Development and analysis of beat and tone detection for boomwhackers

Niels van Dalen - Creative Technology
University of Twente
15/08/2020

Abstract
At primary schools in the Netherlands, providing good music education is a challenge for teachers. Improving the quality of feedback on a musical performance of a group of children can be supported by digital technology. This study researches the potential of a system that can detect beats and tones of boomwhackers, a commonly used instrument at schools. For this, characteristics of boomwhacker sound were first thoroughly examined. A program for beat detection from a live audio feed was, in a noiseless setting, up to 100% accurate. However, processing delays were significant. Additionally, the beat detection is not boomwhacker specific. Furthermore, three different methods of tone detection have been developed iteratively. All were tested on a dedicated sample database, containing single boomwhacker tones and tone combinations. The accuracy scores of these methods were 47.6% (normalisation based), 62.8% (peak based) and 86.0% (subtraction based) respectively. The accuracy scores are an indication, actual performance of a tone detection method depends on the goal of application. Tone detection is possible based using just the fundamental frequency of a boomwhacker's tone.

Keywords: boomwhackers, beat detection, tone detection, spectrum analysis

Acknowledgements
I would like to thank a number of people that have helped me during this project.
First of all, thanks to Benno Spieker for providing the opportunity for me to do this research, as well as taking the time to help me better understand the problem and providing me with useful ideas and feedback along the way. Many thanks to Job Zwiers for providing me consistently with a lot of helpful insights and advice as supervisor. I want to thank everyone who proofread this report and provided comments that helped me to improve it: thank you Rik, Anne and Marjon. Lastly, I would like to thank everyone that has supported me in discipline, especially during the times of corona: thank you Addie, Marco and Martijn.

Content

1. Introduction

At dutch primary schools, teachers are often inexperienced and undertrained to properly teach music [1]. Often it appears to be hard to recognize what could be improved about a musical performance, by only listening. This situation is problematic because good music education has been proven to have many positive effects on children [2].

A possible improvement for music education is a system that can measure the musical performance of a group. Based on these measurements, it then provides feedback to the teacher and/or class. This report examines the potential of such a system specifically for boomwhackers, a commonly used instrument at primary schools. Boomwhackers are a set of tubes with specific lengths which, for each, determine a pitch. A picture of boomwhackers is shown in figure 1.

Figure 1: A set of boomwhackers

This research has been guided along the following questions:

- How can digital technology improve boomwhacker play in music education?
    - How do noise, different playing styles and room characteristics affect a boomwhacker's frequency content?
    - How can the beats and pitch of boomwhackers be measured?

For such a system, the relevant information consists of which tones are played, and when. A thorough analysis of a boomwhacker's characteristic sound is performed. Subsequently, matlab is used to write a script for beat detection and scripts for three different approaches to tone detection. With the proper sensitivity, the beat detection's accuracy was found to be 100%. However, processing delays within the computing environment, among others, motivated the decision to test the tone detection methods over a dedicated database of pre-recorded samples. This database contains recordings of tones and tone combinations of increasing complexity. The total accuracy scores were found to be 47.6% for a normalisation based tone detection, 62.8% for a peak based tone detection and 86.0% for a subtraction based tone detection. Detection rates were found to be the highest with samples at similar volumes. Presumably the tone detection can also work for different instruments. However, this is not thoroughly tested.

2. Literature Review

A literature study has been conducted. This study is done in order to learn from related studies and with that, develop a better problem solving methodology and reduce chances of encountering dead ends in terms of technological possibilities.

2.1 Microphone characteristics and acoustic behaviour of closed spaces

In order to do audio measurements, a wide variety of equipment can be used. For volume-only measurements, sound level meters (SLM's) can be used [3]. SLM's are devices that can measure a sound pressure level. For a sound recognition system, a SLM will perform effectively when used in combination with percussion. Because playing percussion instruments evolves mostly around timing and volume. However, microphones are more prevalent and versatile in use. This is due to the fact that besides just sound pressure levels, they also capture frequency content of a sound: a key property for pitch recognition.

Some microphone types are: directional, omnidirectional and cardioid [4]. Directional microphones perform best pointed directly at a sound source. Omnidirectional microphones are sensitive to sounds from all directions, which means that without the need to aim, they are easier to set up. Yet inevitable to this microphone type is that they also capture more (background) noise, reverb and echo. Since this research focuses on audio recognition purposes only, the relevant data is the sound that directly comes from an instrument. Any other sound artefacts are not desirable.

A cardioid microphone has a sensitivity pattern (or polar pattern) that resembles a heart shaped area, as illustrated in [5, Fig. 1]. Lee (2014) and Kamekawa (2020) conclude that cardioid microphones perform best in 3D audio recordings using microphone arrays. They have the property of being able to record multiple sources from different angles well, and are less sensitive to reverb and echo complications than omnidirectional microphones [4][6]. In the research of Lee (2014) and Kamekawa (2020), microphone arrays are used. A microphone array is a configuration of multiple microphones placed in different angles and sometimes at different positions as well [6]. At a primary school, it is practical and cheap if a stand-alone microphone can suffice for recording. In addition, there is a difference between recording for instrument recognition or for 3D audio purposes. For audio recognition, repeatability/consistency during recording eases (digital) signal processing and feature extraction. In contrast, for 3D audio experience, audio quality and stereo imaging play a big role. It cannot be conclusively stated yet whether cardioid microphones are a suited type for instrument recognition. However, the cardioid characteristic in other applications is promising.

Besides the type of microphone, positioning also plays an important role in audio recording. In a study for sound pressure levels of low frequencies in a closed space, Simmons states that rooms are never acoustically perfect [7]. Different frequencies and their respective levels will vary throughout the entirety of a closed space. This implies that the position of the microphone always influences the characteristics of sound it captures. From that, it can be concluded that a sound captured by any microphone, will bring about a different audio fragment, depending on

the space of recording alone. The same principle applies to usage of multiple microphones as well. Kamekawa's study (2020) shows that three recordings of the same sound, produced subjectively different characteristics of sound. This is caused by different types of microphone array techniques: coincident, near coincident and spaced. The coincident array recording was described as 'hard', the near coincident array as 'rich' and 'wide' and the spaced array as 'present' and 'clear' [4]. Furthermore, Gonzales (2020) found in a study that microphones close to surfaces, especially when flat, produce less accurate loudness measurements [3]. The environment of recording, as well as the positioning of a microphone (array) influences the characteristics of said recording.
Tan (2017) suggests that acoustic performance of a building is commonly rated with parameters such as reverberation time (RT), sound intensity level, noise, sound uniformity and intelligibility. Of these properties, reverberation time is one the most important indicators of acoustic behaviour. Tan also defines reverberation time as: '... the time required for reflections of a direct sound to decay to 60dB' [8]. In a classroom, a desired (and common) reverberation time is in the magnitude of tenths of seconds, and it is influenced by the amount of reflective and absorbing materials in the confined space [9]. Sound recordings in a classroom will also contain this reverberation energy. This implies that, in playing an instrument, the moment of release of any key or hit is slightly earlier than the moment recording shows it. There are two approaches to accounting for this situation. Firstly, it can be ignored, because the reverberation sound is a natural natural phenomenon and it is audible to the player, so he can account for that himself. It might not be of importance that the recording is a bit off due to consistency in this delay. On the other hand, this reverberation component of the sound will differ per room, and therefore complicates consistency in capturing the transient of an instrument. Whether this is a relevant factor depends on how the signal processing and feature extraction is done. See section 2.3.

Sound itself is, obviously, also of great importance for the performance of an audio recognition system. Humans can hear sounds from roughly 20 to 20,000 Hertz [11]. Considering the field of music only, frequencies outside of this domain are irrelevant for this research. Gonzales (2020) conducted a research in which he found that the louder the sounds measured, the higher the measurement inaccuracies became [3]. Whenever a sound becomes 'too loud' is dependent on the specific microphone or SLM used. Furthermore, Bilgic (2017) found that different surface shapes, especially near a microphone, skew the loudness levels of different frequencies that make up the total sound. All objects have an acoustic property of either absorbing or reflecting sound waves [12]. Further research on instrument types and their respective frequency ranges might be necessary in order to get a more concrete overview of the implications. On top of this, Kamekawa states that the frequency band (defined interval of frequencies) of a sound is relevant for sound localization purposes. The higher the frequencies, the harder it becomes to localize the sound source [4]. These phenomena can be accounted for by keeping as much distance between microphones and surfaces as possible.

2.1.1 Conclusion of microphone characteristics and acoustic behaviour of closed spaces

Microphone type and positioning, as well as acoustic properties of a classroom, influence audio recording for instrument recognition purposes. Microphones are more suited for this application than SLM's since they capture frequency content of a sound, whereas SLM's only measure pressure. Cardioid microphones capture audio from various angles effectively, while also keeping sound artefacts to a minimum due to their sensitivity pattern characteristic. Any closed environment is acoustically imbalanced, different frequencies and their respective levels alter throughout the space. Near (flat) surfaces, sound can get distorted easily. Microphone distancing from any surfaces is advised. Finally, the reverberation component of a sound is specific per room, depending on the presence of reflective and absorbing materials. This can potentially complicate audio recognition, depending on the type of feature extraction.

2.2 Measuring pitch and detection of rhythm

First, some terms need to be clarified. Klapuri (2008) states the following concerning pitch: "pitch is a perceptual attribute which allows the ordening of sounds on a frequency-related scale extended from low to high. More exactly, pitch is defined as the frequency of a sine wave that is matched to the target sound by human listeners. Fundamental frequency (or FF) is the corresponding physical term and is defined for periodic or nearly periodic sounds only" [13]. Every periodic signal (such as sound) can be represented by a sum of pure sine waves [14]. From these sine components, the component with the lowest frequency is considered the fundamental frequency or pitch. Sometimes these components are also referred to as 'partials'. The terms 'pitch', 'first partial' and 'fundamental frequency' and abbreviation 'F0' represent the same property of a sound. Note that in this paper all terms are used. All frequency components in a sound that are not F0, have a frequency higher than F0 and are called overtones.

Moreover, in musical terms 'monophonic' is used to describe music in which one tone is played at a time, without overlapping chords or other tones. On the other hand 'polyphonic' refers to simultaneous playing of multiple tones. Think of chords and multiple instruments. Boomwhackers individually are monophonic instruments: they play a single tone. Though when played simultaneously they have a polyphonic sound.

Eronen and Klapuri (2000) developed a pitch independant musical instrument recognition system that recognized individual instruments with up to 81% accuracy. These results were obtained by evaluating 32 temporal and spectral features from 1498 test tones [15]. Eronen (2001) later extended this work by adding more classification features, The best results were obtained by having mel-frequency cepstral coefficients being calculated over both the onset, the steady state and a subset of the earlier spectral and temporal features [16]. In another (similar) study on monophonic instrument recognition, Agostini et al (2001) evaluated 18 audio features, of which 3 were found to be most discriminating for specific instruments. Namely, the mean of the inharmonicity (presence of non-whole multiples of F0), the mean and standard deviation of the spectral centroid, and the mean energy contained by the first partial (F0) [17].

According to this research, the amount of energy in the fundamental frequency relative to the overtones is something that distinguishes an instrument from other instruments, and likely

also sounds or noise in general. Lin (2012) states that the fundamental frequency for most musical instruments lies between 20 and 2000 Hz [18]. It should be examined whether the fundamental frequency energy would suffice in recognizing boomwhackers hits among other (ambient) sounds. If this is the case, it would be useful to filter out all other (audible) frequencies, for the purpose of reducing noise artefacts and processing load. Lin (2012) also suggests that beats in music can be detected by running an algorithm that compares the energy levels of the signal in the time domain. When a beat starts, there is a big increase in energy, that then fades again until the next beat. In the case of boomwhackers, this is likely a valid strategy for beat detection: transients have a sharp and defined attack (the 'whack'). Finally, even in a scenario where a fundamental frequency of a sound is missing from a recording, the fundamental frequency can be retrieved by examining the higher harmonics and computing the largest common divisor of these frequencies [19].

2.2.1 Conclusion of measuring pitch and detection of rhythm

Previous works have proven that it is possible to automatically recognize musical instruments and their pitch. Success has been achieved by evaluating many features of audio, of which the most important ones are the mean of inharmonicity, mean standard deviation of the spectral centroid, and the mean energy in F0. Most instruments have a F0 in the range of 20 to 2000 Hz. Whether this is the case for boomwhackers is not yet clear. It is possible to retrieve a fundamental frequency from an incomplete (audio) signal.

2.3 Data processing and evaluation

The analog input to any measuring system in the scenario for this project is merely a single pressure wave, formed by all sound sources that contribute to this pressure wave simultaneously. A technique called *auditory scene analysis* (ASA) strives to separate ('unmix') this signal into its individual contributing components [20]. ASA is a machine learning (ML) based technique for feature extraction of signals. A problem with ASA is introduced by Roweis (2000). It cannot operate on a single recording, but needs a lot of data in order to be able to do intelligent instrument classification [21]. Furthermore, the structure of a piece of music is very desirable, if not necessary, for ASA to properly function. This is likely to be the case for other (ML) feature extraction techniques as well. A lot of musical sample data is used in feature extraction of multiple researches. Audio recordings of certain instruments are compared to different samples from various instruments, after which a system evaluates similarities [15]. All found researches did not have real-time instrument recognition features, whether this is a possibility is yet unknown.

For feature extraction of a raw analog signal, some preprocessing is proposed by Lin (2012). The most important steps are filtering redundant frequencies (to remove noise and reduce processing load), normalisation (to make the system more robust for varying input levels) and an energy analysis of the signal (done by taking a FFT). For this FFT, one should consider a sampling rate and resolution high enough, as to not introduce aliasing or low accuracy for frequency detection.

2.3.1 Conclusion of data processing and evaluation

Machine learning is a common way for instrument recognition techniques. This method does require samples of the instruments in question, and the more data, the better the performance can get. In the case of boomwhacker it is possible to record samples beforehand in order to compare a (real-time) recording to. Noise has not been discussed much for signal processing, however it is important to realize that a lot of noise can be avoided by filtering as much as possible, without losing essential properties of the boomwhacker's sound. Signal evaluation is done in time and frequency domain and both are important for an instrument's characteristic sound.

3. Requirements

3.1 Capturing requirements

This section describes a process of narrowing down towards a specific requirements set, which is then elaborated on in chapter 3.2.

Before prototyping or implementing any technology, specific, realistic requirements should be set. It is important to have a clear overview of the situation, the occurring (and recurring) problems and technological possibilities to solve them. The current situation is that Dutch primary schools fail to effectively provide music education. This is problematic because music education is proven to have many positive effects. It helps develop motoric skills, learns children how to cooperate as a group and increases their verbal intelligence and memory [2].
This is due to absence of training and confidence among teachers, and because of that, often musical classes or activities don't even take place [1]. At the time of writing, music education in the Netherlands is not compulsory. Music teacher and promovendus on application of interactive technology in music education Benno Spieker has been contacted. From interviews as well as a primary school music workshop observation, the following statements were found to be most relevant:

- In music education, teachers have trouble giving proper feedback to children based on what they can hear
- Quite a few commonly used instruments are rather expensive (for primary schools)
- Music education can be divided in explorative (fe. giving children freedom to discover new sounds and ways of making music) and performative (fe. children cooperate in order to perform a piece of music)
- Boomwhackers are a cheap and commonly used instrument
- Most children stated to prefer instruments over singing
- Children prefer to make sound a.s.a.p., rather than exactly figuring out an instrument

Focusing on a single instrument allows for better results in limited time. Boomwhackers are simple, cheap and therefore common. For this reason narrowing down to boomwhackers guarantees that results will be relevant for the music education field. Another benefit of boomwhackers is that a person usually plays only one or two notes. This means that performance of a certain note (or two notes) can directly be linked to a person. In a group setting it is very likely that many boomwhackers are played at the same time, therefore it should be able to recognize multiple tones at once. A system that can measure the onsets and pitches of a group performing with boomwhackers, whilst having a way to visualize this, is a possible solution in helping teachers judge the musical performance of a classroom.

Currently music education already is a challenge for teachers, this dictates that any technology should be simple to set up and use and does not add cognitive load. A system with, for example, many distributed sensors would add complexity in setting up, making it harder for the children to not break anything and likely raise costs. In contrast, a single measuring point limits these downsides and seems plausible for effective use.

3.2 Specification of requirements

A technological solution is a system that has the capability of being able to do a number of things. First of all, it should be able to detect the beats (onsets) of boomwhackers, in order to provide information on the rhythmical timing of the onset. Assessing rhythmical timing additionally requires the system to be able to possess a reference tempo, this could for example be tempo and onset data of a song. Even better would be live, smart rhythm evaluation capabilities. Furthermore, boomwhacker tones should be able to get properly recognized, even if multiple boomwhackers are playing at the same time. All above aspects should be able to run in real time and preferably as a standalone application.

In this research, it is considered whether the requirements can be achieved with a system using just a microphone and a PC. The microphone records audio and feeds it into a PC that can analyze and process the data. Since it is unlikely a perfectly performing system will be developed right away, the specified requirements are given priorities using the MoSCoW method. The following table shows, using the MoSCoW method, a concrete overview of the requirements of the system.

Table 1: System requirements and their priorities within this research

Priority	**Requirement**
Must have	- Onset detection of a boomwhacker hit - Tone detection - Detection of two simultaneous tones
Should have	- Unnoticeable latency (approximates real time performance) - Visualization/feedback of captured data - Detection of three or more simultaneous tones
Could have	- System can run as a standalone application - System is suited for a larger range of notes
Would have	- Smart dynamic rhythm evaluation capabilities - Extensive (real time) feedback

4. Realisation

This section discusses characteristics of boomwhacker sound, as well as the process of, and findings from, developing a beat and tone detection system for boomwhackers.
A short overview is given below.

Within matlab, multiple scripts have been written iteratively. Firstly, a beat detection program, running on a live audio feed into the PC. This has later been elaborated to being able to also detect single tones. Having found that the beat detection has some inherent limitations (see section 4.2), the approach of a live audio feed has been dropped in order to ease the process of (more advanced) tone detection. This has been done from pre-recorded samples, loaded in as data. Three different algorithms are written for detection of boomwhacker notes from files:

1. Normalisation Based tone detection (4.4.2): An input signal is normalized, and the peak value is compared against ideal frequencies. If this peak value is within a margin of the ideal pitch, a tone is being detected. Limited to one note.
2. Peak Based tone detection (4.4.3): the frequency of peaks in a signal above a set threshold are all evaluated against ideal pitch in a similar fashion as 1. This method can detect multiple peaks.
3. Subtraction Based tone detection (4.4.4): the frequency value at the highest peak of the signal is considered, and again compared to the ideal pitch of the tones. If it is detected as one of the tones, this is being noted and a clean sample of that tone is subtracted from the signal. This process repeats until the highest peak falls below a (soft) threshold.

4.1 Used equipment and recording method

The used hardware consists of a PC and a microphone, in this case:

- Devine USB 50 Microphone
- Lenovo thinkpad P1 (gen 2) PC

The microphone has been chosen for its easy connectivity (plug and play) and cardioid polar pattern. Such a polar pattern pointed towards the classroom is very well suited for recording many sources, without capturing much noise coming from the microphone's back: reverb from boomwhackers, the teacher's voice/accompaniment and/or speaker sounds. Recording is done at a (default) sampling frequency of 44100 Hz, for which according to the Nyquist theorem allows for capturing frequencies up to 22500 Hz. The bit depth is 24 bit, integer.

The PC's required performance is deliberately not considered (in depth), because the system should be able to perform on different PC's with different performances.

The software used is Mathworks' Matlab (R2020A, version 9.8.0.1359463) including the digital signal processing toolbox (DSP, version 9.10). Matlab has been chosen because it has a lot of good reference material on its functionalities, as well as a signal processing toolbox which contains some convenient functions for digital audio processing.

For the boomwhackers, a commonly available set has been used. This set consists of 8 boomwhackers with notes (from low to high): C-D-E-F-G-A-B-C. From here on, the first C will be referred to as ‘Clow’, and the last C will be referred to as ‘Chigh’. The boomwhackers and their exact properties are elaborated on more in section 4.1.

Figure 2: Devine USB 50 microphone (left)[1] and the used boomwhacker set (right)

During the research boomwhackers have often been recorded. All performed recordings are done at one meter from the microphone, in a very dry medium sized room. The recording setup is shown in figure 3. The background noise profile of the room is shown in figure 4.

Figure 3: Used recording set up

[1] Photo from: https://www.bax-shop.be/nl/usb-microfoons/devine-usb-50-usb-podcasting-microfoon

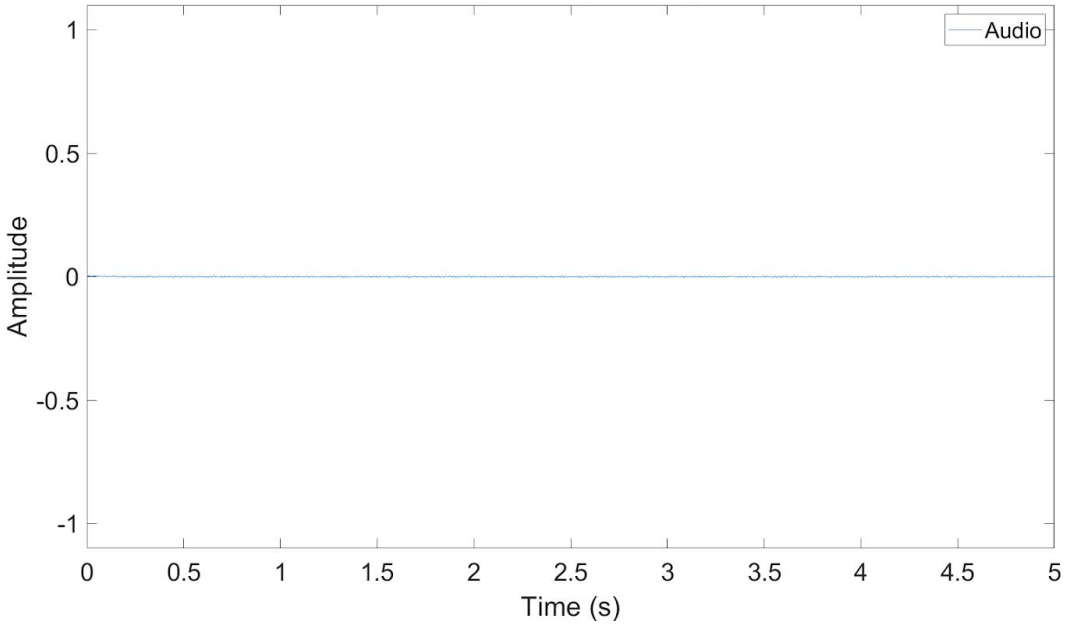


Figure 4: Background noise profile of the recording room

Hitting of the boomwhacker is done on the flat or slightly curled inside of a hand. All recordings are done of boomwhackers being played using this technique, unless explicitly stated otherwise.

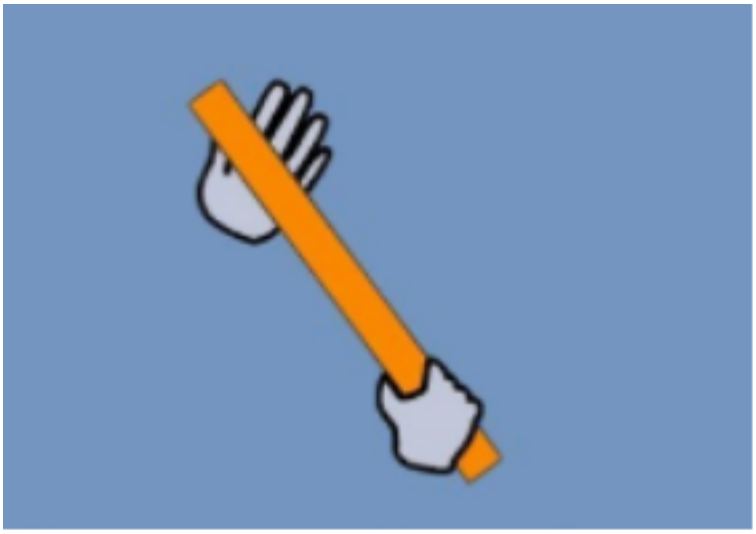


Figure 5: A common way of playing the boomwhacker, as done for recordings

4.2 General Characteristics of boomwhacker sound

4.2.1 Transient and frequency spectrum of a boomwhacker

Characteristics of a sound are made up of two main properties. Its transient: the development of a sound or signal in the time domain, and its frequency content. Each sound source has a unique combination of these two properties. In order to come up with an insightful method for boomwhacker beat and tone detection, it must be determined what properties are useful, consistent and unique to boomwhackers.

Existing audio software is used first as a control. The goal is to get an idea of the properties of boomwhacker sound. Furthermore this control is used as reference, as to confirm that self written data import and visualizations in Matlab correctly represent the properties of the boomwhacker sounds before any further processing is done. The control for this research is done using transient and spectrum visualizers within the FL studio DAW (digital audio workstation). A hit of each individual boomwhacker has been recorded.

For all tones the transient graphs and frequency spectrum of self-written code provided the same results as the control, and is thus performing properly. A full transient and frequency spectrum comparison of the control and matlab figures can be found in Appendix A. As an example, the Clow transient and a relevant part of its frequency spectrum are shown below.

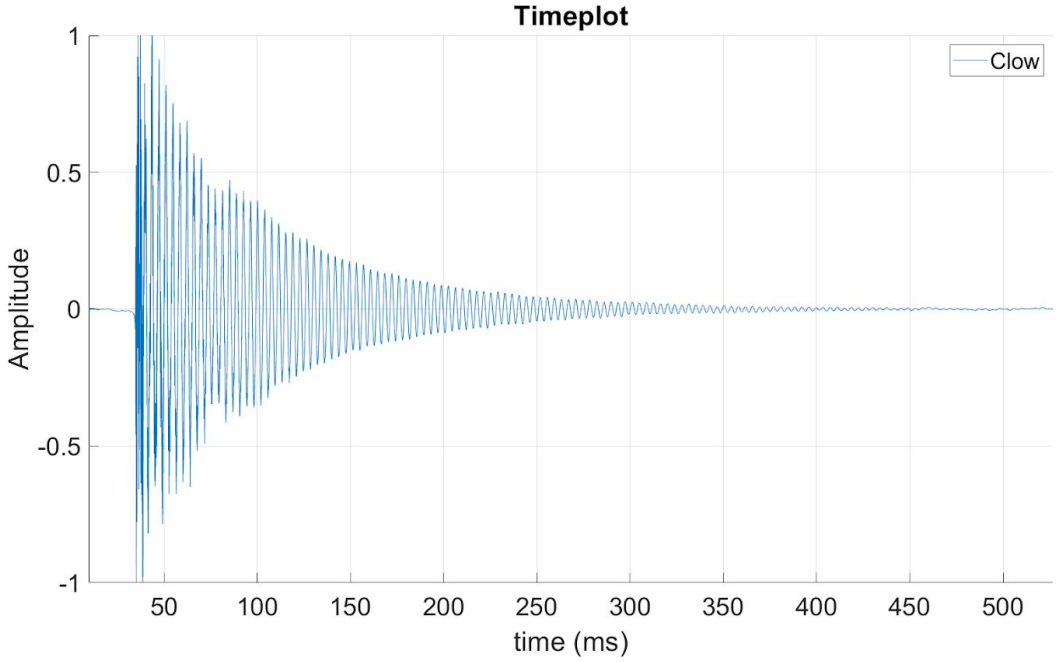


Figure 6: Transient of the Clow boomwhacker note

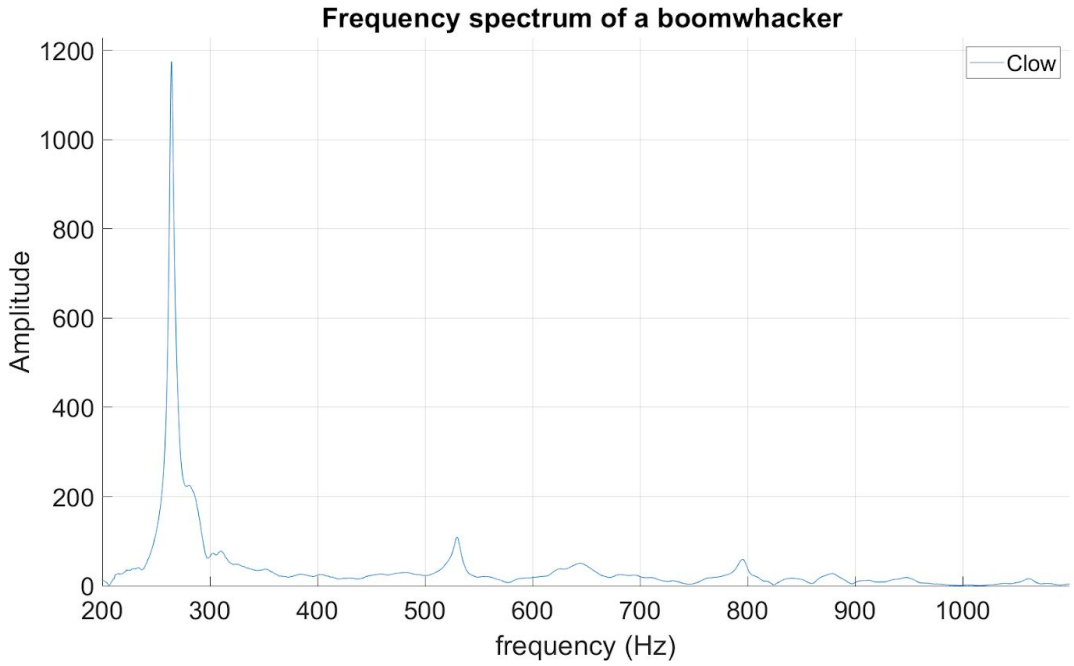


Figure 7: Frequency spectrum of a Clow boomwhacker note

Based on an analysis of all graphs, it can be stated that:

- A (dry) boomwhacker note in this frequency range will last between 500-200ms, with higher pitches decaying faster.
- The transient has a sharp attack and approximately exponential decay.
- Boomwhackers have a well defined, and maximum peak height at their fundamental frequency (in fig. 7: ~260 Hz)
- One or two harmonics are visible, however these are a lot less consistent and defined (in fig. 7: ~530 Hz)
- Frequencies around the fundamental frequency have increased energy as well (which will be referred to as frequency 'bleeding')
- The relevant frequency range of the boomwhackers FF's is between ~260-525 Hz (or ~260-1600 Hz if the first two harmonics are included)

4.2.2 Pitch and Harmonics

The fundamental frequency (FF or pitch) is the lowest frequency component, and is very distinct in the graphs. The FF, or pitch, of notes depends on the musical scale used. In Western countries, by far the most common scale is the equal tempered scale [22]. It should be investigated how much these ideal, theoretical frequencies deviate from the actual frequencies being measured. A table comparing the two is provided on the next page.

Table 2: Ideal pitch versus actual pitch of boomwhackers

Note	**Theoretical ('ideal') pitch, equal tempered scale (Hz)**	**Measured boomwhacker's pitch (Hz)**	**Frequency offset: Measured - ideal (Hz)**
Clow	261.63	263.0	+1.37
D	293.66	295.5	+1.84
E	329.63	330.5	+0.87
F	349.23	350.5	+1.27
G	392.00	393.5	+1.50
A	440.00	441.5	+1.50
B	493.88	494.0	+0.12
Chigh	523.25	522.0	-1.25

On average, the offset with respect to the ideal frequency is 1.22 Hz. Percent wise, that is a maximum deviation of 0.47%. Such a small deviation is not expected to pose any problems.

As can be seen in figure 7, the frequency spectrum of a boomwhacker consists of more peaks than just the fundamental frequency. An overview has been made of the tones and their respective frequency content up until the third harmonic. These harmonics are the octave (1st) and fifth (2nd) of the fundamental frequency.

Table 3: Ideal pitches and two harmonic frequencies

Tone	FF (Hz)	1st Harmonic (Hz)	2nd Harmonic (Hz)	Difference with previous FF (Hz)
Clow	261.63	523.25	784.875 (~G*)	-
D	293.66	587.55	881.325 (~A*)	32.03
E	329.63	659.25	988.875 (~B*)	35.97
F	349.23	698.48	1047.72 (~C*)	19.60
G	392.00	784.00	1176.00 (~D*)	42.77
A	440.00	880.00	1320.00 (~E*) [!]	48.00
B	493.88	987.76	1481.64 (~F#*)	53.88
Chigh	523.25	1046.50 [!]	1569.75 (~G*) [!]	29.37

*Frequency roughly corresponds to the pitch of this note
[!] Harmonic from recordings seems to deviate more than 10Hz of its ideal value

All ideal frequencies were compared against their physical counterparts. Most physical harmonic frequencies were almost identical to the ideal frequencies. A few deviations are marked. It is not surprising that these occur in higher frequencies, absolute frequency intervals between notes increase for higher pitches.

4.2.3 Different playstyles

Previous experiments were all done by playing the boomwhacker in a rather common fashion: by hitting it on a flat or slightly curved hand. However, there are other ways boomwhackers are being played, this section examines a few of those playstyles and their influence on the sound spectrum. For simplicity, all recordings are done using the Clow note. In order to eliminate volume inconsistencies, all plots are normalized to an amplitude of one. Each recording is repeated three times. In this section only one example plot per experiment is shown, an overview of all the plots can be found in Appendix B.

4.2.3.1 Different hand positions

Three different hand positions are examined: far apart, normal and close to each other.

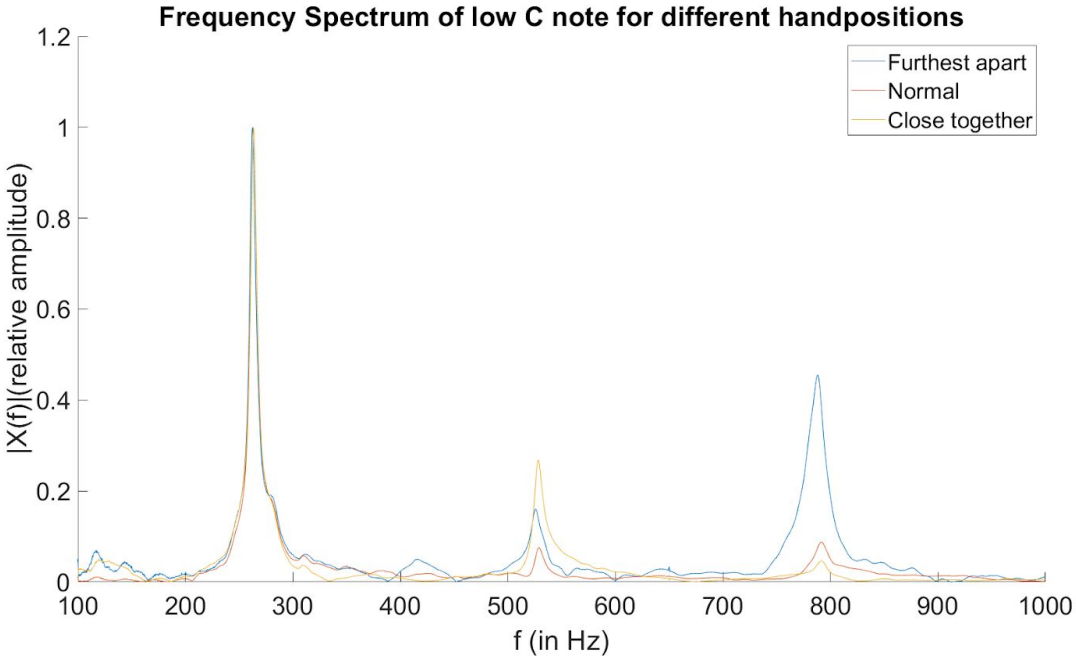


Figure 8: Frequency spectrum of Clow for different hand positions (normalized)

The FF hardly changes at all, whereas the 1st harmonic (~530 Hz) and 2nd harmonic (~780 Hz) do change quite a bit depending on hand position. The 1st harmonic did not behave in a reliable, distinct manner. On the other hand, in all captures the 2nd harmonic showed a much larger amplitude with hands far apart.

4.2.3.2 Different loudness

The loudness of many instruments can greatly affect their timbre, therefore three (subjectively played) loudnesses of boomwhackers are compared. The resulting spectrum can be found in figure 9.

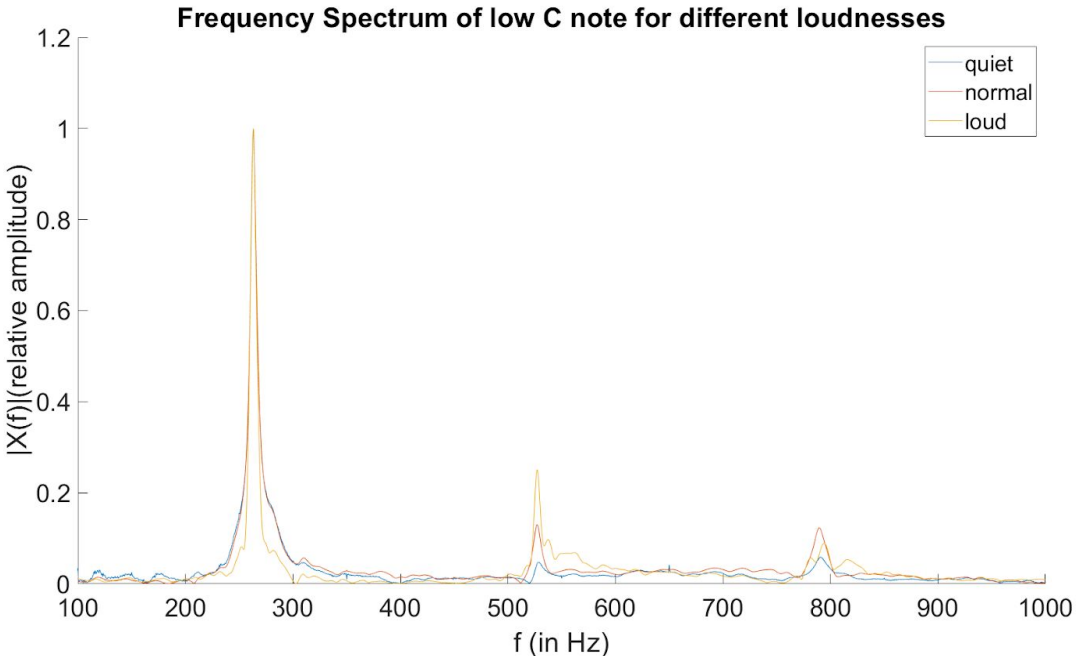


Figure 9: Frequency spectrum of Clow for different loudnesses (normalized)

Louder boomwhacker hits seem to reduce 'bleeding' around the FF, it becomes more defined. It also tends to raise the amplitudes of harmonics. Yet, as can be seen in figure 9, the loudest hit doesn't always have the highest peaks on its harmonics.

4.2.3.3 Different surfaces

Apart from using hands, other surfaces or objects can be used as well. The hand is compared against a knee and a wooden stick.

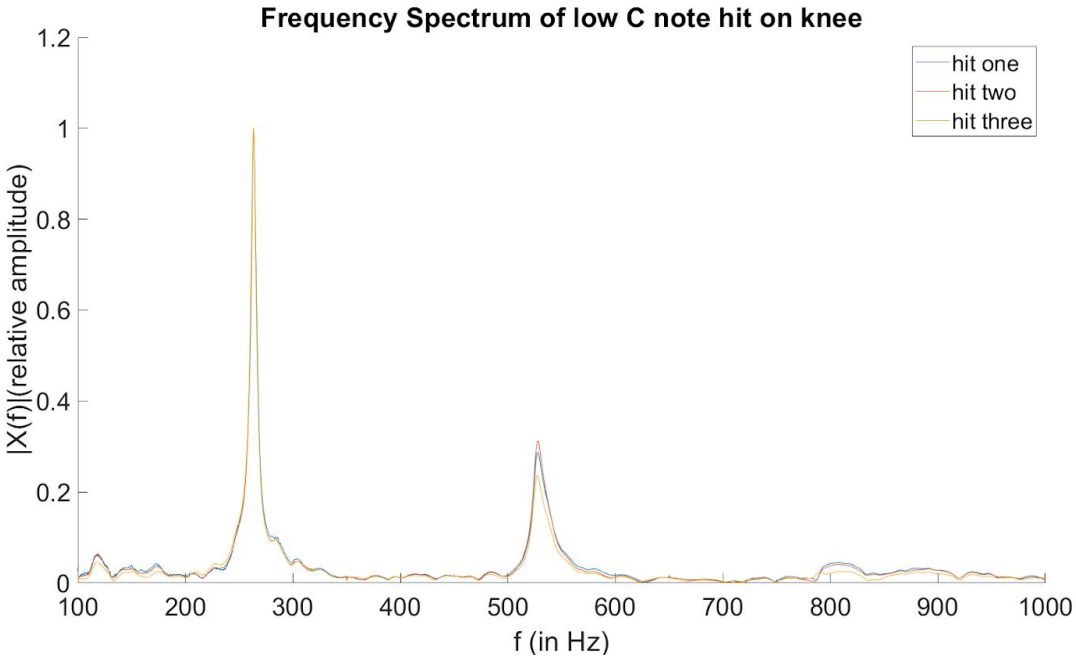


Figure 10: frequency spectrum of Clow for a hit on the knee (normalized)

The hits on the knee are quite different than on hand: a very well defined 1st harmonic, whereas the 2nd harmonic is effectively gone. The FF is still by far the most distinct.

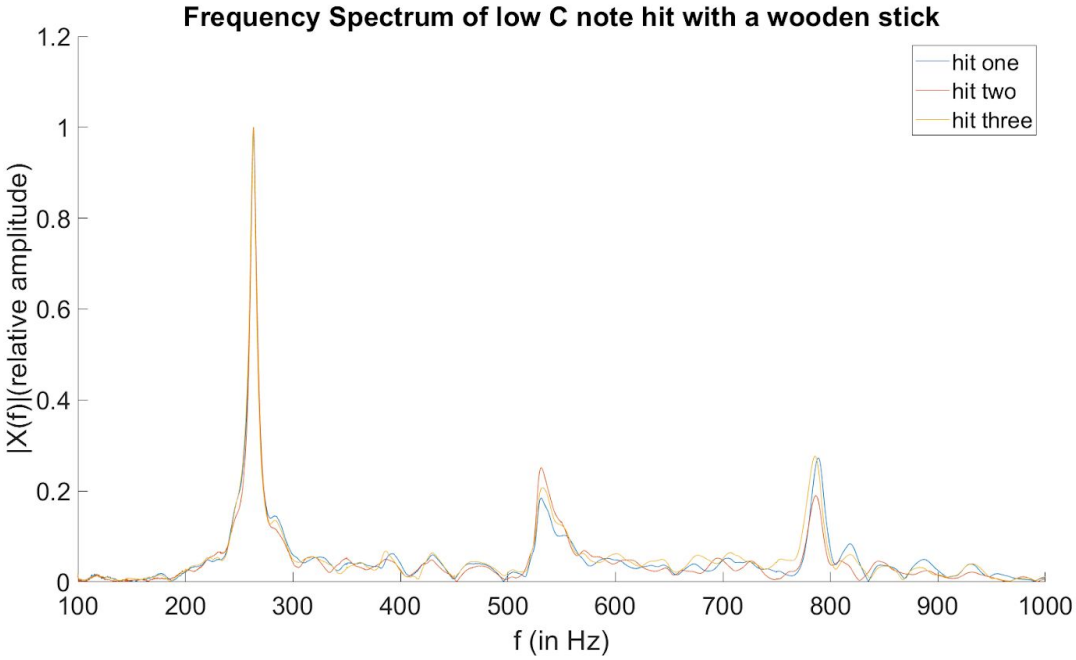


Figure 11: frequency spectrum of Clow, hit with a wooden stick (normalized)

Wooden stick hits produce strong harmonics, as well as more noisy frequencies between the FF and harmonics. Again, the FF is still very distinct.

4.2.3.4 Conclusion of different playstyles

The most apparent characteristic of the boomwhacker is that, no matter the playstyle, the fundamental frequency is very well defined. The louder the hit, the less the FF bleeds to surrounding frequencies as well. The 1st and 2nd harmonic are in most cases, quite noticeable. Nevertheless, the amplitudes can vary a lot depending on hand position, loudness of the tone and used surface. Their unreliability seems to be inconvenient for a tone detecting application.

4.2.4 Clipping

Storing audio in a computer introduces a risk of clipping. Digital audio is stored as a number of values between -1 and +1, whenever a signal trespasses this limit its additional magnitude cannot be stored and will be set to 1, or -1 respectively. Whenever this happens, the signal can get (audibly) distorted with high harmonics. [23] Clipping can happen during recording as well as during processing. The effect of clipping is examined by digitally clipping a signal.

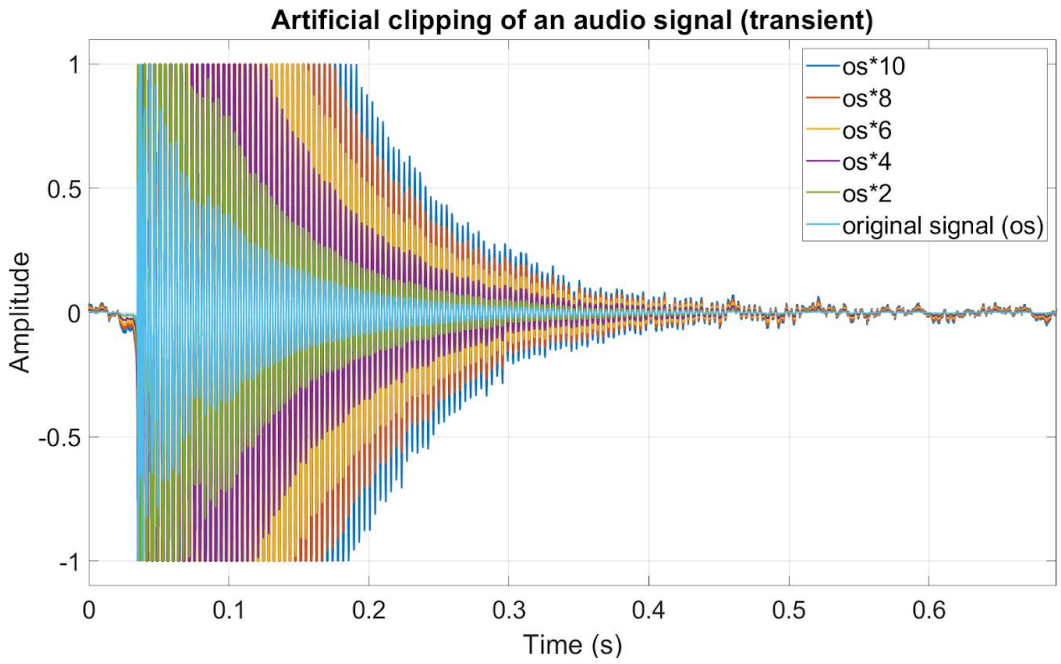


Figure 12: digital clipping of an audio signal (transient)

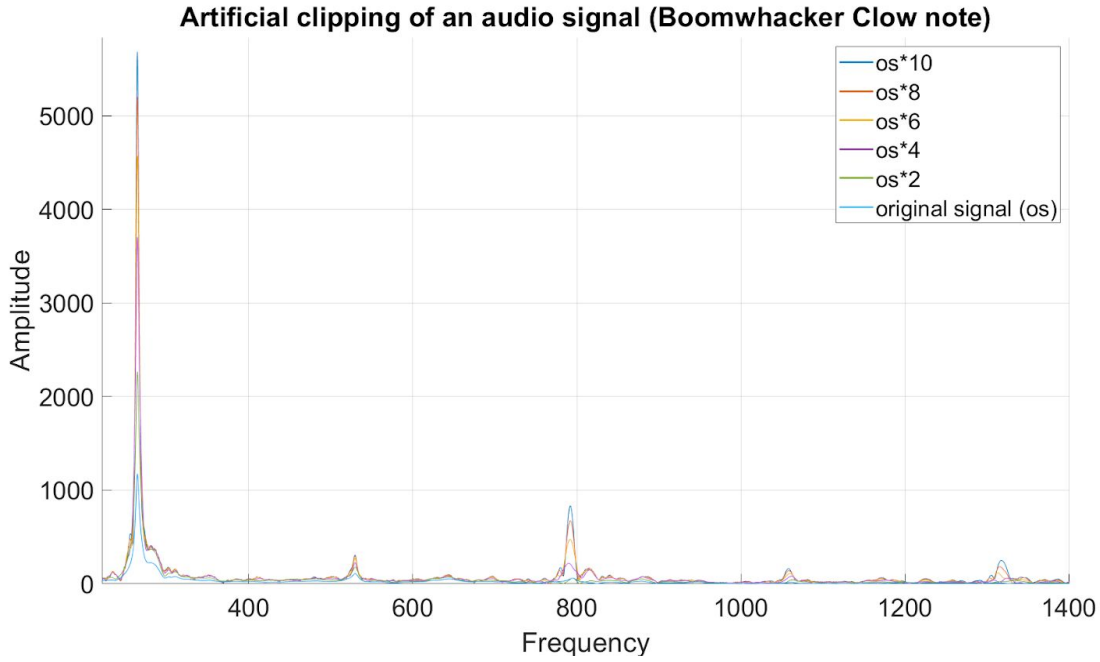


Figure 13: digital clipping of an audio signal (frequency spectrum)

From the graphs above it can be concluded that digital clipping does not introduce any unwanted frequency content in the range of interest. It also does not affect the fast fourier transform of the signal significantly. Clipping is unlikely to be a relevant factor in tone recognition.

4.2.5 Conclusion of general characteristics of boomwhackers

The boomwhackers considered for this research have a relatively short transient, with a very sharp attack. Higher pitch notes have shorter transients. No matter the circumstances, the FF of a boomwhacker is always very high in amplitude. However, frequencies around it can become greater in amplitude as well due to ‘bleeding’ from the FF. This bleeding is less pronounced when playing louder. The 1st and 2nd harmonic of boomwhackers are in many cases pretty well noticeable. However, their amplitudes with respect to the FF and each other can vary greatly depending on numerous factors. These factors include (but are likely not limited to): the positioning of hands, the loudness of playing and the surface that is hit. The 1st harmonic of Clow is at the same frequency as the FF of Chigh, it might therefore be hard to distinguish the two. Given the very stable presence of a strong FF and the rather big unreliability of harmonics, it is recommended to focus on the FF for tone detection. Clipping is of little effect concerning the frequency content.

4.3 Beat detection

This section discusses the process and implementation of a developed beat detection algorithm. The full code of this algorithm can be found in Appendix D.

4.3.1 Beat detection from a real time audio feed

A simple beat detection algorithm has been developed. It is based on comparing values between the first and second half of a short FIFO digital audio buffer. The buffer size is 2048 samples, at a sampling rate of 44100Hz. As determined in section 4.1, the most distinct frequency content of the boomwhackers is in the range of 260-1600 Hz. Additionally, harmonics were found to be rather unreliable, the range of pitches of the boomwhacker set spans approx. 260-525 Hz. Therefore, in order to detect the highest pitch according to Nyquist law, the minimum value of the buffer should be: 2 x 525 = 1050 samples. The exact buffer size has been chosen by practical experimenting, buffer sizes smaller than 2048 false triggered too often, and bigger buffer sizes caused too much delay without performing better. The beat detection program compares the values of the buffer to each other: if the values in the second half of the buffer are substantially higher than in the first half of the buffer, a beat is detected. Results are obtained by doing 7 onsets, increasing in loudness, during a 5 second period. For all tests the ‘Clow’ boomwhacker has been used.

The beat detector operates with a certain difference threshold. It defines how much the minimum difference in the buffer values must be. In order to illustrate the impact of its value, a comparison has been made with this factor between 1 and 25. In figure 14, the performance graph of a sensitivity factor of 15 is shown. In figure 15, an overview of all sensitivities is given. Performance graphs for all separate sensitivity values can be found in Appendix C.

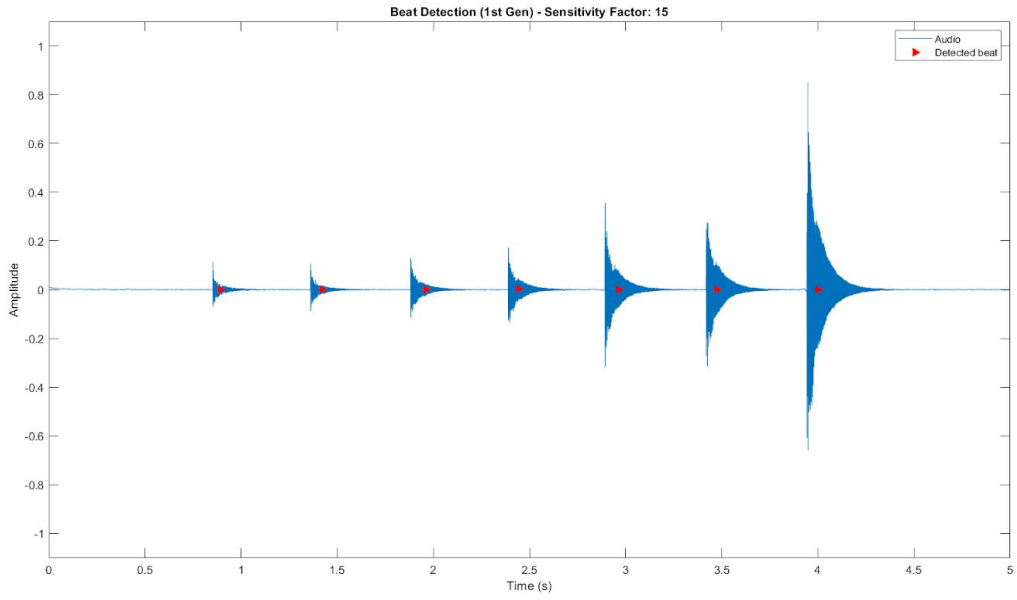


Figure 14: Performance of beat detection at a sensitivity factor of 15

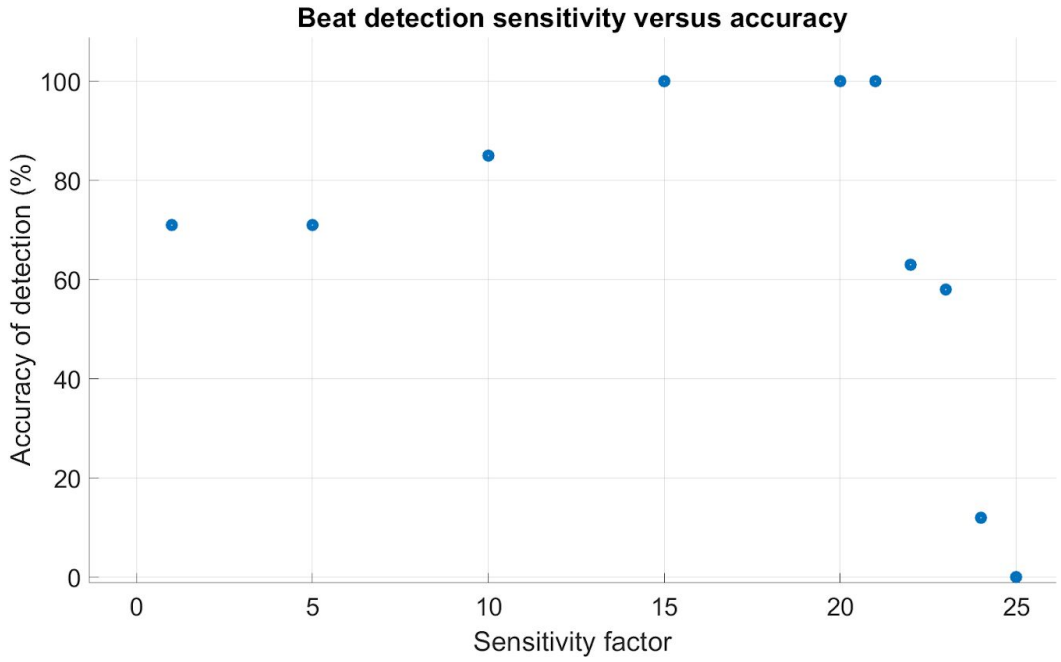


Figure 15: Beat detection sensitivity factor versus its beat detection accuracy

From these figures it can be stated that in noiseless conditions, the beat detection works best at a sensitivity factor of 15-21. With higher sensitivities than 21, false positives are increasing rapidly. This indicates that a high sensitivity, even when performing perfect in this test, is more prone to false positives than a low sensitivity. The best sensitivity factor has the highest accuracy, but the lowest sensitivity, especially for noisy situations. In this case the best sensitivity factor would be 15.

In figure 14, it is clear that detected beats lag the actual beat. This is due to the nature of the beat detector, as well as additional processing delays. The weakest beat being able to be detected is a beat that needs the entire second half of the buffer to be filled with (part of the) boomwhacker transient, before detection. On the other hand, beats with very high energy are detected rather fast: when the beat enters the buffer, the threshold is almost immediately reached and a beat is detected. An illustration of these differences is given in figure x. A weak beat (top) and a strong beat (bottom) are shown. As can be clearly seen: a weaker beat needs more of its transient to fill the buffer before the threshold of detection is passed. This causes a bigger time delay (indicated in orange) compared to a stronger beat. Note that the total amount of energy in the first half of the buffer is the same for both cases.

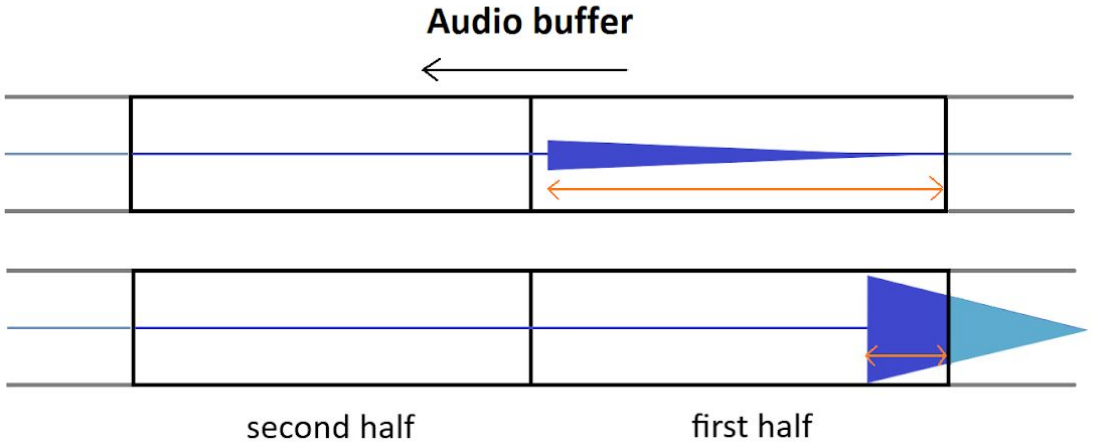


Figure 16: Difference in moment of beat detection for a weak (top) and strong (bottom) beat

At a sampling frequency of 44100 Hz, we can compute the duration of the buffer:

$$t_{buffer} = F_s / N_{buffer} = 44100/2048 = 21.53\ ms$$

Given that in an extreme case, half of the buffer needs to be filled with the transient of a beat before detection, the inherent delay can theoretically reach $21.53/2 = 10.76\ ms$. This delay is rather small, and in practice will often be smaller. Yet the delay is variable depending on how loud a beat is. On top of that, the processing itself unfortunately introduces a much greater delay to the detection. Additionally, using the PC for other audio applications (recording in a different program, playing music etc.) also causes delay and other artefacts in the audio feed. It is suspected that the audio buffer can underruns in such a situation but this has not been researched.

In an attempt to get more accurate beat timings, a beat detection correction factor is introduced. Many beats were recorded and then the detection of the algorithm has been compared versus a manual beat detection based on the waveform's shape. The figure of this experiment can be found in Appendix C. On average, the total delay was 72 ms. Therefore, this value is subtracted from each time stamp of beat detection and increased accuracy. Furthermore, this beat detection algorithm only considers energy in the spectrum, not its frequencies. Therefore it doesn't discriminate on any other sounds that form a beat, such as a clap of hands or other instruments.

4.3.2 Conclusion of beat detection

Beat detection using an audio buffer based algorithm within matlab is possible. For boomwhackers, the pitch is the most distinct feature of its signal and the maximum pitch in consideration is ~525 Hz (Chigh). Therefore the minimum buffer size used should be 1050 samples. At 2048 samples, the detection was found to be the best. With a well tuned sensitivity accuracies of 100% were achieved. Using this approach a delay of up to 10.76ms is inevitable. This delay is pretty small compared to the processing delays within matlab. On average the delay of a beat was 72 ms late, this can be compensated by subtracting this time from the registered timestamp of the beat. Without continuous spectrum analysis, making a beat detector that only triggers on boomwhackers sounds is impossible. With it, together with visualizing, such a program would likely cause even more processing delay.

4.4 Tone detection

This section covers an iterative process of three different approaches to tone detection. These are normalisation based tone detection, peak based tone detection and subtraction based tone detection. As has been concluded for the beat detection, the delays within matlab get too high for practical application. On top of that, the beat detector will require significant and complex additional development if it were to specifically distinguish boomwhackers from any other beats. Together with an inevitable lack of consistency in live recorded data, these factors contributed to the decision of implementing and testing the tone detection methods on pre recorded samples in a database, rather than with a live audio feed. Using this approach, the three methods can also be compared against each other without any other variables. The database contains samples of single boomwhacker tones, as well as more complex combinations of tones. More information about the database is in section 4.3.1.

An important aspect in analysis of a signal's frequency content is frequency resolution. Frequency resolution dictates the smallest frequency difference that is still possible to be distinguished. It is defined as the sample rate divided by the amount of frequency bins of the fast fourier transform (FFT). Frequency bins together make up the horizontal axis of a frequency spectrum. For example, a frequency resolution of 1 Hz means that any frequency difference smaller than whole integer values cannot be detected. The energy of a signal at 100.5 Hz will be divided over the bin for 100 Hz and 101 Hz.

Since the tone detection methods will be run over a database, processing delays are a non factor and a high frequency resolution can be used. At the recording sample rate of 44100 Hz and a using 11025 frequency bins, the frequency resolution of the frequency spectra from the data is:

$F_{res} = 44100/11025 = 0.25\ Hz$.

4.4.1 Sample Database (dedicated test set)

As introduced in 4.3, the tone detection algorithms run over the same database. The samples for this database have been recorded in mono using FL studio. Mono is chosen over stereo because this reduces the amount of data by a half, and the additional stereo data has no additional value for this application. The recording circumstances are identical to those of all other recordings (without noise, 1m from the microphone etc., see section 4.1 for all details). The database contains samples of three different volumes. For each volume, a separate recording is done: a hard, medium and soft hit. Afterwards, these recordings were modified to become (almost) exactly 0dB, -3dB and -6dB respectively, as determined by FL studio's built in dB meter. Samples of 0dB are allowed to clip a bit, since this has insignificant effect on the spectrum (see 4.1.4.).

Three different volumes for all eight notes form the base of the database, and are used to create additional test samples. Note that these additional samples are generated within matlab itself. A brief overview of all (generated) sample categories and their purpose is given in table 4.

Table 4: Types of samples in the database and a motivation for their presence

Type of sample	**Purpose**
Random sounds	Determine if/when the system will false trigger
Combination of Clow and any other tone (for equal and different volumes between notes)	Determine how well the system can handle simultaneous pairs of tones
An additional recording of each individual tone	Check whether the detection of the initial recordings weren't 'lucky positives'. Also used for combinations for subtraction based tone detection (see 4.4.4.)
Extreme condition: two tones with the lowest absolute frequency difference (for equal and different volumes between notes)	Determine whether the system will accurately determine tones very close to each other
Extreme condition: two tones with overlapping frequency content due to fifths (see 4.2.2.) (for equal and different volumes between notes)	Determine whether both tones will get detected in subtraction based tone detection (see 4.4.4.)
Extreme condition: all (eight) tones at the same time (for equal and different volumes between notes)	Determine the limit of simultaneous tones that can be detected.
All standard major and minor chords possible (for equal and different volumes between notes)	Determine whether the system will correctly recognize three simultaneous tones, of which chords are the most likely combination
All of the above combinations, but with a different recording for the tones at 0dB	Check whether detection of initial recordings weren't 'lucky positives', for combined signals. Also used to test the reliability/repeatability of subtraction based tone detection (4.4.4.)

4.4.2 Normalisation Based Tone Detection

This section covers the process of developing a normalisation based tone detection algorithm. The algorithm was first implemented on a real time audio buffer, and later adapted to run within a script that imports files from a folder. The codes can be found in Appendix D.

By now it is clear that the FF component of a boomwhacker signal is very reliable. Since each tone has a unique pitch, accurately detecting the frequency of the FF should be enough to distinguish the tones. The idea was initially built as an extension to the beat detection program, this method is summarized in figure 17. The detection from files is, apart from its signal source, identical.

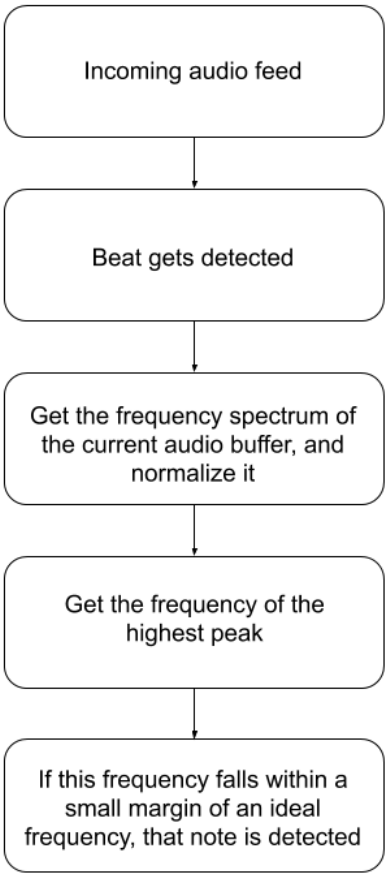


Figure 17: Schematic overview of the normalisation based tone detection mechanism

The ‘current audio buffer’ refers to the same (2048 samples big) buffer used for beat detection. An important factor in obtaining a frequency spectrum of a signal, is its frequency resolution. In this case of a live audio feed the amount of frequency bins is 2048, and thus the frequency resolution:

$$F_{res} = samples/Nbins = 2048/2048 = 1\ Hz$$

The frequency margin for detection is set at 10 Hz, this allows for some deviation from the ideal frequency, whilst also being small enough that a single frequency cannot double trigger two adjacent tones. The algorithm’s performance is tested by playing each tone 10 times. This has been done with both the internal PC microphone, and an external microphone. The results are given in table 5.

Table 5: Performance of normalisation based tone detection from a live audio feed

	Internal PC microphone	**External Microphone (devine USB50)**
Tone	**Correct tones (falsely triggered notes)**	**Correct tones (falsely triggered notes)**
Clow	10	10
D	10	10
E	10	10
F	8 (G & D)	10
G	10	10
A	10	10
B	7 (3x Chigh)	10
Chigh	10	9 (B)

Table 5 suggests that using a dedicated microphone improves the accuracy of detection. When using just the built in microphone of the PC the accuracy of detection was 93.8%, increased to 98.8% when using an external microphone. Although, this is based on a relatively little data.

This implementation still depends on the rather impractical beat detection. That is why this is the only tone detection testing done using a live audio feed. The same methodology has been implemented on a script that runs over the sample database, rather than a live recording. The results thereof are presented and discussed in section 4.4.5.1.

When only frequencies in a small range (~260-525Hz) are being considered, all other frequencies in the signal (which in total ranges from 0-22500 Hz) are unnecessary and could be omitted. Getting rid of unnecessary frequencies can be achieved by using a bandpass filter. However, virtual bandpass filtering more than tripled processing time. Additionally, when bandpassing a signal before normalizing it could lead to false triggers. For example, one of the random sounds that was included in the sample database is: the shaking of keys.

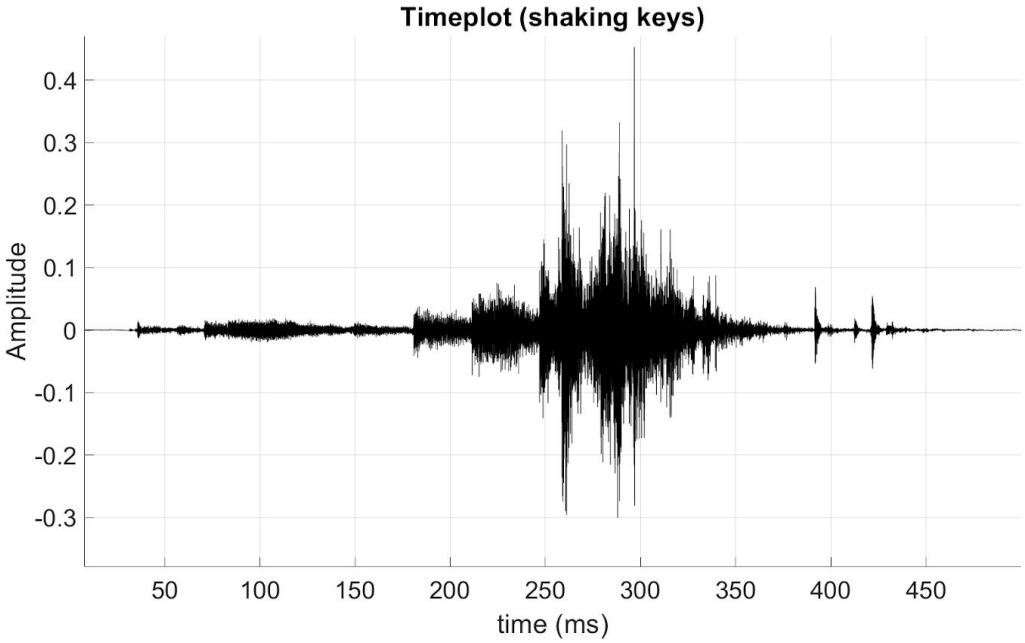


Figure 18: Timeplot of shaken keys' sound

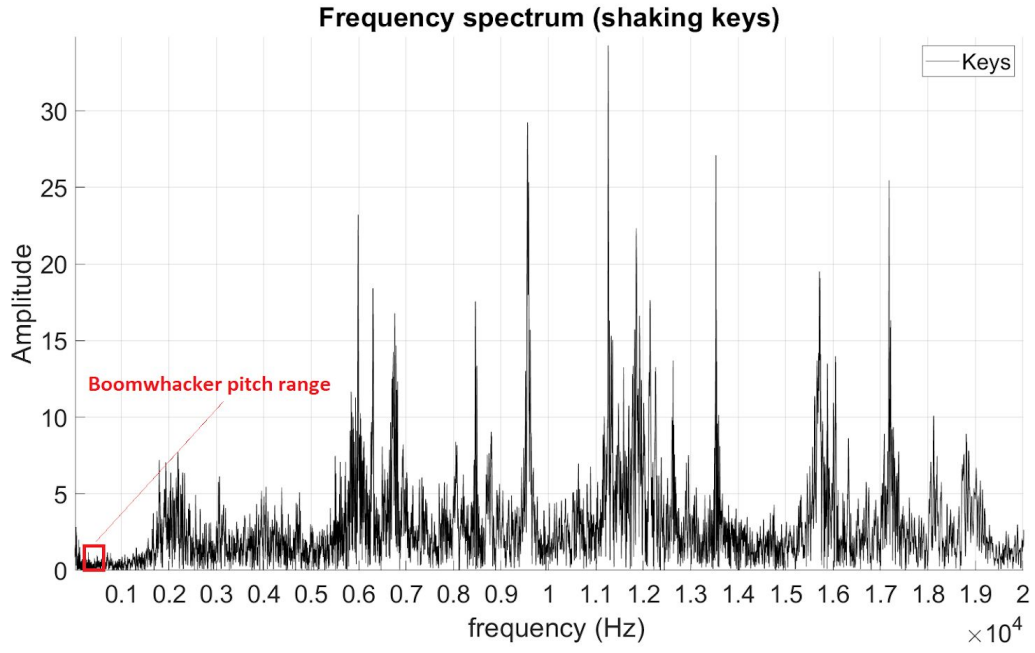


Figure 19: Frequency spectrum of shaken keys' sound

For shaken keys, by far the most energy in the spectrum is within rather high frequencies, especially when compared to the range of boomwhacker pitches, which is annotated in figure 19. When a bandpass filter is applied to the relevant frequency range (~260-525Hz), the signals roloff to 0 outside that range, the resulting (non-zero) frequency spectrum that remains is shown in figure 20. Even after zooming in, the energy of the keys in these frequencies is just barely visible (fig. 21).

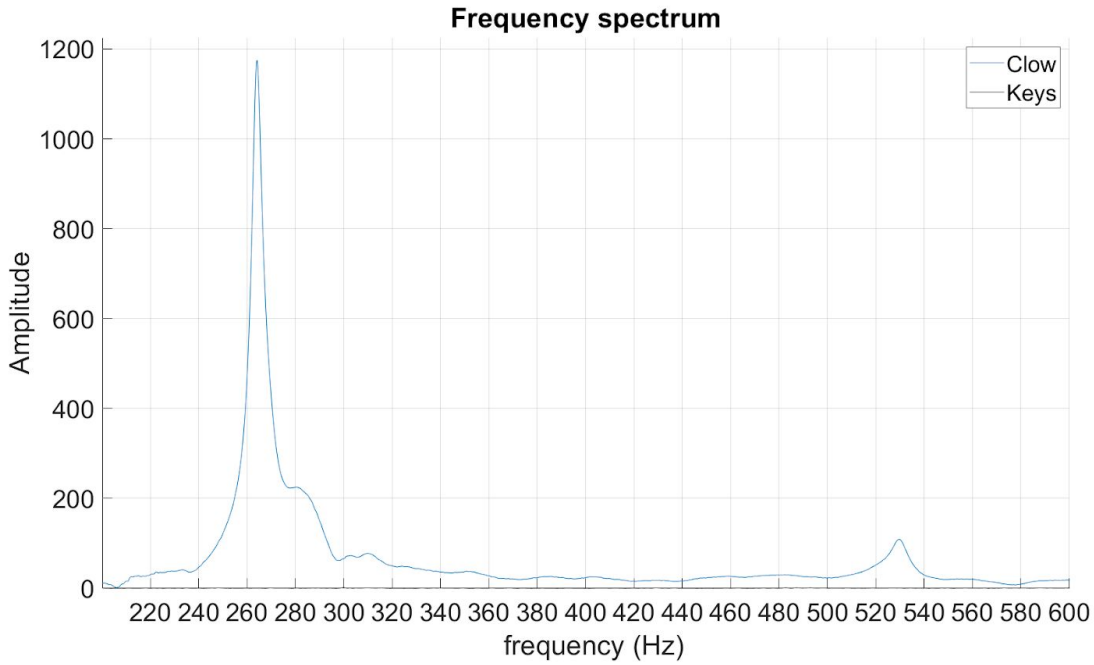


Figure 20: Boomwhacker spectrum and shaken keys spectrum comparison

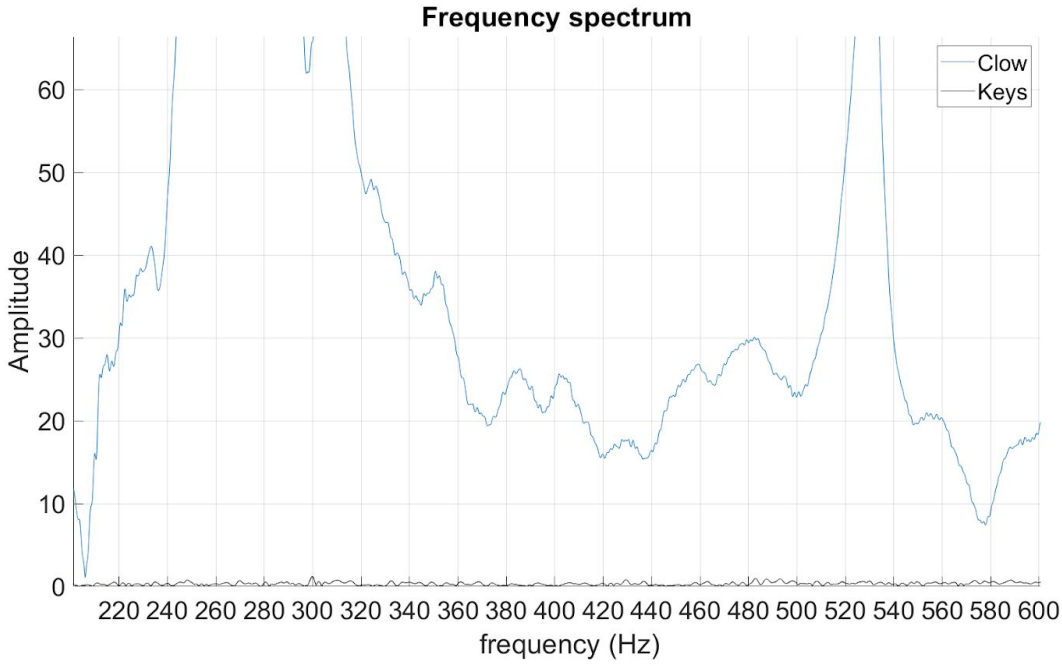


Figure 21: Boomwhacker spectrum and shaken keys spectrum comparison (zoomed)

Yet, if both signals get normalized, the very relevant difference in amplitude is lost (see fig. 22) The noisy sound of the keys, by chance, peaks at ~298 Hz, which is well within the margin of an ideal ‘D’ note (295Hz). The system then proceeds to false trigger a D note.

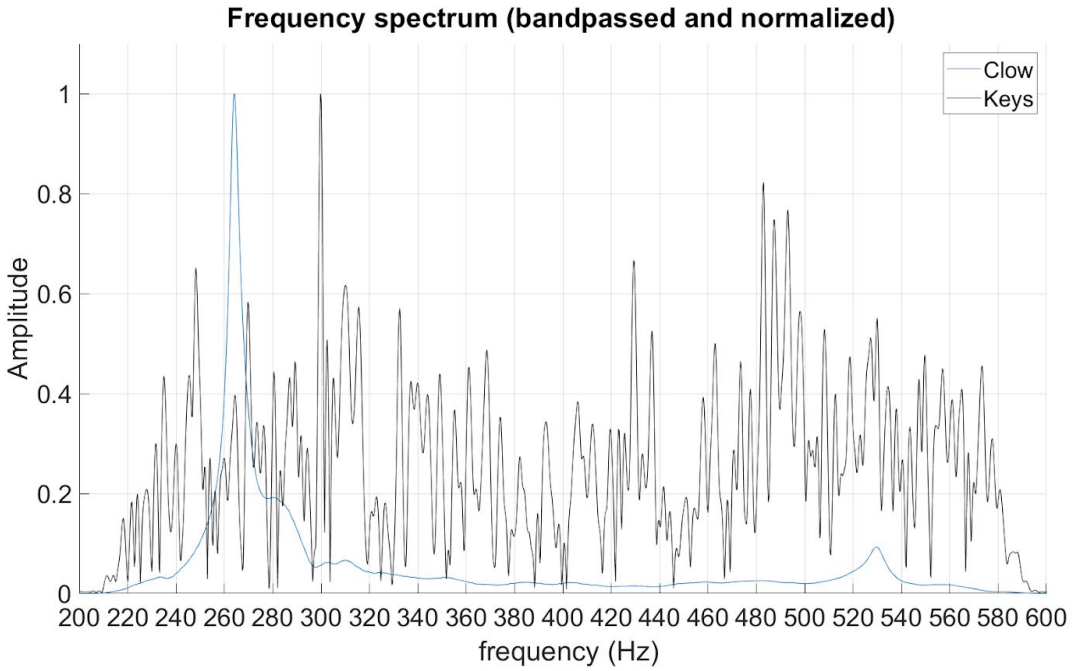


Figure 22: normalisation of a noisy signal that leads to false triggering

Since bandpassing is not necessary, and only tends to degrade effectiveness, it has been removed from the program. Normalisation of the signal is also part of the problem. Although it can be a good method of easily finding the highest peak, it is not necessary to do so.

4.4.3 Peak Based Tone Detection

This section covers the process of developing a peak finding based tone detection algorithm that uses samples from a database as source. The code can be found in Appendix D.

In section 4.3.2. a normalisation based approach to tone detection has been suggested. And although its live accuracy was good (up to 98.8%), normalisation inherently limits this method to detection of only a single tone. While in reality, multiple boomwhackers are often played simultaneously. In order to be able to detect multiple boomwhackers, multiple peaks have to be acquired at the same time. A schematic overview of the developed methodology is shown in figure 23.

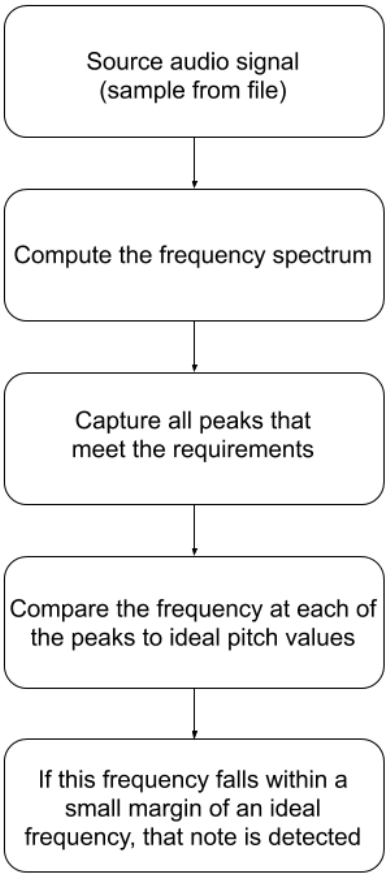


Figure 23: Schematic overview of the peak based tone detection mechanism

A peak finding functionality (`findpeaks()`) within matlab is used in order to find peaks in a signal. A peak will be detected only if it complies with a number of requirements:

1. *The peak's amplitude must be above a certain threshold*
   This prevents small peaks from noise and other artefacts to be detected

2. *A peak cannot be within a small frequency range of another peak*
   Prevents peaks in the 'bleed' (see 4.1) around the fundamental frequency of being detected. Between the 8 tested notes, the smallest frequency difference of pitch is 19.6 Hz (see table 3). This is the maximum possible margin that can be set, without obscuring detection of the neighbouring tone. However, it is recommended to set the margin smaller since this is more forgiving for notes that do not exactly match their ideal pitch.

3. *For n tones, a maximum of n detected peaks is allowed (in this case: 8)*
   If more than 8 peaks are detected, there will at least be n-8 false positives among them. Note: if the previous requirements are tuned properly, a peak limit can be omitted.

This approach seems to yield promising results: a complex signal consisting of the sum of all tones (artificial version of: played at the same time) is given as an example in figure 24.

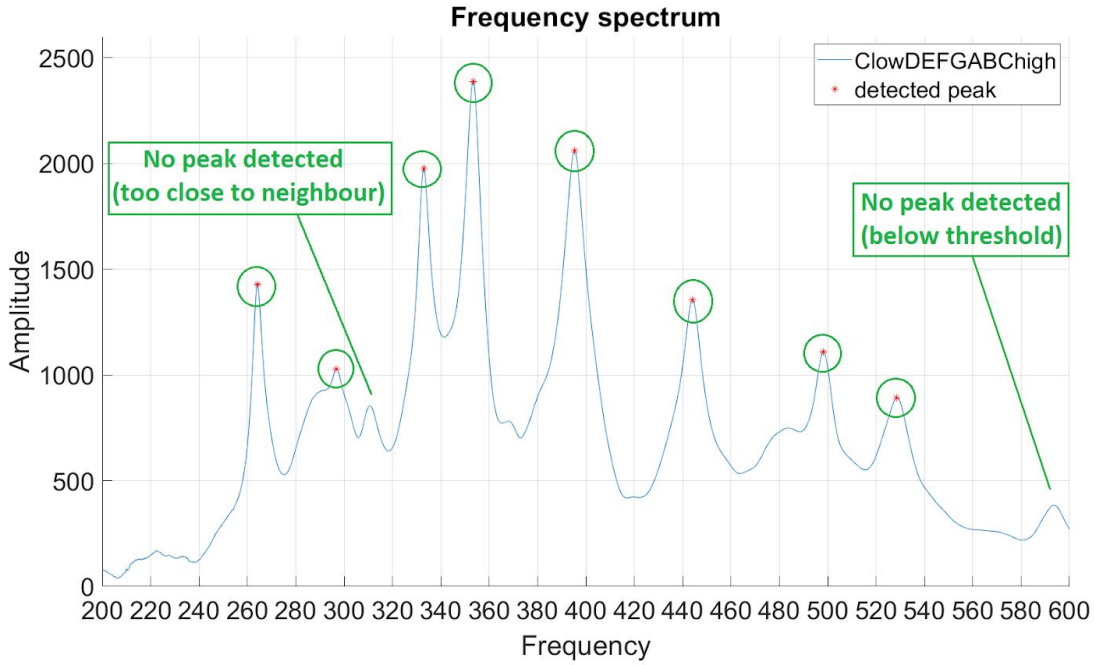


Figure 24: Well performing peak based tone detection on a complex input signal

At similar loudness peak based tone detection works well, even for many simultaneous tones. However, for tones at different volumes, a problem rises. A combination of a Clow note at 0dB and a F note at -6dB is used as an example (see fig. 25). Important to note is that, despite its rather small volume, the F is still well audible together with the Clow.

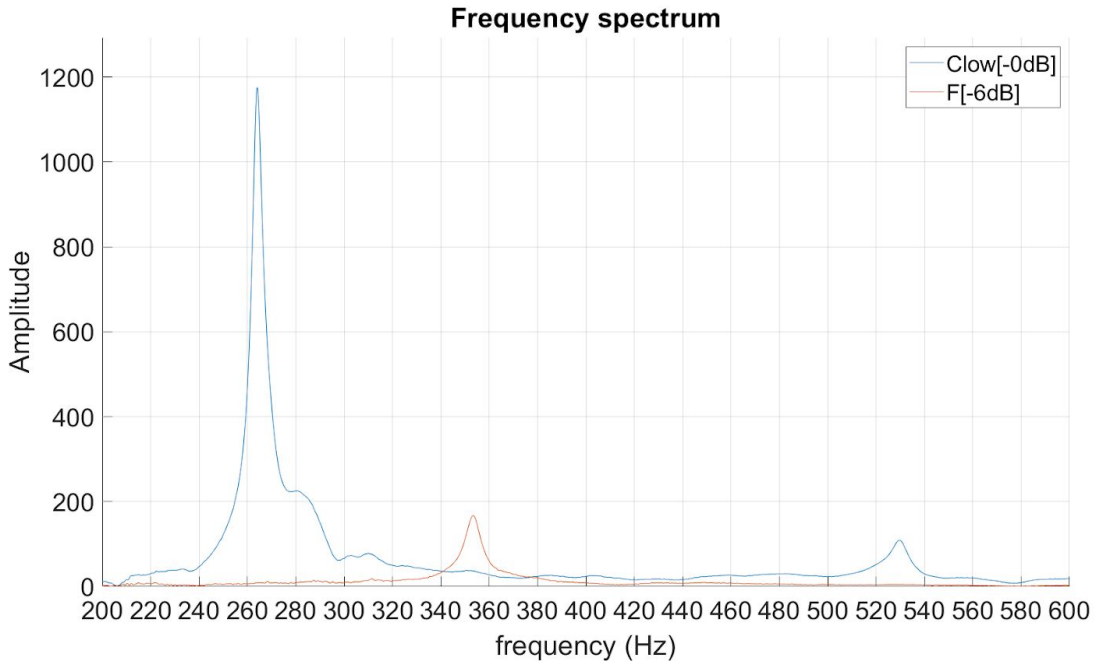


Figure 25: Frequency spectra of Clow [0dB] and F [-6dB]

If such a combination of tones and volumes would occur, which is not unlikely in a classroom with small children using the instruments, a fixed threshold becomes problematic. This phenomenon is illustrated in figure 26 and 27.

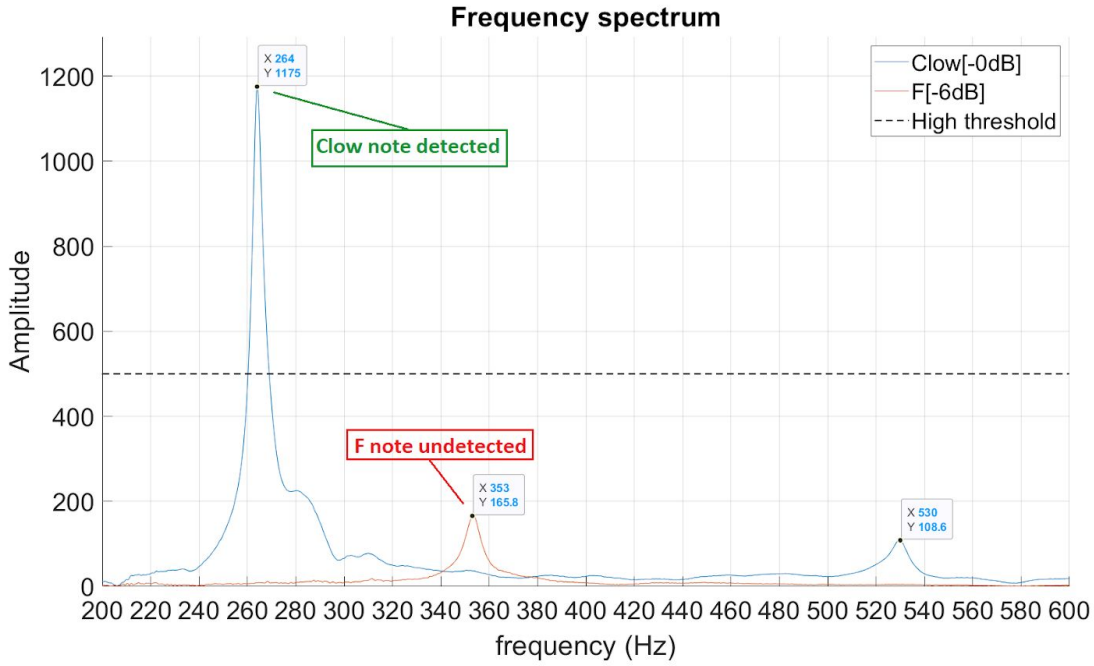


Figure 26: Problem of having a fixed threshold that is too high

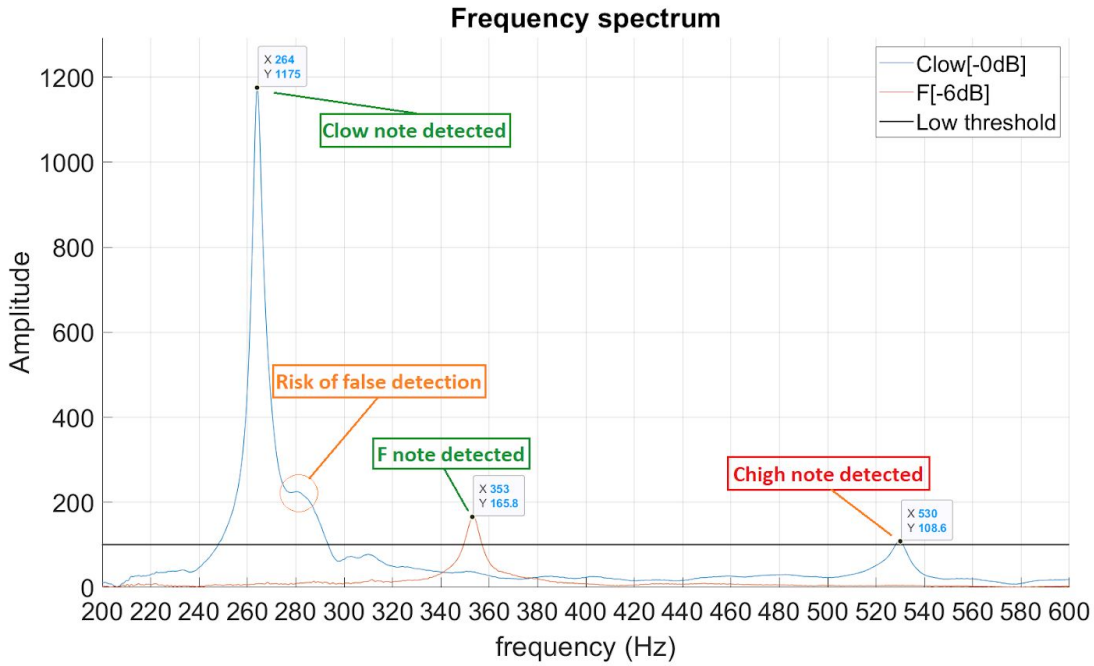


Figure 27: Problem of having a fixed threshold that is too low

A threshold too high fails to capture tones with lower peak values. If the threshold is set too low, harmonic peaks will be detected as separate tones. On top of this, the bleed from high peaks risk detection of peaks. In this situation, this is fortunately still accounted for by the minimum peak distance requirement.

Discussion and conclusion of the exact performance of peak based tone detection is discussed thoroughly in section 4.4.6. Anyway, it is clear that peak based tone detection has limitations. A solution to the illustrated volume difference problem is proposed in the following section (4.4.4).

4.4.4 Subtraction Based Tone Detection

This section covers the process of developing a subtraction based tone detection algorithm that uses samples from a database as source. The code can be found in Appendix D.
A schematic representation of the algorithm is provided in figure 28.

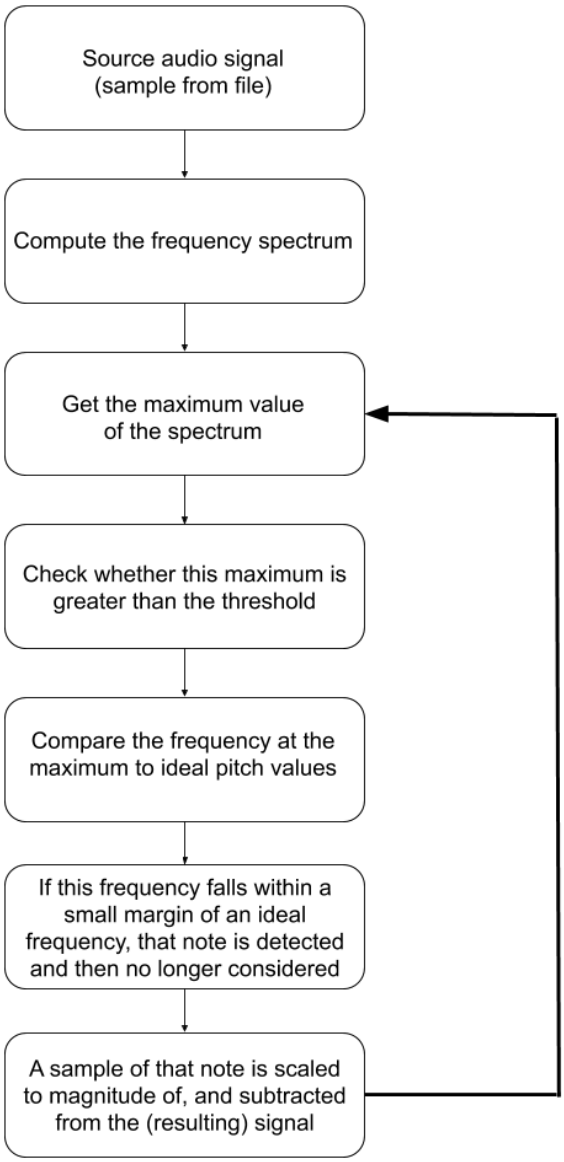


Figure 28: Schematic of the subtraction based tone detection mechanism

Subtraction based tone detection is an iterative program. Instead of trying to detect all peaks at once, every iteration only considers the absolute maximum of the signal. This maximum is compared against ideal frequency values, and tones are (again) detected if the frequency falls within a small margin around the ideal frequency. When a tone is detected, this tone will no longer be considered in future iterations (since it has already been determined to make up part of the signal). Additionally, a clean sample of the detected tone will then be scaled to, and subtracted from the signal.

This subtraction is supposed to cancel out the frequency content of that specific tone within the total spectrum. Inevitably, this cancellation will not be perfect since every recording is slightly different. Nevertheless, the hypothesis is that it will be sufficient in cancelling out enough energy so that a soft(er) tones' pitch will become the new maximum of the signal. This new signal is fed back into the algorithm, so that additional peaks can then be found.
The program finishes when either eight iterations are done (maximum possible number of tones), or the remaining signal has a peak value lower than a soft threshold. This soft threshold is required in order to avoid that very small background noise(s) will keep the program running by accidentally peaking at frequencies within pitch margins. This phenomenon is similar to, and thoroughly explained in the shaken keys problem of 4.4.2.

4.4.5 Result matrices of the different tone detection methods

The performances of the different tone detection methods over the dataset are evaluated by, for each method separately, logging the detections in a matrix (as a '1') and then comparing this matrix against a matrix that contains the correct 'answers' for every input sample. The latter matrix is manually put together and can be found in Appendix E. The database provides 130 input samples, which encompass a total of 250 tones. The rather large raw matrices on themselves can be quite overwhelming and unclear. For this reason, a color scheme is added in order to increase the clarity. Concerning the detection of tones:

- Green indicates that tones in the sample have been detected correctly
- Orange indicates that tones in the sample have partially een detected correctly (fe. two of three notes from a chord)
- Red indicates faulty detection:
    - no detection of a note in the sample (when it should)
    - detection of a wrong note in the sample
    - failing to detect two or more tones in the sample (for signals with three or more tones).

Very important note*:* the color scheme cannot be seen as a universal measure of performance, it is not always representative of how well a tone detection method might work. The same is true for the total accuracy scores that are given. The actual performance of a tone detection method depends greatly on the goal it is expected to achieve. For example, the normalisation based tone detection will score rather poorly on any samples containing multiple tones. This visually results in big red planes in the matrix, and will decrease its percentual score by a lot. Although, if one were to apply normalisation based tone detection in a situation where only single tones have to be detected, the performance will be great. In addition, it can be expected that normalisation based tone detection will have a lot of orange color on inputs with two tones, yet inherent to this algorithm is that it will never be able to detect two tones. In contrast, peak based and subtraction based tone detection do have this capability. There is a nuance in simply not being suited for such signals and, perhaps, barely missing out on a correct detection. Furthermore, an interpretation of the data for each method can be found below the result matrix.

4.4.5.1 Result matrix of Normalisation Based Tone detection

Category description	Index	Input sample	Detected tone						
			Clow	D	E	F	G	A	B	Chigh
Single notes at different volumes	1	Clow [-0dB].wav	1	0	0	0	0	0	0	0
	2	Clow [-3dB].wav	1	0	0	0	0	0	0	0
	3	Clow [-6dB].wav	1	0	0	0	0	0	0	0
	4	D [-0dB].wav	0	1	0	0	0	0	0	0
	5	D [-3dB].wav	0	1	0	0	0	0	0	0
	6	D [-6dB].wav	0	1	0	0	0	0	0	0
	7	E [-0dB].wav	0	0	1	0	0	0	0	0
	8	E [-3dB].wav	0	0	1	0	0	0	0	0
	9	E [-6dB].wav	0	0	1	0	0	0	0	0
	10	F [-0dB].wav	0	0	0	1	0	0	0	0
	11	F [-3dB].wav	0	0	0	1	0	0	0	0
	12	F [-6dB].wav	0	0	0	1	0	0	0	0
	13	G [-0dB].wav	0	0	0	0	1	0	0	0
	14	G [-3dB].wav	0	0	0	0	1	0	0	0
	15	G [-6dB].wav	0	0	0	0	1	0	0	0
	16	A [-0dB].wav	0	0	0	0	0	1	0	0
	17	A [-3dB].wav	0	0	0	0	0	1	0	0
	18	A [-6dB].wav	0	0	0	0	0	1	0	0
	19	B [-0dB].wav	0	0	0	0	0	0	1	0
	20	B [-3dB].wav	0	0	0	0	0	0	1	0
	21	B [-6dB].wav	0	0	0	0	0	0	1	0
	22	Chigh [-0dB].wav	0	0	0	0	0	0	0	1
	23	Chigh [-3dB].wav	0	0	0	0	0	0	0	1
	24	Chigh [-6dB].wav	0	0	0	0	0	0	0	1
Alternate (random) samples	25	hoi'.wav	0	0	0	0	0	0	0	0
	26	A4 virtual piano note.wav	0	0	0	0	0	1	0	0
	27	Bang on a bucket.wav	0	0	0	0	0	0	0	0
	28	Hand Clap.wav	0	0	0	0	0	0	0	0
	29	Keys shaking.wav	0	0	0	0	0	0	0	0
	30	Metal on metal tick.wav	0	0	0	0	0	0	0	0
	31	Snap of fingers.wav	0	0	0	0	0	0	0	0
Clow + tone combination	32	ClowD [-0dB].wav	1	0	0	0	0	0	0	0
	33	ClowE [-0dB].wav	0	0	1	0	0	0	0	0
	34	ClowF [-0dB].wav	0	0	0	1	0	0	0	0
	35	ClowG [-0dB].wav	0	0	0	0	1	0	0	0
	36	ClowA [-0dB].wav	1	0	0	0	0	0	0	0
	37	ClowB [-0dB].wav	1	0	0	0	0	0	0	0
	38	ClowChigh [-0dB].wav	1	0	0	0	0	0	0	0
Different recording of notes	39	Clow2 [-0dB].wav	1	0	0	0	0	0	0	0
	40	D2 [-0dB].wav	0	1	0	0	0	0	0	0
	41	E2 [-0dB].wav	0	0	1	0	0	0	0	0
	42	F2 [-0dB].wav	0	0	0	1	0	0	0	0
	43	G2 [-0dB].wav	0	0	0	0	1	0	0	0
	44	A2 [-0dB].wav	0	0	0	0	0	1	0	0
	45	B2 [-0dB].wav	0	0	0	0	0	0	1	0
	46	Chigh2 [-0dB].wav	0	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes	47	ClowD[-0dB]	1	0	0	0	0	0	0	0
	48	EF[-0dB]	0	0	0	1	0	0	0	0
	49	BChigh[-0dB]	0	0	0	0	0	0	1	0
EC: Overlapping frequency content	50	ClowG[-0dB]	0	0	0	0	1	0	0	0
	51	DA[-0dB]	0	0	0	0	0	1	0	0
	52	EB[-0dB]	0	0	1	0	0	0	0	0
	53	ClowChigh[-0dB]	1	0	0	0	0	0	0	0
Chords	54	ClowEG[-0dB](Cmaj)	0	0	1	0	0	0	0	0
	55	FAChigh[-0dB](Fmaj)	0	0	0	1	0	0	0	0
	56	GBD[-0dB](Gmaj)	0	0	0	0	1	0	0	0
	57	DFA[-0dB](Dmin)	0	0	0	1	0	0	0	0
	58	EGB[-0dB](Emin)	0	0	1	0	0	0	0	0
	59	AChighE(Amin)	0	0	1	0	0	0	0	0
All tones	60	ClowDEFGABChigh[-0dB]	0	0	0	1	0	0	0	0
EC: Lowest absolute frequency difference between notes (3dB volume difference)	61	Clow[-0dB]D[-3dB]	1	0	0	0	0	0	0	0
	62	Clow[-3dB]D[-0dB]	0	1	0	0	0	0	0	0
	63	E[-0dB]F[-3dB]	0	0	1	0	0	0	0	0
	64	E[-3dB]F[-0dB]	0	0	0	1	0	0	0	0
	65	B[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	0
	66	B[-3dB]Chigh[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (3dB volume difference)	67	Clow[-0dB]G[-3dB]	1	0	0	0	0	0	0	0
	68	Clow[-3dB]G[-0dB]	0	0	0	0	1	0	0	0
	69	D[-0dB]A[-3dB]	0	1	0	0	0	0	0	0
	70	D[-3dB]A[-0dB]	0	0	0	0	0	1	0	0
	71	E[-0dB]B[-3dB]	0	0	1	0	0	0	0	0
	72	E[-3dB]B[-0dB]	0	0	0	0	0	0	1	0
	73	Clow[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	0
	74	Clow[-3dB]Chigh[-0dB]	0	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference)	75	Clow[-0dB]D[-6dB]	1	0	0	0	0	0	0	0
	76	Clow[-6dB]D[-0dB]	0	1	0	0	0	0	0	0
	77	E[-0dB]F[-6dB]	0	0	1	0	0	0	0	0
	78	E[-6dB]F[-0dB]	0	0	0	1	0	0	0	0
	79	B[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	0
	80	B[-6dB]Chigh[-0dB]	0	0	0	0	0	0	0	1

EC: Overlapping frequency content (3dB volume difference)	81	Clow[-0dB]G[-6dB]	1	0	0	0	0	0	0	0
	82	Clow[-6dB]G[-0dB]	0	0	0	0	1	0	0	0
	83	D[-0dB]A[-6dB]	0	1	0	0	0	0	0	0
	84	D[-6dB]A[-0dB]	0	0	0	0	0	1	0	0
	85	E[-0dB]B[-6dB]	0	0	1	0	0	0	0	0
	86	E[-6dB]B[-0dB]	0	0	0	0	0	0	1	0
	87	Clow[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	0
	88	Clow[-6dB]Chigh[-0dB]	0	0	0	0	0	0	0	1
Chords (3dB/6dB volume difference)	89	Clow[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	0	0	0	0	0	0
	90	F[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	0	0	0
	91	G[-0dB]B[-3dB]D[-6dB](Gmaj)	0	0	0	0	1	0	0	0
	92	D[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	0	0	0	0	0
	93	E[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	0	0	0	0
	94	A[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	0	0	0	1	0	0
All tones	95	ClowDEFGABChigh[-0/-3/-6dB]	0	0	0	1	0	0	0	0
EC: Lowest absolute frequency difference between notes (3dB volume difference, different 0dB note recording)	96	Clow2[-0dB]D[-3dB]	1	0	0	0	0	0	0	0
	97	Clow[-3dB]D2[-0dB]	0	1	0	0	0	0	0	0
	98	E2[-0dB]F[-3dB]	0	0	1	0	0	0	0	0
	99	E[-3dB]F2[-0dB]	0	0	0	1	0	0	0	0
	100	B2[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	0
	101	B[-3dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (3dB volume difference different 0dB note recording)	102	Clow2[-0dB]G[-3dB]	1	0	0	0	0	0	0	0
	103	Clow[-3dB]G2[-0dB]	0	0	0	0	1	0	0	0
	104	D2[-0dB]A[-3dB]	0	1	0	0	0	0	0	0
	105	D[-3dB]A2[-0dB]	0	0	0	0	0	1	0	0
	106	E2[-0dB]B[-3dB]	0	0	1	0	0	0	0	0
	107	E[-3dB]B2[-0dB]	0	0	0	0	0	0	1	0
	108	Clow2[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	0
	109	Clow[-3dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference, different 0dB note recording)	110	Clow2[0dB]D[-6dB]	1	0	0	0	0	0	0	0
	111	Clow[-6dB]D2[-0dB]	0	1	0	0	0	0	0	0
	112	E2[-0dB]F[-6dB]	0	0	1	0	0	0	0	0
	113	E[-6dB]F2[-0dB]	0	0	0	1	0	0	0	0
	114	B2[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	0
	115	B[-6dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (6dB volume difference, different 0dB note recording)	116	Clow2[-0dB]G[-6dB]	1	0	0	0	0	0	0	0
	117	Clow[-6dB]G2[-0dB]	0	1	0	0	0	0	0	0
	118	D2[-0dB]A[-6dB]	0	1	0	0	0	0	0	0
	119	D[-6dB]A2[-0dB]	0	0	0	0	0	1	0	0
	120	E2[-0dB]B[-6dB]	0	0	1	0	0	0	0	0
	121	E[-6dB]B2[-0dB]	0	0	0	0	0	0	1	0
	122	Clow2[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	0
	123	Clow[-6dB]Chigh2[-0dB]	0	1	0	0	0	0	0	0
Chords (3dB/6dB volume difference, different 0dB note recording)	124	Clow2[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	0	0	0	0	0	0
	125	F2[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	0	0	0
	126	G2[-0dB]B[-3dB]D[-6dB](Gmaj)	0	0	0	0	1	0	0	0
	127	D2[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	0	0	0	0	0
	128	E2[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	0	0	0	0
	129	A2[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	0	0	0	1	0	0
All tones (different 0dB note recording)	130	Clow2DEF2GAB2Chigh[-0/-3/-6dB]	1	0	0	0	0	0	0	0

The normalisation based tone detection performs very well on single tones, and does not false trigger on a select number of random samples. An exception to this is the ‘A4 virtual piano note’ (row 26). Considering how the algorithm is set up, it is likely it might trigger on other instruments as well, as long as the strongest harmonic thereof is within ideal pitch range. As expected, performance is rather poor for any input containing more than a one tone. Over the whole database, 119 detections were done correctly. In total there are 250 correct detections possible. This gives a total accuracy score of:

$$(119/250) * 100\% = 47.6\%.$$

By nature, this algorithm is limited to single tone detection. Possibly a more useful percentual score would be how well single tones inputs are detected. In total, there are 32 single tone inputs, of which 32 are detected correctly. For detection of single tones, the percentual score is 100%.

4.3.5.2 Result matrix of Peak Based Tone detection

Category description	Index	Input sample	Detected tone							
			Clow	D	E	F	G	A	B	Chigh
Single notes at different volumes	1	Clow [-0dB].wav	1	0	0	0	0	0	0	0
	2	Clow [-3dB].wav	1	0	0	0	0	0	0	0
	3	Clow [-6dB].wav	0	0	0	0	0	0	0	0
	4	D [-0dB].wav	0	1	0	0	0	0	0	0
	5	D [-3dB].wav	0	0	0	0	0	0	0	0
	6	D [-6dB].wav	0	0	0	0	0	0	0	0
	7	E [-0dB].wav	0	0	1	0	0	0	0	0
	8	E [-3dB].wav	0	0	1	0	0	0	0	0
	9	E [-6dB].wav	0	0	0	0	0	0	0	0
	10	F [-0dB].wav	0	0	0	1	0	0	0	0
	11	F [-3dB].wav	0	0	0	0	0	0	0	0
	12	F [-6dB].wav	0	0	0	0	0	0	0	0
	13	G [-0dB].wav	0	0	0	0	1	0	0	0
	14	G [-3dB].wav	0	0	0	0	1	0	0	0
	15	G [-6dB].wav	0	0	0	0	0	0	0	0
	16	A [-0dB].wav	0	0	0	0	0	1	0	0
	17	A [-3dB].wav	0	0	0	0	0	0	0	0
	18	A [-6dB].wav	0	0	0	0	0	0	0	0
	19	B [-0dB].wav	0	0	0	0	0	0	1	0
	20	B [-3dB].wav	0	0	0	0	0	0	0	0
	21	B [-6dB].wav	0	0	0	0	0	0	0	0
	22	Chigh [-0dB].wav	0	0	0	0	0	0	0	1
	23	Chigh [-3dB].wav	0	0	0	0	0	0	0	0
	24	Chigh [-6dB].wav	0	0	0	0	0	0	0	0
Alternate (random) samples	25	hoi'.wav	0	0	0	0	0	0	0	0
	26	A4 virtual piano note.wav	0	0	0	0	0	0	0	0
	27	Bang on a bucket.wav	0	0	0	0	0	0	0	0
	28	Hand Clap.wav	0	0	0	0	0	0	0	0
	29	Keys shaking.wav	0	0	0	0	0	0	0	0
	30	Metal on metal tick.wav	0	0	0	0	0	0	0	0
	31	Snap of fingers.wav	0	0	0	0	0	0	0	0
Clow + tone combination	32	ClowD [-0dB].wav	1	1	0	0	0	0	0	0
	33	ClowE [-0dB].wav	1	0	1	0	0	0	0	0
	34	ClowF [-0dB].wav	1	0	0	1	0	0	0	0
	35	ClowG [-0dB].wav	1	0	0	0	1	0	0	0
	36	ClowA [-0dB].wav	1	0	0	0	0	1	0	0
	37	ClowB [-0dB].wav	1	0	0	0	0	0	1	0
	38	ClowChigh [-0dB].wav	1	0	0	0	0	0	0	1
Different recording of notes	39	Clow2 [-0dB].wav	1	0	0	0	0	0	0	1
	40	D2 [-0dB].wav	0	1	0	0	0	0	0	0
	41	E2 [-0dB].wav	0	0	1	0	0	0	0	0
	42	F2 [-0dB].wav	0	0	0	1	0	0	0	0
	43	G2 [-0dB].wav	0	0	0	0	1	0	0	0
	44	A2 [-0dB].wav	0	0	0	0	0	1	0	0
	45	B2 [-0dB].wav	0	0	0	0	0	0	1	0
	46	Chigh2 [-0dB].wav	0	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes	47	ClowD[-0dB]	1	1	0	0	0	0	0	0
	48	EF[-0dB]	0	0	1	1	0	0	0	0
	49	BChigh[-0dB]	0	0	0	0	0	0	1	1
EC: Overlapping frequency content	50	ClowG[-0dB]	1	0	0	0	1	0	0	0
	51	DA[-0dB]	0	1	0	0	0	1	0	0
	52	EB[-0dB]	0	0	1	0	0	0	1	0
	53	ClowChigh[-0dB]	1	0	0	0	0	0	0	1
Chords	54	ClowEG[-0dB](Cmaj)	1	0	1	0	1	0	0	0
	55	FAChigh[-0dB](Fmaj)	0	0	0	1	0	1	0	1
	56	GBD[-0dB](Gmaj)	0	1	0	0	1	0	1	0
	57	DFA[-0dB](Dmin)	0	1	0	1	0	1	0	0
	58	EGB[-0dB](Emin)	0	0	1	0	1	0	1	0
	59	AChighE(Amin)	0	0	1	0	0	1	0	1
All tones	60	ClowDEFGABChigh[-0dB]	1	1	1	1	1	1	1	1
EC: Lowest absolute frequency difference between notes (3dB volume difference)	61	Clow[-0dB]D[-3dB]	1	0	0	0	0	0	0	0
	62	Clow[-3dB]D[-0dB]	1	1	0	0	0	0	0	0
	63	E[-0dB]F[-3dB]	0	0	1	1	0	0	0	0
	64	E[-3dB]F[-0dB]	0	0	1	1	0	0	0	0
	65	B[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	0
	66	B[-3dB]Chigh[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (3dB volume difference)	67	Clow[-0dB]G[-3dB]	1	0	0	0	1	0	0	0
	68	Clow[-3dB]G[-0dB]	0	0	0	0	1	0	0	0
	69	D[-0dB]A[-3dB]	0	1	0	0	0	0	0	0
	70	D[-3dB]A[-0dB]	0	0	0	0	0	1	0	0
	71	E[-0dB]B[-3dB]	0	0	1	0	0	0	0	0
	72	E[-3dB]B[-0dB]	0	0	1	0	0	0	1	0
	73	Clow[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	0
	74	Clow[-3dB]Chigh[-0dB]	1	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference)	75	Clow[-0dB]D[-6dB]	1	0	0	0	0	0	0	0
	76	Clow[-6dB]D[-0dB]	0	1	0	0	0	0	0	0
	77	E[-0dB]F[-6dB]	0	0	1	0	0	0	0	0
	78	E[-6dB]F[-0dB]	0	0	0	1	0	0	0	0
	79	B[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	0
	80	B[-6dB]Chigh[-0dB]	0	0	0	0	0	0	0	1

EC: Overlapping frequency content (3dB volume difference)	81	Clow[-0dB]G[-6dB]	1	0	0	0	0	0	0	0
	82	Clow[-6dB]G[-0dB]	0	0	0	0	1	0	0	0
	83	D[-0dB]A[-6dB]	0	1	0	0	0	0	0	0
	84	D[-6dB]A[-0dB]	0	0	0	0	0	1	0	0
	85	E[-0dB]B[-6dB]	0	0	1	0	0	0	0	0
	86	E[-6dB]B[-0dB]	0	0	0	0	0	0	1	0
	87	Clow[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	0
	88	Clow[-6dB]Chigh[-0dB]	0	0	0	0	0	0	0	1
Chords (3dB/6dB volume difference)	89	Clow[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	0	0	0	0	0	0
	90	F[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	0	0	0
	91	G[-0dB]B[-3dB]D[-6dB](Gmaj)	0	0	0	0	1	0	0	0
	92	D[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	0	0	0	0	0
	93	E[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	1	0	0	0
	94	A[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	0	0	0	1	0	0
All tones	95	ClowDEFGABChigh[-0/-3/-6dB]	1	0	0	1	1	0	1	0
EC: Lowest absolute frequency difference between notes (3dB volume difference, different 0dB note recording)	96	Clow2[-0dB]D[-3dB]	1	0	0	0	0	0	0	1
	97	Clow[-3dB]D2[-0dB]	1	1	0	0	0	0	0	0
	98	E2[-0dB]F[-3dB]	0	0	1	0	0	0	0	0
	99	E[-3dB]F2[-0dB]	0	0	0	1	0	0	0	0
	100	B2[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	0
	101	B[-3dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (3dB volume difference different 0dB note recording)	102	Clow2[-0dB]G[-3dB]	1	0	0	0	1	0	0	1
	103	Clow[-3dB]G2[-0dB]	1	0	0	0	1	0	0	0
	104	D2[-0dB]A[-3dB]	0	1	0	0	0	0	0	0
	105	D[-3dB]A2[-0dB]	0	0	0	0	0	1	0	0
	106	E2[-0dB]B[-3dB]	0	0	1	0	0	0	0	0
	107	E[-3dB]B2[-0dB]	0	0	1	0	0	0	1	0
	108	Clow2[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	1
	109	Clow[-3dB]Chigh2[-0dB]	1	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference, different 0dB note recording)	110	Clow2[0dB]D[-6dB]	1	0	0	0	0	0	0	1
	111	Clow[-6dB]D2[-0dB]	0	1	0	0	0	0	0	0
	112	E2[-0dB]F[-6dB]	0	0	1	0	0	0	0	0
	113	E[-6dB]F2[-0dB]	0	0	0	1	0	0	0	0
	114	B2[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	0
	115	B[-6dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (6dB volume difference, different 0dB note recording)	116	Clow2[-0dB]G[-6dB]	1	0	0	0	0	0	0	1
	117	Clow[-6dB]G2[-0dB]	0	1	0	0	1	0	0	0
	118	D2[-0dB]A[-6dB]	0	1	0	0	0	0	0	0
	119	D[-6dB]A2[-0dB]	0	0	0	0	0	1	0	0
	120	E2[-0dB]B[-6dB]	0	0	1	0	0	0	0	0
	121	E[-6dB]B2[-0dB]	0	0	0	0	0	0	1	0
	122	Clow2[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	1
	123	Clow[-6dB]Chigh2[-0dB]	0	1	0	0	0	0	0	1
Chords (3dB/6dB volume difference, different 0dB note recording)	124	Clow2[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	0	0	0	0	0	1
	125	F2[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	0	0	0
	126	G2[-0dB]B[-3dB]D[-6dB](Gmaj)	0	0	0	0	1	0	0	0
	127	D2[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	0	0	0	0	0
	128	E2[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	1	0	0	0
	129	A2[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	0	0	0	1	0	0
All tones (different 0dB note recording)	130	Clow2DEF2GAB2Chigh[-0/-3/-6dB]	1	0	0	1	1	0	1	1

For the peak based tone detection, the effect of thresholding is clearly visible: tones with lower volumes are detected much less compared to tones at 0dB, which are detected well. Even for complex combined tones the algorithm scores very well when all components are high enough in volume. Another thing worth noting is the fact that sometimes the 1st harmonic of Clow (which is a peak at Chigh), is sometimes high enough in amplitude for an additional detection of Chigh. See for example row 39. In total, 157 out of 250 tones were detected correctly. The total accuracy score amounts to:

$$(157/250) * 100\% = 62.8\%.$$

4.4.5.3 Result matrix of Subtraction Based Tone detection

Category description	Index	Input sample	Detected tone							
			Clow	D	E	F	G	A	B	Chigh
Single notes at different volumes	1	Clow [-0dB].wav	1	0	0	0	0	0	0	0
	2	Clow [-3dB].wav	1	0	0	0	0	0	0	0
	3	Clow [-6dB].wav	1	0	0	0	0	0	0	0
	4	D [-0dB].wav	0	1	0	0	0	0	0	0
	5	D [-3dB].wav	0	1	0	0	0	0	0	0
	6	D [-6dB].wav	0	1	0	0	0	0	0	0
	7	E [-0dB].wav	0	0	1	0	0	0	0	0
	8	E [-3dB].wav	0	0	1	0	0	0	0	0
	9	E [-6dB].wav	0	0	1	0	0	0	0	0
	10	F [-0dB].wav	0	0	0	1	0	0	0	0
	11	F [-3dB].wav	0	0	0	1	0	0	0	0
	12	F [-6dB].wav	0	0	0	1	0	0	0	0
	13	G [-0dB].wav	0	0	0	0	1	0	0	0
	14	G [-3dB].wav	0	0	0	0	1	0	0	0
	15	G [-6dB].wav	0	0	0	0	1	0	0	0
	16	A [-0dB].wav	0	0	0	0	0	1	0	0
	17	A [-3dB].wav	0	0	0	0	0	1	0	0
	18	A [-6dB].wav	0	0	0	0	0	1	0	0
	19	B [-0dB].wav	0	0	0	0	0	0	1	0
	20	B [-3dB].wav	0	0	0	0	0	0	1	0
	21	B [-6dB].wav	0	0	0	0	0	0	1	0
	22	Chigh [-0dB].wav	0	0	0	0	0	0	0	1
	23	Chigh [-3dB].wav	0	0	0	0	0	0	0	1
	24	Chigh [-6dB].wav	0	0	0	0	0	0	0	1
Alternate (random) samples	25	hoi'.wav	0	0	0	0	0	0	0	0
	26	A4 virtual piano note.wav	0	0	0	0	0	1	0	0
	27	Bang on a bucket.wav	0	0	0	0	0	0	0	0
	28	Hand Clap.wav	0	0	0	0	0	0	0	0
	29	Keys shaking.wav	0	0	0	0	0	0	0	0
	30	Metal on metal tick.wav	0	0	0	0	0	0	0	0
	31	Snap of fingers.wav	0	0	0	0	0	0	0	0
Clow + tone combination	32	ClowD [-0dB].wav	1	1	0	0	0	0	0	0
	33	ClowE [-0dB].wav	1	0	1	0	0	0	0	0
	34	ClowF [-0dB].wav	1	0	0	1	0	0	0	0
	35	ClowG [-0dB].wav	1	0	0	0	1	0	0	0
	36	ClowA [-0dB].wav	1	0	0	0	0	1	0	0
	37	ClowB [-0dB].wav	1	0	0	0	0	0	1	0
	38	ClowChigh [-0dB].wav	1	0	0	0	0	0	0	1
Different recording of notes	39	Clow2 [-0dB].wav	1	0	0	0	0	0	0	1
	40	D2 [-0dB].wav	0	1	0	0	0	0	0	0
	41	E2 [-0dB].wav	0	0	1	0	0	0	0	0
	42	F2 [-0dB].wav	0	0	0	1	0	0	0	0
	43	G2 [-0dB].wav	0	0	0	0	1	0	0	0
	44	A2 [-0dB].wav	0	0	0	0	0	1	0	0
	45	B2 [-0dB].wav	0	0	0	0	0	0	1	0
	46	Chigh2 [-0dB].wav	0	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes	47	ClowD[-0dB]	1	1	0	0	0	0	0	0
	48	EF[-0dB]	0	0	1	1	0	0	0	0
	49	BChigh[-0dB]	0	0	0	0	0	0	1	1
EC: Overlapping frequency content	50	ClowG[-0dB]	1	0	0	0	1	0	0	0
	51	DA[-0dB]	0	1	0	0	0	1	0	0
	52	EB[-0dB]	0	0	1	0	0	0	1	0
	53	ClowChigh[-0dB]	1	0	0	0	0	0	0	1
Chords	54	ClowEG[-0dB](Cmaj)	1	0	1	0	1	0	0	0
	55	FAChigh[-0dB](Fmaj)	0	0	0	1	0	1	0	1
	56	GBD[-0dB](Gmaj)	0	1	0	0	1	0	1	0
	57	DFA[-0dB](Dmin)	0	1	0	1	0	1	0	0
	58	EGB[-0dB](Emin)	0	0	1	0	1	0	1	0
	59	AChighE(Amin)	0	0	1	0	0	1	0	1
All tones	60	ClowDEFGABChigh[-0dB]	1	1	1	1	1	1	1	1
EC: Lowest absolute frequency difference between notes (3dB volume difference)	61	Clow[-0dB]D[-3dB]	1	1	0	0	0	0	0	0
	62	Clow[-3dB]D[-0dB]	1	1	0	0	0	0	0	0
	63	E[-0dB]F[-3dB]	0	0	1	1	0	0	0	0
	64	E[-3dB]F[-0dB]	0	0	1	1	0	0	0	0
	65	B[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	1
	66	B[-3dB]Chigh[-0dB]	0	0	0	0	0	0	1	1
EC: Overlapping frequency content (3dB volume difference)	67	Clow[-0dB]G[-3dB]	1	0	0	0	1	0	0	0
	68	Clow[-3dB]G[-0dB]	1	0	0	0	1	0	0	0
	69	D[-0dB]A[-3dB]	0	1	0	0	0	1	0	0
	70	D[-3dB]A[-0dB]	0	1	0	0	0	1	0	0
	71	E[-0dB]B[-3dB]	0	0	1	0	0	0	1	0
	72	E[-3dB]B[-0dB]	0	0	1	0	0	0	1	0
	73	Clow[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	1
	74	Clow[-3dB]Chigh[-0dB]	1	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference)	75	Clow[-0dB]D[-6dB]	1	1	0	0	0	0	0	0
	76	Clow[-6dB]D[-0dB]	1	1	0	0	0	0	0	0
	77	E[-0dB]F[-6dB]	0	0	1	1	0	0	0	0
	78	E[-6dB]F[-0dB]	0	0	1	1	0	0	0	0
	79	B[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	1
	80	B[-6dB]Chigh[-0dB]	0	0	0	0	0	0	1	1

EC: Overlapping frequency content (3dB volume difference)	81	Clow[-0dB]G[-6dB]	1	0	0	0	1	0	0	0
	82	Clow[-6dB]G[-0dB]	1	0	0	0	1	0	0	0
	83	D[-0dB]A[-6dB]	0	1	0	0	0	1	0	0
	84	D[-6dB]A[-0dB]	0	1	0	0	0	1	0	0
	85	E[-0dB]B[-6dB]	0	0	1	0	0	0	1	0
	86	E[-6dB]B[-0dB]	0	0	1	0	0	0	1	0
	87	Clow[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	1
	88	Clow[-6dB]Chigh[-0dB]	1	0	0	0	0	0	0	1
Chords (3dB/6dB volume difference)	89	Clow[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	1	0	1	0	0	0
	90	F[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	1	0	1
	91	G[-0dB]B[-3dB]D[-6dB](Gmaj)	0	1	0	0	1	0	1	0
	92	D[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	1	0	1	0	0
	93	E[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	1	0	1	0
	94	A[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	1	0	0	1	0	1
All tones	95	ClowDEFGABChigh[-0/-3/-6dB]	1	1	1	1	1	1	1	1
EC: Lowest absolute frequency difference between notes (3dB volume difference, different 0dB note recording)	96	Clow2[-0dB]D[-3dB]	1	0	0	0	0	0	0	1
	97	Clow[-3dB]D2[-0dB]	1	1	0	0	0	0	0	0
	98	E2[-0dB]F[-3dB]	0	0	1	0	0	0	0	0
	99	E[-3dB]F2[-0dB]	0	0	1	1	0	0	0	0
	100	B2[-0dB]Chigh[-3dB]	0	0	0	0	0	0	1	0
	101	B[-3dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (3dB volume difference different 0dB note recording)	102	Clow2[-0dB]G[-3dB]	1	0	0	0	1	0	0	1
	103	Clow[-3dB]G2[-0dB]	1	0	0	0	1	0	0	0
	104	D2[-0dB]A[-3dB]	0	1	0	0	0	1	0	0
	105	D[-3dB]A2[-0dB]	0	1	0	0	0	1	0	0
	106	E2[-0dB]B[-3dB]	0	0	1	0	0	0	0	0
	107	E[-3dB]B2[-0dB]	0	0	1	0	0	0	1	0
	108	Clow2[-0dB]Chigh[-3dB]	1	0	0	0	0	0	0	1
	109	Clow[-3dB]Chigh2[-0dB]	1	0	0	0	0	0	0	1
EC: Lowest absolute frequency difference between notes (6dB volume difference, different 0dB note recording)	110	Clow2[0dB]D[-6dB]	1	0	0	0	0	0	0	1
	111	Clow[-6dB]D2[-0dB]	0	1	0	0	0	0	0	0
	112	E2[-0dB]F[-6dB]	0	0	1	0	0	0	0	0
	113	E[-6dB]F2[-0dB]	0	0	1	1	0	0	0	0
	114	B2[-0dB]Chigh[-6dB]	0	0	0	0	0	0	1	0
	115	B[-6dB]Chigh2[-0dB]	0	0	0	0	0	0	0	1
EC: Overlapping frequency content (6dB volume difference, different 0dB note recording)	116	Clow2[-0dB]G[-6dB]	1	0	0	0	0	0	0	1
	117	Clow[-6dB]G2[-0dB]	0	1	0	0	1	0	0	0
	118	D2[-0dB]A[-6dB]	0	1	0	0	0	0	0	0
	119	D[-6dB]A2[-0dB]	0	0	0	0	0	1	0	0
	120	E2[-0dB]B[-6dB]	0	0	1	0	0	0	0	0
	121	E[-6dB]B2[-0dB]	0	0	0	0	0	0	1	0
	122	Clow2[-0dB]Chigh[-6dB]	1	0	0	0	0	0	0	1
	123	Clow[-6dB]Chigh2[-0dB]	0	1	0	0	0	0	0	1
Chords (3dB/6dB volume difference, different 0dB note recording)	124	Clow2[-0dB]E[-3dB]G[-6dB](Cmaj)	1	0	0	0	0	0	0	1
	125	F2[-0dB]A[-3dB]Chigh[-6dB](Fmaj)	0	0	0	1	0	1	0	0
	126	G2[-0dB]B[-3dB]D[-6dB](Gmaj)	0	0	0	0	1	0	0	0
	127	D2[-0dB]F[-3dB]A[-6dB](Dmin)	0	1	0	1	0	1	0	0
	128	E2[-0dB]G[-3dB]B[-6dB](Emin)	0	0	1	0	1	0	0	0
	129	A2[-0dB]Chigh[-3dB]E[-6dB](Amin)	0	0	1	0	0	1	0	0
All tones (different 0dB note recording)	130	Clow2DEF2GAB2Chigh[-0/-3/-6dB]	1	0	1	1	1	0	1	1

It can be seen that this is the best overall performing method of tone detection. Again interesting is the trigger on the ‘A4 virtual piano note’. A difference with the previous two methods is that the subtraction based tone detection worsens when a different recording of the same note (at 0dB) is used. For example, consider line 61 and line 96. Both lines have the combination of a Clow [0dB] with a D [-3dB]. In the situation at line 61 the generation of the combined signal and the subtraction both use the same Clow sample. For this reason, the subtraction is ideal and it is evident that the D [-3dB] will then also be detected. On the other hand, in the situation of line 96, a scaled version of a different Clow recording is subtracted from the signal. From here on, the subtraction fails to expose D [-3dB] as the highest peak over the 1st harmonic of Clow, which is Chigh. Clow and Chigh are detected but the D [-3dB] is not. This example proves a significant, and performance limiting, difference in the spectrum of separate boomwhacker recordings, even at similar volumes. In total, 215 out of 250 tones were correctly detected. This gives a total accuracy score of:

$$(215/250) * 100\% = 86.0\%.$$

4.4.6 Conclusion of Tone Detection

In this research, it is proven that tone detection of boomwhacker notes is possible. Three different methods are developed and evaluated. Data is gathered in different ways, but all use a mechanism of comparison between the data and the ideal pitch of a note. A small deviation from the ideal pitch is allowed for the detection of that note. It is expected that all three methods don't discriminate on instrument, and will (correctly) detect those pitches as well. That is, if the instrument has its stronger frequency content at the fundamental frequency.

Normalisation based tone detection normalizes a signal, then captures the frequency at the maximum value of the spectrum. The method is viable for both live audio feed and pre recorded samples. Yet, inherent to this approach is that it can only detect a single note at best. As a consequence, the total accuracy of tone detection from the database is poor at 47.6%. Despite this limitation, it performs well on single note data. With a dedicated microphone 98.8% of live played tones were detected correctly, and single note samples from the database were detected with an accuracy of 100%. Bandpassing a signal before running this algorithm increases false triggers.

Peak based tone detection searches for all peaks in a signal that satisfy certain conditions. It can detect multiple notes at the same time, and there is no limit to this amount. On the full sample database it outperforms normalisation based tone detection with a total accuracy of 62.8%. Peak based tone detection is limited by a required threshold for the height of peaks in the frequency spectrum. Samples with sufficient volume to pass the threshold were, no matter the total frequency content of the signal, always detected correctly. On the other hand, samples that didn't surpass the threshold, no matter how clean the signal was, were never detected.

Subtraction based tone detection is an iterative program. For each loop, only the maximum value of the signal is considered. If a tone is detected, a (clean) sample of this tone is subtracted from the total signal in order to expose smaller peaks. Subtraction based tone detection performed the best on the database with an overall accuracy of 86.0%. The algorithm's functionality is limited by difference in the spectrum of the pre recorded tone samples (that are used for subtraction), and any newly recorded tones. This inevitably leads to incomplete cancellation before the next iteration and can therefore false trigger notes, or fail to detect them.

5. Conclusion

Boomwhackers transients consist of a sharp attack and are very short (200-500ms). Consistent across a number of possible ways of playing boomwhackers is the presence of its fundamental frequency (FF). This peak also ‘bleeds’ into neighbouring frequencies. Harmonics of tones are present, but have very unreliable magnitudes depending on playstyle.

In order to guide tone detection of boomwhackers, initially a beat detection algorithm has been written. It uses a short buffer of 2048 samples and compares the values of the first half to those of the second half. If the values in the first half of the buffer are greater than those in the second half of the buffer by a certain amount (determined by a sensitivity factor), a beat is detected. Because this method needs time to fill up the buffer with audio samples, a delay of up to 10.76ms is inevitable. Much bigger delays occur due to processing, making the beat detection insufficient for practical use. On top of that, the beat detection is not boomwhacker specific: it triggers on all onsets or sounds with short attack. It has been chosen to test tone detection methods on a dedicated sample database, rather than on a live audio feed.

Three different types of tone detection algorithms are developed. Given the previously mentioned unreliability of harmonics, all proposed tone detection methods are focused on the FF. Data is obtained in different ways, but a mechanism of comparing the data with the ideal pitch of a note is also the same for all methods. Notes are allowed a small deviation from the ideal pitch, while still being detected. The methods are evaluated by running them over a dedicated database of separate tones and tone combinations. The tone detection methods are expected to be able to detect other instruments as well, given their FF is the strongest frequency component in the signal. Though, this has not been extensively tested.

Normalisation based tone detection normalizes a signal, then captures the frequency at the maximum value of the spectrum. The method is viable for both live audio feed and pre-recorded samples. It performed very well for data containing merely single tones. Yet, inherent to this approach is that it can only detect a single note at best. Its total accuracy over the sample database is the lowest at 47.6%.

Peak based tone detection searches for all peaks in a signal that satisfy certain conditions. It can detect multiple notes at the same time, there is no limit to this amount. Peak based tone detection is limited by a required threshold for the height of peaks in the frequency spectrum. Samples with sufficient volume to pass the threshold were, no matter the complexity of the signal, detected correctly. In contrast, samples that didn’t pass the threshold, no matter how clean the signal was, were never detected. The total accuracy score is 62.8%.

Subtraction based tone detection is an iterative program. For each loop, only the maximum value of the signal is considered. If a tone is detected, a (clean) sample of this tone is subtracted from the total signal in order to expose smaller peaks. The algorithm’s functionality is limited by difference in the spectrum of the pre recorded tone samples (that are used for subtraction), and any newly recorded tones. Still it performs best at a total accuracy of 86.0%.

6. Discussion and recommendations

In this section the research is evaluated and based on that recommendations are given for future research. Please note this section concerns, among others, personal processes and experiences, statements might not be true for everyone.

First of all, at the start of this research, all code implementations were focused on a live audio feed. This is not necessarily bad, but in this case it has cost a lot of time in order for it to get to work. Starting out with live recording is a lot more laborious than with pre recorded data. A few reasons for this are:

- Live audio recording is more complex code-wise. Input buffers are finite and therefore overwrite themselves continuously, making it harder to capture and keep track of the exact input data.
- Testing a script (especially in early stages) on live audio is a less structured approach compared to sourcing from file. Live audio has a worse repeatability, it can be hard to distinguish a poor script from (unexpected) variations of the input.

The discussed beat detection is plausible. At a fixed sensitivity, children might be encouraged to play loud enough to get their beats detected. However, it is unlikely to perform well when implemented in a realistic scenario. The sensitivity of the beat detection should be automated based on the characteristics of the input signal. If for some reason background noise raises in volume, beats that are just barely detected should increase their volume in order to keep being detected. This wouldn't improve the quality of the music.

Furthermore, it would be insightful to extend the existing database with noisy samples, in order to quickly test performance among more realistic circumstances. Additionally, samples of other instruments could be included to test whether the tone detection is viable for other instruments as well. Whether this would be a positive or negative property is a conflicting question. Being able to use the system for different other instruments as well would be a great side effect. On the other hand, if multiple instruments are used simultaneously, the results will be of little (individual) meaning.

As of yet, the tone detections have been run with high quality data: sampling at 44100 Hz and with 24 bit integer values. Needless to say, this data format is unnecessarily big for an application that only examines frequencies up to ~530 Hz. For a system that should eventually be able to limit any significant delay, low amounts of data that allow for shorter processing times are obviously desired. In a follow up study, a great experiment would be downscaling the test data until the system shows decrease of performance. That is: the data is being simplified until proper detection of features fails.

Normalisation based tone detection is a devious way of detecting tones. In fact, normalisation is completely unnecessary. Making matters worse, normalisation can even aid false triggers. A rather simple highest peak finding function (as used in subtraction based tone detection) would suffice. Normalisation was only done because I initially thought of it as an easy way of always detecting the highest peak.

For peak based tone detection, additional testing at different thresholds would give a nice overview of which threshold performs best. A possible major improvement could be a peak prominence requirement. The rather sharp peaks of boomwhacker fundamental frequencies can be exploited more. On top of a threshold (which could likely be set lower with such an implementation), the peaks are required to possess a certain prominence. This can be achieved by comparing the peak value to its neighbouring samples and evaluating how 'steep' the increase in value is. For example, a peak value at index n could be required to have more than twice the value of the samples value at indices n-100 and n+100, before being considered a valid peak in the signal. Note that this index span also depends on the used frequency resolution.

Subtraction based tone detection showed the best potential, but performance was not great when subtracting another recording (of the same note) from the signal. As shown in section 4.1, difference in the frequency spectrum of each recording, despite it being from the same note at a similar volume, is inevitable. It could probably be improved by subtracting signal specific samples based on the characteristics of the input signal's peak. For example: a peak with a very high amplitude (which generally contains less bleed) would be considered a very loud hit, after which the program will choose, from a number of available samples to subtract, a sample of a very loud tone as well.

Finally, instead of just looking at peaks, tone detection could be implemented based on the entire signal. Despite it being able to vary quite a lot to the eye, playing the same tone using the same boomwhacker must produce some consistent aspects in the signal. It would be interesting to research how for example a machine learning based application would perform. Similar existing applications such as Shazam[2] prove that there is a lot of potential, even for very complex signals.

[2] https://www.shazam.com/

7. References

[1] C. Nieuwmeijer, *“Dutch early years classroom teachers facilitating and guiding musical play: problems and opportunities”, European Early Childhood Education Research Journal Volume 27, Issue 6, 2 November 2019, Pages 860-871.* [Online]. Available at: https://www.tandfonline.com/doi/full/10.1080/1350293X.2019.1678926 [Accessed April 14th, 2020]

[2] Agnes S. Chan, *“Music Training Improves Verbal But Not Visual Memory”, Nature, vol. 396, 12 November 1998.* [Online]. Available at: https://www.researchgate.net/publication/13462612. [Accessed March 10, 2020]

[3] D. Gonzales, “*Microphone position and noise exposure assessment of building façades”, Applied acoustics vol 160, March 2020.* [Online]. Available at: https://www.sciencedirect.com/science/article/pii/S0003682X19305079 [Accessed March 12, 2020]

[4] T. Kamekawa, “*Evaluation of recording techniques for three-dimensional audio recordings: Comparison of listening impressions based on difference between listening positions and three recording techniques”, Acoustic science and technology, vol. 1, January 2020.* [Online]. Available at: https://www.jstage.jst.go.jp/article/ast/41/1/41_E19223/_pdf/-char/en [Accessed March 4, 2020]

[5] H. Lee *“Capturing 360◦ Audio Using an Equal Segment Microphone Array (ESMA),” J. Audio Eng. Soc., vol. 67, no. 1/2, pp. 13–26, January 2019.* [Online]. Available at: http://www.aes.org/tmpFiles/elib/20200415/19883.pdf [Accessed March 29th, 2020]

[6] H. Lee “*Effect of Vertical Microphone Layer Spacing for a 3D Microphone Array”, J. Audio Eng. Soc., Vol. 62, No. 12, 2014 December.* [Online]. Available at: http://eprints.hud.ac.uk/id/eprint/23149/1/LeeEffectofVertical.pdf [Accessed March 4, 2020]

[7] C. Simmons, “*Measurement of Sound Pressure Levels at Low Frequencies in Rooms. Comparison of Available Methods and Standards with Respect to Microphone Positions”, Acta Acustica united with Acustica, Volume 85, Number 1, January/February 1999, pp. 88-100.* [Online]. Available at: https://www.ingentaconnect.com/content/dav/aaua/1999/00000085/00000001/art00015# [Accessed March 7, 2020]

[8] Y. Tan, "*Improve Indoor Acoustics Performance by Using Building Information Modeling", in proc. of the 34th ISARC, June 2017, Tapei.* [Online]. *Available at: https://pdfs.semanticscholar.org/3137/4deabcbf52a99e3a3e764778b85d97bcded0.pdf*
[Accessed April 13th, 2020]

[9] C. Wu, "*BIM-based acoustic simulation framework", in proc. of the 30th international conference of CIB, October 2013, Beijing.* [Online]*. Available at:*
https://www.researchgate.net/publication/304625903
[Accessed April 2nd, 2020]

[10] E. George, *"Measuring the Effects of Reverberation and Noise on Senetence Intelligibility for Hearing-Impaired listeners", Journal of Speech Language and Hearing Research, December 2010 .* [Online]. Available at: https://www.researchgate.net/publication/45508426
[Accessed April 10th, 2020]

[11] Y. Lee, *"Behavioral Hearing Thresholds Between 0.125 and 20 kHz Using Depth-Compensated Ear Simulator Calibration", Ear Hear; 33(3): 315–329, 2012.* [Online]. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3606020
[Accessed April 5th, 2020]

[12] E. Bilgic, "*Evaluation of Uncertainty Contributions of Measurement Surface and Number of Microphone Positions in Determination of Sound Power Levels", Acta Physica Polonica, vol. 132, no 3. 2017.* [Online]. Available at:
http://przyrbwn.icm.edu.pl/APP/PDF/132/app132z3p065.pdf
[Accessed March 7, 2020]

[13] G. Tzanetakis, A. Klapuri, "Terminology and concepts"*,* in *"Signal Processing Methods for Music Transcription",* Massachusetts, USA: MITP, 2008, p. 8. [Online]. Available at:
https://books.google.nl/books?id=AF30yR41GIAC&pg=PA8&redir_esc=y#v=onepage&q&f=false
[Accessed April 13th, 2020]

[14] X. Serra, *"A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic plus Stochastic Decomposition", Stanford University, 1989.* [Online]. Available at:
https://176-31-21-26.pool.sistemaip.net/bitstream/handle/10230/34072/Serra_PhDthesis.pdf
[Accessed April 20th, 2020]

[15] A. Eronen, A. Klapuri, *"Musical instrument recognition using cepstral features and temporal features"*, *Acoustics, Speech, and Signal Processing*, February 2000.
[Online] Available at: https://www.researchgate.net/publication/3858745
[Accessed April 28th, 2020]

[16 Eronen 2001] A. Eronen, *"Comparison of features for musical instrument recognition", IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, October 2001.* [Online]
Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=969532&tag=1
[Accessed April 28th, 2020]

[17] G. Agostini et al., *“Musical Instrument Timbres Classification with Spectral Features”, EURASIP Journal on Applied Signal Processing 2003:1, 5–14.* [Online] Available at: https://www.researchgate.net/publication/26532361
[Accessed April 28th, 2020]

[18] Y. Lin, *“Real-Time Analysis of Beats in Music for Entertainment Robots”, auSMT journal Vol. 2 No.4 (dec. 2012).* [Online] *Available at:*
https://www.researchgate.net/publication/272728069
[Accessed April 28th, 2020]

[19] M.R. Schroeder, *“Period Histogram and Product Spectrum: New Methods for Fundamental-Frequency Measurement”, Journal of the Acoustical Society of America 43, 829 1968.* [Online] Available at: https://asa.scitation.org/doi/pdf/10.1121/1.1910902
[Accessed April 29th, 2020]

[20] A. Bregman, *“Auditory Scene Analysis: the perceptual organization of sound”, Journal of the Acoustical Society of America, January 1990.* [Online] Available at:
https://www.researchgate.net/publication/200045100
[Accessed April 29th, 2020]

[21] S.T. Roweis, “O*ne Microphone Source Separation”, Advances in Neural Information Processing Systems 13, NIPS, 2000.* [Online] Available at:
https://papers.nips.cc/paper/1885-one-microphone-source-separation
[Accessed April 29th, 2020]

[22] *“Physics of musical notes, frequencies for equal tempered scale*”, *Michigan Technological University (MTU).* [Online] Available at:
https://pages.mtu.edu/~suits/notefreqs.html
[Accessed June 15th, 2020]

[23] D. Self, B. Duncan, *“Audio Engineering, know it all ”* , 1st ed., Oxford: Newnes, 2009. pp. 278. [Accessed August 12th 2020]

8. Appendices

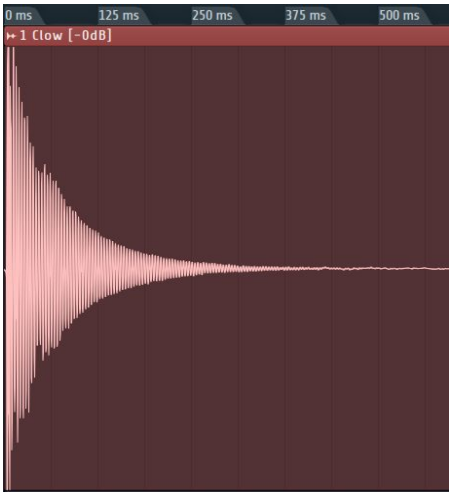
Appendix A: Boomwhacker transients and frequency spectra

Control figures can be found in the top row, the matlab figures in the 2nd and 3rd rows. From the control figures it is clear that high frequencies are very noisy, and low frequencies are (almost) absent. Therefore the matlab figures are zoomed, disregarding these ranges and showing only the most distinct peaks of the signal. Mind the different scaling on the horizontal axis.

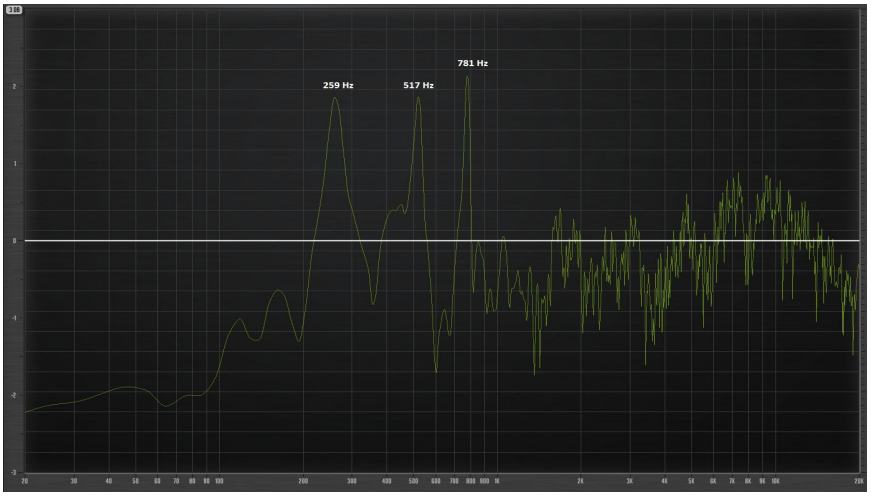
Clow

Control

Transient

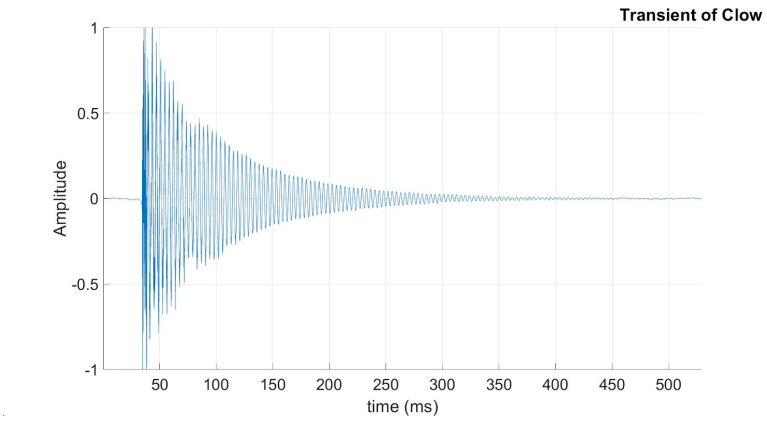


Frequency Spectrum

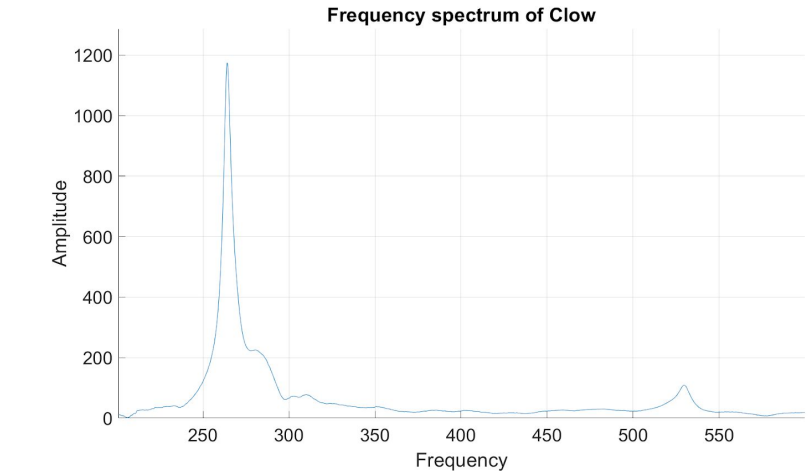


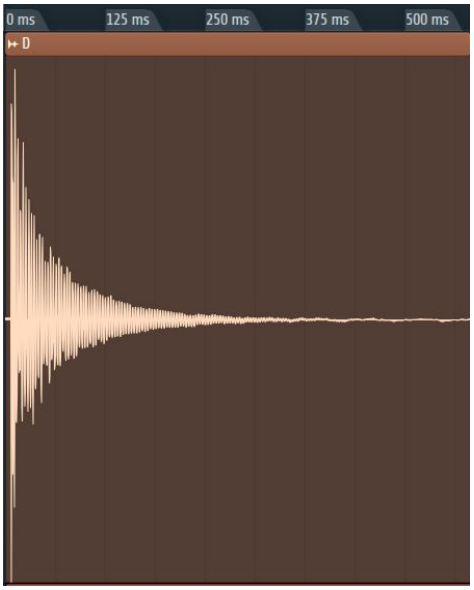
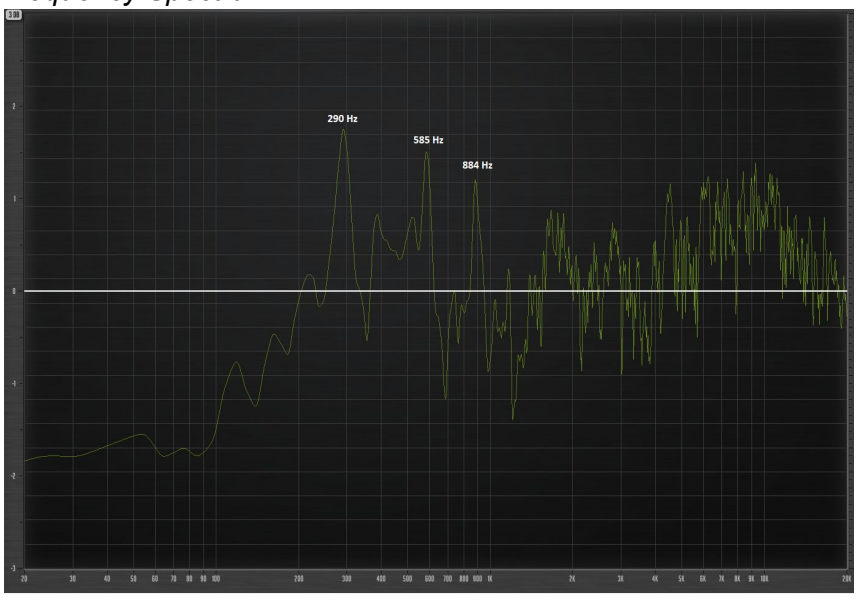
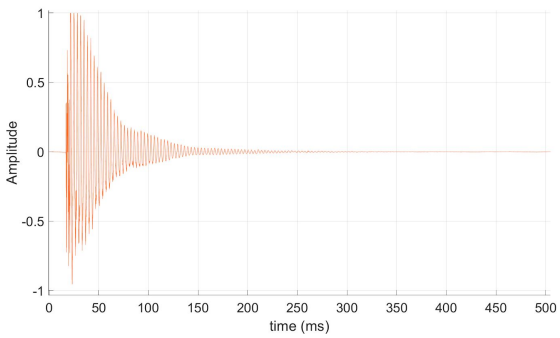
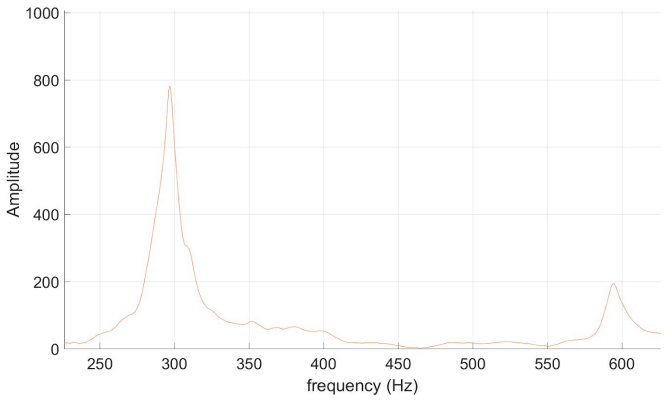
Matlab

Transient

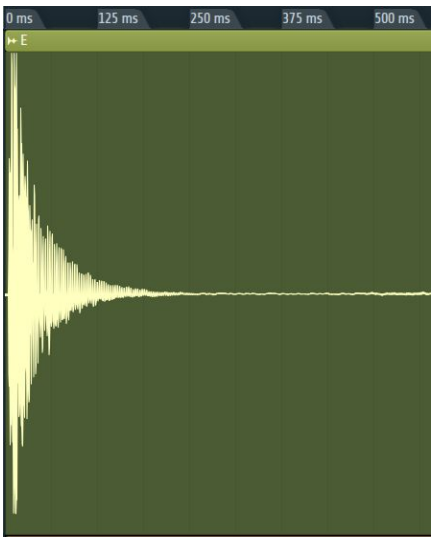
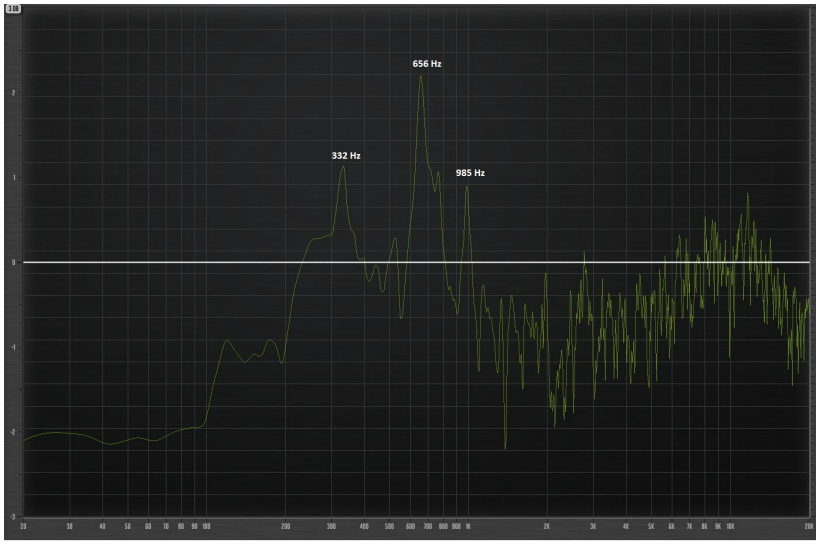
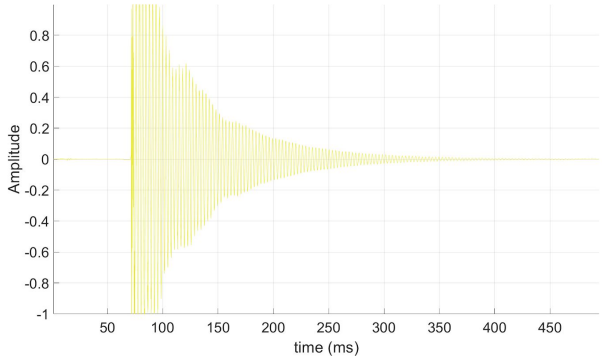
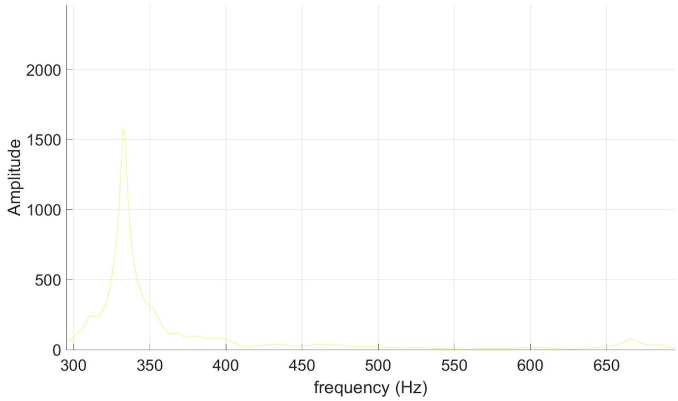


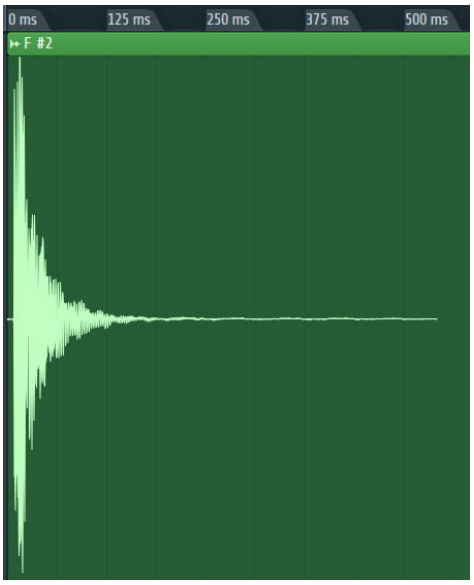
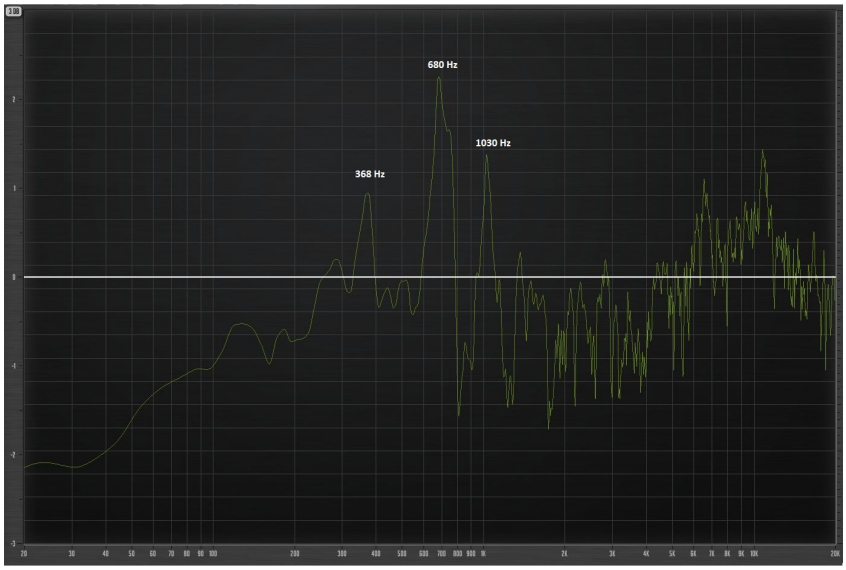
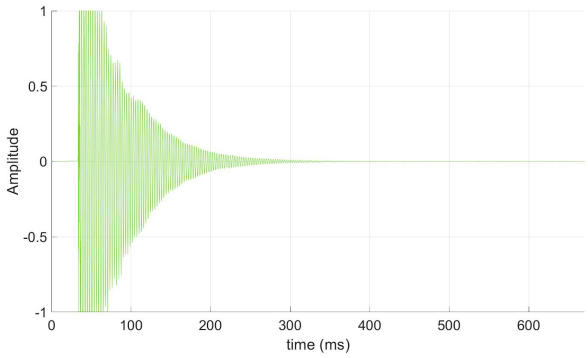
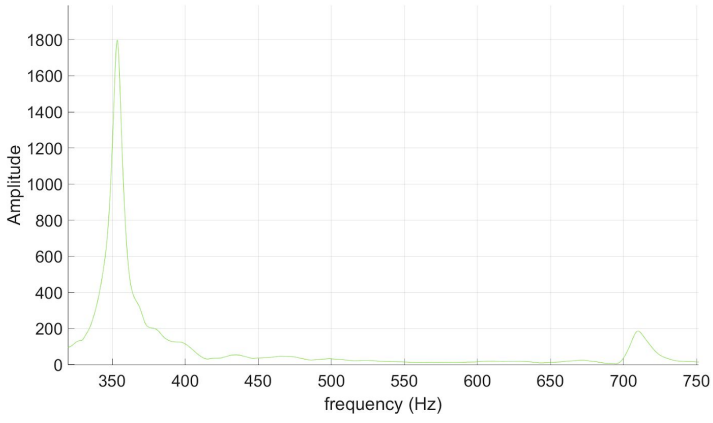
Frequency spectrum



D		
Control	*Transient* 	*Frequency Spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

E

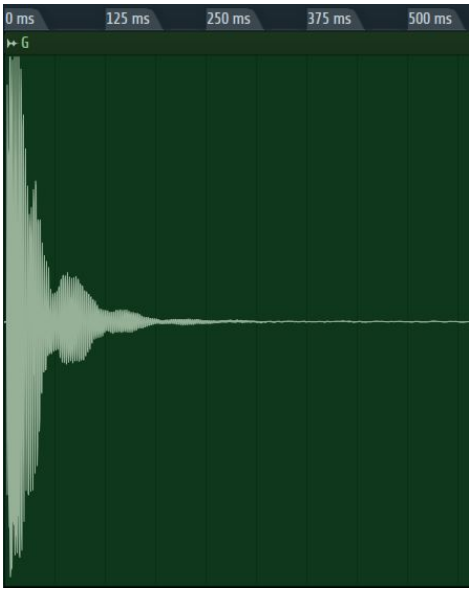
Control	*Transient* 	*Frequency Spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

F		
Control	*Transient* 	*Frequency spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

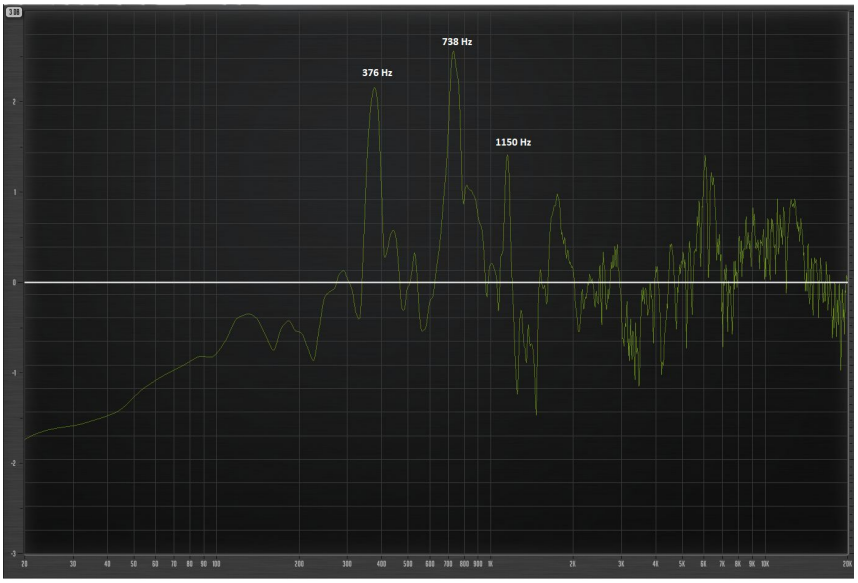
G

Control

Transient

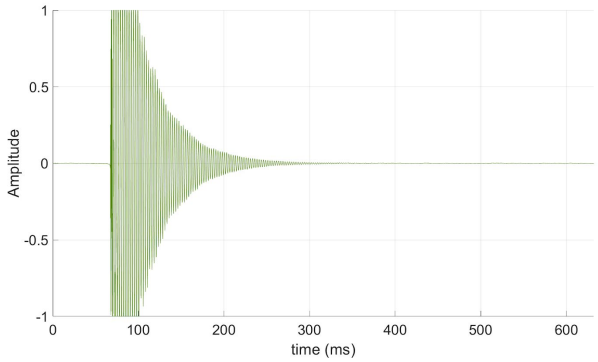


Frequency spectrum

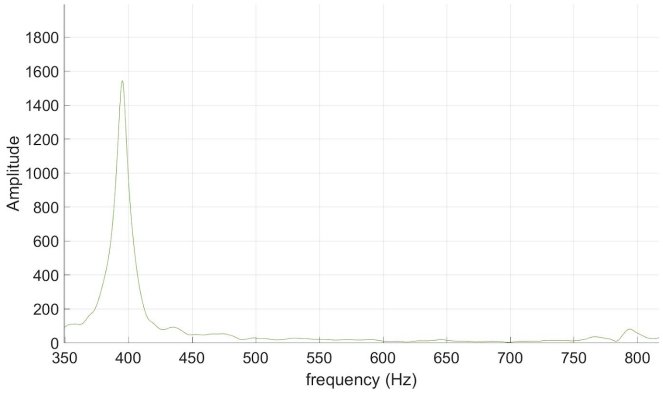


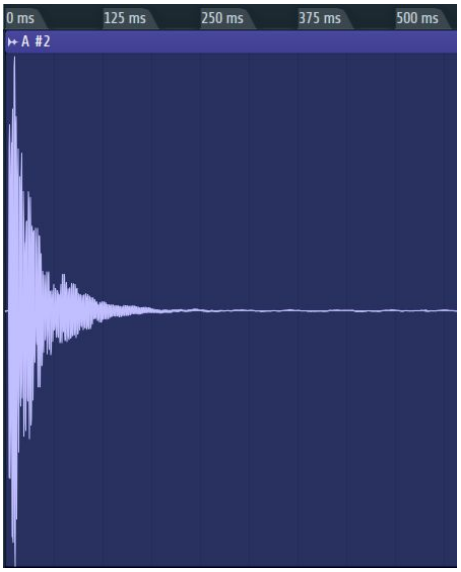
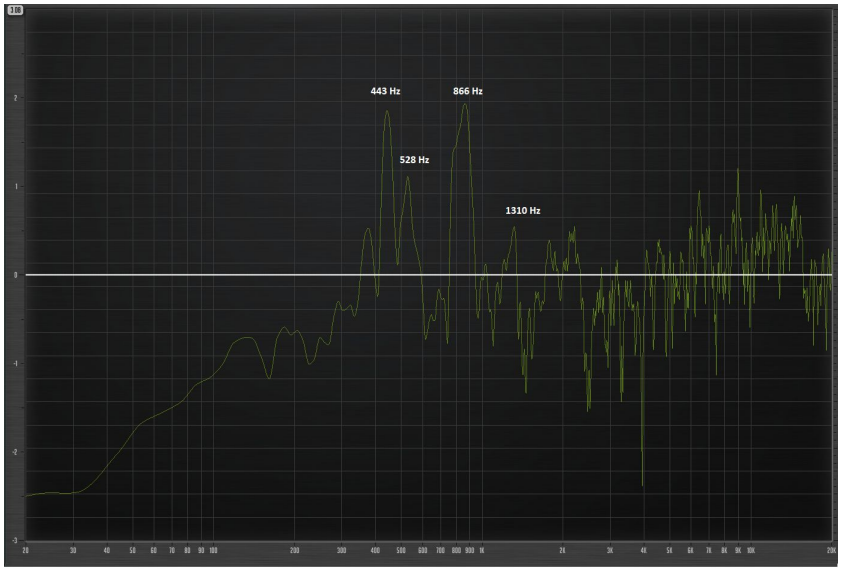
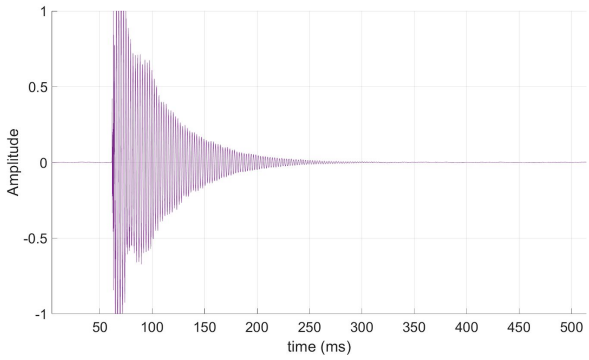
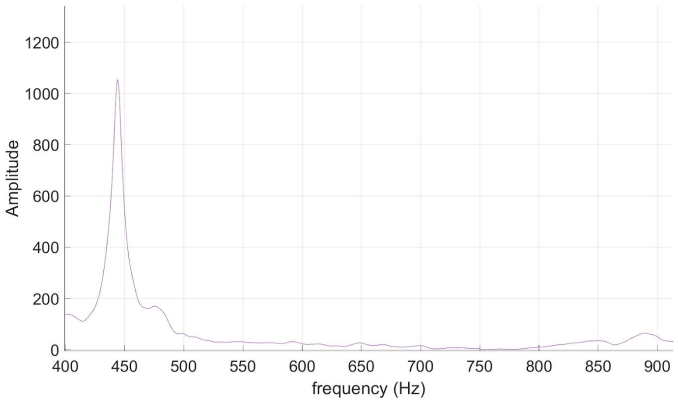
Matlab

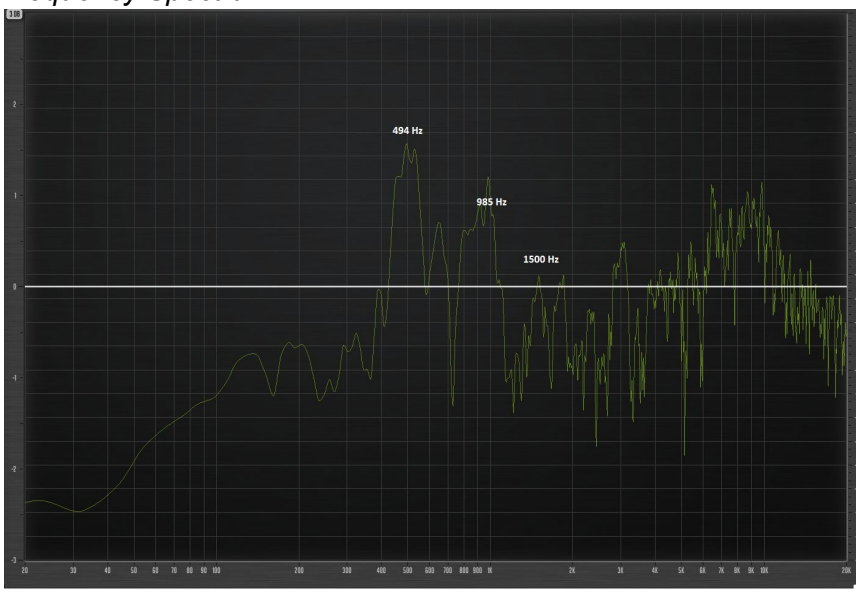
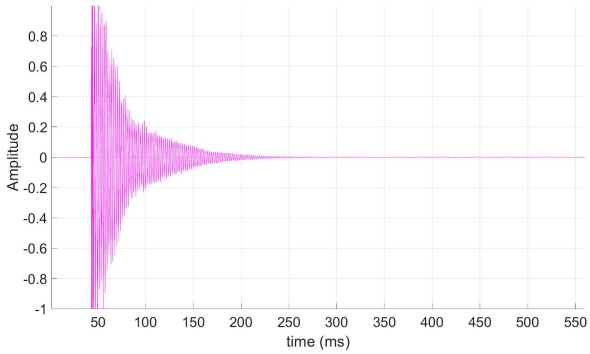
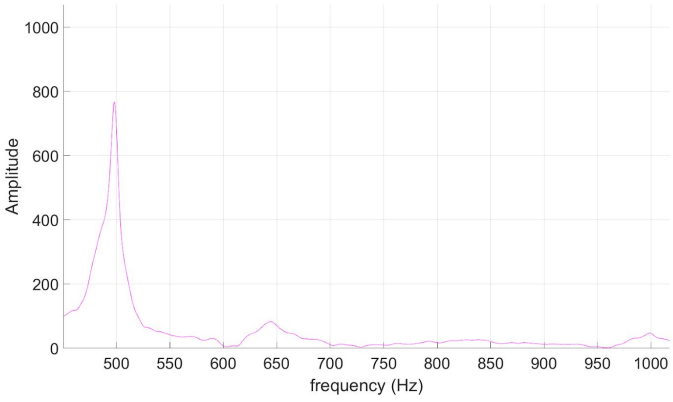
Transient



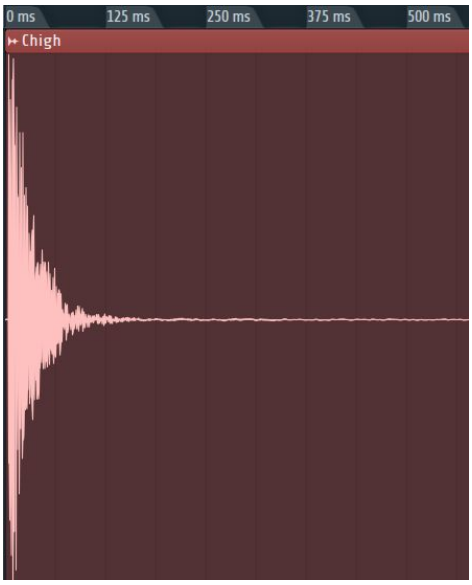
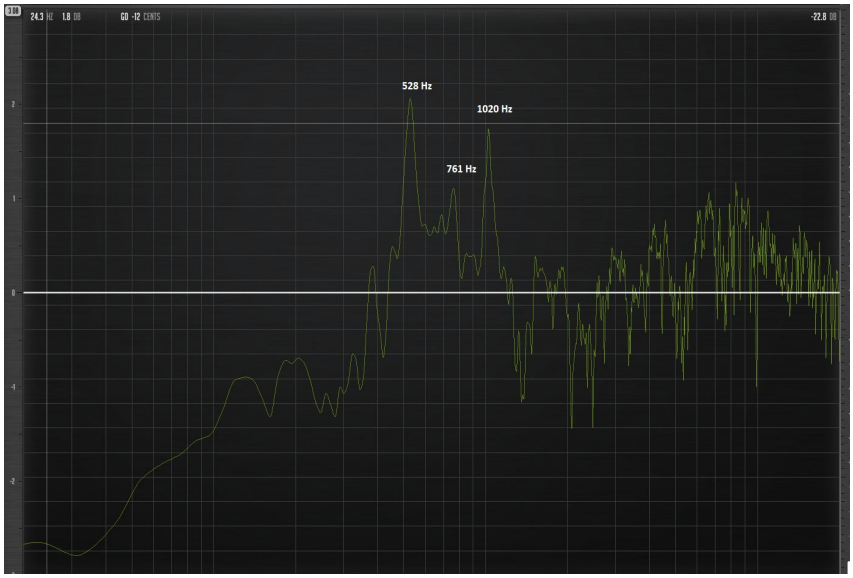
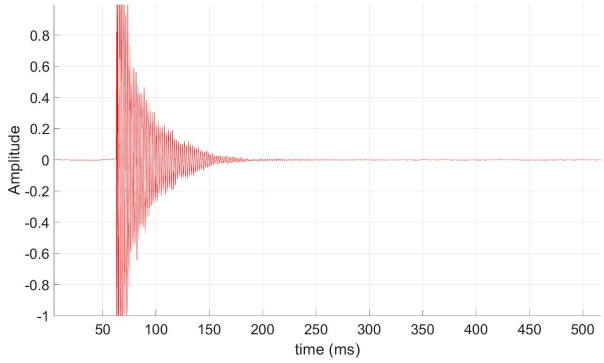
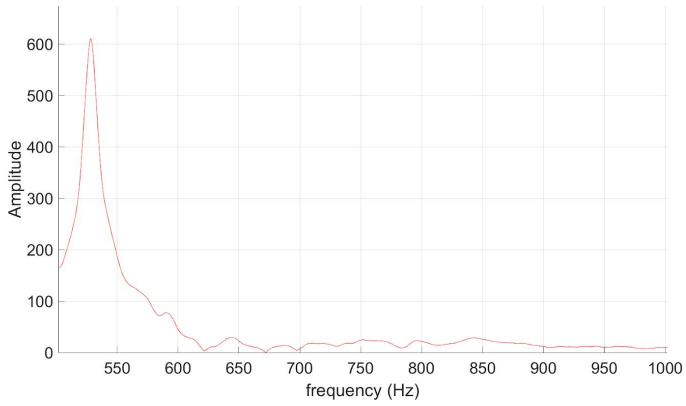
Frequency spectrum



A		
Control	*Transient* 	*Frequency spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

B		
Control	*Transient* 	*Frequency Spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

Chigh

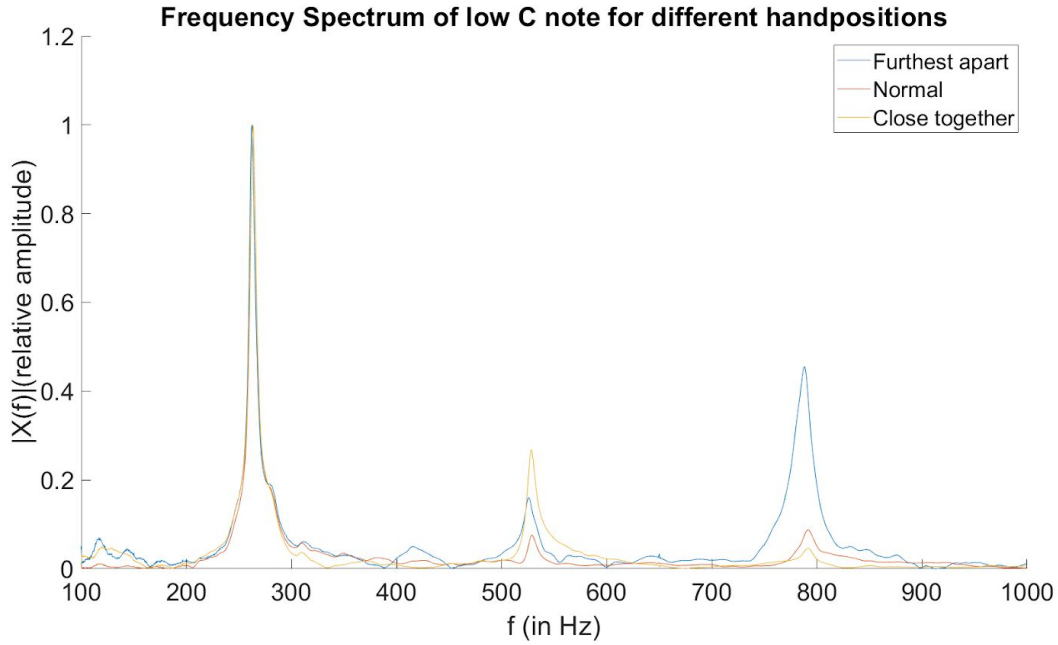
Control	*Transient* 	*Frequency spectrum* 
Matlab	*Transient* 	
	Frequency spectrum 	

Appendix B: Different playing styles

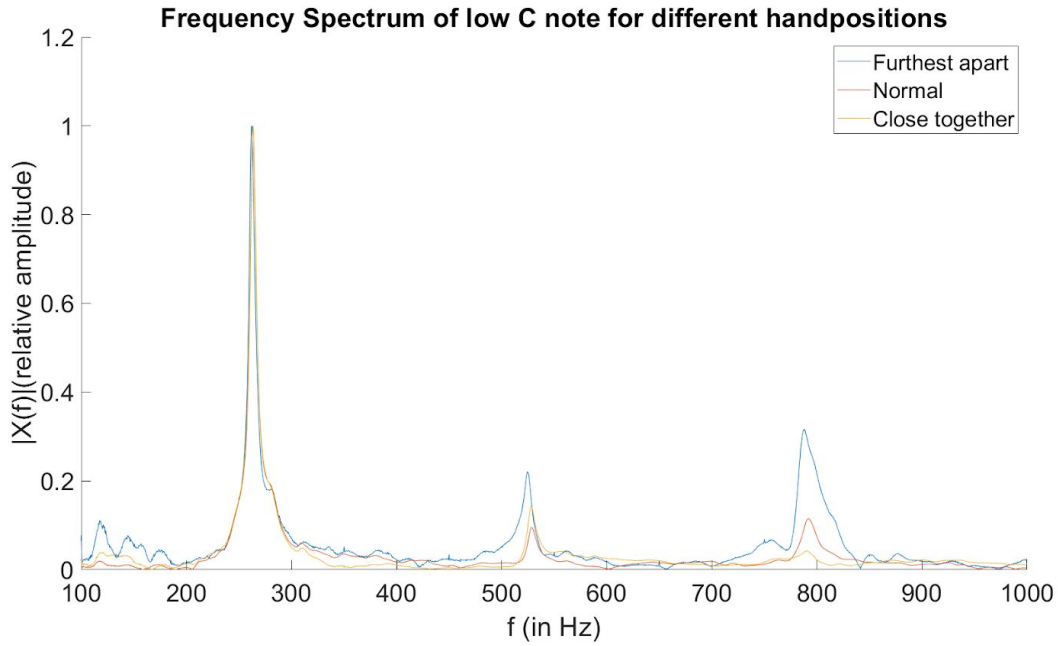
The following plots are all normalized.

Different hand positions:

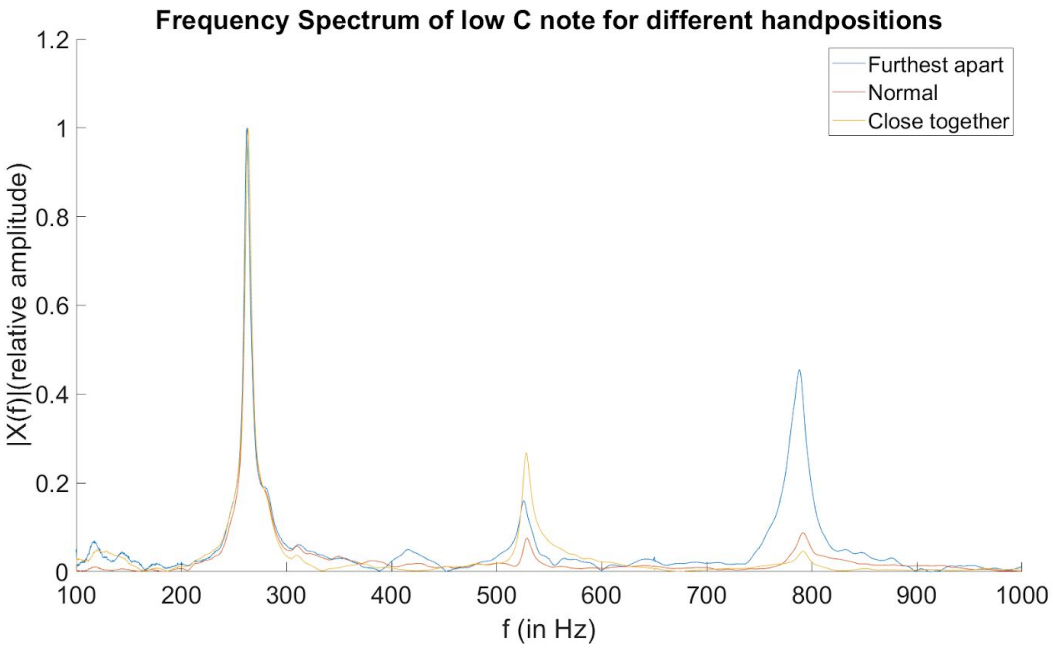
Recording 1:



Recording 2:

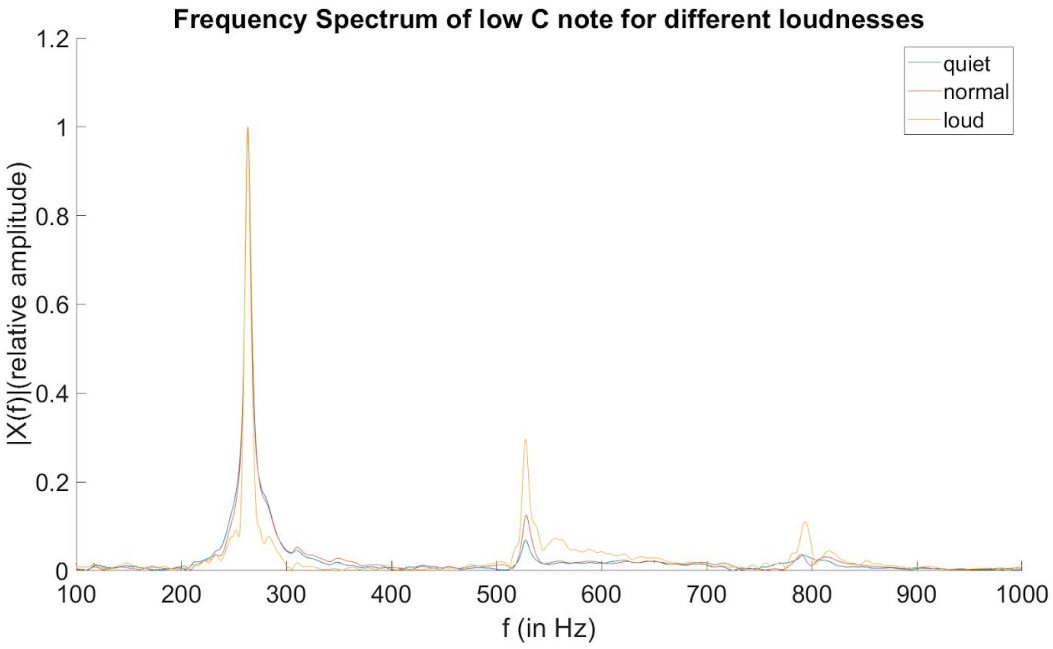


Recording 3:

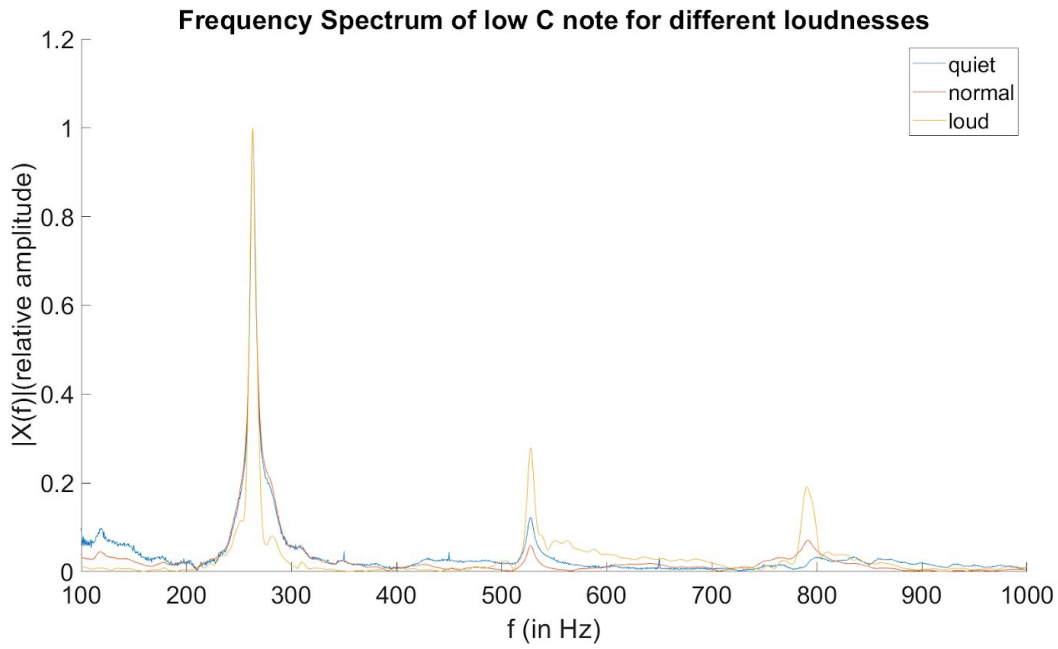


Different loudness:

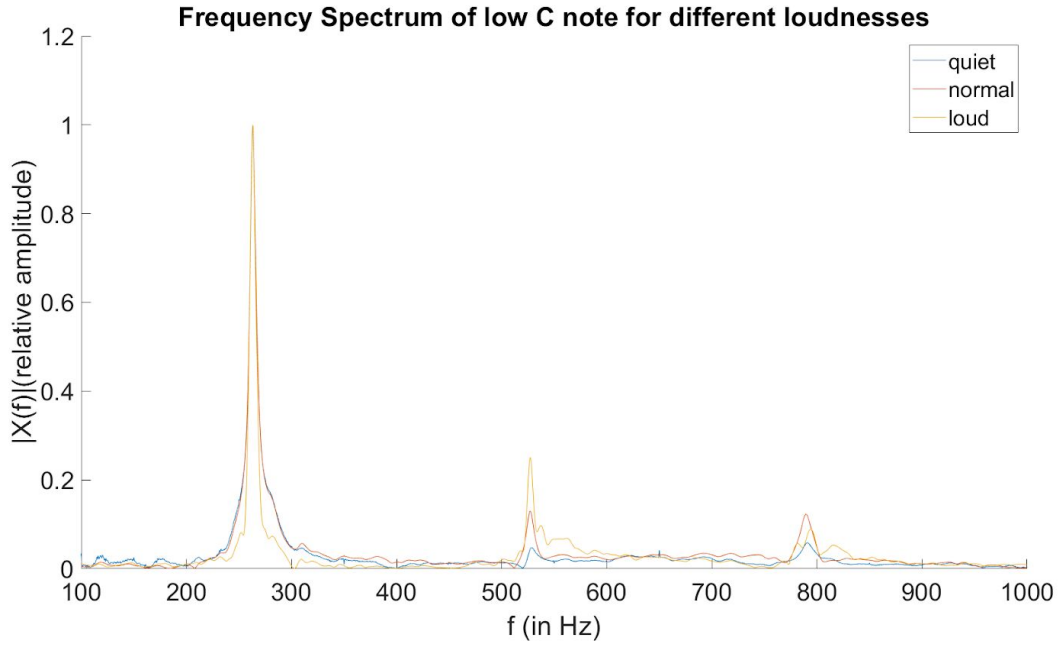
Recording 1:



Recording 2:

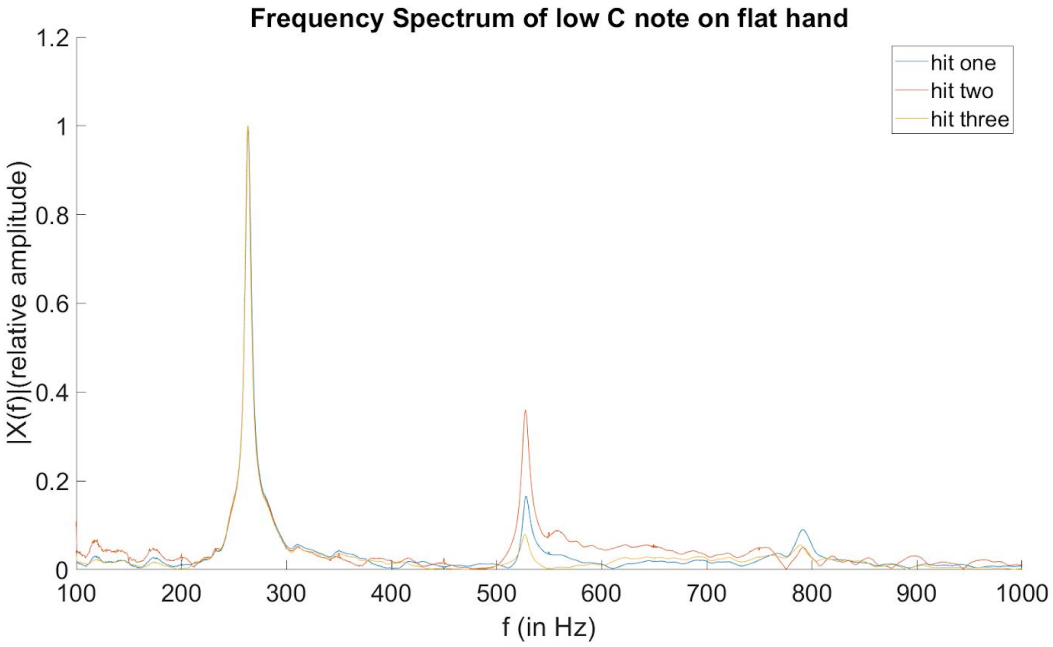

Frequency Spectrum of low C note for different loudnesses
quiet
normal
loud
|X(f)|(relative amplitude)
f (in Hz)
0
0.2
0.4
0.6
0.8
1
1.2
100
200
300
400
500
600
700
800
900
1000

Recording 3:

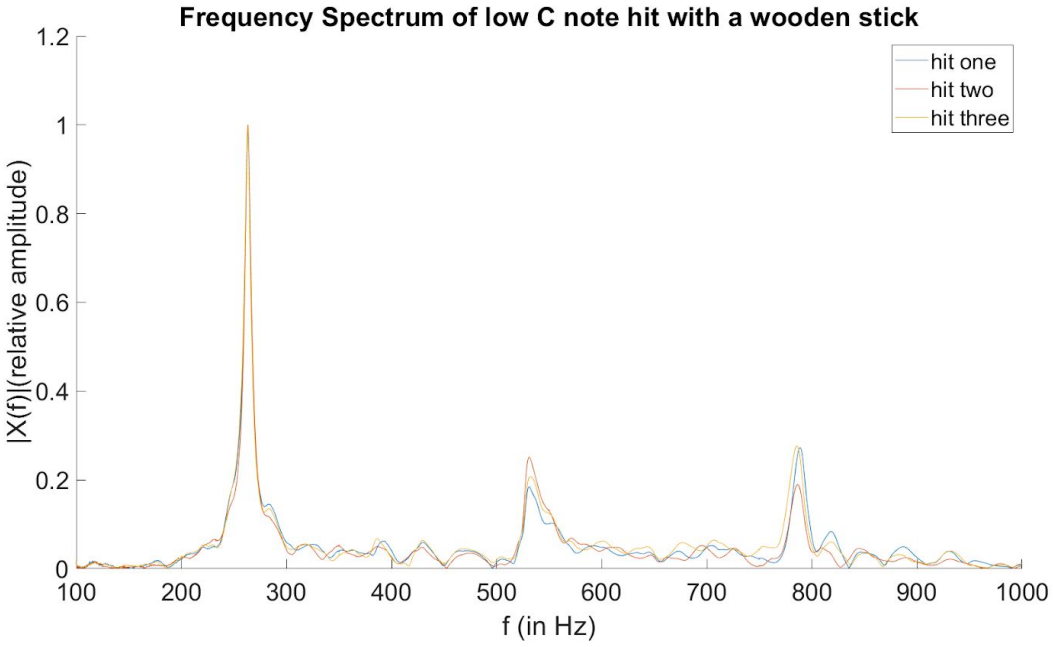

Frequency Spectrum of low C note for different loudnesses
quiet
normal
loud
|X(f)|(relative amplitude)
f (in Hz)
0
0.2
0.4
0.6
0.8
1
1.2
100
200
300
400
500
600
700
800
900
1000

Different surfaces

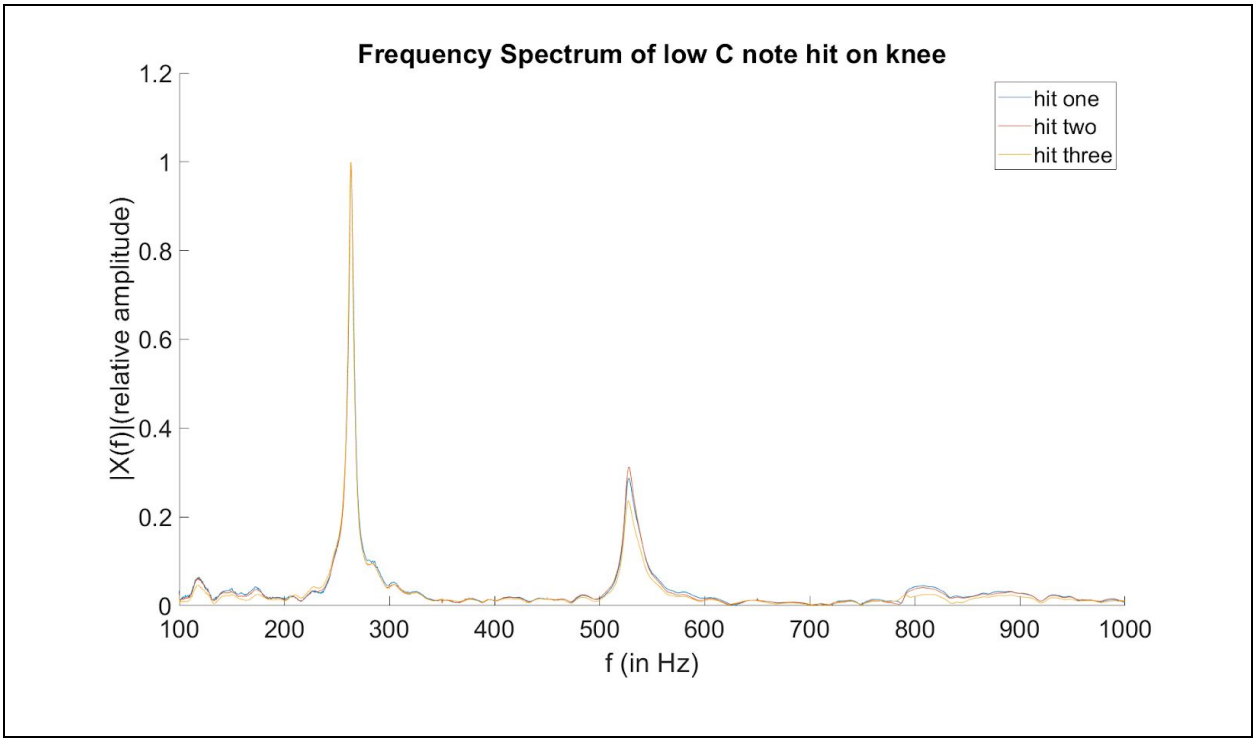
Flat hand:



Wooden stick

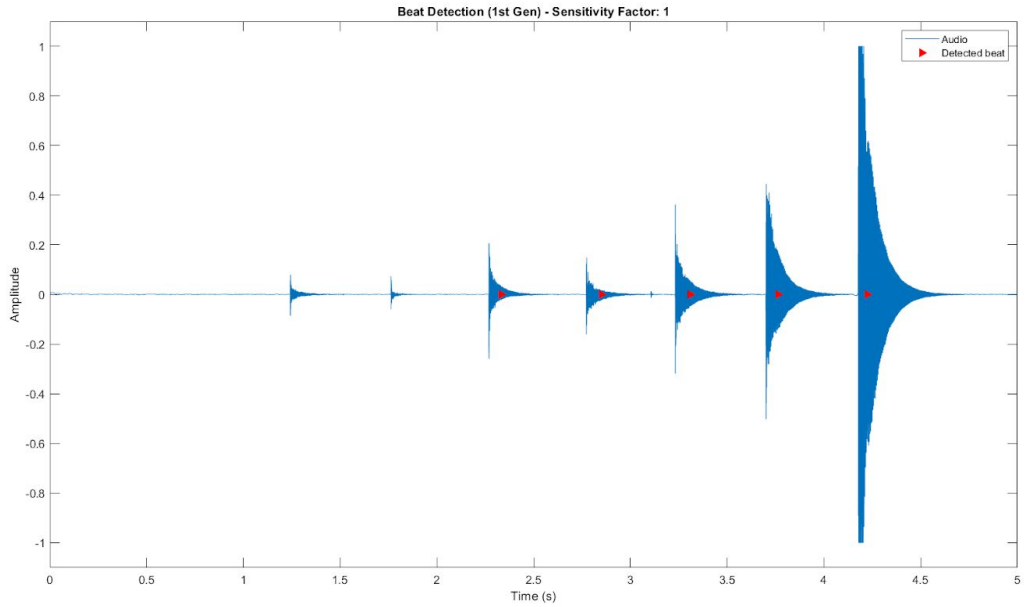


Knee

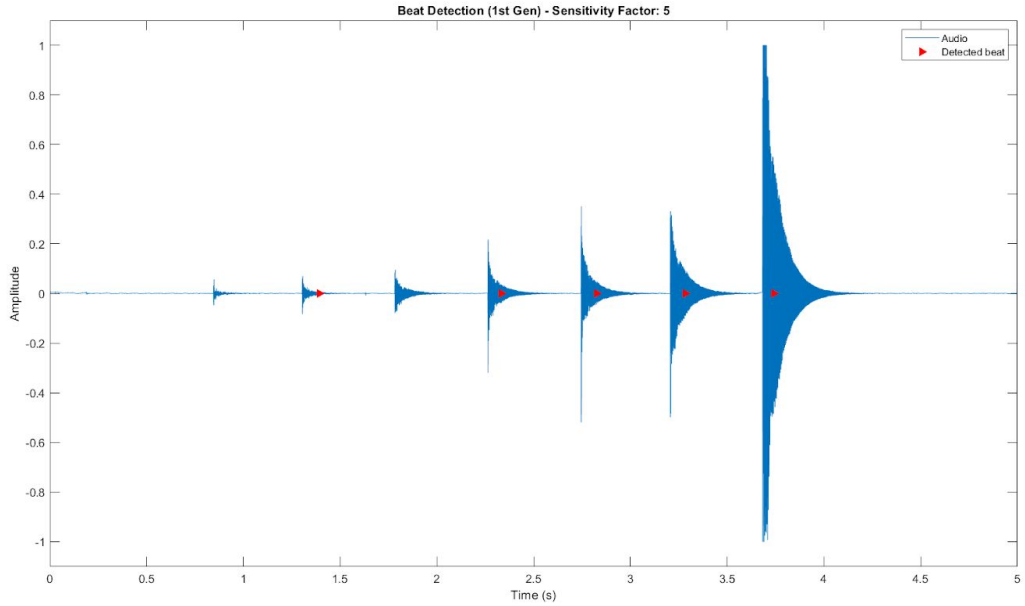

Frequency Spectrum of low C note hit on knee
hit one
hit two
hit three
|X(f)|(relative amplitude)
f (in Hz)
0
0.2
0.4
0.6
0.8
1
1.2
100
200
300
400
500
600
700
800
900
1000

Appendix C: Beat detection performance graphs

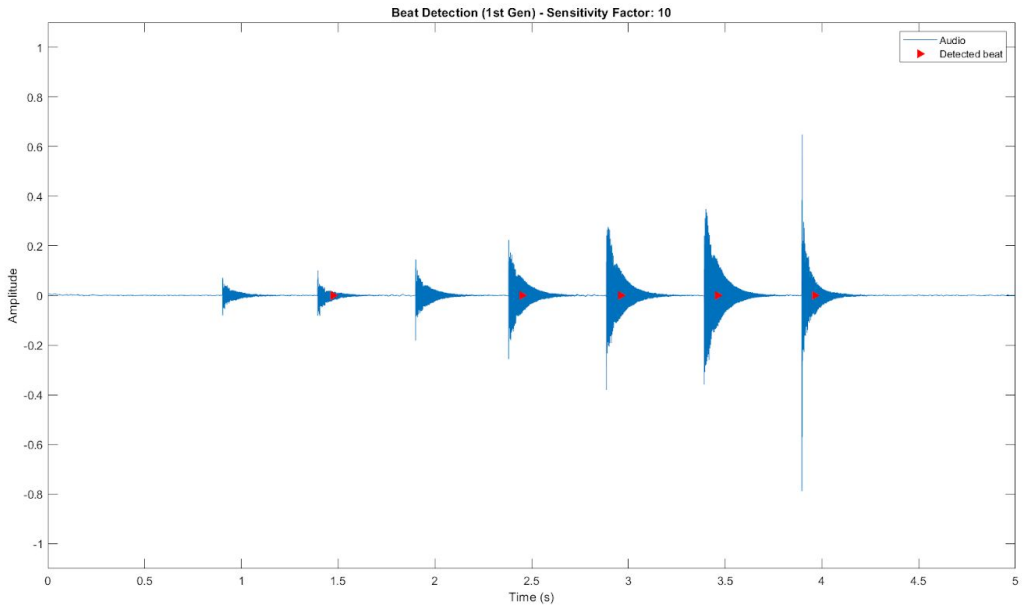
Comparison of different sensitivity factors.



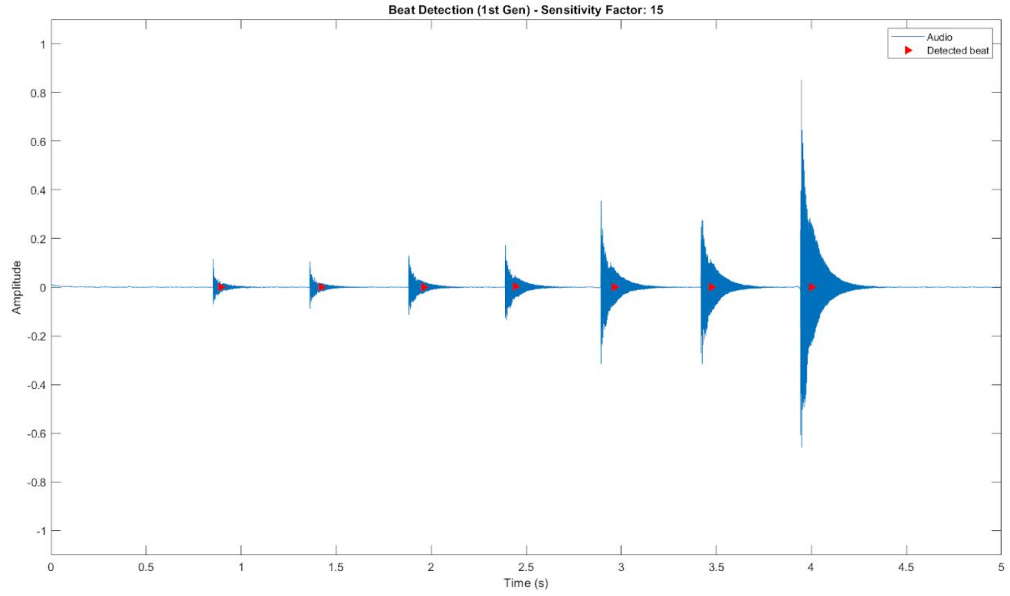
Beat detection performance for a sensitivity factor of 1



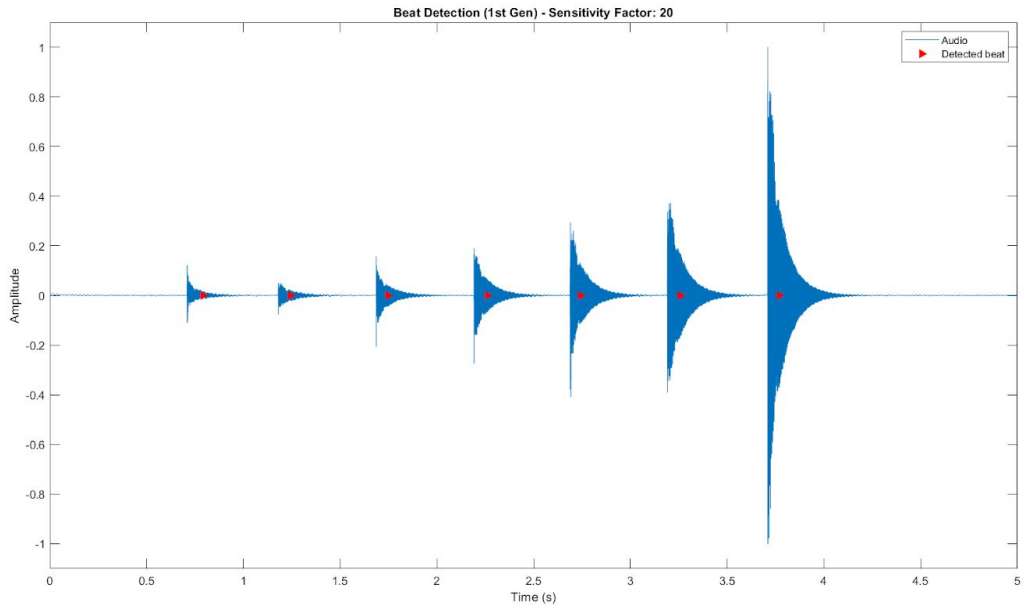
Beat detection performance for a sensitivity factor of 5



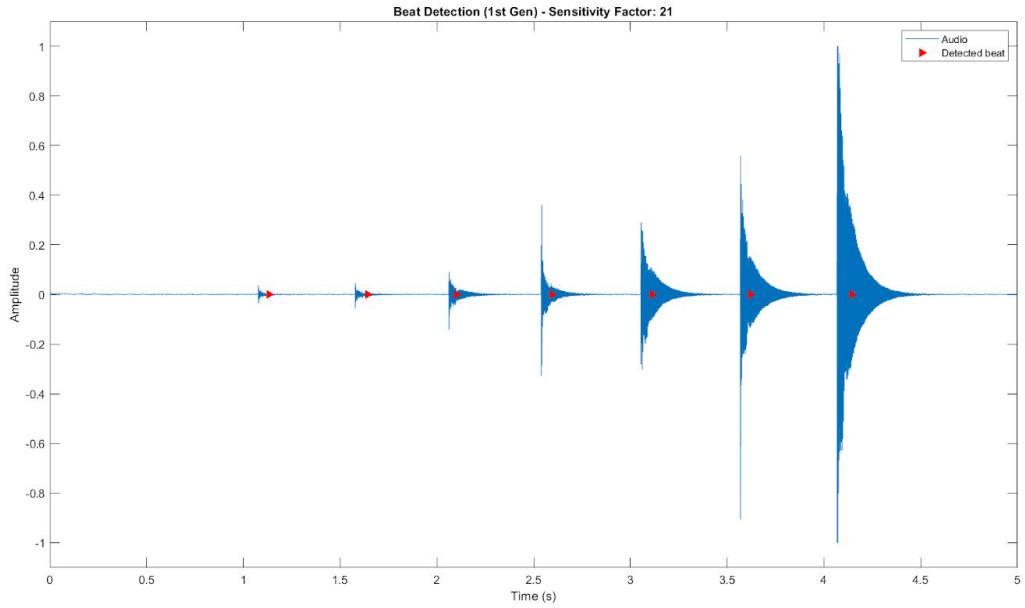
Beat detection performance for a sensitivity factor of 10



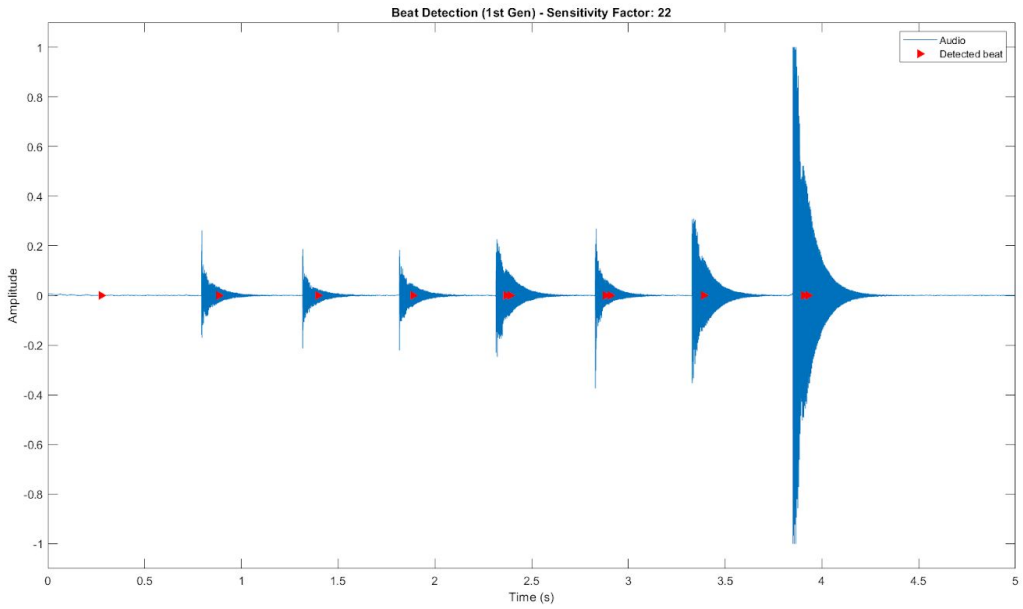
Beat detection performance for a sensitivity factor of 15



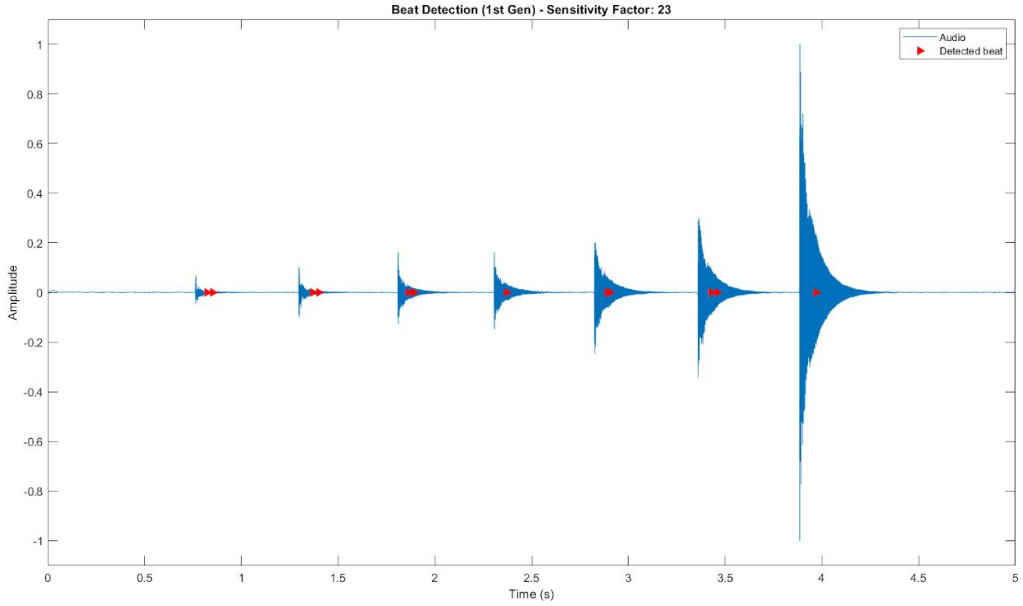
Beat detection performance for a sensitivity factor of 20



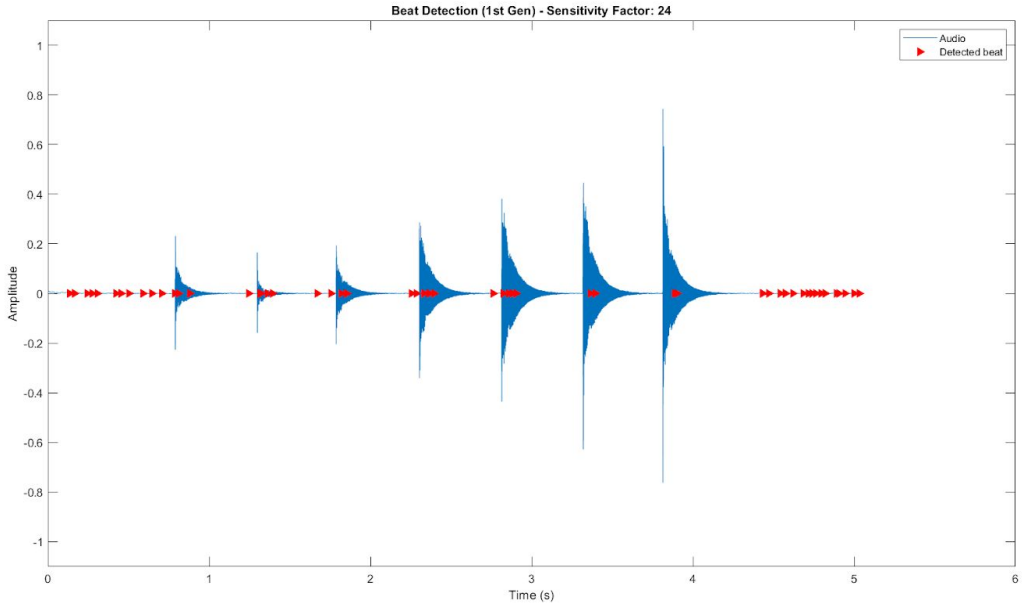
Beat detection performance for a sensitivity factor of 21



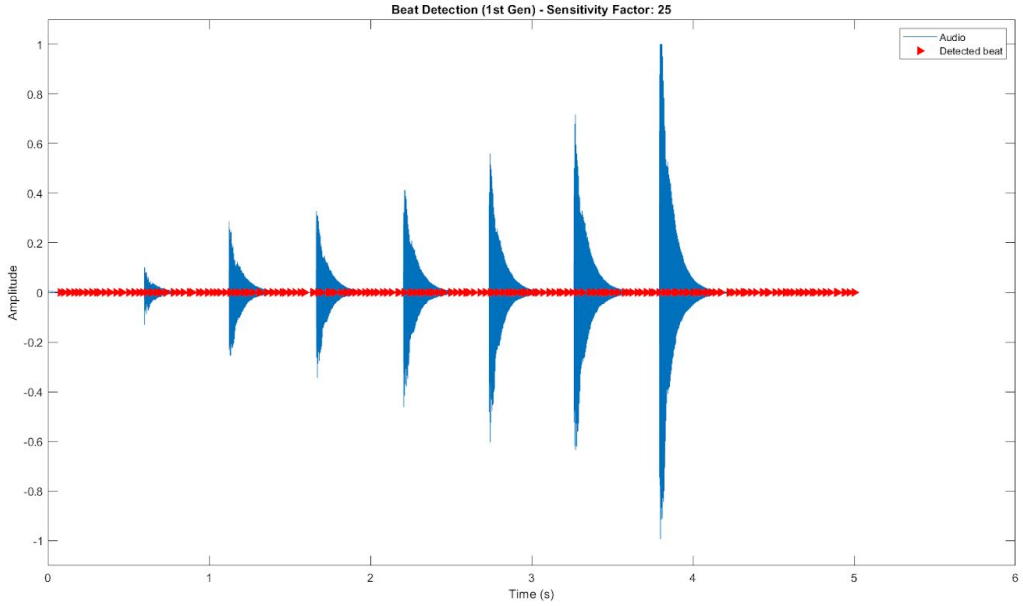
Beat detection performance for a sensitivity factor of 22



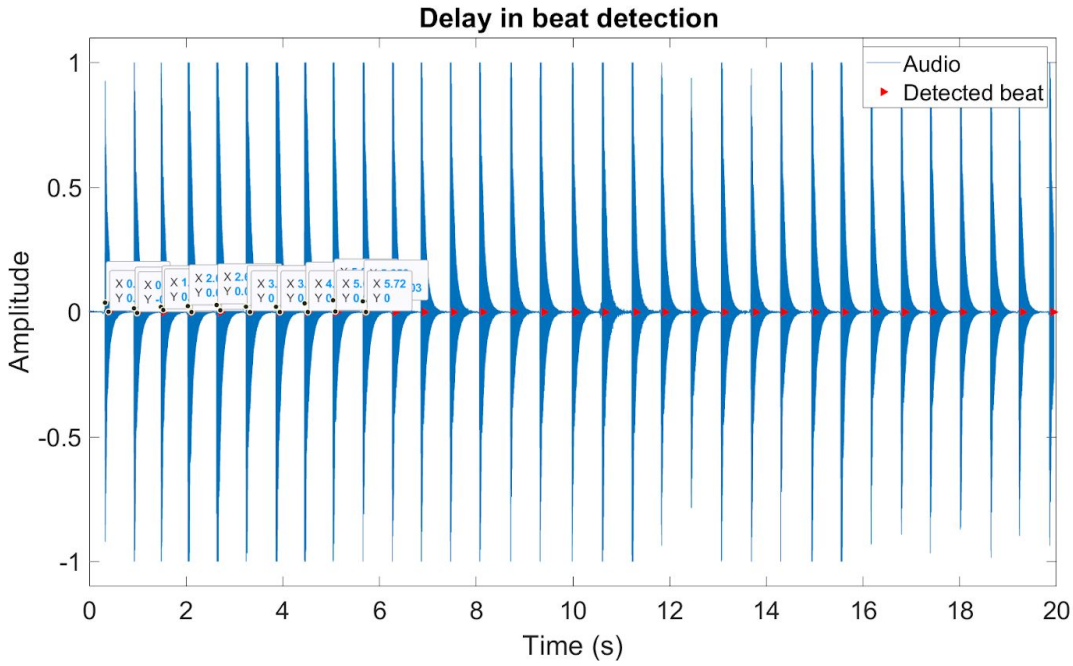
Beat detection performance for a sensitivity factor of 23



Beat detection performance for a sensitivity factor of 24



Beat detection performance for a sensitivity factor of 25



Beat detection time comparing to manual determination of the onset (used for calculating the beat detection correction factor)

Appendix D: Matlab code

This section contains all the codes written for this research. Please note that the formatting might be a bit off due to the difference in line width with matlab.

Appendix D1: Beat Detection from a live audio feed

```
%% BEAT DETECTION FROM A RUNNING AUDIO FEED
%This script detects beats from a realtime audio feed. The audio
%feed uses the PC's microphone, or any other recording device
%connected to the PC.The length of the recording is finite and
%can be set by adjusting the 'runtime' variable. Detected beats
%have their timestamps stored, and are
%afterwards plotted in a graph, together with the captured audio
%signal.
%Written by Niels van Dalen, last modified 12/06/2020

%% Initialisations
clear
clc
clf

fs = 44100;                          %samplerate
T = 1/fs;                            %sampling time (period)
frameSize = 1024;                    %sample per frame
runtime = 5;                         %runtime of the program (s)
runtimeVec = [0:T:runtime];
runtimeVec = runtimeVec(1:end-1); %vector of time points (s)

%Beat Detection Correction Factor
%decreases delay offsets (its value iscalculated in
'BDsensitivityPerformance.m')
BDCF = 0.0728;

%amount of frames in the short buffer
%lower values increase sensitivity, but lower accuracy
nFrames = 2;  %11/06/2020 value 2 is performing best
%Sensitivity, multiplier for the difference between first and
%last half of the buffer, low value gives high sensitivity,
%high sensitivity is more robust against false triggers
```

```
bdSens = 10;

sens = 25-bdSens; %Doesn't affect the beat detection itself,
only used for automated plot titling

shortBuffCap = nFrames*frameSize;
longBuffCap = fs*runtime;

audioIn = audioDeviceReader(fs,frameSize);
%a short and a long buffer
%short one is used for beat detection, long for plotting
shortBuff = dsp.AsyncBuffer('Capacity',shortBuffCap);
longBuff  = dsp.AsyncBuffer('Capacity',longBuffCap);

%counter
onsetCounter = 1;
onsetTimes = {};
frameCounter = 0;

%% Beat detection
tic;
while toc<runtime
    % Read from mic
    frame = audioIn();
    frameCounter = frameCounter +1;
    % Write the frame to the buffers
    write(shortBuff,frame);
    write(longBuff, frame);

    %peek:get values without 'burning' them (deleting afterward)
    longBuffer  = peek(longBuff, longBuffCap);
    shortBuffer = peek(shortBuff, shortBuffCap);
    %Get absolute value to get a magnitude of amplitude.
    %(netto sum for an oscillation is always ~0)
    absBuffer = abs(shortBuffer);

    %Get and compare buffer values
    sum1 = sum(absBuffer(1:shortBuffCap/2));
    sum2 = sum(absBuffer(shortBuffCap/2:shortBuffCap));
```

```
    if bdSens*sum1<sum2
        %Store and show time of detected beat
        onsetTimes{onsetCounter} = toc
        onsetCounter = onsetCounter +1;
    end
%     Uncomment to plot in realtime
%     note: realtime plotting causes latency
%     figure(1);
%     plot(longBuffer)
%     drawnow; %update figure
end
%Display the average time it took for a frame
frameTime_ms = 1000*runtime/frameCounter;

%% Plot the recording
plot(runtimeVec, longBuffer);
hold on

%Mark the detected beats
for ii = 1:length(onsetTimes)
     %Plot a marker at every index of onsetTimes{}. BDCF shifts
     beat %detection time back in time to improve accuracy by
     accounting for delays.
     plot(onsetTimes{ii}-BDCF,0,'r>', 'MarkerFaceColor', [1 0
     0]);
end

% plot layout
ylim([-1.1, 1.1]);
title("Beat Detection (1st Gen) - Sensitivity Factor: " + sens)
xlabel('Time (s)');
ylabel('Amplitude');
legend('Audio', 'Detected beat')
set(gca,'FontSize', 22);
```

Appendix D2: Normalisation Based Tone Detection (from live audio feed)

```
%% NORMALISATION BASED TONE DETECTION (FROM LIVE AUDIO FEED)
%This script detects boomwhacker tones from an audio feed.
%It uses the same beat detection mechanism as the beat detection
%algorithm in appendix D1. If a beat is detected, a fourier
%transform of the signal is taken and normalized. The frequency
%value of the peak at 1 (the highest peak) is evaluated against
%ideal frequencies of the tones. If the value is within a small
%margin of them, the tone is displayed in the command window of
%matlab.
%Written by Niels van Dalen, 12/06/2020

%General comments:
%- filtering inside while loop delays a lot and breaks the
signal (~60ms)
%- Real-time plotting causes (too much) latency, if done
include:
%  drawnow; %update figure

%% Initiliasations
clear
clc
clf

%Total runtime in seconds
runtime = 10;
%Beat detection sensitivity
bdSens = 15;

%Input samplerate, samples per frame
%smaller framesize allows for better detection of rapid
%succesive beats
fs = 44100;
%samples/frame, frames in shortBuffer
frameSize = 1024;
nFrames = 2;  %11/06/2020 value 2 is performing best
shortBuffCap = nFrames*frameSize;
```

```
longBuffCap = fs*runtime;

L = frameSize;           %number of time points
N = frameSize*nFrames;   %number of FFT points
freq_Res = fs/N;
freqMargin  = 10;         %Note detection margin (Hz)

audioIn = audioDeviceReader(fs,frameSize);
%a short and a long buffer
%short one is used for beat detection, long for continuous
plotting
shortBuff = dsp.AsyncBuffer('Capacity',shortBuffCap);
longBuff  = dsp.AsyncBuffer('Capacity',longBuffCap);

%Counters
onsetTimes = {};
ostIndex = 1;
frameCounter = 0;
figure(1);

%% Filtering (optional)
%Global filter
passBand = [200, 2000];

% %Clow filter
% f_passL1 = 260;
% f_passL2 = 519;
% passBandWidth = 5;
% passBandClow1 = [f_passL1, f_passL1+passBandWidth];
% passBandClow2 = [f_passL2, f_passL2+passBandWidth];

tic;

%% Audio capturing (live from mic)
while toc<runtime
    %% Capturing and storing audio from mic
    frame = bandpass(audioIn(), passBand, fs); %filter input,
    frameCounter = frameCounter +1;
    % Write the frame to the buffers
```

```
    write(shortBuff,frame);
    write(longBuff, frame);

    %get values without 'burning' them
    longBuffer  = peek(longBuff, longBuffCap);
    shortBuffer = peek(shortBuff, shortBuffCap);

    %% Beat (and note) detection
    %Compare buffer values for beat detection
    %Absolute value taken to determine relative energy of
vibration
    absBuffer = abs(shortBuffer);
    sum1 = sum(absBuffer(1:shortBuffCap/2));
    sum2 = sum(absBuffer(shortBuffCap/2:shortBuffCap));

    if (bdSens*sum1)<sum2
        %Store beat timestamps
        onsetTimes{ostIndex} = toc;
        ostIndex = ostIndex +1;

        %% Get spectrum
        %get fft and do some additional processing to get rid of
        %complex numbers, normalize afterwards
        %When a beat is detected, calculate and get fft
        fftShort = fft(shortBuffer);
        SSB = fftShort(1:N/2);
        SSB(2:end) = 2*SSB(2:end); %can xclude 1st element: 0 Hz
        %(0:N/2-1) defines vectorlength!
        freqBin = (0:N/2-1)*(fs/N);
        SSB = abs(SSB);
        Max = max(SSB);
        Min = min(SSB);
        SSB_norm = interp1([Min, Max],[0,1],SSB);
        %plot(SSB_norm);
        %drawnow();
        %axis([0 2000, 0 1.1]);
        %% Tone detection
```

```
for ii = 1:frameSize
            if SSB_norm(ii) == 1;
                currentFreq = freqBin(ii);
%Check which tone it is, the numerical value subtracted is the
absolute (perfect) frequency of the tone
                %Clow
               if abs(currentFreq-263)<freqMargin,disp('Clow');
end
                %D
               if abs(currentFreq-295)<freqMargin, disp('D');
end
                %E
               if abs(currentFreq-330)<freqMargin, disp('E');
end
                %F
               if abs(currentFreq-344)<freqMargin, disp('F');
end
                %G
               if abs(currentFreq-387)<freqMargin, disp('G');
end
                %A
               if abs(currentFreq-427)<freqMargin, disp('A');
end
                %B
               if abs(currentFreq-478)<freqMargin, disp('B');
end
                %Chigh
               if abs(currentFreq-513)<freqMargin,disp('Chigh');
end
            end %if
        end %for
    end
end

%% Post-run commands
%Calculate frametime (and therefore check how quick the program
can run)
%frameTime_ms = 1000*runtime/frameCounter
%sound(longBuffer, fs);
```

Appendix D3: Three Tone Detection Methods (using a sample database)

```
%% Three Tone Detection Methods for a sample database
%This program can run three different tone detection methods on
%a database. It can also plot one or more signals in time and
%frequency domain. The program's functionality can be tweaked in
%the 'Settings' section.
%Written by Niels van Dalen, 02/08/2020

%% Importing audio (from directory)
clear
clf
clc
close all
%disp('workspace and figures cleared');
warning('off'); %turn off warning messages

%Read files in the folder and import the audio data
%Note: files have to be mono and .wav at 44100Hz
FileList = ...
    dir('C:\Users\niels\Documents\Utwente\Jaar 4\Graduation
Project (& M12)\Graduation Project\Matlab scripts\audio from
file\Main Database\*.wav');
count = 0;
for File = FileList'
    count = count + 1;
    %read every file and store the data and its samplerate
    (44100)
    [audioData{count}, fs] = audioread(File.name);
end
audioData = audioData';

numSamples = 44100*4; %largest possible audioData fragment size:
4 seconds
%For loop that adds zeros up untill 'numSamples' to all files in
audioData
%This is done so all audio data arrays have the same size for
further processing
```

```
for ii=1:length(audioData)
    audioData{ii,:}(end+1:numSamples) = 0; %end+1 as to not
%overwrite the last data point
end

%% Settings
% Sample numbers that are loaded in (see FileList.name for
indices. Load all by setting files to: [1:length(FileList)];
files = [1:length(FileList)];

%0 = don't run, 1 = do run.
timePlot                = 0; %Plot of the audio files in time
freqPlotRaw             = 1; %Frequency spectrum
combineSignals          = 1; %allows (generation within matlab
                             of) simultaneous tone combinations
bandpassFilter          = 0; %Not recommended for normToneDetect
audio                   = 0; %Play the sound of a selected
                             sample

%Choose tone detection method:
%1: Detect tones with a normalisation based algorithm (detects 1
tone max.)
%2: Detect tones with a thresholded peak finding algorithm
%3: Detect tones with a subtraction based algorithm
%0: (or any other value) Don't detect tones
toneDetectionMethod     = 3;

%% Initilisations
nLoadedFiles = length(files); %amount of loaded files, used in
                              loops

%Time vector for plotting wrt. time instead of sample index
t = [0:1/fs:(numSamples/fs)];
t(:,176401) = []; %bit of a hardcode..
```

```
%Passband of the (universal) filter
bandLow = 240;
bandHigh = 560;

%freqPlot limits
freqLow = 200;
freqHigh = 600;

%Expected (ideal) frequencies (Hz) of the FF (which is the
strongest harmonic)
Clow_expFreq = 263;
D_expFreq    = 295;
E_expFreq    = 330;
F_expFreq    = 349;
G_expFreq    = 392;
A_expFreq    = 440;
B_expFreq    = 493;
Chigh_expFreq= 523;

%array of expected frequencies
expFreqs = [Clow_expFreq, D_expFreq, E_expFreq, F_expFreq,...
    G_expFreq, A_expFreq, B_expFreq, Chigh_expFreq];

%Margin around the ideal frequencies for detection
freqMargin = 10;

%MATRICES: contain (processed) data about the audio files, each
file has its own row
samples = [];   %Matrix that stores the values of a sample over
                time in columns
ffts = [];      %The raw ffts of samples()
SSBs = [];      %The processed version of ffts, which makes it
suitable for plotting
Maxes =[];      %Maximum values in SSBs
Mins = [];      %Minimum values in SSBs
SSBsNorm = [];  %SSBs, normalized from 0 to 1 (uses Maxes() and
Mins().)
freqBin = (1:(numSamples-1)/2)*(fs/numSamples); %array w/
frequency 'bins' for the fft
```

```
fontSize = 22; %for figures
legend_string = {'Clow [-0dB].wav','Clow [-3dB].wav','Clow
[-6dB].wav',...
    'D [-0dB].wav','D [-3dB].wav','D [-6dB].wav','E
[-0dB].wav',...
    'E [-3dB].wav','E [-6dB].wav','F [-0dB].wav','F
[-3dB].wav',...
    'F [-6dB].wav','G [-0dB].wav','G [-3dB].wav','G
[-6dB].wav',...
    'A [-0dB].wav','A [-3dB].wav','A [-6dB].wav','B
[-0dB].wav',...
    'B [-3dB].wav','B [-6dB].wav','Chigh [-0dB].wav',...
    'Chigh [-3dB].wav','Chigh [-6dB].wav'};

%% Processing/generation of audio files
%NOTE: audioData{files(1)} does not necessarily mean the 1st
sample (check 'files()'for the exact sample numbers)
for ii = 1:nLoadedFiles
    %Get the audio samples from the cell array and store them as
rows in 'samples'
    samples(ii,:) = audioData{files(ii)};
    %Filter the audio to remove unnecessary frequency content:
doesn't
    %remove the sample points > (sets freqs outside the band to
0)
    %BANDPASS FILTER
    if bandpassFilter ==1
        samples(ii,:) =
        bandpass(samples(ii,:), [bandLow bandHigh],fs);
    end
    %Take the fft of all loaded samples
    ffts(ii,:) = fft(samples(ii,:));
    %Single Side band, we need only the left side of the
    (symetric!) transform
    SSBs(ii,:) = ffts(ii,1:(numSamples-1)/2); %+1 or -1 added to
cover (irrelevant) warning
    SSBs(ii(2:end),:) = 2*SSBs(ii(2:end),:); %Not quite sure why
this is needed
```

```
    %it's an 'implicit Hilbert transform'
(https://medium.com/analytics-vidhya/breaking-down-confusions-ov
er-fast-fourier-transform-fft-1561a029b1ab)
    SSBs(ii,:) = abs(SSBs(ii,:));%get rid of complex values
end

%Artificially combine audio signals and store them at the end of
%SSBs, row-wise
if combineSignals ==1
    %DISCLAIMER: this is hardcoded on purpose, to keep it clear
    %which actual tone combinations are being made.

    %SSBs(size(SSBs,1)+newRow,:) --> inserts a combo at a new
    row. The content of the new signal is specified by the
    selected rows being added.

    newRow = 1;

    %SMALLEST ABSOLUTE FREQUENCY DIFFERENCES [-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(4,:);
%ClowD[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(10,:);
%EF[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(19,:)+SSBs(22,:);
%BChigh[-0dB]

    %OVERLAPPING FREQUENCY CONTENT [-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(13,:);
%ClowG[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(4,:)+SSBs(16,:);
%DA[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(19,:);
%EB[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(22,:);
%ClowChigh[-0dB]

    %CHORDS (LIKELY COMBINATIONS OF NOTES, [-0dB])
    %note: If the root position of the chord cannot be made, an
inversion version is made.
```

```
    %Major
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(1,:)+SSBs(7,:)+SSBs(13,:);          %ClowEG[-0dB]  -Cmaj
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(10,:)+SSBs(16,:)+SSBs(22,:);        %FAChigh[-0dB] -Fmaj
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(13,:)+SSBs(19,:)+SSBs(4,:);         %GBD[-0dB]     -Gmaj
    %Minor
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(4,:)+SSBs(10,:)+SSBs(16,:);         %DFA[-0dB]     -Dmin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(10,:)+SSBs(16,:)+SSBs(22,:);        %FAChigh[-0dB] -Emin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(16,:)+SSBs(22,:)+SSBs(7,:);         %AChighE[-0dB] -Amin

    %OTHER
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:) +SSBs(4,:)
+SSBs(7,:) +SSBs(10,:)...
    +SSBs(13,:)+SSBs(16,:)+SSBs(19,:)+SSBs(22,:); %all tones
[-0dB]

    %%SMALLEST ABSOLUTE FREQUENCY DIFFERENCES [0db vs -3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(5,:);
%Clow[-0dB]D[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(2,:)+SSBs(4,:);
%Clow[-3dB]D[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(11,:);
%E[-0dB]F[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(8,:)+SSBs(10,:);
%E[-3dB]F[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(19,:)+SSBs(23,:);
%B[-0dB]Chigh[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(20,:)+SSBs(22,:);
%B[-3dB]Chigh[-0dB]

    %OVERLAPPING FREQUENCY CONTENT [0dB vs -3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(14,:);
%Clow[-0dB]G[-3dB]
```

```
    SSBs(size(SSBs,1)+newRow,:) = SSBs(3,:)+SSBs(13,:);
%Clow[-3dB]G[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(4,:)+SSBs(17,:);
%D[-0dB]A[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(5,:)+SSBs(16,:);
%D[-3dB]A[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(20,:);
%E[-0dB]B[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(8,:)+SSBs(19,:);
%E[-3dB]B[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(23,:);
%Clow[-0dB]Chigh[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(2,:)+SSBs(22,:);
%Clow[-3dB]Chigh[-0dB]

    %%SMALLEST ABSOLUTE FREQUENCY DIFFERENCES [0dB vs -6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(6,:);
%Clow[-0dB]D[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(3,:)+SSBs(4,:);
%Clow[-6dB]D[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(12,:);
%E[-0dB]F[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(9,:)+SSBs(10,:);
%E[-6dB]F[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(19,:)+SSBs(24,:);
%B[-0dB]Chigh[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(21,:)+SSBs(22,:);
%B[-6dB]Chigh[-0dB]

    %OVERLAPPING FREQUENCY CONTENT [0dB vs -6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(15,:);
%Clow[-0dB]G[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(3,:)+SSBs(13,:);
%Clow[-6dB]G[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(4,:)+SSBs(18,:);
%D[-0dB]A[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(6,:)+SSBs(16,:);
%D[-6dB]A[-0dB]
```

```
    SSBs(size(SSBs,1)+newRow,:) = SSBs(7,:)+SSBs(21,:);
%E[-0dB]B[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(9,:)+SSBs(19,:);
%E[-6dB]B[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:)+SSBs(24,:);
%Clow[-0dB]Chigh[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(3,:)+SSBs(22,:);
%Clow[-6dB]Chigh[-0dB]

    %CHORDS (LIKELY COMBINATIONS OF NOTES, Various volumes)
    %note: If the root position of the chord cannot be made, an
    inversion version is made.
    %Major
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(1,:)+SSBs(9,:)+SSBs(15,:);
%Clow[-0dB]E[-3dB]G[-6dB]  -Cmaj
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(10,:)+SSBs(17,:)+SSBs(24,:);
%F[-0dB]A[-3dB]Chigh[-6dB] -Fmaj
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(13,:)+SSBs(20,:)+SSBs(6,:);
%G[-0dB]B[-3dB]D[-6dB]     -Gmaj
    %Minor
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(4,:)+SSBs(11,:)+SSBs(18,:);
%D[-0dB]F[-3dB]A[-6dB]     -Dmin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(10,:)+SSBs(17,:)+SSBs(24,:);
%F[-0dB]A[-3dB]Chigh[-6dB] -Emin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(16,:)+SSBs(23,:)+SSBs(9,:);
%A[-0dB]Chigh[-3dB]E[-6dB] -Amin

    SSBs(size(SSBs,1)+newRow,:) = SSBs(1,:) +SSBs(5,:)
+SSBs(9,:) +SSBs(10,:)...
    +SSBs(14,:)+SSBs(18,:)+SSBs(19,:)+SSBs(23,:); %all tones 0
-3 -6 0 -3 -6 0 -3
```

```
    %%SUBTRACTION WITH A (SCALED) DIFFERENT RECORDING OF THE
NOTE
    %%SMALLEST ABSOLUTE FREQUENCY DIFFERENCES [0dB vs -3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(5,:);
%Clow2[-0dB]D[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(2,:);
%Clow[-3dB]D2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(41,:)+SSBs(11,:);
%E2[-0dB]F[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(8,:)+SSBs(42,:);
%E[-3dB]F2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(45,:)+SSBs(23,:);
%B2[-0dB]Chigh[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(20,:)+SSBs(46,:);
%B[-3dB]Chigh2[-0dB]

    %OVERLAPPING FREQUENCY CONTENT [0dB vs -3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(14,:);
%Clow2[-0dB]G[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(2,:)+SSBs(43,:);
%Clow[-3dB]G2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(17,:);
%D2[-0dB]A[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(5,:)+SSBs(44,:);
%D[-3dB]A2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(41,:)+SSBs(20,:);
%E2[-0dB]B[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(8,:)+SSBs(45,:);
%E[-3dB]B2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(23,:);
%Clow2[-0dB]Chigh[-3dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(2,:)+SSBs(46,:);
%Clow[-3dB]Chigh2[-0dB]
```

```
    %%SMALLEST ABSOLUTE FREQUENCY DIFFERENCES [0dB vs -6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(6,:);
%Clow2[-0dB]D[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(4,:);
%Clow[-6dB]D2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(41,:)+SSBs(12,:);
%E2[-0dB]F[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(9,:)+SSBs(42,:);
%E[-6dB]F2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(45,:)+SSBs(24,:);
%B2[-0dB]Chigh[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(21,:)+SSBs(46,:);
%B[-6dB]Chigh2[-0dB]

    %OVERLAPPING FREQUENCY CONTENT [0dB vs -6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(15,:);
%Clow2[-0dB]G[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(43,:);
%Clow[-6dB]G2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(18,:);
%D2[-0dB]A[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(6,:)+SSBs(44,:);
%D[-6dB]A2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(41,:)+SSBs(21,:);
%E2[-0dB]B[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(9,:)+SSBs(45,:);
%E[-6dB]B2[-0dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:)+SSBs(24,:);
%Clow2[-0dB]Chigh[-6dB]
    SSBs(size(SSBs,1)+newRow,:) = SSBs(40,:)+SSBs(46,:);
%Clow[-6dB]Chigh2[-0dB]

    %CHORDS (LIKELY COMBINATIONS OF NOTES, Various volumes)
    %note: If the root position of the chord cannot be made, an
    %inversion version is made.
    %Major
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(39,:)+SSBs(9,:)+SSBs(15,:);
%Clow[-0dB]E[-3dB]G[-6dB]  -Cmaj
```

```
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(42,:)+SSBs(17,:)+SSBs(24,:);
%F[-0dB]A[-3dB]Chigh[-6dB] -Fmaj
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(43,:)+SSBs(20,:)+SSBs(6,:);
%G[-0dB]B[-3dB]D[-6dB]     -Gmaj
    %Minor
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(4,:)+SSBs(11,:)+SSBs(18,:);
%D[-0dB]F[-3dB]A[-6dB]     -Dmin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(42,:)+SSBs(17,:)+SSBs(24,:);
%F[-0dB]A[-3dB]Chigh[-6dB] -Emin
    SSBs(size(SSBs,1)+newRow,:) =
SSBs(44,:)+SSBs(23,:)+SSBs(9,:);
%A[-0dB]Chigh[-3dB]E[-6dB] -Amin

    SSBs(size(SSBs,1)+newRow,:) = SSBs(39,:) +SSBs(5,:)
+SSBs(9,:) +SSBs(42,:)...
    +SSBs(14,:)+SSBs(18,:)+SSBs(45,:)+SSBs(23,:); %all tones
(#2): 0 -3 -6 0 -3 -6 0 -3 [dB]
end %if combinedSignals ==1;

%Get the extreme values of all signals (incl. the combined ones)
%Used for normalisation based tone detection, and setting
%ylimits in plots
for jj = 1:size(SSBs,1)
    Maxes(jj,:)    = max(SSBs(jj,:));
    Mins(jj,:)     = min(SSBs(jj,:));
    SSBsNorm(jj,:) =
(interp1([Mins(jj,:),Maxes(jj,:)],[0,1],SSBs(jj,:)));
end

%Off all signals, get the lowest peak value (note: that is still
%a maximum of the signal. Only used for clarity (29/07)
lowestPeak = min(Maxes);
```

```
%% Time plot (excludes combinedSignals)
if timePlot == 1
    figure(1);
    grid on;
    hold on
    for kk = 1:nLoadedFiles %or if you also want
                            combinedsignals: 1:(size(SSBs,1))
        plot(t*1000,samples(kk,:));
    end
    title('Timeplot');
    xlabel('time (ms)');
    ylabel('Amplitude');
    legend('Clow[-0dB]', 'F[-6dB]', 'High threshold', 'Low
    threshold' )%legend_string;
    set(gca,'FontSize',fontSize);
end

%% Raw frequency plot
if freqPlotRaw==1
    figure(2);
    grid on;
    hold on
    title('Frequency spectrum');
    xlabel('frequency (Hz)');
    ylabel('Amplitude');

    %plot every SSB row in SSBs
     for kk=1:(size(SSBs,1))
         plot(freqBin, SSBs(kk,:));
     end
    %Single plot:
    %plot(freqBin,SSBs(60,:))

    axis([freqLow freqHigh, 0 max(Maxes)*1.1]);
    set(gca,'FontSize',fontSize, 'xtick',
[freqLow:20:freqHigh]);
    legend('Clow[-0dB]', 'Clow[-3dB]', 'Clow[-6dB]',
'Clow2[-0dB]')
end
```

```
%% Result matrix
%Import the table with the correct results and convert to
matrix.
correctAnswersTable =
readtable("C:\Users\niels\Documents\Utwente\Jaar 4\Gradution
Project (& M12)\Graduation Project\Matlab scripts\Result
tables\Correct detections results matrix.csv");
correctAnswersMatrix = correctAnswersTable{:,:};

%Cell matrix for logging the results from the tone detection
%script
offset = 1; %used to keep row 1 and column 1 for name of file
%and detected tone
toneDetected = ["Clow", "D", "E", "F", "G", "A", "B", "Chigh"];
combinedToneNames = {};

%combinedToneNames consists of custom given row names
if combineSignals ==1
    combinedToneNames = {'ClowD[-0dB]', 'EF[-0dB]',
'BChigh[-0dB]',... %smallest freq diff (0dB)
'ClowG[-0dB]', 'DA[-0dB]', 'EB[-0dB]',
'ClowChigh[-0dB]'...%overlap freqs (0dB)
'ClowEG[-0dB](Cmaj)','FAChigh[-0dB](Fmaj)','GBD[-0dB](Fmaj)'...
%Major Chords (0dB)
'DFA[-0dB](Dmin)','EGB[-0dB](Emin)','AChighE(Amin)'...%Minor
Chords (0dB)
'ClowDEFGABChigh[-0dB]'... %all tones
'Clow[-0dB]D[-3dB]','Clow[-3dB]D[-0dB]','E[-0dB]F[-3dB]'...%smal
lest freq diff (0dB vs -3dB)
'E[-3dB]F[-0dB]','B[-0dB]Chigh[-3dB]','B[-3dB]Chigh[-0dB]'...
'Clow[-0dB]G[-3dB]','Clow[-3dB]G[-0dB]','D[-0dB]A[-3dB]','D[-3dB
]A[-0dB]'...%overlap freqs (0 vs -3)
'E[-0dB]B[-3dB]','E[-3dB]B[-0dB]','Clow[-0dB]Chigh[-3dB]','Clow[
-3dB]Chigh[-0dB]'...
'Clow[0dB]D[-6dB]', 'Clow[-6dB]D[-0dB]', 'E[-0dB]F[-6dB]'...
%smallest freq diff (0dB vs -6dB)
'E[-6dB]F[-0dB]', 'B[-0dB]Chigh[-6dB]', 'B[-6dB]Chigh[-0dB]'...
'Clow[-0dB]G[-6dB]', 'Clow[-6dB]G[-0dB]', 'D[-0dB]A[-6dB]'...
%overlap freqs (0 vs -6)
```

```
'D[-6dB]A[-0dB]', 'E[-0dB]B[-6dB]', 'E[-6dB]B[-0dB]'...
'Clow[-0dB]Chigh[-6dB]', 'Clow[-6dB]Chigh[-0dB]'...
'Clow[-0dB]E[-3dB]G[-6dB](Cmaj)',
'F[-0dB]A[-3dB]Chigh[-6dB](Fmaj)'...%Chords (0dB -3dB -6dB)
'G[-0dB]B[-3dB]D[-6dB](Gmaj)', 'D[-0dB]F[-3dB]A[-6dB](Dmin)'...
'E[-0dB]G[-3dB]B[-6dB](Emin)','A[-0dB]Chigh[-3dB]E[-6dB](Amin)'
'ClowDEFGABChigh[-0/-3/-6dB]'... %All tones (diff volumes)

'Clow2[-0dB]D[-3dB]','Clow[-3dB]D2[-0dB]','E2[-0dB]F[-3dB]'...
%smallest freq diff (0 vs -3) [diff sample]
'E[-3dB]F2[-0dB]','B2[-0dB]Chigh[-3dB]','B[-3dB]Chigh2[-0dB]'...
'Clow2[-0dB]G[-3dB]','Clow[-3dB]G2[-0dB]','D2[-0dB]A[-3dB]','D[-
3dB]A2[-0dB]'...%overlap freqs 0 vs -3 [diff sample]
'E2[-0dB]B[-3dB]','E[-3dB]B2[-0dB]','Clow2[-0dB]Chigh[-3dB]','Cl
ow[-3dB]Chigh2[-0dB]'...
'Clow2[0dB]D[-6dB]', 'Clow[-6dB]D2[-0dB]', 'E2[-0dB]F[-6dB]'...
%smallest freq diff (diff samples 0 vs -6dB)
`E[-6dB]F2[-0dB]',
'B2[-0dB]Chigh[-6dB]','B[-6dB]Chigh2[-0dB]'...
'Clow2[-0dB]G[-6dB]', 'Clow[-6dB]G2[-0dB]', 'D2[-0dB]A[-6dB]'...
%overlap freqs (diff volumes)
'D[-6dB]A2[-0dB]', 'E2[-0dB]B[-6dB]', 'E[-6dB]B2[-0dB]'...
'Clow2[-0dB]Chigh[-6dB]', 'Clow[-6dB]Chigh2[-0dB]'...
'Clow2[-0dB]E[-3dB]G[-6dB](Cmaj)',
'F2[-0dB]A[-3dB]Chigh[-6dB](Fmaj)'...%Chords (diff volumes)
'G2[-0dB]B[-3dB]D[-6dB](Gmaj)',
'D2[-0dB]F[-3dB]A[-6dB](Dmin)'...
'E2[-0dB]G[-3dB]B[-6dB](Emin)','A2[-0dB]Chigh[-3dB]E[-6dB](Amin)
'...
'Clow2DEF2GAB2Chigh[-0/-3/-6dB]' %All tones (diff volumes)
}; %Hardcode the names <3 :)
end
%Title rows with the names of the samples loaded in, 1st row
empty
results = [{[]}, FileList.name, combinedToneNames]';

%Title columns with the tone names: Clow,D,E,F,G,A,B,Chigh
%Is used to indicate which tones are detected for each sample
```

```
for jj=1:length(toneDetected)
    results(offset, offset+jj) = {toneDetected(jj)};
end
%Set all data to 0 by default, will later be set to 1 if a tone
is detected
for ii=2:length(results) %Note: first row/column is reserved for
                         names
    results(ii,(offset+1):end) = {zeros};
end

%% Normalisation Based Tone Detection (single tones)
if toneDetectionMethod ==1
    %Tone detection:
    %Maximum frequency offset w.r.t. expected frequency for a
    %tone, based on the spectra 10 appears to perform best
    %(02/07/2020)

    freqMargin = 10;

    for rr =1:size(SSBsNorm,1)
        %Set all result rows to '0' by default
        for ss = 1:length(SSBsNorm)
            if SSBsNorm(rr,ss) == 1 %whenever the highest
%frequency in a row is found.. (=1, normalised)..store the
%biggest frequency component in currentFreq
                currentFreq = freqBin(ss);
%Check which tone it is, the numerical value subtracted
%from currentFreq is the expected frequency of the tone
%If a certain tone is detected, the tone is marked '1' in
%the 'results' matrix.
                if abs(currentFreq-Clow_expFreq)  <freqMargin,
results{rr+1,1+offset} = 1; end
                if abs(currentFreq-D_expFreq)     <freqMargin,
results{rr+1,2+offset} = 1; end
                if abs(currentFreq-E_expFreq)     <freqMargin,
results{rr+1,3+offset} = 1; end
                if abs(currentFreq-F_expFreq)     <freqMargin,
results{rr+1,4+offset} = 1; end
```

```
                if abs(currentFreq-G_expFreq)     <freqMargin,
results{rr+1,5+offset} = 1; end
                if abs(currentFreq-A_expFreq)     <freqMargin,
results{rr+1,6+offset} = 1; end
                if abs(currentFreq-B_expFreq)     <freqMargin,
results{rr+1,7+offset} = 1; end
                if abs(currentFreq-Chigh_expFreq) <freqMargin,
results{rr+1,8+offset} = 1; end
            end %if SSBsNorm(jj) == 1
        end %for ss = 1:length(SSBsNorm)
    end %for rr =1:nLoadedFiles
end %if normToneDetection ==1

%% Peak based Tone Detection
if toneDetectionMethod ==2
    nPeaks = 8; %max nrof peaks
    corFact = length(SSBs)/(fs*0.5); %=4 ,SSBs is longer than
fs, so the plot is scaled a bit. Prevent peaks from showing up
close to each other. Note: 19.6 Hz is the smallest absolute
freq.diff. for this set of notes.
    minPeakDistance = corFact*19.6;
    %FOR EVERY SIGNAL SEPERATELY, GET MAX AND ADJUST
    MINPEAKHEIGHT FOR IT
    minPeakHeight = 500;%lowestPeak-1; %lowestPeak only excludes
    the lowest peak

    %Matrix for storing frequencies of peaks, row-wise per
    sample
    peakFreqsSize = size(SSBs,1);
    peakFreqs = zeros(peakFreqsSize,nPeaks);
    %Get the peaks values along with the frequency at which they
    occur
    for ii=1:size(peakFreqs,1)
        [peakVals,peakIndex] = findpeaks(SSBs(ii,:), 'NPeaks',
        nPeaks, 'MinPeakDistance',...
        minPeakDistance, 'MinPeakHeight', minPeakHeight);
        peakIndex = peakIndex/corFact; %Divide by 4 to get
        frequency
```

```
        %Store the frequencies of the peaks together in matrix,
        %each row contains the peaks found in a file and each
        %column. Sets the amount of peaks (multiple peaks per
        %file are allowed)

        for jj=1:size(peakIndex,2) %size of column
            peakFreqs(ii,jj) = peakIndex(jj);
            %Compare the peaks in the matrix to the theoretical
            frequency values
            if abs(peakFreqs(ii,jj)-Clow_expFreq)  <freqMargin,
results{offset+ii,1+offset} = 1; end
            if abs(peakFreqs(ii,jj)-D_expFreq)     <freqMargin,
results{offset+ii,2+offset} = 1; end
            if abs(peakFreqs(ii,jj)-E_expFreq)     <freqMargin,
results{offset+ii,3+offset} = 1; end
            if abs(peakFreqs(ii,jj)-F_expFreq)     <freqMargin,
results{offset+ii,4+offset} = 1; end
            if abs(peakFreqs(ii,jj)-G_expFreq)     <freqMargin,
results{offset+ii,5+offset} = 1; end
            if abs(peakFreqs(ii,jj)-A_expFreq)     <freqMargin,
results{offset+ii,6+offset} = 1; end
            if abs(peakFreqs(ii,jj)-B_expFreq)     <freqMargin,
results{offset+ii,7+offset} = 1; end
            if abs(peakFreqs(ii,jj)-Chigh_expFreq) <freqMargin,
results{offset+ii,8+offset} = 1; end
        end %jj
    end %ii
    %Plot the last signal's peaks
    %plot(peakIndex, peakVals,'r*')
end %if peakBasedToneDetection ==1
```

```
%% Subtraction based Tone Detection
if toneDetectionMethod ==3
    %Subtraction based tone detection
    %Create a copy of SSBs, the copy loses data in the
    %processing and will afterwards not produce logical figures!
  SSBsCopy = SSBs;
    %Signals to be scaled and subtracted, contains a [-0dB]
    'octave' of Clow,D,E,F,G,A,B,Chigh
    SSBsSubSig = SSBs([1,4,7,10,13,16,19,22],:);

    Clow=1; D =2; E=3; F=4; G=5; A=6; B=7; Chigh=8; %for some
    extra clarity in the coming code.

    softThreshold = 80; %lowest peak of samples from file is
    %86.1587 (see lowestPeak), combined signals will have higher
    %values...
    maxTones = 8; %Used to loop and with that set the maximum
    number of tones that can be detected

    corFact = length(SSBsCopy)/(fs*0.5); %=4 ,SSBs is longer
    than fs, so the plot (indices) are scaled

    %For every frequency spectrum...
    for rr = 1:length(SSBsCopy(:,1))
        %Check which tone it is, the numerical value subtracted
        %from currentFreq is the expected frequency of the tone,
        %absolute value makes sure the freqMargin is the same
        %for higher and lower frequencies wrt. expected
        %frequency. If a certain tone is detected (and not
        %detected in the signal already) %the tone is marked '1'
        %in the 'results' matrix.

        for ss =1:maxTones
            %Get maximum value and its index of the spectrum
            %('highest peak coordinates')
            [val,index] = max(SSBsCopy(rr,:));
            currentFreq = index/corFact; %divide by 4 to get
            %correct frequency in Hz
```

```
            if val>softThreshold %set soft threshold (to prevent
            possible infinite subtracting)
            %The approach of tone detection and subtraction is
            %identical for each tone, elaborate comments r only
            %added to Clow in order to keep things compact asp.

%Clow
        if  abs(currentFreq-Clow_expFreq) <freqMargin &&
            results{rr+offset,Clow+offset} == 0
            %If the peak is near Clow's pitch AND it hasn't
            %been detected yet (latter prevents subtraction
            %from keeping looping):
            results{rr+offset,Clow+offset} = 1; %log
            %detection of Clow
            maxSubSigClow = max(SSBsSubSig(Clow,:)); %take
            %the max of signal to subtract (used for scaling)
            maxCurrent = max(SSBsCopy(rr,:)); %Get height of
            the signal's peak (used for scaling) Scale the
            signal of a Clow note so that subtraction will be
            done to scale of the current signal (ie. not drop
            everything<0 or have very little effect)
            SSBsSubSig(Clow,:) =
            interp1([0,maxSubSigClow],[0,maxCurrent],
            SSBsSubSig(Clow,:));
            SSBsCopy(rr,:) =
            SSBsCopy(rr,:)-SSBsSubSig(Clow,:); %SUBTRACT a
            scaled Clow sample
        end %Clow
%D
        if  abs(currentFreq-D_expFreq)    <freqMargin &&
            results{rr+offset,D+offset} == 0
            results{rr+offset,D+offset} = 1;
            maxSubSigClow = max(SSBsSubSig(D,:));
            maxCurrent = max(SSBsCopy(rr,:));
            SSBsSubSig(D,:) =
            interp1([0,maxSubSigClow],[0,maxCurrent],
            SSBsSubSig(D,:));
            SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(D,:);
        end %D
```

```
%E
            if  abs(currentFreq-E_expFreq)    <freqMargin &&
                results{rr+offset,E+offset} == 0
                results{rr+offset,E+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(E,:));
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(E,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(E,:));
                SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(E,:);
            end %E

%F
            if  abs(currentFreq-F_expFreq)    <freqMargin &&
                results{rr+offset,F+offset} == 0
                results{rr+offset,F+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(F,:));
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(F,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(F,:));
                SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(F,:);
            end %F
%G
            if  abs(currentFreq-G_expFreq)    <freqMargin &&
                results{rr+offset,G+offset} == 0
                results{rr+offset,G+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(G,:));
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(G,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(G,:));
                SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(G,:);
            end %G
%A
            if  abs(currentFreq-A_expFreq)    <freqMargin &&
                results{rr+offset,A+offset} == 0
                results{rr+offset,A+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(A,:));
```

```
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(A,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(A,:));
                SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(A,:);
            end %A
%B
            if  abs(currentFreq-B_expFreq)    <freqMargin &&
                results{rr+offset,B+offset} == 0
                results{rr+offset,B+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(B,:));
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(B,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(B,:));
                SSBsCopy(rr,:) = SSBsCopy(rr,:)-SSBsSubSig(B,:);
            end %B
%Chigh
            if  abs(currentFreq-Chigh_expFreq)    <freqMargin &&
                results{rr+offset,Chigh+offset} == 0
                results{rr+offset,Chigh+offset} = 1;
                maxSubSigClow = max(SSBsSubSig(Chigh,:));
                maxCurrent = max(SSBsCopy(rr,:));
                SSBsSubSig(Chigh,:) =
                interp1([0,maxSubSigClow],[0,maxCurrent],
                SSBsSubSig(Chigh,:));
                SSBsCopy(rr,:) =
                SSBsCopy(rr,:)-SSBsSubSig(Chigh,:);
          end %Chigh


             end %if val>
        end %for ss = 1:loops
    end
end
```

```
%% sound
if audio == 1
    %Change the number for different samples
    sound(samples(1,:)+samples(2,:),fs);
end

%% Write result matrix to file:
%writecell(results,'C:\Users\niels\Documents\Utwente\Jaar
4\Gradution Project (& M12)\Graduation Project\Matlab
scripts\Result tables\name.xls');
```

Appendix E: Correct tone detection answer matrix

1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0
0	0	0	1	0	1	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	1
1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0

1	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0
0	0	0	1	0	1	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	1
1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0

1	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0
0	0	0	1	0	1	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	1
1	1	1	1	1	1	1	1