participants stated that the tasks were clear. However, it should be noted that since the participants' main instrument was not drums, the users were guided by the operator to ensure that they understood the notation of the drum pattern and the timing feedback. The users responded that the feedback given by the system was useful as a guide to better understand their timing issues and provided insight into how to remedy them.

Figure 6.22: Drum-kit and computer used for user evaluation

During the evaluation process, it became clear that non-drummers might not be familiar with basic drumming techniques and therefore the template dictionary generated by the author in the setup process might not correspond with the sounds generated during a participant's performance.

The built-in microphone in a 2014-model Apple Macbook Pro 15" was used for the system evaluation and a set of Audio-Technica ATH-M50x[3] sealed headphones were connected directly to the computer's headphone output to play back the metronome audio pulse for the participants to synchronize to. The gain of the built-in microphone can be adjusted via the audio settings in the operating system, but unfortunately the minimum gain setting was still too sensitive for the sound pressure levels generated by an acoustic drum-kit, unless played very softly. This may have affected the source separation process, since clipping the audio signal will distort the energy distribution in the spectrograms.

The pattern used for the preliminary user evaluation was a standard rock rhythm as seen in figure 6.19. However, while the inclusion of an additional bass drum event together with the snare drum event on the fourth beat, is useful for evaluating system performance with overlapping events in a real-life setting, the pattern's complexity and requirements for inter-limb coordination had a negative effect on non-drummers' timing performance.

Plots of the onset accuracy and distribution for the performances by participants LK and M are available in figures E.1 to E.8 in the appendix.

For participant LK, there is an indication of lowered asynchrony in the third test attempt without timing feedback, after which further improvement is less evident in the tests with

[3] `https://www.audio-technica.com/cms/headphones/99aff89488ddd6b1/index.html`

timing feedback.

Participant M's timing performance is similar in all cases, but a slight improvement can be detected in the third performance without timing feedback and thereafter the timing performance remains similar for the tests with timing feedback. It was noted that compared to LK, participant M more frequently omitted playing required events.

6.2.3 Revised Method

- The exercise drum pattern has been simplified as seen in figure 6.20, so that a maximum of two limbs are used simultaneously. This decision was made to reduce the effect of perceptual or motoric limitations on the participant's timing accuracy. The pattern is identical to the one used in [12].
- The drum-kit component corresponding to each line in the piano-roll is now more clearly labeled.
- The participant now has the option to hear the drum pattern played back multiple times before starting the test.
- A standard Apple iPhone headset[4] connected to the computer is used both as an external microphone and for audio and metronome playback, since it provides a wider gain adjustment range for the microphone to better match the acoustic output of the drum-kit.
- The timing feedback is plotted in a format that fits better on a single laptop screen, while leaving room so the text prompts are still visible.
- The waveform subplot has been removed from the user evaluation timing feedback, since the added complexity was not beneficial for the user experience.
- The bar plot showing accuracy now has the text *Early*, *Perfect* and *Late* on the far right aligned to the Y axis to make it more clear to the user what the plot indicates.
- The tests with and without feedback are now performed continuously without requiring operator interaction.
- The participant is asked to fill out the questionnaire in appendix A after completing the test to gather qualitative data.

  An opportunity presented itself to test the revised evaluation design with a professional musician, participant MP, who reported being a hobbyist level drummer. The participant does not usually practice with a metronome or similar timing tools, but regularly composes music in a digital audio workstation to a specific musical grid.

  The revised evaluation was performed in the same room as the previous evaluations and with the same drum-kit, but located slightly differently. The microphone and headphones of an Apple iPhone headset were used instead of the built-in microphone in the Apple MacBook Pro and the Audio-Technica ATH-M50X headphones. This provided a more equidistant placement of the microphone relative to each drum and also enabled the microphone gain to be set sufficiently low to avoid distortion of the audio signal.

[4] `https://www.apple.com/shop/product/MNHF2AM/A/earpods-with-35-mm-headphone-plug`

6.2.4 Revised Result

On the questionnaire, the participant reported that synchronising to the metronome was easy. But in a subsequent informal discussion, the participant mentioned that the metronome audio pulse could be difficult to hear if playing loudly on the drum-kit, due to the open design of the headphones implemented in the Apple headset, and felt that this affected the performance negatively.

The participant was asked to perform the setup tasks prior to the exercises to ensure that the dictionary correlated to the participant's playing style. It was noted that care had to be taken during setup to avoid spurious signals caused by movement of the microphone. The addition of an option to restart the setup process was therefore suggested by the participant.

Plots of the onset accuracy and distribution without and with timing feedback are available in figures E.13 to E.16. Note that only three attempts were made for each modality using the simplified patterns.

The plots show that the participant is able to a achieve low asynchrony from the first attempt for both test modalities. This indicates that participant MP is more familiar with playing drums to a metronome than participants LK and M.

For completeness, the participant was also asked to perform the evaluation tasks using the original exercise pattern, but as expected the performance suffered due to the increased complexity of the pattern as seen in figures E.9 to E.10. However, as indicated by the results for the attempts with timing feedback shown in figures E.11 to E.12, MP is able to recover and achieve similar timing accuracy as seen in the performances with the simplified pattern.

6.3 Descriptive Statistics

The purpose of performing a statistical analysis of the test results is to establish a simple metric that can be used to compare different performances by the same performer and between different performers. The metric can also function as a simplified score that can be presented to the performer as an overall indicator of their performance as it changes over time in addition to the more nuanced timing feedback currently provided.

As seen in [12, 16, 52], a common metric for measuring asynchrony in drumming performances is the mean and standard deviation of the performed onsets relative to a timing reference, such as a metronome. The benefit of this method is that it relates directly to an intuitive understanding of the problem.

Intuitively, the mean should tend towards zero if there is an improvement in the performers ability to synchronize with the metronome guide and the standard deviation should become

smaller if the performer's timing consistency is improving. Comparing the performances with and without feedback should therefore reveal whether a performer is making progress.

Combining the mean and standard deviation into one metric was considered. However, this requires a definition of what constitutes a performance improvement in a combined metric. A mean close to zero with a large standard deviation is not necessarily better than a mean far from zero with a small standard deviation. In the intended use case, the ideal is that both mean and standard deviation tend towards zero as the user progresses.

One option could be to simply multiply the mean with the standard deviation and use the result as a performance score, since it equalizes the importance of either metric on the final score. Due to the small number of participants in the user evaluation, this was not explored further, but might be worth revisiting in the future.

To perform the analysis, the MATLAB script, *analyzeUserData.m* available in appendix N, was created, which loads the onset data stored to files after each performance in the user evaluation process.

The script handles post-processing of the data by calling the *matchOnsets* function with the post-processing options enabled to remove duplicate onsets and outliers and to interpolate missing onset.

The script also calls the *userdataStats* function, which performs the calculations and plotting for the statistical analysis described below.

To calculate the mean and standard deviation, the last three performances without and with timing feedback are pooled into two vectors - one vector for each test modality. Prior to using MATLAB's *mean* and *std* functions to perform the calculations, the asynchronies are converted to absolute values using the *abs* function.

The results for participant LK are presented in figure 6.23 as a bar graph with the two test modalities shown side by side indicated as *without FB* and *with FB* (FB = feedback). The results show that a marked decrease in both onset mean and standard deviation occurs for the tests with timing feedback compared to the tests without feedback.

Similarly, the results for participant M are shown in figure 6.24 and also show a decrease in onset mean and standard deviation, although not as substantial.

In figure 6.26 the mean and standard deviation values for MP performing the simplified pattern show an increase in both metrics for the performances with timing feedback. However, it could be argued that when mean asynchrony and standard deviation variations are below 15ms, the difference can just as likely be attributed to chance as much as differing test modalities.

The statistics also provide insights into the evaluation method itself. For instance, the fact that an improvement can be seen for all participants when performing the original drum pattern with timing feedback - also participant MP, who did not show a marked difference in asynchrony values between modalities when performing the simplified drum pattern - indicates that the improvements might be more related to increasing familiarity with the pattern over the evaluation period, since it requires inter-limb coordination proficiency not common for non-drummers.

Looking closer at participant MP's performance in figure 6.25 shows that the largest improvement is seen in the bass drum mean value, which correlates with added complexity of an additional event for the bass drum on the fourth beat.

Overall, the statistical analysis results are in line with the manual inspection of the onset accuracies performed in section 6.2.2 and 6.2.4, which indicates that the chosen metrics are a good fit for the use case.

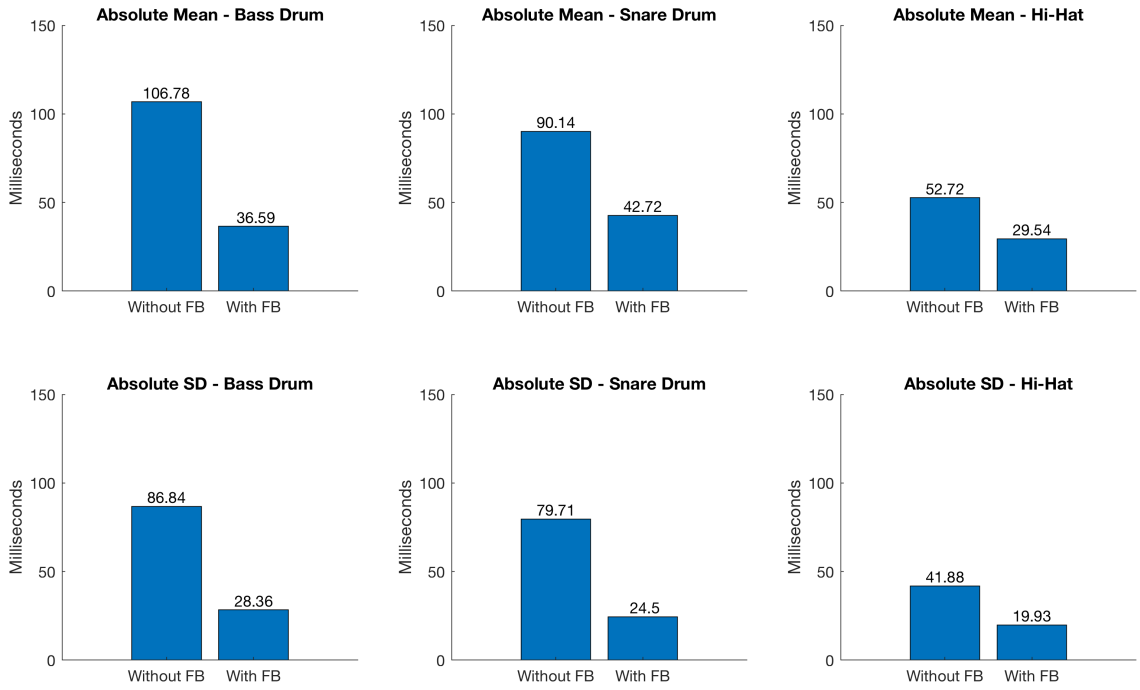


Figure 6.23: Asyncrony statistics for participant LK performing the original pattern with and without feedback

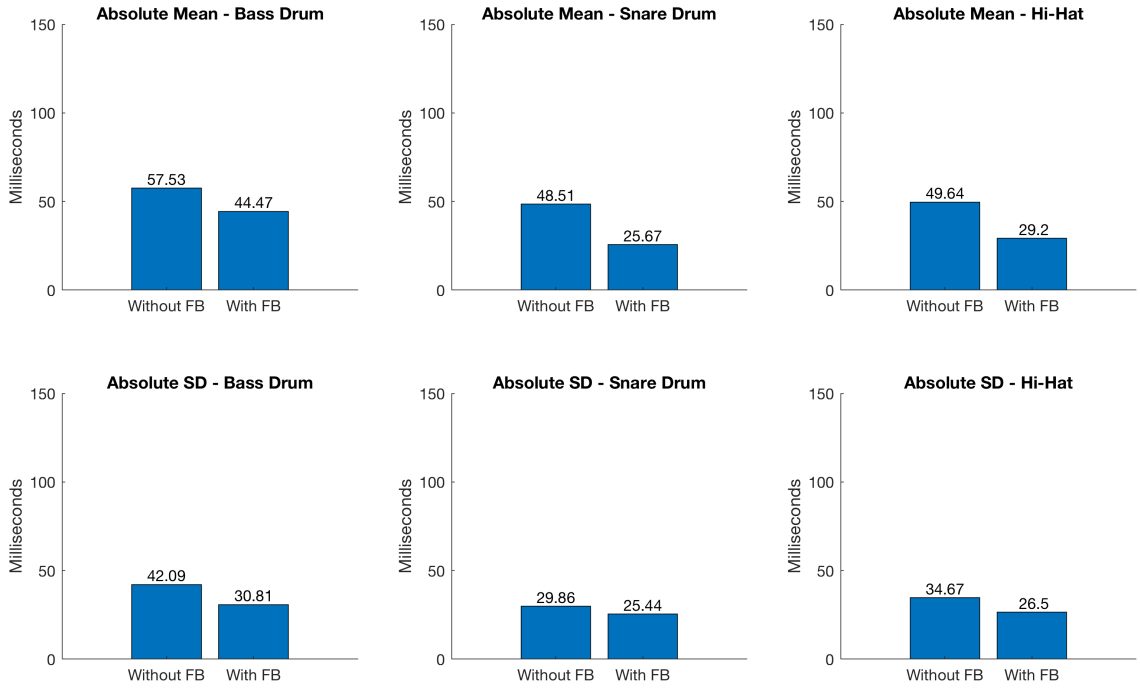


Figure 6.24: Asyncrony statistics for participant M performing the original pattern with and without feedback

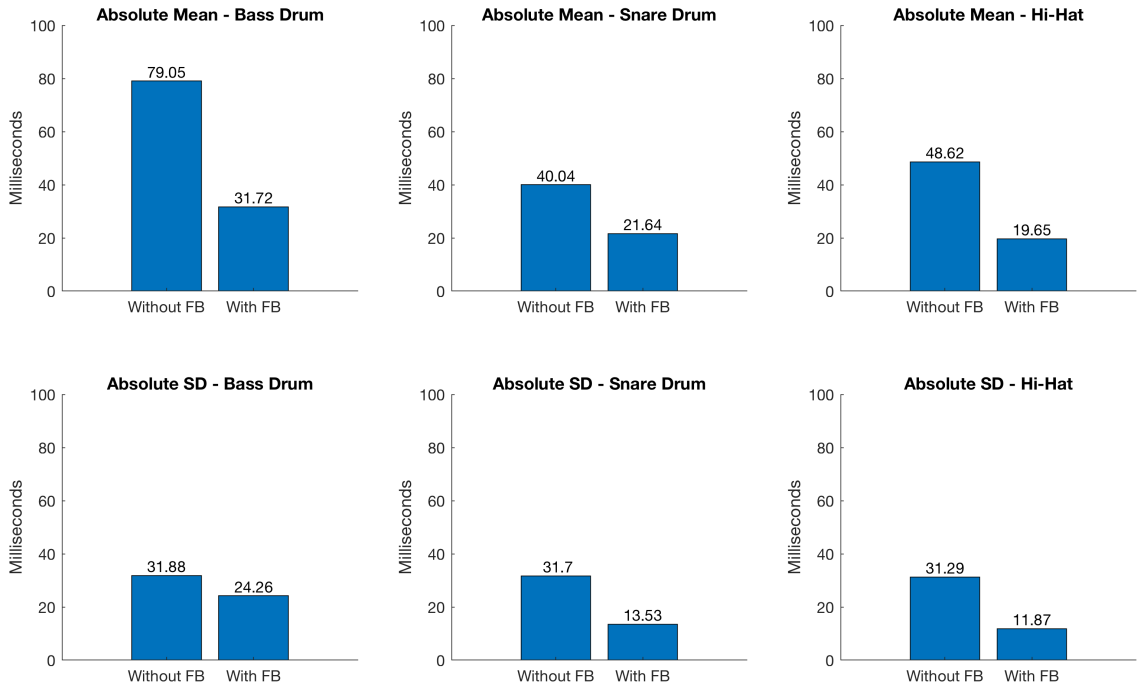


Figure 6.25: Asyncrony statistics for participant MP performing the original pattern with and without feedback

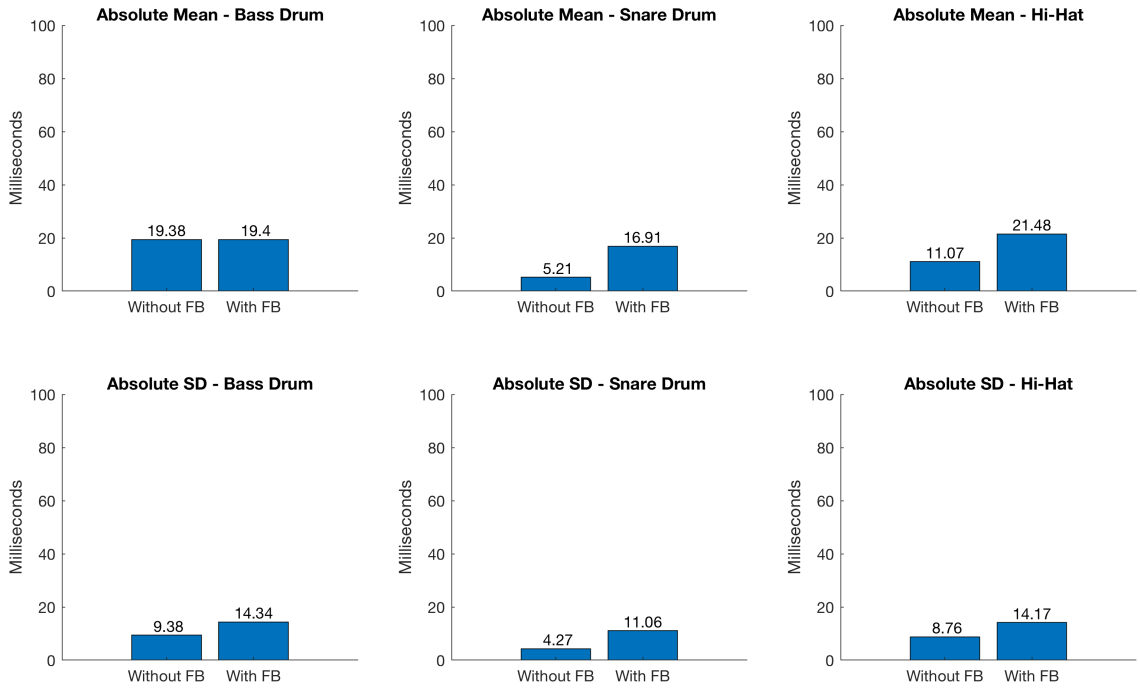


Figure 6.26: Asyncrony statistics for participant MP performing the simplified pattern with and without feedback

6.4 Inferential Statistics

While it is possible to use derived statistics to manually analyse test results for a small number of participants, a method for determining whether changes seen in the mean and standard deviation values as a result of different modalities are statistically significant is essential for larger scale testing.

The *Repeated Measures Analysis-Of-Variance* (RM-ANOVA) test is often utilized for this purpose and operates on the basis of a null hypothesis and an alternative hypothesis.

In this case, the null hypothesis is that presenting the user with timing feedback after the exercises has no effect on the user's timing accuracy compared to performing the same exercises without feedback. The alternative hypothesis is that timing feedback has an effect on timing accuracy.

In order to verify that the RM-ANOVA test would provide meaningful results with the available data, the RM-ANOVA test was performed on asynchrony data from test participant MP, performing the simplified pattern without receiving timing feedback, as the control group and the ground truth data as the experimental group. In essence, a simulation of the results if a participant's performance reached perfection when receiving timing feedback.

Only data from the last three attempts of a participant's performance was used to reduce the effect of a participant perhaps not fully understanding the task and how to use the timing feedback effectively in the beginning.

RM-ANOVA assumes normally distributed data and therefore the Shapiro-Wilk test was performed on the data using the *swtest.m* function available from [53]. The null hypothesis for the Shapiro-Wilk test is that the data is normal with unspecified mean and variance.

Performing the Shapiro-Wilk test on the control data and the experimental data for all participants showed that the data was normally distributed except for the bass drum and hi-hat data in the performances with timing feedback for participant M at $p=0.011$ and $p=0.0093$, respectively. Also, the hi-hat data for the performances without timing feedback for participant MP with the original drum pattern failed the test at $p=9.247e-06$.

Ordinarily, the Mauchly test would be performed to verify sphericity of the data, but since the user evaluation is a within-subjects design with only two levels, sphericity is inherent and the Mauchly test becomes redundant [54, p. 561].

To avoid issues due to the data containing both negative and positive asynchrony, the data was converted to absolute values using the *abs* function prior to the RM-ANOVA test. A significance level of $p < 0.05$ is used to indicate a statistically significant difference.

Inspecting the result of the RM-ANOVA test in table 6.3 indicates that a statistically significant

difference can be detected in all drum-kit components, which fits with the expected result given the data.

Seeing that RM-ANOVA produces meaningful results on the synthetic data, the user performances were analyzed and are presented in tables 6.4 to 6.7.

The results show that all hi-hat performances show a statistically significant difference, while the results for the bass drum and snare drum are less consistent. As an example, in table 6.5 participant M's results do not show a statistically significant difference for the bass drum and the snare drum is borderline significant, while the hi-hat is significantly different.

MP vs. Ground Truth	**SumSq**	**DF**	**MeanSq**	**F**	**pValue**
Bass Drum					
(Intercept):Asynchrony	3050.82966	1	3050.82966	65.4632218	3.12825E-07
Error(Asynchrony)	792.26324	17	46.60372	1	0.5
Snare Drum					
(Intercept):Asynchrony	162.849006	1	162.849006	17.8947006	0.001411912
Error(Asynchrony)	100.104445	11	9.10040406	1	0.5
Hi-Hat					
(Intercept):Asynchrony	3081.53601	1	3081.53601	84.8177254	4.26044E-12
Error(Asynchrony)	1707.56987	47	36.3312738	1	0.5

Table 6.3

LK	**SumSq**	**DF**	**MeanSq**	**F**	**pValue**
Bass Drum					
(Intercept):Asynchrony	50866.7846	1	50866.7846	10.857985	0.004274657
Error(Asynchrony)	79640.4986	17	4684.73521	1	0.5
Snare Drum					
(Intercept):Asynchrony	26218.0779	1	26218.0779	8.23979572	0.015227775
Error(Asynchrony)	35000.729	11	3181.88446	1	0.5
Hi-Hat					
(Intercept):Asynchrony	13732.0717	1	13732.0717	14.4295731	0.000417145
Error(Asynchrony)	44728.0987	47	951.661674	1	0.5

Table 6.4

M	SumSq	DF	MeanSq	F	pValue
Bass Drum					
(Intercept):Asynchrony	1083.07158	1	1083.07158	0.91381997	0.352500385
Error(Asynchrony)	20148.6261	17	1185.2133	1	0.5
Snare Drum					
(Intercept):Asynchrony	3188.82907	1	3188.82907	5.32766542	0.041431691
Error(Asynchrony)	6583.95696	11	598.541542	1	0.5
Hi-Hat					
(Intercept):Asynchrony	10452.6009	1	10452.6009	21.8489579	2.50692E-05
Error(Asynchrony)	22484.9277	47	478.402718	1	0.5

Table 6.5

MP - Original	SumSq	DF	MeanSq	F	pValue
Bass Drum					
(Intercept):Asynchrony	420.668471	1	420.668471	0.82554607	0.376264898
Error(Asynchrony)	8662.58619	17	509.563894	1	0.5
Snare Drum					
(Intercept):Asynchrony	1651.96002	1	1651.96002	4.04023975	0.069596379
Error(Asynchrony)	4497.64405	11	408.876732	1	0.5
Hi-Hat					
(Intercept):Asynchrony	11157.3714	1	11157.3714	80.7596787	8.95791E-12
Error(Asynchrony)	6493.29548	47	138.155223	1	0.5

Table 6.6

MP - Simplified	SumSq	DF	MeanSq	F	pValue
Bass Drum					
(Intercept):Asynchrony	14.2259541	1	14.2259541	0.11415493	0.73959884
Error(Asynchrony)	2118.53509	17	124.619711	1	0.5
Snare Drum					
(Intercept):Asynchrony	821.966242	1	821.966242	13.9168217	0.003320424
Error(Asynchrony)	649.69063	11	59.0627845	1	0.5
Hi-Hat					
(Intercept):Asynchrony	2534.95213	1	2534.95213	16.3189933	0.000196605
Error(Asynchrony)	7300.86396	47	155.337531	1	0.5

Table 6.7

Chapter 7

Discussion

The system evaluation results show that the system is able to meet the requirements defined in the requirements specification in section 4. The IDMT-Drums dataset test shows that performance equal to the state-of-the-art methods is achieved with the standard 50ms F-measure window size, but reduced performance with smaller window sizes. Inspecting the automatically detected onset-to-peak lag values as well as the source separated streams more closely might reveal where further optimizations can be made to align the IDMT-Drums results with the other systems evaluation results.

One option would be to apply bandpass filtering optimized for each drum-kit component to reduce interaction between components since the peaks of bass drum, snare drum and hi-hat lie in different frequency ranges.

Currently, onset detection is performed on the individual reconstructed audio streams, however as seen in [45], onset detection could just as well be performed on the magnitude spectrograms prior to reconstruction, which would also allow detecting peaks in specific frequency bands without the use of bandpass filtering.

Although the accuracy of the onset detection has been verified with synthetic audio signals, there is still a risk that real-world audio, where the components mix together acoustically, can contain content that degrades the performance of the system. However, the empirically evaluated stability during the user evaluations indicates that system evaluation results are transferable to the real world if the assumptions in section 4 are met.

The reduced number of participants in the user evaluation due to the COVID-19 lockdown renders the evaluation results less conclusive and further testing is therefore required to properly determine the effects of the timing feedback on a performer's timing accuracy.

However, inspecting the results available so far indicates that the improvements seen in the performances with timing feedback are related to an increased familiarity with the task rather than a result of changing the modality.

The evaluation design could be improved by dividing the participants into two groups - an experimental group and a control group. Modifying the independent variable for each group - i.e. whether the participant receives timing feedback - but otherwise performing the exercises identically in both groups, should improve the quality of the evaluation given a sufficiently large number of participants.

As an alternative to separate groups, the order of when feedback is provided could be randomized to rule out the effect of the participant becoming increasingly familiar with the exercise pattern over time. This constant change of modality could however confuse the participant, since a certain amount of trial and error is required to understand how actions - i.e. hitting the drums earlier or later in time - are reflected in the timing feedback in order to utilise the feedback efficiently.

The results indicate that the drum pattern used for the user evaluation can have a significant effect on the timing performance due to perceptual and motoric limitations of the performer. Further evaluations should therefore be performed with the simplified drum pattern to ensure that the results solely reflect sensorimotor synchronisation accuracy.

Allowing the user evaluation participant to practice the pattern without any timing aids for an certain amount of time prior to the actual test, could perhaps also equalize the effect of perceptual and motoric limitations among the participants - especially for non-drummers or beginners.

Including the questionnaire in appendix A in the user evaluation process provides a means to correlate the quantitative data with qualitative data. Especially, investigating if there is a connection between how participants rate their ability to synchronize with the metronome as well as the accuracy of timing feedback and the level of improvement seen in the quantitative data would be interesting to explore further with a larger number of user evaluation participants.

The qualitative aspect of the user evaluation could also be expanded further to help guide future development efforts and feature implementation, as the informal discussions with participants following the evaluation provided important user feedback.

Optimal system performance requires the performer to maintain a consistent playing style, since excessive sonic and dynamic variation might degrade the performance of the source separation and onset detection. However, since this is a learning tool, playing within certain constraints and maintaining consistency could be incorporated into the exercises, however care has to be taken to not make the constraints too strict and cause frustration for beginners. Therefore, it might be useful to adjust settings in the system, such as the temporal resolution of the timing feedback, depending on the skill level of the performer, so that beginners can focus on the broader aspects of their timing and when improved performance is detected, the required level of accuracy can be increased to keep the exercises challenging.

Chapter 8

Conclusion

The goal of this thesis project was to design and implement a prototype tool to help drummers improve their timing accuracy based on research into the human sensory and motor system as well as the state-of-the-art in automatic drum transcription methods.

A number of automated system tests were performed to evaluate the accuracy and performance of the implementation and underlying algorithms. The result indicate that the system meets the specified requirements and achieves state-of-the-art results when compared to other automatic drum transcription methods using the IDMT-Drums dataset.

Limited by the Covid-19 lockdown in place during the finalization of the project, a user evaluation was conducted with three participants, who all were musicians, but non-drummers.

The results were analysed with descriptive statistics methods and repeated measures ANOVA, which established that a statistically significant difference could be detected between the test modalities.

However, the small number of participants and the lack of a control group renders the results inconclusive as to whether the difference is a result of the timing feedback or just due to extended exposure to the same drum exercise pattern.

Chapter 9

Future Work

For future implementations, porting the prototype to a custom embedded device could be considered, but development costs and the resulting end user cost would likely be prohibitive for the market. Ultimately, a mobile device implementation of the prototype would be advantageous, since mobile devices already contain the required audio and graphics peripherals. An attempt was made to create an iPhone app using MATLAB Coder[1], but to create a C/C++ code version of the prototype would require replacing the third-party toolboxes used with custom functions that comply with the MATLAB Coder requirements, which was not feasible within the time-frame of the thesis.

Additionally, it would beneficial to port the prototype to a real-time audio-focused framework, such as JUCE[2], to gain more flexibility in further development of the system and to enable custom user interface designs.

Utilizing a stereophonic mixture and thus enabling the use of separate spectrograms for the left and right audio signals, the NMFD algorithm could be augmented to provide better separation of similar sounding drums, since the drum-kit components are usually located in distinct locations within the pickup pattern of a stereo microphone. However, since stereophonic recording has only become available in more recent smartphones, this would limit the user base.

While the exercise patterns should be optimized for the capabilities of the algorithms - i.e. a bass drum event should not be placed at the same time as a snare drum event - further optimization of the system could possibly improve detection of similar sounding drums. To this end, exploring the use of *Continous Wavelet Transform* (CWT) to generate the input to the NMFD algorithm instead of STFT magnitudes would be interesting, since CWT could potentially provide increased time-frequency resolution in the salient areas compared to STFT.

[1] https://www.mathworks.com/products/matlab-coder.html
[2] https://juce.com

Bibliography

[1] Steven Feld. "Aesthetics as Iconicity of Style, or 'Lift-up-over Sounding': Getting into the Kaluli Groove". In: *Yearbook for Traditional Music* 20.1 (1988). ID: proquest1306837558, p. 74. DOI: 10.2307/768167.

[2] Alex Hofmann et al. "The Tight-interlocked Rhythm Section: Production and Perception of Synchronisation in Jazz Trio Performance". In: *Journal of new music research* 46.4 (2017). ID: proquest1977187247, pp. 329–341. DOI: 10.1080/09298215.2017.1355394.

[3] Jan Frühauf, Reinhard Kopiez, and Friedrich Platz. "Music on the timing grid: The influence of microtiming on the perceived groove quality of a simple drum pattern performance". In: *Musicae Scientiae* 17 (2013), pp. 246–260. DOI: 10.1177/1029864913486793.

[4] Mark Levine. *The Jazz Theory Book*. Petaluma, Calif.: Sher Music, 1995.

[5] Vijay Iyer. "Embodied Mind, Situated Cognition, and Expressive Microtiming in African-American Music". In: *Music Perception: An Interdisciplinary Journal* 19.3 (2002). ID: jstor_csp10.1525/mp.2002.19.3.387, pp. 387–414. DOI: 10.1525/mp.2002.19.3.387.

[6] Anne Danielsen. *Musical rhythm in the age of digital reproduction*. Includes bibliographical references and index.; ID: alma9920760179605762. Burlington, VT: Ashgate, 2010. ISBN: 1-317-09138-8.

[7] Dave Atkinson. *Quick drum lesson: how to improve timing and note spacing*. Apr. 2015. URL: https://www.youtube.com/watch?v=gqUITaQ3M8g. Retrieved on Mar. 05, 2020.

[8] Petr Janata and T. Grafton Scott. "Swinging in the brain: shared neural substrates for behaviors related to sequencing and music". In: *Nature neuroscience* 6.7 (2003). ID: nature_a10.1038/nn1081, p. 682. DOI: 10.1038/nn1081.

[9] Lorenz Kilchenmann, Olivier Senn, and Lorenz Kilchenmann. "Microtiming in Swing and Funk affects the body movement behavior of music expert listeners". In: *Frontiers in psychology* 6 (2015). ID: proquest1710653663, p. 1232. DOI: 10.3389/fpsyg.2015.01232.

[10] László Stachó et al. *Perception of emotional content in musical performances by 3–7-year-old children*. ID: sage_s10_1177_1029864913497617. 2013. DOI: 10.1177/1029864913497617.

[11] Olivier Senn et al. "The Effect of Expert Performance Microtiming on Listeners' Experience of Groove in Swing or Funk Music". In: *Frontiers in psychology* 7 (2016), p. 1487.

[12] Shinya Fujii et al. "Synchronization Error of Drum Kit Playing with a Metronome at Different Tempi by Professional Drummers". In: *Music Perception: An Interdisciplinary Journal* 28.5 (2011), pp. 491–503. DOI: 10.1525/mp.2011.28.5.491. URL: https://mp.ucpress.edu/content/28/5/491.

[13] Michael H. Thaut. *Rhythm, music, and the brain scientific foundations and clinical applications*. 1st ed. Includes bibliographical references (p. 219-234) and index.; ID: alma9920810118105762. New York: Routledge, 2008. ISBN: 1-283-96666-2.

[14] Bruno H. Repp and Rebecca Doggett. "Tapping to a Very Slow Beat: A Comparison of Musicians and Nonmusicians". In: *Music Perception: An Interdisciplinary Journal* 24.4 (2007). ID: jstor_csp10.1525/mp.2007.24.4.367, pp. 367–376. DOI: 10.1525/mp.2007.24.4.367.

[15] Brendan Bache. *Guide to Drum Stick Grips*. Feb. 2017. URL: https://www.libertyparkmusic.com/drum-sticks-grip-guide/. Retrieved on May 05, 2020.

[16] Shinya Fujii et al. "Tapping performance and underlying wrist muscle activity of non-drummers, drummers, and the world's fastest drummer". In: *Neuroscience letters* 459.2 (2009), p. 69. DOI: https://doi.org/10.1016/j.neulet.2009.04.055". URL: http://www.sciencedirect.com/science/article/pii/S0304394009005795.

[17] Jared Falk. *Speed Drumming With Finger Control Technique*. Oct. 2014. URL: https://www.drumeo.com/beat/finger-control-technique/. Retrieved on May 11, 2020.

[18] Thomas Lang. *Creative Coordination & Advanced Foot Technique*. 1st ed. Hudson Music, 2007.

[19] Thomas Lang. *Thomas Lang: 5 Ideas For Faster Feet*. Jan. 2013. URL: https://drummagazine.com/thomas-lang-5-ideas-for-faster-feet/. Retrieved on May 11, 2020.

[20] Ramon E. Meyer. "The Functions of the Snare Drum Rudiments". In: *Music Educators Journal* 48.3 (1962). ID: sage_s10_2307_3389639, p. 80. DOI: 10.2307/3389639.

[21] Austin Burcham. *Using Ghost Clicks To Improve Your Time*. Apr. 2015. URL: https://www.youtube.com/watch?v=AUdcgKQNXVQ. Retrieved on May 11, 2020.

[22] Sofia Dahl. "Movements, Timing, and Precision of Drummers". English. In: (2018), pp. 1839–1857. DOI: 10.1007/978-3-319-14418-4_110. URL: https://sfx.aub.aau.dk/sfxaub?sid=pureportal&doi=10.1007/978-3-319-14418-4\%5f110.

[23] Rina A. Doherty and Paul Sorenson. *Keeping Users in the Flow: Mapping System Responsiveness with User Experience*. 2015. DOI: https://doi.org/10.1016/j.promfg.2015.07.436. URL: http://www.sciencedirect.com/science/article/pii/S2351978915004370.

[24] Agnieszka Wykowska and Valtteri Arstila. *On the flexibility of human temporal resolution*. 2014, pp. 431–452. ISBN: 9780262019941.

[25] A. D. Brown, G. C. Stecker, and D. J. Tollin. "The precedence effect in sound localization". eng. In: *Journal of the Association for Research in Otolaryngology : JARO* 16.1 (Feb. 2015), pp. 1–28. DOI: 10.1007/s10162-014-0496-2[doi].

[26] Walter Schloss. *ON THE AUTOMATIC TRANSCRIPTION OF PERCUSSIVE MUSIC–FROM ACOUSTIC SIGNAL TO HIGH-LEVEL ANALYSIS*. 1985.

[27] Chih-Wei Wu et al. "A Review of Automatic Drum Transcription". In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 26.9 (2018), pp. 1457–1483. DOI: 10.1109/TASLP.2018.2830113.

[28] Nikolaos Stefanakis, Yannis Mastorakis, and Athanasios Mouchtari. "Instantaneous Detection And Classification Of Impact Sound: Turning Simple Objects Into Powerful Musical Control Interfaces". In: (2014). DOI: 10.5281/zenodo.850806.

[29] *Pure Data — Pd Community Site*. URL: https://puredata.info.

[30] Miller S. Puckette, Theodore Apel, and David Zicarelli. "Real-time audio analysis tools for Pd and MSP". In: *Proceedings of the 1998 International Computer Music Conference, ICMC 1998, Ann Arbor, Michigan, USA, October 1-6, 1998*. Michigan Publishing, 1998. URL: http://hdl.handle.net/2027/spo.bbp2372.1998.315.

[31] Daniel D. Lee and H. Sebastian Seung. "Algorithms for Non-Negative Matrix Factorization". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Cambridge, MA, USA: MIT Press, 2000, 535–541.

[32] Paris Smaragdis. "Non-negative Matrix Factor Deconvolution; Extraction of Multiple Sound Sources from Monophonic Inputs". In: *Independent Component Analysis and Blind Signal Separation*. Ed. by Carlos G. Puntonet and Alberto Prieto. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 494–499. ISBN: 9783-540301103.

[33] Patricio López-Serrano et al. "NMF Toolbox: Music Processing Applications of Nonnegative Matrix Factorization". In: *Proceedings of the International Conference on Digital Audio Effects (DAFx)*. Birmingham, UK, Sept. 2019.

[34] Umut Şimşekli et al. "Real-Time Recognition of Percussive Sounds by a Model-Based Method". In: *EURASIP Journal on Advances in Signal Processing* 2011.1 (2010), p. 291860. DOI: 10.1155/2011/291860. URL: https://doi.org/10.1155/2011/291860.

[35] Paris Smaragdis and Shrikant Venkataramani. "A Neural Network Alternative to Non-Negative Audio Models". In: *CoRR* abs/1609.03296 (2016). 1609.03296. URL: http://arxiv.org/abs/1609.03296.

[36] Vinay Maddali. "Comparison of Neural Networks and Nonnegative Matrix Factorization approaches for Speech Denoising". In: 2014.

[37] Lee Callender, Curtis Hawthorne, and Jesse Engel. *Improving Perceptual Quality of Drum Transcription with the Expanded Groove MIDI Dataset*. 2020.

[38] Curtis Hawthorne et al. "Onsets and Frames: Dual-Objective Piano Transcription". In: *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, 2018*. 2018. URL: https://arxiv.org/abs/1710.11153.

[39] Jon Gillick et al. "Learning to Groove with Inverse Sequence Transformations". In: *International Conference on Machine Learning (ICML)*. 2019.

[40] *Magenta*. en. URL: https://magenta.tensorflow.org/. Retrieved on Apr. 28, 2020.

[41] Colin Raffel et al. *Mir_Eval: A Transparent Implementation Of Common Mir Metrics*. ID: datacite16454804. 2014. DOI: 10.5281/ZENODO.1416527.

[42] Yutaka Sasaki. "The truth of the F-measure". In: *Teach Tutor Mater* (2007).

[43] Anne Danielsen et al. "Where Is the Beat in That Note? Effects of Attack, Duration, and Frequency on the Perceived Timing of Musical and Quasi-Musical Sounds". In: *Journal*

of Experimental Psychology: Human Perception and Performance 45.3 (2019), pp. 402–418. DOI: 10.1037/xhp0000611.

[44] J. P. Bello et al. "A tutorial on onset detection in music signals". In: *IEEE Transactions on Speech and Audio Processing* 13.5 (2005), pp. 1035–1047. DOI: 10.1109/TSA.2005.851998.

[45] Christian Dittmar and Daniel Gärtner. "Real-Time Transcription and Separation of Drum Recordings based on NMF Decomposition". In: *Proceedings of the International Conference on Digital Audio Effects (DAFx)*. Erlangen, Germany, Sept. 2014, pp. 187–194.

[46] MATLAB. *version 9.7.0.1296695 (R2019b Update 4)*. Natick, Massachusetts, United States: The MathWorks Inc, 2020.

[47] Olivier Lartillot et al. *A Matlab Toolbox for Music Information Retrieval*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 261–268. ISBN: 9783540782391. DOI: 10.1007/978-3-540-78246-9_31.

[48] Tuomas Eerola and Petri Toiviainen. *MIDI Toolbox: MATLAB Tools for Music Research*. Jyväskylä, Finland: University of Jyväskylä, 2004. URL: www.jyu.fi/musica/miditoolbox/.

[49] Richard G. Lyons. *Understanding digital signal processing*. 2nd ed. ID: alma9920758531705762. Upper Saddle River, NJ: Prentice Hall PIR, 2004. ISBN: 0131089897.

[50] Greg Stanley. *Exponential Filter*. URL: https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Filtering/Exponential-Filter/exponential-filter.htm. Retrieved on May 23, 2020.

[51] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[52] Anne Danielsen et al. "Effects of instructed timing and tempo on snare drum sound in drum kit performance". In: *The Journal of the Acoustical Society of America* 138.4 (2015), pp. 2301–2316. DOI: 10.1121/1.4930950. URL: https://doi.org/10.1121/1.4930950.

[53] Ahmed BenSaïda. *Shapiro-Wilk and Shapiro-Francia normality tests*. June 2014. URL: https://www.mathworks.com/matlabcentral/fileexchange/13964-shapiro-wilk-and-shapiro-francia-normality-tests. Retrieved on May 18, 2020.

[54] Andy P. Field. *Discovering statistics using IBM SPSS statistics : and sex and drugs and rock 'n' roll*. 4th ed. ID: alma9920567164905762. Los Angeles: Sage, 2013. ISBN: 9781446249185.

Appendix A

User Evaluation Questionnaire

- **Age?**
- **How many years have you been a musician?**

  [0-1 | 1-3 | 3-5 | 5-10 | 10+]
- **How would you rate your musical skills?**

  [Beginner | Hobbyist | Experienced | Professional]
- **Are drums your main instrument?**

  [Yes | No]
- **How would you rate your drumming skills?**

  [Beginner | Hobbyist | Experienced | Professional]
- **Do you usually practice and/or play to a metronome?**

  [Yes | No]
- **Have you used the Rhythm Coach function in a metronome or electronic drum kit**

  [Yes | No]
- **Do you have experience editing audio or composing music to a musical grid in a digital audio workstation, such Logic Pro X, or Ableton Live?**

  [Yes | No]
- **Do you practice with the Melodics app?**

  [Yes | No]
- **Did you find it easy to synchronize to the metronome?**

  [Yes | No]
- **Did you feel that the timing feedback was accurate?**

  [Yes | No]

Appendix B

Additional Exercise Pattern Figures

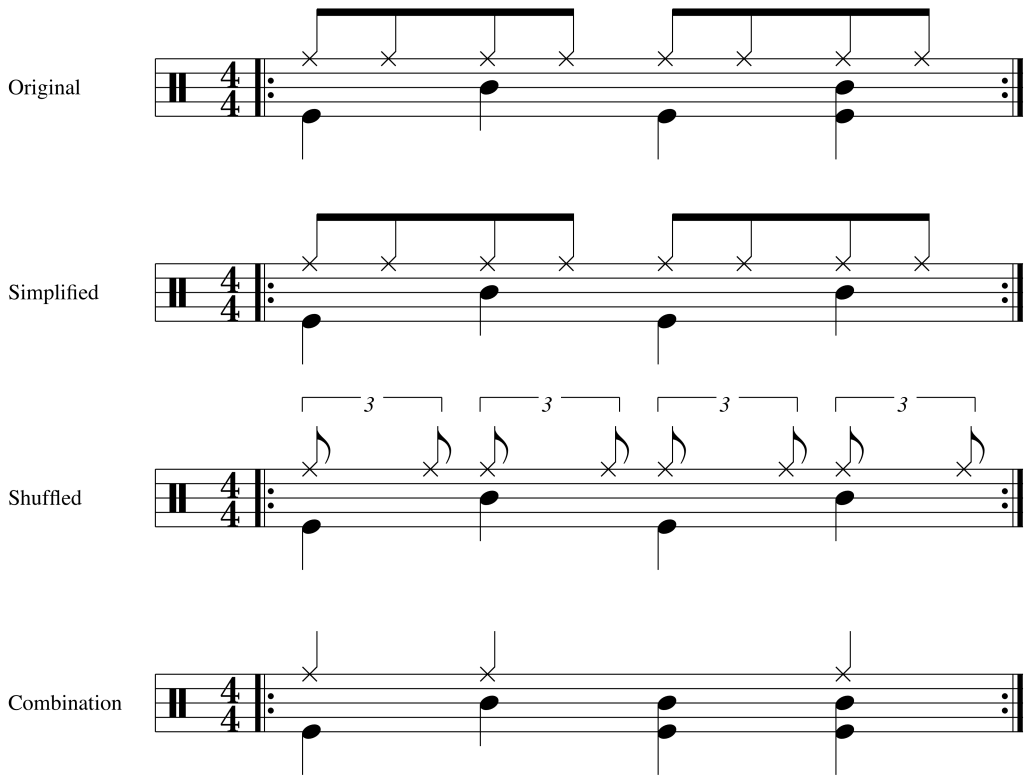


Figure B.1: Score of drum patterns used for system evaluation. Notation starting from the top: hi-hat, snare drum, bass drum

Appendix C

Additional Implementation Figures

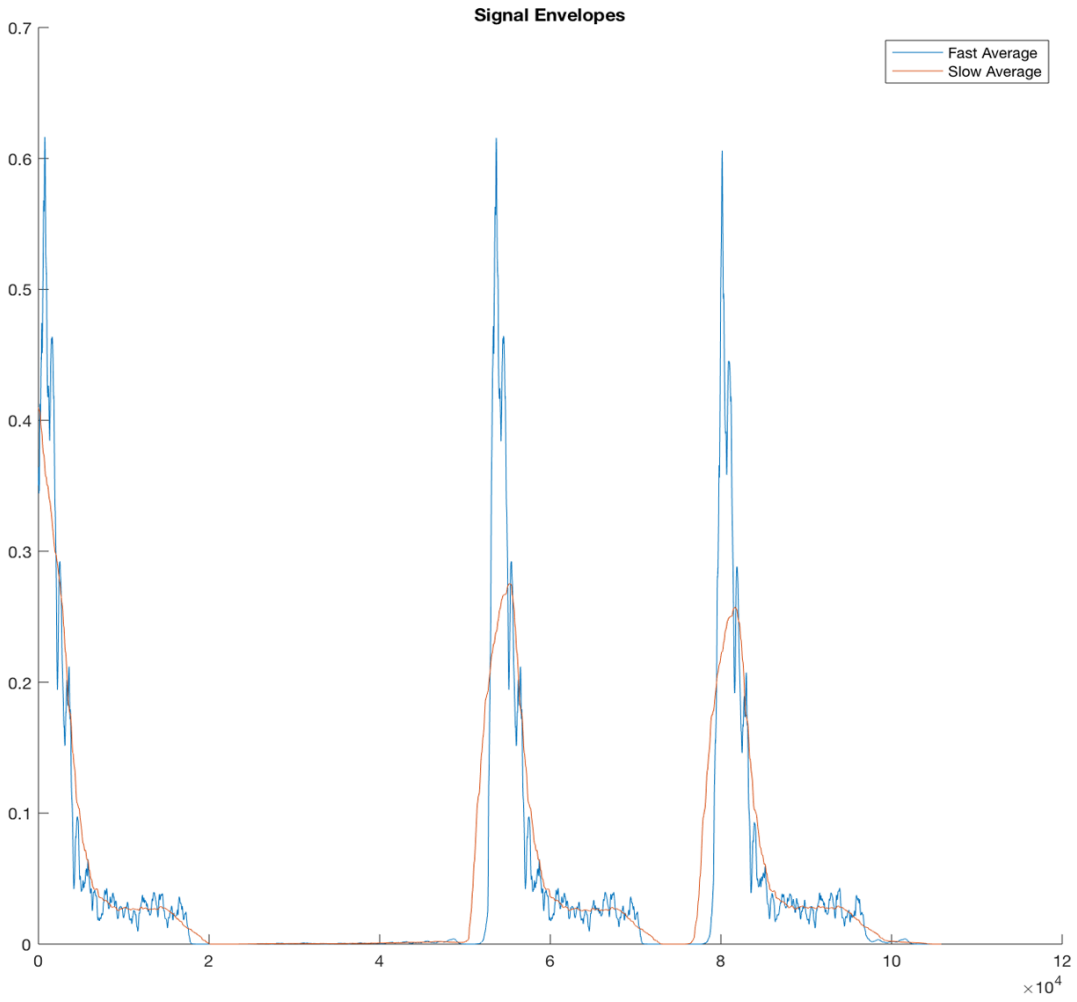


Figure C.1: Signal envelopes using slow and fast detectors

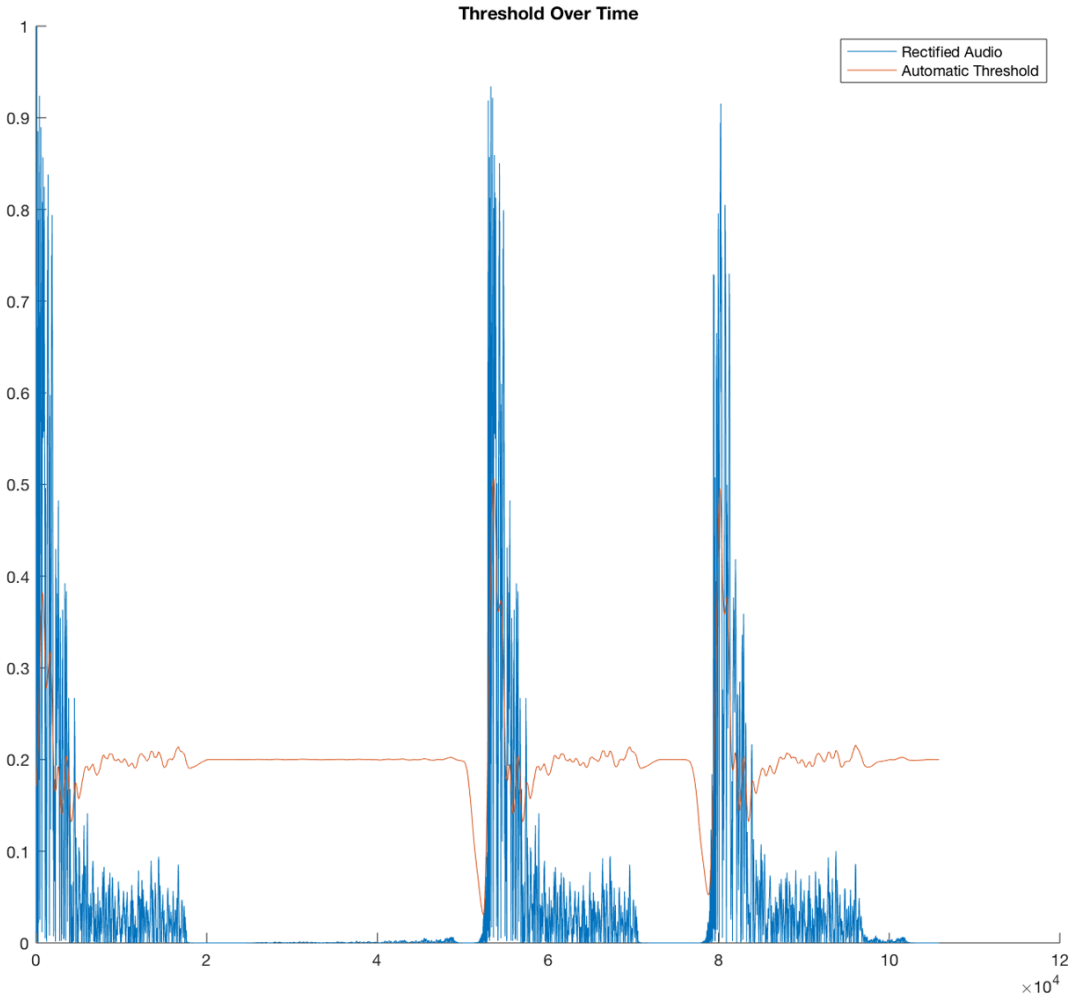


Figure C.2: Automatically derived threshold over time

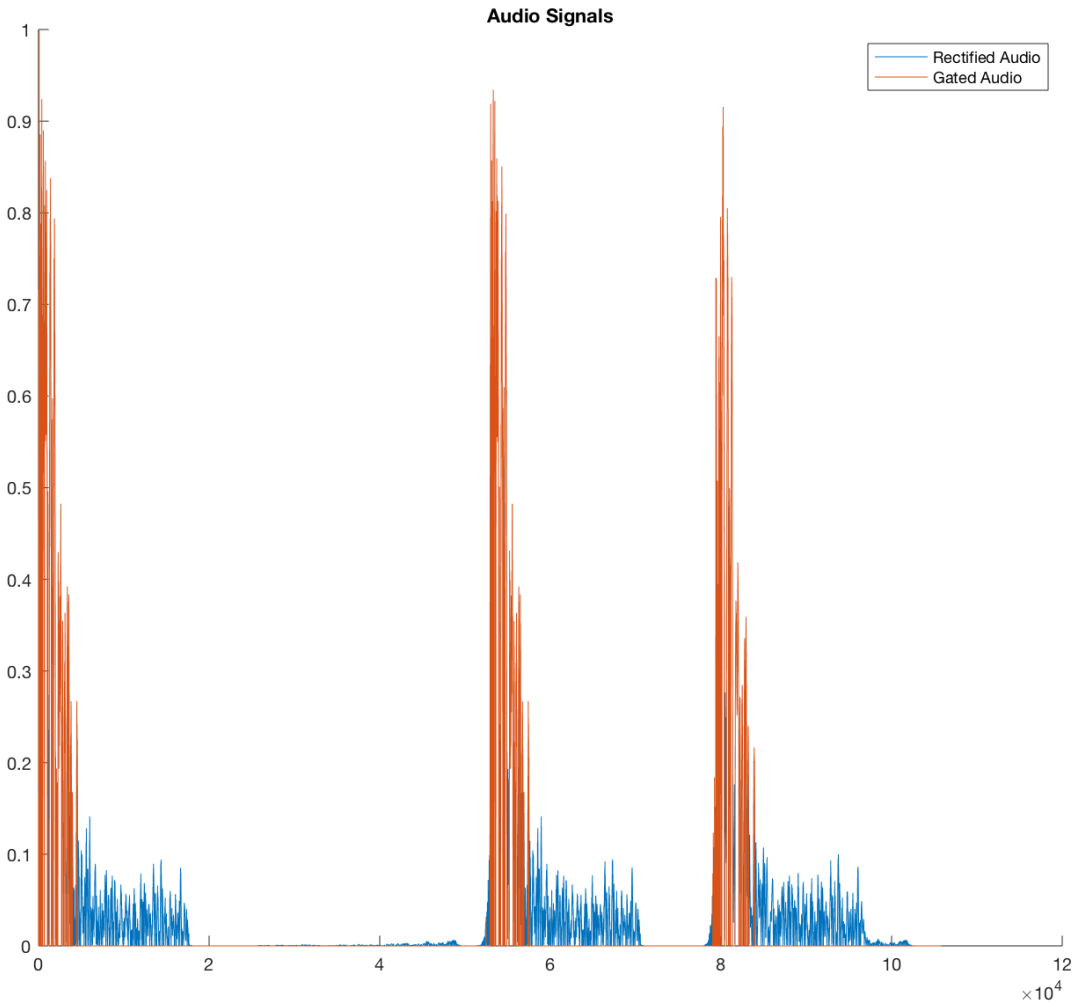


Figure C.3: Audio before and after thresholding

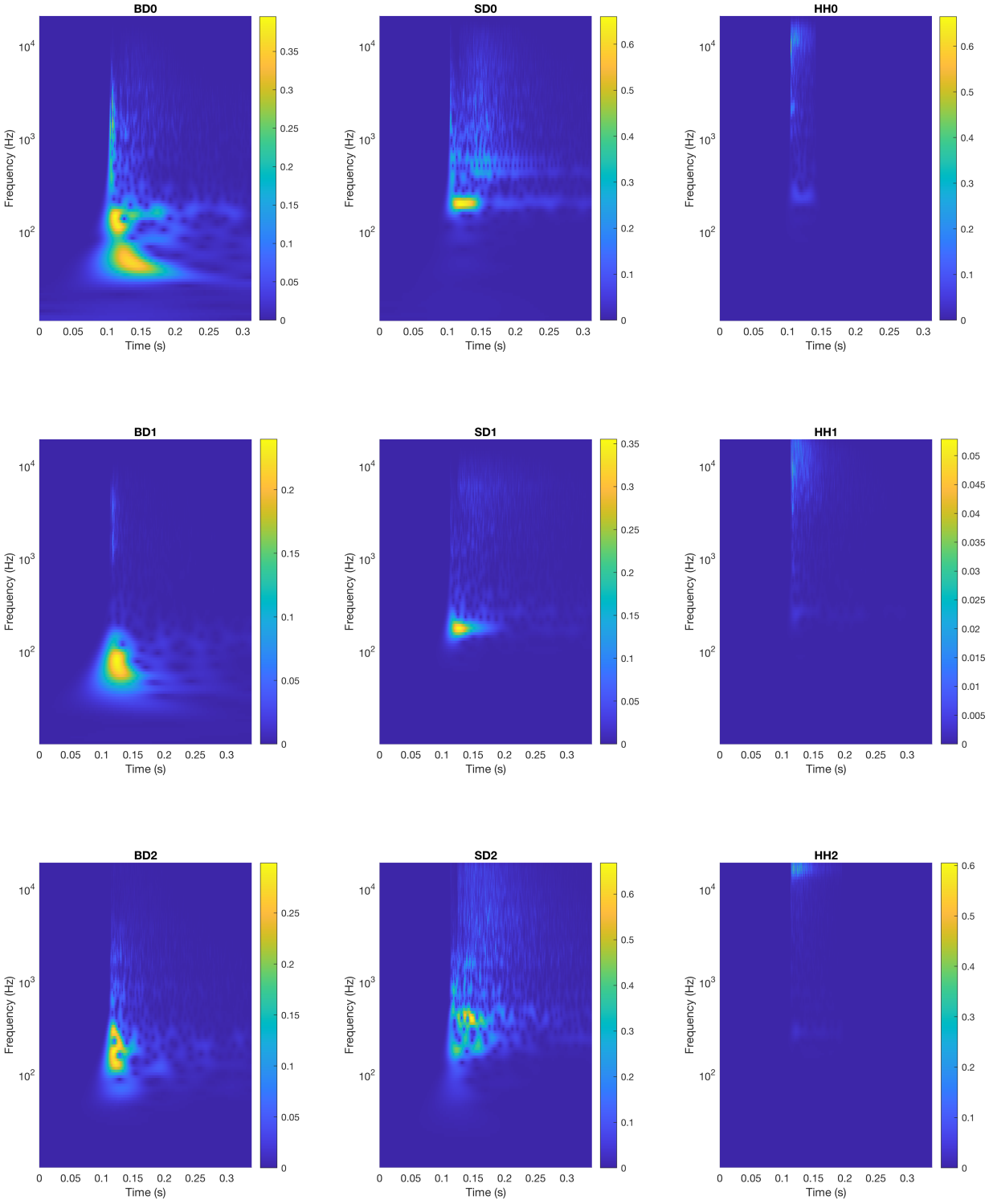


Figure C.4: CWT scalograms of bass drum, snare drum and hi-hat for three drum-kits used for system evaluation. 1: Samples recorded with an iPhone 6S in a large theater, 2: Samples from Logic Pro X virtual instrument, 3: Samples recorded with a MacBook Pro's built-in microphone in a medium-sized, reverberant room (taken from user evaluation).

Appendix D

Additional System Evaluation Figures

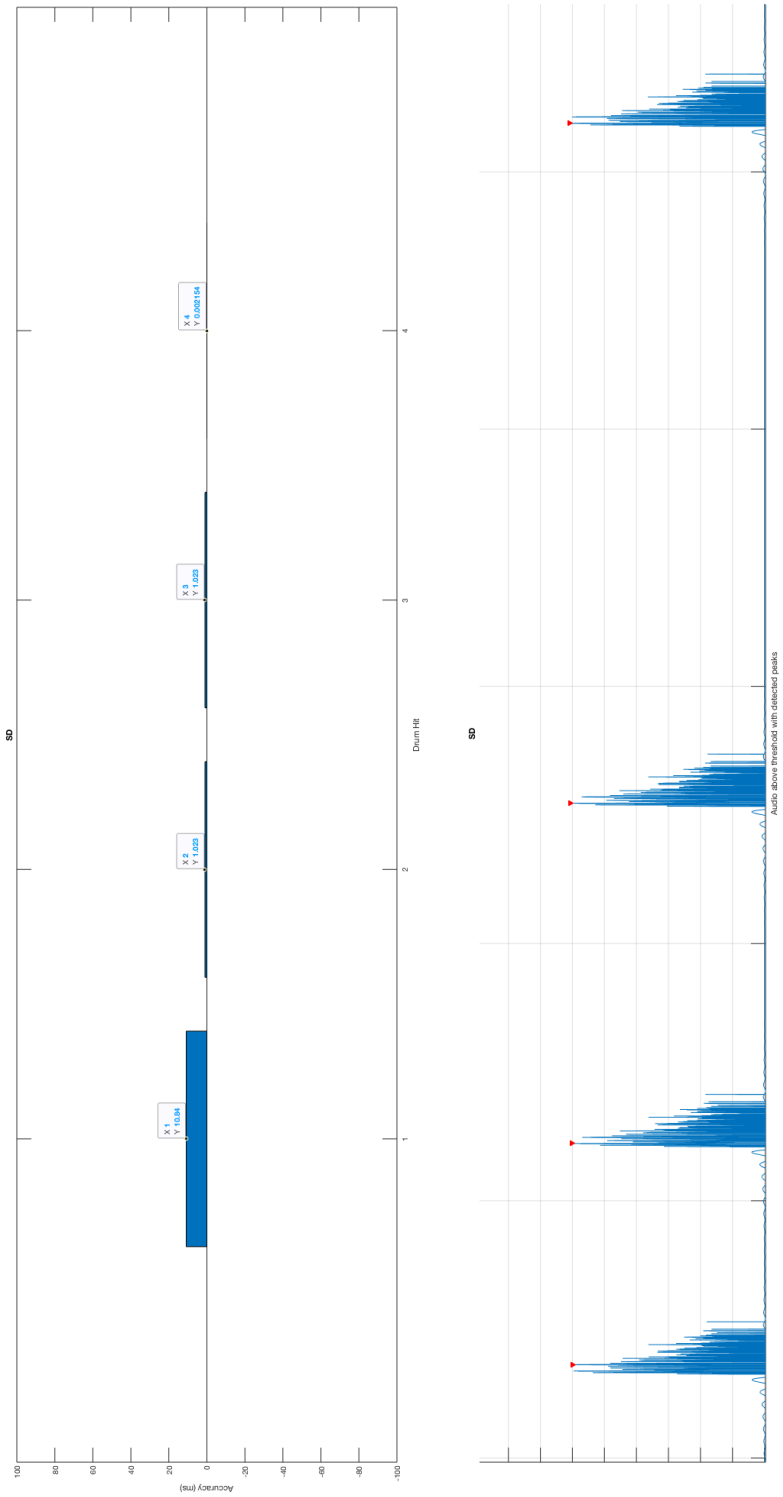


Figure D.1: Snare drum peak-picking issue prior to EMA addition. The first and last onsets are perfectly timed reference onsets, the second and third onsets are deliberately shifted forward by 1ms. The first onset is incorrectly reported as lagging by 10.84ms due to the waveform having changed shape as seen in the second row. Comparatively, the last peak is correctly detected as being on time. The second and third onsets, which are deliberately shifted, are correctly detected as 1.02ms

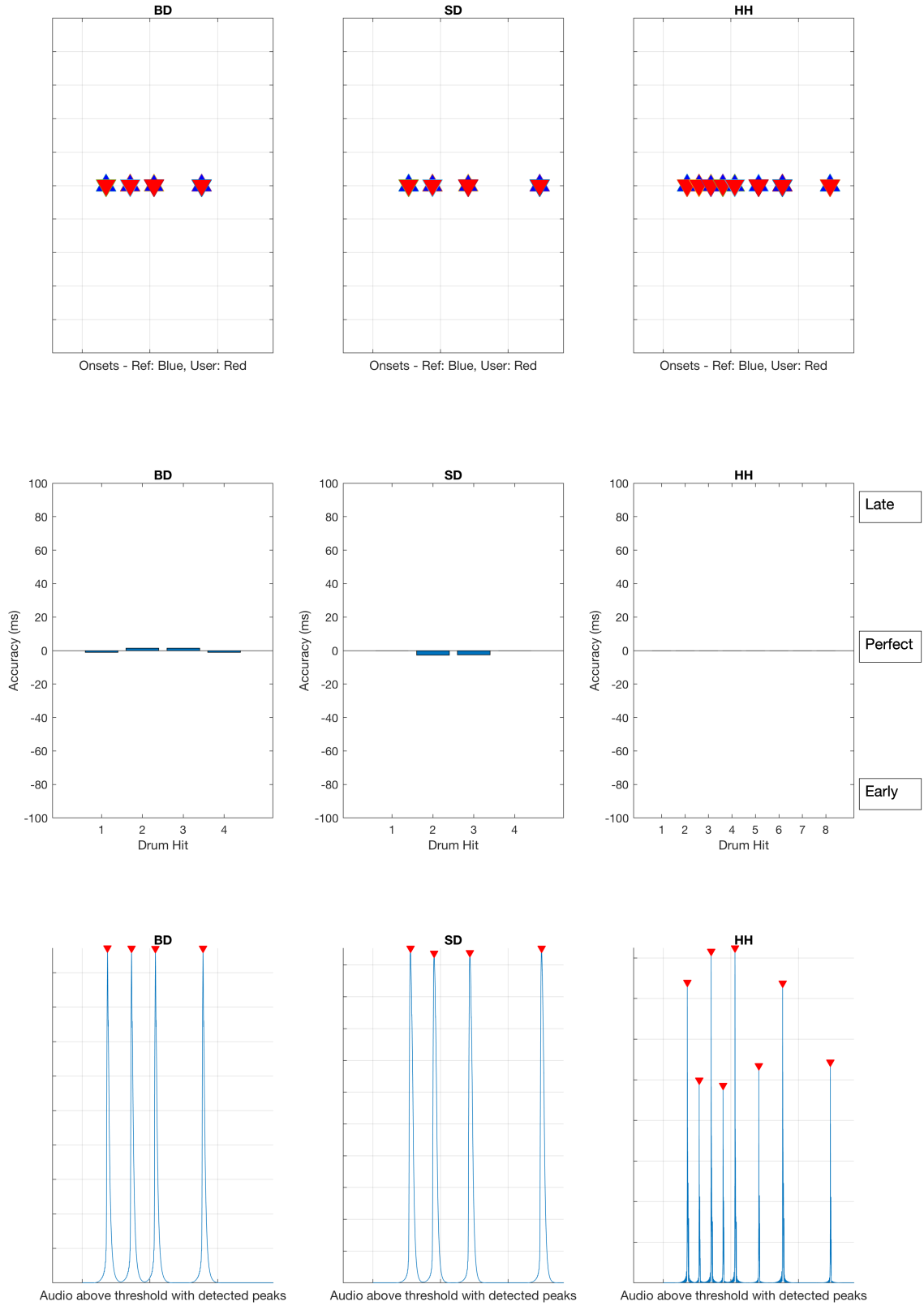


Figure D.2: System test results *(testMode = inter-limb)* showing inter-limb timing detection accuracy with deliberate 1ms shifts in BD and SD with the reference synthetic drum-kit (BD1,SD1,HH1 in figure C.4)

Appendix E

Additional User Evaluation Figures

User Evaluation Figures

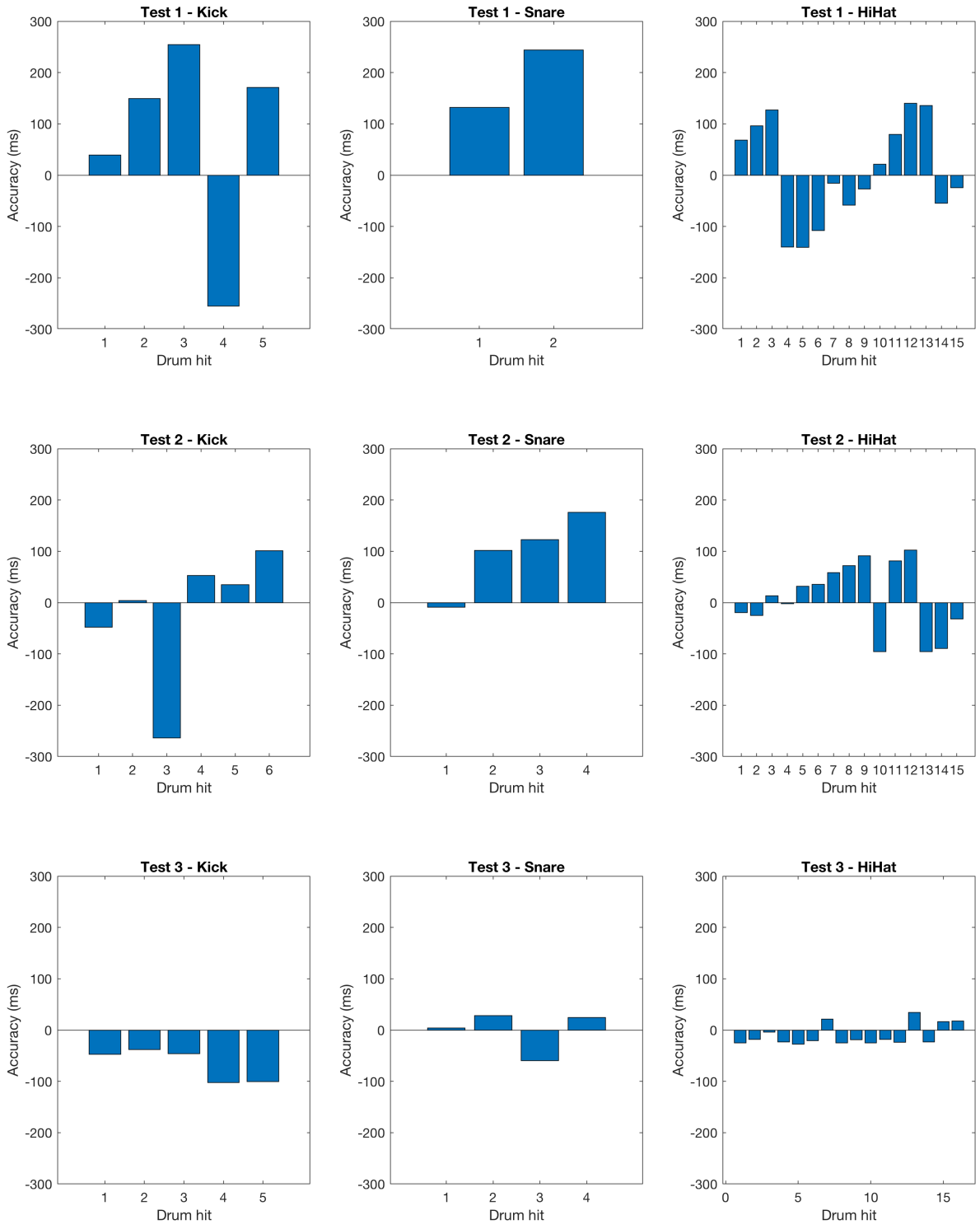


Figure E.1: Onset accuracy for test participant LK without timing feedback performing the original exercise pattern

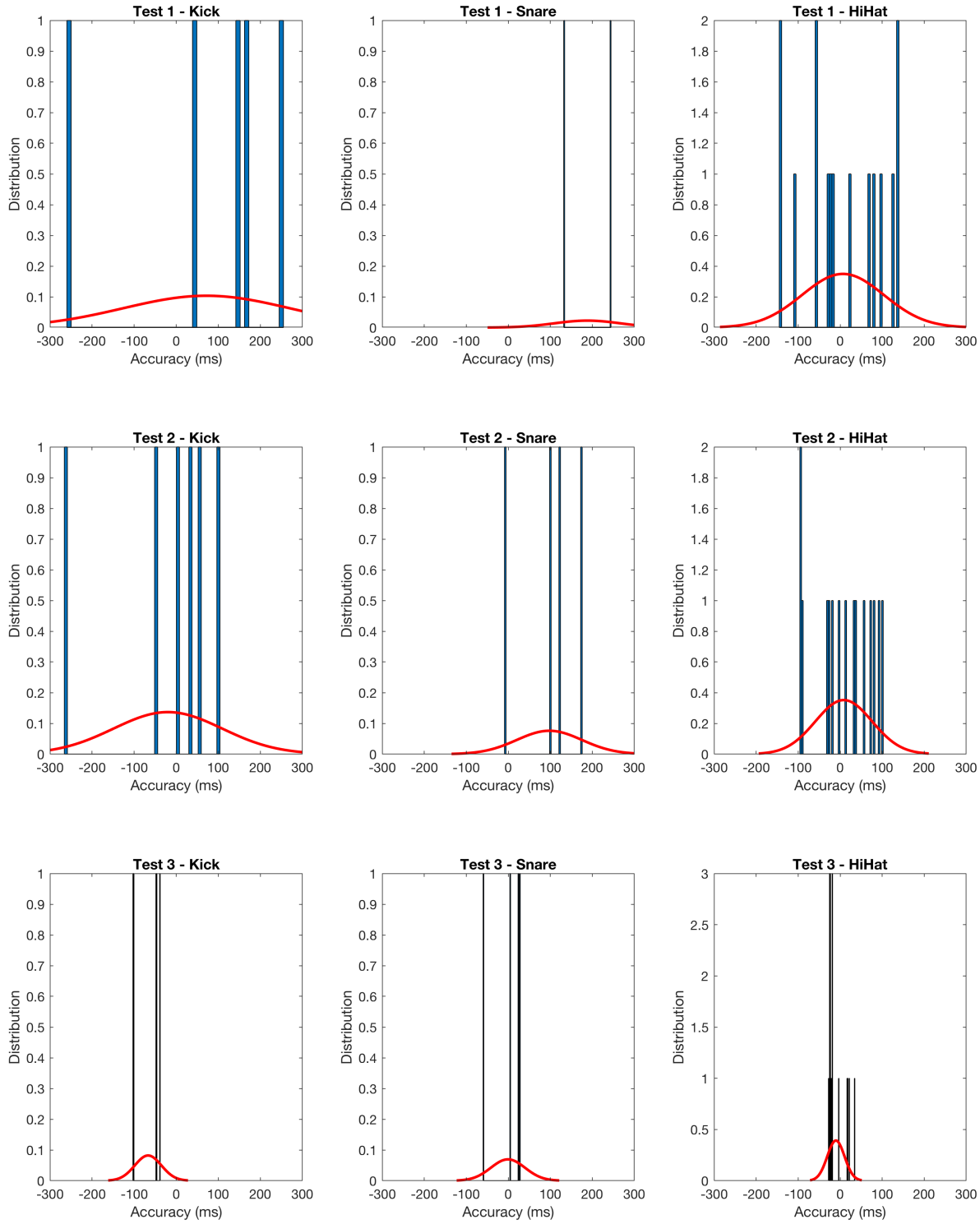


Figure E.2: Onset distribution for test participant LK without timing feedback performing the original exercise pattern

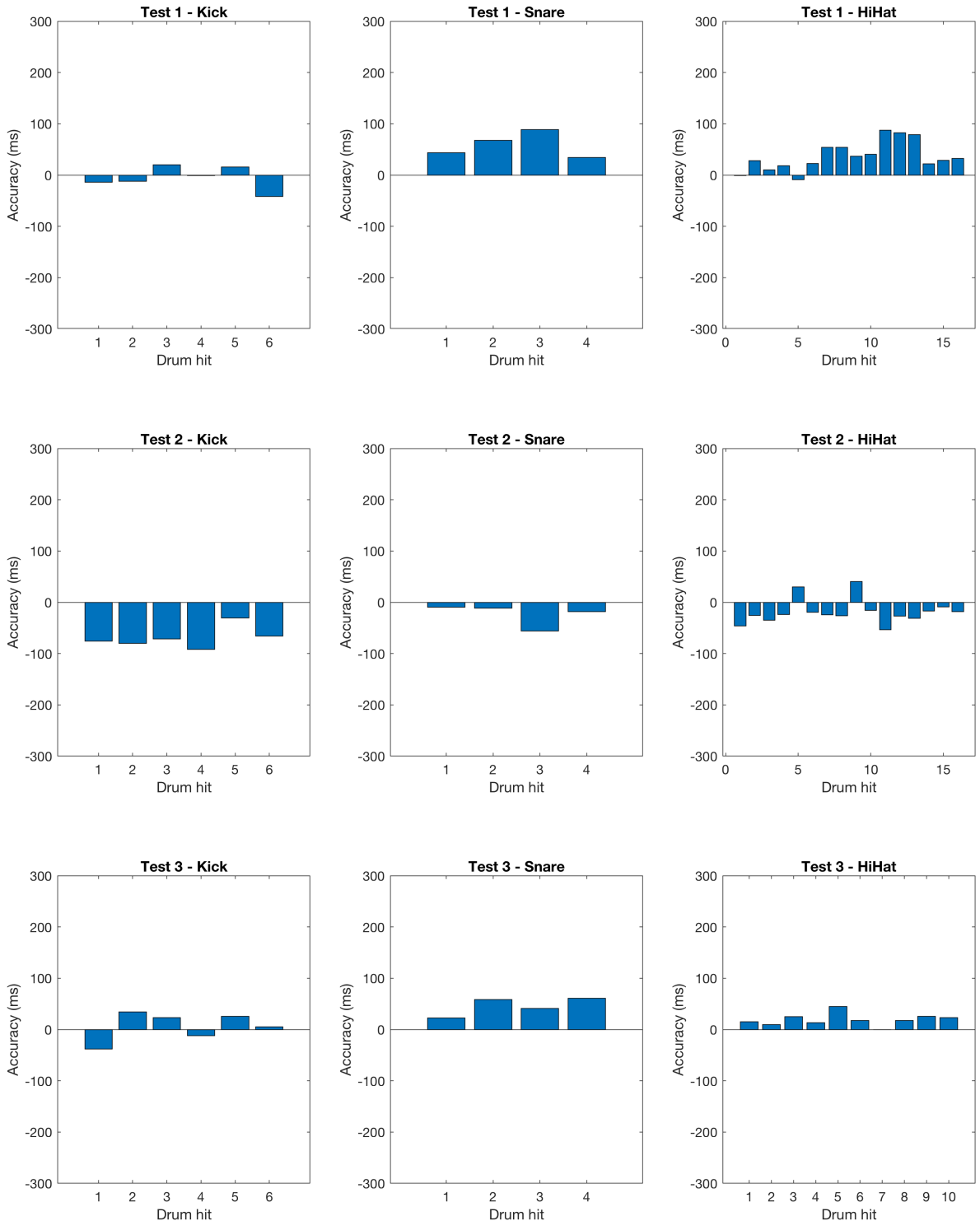


Figure E.3: Onset accuracy for test participant LK with timing feedback performing the original exercise pattern

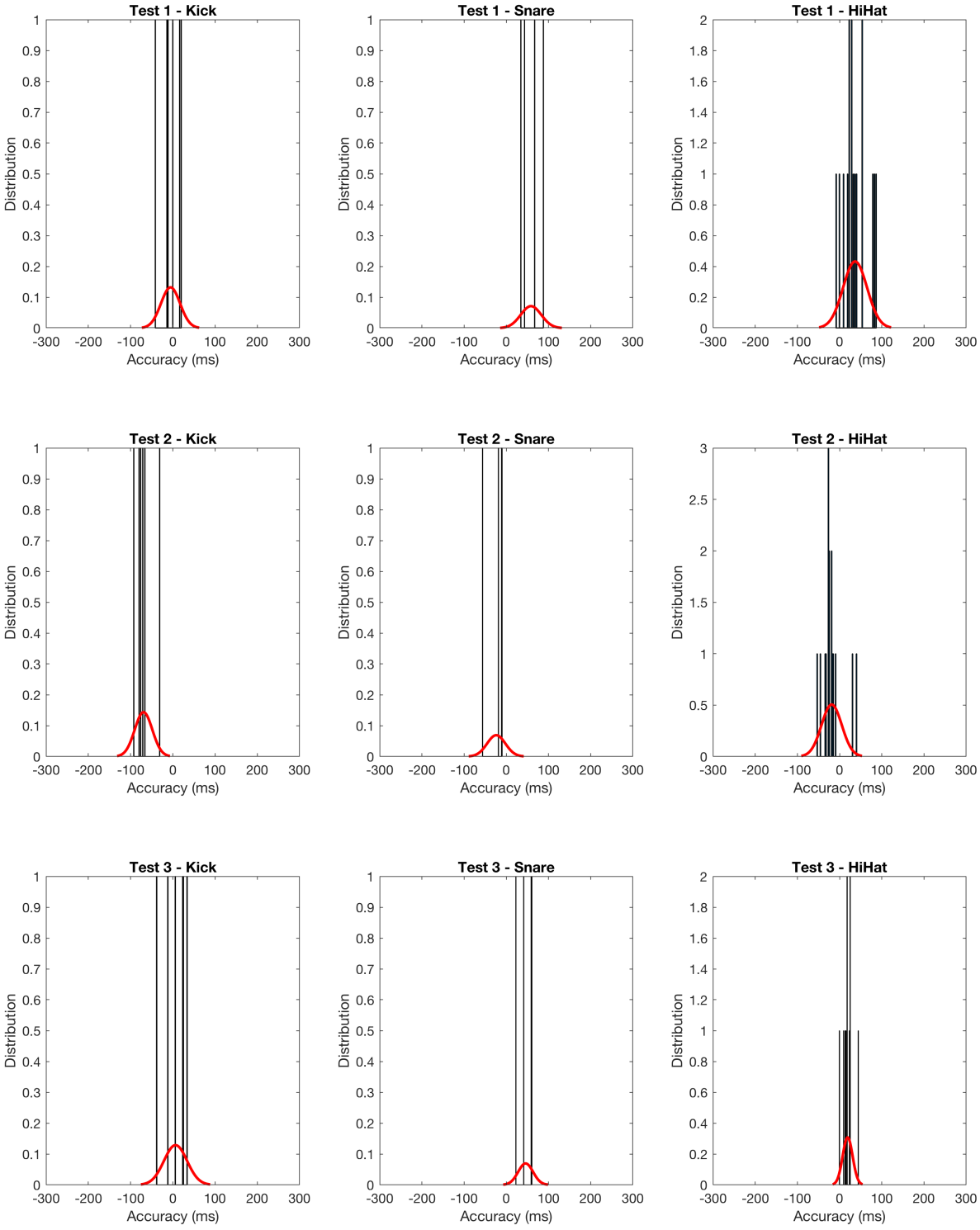


Figure E.4: Onset distribution for test participant LK with timing feedback performing the original exercise pattern

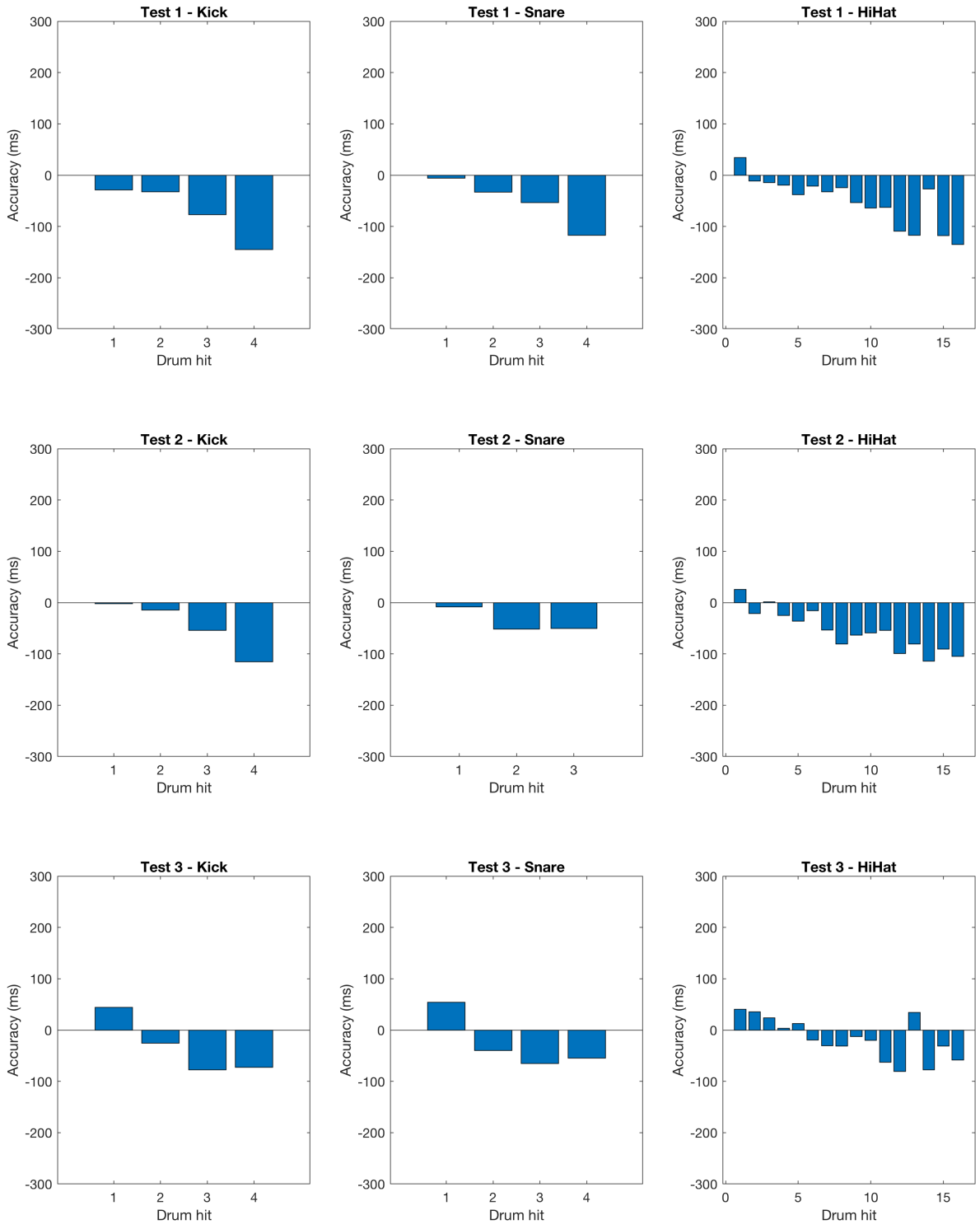


Figure E.5: Onset accuracy for test participant M without timing feedback performing the original exercise pattern

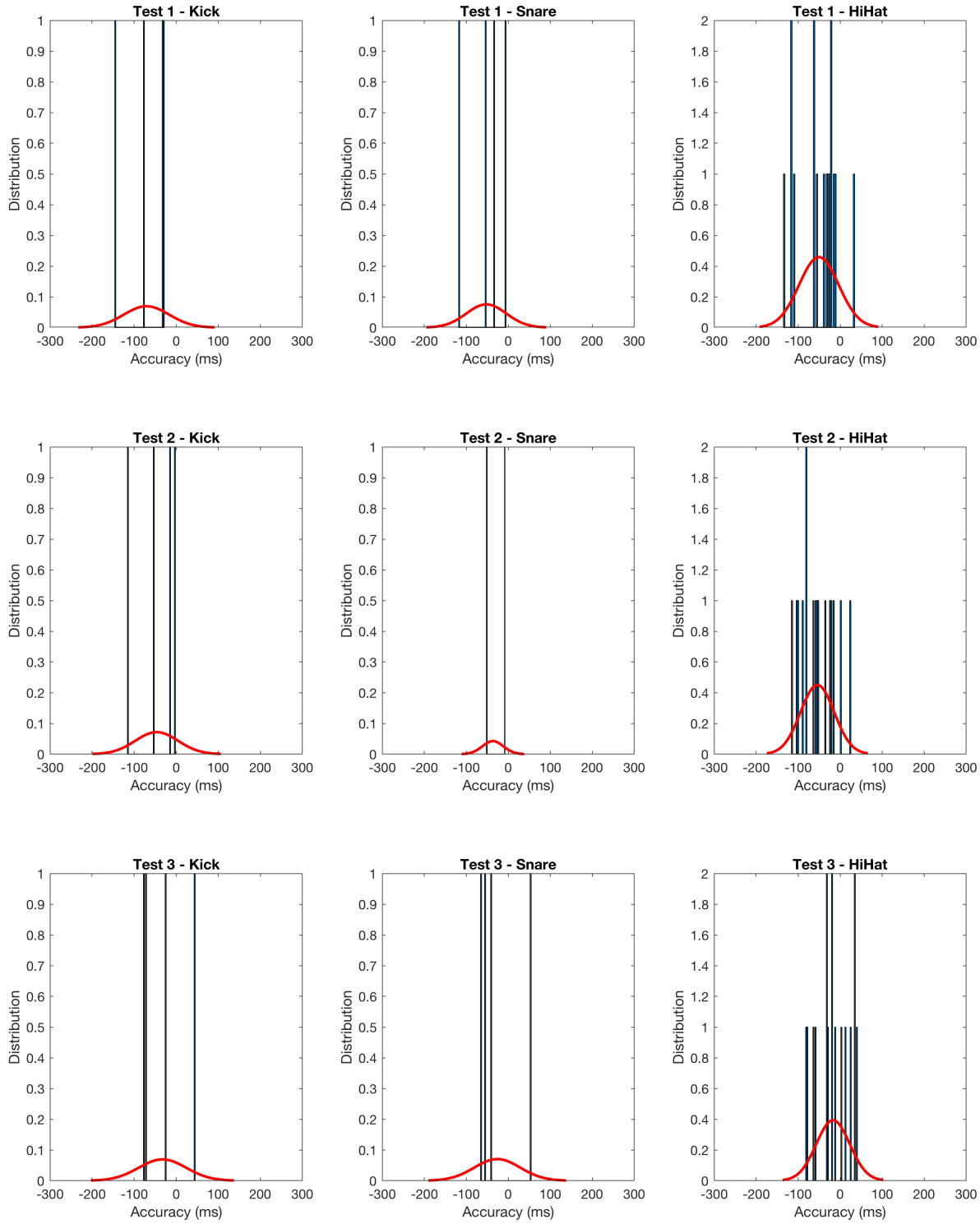


Figure E.6: Onset distribution for test participant M without timing feedback performing the original exercise pattern

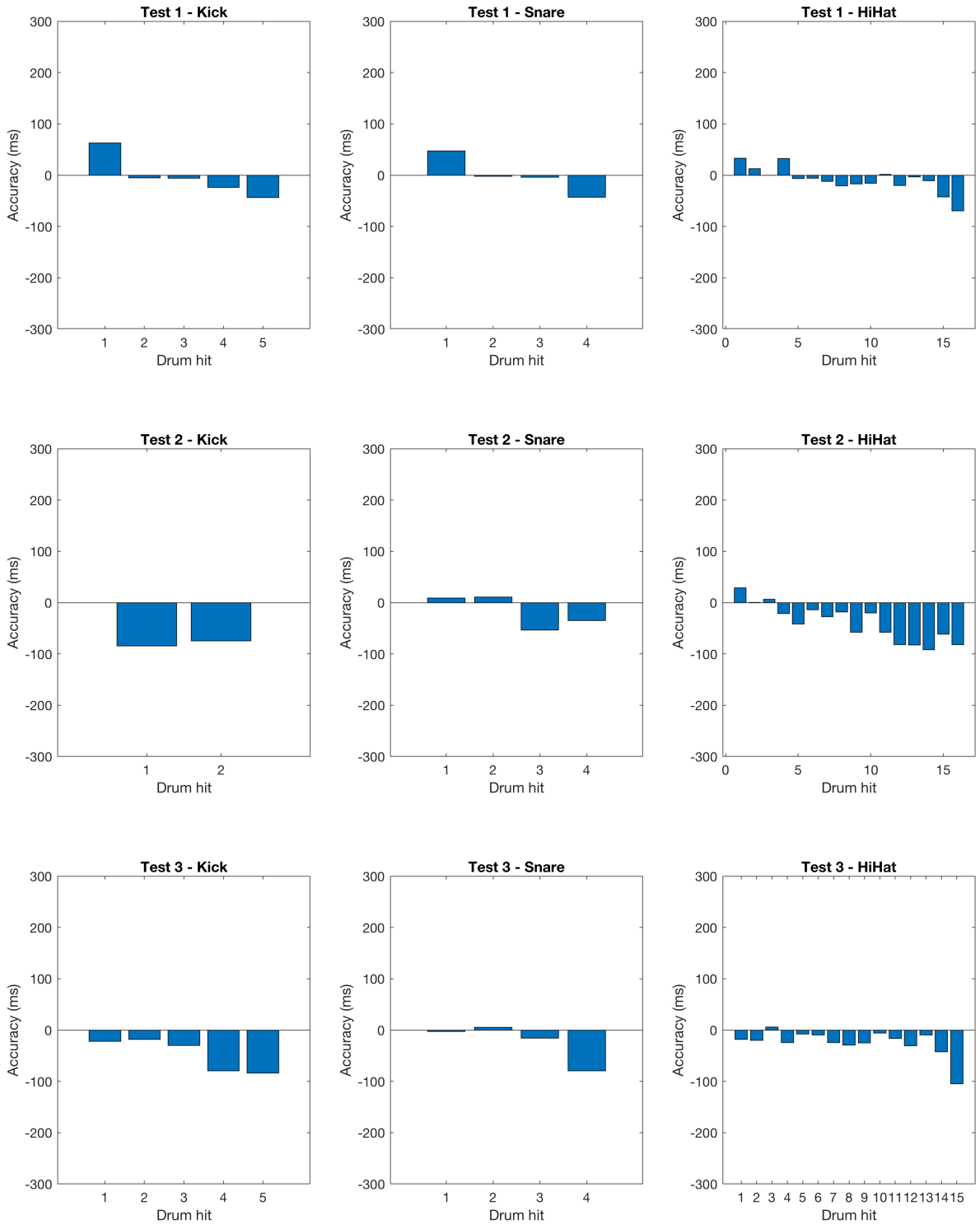


Figure E.7: Onset accuracy for test participant M with timing feedback performing the original exercise pattern

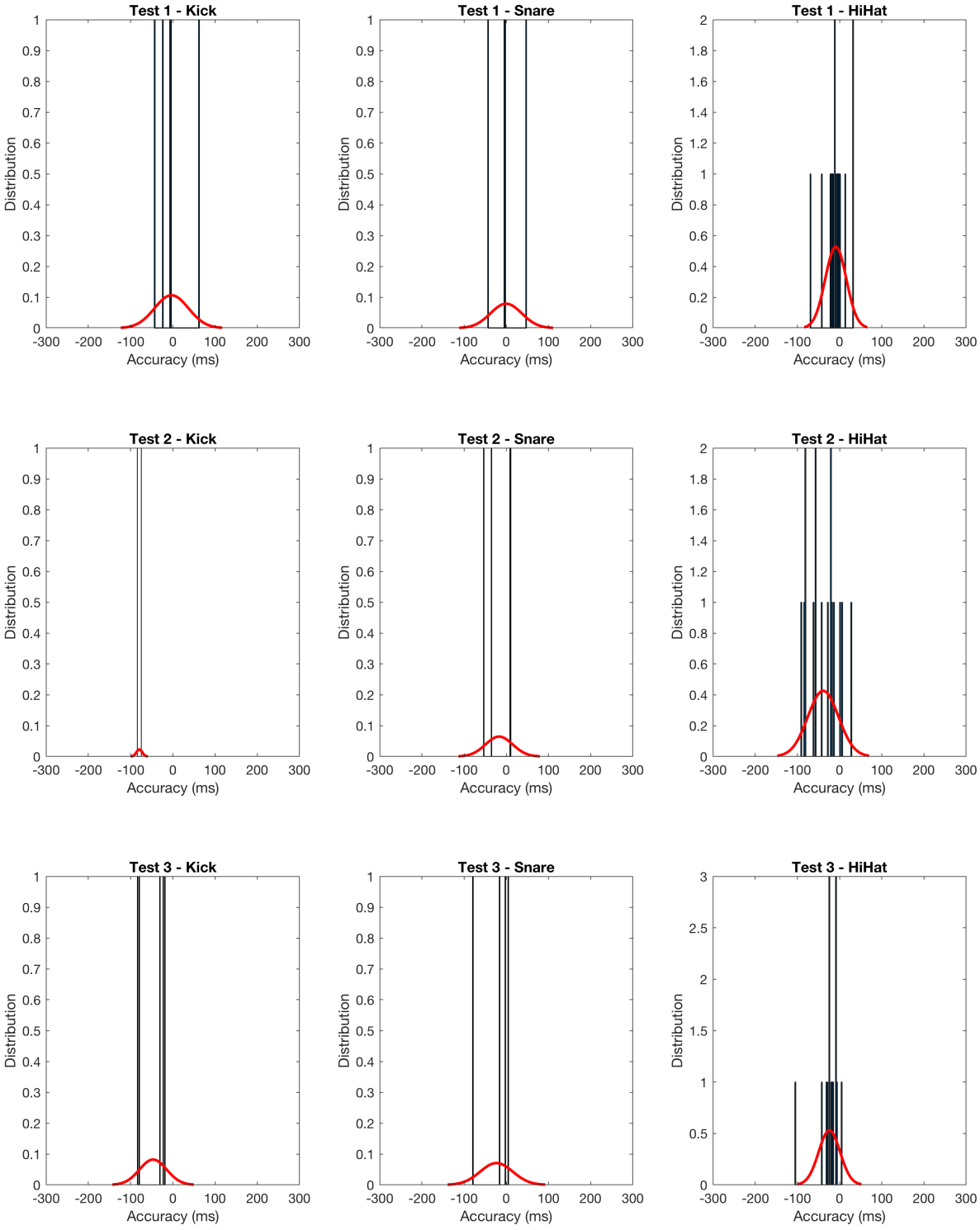


Figure E.8: Onset distribution for test participant M with timing feedback performing the original exercise pattern

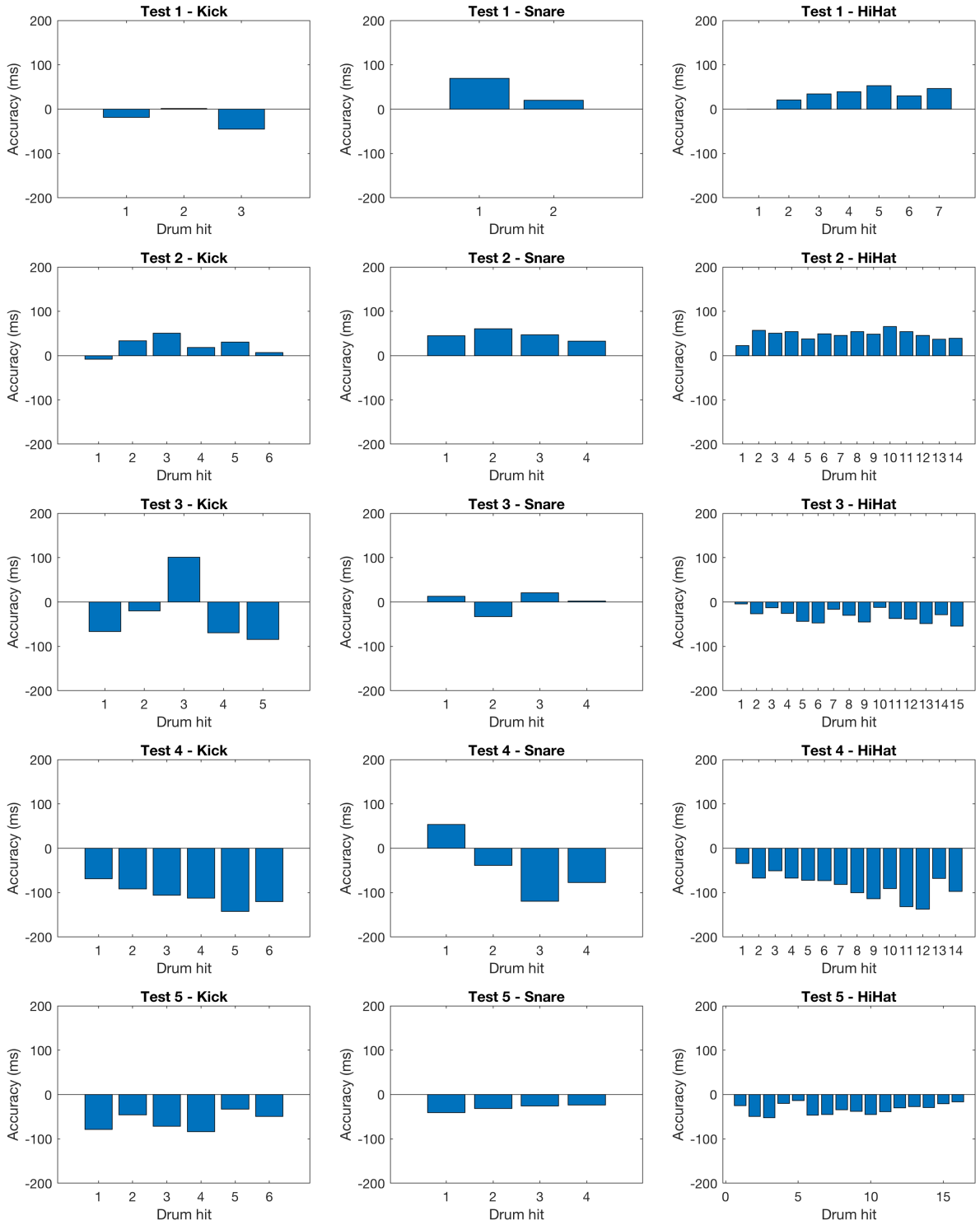


Figure E.9: Onset accuracy for test participant MP without timing feedback performing the original exercise pattern

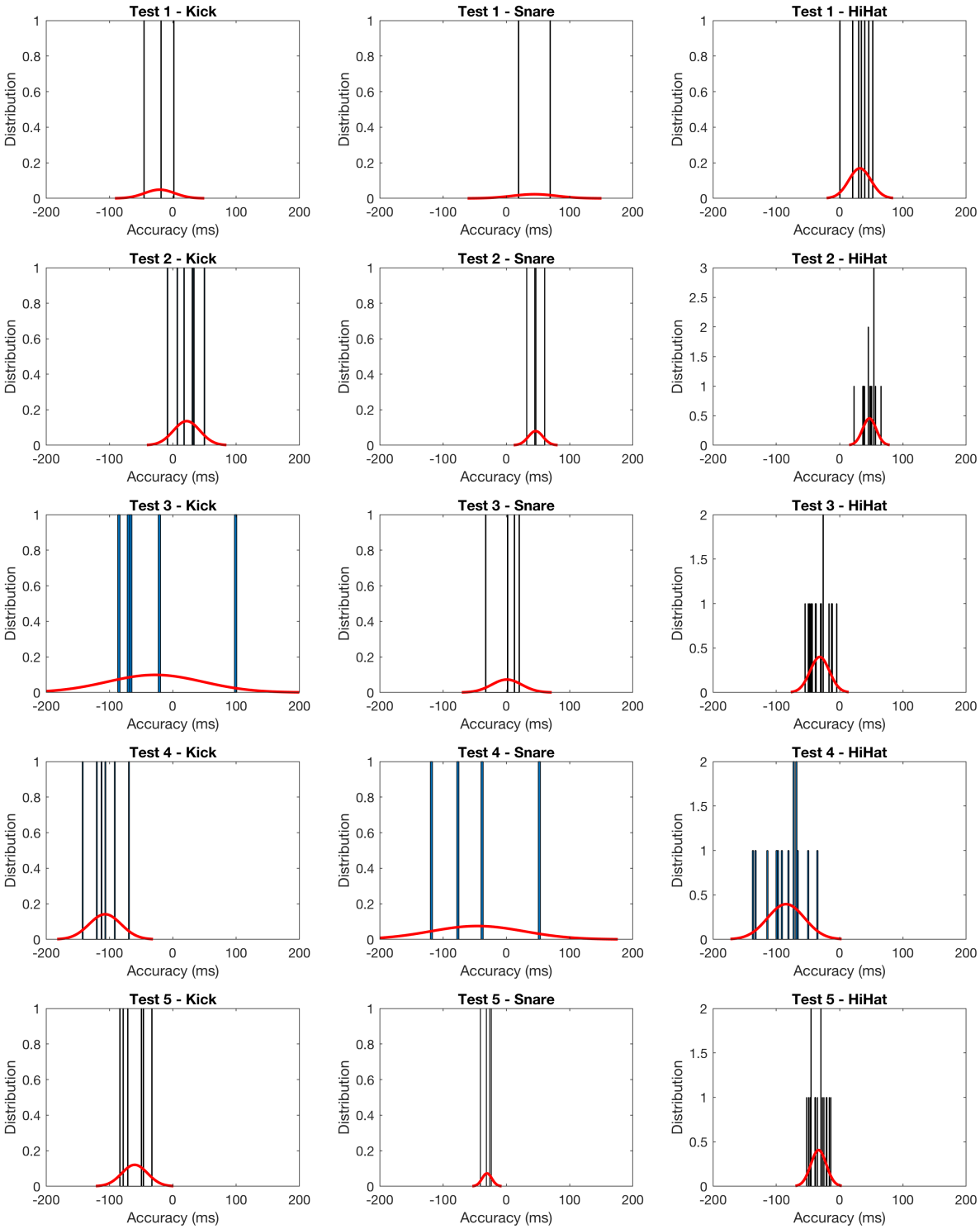


Figure E.10: Onset distribution for test participant MP without timing feedback performing the original exercise pattern

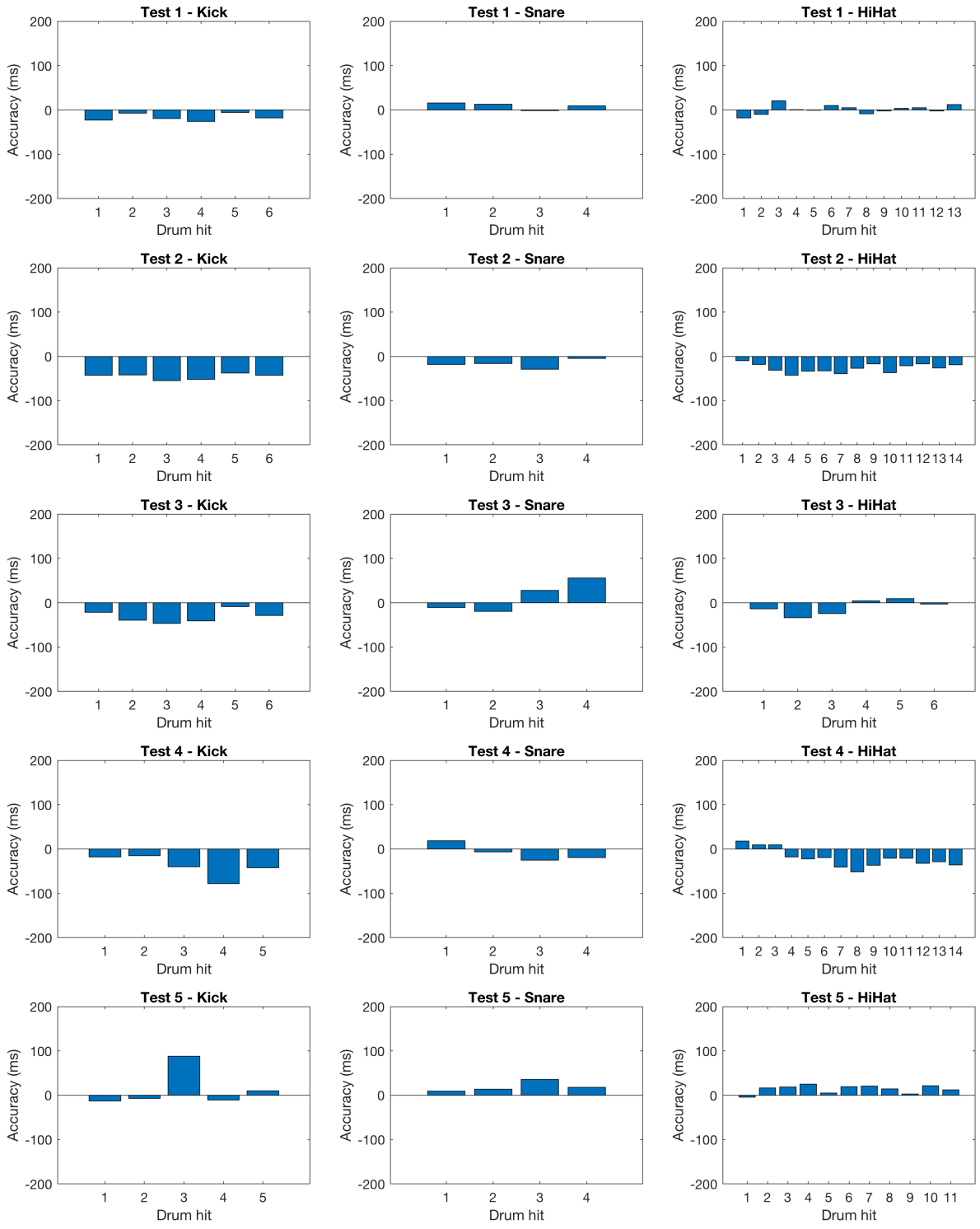


Figure E.11: Onset accuracy for test participant MP with timing feedback performing the original exercise pattern

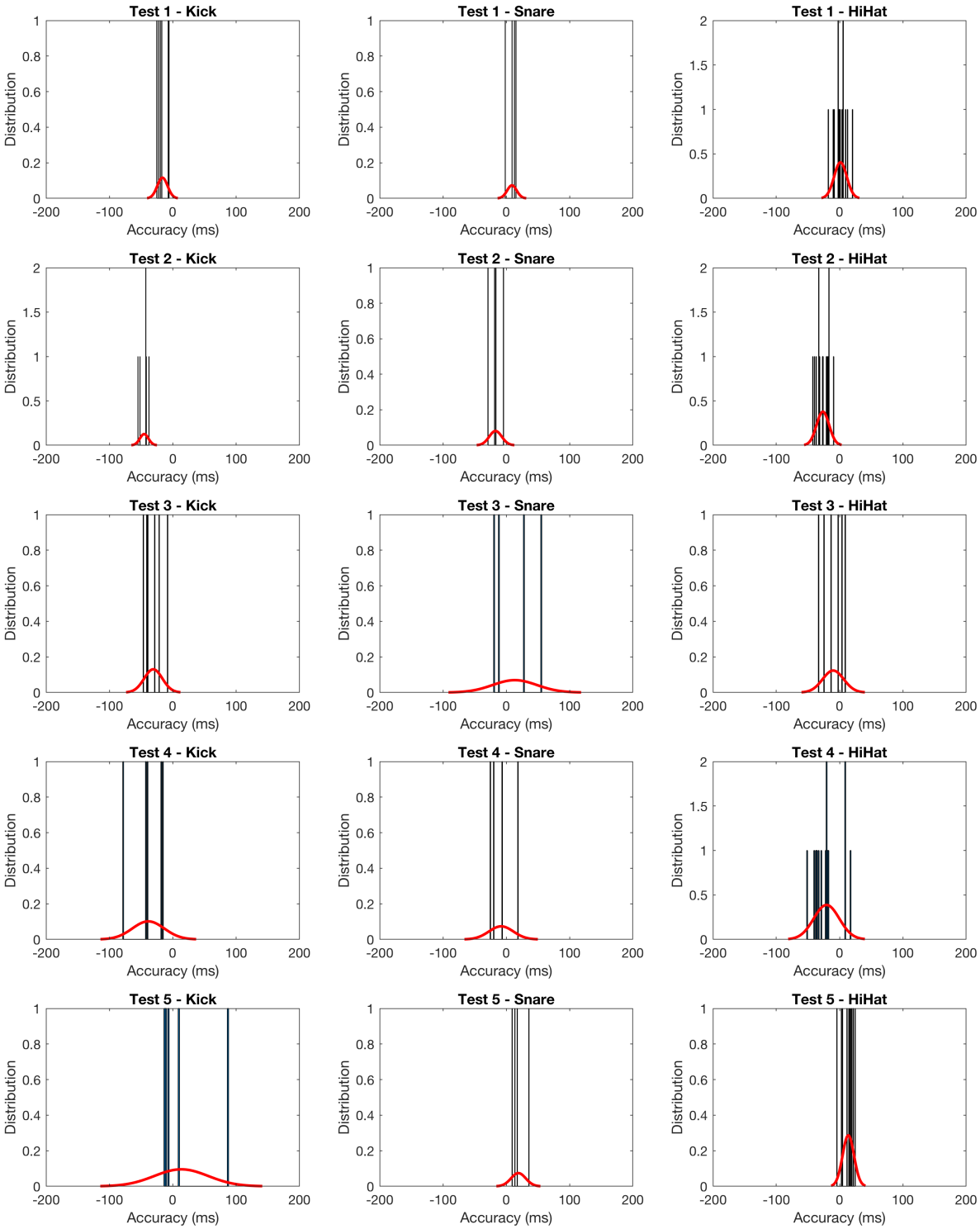


Figure E.12: Onset distribution for test participant MP with timing feedback performing the original exercise pattern

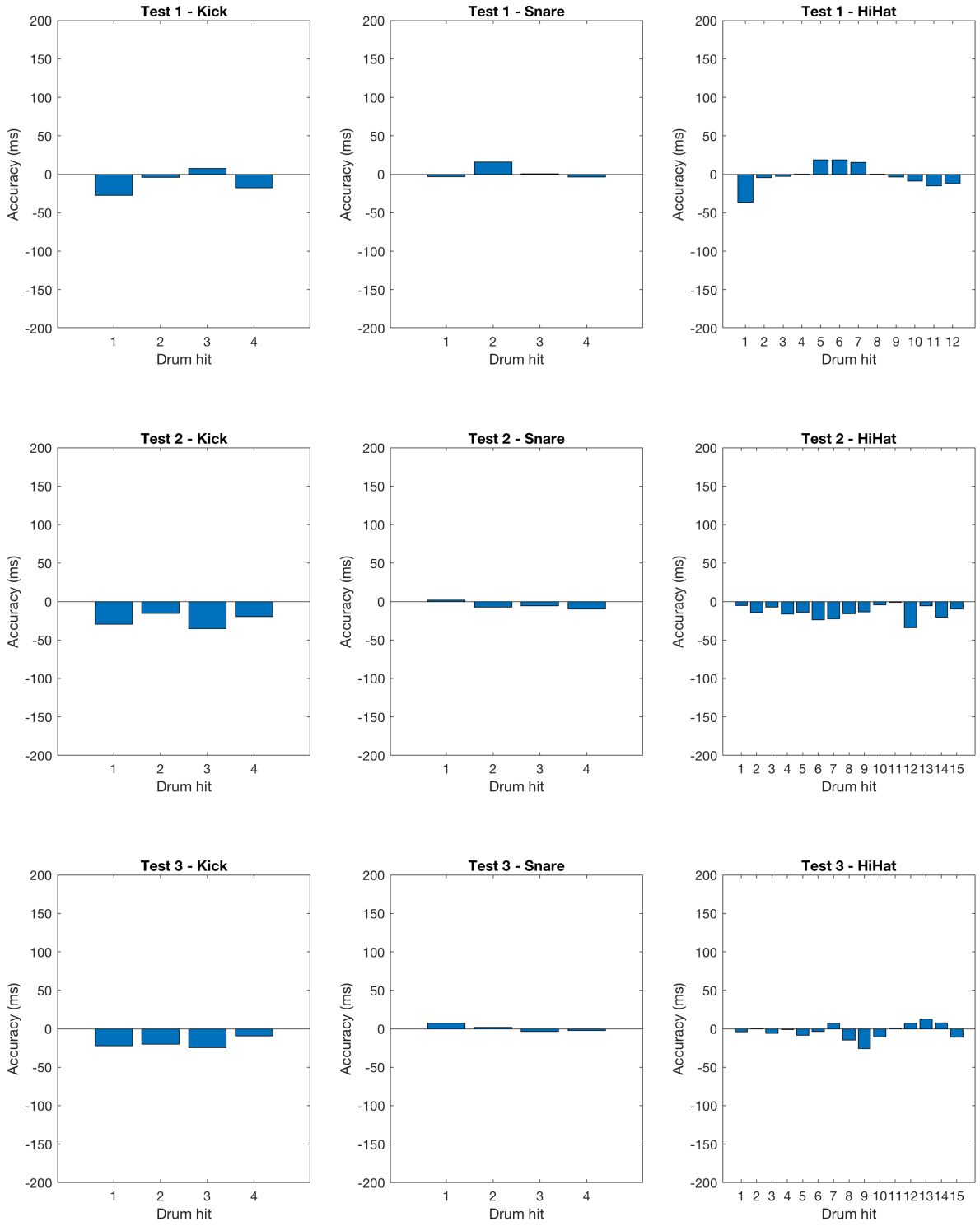


Figure E.13: Onset accuracy for test participant MP without timing feedback performing the simplified exercise pattern

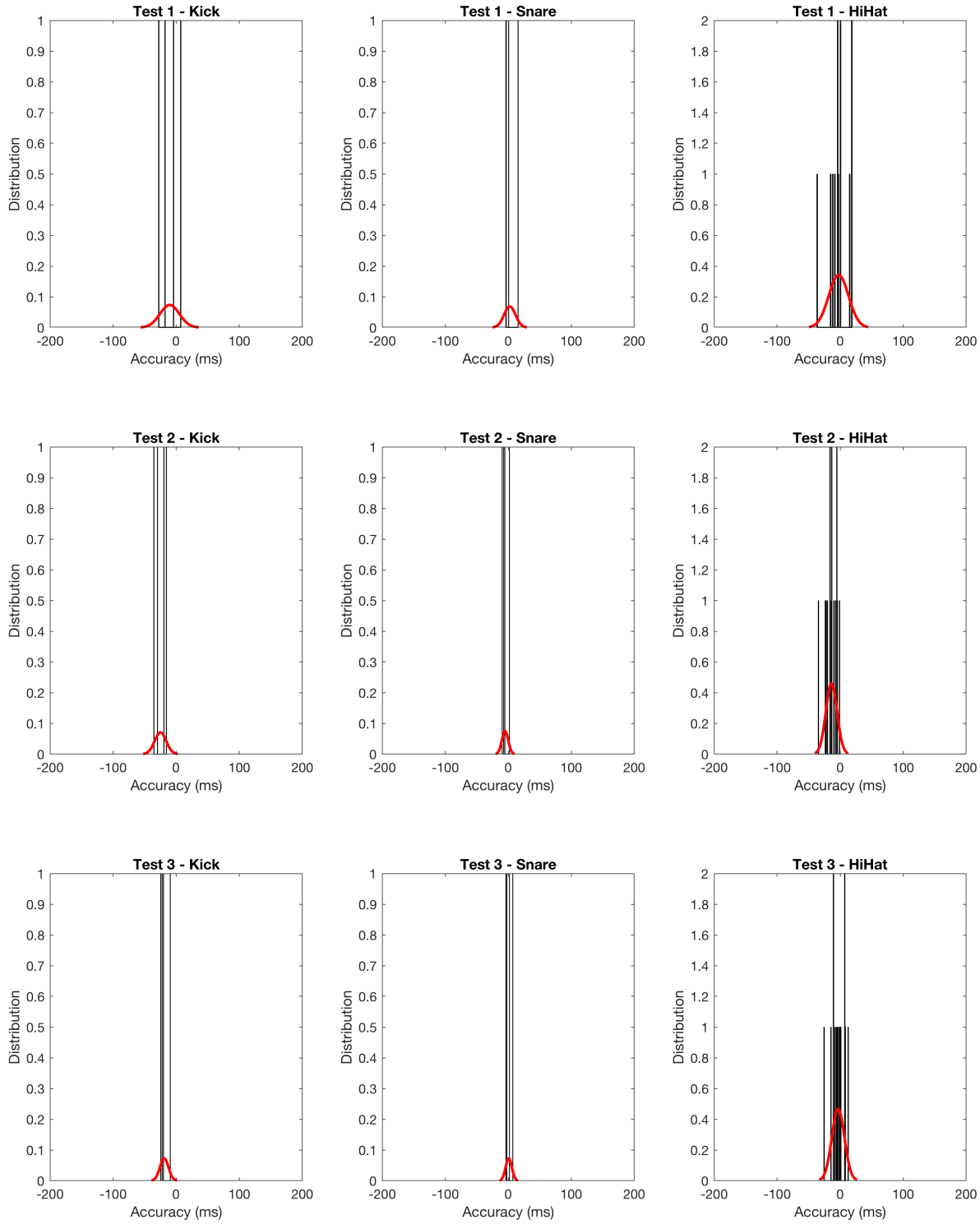


Figure E.14: Onset distribution for test participant MP without timing feedback performing the simplified exercise pattern

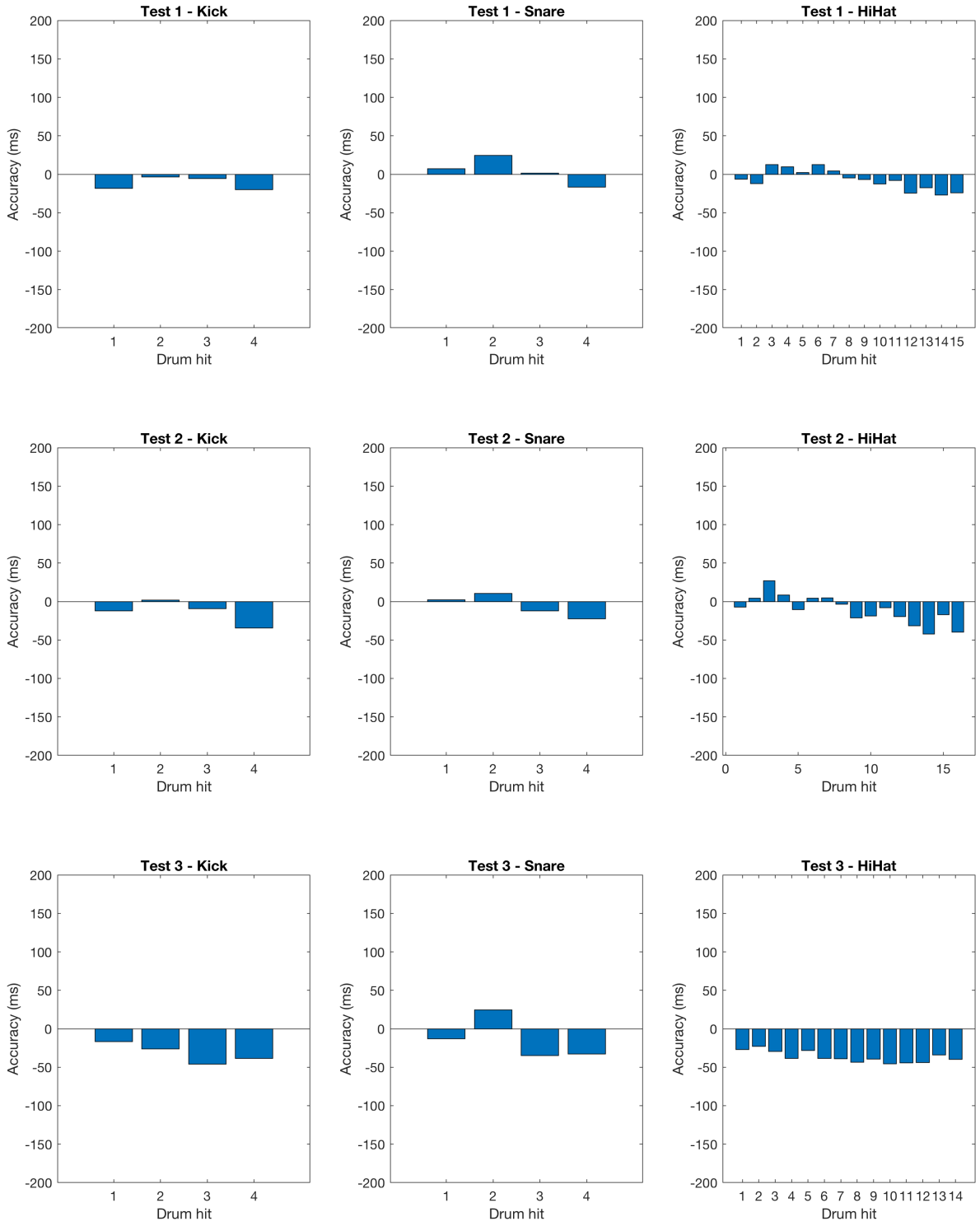


Figure E.15: Onset accuracy for test participant MP with timing feedback performing the simplified exercise pattern

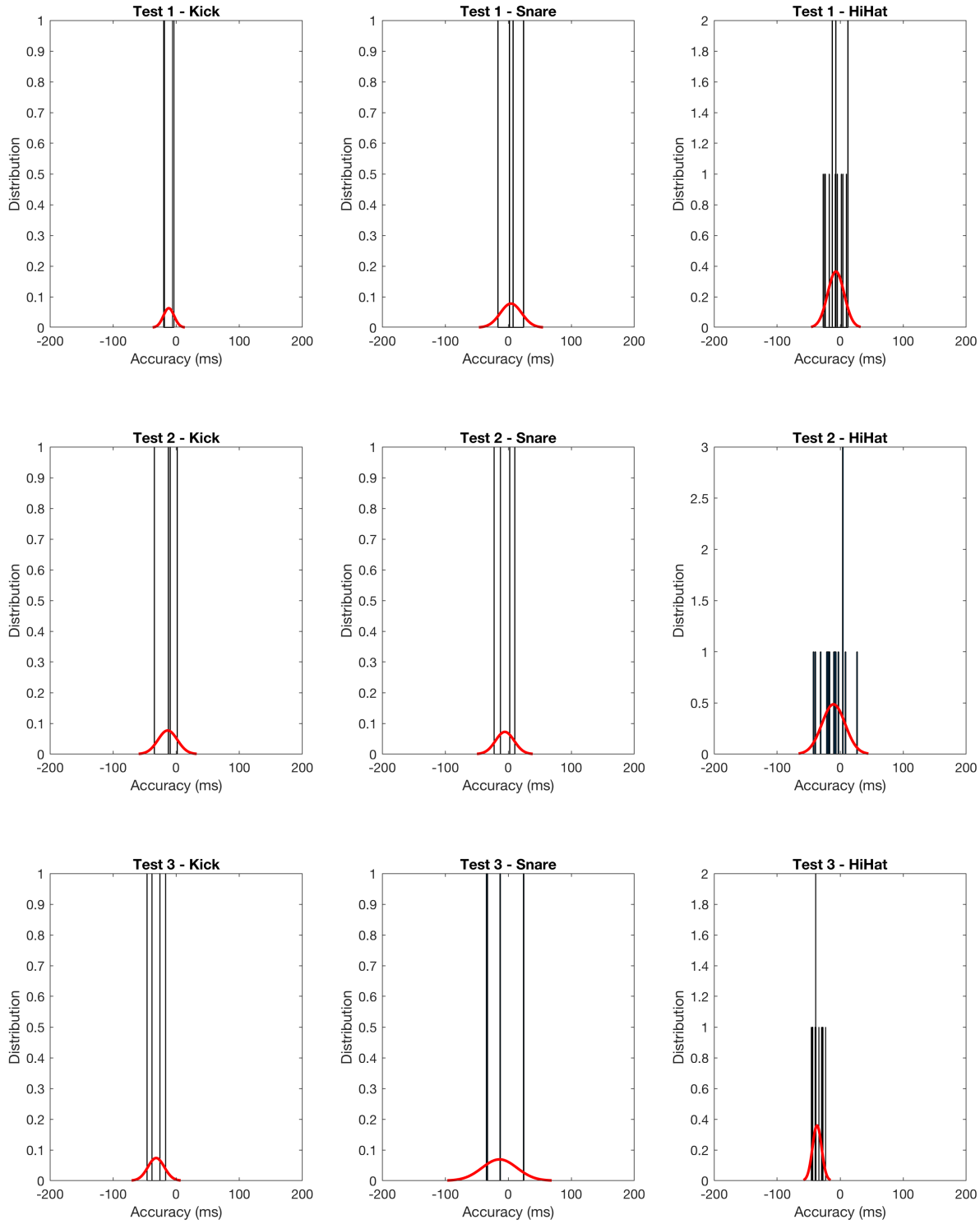


Figure E.16: Onset distribution for test participant MP with timing feedback performing the simplified exercise pattern

Appendix F

MATLAB Implementation of Automatic Threshold Function

```
function [gatedAudio] = autoThreshold(audio)

    % Baldur Kampmann
    % SMC Master Thesis 2020

    monoAudio = mean(audio,2);
    rectAudio = abs(monoAudio);
    slowAvg = movmean(rectAudio, 5000);
    fastAvg = movmean(rectAudio, 500);

    diff = fastAvg-slowAvg;
    offset = 0.2; % lift threshold a certain amount above diff value

    threshold = movmean(diff+offset, 500);

    gate = rectAudio>threshold;
    gatedAudio = rectAudio.*gate;

end
```

Appendix G

MATLAB Implementation of Onset-To-Peak Lag Calculation Function

```
1
2  function onset2peakLagMs = calcOnset2peakLag(audio, smoothingCoeff)
3
4  %CALCONSET2PEAKLAG Trim audio to start at onset location, smooth audio to
5  %get envelope and get onset-to-peak lag by finding the location on the peak
6
7      % Find samples > 1% of peak value
8      trimIndex = find(abs(audio) > (max(abs(audio))*1)/100);
9
10     % Trim beginning and end of audio to remove samples below 1% of peak
11     trimmedAudio = audio(trimIndex(1):length(audio));
12
13     % find onset-to-peak lag, which is simply the location of peak since
14     % the audio has trimmed to start at the onset location
15
16     % smooth audio
17     absAudio = abs(audio);
18     numSamples = length(absAudio);
19     w = smoothingCoeff;
20     envelope = zeros(numSamples, 1);
21
22     for i=1:numSamples
23         if (i < 2)
24             envelope(i) = expAvg(w,absAudio(i), envelope(1));
25         else
26             envelope(i) = expAvg(w,absAudio(i), envelope(i-1));
27         end
28     end
29
30     % find peak
31     [~, peakLocation] = max(envelope);
32     onset2peakLagMs = samples2ms(peakLocation);
33 end
```

Appendix H

MATLAB Implementation of Onset Matching Function

```
function [error] = matchOnsets(audioOnsets,midiOnsets, removeOutliers,
    removeDuplicates, addMissing)
%MATCHONSETS Calculate Euclidean distance from each detected onset to each
% ground truth onset.

% Optionally remove duplicate onsets and outliers as well
% as interpolate missing onsets.

    if nargin < 5
        addMissing = false;
    end

    % clear variables
    absDistMatrix = [];
    distMatrix = [];
    absDiff = [];
    absDiffLocation = [];

    % calculate distance
    for j = 1:length(audioOnsets)
        for k = 1: length(midiOnsets)
            absDistMatrix(k,j) = abs(audioOnsets(j) - midiOnsets(k));
            distMatrix(k,j) = audioOnsets(j) - midiOnsets(k);
        end
    end

    % find minimum distance using absolute values
    [ absDiff, absDiffLocation ] = (min(absDistMatrix));

    % remove duplicate onsets
    if removeDuplicates == true
        numDetectedOnsets = length(absDiffLocation);
        h = 0;
```

```
33
34          % loop
35          while h < numDetectedOnsets-1
36
37              % Update counter
38              h = h + 1;
39
40               % look for two detected onsets pointing to the same % reference
    onset
41              if absDiffLocation(h) == absDiffLocation(h+1)
42
43                  % find index of the onset with shortest distance to the
    reference
44                  [~, minIndex] = min([absDiff(h) absDiff(h+1)]);
45
46                  % if first onset is closer, remove the following
47                  if minIndex == 1
48                      absDiffLocation(h+1) = [];
49                      absDiff(h+1) = [];
50                      numDetectedOnsets = numDetectedOnsets - 1; % adjust loop
    length after removing entry
51                      distMatrix(:,h+1) = [];
52                      h = h - 1; % ensure that the last index is included
53
54                  % if second onset is closer, remove the previous
55                  else
56                      absDiffLocation(h) = [];
57                      absDiff(h) = [];
58                      numDetectedOnsets = numDetectedOnsets - 1; % adjust loop
    length after removing entry
59                      distMatrix(:,h) = [];
60                      h = h - 1; % ensure that the last index is included
61                  end
62              end
63          end
64      end
65
66      %%
67
68      % use absolute values as index into non-absolute matrix to get
69      % minimum distance from each detected onset to each ground truth
70      % onset regardsless of whether the hit is early or late
71      error = [];
72      for l = 1:length(absDiffLocation)
73          error(l) = distMatrix(absDiffLocation(l),l); % TODO: exceeds diffMatrix
    array bounds occasionally if duration in bars < 2.
74      end
75
```

```
76      % remove outliers
77      if removeOutliers == true
78          errorLimit = tempo2samples(100,8); % 8-note
79          error(absDiff > errorLimit) = [];
80
81          % update absDiff
82
83          absDiffLocation(absDiff > errorLimit) = [];
84          absDiff(absDiff > errorLimit) = [];
85      end
86
87       % interpolate missing onsets
88      if addMissing == true
89          if length(midiOnsets) ~= length(absDiff)
90              for i=1:length(midiOnsets)
91                  if sum(absDiffLocation == i) == 0
92                      % make room for missing value
93                      temp(1:i-1) = error(1:i-1)
94                      temp(i+1:length(error)+1) = error(i:length(error))
95                      % interpolate
96
97
98                      if i > 1 && i < length(temp)
99                          temp(i) = round(mean([temp(i-1) temp(i+1)]));
100                     elseif i == 1 % handle start of vector
101                         temp(i) = round(mean([temp(i+1) temp(i+2)]));
102                     else % handle end of vector
103                         temp(i) = round(mean([temp(i-2) temp(i-1)]));
104                     end
105                     error = temp;
106                 end
107             end
108         end
109     end
110
111 end
```

Appendix I

NMF Toolbox Modifications

The NMF Toolbox was modified to support templates with T number of frames in order to facilitate the temporal aspect of the NMFD algorithm.

First, the original *dictW.mat* file, which contains the dictionary, was replaced with *dictWNFMD.mat*, which supports a *MxT* matrix for each component, whereas the original *dictW.mat* only utilises a *Mx1* vector.

The *setupDictionary.m* file, which contains code that generates templates from audio samples, was then modified to store the generated MxT templates in the new dictionary file instead.

Finally, the NMFD implementation was modified by adding an additional case to *initTemplates.m* as seen in the code below.

This loads the new dictionary from the *dictWNMFD.mat* file as seen in line 3. In line 8 the initialization values are copied from the dictionary one frame at time in a for loop.

The original code, which has been commented out in line 9, shows that the implementation by López Serrano et al. fades out a single template over time by multiplying it with a coefficient to get the number of frames specified in *numTemplateFrames*.

```
1
2  case 'nmfd' % use correct NMFD template format wrt. value of T
3      s = load('../data/dictWNMFD.mat');
4
5      % sanity check
6      if parameter.numBins == size(s.dictionary{1},1)
7        for k = 1:size(s.dictionary,2)
8          initW{k} = s.dictionary{k};
9          %initW{k} = s.dictW(:,k)*linspace(1,0.1,parameter.numTemplateFrames);
10       end
11     end
```

Appendix J

System Evaluation MATLAB Script

```
1
2  % Baldur Kampmann
3  % SMC Master Thesis 2020
4
5  % System Evaluation Script
6
7  % Performs automated testing of system performance. By linking test
8  % variables to iteration index, a number of parameters can be tested and
9  % compared by changing the testMode variable. See the switch case in the
10 % code for test options.
11
12 % Source separation is performed using the NMFD algorithm by Smaragdis
13 % implemented in the NMF Toolbox by Dittmar et al.
14
15 % ********************************************************************** %
16
17 % Additional user settings are available inside the called functions.
18
19 % Requires the following 3rd party MATLAB toolboxes:
20 % - NMF Toolbox by Lopez-Serrano et al.
21 % - MIDI Toolbox by Eerola & Toiviainen
22 % - MIR Toolbox by Lartillot et al.
23
24 % References:
25 %     Patricio Lopez-Serrano, Christian Dittmar, Yigitcan Ozer, and Meinard
26 %     Muller
27 %     NMF Toolbox: Music Processing Applications of Nonnegative Matrix
28 %     Factorization
29 %     In Proceedings of the International Conference on Digital Audio Effects
30 %     (DAFx), 2019.
31
32 %     Eerola, T. & Toiviainen, P. (2004).
33 %     MIDI Toolbox: MATLAB Tools for Music Research.
34 %     University of Jyvaskyla: Kopijyva, Jyvaskyla, Finland.
35
36 %     Lartillot O., Toiviainen P., Eerola T. (2008)
```

```
%     A Matlab Toolbox for Music Information Retrieval.
%     In: Preisach C., Burkhardt H., Schmidt-Thieme L., Decker R. (eds)
%     Data Analysis, Machine Learning and Applications. Studies in
%     Classification, Data Analysis, and Knowledge Organization.
%     Springer, Berlin, Heidelberg

% ********************************************************************* %

clear variables;
close all;
clc;

%% Setup

% Test options:  'drumkit', 'tempo', 'pattern', 'velocity', 'combinations',
%                'combinations-staggered', 'inter-limb'

% select test variable
testMode = 'velocity';

% Test options:     'drumkit', 'tempo', 'pattern', 'velocity', 'combinations',
%                   'combinations-staggered', 'inter-limb'

% test iterations
numTests = 3;
numComponents = 3;

% Drum names
componentNames{1} = 'Bass Drum';
componentNames{2} = 'Snare Drum';
componentNames{3} = 'HiHat';

% Original MIDI pattern
origMidiFileName = '../data/patterns/exercisePattern.mid';

% MIDI data filenames
selMidiFileName{1} = '../data/patterns/Simplified.mid';
selMidiFileName{2} = '../data/patterns/Shuffled.mid';
selMidiFileName{3} = '../data/patterns/Combinations.mid';
selMidiFileName{4} = '../data/patterns/twolimbs.mid';

% create figures
figure(1)
figure(2)

% system settings
outPath = '../output/';
saveAudio = false;
```

```
onsetMismatch = false;
sysTest = true;
numAttempts = 1;
plotPattern = false;
timingFeedback = false;
playAudio = false;
normalize = true;
selPattern = 1;


drumkit = 1;
userTempo = 100; % 1/4-notes at 600BPM = IOI of 100ms

%% Init variables


set(0,'DefaultTextInterpreter','none') % remove Tex interpretation
patternMidiFileName = selMidiFileName{selPattern};

maxRange = 0;
rangeLimit = 2.5; % +-2.5ms - maximum range before test fails
compareToUnmodifiedMIDI = false;

shifts = [1, 2.5, 5];

%% % Init test-specific settings before loop
switch testMode

    case 'velocity'
         tempo = 75;
         drumkit = 1;
         patternMidiFileName = '../data/patterns/twolimbs.mid';

    case 'tempo'
         tempo = 75; % init value - not used in test
         drumkit = 2;
         selPattern = 1;
    case 'combinations'

        tempo = 75;
        drumkit = 1;
    case 'combinations-staggered'

       tempo = 75;
       drumkit = 1;
       compareToUnmodifiedMIDI = true;

    case 'inter-limb'
```

```
133         tempo = 75;
134         drumkit = 1;
135         timingFeedback = true;
136         compareToUnmodifiedMIDI = true;
137 
138     otherwise
139         tempo = userTempo;
140 end
141 
142 %% Analyze and extract data
143 
144 % train dictionary
145 onset2peakLag = setupDictionary(sysTest, drumkit);
146 
147 % store onset-to-peak lag for each drum
148 componentLag{1} = onset2peakLag{1}
149 componentLag{2} = onset2peakLag{2}
150 componentLag{3} = onset2peakLag{3}
151 
152 % extract MIDI data
153 [midiData numComponents] = loadExerciseMIDI(patternMidiFileName, tempo,
        plotPattern);
154 
155 % store unmodified MIDI data for testMode=velocity
156 origMidiData = midiData;
157 
158 %% Iterate through test variables
159 for i=1:numTests
160 
161         % avoid figures being overwritten
162         if timingFeedback == true
163             figure(2+i)
164         end
165 
166         % update parameters programmatically depening on test mode
167         switch testMode
168             case 'drumkit' % update which audio samples are used
169 
170                 drumkit = i-1;
171 
172                 testDesc = 'Drumkit';
173                 testValue = ['Drumkit ', num2str(drumkit)];
174 
175                 % update dictionary
176                 onset2peakLag = setupDictionary(sysTest, drumkit);
177 
178                 % update onset-to-peak lag
179                 componentLag{1} = onset2peakLag{1};
```

```
180                 componentLag{2} = onset2peakLag{2};
181                 componentLag{3} = onset2peakLag{3};
182                 disp(componentLag);
183 
184             case 'pattern'
185                 selPattern = i;
186                 patternMidiFileName = selMidiFileName{selPattern};
187                 [midiData numComponents] = loadExerciseMIDI(patternMidiFileName,
    tempo, plotPattern);
188                 testDesc = 'Pattern';
189                 testValue = num2str(selPattern);
190 
191             case 'tempo'
192                 tempo = 60*i;
193                 [midiData numComponents] = loadExerciseMIDI(patternMidiFileName,
    tempo, plotPattern);
194                 testDesc = 'Tempo';
195                 testValue = [num2str(tempo), ' BPM'];
196 
197             case 'velocity'
198                 component = 3; % hi-hat velocity is modified
199                 normalize = false;
200                 coeff = ((1/numTests)*i);
201                 midiData = modMIDIVelocity(origMidiData, component, coeff);
202                 testDesc = 'HH Velocity';
203                 testValue = num2str(round(coeff,2));
204 
205             case 'combinations' % test ability to detect onsets when varying IOI
206 
207                 patternMidiFileName = '../data/patterns/comb_Accuracy.mid';
208 
209                 testDesc = 'Increasing IOI';
210                 testValue = 'N/A';
211                 [midiData numComponents] = loadExerciseMIDI(patternMidiFileName,
    tempo, plotPattern);
212 
213             case 'combinations-staggered' % test ability to detect onsets when
    snare has lag of 64th-note (50ms@75BPM)
214 
215                 patternMidiFileName = '../data/patterns/comb_Accuracy.mid';
216                 [origMidiData, ~] = loadExerciseMIDI(patternMidiFileName, tempo,
    plotPattern);
217 
218                 testDesc = 'Increasing IOI (SD +50ms)';
219                 testValue = 'N/A';
220                 exp_patternMidiFileName = '../data/patterns/
    comb_Accuracy_Staggered.mid';
221                 [midiData numComponents] = loadExerciseMIDI(
```

```
    exp_patternMidiFileName, tempo, plotPattern);
222
223             case 'inter-limb' % test ability to detect inter-limb error with 16
    th-notes and longer
224
225                 patternMidiFileName = '../data/patterns/twolimbs.mid';
226                 simAsync = shifts(i) / 1000; % (sec to ms)
227
228                 testDesc = 'Inter-Limb Asynchrony';
229                 testValue = [num2str(shifts(i)), 'ms'];
230                 [midiData numComponents] = loadExerciseMIDI(patternMidiFileName,
     tempo, plotPattern);
231
232                 % store unmodified MIDI data
233                 origMidiData = midiData;
234
235                 % shift BD backwards in time. Leave first and last hit intact
    for comparison
236                 midiData{1}(2:3,6) = midiData{1}(2:3,6) + simAsync;
237
238                 % shift SD forwards in time. Leave first and last hit intact for
     comparison
239                 midiData{2}(2:3,6) = midiData{2}(2:3,6) - simAsync;
240
241             otherwise
242                 disp('Unknown test')
243                 break;
244         end
245
246
247         % prepare text
248         testName = ['Test Variable: ',testDesc];
249         testValue = ['Test Value: ', testValue];
250
251         % convert MIDI to audio
252         [audio, fs] = midi2Audio(midiData, drumkit, tempo, normalize);
253
254         % play back audio
255         if playAudio == true
256             sound(audio, fs);
257         end
258
259         if saveAudio == true
260             audiowrite([outPath,datestr(now), ' - Accuracy_Test.wav'],audio,fs);
261         end
262
263         % perform analysis
264         audioOnsets = performAnalysis(timingFeedback, numAttempts, componentLag,
```

```
    patternMidiFileName, sysTest, audio, fs, tempo, numComponents);
265
266         % for each drum
267         for h=1:numComponents
268             % convert onsets to ms
269             audioOnsets{h} = samples2ms(audioOnsets{h}, fs);
270
271             if compareToUnmodifiedMIDI == true
272                 midiOnsets{h} = origMidiData{h}(:,6) * 1000';
273             else
274                  midiOnsets{h} = midiData{h}(:,6) * 1000';
275             end
276
277             % detect mismatch between number of onsets and pattern
278             if length(audioOnsets{h}) > length(midiOnsets{h})
279                 excessOnsets = length(audioOnsets{h}) - length(midiOnsets{h});
280                 disp([newline num2str(length(audioOnsets{h})), ' onsets detected
    . Correct number is ', num2str(length(midiOnsets{h})), '. Test halted for
    component ', num2str(h)])
281                 onsetMismatch = true;
282                 break;
283
284             elseif length(audioOnsets{h}) < length(midiOnsets{h})
285                  missingOnsets = length(midiOnsets{h}) - length(audioOnsets{h});
286                  disp([newline num2str(length(audioOnsets{h})), ' onsets
    detected. Correct number is ', num2str(length(midiOnsets{h})), '. Test
    halted for component ', num2str(h)])
287                  onsetMismatch = true;
288                  break;
289
290             else % if numbers match, calculate timing error
291
292                 onsetMismatch = false;
293
294                 error{i}{h} = audioOnsets{h} - midiOnsets{h}';
295
296                 % set up subplot
297                 figure(1)
298                 subplot(numTests,numComponents,h+((i-1)*3))
299                 title('System Test Results')
300
301                 % plot bars
302                 bar(error{i}{h})
303
304                 ylim([-5 5]);
305                 ylabel('Accuracy (ms)');
306                 xlabel('Drum hit');
307                 title({['Test ', num2str(i)], testName, testValue, ['Tempo: ',
```

```
    num2str(tempo), ' BPM'], ['Drum-kit ', num2str(drumkit), ' - ',
    componentNames{h}], ['Pattern: ', extractAfter(patternMidiFileName,'patterns
    /')]})
308                 %sgtitle('Onset Detection Consistency');
309                 fig1 = gcf;
310 
311                 % calculate diff between smallest and largest error to determine
     range,
312                 % i.e. the consistency of the algorithm itself. Eliminates
    variations caused
313                 % by differing drum sample start trimming and similar factors
314                 range{i}(h) = max(error{i}{h}) - min(error{i}{h});
315                 maxRange(i) = max(range{i});
316             end
317         end
318 
319         if onsetMismatch == true
320             break; % skip plotting if onset count is incorrect
321         else
322             % plot range
323             figure(2);
324             subplot(3,1,i);
325             bar(range{i})
326 
327             % plot values above bars
328             text(1:length(range{i}),range{i},num2str(round(range{i},4)'),'vert',
    'bottom','horiz','center');
329             box off
330 
331             ylim([-5 5]);
332             yticks([-5:1:5])
333             xticklabels({'Kick', 'Snare', 'Hihat'})
334             ylabel('Accuracy Range (ms)');
335             title({['Test ', num2str(i)], testName, ['Tempo: ', num2str(tempo),
    ' BPM'], ['Drum-kit ', num2str(drumkit)], ['Pattern: ', extractAfter(
    patternMidiFileName,'patterns/')]});
336 
337             %sgtitle('Asynchrony Range');
338             fig2 = gcf;
339         end
340 end
341 
342 %% scale and position windows
343 fig1.Units = 'normalized';
344 fig1.OuterPosition = [0.05 0.3 0.35 0.9];
345 %fig1.OuterPosition = [0.05 0.3 0.15 0.36]; % Single
346 
347 fig2.Units = 'normalized';
```

```
348 fig2.OuterPosition = [0.5 0.3, 0.15 0.5];
349
350 %% Test success/fail handling
351
352 % Onset count test
353 if onsetMismatch == true
354     if tempo >= 600
355         disp(newline, 'ONSET COUNT TEST = SUCCESS: IOI goal of 100ms
    successfully attained.')
356     else
357         disp([newline, 'ONSET COUNT TEST = FAIL: Onset mismatch at current IOI
    of ', num2str(tempo2ms(tempo)), ' ms']);
358     end
359 else
360     disp([newline, 'ONSET COUNT TEST = SUCCESS: Correct number of onsets
    detected'])
361 end
362
363 % Range test
364 finalRange = max(maxRange);
365 if finalRange > rangeLimit
366     disp([newline 'RANGE TEST = FAIL: Range value ', num2str(finalRange), ' ms
    exceeds limit at tempo ', num2str(tempo), ' BPM']);
367 else
368     disp([newline 'RANGE TEST = SUCCESS: Range value ', num2str(finalRange), '
    ms is within limit']);
369 end
```

Appendix K

User Evaluation MATLAB Script

```
1
2
3  % Baldur Kampmann
4  % SMC Master Thesis 2020
5
6  % USER EVALUATION SCRIPT
7
8  % Automates the user evaluation process by first recording samples of the
9  % user's drumkit to set up a NMFD template dictionary.
10 % Next, the user is asked to perform exercises without timing feedback.
11 % Finally, the exercises are repeated but this time with feedback.
12
13 % Behind the scenes, the user performance is recorded as audio, source
14 % separated and analysed for timing accuracy by comparing the user's
15 % performance ground truth onset timing.
16
17 % Source separation is performed using the NMFD algorithm by Smaragdis
18 % implemented in the NMF Toolbox by Dittmar et al.
19
20 % ********************************************************************* %
21
22 % Additional user settings are available inside the called functions.
23
24 % Requires the following 3rd party MATLAB toolboxes:
25 % - NMF Toolbox by Lopez-Serrano et al.
26 % - MIDI Toolbox by Eerola & Toiviainen
27 % - MIR Toolbox by Lartillot et al.
28
29 % References:
30 %     Patricio Lopez-Serrano, Christian Dittmar, Yigitcan Ozer, and Meinard
31 %     Muller
32 %     NMF Toolbox: Music Processing Applications of Nonnegative Matrix
33 %     Factorization
34 %     In Proceedings of the International Conference on Digital Audio Effects
35 %     (DAFx), 2019.
36
```

```
%     Eerola, T. & Toiviainen, P. (2004).
%     MIDI Toolbox: MATLAB Tools for Music Research.
%     University of Jyvaskyla: Kopijyva, Jyvaskyla, Finland.

%     Lartillot O., Toiviainen P., Eerola T. (2008)
%     A Matlab Toolbox for Music Information Retrieval.
%     In: Preisach C., Burkhardt H., Schmidt-Thieme L., Decker R. (eds)
%     Data Analysis, Machine Learning and Applications. Studies in
%     Classification, Data Analysis, and Knowledge Organization.
%     Springer, Berlin, Heidelberg

% ************************************************************************ %

clear variables;
close all;
clc;

% Settings
    numAttempts = 5 % exercise iterations

% Setup dictionary

    disp('Welcome')
    disp([newline, 'Press any key to begin setup'])
    pause;

    onset2PeakLag = setupDictionary();

% Perform analysis

    % without timing feedback
    disp([newline, 'Setup complete'])
    disp([newline,'Press any key to start the exercise'])
    pause;
    clc;

    enableFeedback = false;
    performAnalysis(enableFeedback, numAttempts, onset2PeakLag);

    % with timing feedback
    clc;
    disp([newline, 'You will now be shown a timing analysis of your performance
    after each exercise'])
    disp([newline, 'Please take some time to examine the timing of each drum and
     see if you can improve your accuracy'])
    disp([newline, 'Press any key to start the exercise'])
    pause;
    enableFeedback = true;
```

```
83      performAnalysis(enableFeedback, numAttempts, onset2PeakLag);
84
85      %
86      clc;
87      disp([newline, 'Thank you for participating']);
88      disp([newline, 'Please fill out the questionnaire before leaving'])
```

Appendix L

IDMT-Drums Dataset Test MATLAB Script

```
1
2 % Requires the following 3rd party MATLAB toolboxes:
3 % - NMF Toolbox by Lopez-Serrano et al.
4 % - MIDI Toolbox by Eerola & Toiviainen
5 % - MIR Toolbox by Lartillot et al.
6
7 % References:
8 %     Patricio Lopez-Serrano, Christian Dittmar, Yigitcan Ozer, and Meinard
9 %     Muller
10 %     NMF Toolbox: Music Processing Applications of Nonnegative Matrix
11 %     Factorization
12 %     In Proceedings of the International Conference on Digital Audio Effects
13 %     (DAFx), 2019.
14
15 %     Eerola, T. & Toiviainen, P. (2004).
16 %     MIDI Toolbox: MATLAB Tools for Music Research.
17 %     University of Jyvaskyla: Kopijyva, Jyvaskyla, Finland.
18
19 %     Lartillot O., Toiviainen P., Eerola T. (2008)
20 %     A Matlab Toolbox for Music Information Retrieval.
21 %     In: Preisach C., Burkhardt H., Schmidt-Thieme L., Decker R. (eds)
22 %     Data Analysis, Machine Learning and Applications. Studies in
23 %     Classification, Data Analysis, and Knowledge Organization.
24 %     Springer, Berlin, Heidelberg
25
26 % ********************************************************************* %
27
28 clear variables;
29 close all;
30 clc;
31
32 %% Setup
33
34 % select data subset
35 set='TD'; % see avaiable cases below
36
```

```
% Debug settings
saveSampleAudio = false;
saveOnset2PeakLag = false
saveMirEvalData = true;

% Plot settings
plotting = false
timingFeedback = false;
plotRange = false;

%% System settings

numComponents = 3;

% Drum names
componentNames{1} = 'Bass Drum';
componentNames{2} = 'Snare Drum';
componentNames{3} = 'HiHat';

% create figures

if plotting == true
figure(1)
figure(2)
figure(3)
end

% system settings
onsetMismatch = false;
sysTest = true;
normalize = true;

%% Init variables

maxRange = 0;
rangeLimit = 25; % F-Measure in Mir_Eval is usually done with 50ms

% create folders
outPath = ['../data/mir_eval/', datestr(now,30), '/'];
if ~exist(outPath, 'dir')
     mkdir(outPath);
end

% Folder for segmented samples
segmentedPath = [outPath, 'SegmentedSamples/'];
if ~exist(segmentedPath, 'dir')
     mkdir(segmentedPath);
end
```

```
85
86 % Folder for trimmed samples
87 trimmedPath = [outPath, 'TrimmedSamples/'];
88 if ~exist(trimmedPath, 'dir')
89      mkdir(trimmedPath);
90 end
91
92 %% Analyze and extract data
93
94 % Test-specific variables
95 offset = 1;
96 suffix{1} = '#KD#train'
97 suffix{2} = '#SD#train'
98 suffix{3} = '#HH#train'
99
100 switch set
101     case 'RD'
102         numTests = 14;
103         testName = 'RealDrum01_'
104
105     case 'WD1'
106         numTests = 10;
107         testName = 'WaveDrum01_'
108
109   case 'WD2'
110         numTests = 60;
111         testName = 'WaveDrum02_'
112         offset = 0;
113         suffix{1} = '#KD';
114         suffix{2} = '#SD';
115         suffix{3} = '#HH';
116
117     case 'TD'
118         numTests = 10;
119         testName = 'TechnoDrum01_'
120 end
121
122 for i=1:numTests
123
124     % match IDMT count format
125     test = i-offset;
126
127     % update filenames
128     sampleFn{1} = ['../data/IDMT/', testName, num2str(test, '%02d'), suffix{1},
    '.wav']; %'#KD.wav']; % '#KD#train.wav']
129     sampleFn{2} = ['../data/IDMT/', testName, num2str(test, '%02d'), suffix{2},
    '.wav']; %'#SD.wav']; %'#SD#train.wav'];
130     sampleFn{3} = ['../data/IDMT/', testName, num2str(test, '%02d'), suffix{3},'
```

```
    .wav']; %'#HH.wav']; % '#HH#train.wav'];
131
132     mixFn = ['../data/IDMT/', testName, num2str(test, '%02d'), '#MIX.wav'];
133
134     refOnsetFn{1} = ['../data/IDMT/', testName, num2str(test, '%02d'), '#KD.svl'
        ];
135     refOnsetFn{2} = ['../data/IDMT/', testName, num2str(test, '%02d'), '#SD.svl'
        ];
136     refOnsetFn{3} = ['../data/IDMT/', testName, num2str(test, '%02d'), '#HH.svl'
        ];
137
138     for j=1:numComponents
139
140         [sampleData,sampleFs] = audioread(sampleFn{j});
141
142         mirSample = mirsegment(sampleData, 'RMS');
143
144         index = get(mirSample,'FramePos')
145
146         % Use 2nd sample for training
147         indexSamples = ms2samples(index{1}{2}*1000);
148
149         indexBegin = indexSamples(1);
150         indexEnd = indexSamples(2);
151
152         if indexBegin == 0
153             indexBegin = 1;
154         end
155
156         segmentedSample = sampleData(indexBegin:indexEnd);
157
158         % Trim beginning and end to 1% below peak value
159         % trimIndex = find(abs(segmentedSample) > (max(sampleData)*1)/100); % 1%
         percent below peak
160         trimIndex = find(abs(segmentedSample) > (max(segmentedSample)*1)/100); %
         1% percent below peak
161
162         trimmedSample{j} = segmentedSample(trimIndex(1):trimIndex(length(
        trimIndex)));
163
164         % find onset-to-peak lag
165         [~,onset2peakLag{j}] = max(abs(trimmedSample{j}));
166         onset2peakLagMs{j} = samples2ms(onset2peakLag{j})
167
168         %figure
169         %plot(trimmedSample{j});
170
171         if saveSampleAudio == true
```

```
172             % save samples for debugging
173             audiowrite([segmentedPath, testName, num2str(test, '%02d'), suffix{j
    }, ' - segmented', '.wav'],segmentedSample,sampleFs);
174             audiowrite([trimmedPath, testName, num2str(test, '%02d'), suffix{j},
     ' - trimmed', '.wav'], trimmedSample{j},sampleFs);
175         end
176     end
177
178
179     if saveOnset2PeakLag == true
180         % write onset-to-peak lag to text file in mir_eval format
181         o2pFn = [trimmedPath, testName, num2str(test, '%02d'), '#onset2peak.txt'
    ];
182
183         fileID = fopen(o2pFn,'w');
184         fprintf(fileID,'%.6f %.6f %.6f\n', onset2peakLagMs{1}, onset2peakLagMs
    {2}, onset2peakLagMs{3});
185         fclose(fileID);
186     end
187
188     % train dictionary
189     setupDictionary(sysTest, 1, 3, '', trimmedSample, true);
190
191
192     for j=1:numComponents
193       refOnsets{j} = parseSVLAnnotations(refOnsetFn{j});
194     end
195
196     % Drum peak position lag correction in ms
197     componentLag{1} = onset2peakLagMs{1}
198     componentLag{2} = onset2peakLagMs{2}
199     componentLag{3} = onset2peakLagMs{3}
200
201     % load mixture
202     [audio,fs] = audioread(mixFn);
203
204     if timingFeedback == true
205         figure(2+i)
206     end
207
208     % perform analysis
209     audioOnsets = [];
210     audioOnsets = idmtAnalysis(timingFeedback, sysTest, audio, fs, numComponents
    ,componentLag);
211
212     % plot onsets
213     % plotResults(componentName,vizEnd, detectedOnsets, iteration, patternOnsets
    , errorMs, audio, preCountOffset, prePadding, plotWaveform, overlayOnsets)
```

```
214
215         % for each drum
216     for h=1:numComponents
217         % convert onsets to ms
218         audioOnsets{h} = samples2ms(audioOnsets{h}, fs);
219         midiOnsets{h} = [];
220         midiOnsets{h} = refOnsets{h} *1000; % midiData{h}(:,6) * 1000';
221
222         % detect mismatch between number of onsets and pattern
223         if length(audioOnsets{h}) > length(midiOnsets{h})
224             excessOnsets = length(audioOnsets{h}) - length(midiOnsets{h});
225             disp([newline num2str(length(audioOnsets{h})), ' onsets detected.
    Correct number is ', num2str(length(midiOnsets{h})), '. Test halted for
    component ', num2str(h)])
226             onsetMismatch = true;
227             %break;
228
229         elseif length(audioOnsets{h}) < length(midiOnsets{h})
230             missingOnsets = length(midiOnsets{h}) - length(audioOnsets{h});
231             disp([newline num2str(length(audioOnsets{h})), ' onsets detected.
    Correct number is ', num2str(length(midiOnsets{h})), '. Test halted for
    component ', num2str(h)])
232             onsetMismatch = true;
233             %break;
234         end
235
236      % if numbers match, calculate timing error
237         if onsetMismatch == true
238             error{i}{h} = 0;
239             onsetMismatch = false;
240         else
241             error{i}{h} = audioOnsets{h} - midiOnsets{h};
242         end
243
244         % convert to samples
245         audioOnsetsSamples{h} = ms2samples(audioOnsets{h});
246         midiOnsetsSamples{h} = ms2samples(midiOnsets{h});
247
248         if saveMirEvalData == true
249             % write to text file in mir_eval format
250             estFn{1} = [outPath, testName, num2str(test, '%02d'), '#KD_est.txt'
    ];
251             estFn{2} = [outPath, testName, num2str(test, '%02d'), '#SD_est.txt'
    ];
252             estFn{3} = [outPath, testName, num2str(test, '%02d'), '#HH_est.txt'
    ];
253
254             refFn{1} = [outPath, testName, num2str(test, '%02d'), '#KD_ref.txt'
```

```
    ];
255             refFn{2} = [outPath, testName, num2str(test, '%02d'), '#SD_ref.txt'
    ];
256             refFn{3} = [outPath, testName, num2str(test, '%02d'), '#HH_ref.txt'
    ];
257 
258             fileID = fopen(estFn{h},'w');
259             fprintf(fileID,'%.15f\n', audioOnsets{h}/1000);
260             fclose(fileID);
261 
262             fileID = fopen(refFn{h},'w');
263             fprintf(fileID,'%.15f\n', midiOnsets{h}/1000);
264             fclose(fileID);
265         end
266 
267         % plot
268         if plotting == true
269             figure(3)
270             plotResults(componentNames,length(audio), audioOnsetsSamples{h}, h,
    midiOnsetsSamples{h}, error{i}{h}, audio, 0, 0, true, audioOnsetsSamples{h})
271 
272             % set up subplot
273             figure(1)
274             subplot(numTests,numComponents,h+((i-1)*3))
275             title('System Test Results')
276 
277             % plot bars
278             bar(error{i}{h})
279             ylim([-50 50]);
280             ylabel('Accuracy (ms)');
281             xlabel('Drum hit');
282             title(componentNames{h})
283             sgtitle('Detected Onset Error ');
284             fig1 = gcf;
285         end
286     end
287 
288     if plotRange == true
289         if onsetMismatch == true
290             break; % skip plotting if onset count is incorrect
291         else
292             % plot range
293             figure(2);
294             subplot(3,1,i);
295             bar(range{i})
296 
297             % plot numbers above bars
298             text(1:length(range{i}),range{i},num2str(round(range{i},4)'),'vert',
```

```
    'bottom','horiz','center');
299             box off
300
301             ylim([-5 5]);
302             yticks([-5:1:5])
303             xticklabels({'Kick', 'Snare', 'Hihat'})
304             ylabel('Accuracy Range (ms)');
305             title({['Test ', num2str(i)]});
306
307             sgtitle('Baseline Algorithm Accuracy Measured as Range of Error');
308             fig2 = gcf;
309         end
310     end
311 end
312
313 %% scale and position windows
314 if plotting == true
315
316     fig1.Units = 'normalized';
317     fig1.OuterPosition = [0.05 0.3 0.45 0.7];
318
319     fig2.Units = 'normalized';
320     fig2.OuterPosition = [0.5 0.3, 0.45 0.7];
321
322     % wait for user to finish
323     pause
324 end
325
326 %% Test success/fail handling
327
328 % Onset count test
329 if onsetMismatch == true
330     disp([newline, 'ONSET COUNT TEST = FAIL: Incorrect number of onsets detected
    ']);
331 else
332     disp([newline, 'ONSET COUNT TEST = SUCCESS: Correct number of onsets
    detected'])
333 end
334
335 % Range test
336 finalRange = max(maxRange);
337 if finalRange > rangeLimit
338     disp([newline 'RANGE TEST = FAIL: Range value ', num2str(finalRange), ' ms
    exceeds limit at tempo ', num2str(tempo), ' BPM. Test halted']);
339 else
340     disp([newline 'RANGE TEST = SUCCESS: Range value ', num2str(finalRange), '
    ms is within limit']);
341 end
```

Appendix M

Mir_Eval Python Script

```
# Baldur Kampmann
# SMC Master Thesis 2020

# This script calculates F-measure with Mir_Eval and exports the results
# to a CSV file for subsequent analysis in Excel.

import mir_eval
import os
import csv

# Mir_eval.onset window size in ms (default=0.05)
windowSize  = 0.05

path="/data/mir_eval/"
os.chdir(path)

# Test settings
drumkit = ["RealDrum01_", "TechnoDrum01_", "WaveDrum01_", "WaveDrum02_"]

beginDrumkit = 0;
endDrumkit = len(drumkit);

endTest = [14, 10, 10, 61]
startTest= [0, 0, 0, 1]

numTests = [14, 10, 10, 60]

# CSV output setup
with open('F_measure.csv', mode='w') as fn:
    F_writer = csv.writer(fn, delimiter=',', quotechar='"', quoting=csv.
    QUOTE_MINIMAL)
    F_writer.writerow(['BD', 'SD', 'HH'])

    # Iterate through all drum-kits
    for h in range(beginDrumkit,endDrumkit):
        print('Drumkit', h)
```

```
36
37          # Iterate through all tests for current drumkit
38          for i in range(startTest[h], endTest[h]):
39
40              # Update variables
41              insert = drumkit[h]
42              num = '%02d' % (i)
43              print('Test #', num)
44
45              # BD
46              estimated_onsets = mir_eval.io.load_events(insert+ str(num) + '#
    KD_est.txt')
47              reference_onsets = mir_eval.io.load_events(insert + str(num) + '#
    KD_ref.txt')
48              BD_F , P, R = mir_eval.onset.f_measure(reference_onsets,
    estimated_onsets, window=windowSize)
49
50              # SD
51              estimated_onsets = mir_eval.io.load_events(insert + str(num) + '#
    SD_est.txt')
52              reference_onsets = mir_eval.io.load_events(insert + str(num) + '#
    SD_ref.txt')
53              SD_F, P, R = mir_eval.onset.f_measure(reference_onsets,
    estimated_onsets, window=windowSize)
54
55              # Hi-Hat
56              estimated_onsets = mir_eval.io.load_events(insert + str(num) + '#
    HH_est.txt')
57              reference_onsets = mir_eval.io.load_events(insert + str(num) + '#
    HH_ref.txt')
58              HH_F, P, R = mir_eval.onset.f_measure(reference_onsets,
    estimated_onsets, window=windowSize)
59
60              # Write F-meausure results to CSV file
61              F_writer.writerow([BD_F, SD_F, HH_F])
```

Appendix N

User Evaluation Data Analysis MATLAB Script

```
1
2 % Baldur Kampmann
3 % SMC Master Thesis 2020
4
5 % Performance Data Analysis Script
6
7 % Analyzes user evaluation performance data and handle descriptive and
    inferential
8 % statistics.
9
10 clear variables;
11 close all
12 clc;
13
14 %% Setup
15
16 % Enable or disable plots to speed up process
17 plot = true;
18
19 numTests = 3;
20 numDrums = 3;
21
22 fs = 44100;
23
24 % Post-processing options
25 removeOutliers = true;
26 removeDuplicates = true;
27 addMissing = true;
28
29 % Export options
30 exportExcel = false % export ANOVA data to excel file
31
32 % Unit test compares the ground truth data with it itself.
```

```
% Should result in zero in all plots.
unitTest = false;

% compare user data to ground truth
% Should show zero in plots of data with timing feedback
refTest = false;

% Drum names
componentNames{1} = 'Kick';
componentNames{2} = 'Snare';
componentNames{3} = 'HiHat';

% load MIDI data
patternMidiFilename = '../data/patterns/exercisePattern.mid';

% extract MIDI data
tempo = 100; % make sure to use same tempo as was used for the test
[midiData numComponents] = loadExerciseMIDI(patternMidiFilename, tempo, false);

% import previosly recorded onset data

% Original test, participant: M
%fnWithoutFB = '../data/onsets/M - Unguided.mat';
%fnWithFB ='../data/onsets/M - Guided.mat';

% Original test, participant: LK
fnWithoutFB = '../data/onsets/LK - Unguided.mat';
fnWithFB ='../data/onsets/LK - Guided.mat';

%userOnsets_WithFeedback = importUserOnsets('../data/onsets/refUser.mat');

% Original test, participant : MP
%fnWithoutFB = '../data/onsets/MP - Orig - Unguided.mat';
%fnWithFB = '../data/onsets/MP - Orig - Guided.mat';

% Revised test, participant : MP
%fnWithoutFB = '../data/onsets/MP - Simp - Unguided.mat';
%fnWithFB = '../data/onsets/MP - Simp - Guided.mat';

userOnsets_WithoutFeedback = importUserOnsets(fnWithoutFB);
userOnsets_WithFeedback = importUserOnsets(fnWithFB);

% create figures
if plot == true
    figure(1)
    figure(2)
    figure(3)
    figure(4)
```

```
end


%% Calculate statistics from unguided data

    % Calculate start and end of pattern in samples
    barSamples = tempo2samples(tempo,0.5); % 1 bar
    padding = tempo2samples(tempo,8); % 8-note padding
    rangeStart = -(padding);
    rangeEnd = barSamples + padding;

    barSamples = tempo2samples(tempo,0.5); % 1 bar
    padding = tempo2samples(tempo,8); % 8-note padding
    rangeStart = -(padding);
    rangeEnd = barSamples + padding;

for i=1:numTests
    for h=1:numDrums

        % remove onsets outside time range
        userOnsets_WithoutFeedback.combOnsets{i}{h} = trimOnsetRange(userOnsets_WithoutFeedback.combOnsets{i}{h}, rangeStart, rangeEnd);
        userOnsets_WithFeedback.combOnsets{i}{h} = trimOnsetRange(userOnsets_WithFeedback.combOnsets{i}{h}, rangeStart, rangeEnd);

        % calc number of detected onsets for statistics
        numDetectedOnset_WithoutFeedback{i}{h} = length(userOnsets_WithoutFeedback.combOnsets{i}{h});
        numDetectedOnset_WithFeedback{i}{h} = length(userOnsets_WithFeedback.combOnsets{i}{h});

        % convert reference onset to ms
        midiOnsets{h} = midiData{h}(:,6) * 1000';

        % convert ref onsets to samples
        midiOnsets{h} = ms2samples(midiOnsets{h});

        % extend pattern
        numBars = 2;
        midiOnsets{h} = extendPattern(numBars, midiOnsets{h}', tempo2samples(tempo,1), 0)';

        % calc number of reference onsets for statistics
        numRefOnsets{h} = size(midiOnsets{h},1);

        % calculate Euclidean distance from each detected onset to each ground truth onset
        if unitTest == true
```

```
123             async_withoutFeedback{i}{h} = matchOnsets(midiOnsets{h}', midiOnsets
    {h}', false, false);
124             async_withFeedback{i}{h} = matchOnsets(midiOnsets{h}', midiOnsets{h
    }', false, false);
125         elseif refTest == true
126             async_withoutFeedback{i}{h} = matchOnsets(userOnsets_WithoutFeedback
    .combOnsets{i}{h}, midiOnsets{h}', removeOutliers, removeDuplicates,
    addMissing);
127             async_withFeedback{i}{h} = matchOnsets(midiOnsets{h}'+(randn(length(
    midiOnsets{h}),1)*1000), midiOnsets{h}', false, false, false);
128         else
129             async_withoutFeedback{i}{h} = matchOnsets(userOnsets_WithoutFeedback
    .combOnsets{i}{h}, midiOnsets{h}', removeOutliers, removeDuplicates,
    addMissing);
130             async_withFeedback{i}{h} = matchOnsets(userOnsets_WithFeedback.
    combOnsets{i}{h}, midiOnsets{h}', removeOutliers, removeDuplicates,
    addMissing);
131         end
132
133         % convert to ms
134         async_withoutFeedback{i}{h} = samples2ms(async_withoutFeedback{i}{h});
135         async_withFeedback{i}{h} = samples2ms(async_withFeedback{i}{h});
136
137         % plot results
138         if plot == true
139             [plotBarsUnguided plotHistfitUnguided] = plotUDAnalysis(numTests,
    numDrums,h,i, async_withoutFeedback,componentNames, 'Without Timing Feedback
    ', 1);
140             [plotBarsGuided plotHistfitGuided] = plotUDAnalysis(numTests,
    numDrums,h,i, async_withFeedback,componentNames, 'With Timing Feedback', 3);
141         end
142
143         %% calculate means and standard dev
144         asyncMean_WithoutFeedback{i}{h} = mean(async_withoutFeedback{i}{h});
145         asyncStd_WithoutFeedback{i}{h} = std(async_withoutFeedback{i}{h});
146         asyncMean_WithFeedback{i}{h} = mean(async_withFeedback{i}{h});
147         asyncStd_WithFeedback{i}{h} = std(async_withFeedback{i}{h});
148     end
149 end
150
151 %% Scale and position figures
152
153 if plot == true
154
155     plotBarsUnguided.Units = 'normalized';
156     plotBarsUnguided.OuterPosition = [0.0 0.1 0.35 0.8];
157     plotBarsUnguided.Name = 'Without Feedback - Accuracy Per Hit';
158
```

```
159     plotHistfitUnguided.Units = 'normalized';
160     plotHistfitUnguided.OuterPosition = [0.0 0.5 0.35 0.8];
161     plotHistfitUnguided.Name = 'Without Feedback - Distribution';
162 
163     plotBarsGuided.Units = 'normalized';
164     plotBarsGuided.OuterPosition = [0.5 0.1 0.35 0.8];
165     plotBarsGuided.Name = 'With Feedback - Accuracy Per Hit';
166 
167     plotHistfitGuided.Units = 'normalized';
168     plotHistfitGuided.OuterPosition = [0.5 0.5 0.35 0.8];
169     plotHistfitGuided.Name = 'With Feedback - Distribution';
170 
171 end
172 
173 %% Check for improvement in timing as a result of training
174 
175 % Pool data from last three test iterations
176 for h=1:numDrums
177 
178     controlGroup{h} = [];
179     experimentalGroup{h} = [];
180 
181     for i=(numTests-2):numTests
182      controlGroup{h} = [controlGroup{h}; async_withoutFeedback{i}{h}'];
183      experimentalGroup{h} = [experimentalGroup{h}; async_withFeedback{i}{h}'];
184     end
185 end
186 
187 % Calculate stats
188 userdataStats(controlGroup, experimentalGroup);
189 rm_anova = userdataANOVA(controlGroup, experimentalGroup, 3, 3);
190 
191 if exportExcel == true
192     insert = extractBetween(fnWithoutFB, 'onsets/', ' -');
193     filename = ['../data/rm-anova - ', insert{1} ,'.xlsx'];
194     writetable(rm_anova{1},filename,'Sheet',1,'Range','B1')
195     writetable(rm_anova{2},filename,'Sheet',1,'Range','B5')
196     writetable(rm_anova{3},filename,'Sheet',1,'Range','B10')
197 end
```

Appendix O

MATLAB Profiler Results

Profile Summary

Generated 14-May-2020 12:39:16 using performance time.

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
systemEval	1	15.507 s	0.891 s	
NMFD	4	7.741 s	3.481 s	
setupDictionary	1	7.708 s	0.021 s	
performAnalysis	1	5.729 s	0.031 s	
highpass	3	3.684 s	0.003 s	
convModel	3016	2.461 s	2.151 s	
highpass>designFilter	3	2.417 s	0.004 s	
designfilt	3	2.392 s	0.007 s	
designfilt>parseAndDesignFilter	3	2.384 s	0.004 s	
...signfiltProcessCheck.checkConstraints	3	2.358 s	0.019 s	
shiftOperator	72288	1.719 s	1.719 s	
detectOnsets	3	1.632 s	0.009 s	
abstracttype.design	3	1.392 s	0.002 s	
abstracttype.superdesign	3	1.389 s	0.005 s	
abstracttype.kaiserwin	3	1.368 s	0.002 s	
abstracttype.privdesigngateway	3	1.367 s	0.004 s	
abstracttypewspecs.thisdesign	3	1.332 s	0.001 s	
abstractspec.kaiserwin	3	1.331 s	0.001 s	
abstractspec.design	3	1.330 s	0.006 s	
abstractdesign.design	3	1.310 s	0.002 s	
abstractdesign.designcoeffs	3	1.290 s	0.004 s	
abstractwindow.actualdesign	3	1.276 s	0.001 s	
kaiserhpmin.designargs	3	1.264 s	0.002 s	
abstractkaisermin.postprocessargs	3	1.258 s	0.004 s	
filterData	3	1.248 s	0.004 s	
digitalFilter>digitalFilter.filter	3	1.236 s	0.002 s	
digitalFilter>filterFIR	3	1.234 s	1.234 s	
abstractfir.iskaiserequripminspecmet	3	1.113 s	0.005 s	
basefilter.freqz	6	1.106 s	0.002 s	